2) Time series

August 8, 2023

1 1) Preparación previa

Carga de librerías

[1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
%matplotlib inline

import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt

from scipy import stats
from statistics import mode

from sklearn.model_selection import train_test_split

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_predict
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error

# Se debe instalar pmdarima
from pmdarima import auto_arima #!pip install pmdarima

import warnings
warnings.filterwarnings('ignore')
```

Lectura del dataset

[2]:
```python
df = pd.read_csv('https://raw.githubusercontent.com/Agustin-Bulzomi/Projects/
↪main/Programming/Digital%20House%20(Python)/Support%20Files/Final%20Project/
↪coin_Bitcoin.csv', delimiter=',')
```

Se realizan las modificaciones del dataset pertinentes para el análisis de series de tiempo

[3]:
```python
df['Date'] = pd.to_datetime(df['Date'])
df.index = pd.PeriodIndex(df.Date, freq='D')
df.head()
```

[3]:
```
            SNo     Name Symbol       Date        High         Low  \
Date
2013-04-29    1  Bitcoin    BTC 2013-04-29  147.488007  134.000000
2013-04-30    2  Bitcoin    BTC 2013-04-30  146.929993  134.050003
2013-05-01    3  Bitcoin    BTC 2013-05-01  139.889999  107.720001
2013-05-02    4  Bitcoin    BTC 2013-05-02  125.599998   92.281898
2013-05-03    5  Bitcoin    BTC 2013-05-03  108.127998   79.099998

                  Open       Close  Volume     Marketcap
Date
2013-04-29  134.444000  144.539993     0.0  1.603769e+09
2013-04-30  144.000000  139.000000     0.0  1.542813e+09
2013-05-01  139.000000  116.989998     0.0  1.298955e+09
2013-05-02  116.379997  105.209999     0.0  1.168517e+09
2013-05-03  106.250000   97.750000     0.0  1.085995e+09
```

Dummies

[4]:
```python
df["timeIndex"] = pd.Series(np.arange(len(df['Close'])), index=df.index)

df.head()
```

[4]:
```
            SNo     Name Symbol       Date        High         Low  \
Date
2013-04-29    1  Bitcoin    BTC 2013-04-29  147.488007  134.000000
2013-04-30    2  Bitcoin    BTC 2013-04-30  146.929993  134.050003
2013-05-01    3  Bitcoin    BTC 2013-05-01  139.889999  107.720001
2013-05-02    4  Bitcoin    BTC 2013-05-02  125.599998   92.281898
2013-05-03    5  Bitcoin    BTC 2013-05-03  108.127998   79.099998

                  Open       Close  Volume     Marketcap  timeIndex
Date
2013-04-29  134.444000  144.539993     0.0  1.603769e+09          0
2013-04-30  144.000000  139.000000     0.0  1.542813e+09          1
2013-05-01  139.000000  116.989998     0.0  1.298955e+09          2
2013-05-02  116.379997  105.209999     0.0  1.168517e+09          3
2013-05-03  106.250000   97.750000     0.0  1.085995e+09          4
```

[5]:
```python
df.tail()
```

[5]:
```
                 SNo     Name Symbol       Date         High          Low  \
Date
2021-02-23  2858  Bitcoin    BTC 2021-02-23  54204.92976  45290.59027
2021-02-24  2859  Bitcoin    BTC 2021-02-24  51290.13669  47213.49816
2021-02-25  2860  Bitcoin    BTC 2021-02-25  51948.96698  47093.85302
2021-02-26  2861  Bitcoin    BTC 2021-02-26  48370.78526  44454.84211
2021-02-27  2862  Bitcoin    BTC 2021-02-27  48253.27010  45269.02577

                   Open        Close        Volume     Marketcap  timeIndex
Date
2021-02-23  54204.92976  48824.42687  1.061020e+11  9.099260e+11       2857
2021-02-24  48835.08766  49705.33332  6.369552e+10  9.263930e+11       2858
2021-02-25  49709.08242  47093.85302  5.450657e+10  8.777660e+11       2859
2021-02-26  47180.46405  46339.76008  3.509680e+11  8.637520e+11       2860
2021-02-27  46344.77224  46188.45128  4.591095e+10  8.609780e+11       2861
```

Se agregan las columnas necesarias

[6]:
```python
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year
```

Se crean las dummies

[7]:
```python
dummies_mes = pd.get_dummies(df['Month'], drop_first=True, prefix='Month')
df = df.join(dummies_mes)
df.sample(10)
```

[7]:
```
                 SNo     Name Symbol       Date          High           Low  \
Date
2019-08-12  2297  Bitcoin    BTC 2019-08-12  11528.189370  11320.951500
2015-05-27   759  Bitcoin    BTC 2015-05-27    238.636002    236.695007
2020-02-23  2492  Bitcoin    BTC 2020-02-23   9937.404106   9657.791145
2015-12-14   960  Bitcoin    BTC 2015-12-14    447.141998    430.455994
2014-08-05   464  Bitcoin    BTC 2014-08-05    589.864990    584.101990
2018-07-16  1905  Bitcoin    BTC 2018-07-16   6741.750000   6357.009766
2018-10-19  2000  Bitcoin    BTC 2018-10-19   6493.680000   6445.310000
2019-02-22  2126  Bitcoin    BTC 2019-02-22   4006.538387   3950.816427
2019-04-16  2179  Bitcoin    BTC 2019-04-16   5238.945238   5055.194961
2018-10-28  2009  Bitcoin    BTC 2018-10-28   6502.280000   6447.910000

                    Open         Close        Volume     Marketcap  …  \
Date                                                                 …
2019-08-12  11528.189370  11382.615960  1.364720e+10  2.034410e+11  …
2015-05-27    237.065002    237.283005  1.883700e+07  3.371258e+09  …
2020-02-23   9663.318642   9924.515228  4.118519e+10  1.809630e+11  …
2015-12-14    433.272003    444.182007  1.304960e+08  6.645396e+09  …
2014-08-05    589.010986    585.435974  1.079080e+07  7.671275e+09  …
2018-07-16   6357.009766   6741.750000  4.725800e+09  1.156380e+11  …
```

```
2018-10-19   6478.070000   6465.410000  3.578870e+09  1.120530e+11  …
2019-02-22   3952.406374   4005.526604  7.826525e+09  7.030856e+10  …
2019-04-16   5066.577601   5235.559419  1.161866e+10  9.240461e+10  …
2018-10-28   6482.660000   6486.390000  3.445190e+09  1.125180e+11  …

            Month_3  Month_4  Month_5  Month_6  Month_7  Month_8  Month_9  \
Date
2019-08-12        0        0        0        0        0        1        0
2015-05-27        0        0        1        0        0        0        0
2020-02-23        0        0        0        0        0        0        0
2015-12-14        0        0        0        0        0        0        0
2014-08-05        0        0        0        0        0        1        0
2018-07-16        0        0        0        0        1        0        0
2018-10-19        0        0        0        0        0        0        0
2019-02-22        0        0        0        0        0        0        0
2019-04-16        0        1        0        0        0        0        0
2018-10-28        0        0        0        0        0        0        0

            Month_10  Month_11  Month_12
Date
2019-08-12         0         0         0
2015-05-27         0         0         0
2020-02-23         0         0         0
2015-12-14         0         0         1
2014-08-05         0         0         0
2018-07-16         0         0         0
2018-10-19         1         0         0
2019-02-22         0         0         0
2019-04-16         0         0         0
2018-10-28         1         0         0

[10 rows x 24 columns]
```

Se divide el dataset en Train y Test, usando rangos personalizados

[8]:
```python
# Se utiliza este método para manejar el shape que se aplicará a los demás␣
↪modelos
end_date = '2021-01-27'
mask1 = (df['Date'] <= end_date)
mask2 = (df['Date'] > end_date)
```

[9]:
```python
# Se pasan las máscaras para obtener train y test:
df_train = df.loc[mask1]
df_test = df.loc[mask2]
print("train shape",df_train.shape)
print("test shape",df_test.shape)
```

```
train shape (2831, 24)
```

```
test shape (31, 24)
```

Chequeo de que la primer fecha de test sea la siguiente al final de train:

[10]:
```
df_train.tail()
```

[10]:
```
             SNo     Name Symbol        Date         High          Low  \
Date
2021-01-23  2827  Bitcoin    BTC 2021-01-23  33360.97819  31493.15967
2021-01-24  2828  Bitcoin    BTC 2021-01-24  32944.00894  31106.68577
2021-01-25  2829  Bitcoin    BTC 2021-01-25  34802.74298  32087.78797
2021-01-26  2830  Bitcoin    BTC 2021-01-26  32794.54959  31030.26597
2021-01-27  2831  Bitcoin    BTC 2021-01-27  32564.03024  29367.13922

                   Open        Close        Volume     Marketcap  …  \
Date                                                               …
2021-01-23  32985.75691  32067.64288  4.835474e+10  5.967330e+11  …
2021-01-24  32064.37632  32289.37809  4.864383e+10  6.008890e+11  …
2021-01-25  32285.79891  32366.39305  5.989705e+10  6.023500e+11  …
2021-01-26  32358.61317  32569.84956  6.025542e+10  6.061690e+11  …
2021-01-27  32564.03024  30432.54708  6.257676e+10  5.664170e+11  …

            Month_3  Month_4  Month_5  Month_6  Month_7  Month_8  Month_9  \
Date
2021-01-23        0        0        0        0        0        0        0
2021-01-24        0        0        0        0        0        0        0
2021-01-25        0        0        0        0        0        0        0
2021-01-26        0        0        0        0        0        0        0
2021-01-27        0        0        0        0        0        0        0

            Month_10  Month_11  Month_12
Date
2021-01-23         0         0         0
2021-01-24         0         0         0
2021-01-25         0         0         0
2021-01-26         0         0         0
2021-01-27         0         0         0

[5 rows x 24 columns]
```

[11]:
```
df_test.head()
```

[11]:
```
             SNo     Name Symbol        Date         High          Low  \
Date
2021-01-28  2832  Bitcoin    BTC 2021-01-28  33858.31099  30023.20683
2021-01-29  2833  Bitcoin    BTC 2021-01-29  38406.26096  32064.81419
2021-01-30  2834  Bitcoin    BTC 2021-01-30  34834.70830  32940.18691
2021-01-31  2835  Bitcoin    BTC 2021-01-31  34288.33148  32270.17602
```

```
2021-02-01  2836  Bitcoin    BTC 2021-02-01  34638.21349  32384.22811

                   Open         Close        Volume     Marketcap  …  \
Date                                                                …
2021-01-28  30441.04182  33466.09636  7.651716e+10  6.229100e+11  …
2021-01-29  34318.67169  34316.38765  1.178950e+11  6.387690e+11  …
2021-01-30  34295.93504  34269.52154  6.514183e+10  6.379250e+11  …
2021-01-31  34270.87759  33114.35775  5.275454e+10  6.164530e+11  …
2021-02-01  33114.57724  33537.17682  6.140040e+10  6.243490e+11  …

            Month_3  Month_4  Month_5  Month_6  Month_7  Month_8  Month_9  \
Date
2021-01-28        0        0        0        0        0        0        0
2021-01-29        0        0        0        0        0        0        0
2021-01-30        0        0        0        0        0        0        0
2021-01-31        0        0        0        0        0        0        0
2021-02-01        0        0        0        0        0        0        0

            Month_10  Month_11  Month_12
Date
2021-01-28         0         0         0
2021-01-29         0         0         0
2021-01-30         0         0         0
2021-01-31         0         0         0
2021-02-01         0         0         0

[5 rows x 24 columns]
```

Ploteo de los dos datasets obtenidos:

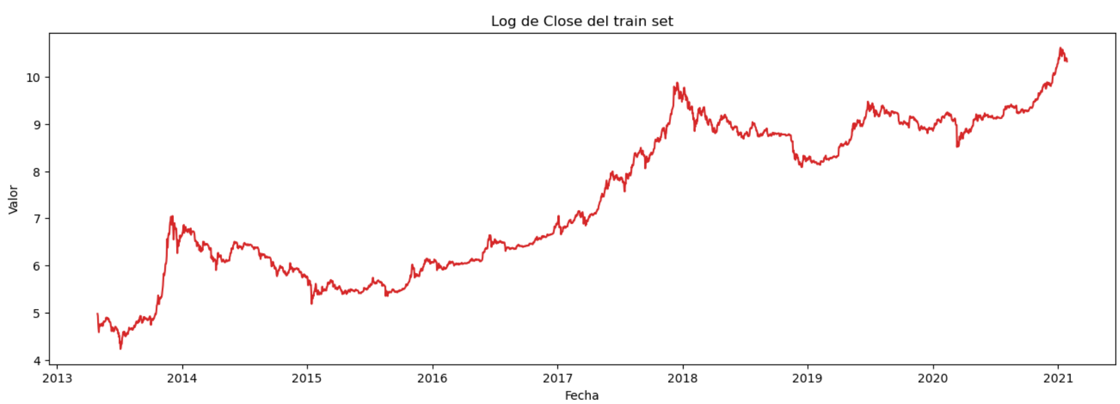
[12]:
```
pd.plotting.register_matplotlib_converters()
f, ax = plt.subplots(figsize=(14,5))
df_train.plot(kind='line', x='Date', y='Close', color='blue', label='Train', ax␣
  ↪= ax)
df_test.plot(kind='line', x='Date', y='Close', color='red', label='Test', ax =␣
  ↪ax)
ax.legend(loc= "upper left")
plt.title('Rango para Train y para Test')
plt.show()
```

Ploteo del Target en escala logarítmica:

[13]:
```
df_train['log_value'] = np.log(df_train['Close'])
df_test['log_value'] = np.log(df_test['Close'])
```

[14]:
```
def plot_df(df, x, y, title = "/", xlabel='Fecha', ylabel='Valor', dpi=100):
    plt.figure(figsize=(16,5), dpi=dpi),
    plt.plot(x, y, color='tab:red'),
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel),
    plt.show()
```

[15]:
```
plot_df(df_train, x=df_train.Date, y=df_train['log_value'],\
    title='Log de Close del train set')
```



Entrenaiento del modelo para analizar el Summary:

[16]:
```
model_log = smf.ols('log_value ~ timeIndex',\
                          data = df_train).fit()
```

[17]:
```
model_log.summary()
```

[17]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_value   R-squared:                       0.835
Model:                            OLS   Adj. R-squared:                  0.835
Method:                 Least Squares   F-statistic:                 1.434e+04
Date:                Tue, 08 Aug 2023   Prob (F-statistic):               0.00
Time:                        21:29:25   Log-Likelihood:                -2770.3
No. Observations:                2831   AIC:                             5545.
Df Residuals:                    2829   BIC:                             5556.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      4.8828      0.024    201.754      0.000       4.835       4.930
timeIndex      0.0018   1.48e-05    119.736      0.000       0.002       0.002
==============================================================================
Omnibus:                      222.150   Durbin-Watson:                   0.004
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              270.407
Skew:                           0.746   Prob(JB):                     1.91e-59
Kurtosis:                       2.747   Cond. No.                     3.27e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.27e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

2 2) Modelos

Se utilizará una plétora de herramientas y recursos para analizar las series de tiempo y sus implicancias. En cada paso se irá visualizando los resultados y almacenando su información para, al final de la notebook, compararlos

2.1 a) Mean

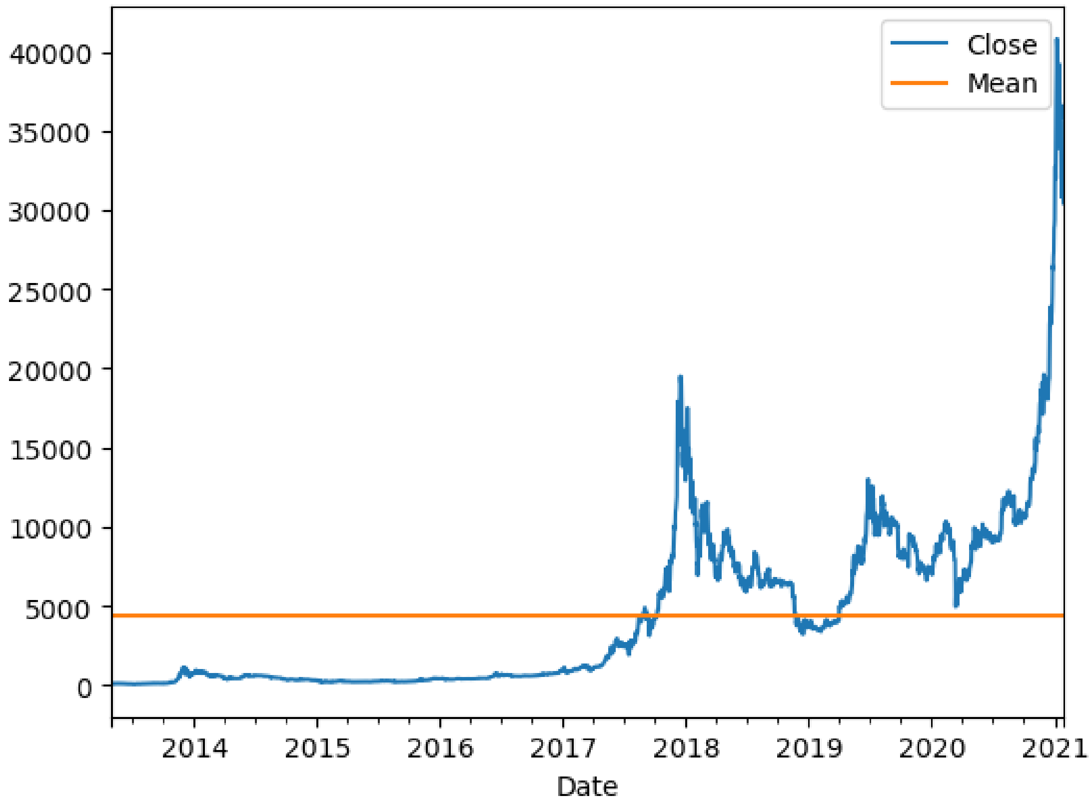
Se aplica el modelo de media constante a train y test:

[18]:
```
# Se calcula el promedio:
model_mean_pred = df_train['Close'].mean()

# La predicción es fija y es la misma para el set de testeo y de entrenamiento:
```

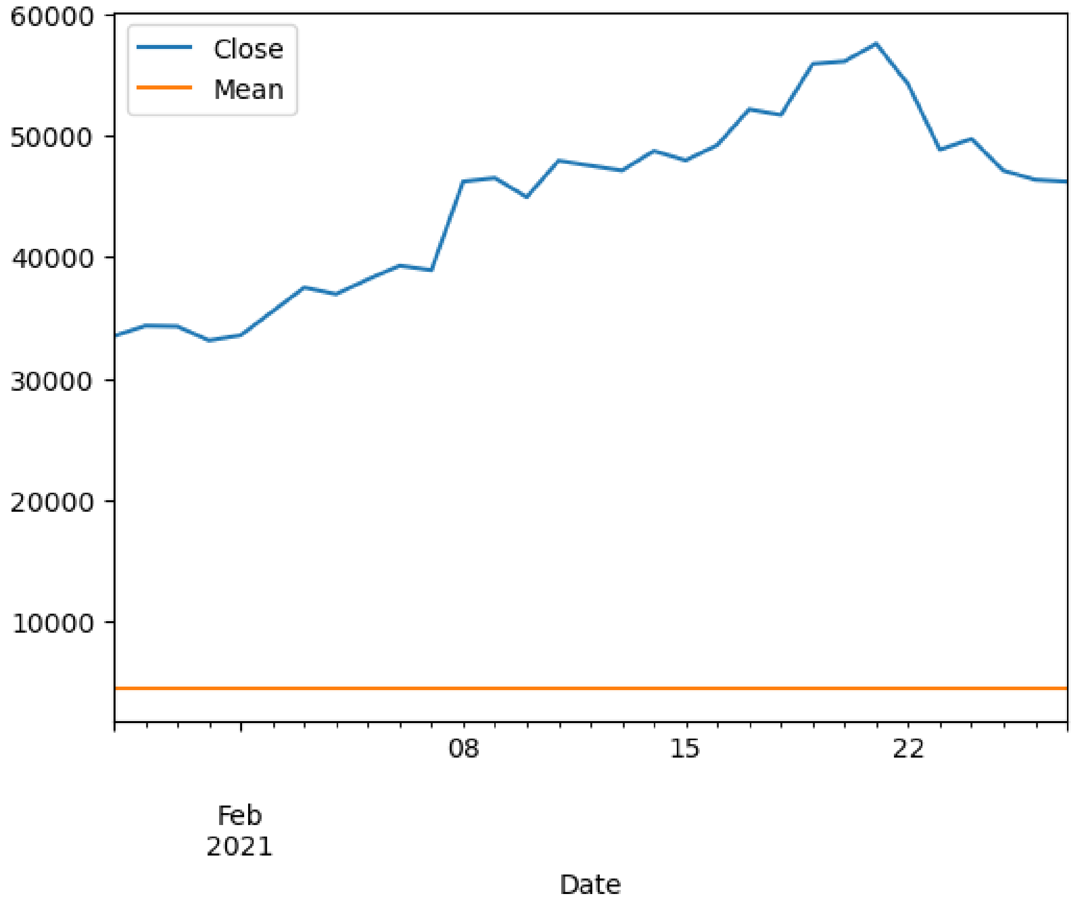
```
df_train["Mean"] = model_mean_pred
df_test["Mean"] = model_mean_pred
```

Ploteo de las predicciones vs las series reales, en train y test:

[19]:
```
df_train.plot(kind="line", y = ["Close", "Mean"]);
```



[20]:
```
df_test.plot(kind="line", y = ["Close", "Mean"]);
```



Se define una función para calcular el RMSE:

[21]:
```
def RMSE(predicted, actual):
    mse = (predicted - actual) ** 2
    rmse = np.sqrt(mse.sum() / mse.count())
    return rmse
```

Se define una función para calcular el MAPE:

[22]:
```
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

[23]:
```
# Aplicación del MAPE en Mean
model_MAPE = mean_absolute_percentage_error(df_test.Close , df_test.Mean)
print("El MAPE es ", model_MAPE)
```

```
El MAPE es  89.84516144847098
```

[24]:
```python
# Aplicación del RMSE en Mean
print("El RMSE es ", RMSE(df_test.Mean, df_test.Close))
```

```
El RMSE es  40968.433969121186
```

Se guardan los resultados en un DataFrame: El mismo será reutilizado para almacenar los resultados de los distintos modelos a utilizar

[25]:
```python
df_Results = pd.DataFrame(columns = ["Model", "RMSE","MAPE"])
df_Results.loc[0, "Model"] = "Mean"
df_Results.loc[0, "RMSE"] = RMSE(df_test.Mean, df_test.Close)
df_Results.loc[0, "MAPE"] = model_MAPE
df_Results.head()
```

[25]:
```
  Model          RMSE       MAPE
0  Mean  40968.433969  89.845161
```

2.2 b) RandomWalk

Se crea el shift de target en train:

[26]:
```python
df_train["CloseShift"] = df_train.Close.shift()
# La primera observación va a quedar en nan, por lo que se reemplaza por el␣
↪valor siguiente:
df_train["CloseShift"].fillna(method='bfill', inplace=True)
df_train.head()
```

[26]:
```
            SNo     Name Symbol       Date        High         Low  \
Date
2013-04-29    1  Bitcoin    BTC 2013-04-29  147.488007  134.000000
2013-04-30    2  Bitcoin    BTC 2013-04-30  146.929993  134.050003
2013-05-01    3  Bitcoin    BTC 2013-05-01  139.889999  107.720001
2013-05-02    4  Bitcoin    BTC 2013-05-02  125.599998   92.281898
2013-05-03    5  Bitcoin    BTC 2013-05-03  108.127998   79.099998

                  Open       Close  Volume     Marketcap  …  Month_6  \
Date                                                      …
2013-04-29  134.444000  144.539993     0.0  1.603769e+09  …        0
2013-04-30  144.000000  139.000000     0.0  1.542813e+09  …        0
2013-05-01  139.000000  116.989998     0.0  1.298955e+09  …        0
2013-05-02  116.379997  105.209999     0.0  1.168517e+09  …        0
2013-05-03  106.250000   97.750000     0.0  1.085995e+09  …        0

            Month_7  Month_8  Month_9  Month_10  Month_11  Month_12  \
Date
2013-04-29        0        0        0         0         0         0
2013-04-30        0        0        0         0         0         0
2013-05-01        0        0        0         0         0         0
```

```
2013-05-02         0         0         0         0         0         0
2013-05-03         0         0         0         0         0         0

            log_value         Mean   CloseShift
Date
2013-04-29   4.973556  4415.425613   144.539993
2013-04-30   4.934474  4415.425613   144.539993
2013-05-01   4.762088  4415.425613   139.000000
2013-05-02   4.655958  4415.425613   116.989998
2013-05-03   4.582413  4415.425613   105.209999

[5 rows x 27 columns]
```

Se crea el shift de target en test:

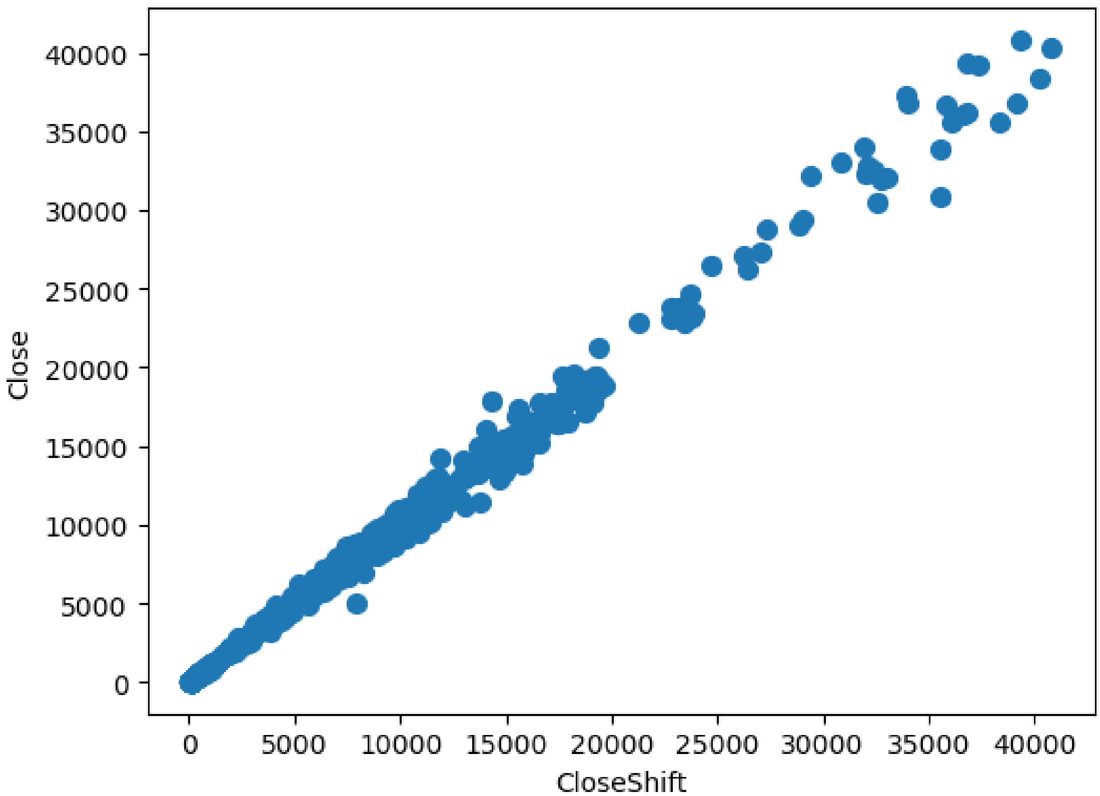
[27]:
```python
df_test["CloseShift"] = df_test.Close.shift()
# Se puede reemplazar el primer nan con el último valor del set de␣
↪entrenamiento:
df_test.iloc[0,26] = df_train.iloc[-1,0]
df_test.head()
```

[27]:
```
            SNo     Name Symbol        Date         High          Low  \
Date
2021-01-28  2832  Bitcoin    BTC 2021-01-28  33858.31099  30023.20683
2021-01-29  2833  Bitcoin    BTC 2021-01-29  38406.26096  32064.81419
2021-01-30  2834  Bitcoin    BTC 2021-01-30  34834.70830  32940.18691
2021-01-31  2835  Bitcoin    BTC 2021-01-31  34288.33148  32270.17602
2021-02-01  2836  Bitcoin    BTC 2021-02-01  34638.21349  32384.22811

                  Open        Close        Volume     Marketcap  …  \
Date                                                             …
2021-01-28  30441.04182  33466.09636  7.651716e+10  6.229100e+11  …
2021-01-29  34318.67169  34316.38765  1.178950e+11  6.387690e+11  …
2021-01-30  34295.93504  34269.52154  6.514183e+10  6.379250e+11  …
2021-01-31  34270.87759  33114.35775  5.275454e+10  6.164530e+11  …
2021-02-01  33114.57724  33537.17682  6.140040e+10  6.243490e+11  …

            Month_6  Month_7  Month_8  Month_9  Month_10  Month_11  Month_12  \
Date
2021-01-28        0        0        0        0         0         0         0
2021-01-29        0        0        0        0         0         0         0
2021-01-30        0        0        0        0         0         0         0
2021-01-31        0        0        0        0         0         0         0
2021-02-01        0        0        0        0         0         0         0

            log_value         Mean   CloseShift
Date
2021-01-28  10.418288  4415.425613   2831.00000
```

```
2021-01-29  10.443378  4415.425613  33466.09636
2021-01-30  10.442012  4415.425613  34316.38765
2021-01-31  10.407722  4415.425613  34269.52154
2021-02-01  10.420410  4415.425613  33114.35775

[5 rows x 27 columns]
```

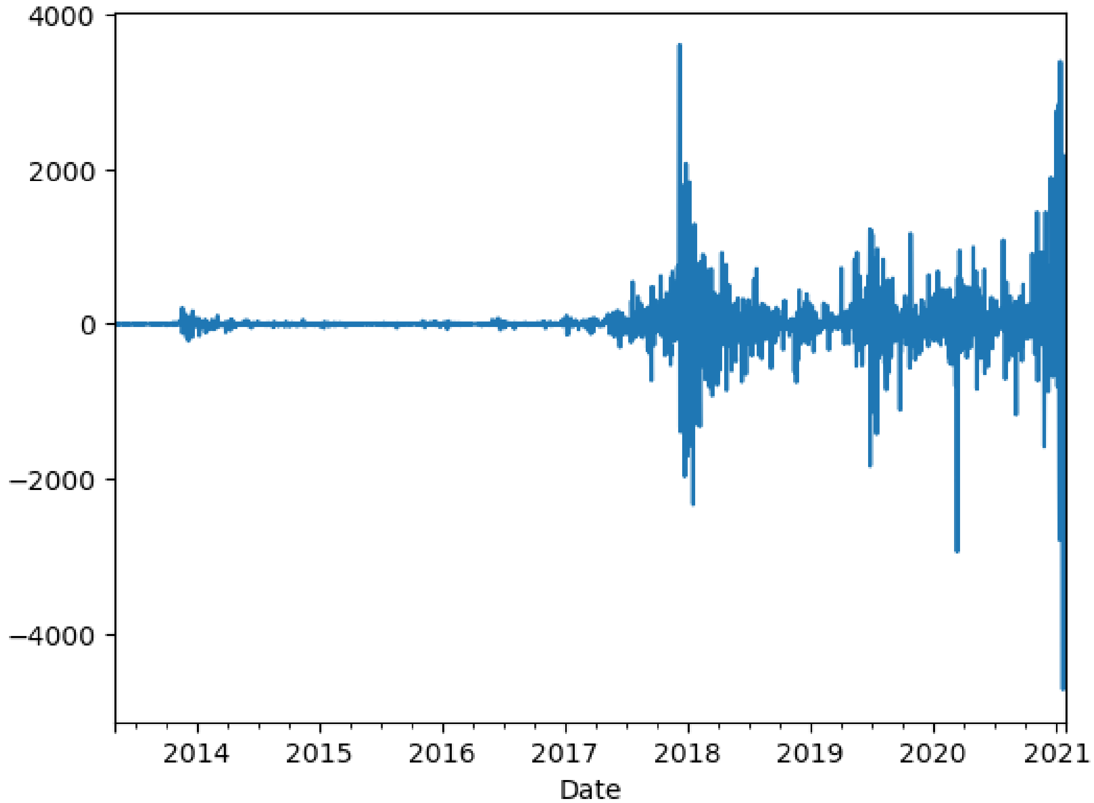
Lag de un período:

[28]:
```python
df_train.plot(kind= "scatter", y = "Close", x = "CloseShift", s = 50);
```



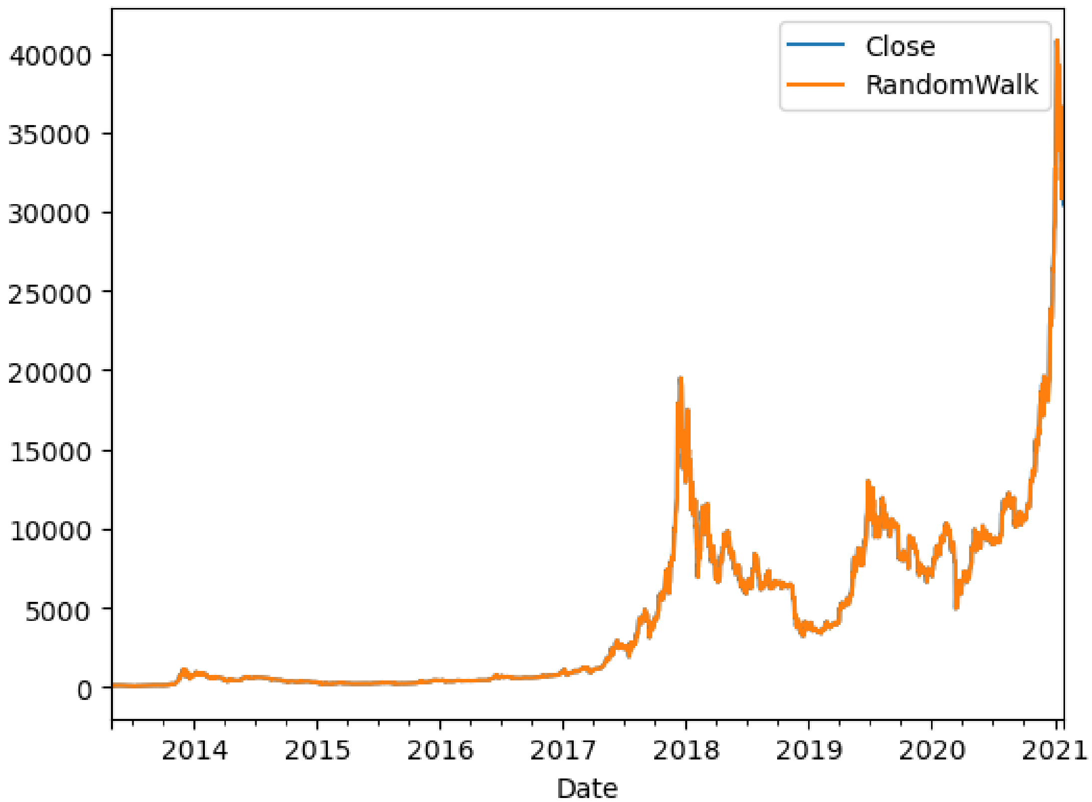
Diferencias entre Target y el lag:

[29]:
```python
df_train["Closediff"] = df_train.Close - df_train.CloseShift
df_train.Closediff.plot();
```



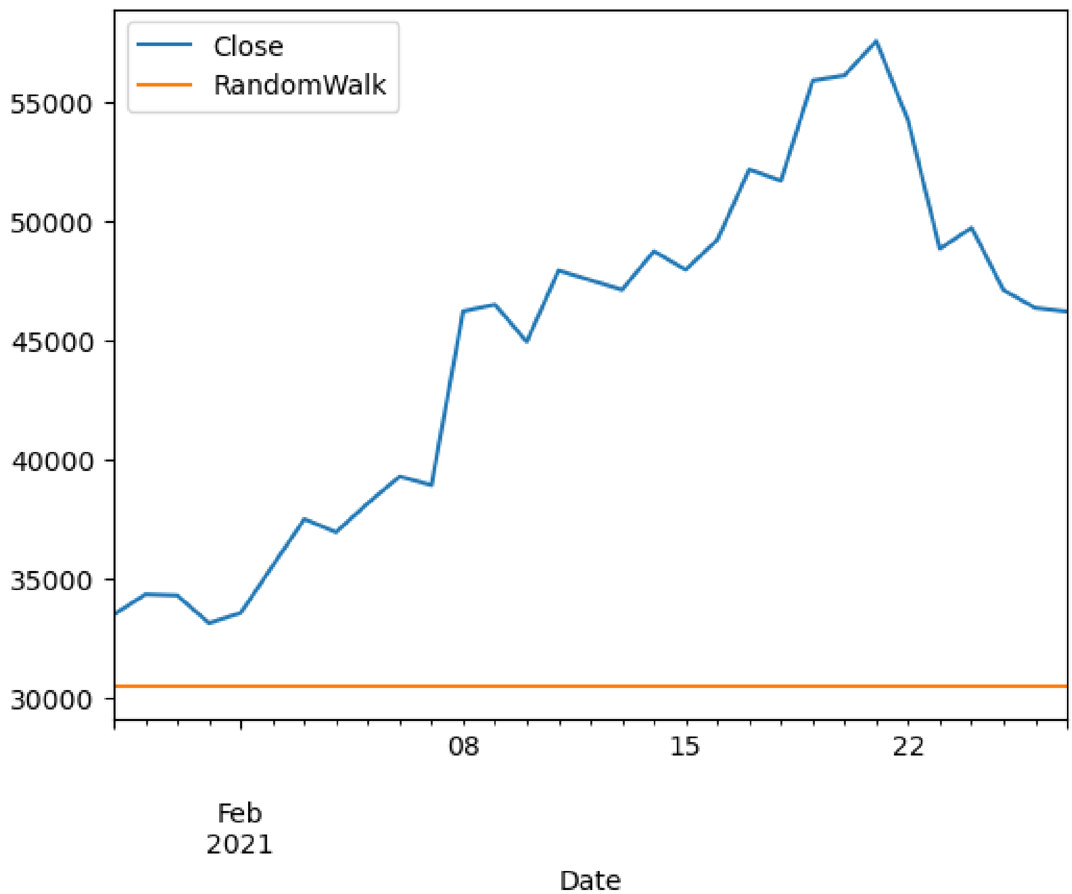
Ploteo de las predicciones vs las series reales, en train y test:

[30]:
```
df_train["RandomWalk"] = df_train.CloseShift
df_train.plot(kind="line", y = ["Close", "RandomWalk"]);
```



[31]:
```
df_test["RandomWalk"] = pd.Series(df_train["Close"][-1], index=df_test.index)
```

[32]:
```
df_test.plot(kind="line", y = ["Close", "RandomWalk"]);
```



Se calcula el MAPE + RMSE y se almacena

[33]:
```
model_MAPE = mean_absolute_percentage_error (df_test.Close , df_test.RandomWalk)
```

[34]:
```
df_Results.loc[1, "Model"] = "Random Walk"
df_Results.loc[1, "RMSE"] = RMSE(df_test.RandomWalk, df_test.Close)
df_Results.loc[1, "MAPE"] = model_MAPE
df_Results
```

[34]:

	Model	RMSE	MAPE
0	Mean	40968.433969	89.845161
1	Random Walk	16049.403917	30.009555

[35]:
```
# Para un resumen, se utiliza el Summary:
model_linear = smf.ols('Close ~ timeIndex', data = df_train).fit()
```

[36]:
```
model_linear.summary()
```

[36]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Close   R-squared:                       0.582
Model:                            OLS   Adj. R-squared:                  0.582
Method:                 Least Squares   F-statistic:                     3945.
Date:                Tue, 08 Aug 2023   Prob (F-statistic):               0.00
Time:                        21:29:28   Log-Likelihood:                -27197.
No. Observations:                2831   AIC:                         5.440e+04
Df Residuals:                    2829   BIC:                         5.441e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -2939.2871    135.218    -21.737      0.000   -3204.423   -2674.151
timeIndex      5.1977      0.083     62.811      0.000       5.035       5.360
==============================================================================
Omnibus:                     2081.927   Durbin-Watson:                   0.008
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            43129.622
Skew:                           3.304   Prob(JB):                         0.00
Kurtosis:                      20.943   Cond. No.                     3.27e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.27e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

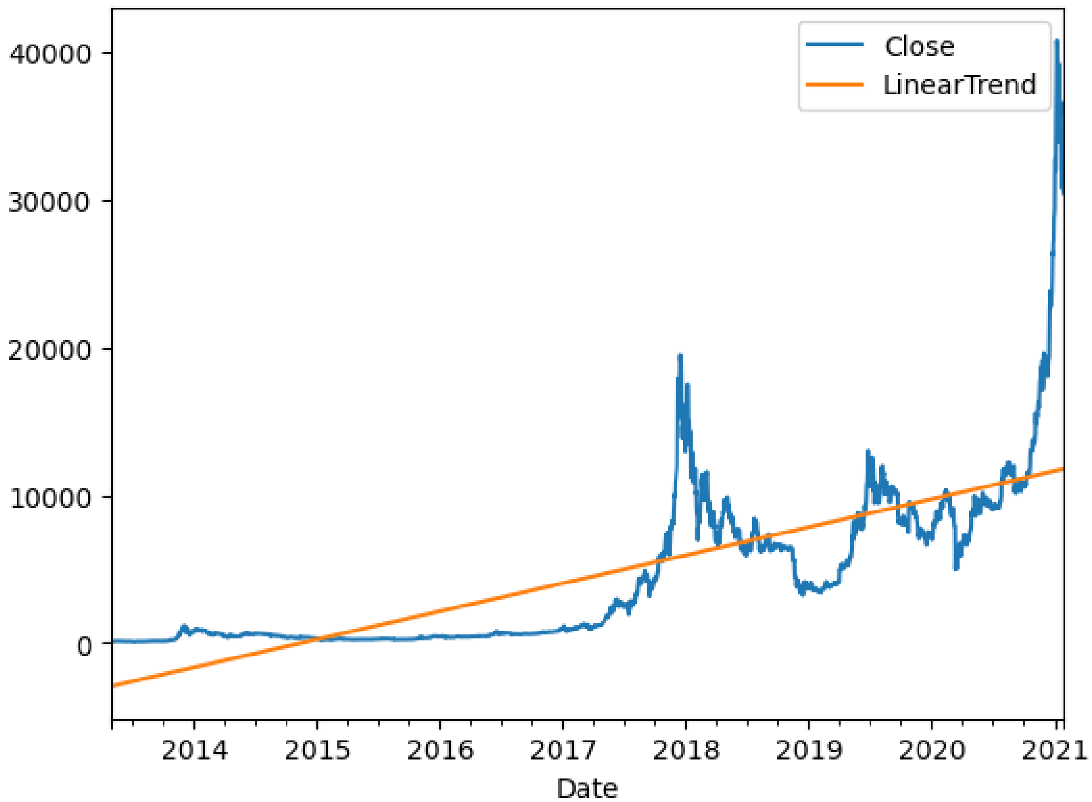
2.3 c) Linear Trend

Se crea una columna en train con el predict:

[37]:
```
df_train["LinearTrend"] = model_linear.predict(df_train.timeIndex)
```

Ploteo de las predicciones vs las series reales, en train y test:

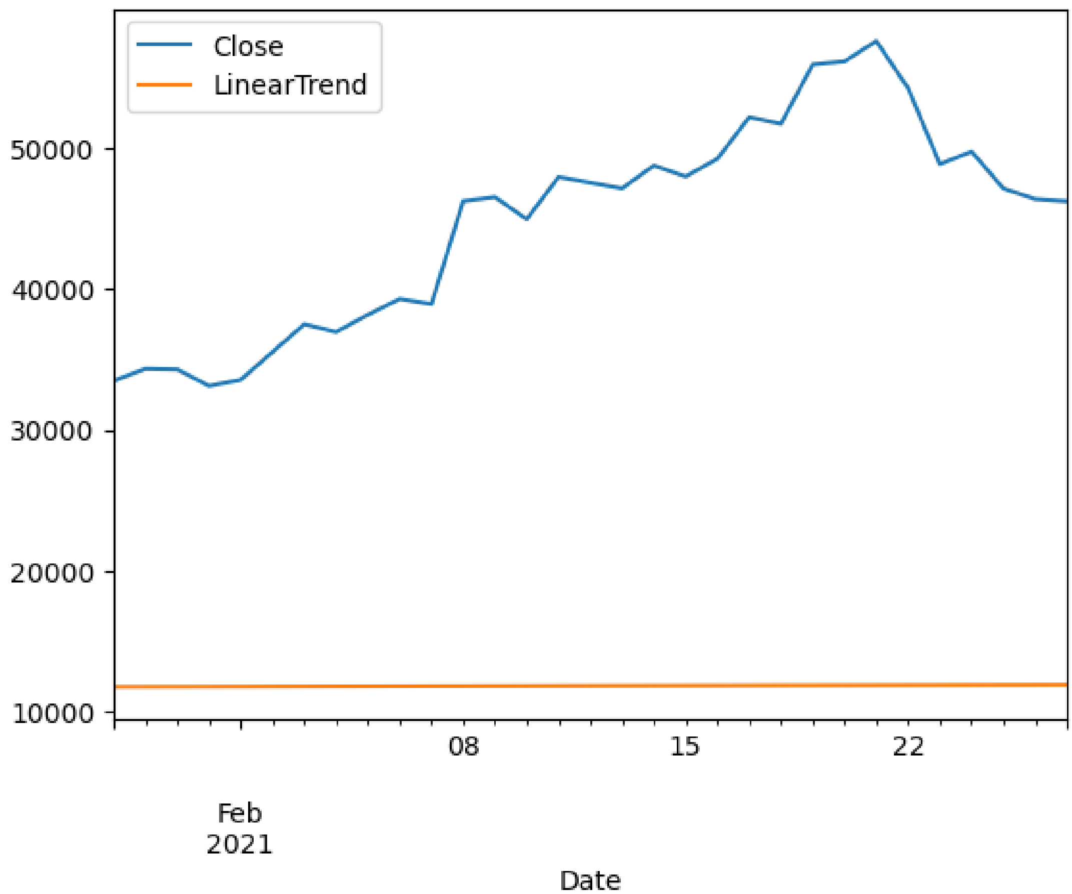
[38]:
```
df_train.plot(kind = "line", y = ["Close","LinearTrend"]);
```



Se repete en Test:

[39]:
```
df_test["LinearTrend"] = model_linear.predict(df_test.timeIndex)
```

[40]:
```
df_test.plot(kind = "line", y = ["Close","LinearTrend"]);
```



Se calcula el MAPE + RMSE y se almacena

[41]:
```
model_MAPE = mean_absolute_percentage_error (df_test.Close , df_test.
  ↪LinearTrend)
```

[42]:
```
df_Results.loc[2, "Model"] = "LinearTrend"
df_Results.loc[2, "RMSE"] = RMSE(df_test.LinearTrend, df_test.Close)
df_Results.loc[2, "MAPE"] = model_MAPE
df_Results
```

[42]:

	Model	RMSE	MAPE
0	Mean	40968.433969	89.845161
1	Random Walk	16049.403917	30.009555
2	LinearTrend	33666.922165	72.755041

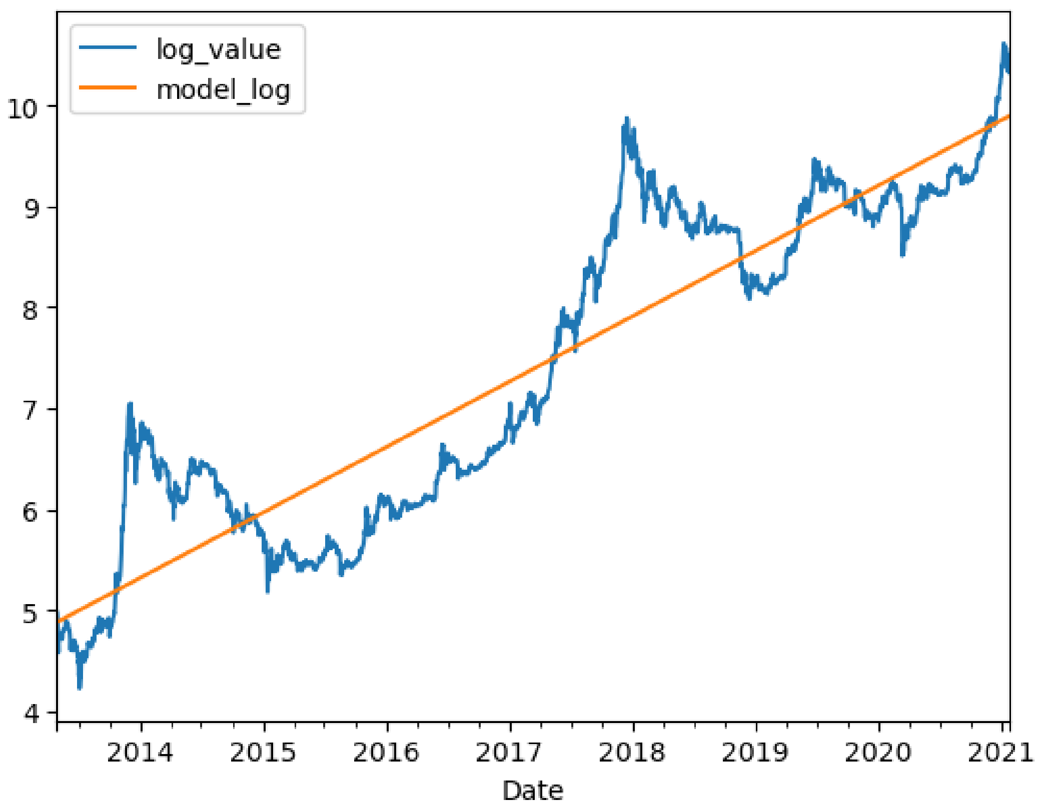
2.4 d) Transf Log

[43]:
```
df_train['model_log'] = model_log.predict(df_train[["timeIndex"]])
df_test['model_log'] = model_log.predict(df_test[["timeIndex"]])
```

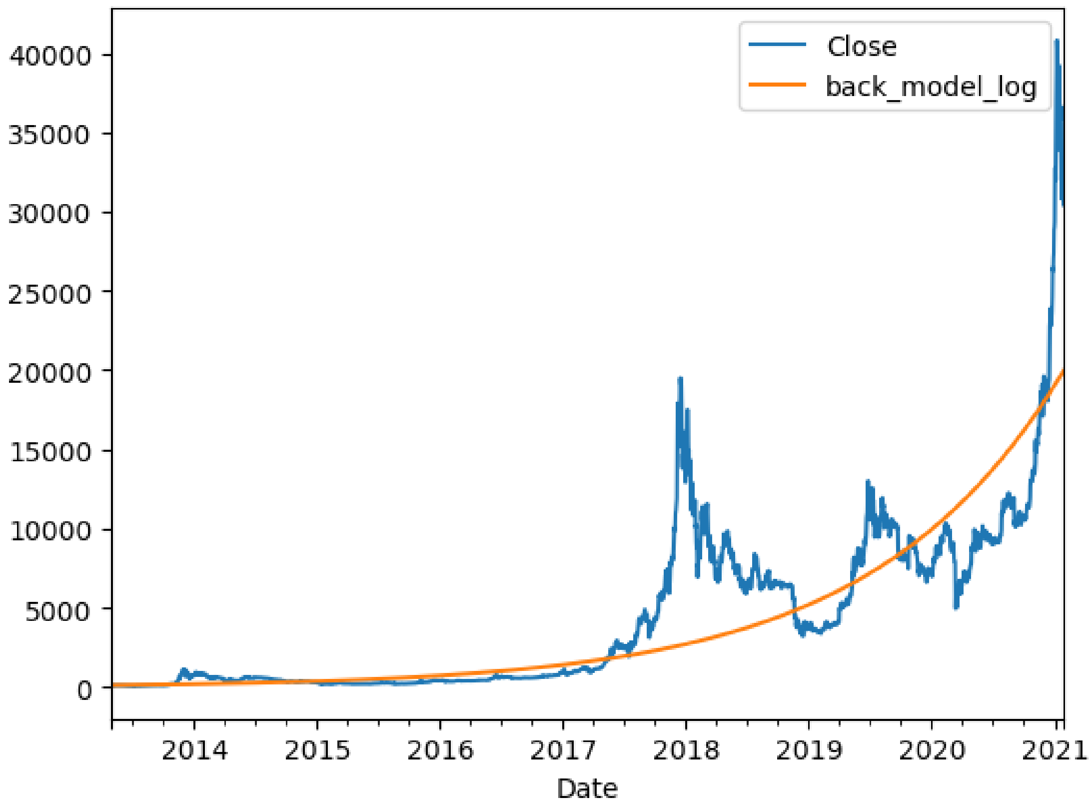
[44]:
```
df_train['back_model_log'] = np.exp(df_train['model_log'])
df_test['back_model_log'] = np.exp(df_test['model_log'])
```

Ploteo de las predicciones vs las series reales, en train y test:

[45]:
```
df_train.plot(kind = "line", x = "Date", y = ['log_value', 'model_log']);
```

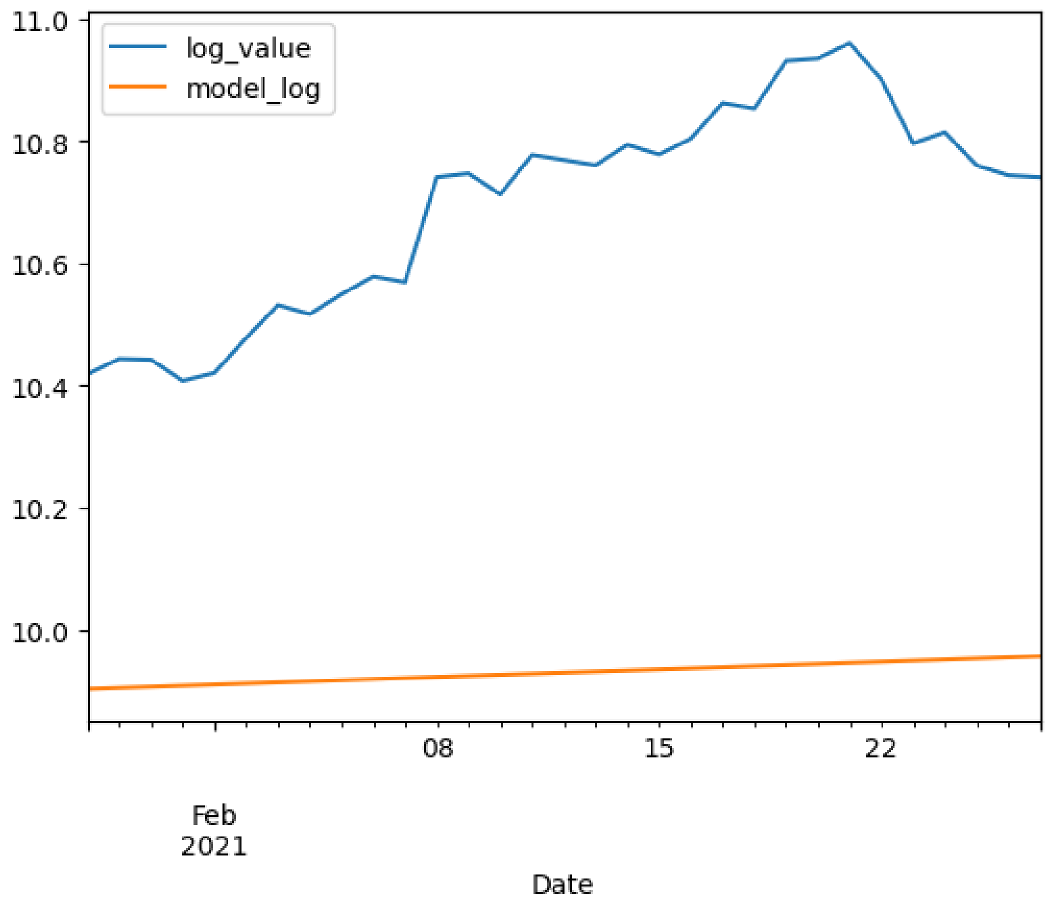


[46]:
```
df_train.plot(kind = "line", x = "Date", y = ['Close', 'back_model_log']);
```

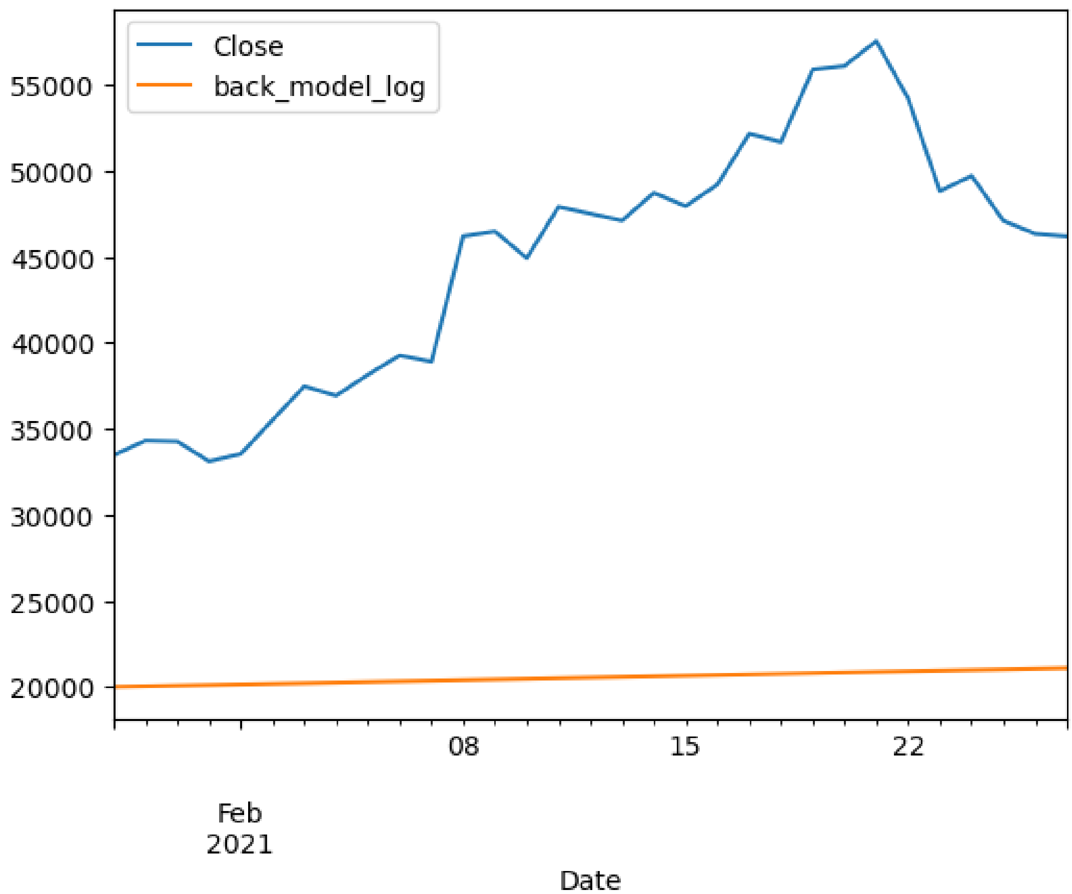


[47]:

```
df_test.plot(kind = "line", x = "Date", y = ['log_value', 'model_log']);
```




```
[48]: df_test.plot(kind = "line", x = "Date", y = ['Close', 'back_model_log']);
```



Se calcula el MAPE + RMSE y se almacena

[49]:
```
model_MAPE = mean_absolute_percentage_error (df_test.Close , df_test.
  ↪back_model_log)
```

[50]:
```
df_Results.loc[3, "Model"] = "Transf Log"
df_Results.loc[3, "RMSE"] = RMSE(df_test['back_model_log'], df_test['Close'])
df_Results.loc[3, "MAPE"] = model_MAPE
df_Results
```

[50]:

	Model	RMSE	MAPE
0	Mean	40968.433969	89.845161
1	Random Walk	16049.403917	30.009555
2	LinearTrend	33666.922165	72.755041
3	Transf Log	25190.683145	52.878589

2.5 e) Transf Log + Est

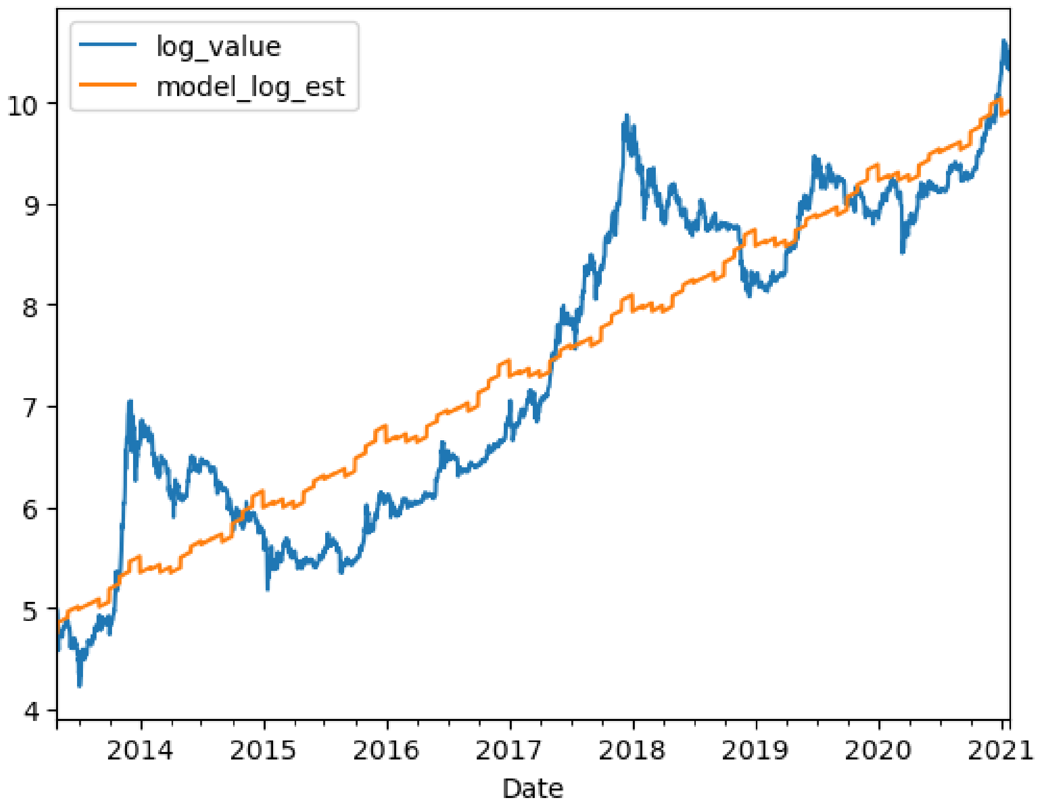
[51]:
```
model_log_est = smf.ols('log_value ~ timeIndex + Month_2 + Month_3 + Month_4 +␣
↪Month_5 + Month_6 + Month_7 + Month_8 + Month_9 + Month_11 + Month_12',\
                          data = df_train).fit()
```

[52]:
```
df_train['model_log_est'] = model_log_est.predict(df_train[["timeIndex",\
                                              "Month_2", "Month_3", "Month_4",␣
↪"Month_5",\
                                               "Month_6","Month_7", "Month_8",␣
↪"Month_9","Month_10","Month_11","Month_12"]])


df_test['model_log_est'] = model_log_est.predict(df_test[["timeIndex",\
                                               "Month_2", "Month_3", "Month_4",␣
↪"Month_5",\
                                               "Month_6","Month_7", "Month_8",␣
↪"Month_9","Month_10","Month_11","Month_12"]])
```

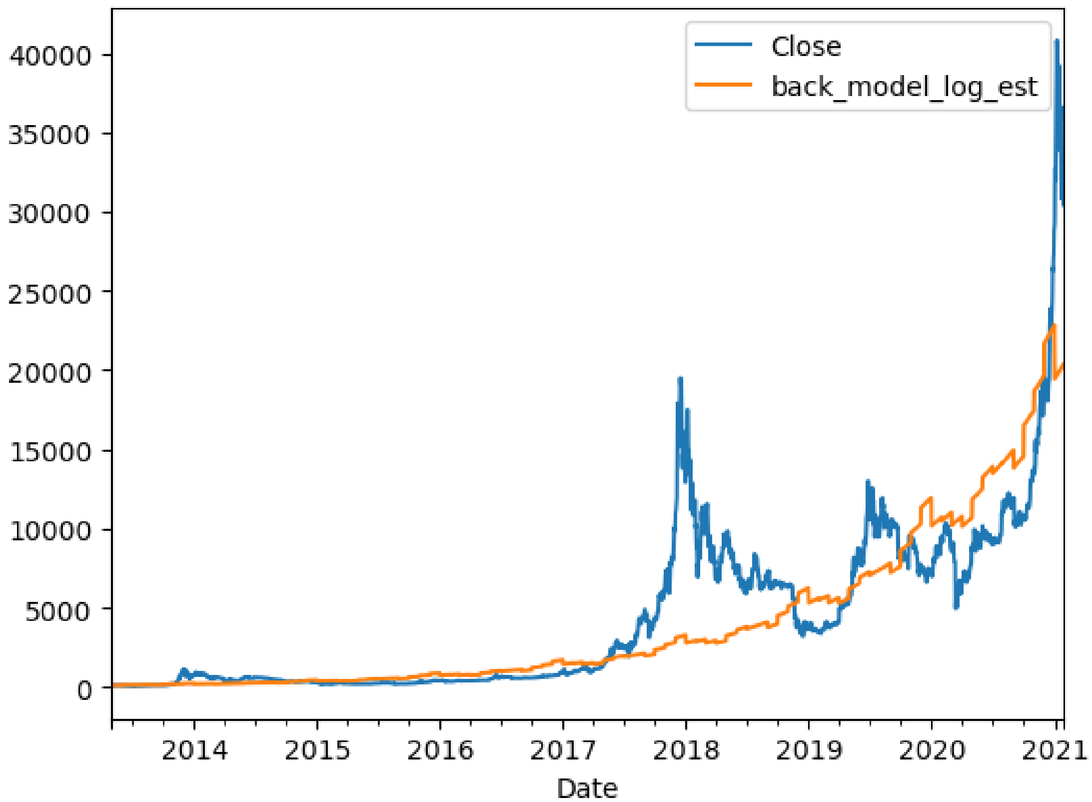
[53]:
```
df_train['back_model_log_est'] = np.exp(df_train['model_log_est'])
df_test['back_model_log_est'] = np.exp(df_test['model_log_est'])
```

Ploteo de las predicciones vs las series reales, en train y test:

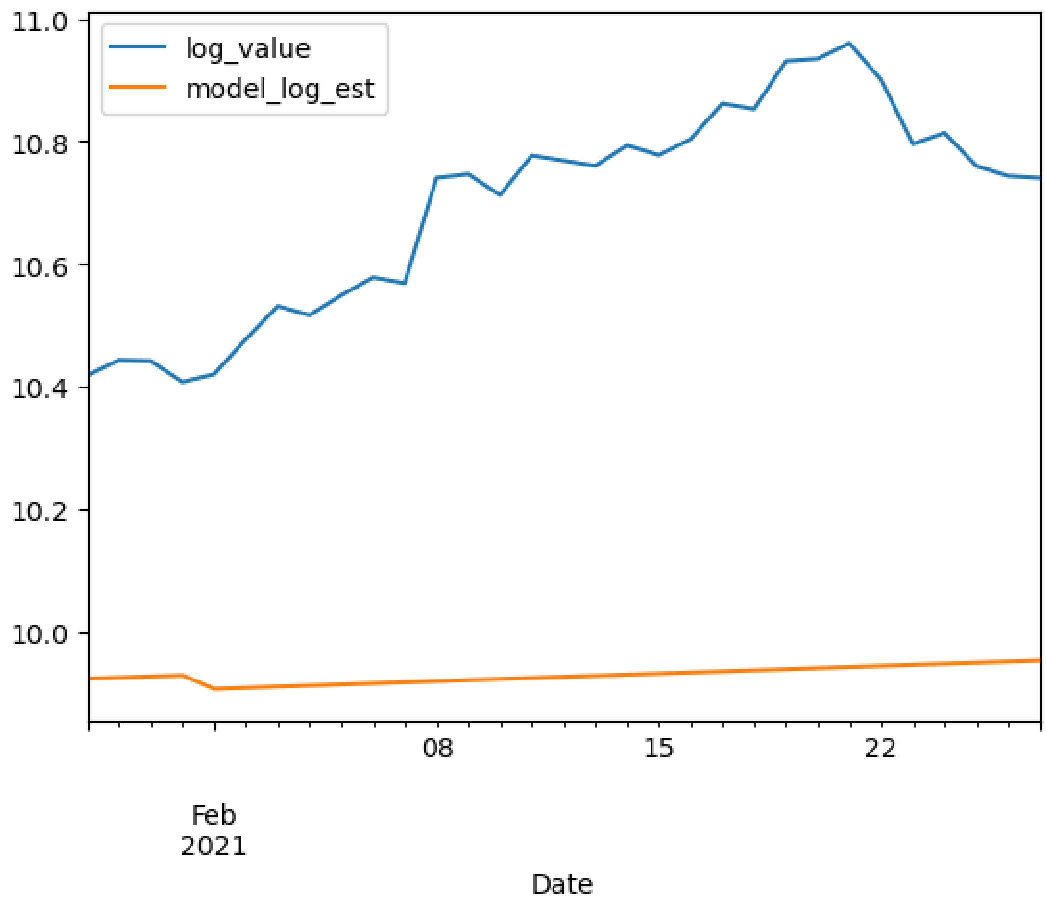
[54]:
```
df_train.plot(kind = "line", x = "Date", y = ['log_value', 'model_log_est']);
```



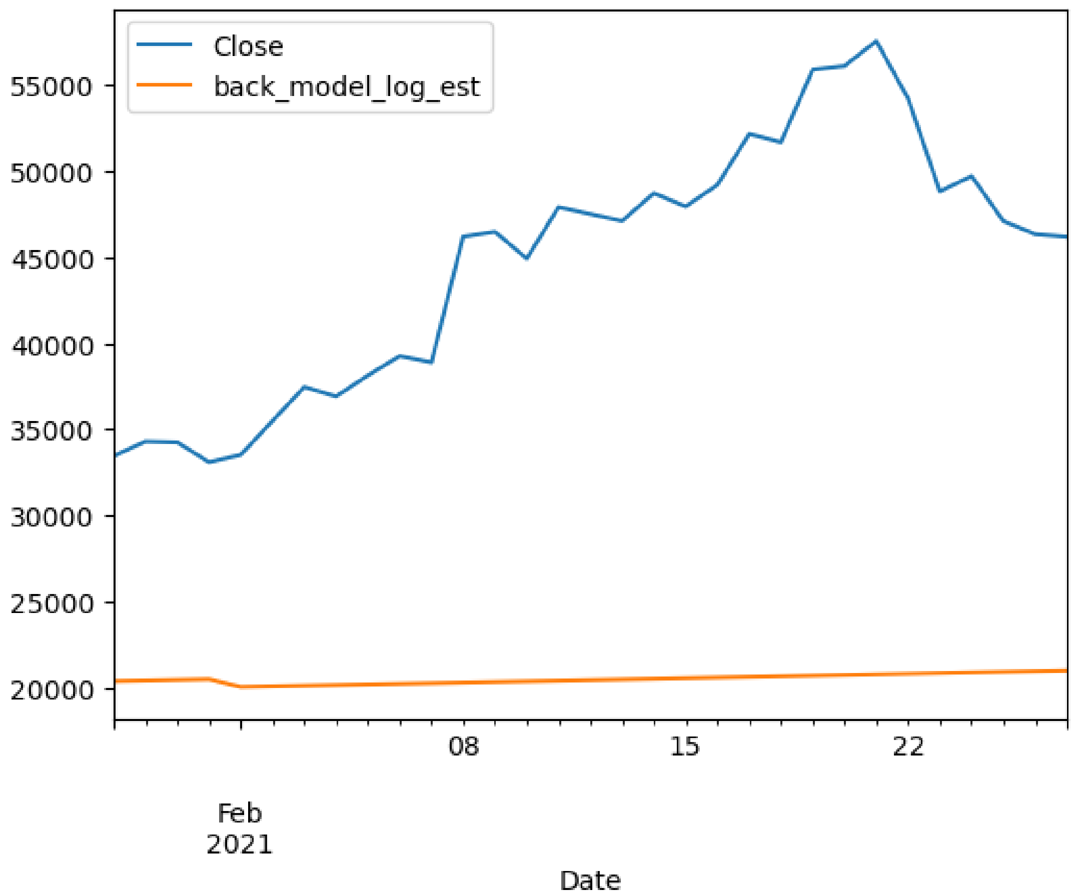
[55]:
```
df_train.plot(kind = "line", x = "Date", y = ['Close', 'back_model_log_est']);
```



[56]:
```
df_test.plot(kind = "line", x = "Date", y = ['log_value', 'model_log_est']);
```



[57]:
```
df_test.plot(kind = "line", x = "Date", y = ['Close', 'back_model_log_est']);
```



Se calcula el MAPE + RMSE y se almacena

[58]:
```
model_MAPE = mean_absolute_percentage_error (df_test.Close , df_test.
  ↪back_model_log_est)
```

[59]:
```
df_Results.loc[4, "Model"] = "Transf Log + est"
df_Results.loc[4, "RMSE"] = RMSE(df_test['back_model_log_est'],␣
  ↪df_test['Close'])
df_Results.loc[4, "MAPE"] = model_MAPE
df_Results
```

[59]:

	Model	RMSE	MAPE
0	Mean	40968.433969	89.845161
1	Random Walk	16049.403917	30.009555
2	LinearTrend	33666.922165	72.755041
3	Transf Log	25190.683145	52.878589
4	Transf Log + est	25219.962947	52.844384

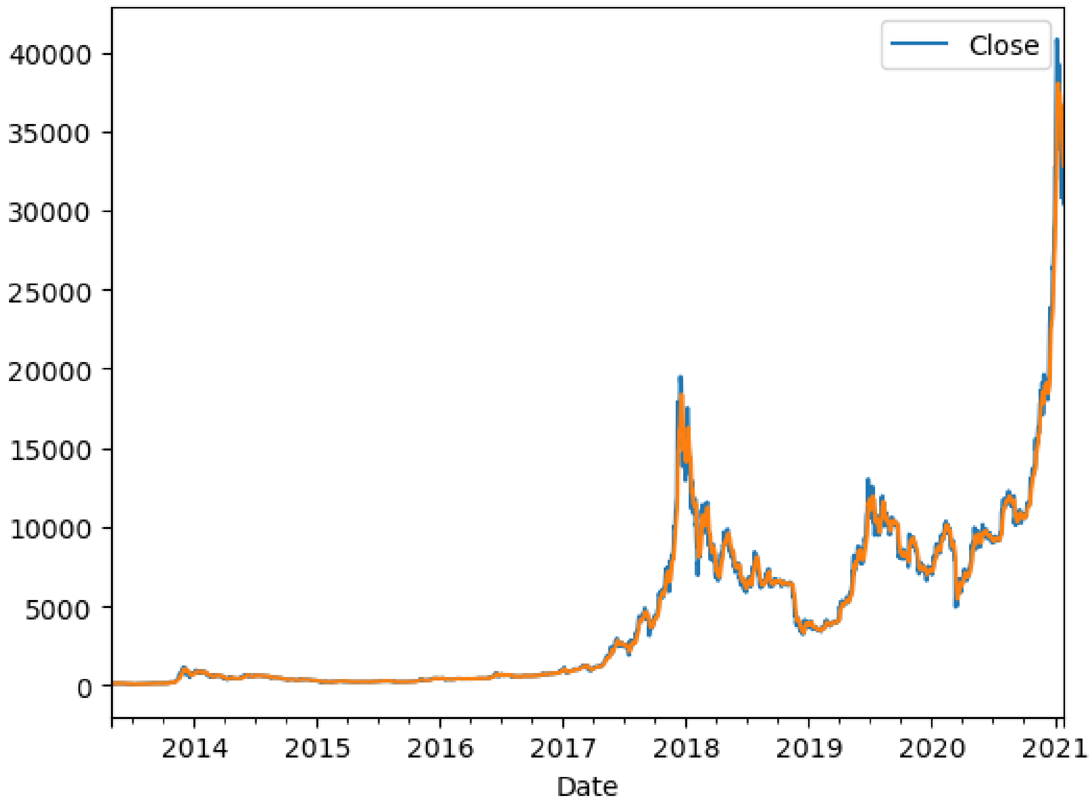
2.6 f) Simple Smoothing

Fiteo del modelo:

[60]:
```
model_exp_smoothing = SimpleExpSmoothing(df_train.Close).fit(smoothing_level=0.
↪3, optimized=False)
```

Ploteo de las predicciones vs las series reales, en train y test:

[61]:
```
df_train.plot(kind = "line", y = "Close")
model_exp_smoothing.fittedvalues.plot();
```



[62]:
```
# Se define cantidad de splits:
tscv = TimeSeriesSplit(n_splits=5)
```

[63]:
```
for train_index, val_index in tscv.split(df_train):
    print("TRAIN:", train_index, "VAL:", val_index)
```

```
TRAIN: [   0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17
  18   19   20   21   22   23   24   25   26   27   28   29   30   31   32   33   34   35
  36   37   38   39   40   41   42   43   44   45   46   47   48   49   50   51   52   53
  54   55   56   57   58   59   60   61   62   63   64   65   66   67   68   69   70   71
  72   73   74   75   76   77   78   79   80   81   82   83   84   85   86   87   88   89
  90   91   92   93   94   95   96   97   98   99  100  101  102  103  104  105  106  107
```

```
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
 468 469 470 471 472 473 474 475] VAL: [476 477 478 479 480 481 482 483 484 485
486 487 488 489 490 491 492 493
 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511
 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529
 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547
 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565
 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583
 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601
 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619
 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637
 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655
 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673
 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691
 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709
 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727
 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745
 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763
 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781
 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799
 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817
 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835
 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853
 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871
 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889
 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907
 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925
 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943
 944 945 946]
```

```
TRAIN: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503
 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521
 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539
 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593
 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611
 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629
 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647
 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665
 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683
 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701
 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719
 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737
 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755
 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773
 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791
 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809
 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827
 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845
 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863
```

```
 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881
 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899
 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917
 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935
 936 937 938 939 940 941 942 943 944 945 946] VAL: [ 947  948  949  950  951
952  953  954  955  956  957  958  959  960
  961  962  963  964  965  966  967  968  969  970  971  972  973  974
  975  976  977  978  979  980  981  982  983  984  985  986  987  988
  989  990  991  992  993  994  995  996  997  998  999 1000 1001 1002
 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016
 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030
 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044
 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058
 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072
 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086
 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100
 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114
 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128
 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142
 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156
 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170
 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184
 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198
 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212
 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226
 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240
 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254
 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268
 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282
 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296
 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310
 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324
 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338
 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352
 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366
 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380
 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394
 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408
 1409 1410 1411 1412 1413 1414 1415 1416 1417]
TRAIN: [   0    1    2 … 1415 1416 1417] VAL: [1418 1419 1420 1421 1422 1423
1424 1425 1426 1427 1428 1429 1430 1431
 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445
 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459
 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473
 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487
 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501
 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515
 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529
```

```
 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543
 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557
 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571
 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585
 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599
 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613
 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627
 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641
 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655
 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669
 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683
 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697
 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711
 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725
 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739
 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753
 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767
 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781
 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795
 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809
 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823
 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837
 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851
 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865
 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879
 1880 1881 1882 1883 1884 1885 1886 1887 1888]
TRAIN: [   0    1    2 … 1886 1887 1888] VAL: [1889 1890 1891 1892 1893 1894
1895 1896 1897 1898 1899 1900 1901 1902
 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916
 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930
 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944
 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958
 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972
 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986
 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000
 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028
 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042
 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056
 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070
 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084
 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098
 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112
 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126
 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140
 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154
 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168
 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182
```

```
 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196
 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210
 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224
 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238
 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252
 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266
 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280
 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294
 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308
 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322
 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336
 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350
 2351 2352 2353 2354 2355 2356 2357 2358 2359]
TRAIN: [   0    1    2 … 2357 2358 2359] VAL: [2360 2361 2362 2363 2364 2365
2366 2367 2368 2369 2370 2371 2372 2373
 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387
 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401
 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415
 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429
 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443
 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457
 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471
 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485
 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499
 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513
 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527
 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541
 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555
 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569
 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583
 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597
 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611
 2612 2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625
 2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638 2639
 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653
 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2665 2666 2667
 2668 2669 2670 2671 2672 2673 2674 2675 2676 2677 2678 2679 2680 2681
 2682 2683 2684 2685 2686 2687 2688 2689 2690 2691 2692 2693 2694 2695
 2696 2697 2698 2699 2700 2701 2702 2703 2704 2705 2706 2707 2708 2709
 2710 2711 2712 2713 2714 2715 2716 2717 2718 2719 2720 2721 2722 2723
 2724 2725 2726 2727 2728 2729 2730 2731 2732 2733 2734 2735 2736 2737
 2738 2739 2740 2741 2742 2743 2744 2745 2746 2747 2748 2749 2750 2751
 2752 2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765
 2766 2767 2768 2769 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779
 2780 2781 2782 2783 2784 2785 2786 2787 2788 2789 2790 2791 2792 2793
 2794 2795 2796 2797 2798 2799 2800 2801 2802 2803 2804 2805 2806 2807
 2808 2809 2810 2811 2812 2813 2814 2815 2816 2817 2818 2819 2820 2821
 2822 2823 2824 2825 2826 2827 2828 2829 2830]
```

Creación de una función para aplicar Cross Validation

[64]:
```
def timeseriesCVscore_exp_smoot(alpha, series):
    """
        Devuelve errores en CV

        slen - longitud de la sesión para modelo Holt-Winters
    """
    # Se crea un array de errores:
    errors = []
    values = series.values

    # Se instancia el objeto que realiza el tscv:
    tscv = TimeSeriesSplit(n_splits=5)

    # Se aplica cross validation:
    for train, test in tscv.split(values):

        model = SimpleExpSmoothing(values[train]).fit(smoothing_level = alpha,␣
 ↪optimized=False)
        predictions = model.forecast(len(test))
        actual = values[test]

        error = mean_squared_error(predictions, actual)
        errors.append(error)

    return np.mean(np.array(errors))
```

Aplicación de la función

[65]:
```
alphas = [0.001, 0.01, 0.1, 0.2, 0.3, 0.35, 0.4, 0.5, 0.7]
errors = []

for alpha in alphas:
    error = timeseriesCVscore_exp_smoot(alpha, df_train.Close)
    errors.append(error)

print('Alpha óptimo:', alphas[np.argmin(errors)])
```

```
Alpha óptimo: 0.1
```

[66]:
```
model_exp_smoothing = SimpleExpSmoothing(df_train.Close).
 ↪fit(smoothing_level=alphas[np.argmin(errors)], optimized=False)
```

[67]:
```
df_test["Simple_Smoothing"] = model_exp_smoothing.forecast(len(df_test))
df_test.head()
```

[67]:
```
                SNo     Name Symbol       Date         High          Low  \
Date
2021-01-28  2832  Bitcoin    BTC 2021-01-28  33858.31099  30023.20683
2021-01-29  2833  Bitcoin    BTC 2021-01-29  38406.26096  32064.81419
2021-01-30  2834  Bitcoin    BTC 2021-01-30  34834.70830  32940.18691
2021-01-31  2835  Bitcoin    BTC 2021-01-31  34288.33148  32270.17602
2021-02-01  2836  Bitcoin    BTC 2021-02-01  34638.21349  32384.22811

                   Open        Close        Volume     Marketcap  …  \
Date                                                              …
2021-01-28  30441.04182  33466.09636  7.651716e+10  6.229100e+11  …
2021-01-29  34318.67169  34316.38765  1.178950e+11  6.387690e+11  …
2021-01-30  34295.93504  34269.52154  6.514183e+10  6.379250e+11  …
2021-01-31  34270.87759  33114.35775  5.275454e+10  6.164530e+11  …
2021-02-01  33114.57724  33537.17682  6.140040e+10  6.243490e+11  …

            log_value         Mean   CloseShift   RandomWalk   LinearTrend  \
Date
2021-01-28  10.418288  4415.425613   2831.00000  30432.54708  11775.336020
2021-01-29  10.443378  4415.425613  33466.09636  30432.54708  11780.533697
2021-01-30  10.442012  4415.425613  34316.38765  30432.54708  11785.731374
2021-01-31  10.407722  4415.425613  34269.52154  30432.54708  11790.929051
2021-02-01  10.420410  4415.425613  33114.35775  30432.54708  11796.126727

            model_log  back_model_log  model_log_est  back_model_log_est  \
Date
2021-01-28   9.903266    19995.562080       9.923727        20408.903014
2021-01-29   9.905039    20031.053582       9.925496        20445.051836
2021-01-30   9.906812    20066.608080       9.927266        20481.264685
2021-01-31   9.908586    20102.225685       9.929035        20517.541675
2021-02-01   9.910359    20137.906512       9.907257        20075.524059

            Simple_Smoothing
Date
2021-01-28       33347.84119
2021-01-29       33347.84119
2021-01-30       33347.84119
2021-01-31       33347.84119
2021-02-01       33347.84119

[5 rows x 34 columns]
```

Se calcula el MAPE + RMSE y se almacena

[68]:
```
model_MAPE = mean_absolute_percentage_error(df_test.Close , df_test.
  ↪Simple_Smoothing)
```

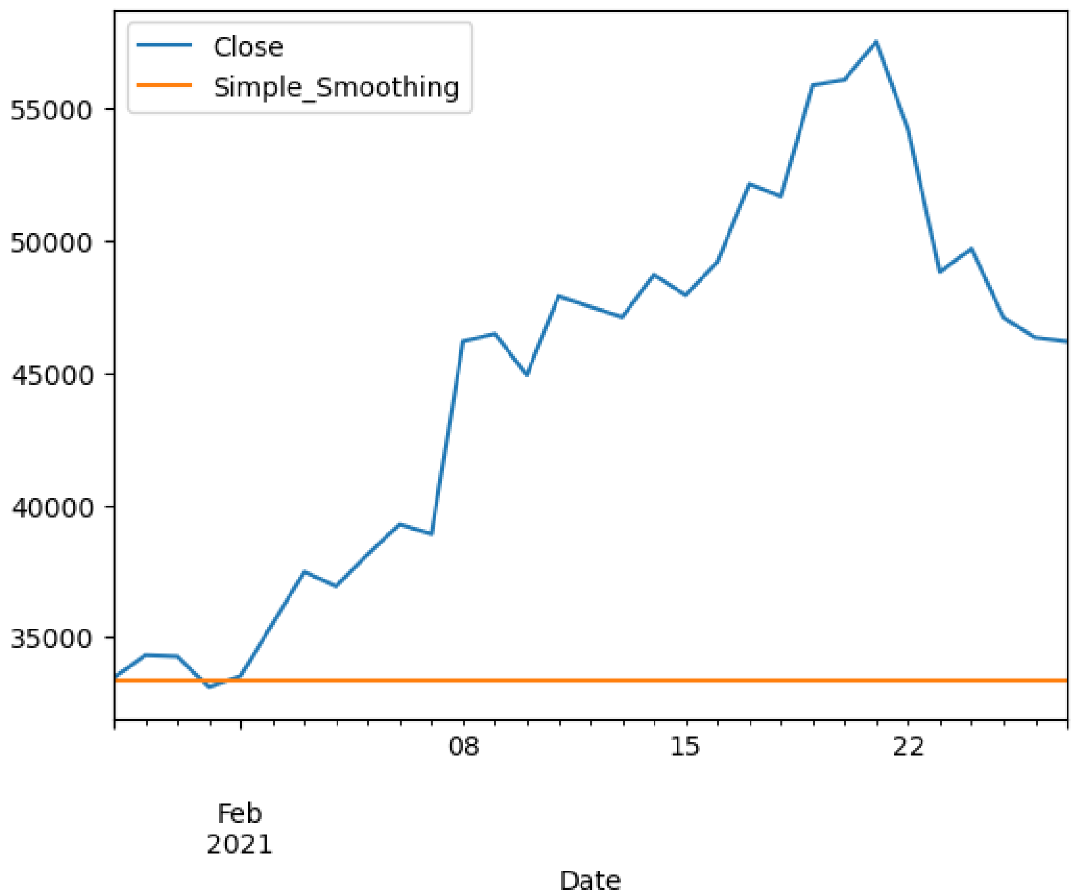
[69]:
```
# Calculamos el RMSE y almacenamos los resultados
df_Results.loc[5, "Model"] = "Simple Smoothing"
df_Results.loc[5, "RMSE"] = RMSE(df_test["Simple_Smoothing"], df_test.Close)
df_Results.loc[5, "MAPE"] = model_MAPE

df_Results
```

[69]:

	Model	RMSE	MAPE
0	Mean	40968.433969	89.845161
1	Random Walk	16049.403917	30.009555
2	LinearTrend	33666.922165	72.755041
3	Transf Log	25190.683145	52.878589
4	Transf Log + est	25219.962947	52.844384
5	Simple Smoothing	13517.463589	23.350291

Ploteo de las predicciones vs las series reales, en train y test:

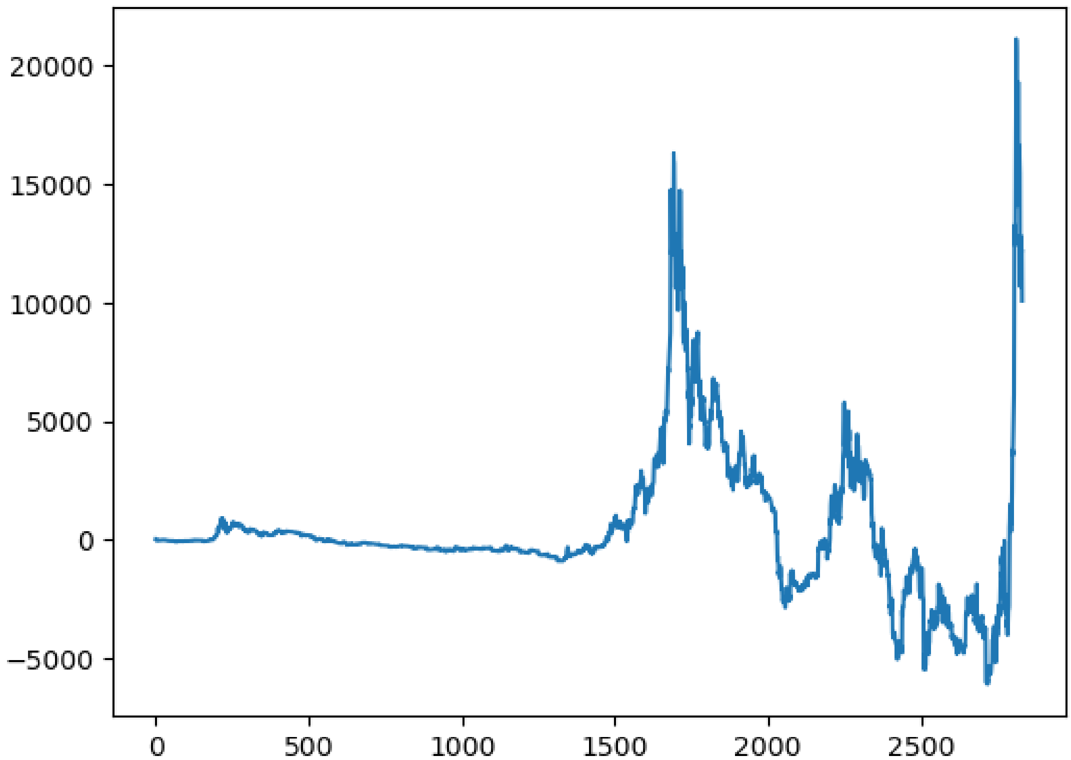
[70]:
```
df_test.plot(kind="line", y = ["Close", "Simple_Smoothing"]);
```



2.7 g) Dickey Fuller + Autocorrelación

Se prueba si los residuos son estacionarios

[71]:
```
residuo = df_train['Close'] - df_train['back_model_log_est']
plt.plot(df_train.timeIndex, residuo, '-');
```



Aplicación de Dickey Fuller al residuo:

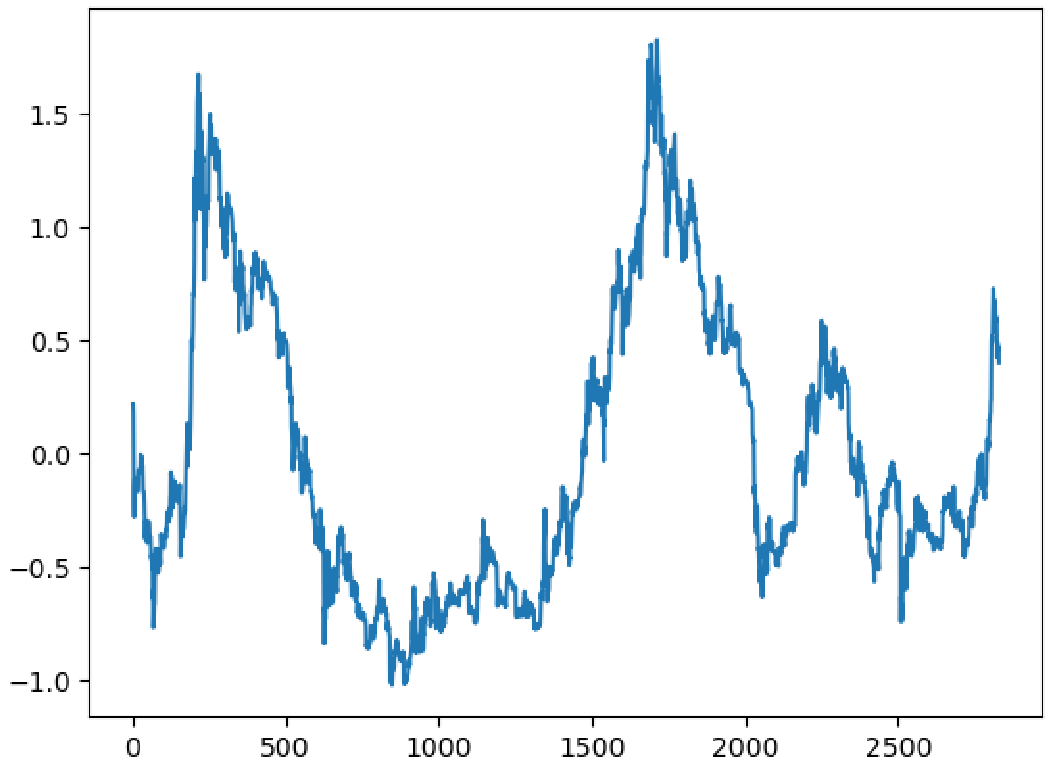
[72]:
```
result = adfuller(residuo)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in  result[4].items():
    print('Valor crítico %s: %.2f' % (key,value))
```

```
ADF Statistic: -2.831115
p-value: 0.053965
Valor crítico 1%: -3.43
Valor crítico 5%: -2.86
Valor crítico 10%: -2.57
```

No se puede rechazar la H0 con un nivel de significación del 5%.

[73]:
```
# Se prueba ahora con los residuos antes de realizar back transform:
res_log_est = df_train['log_value'] - df_train['model_log_est']
```

```
plt.plot(df_train.timeIndex, res_log_est, '-');
```



Segundo testeo de la estacionalidad de los residuos:

[74]:
```
from statsmodels.tsa.stattools import adfuller

result = adfuller(res_log_est)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in  result[4].items():
    print('Valor crítico %s: %.2f' % (key,value))
```

```
ADF Statistic: -2.176515
p-value: 0.214879
Valor crítico 1%: -3.43
Valor crítico 5%: -2.86
Valor crítico 10%: -2.57
```

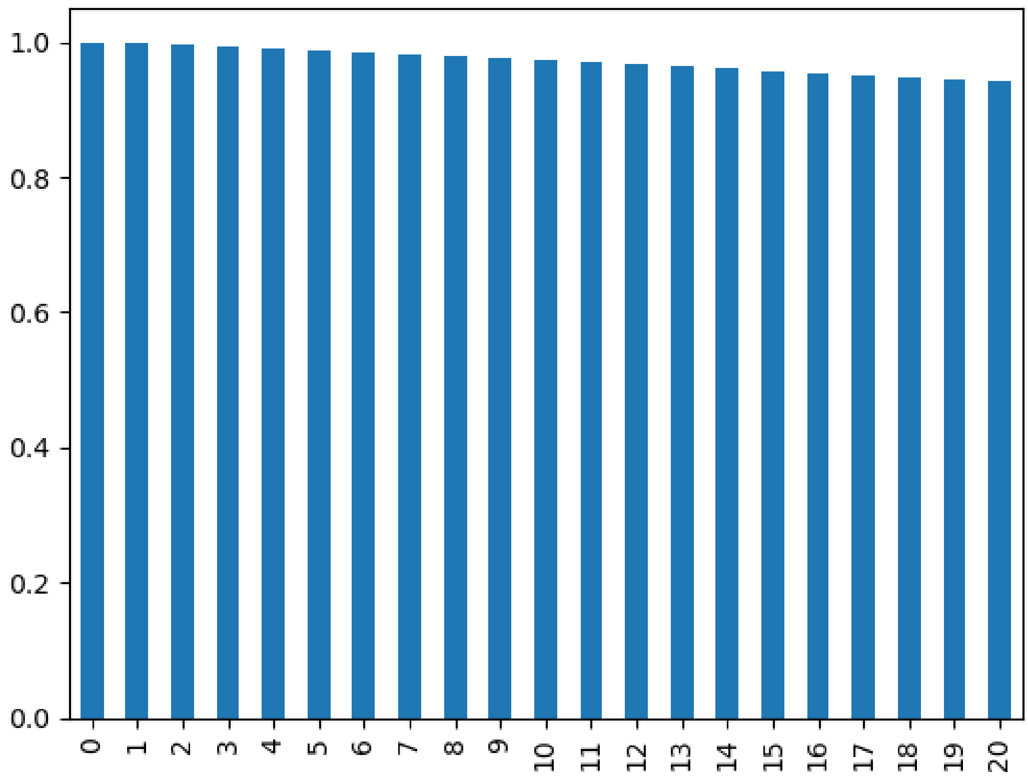
Aún no se puede rechazar la H0, pero se continúa el análisis

[75]:
```
# Cálculo del res_log con 20 rezagos:
lag_acf = acf(res_log_est, nlags = 20)
lag_acf
```

[75]: array([1.   , 0.9973684 , 0.99471906, 0.99215484, 0.98968734,
        0.98710619, 0.98434791, 0.98131918, 0.97840278, 0.97558444,
        0.97280571, 0.969828 , 0.96671826, 0.9635972 , 0.96047466,
        0.95735437, 0.95426951, 0.95116815, 0.94768699, 0.94429688,
        0.94100531])

Se almacena la serie ACF y se plotea:

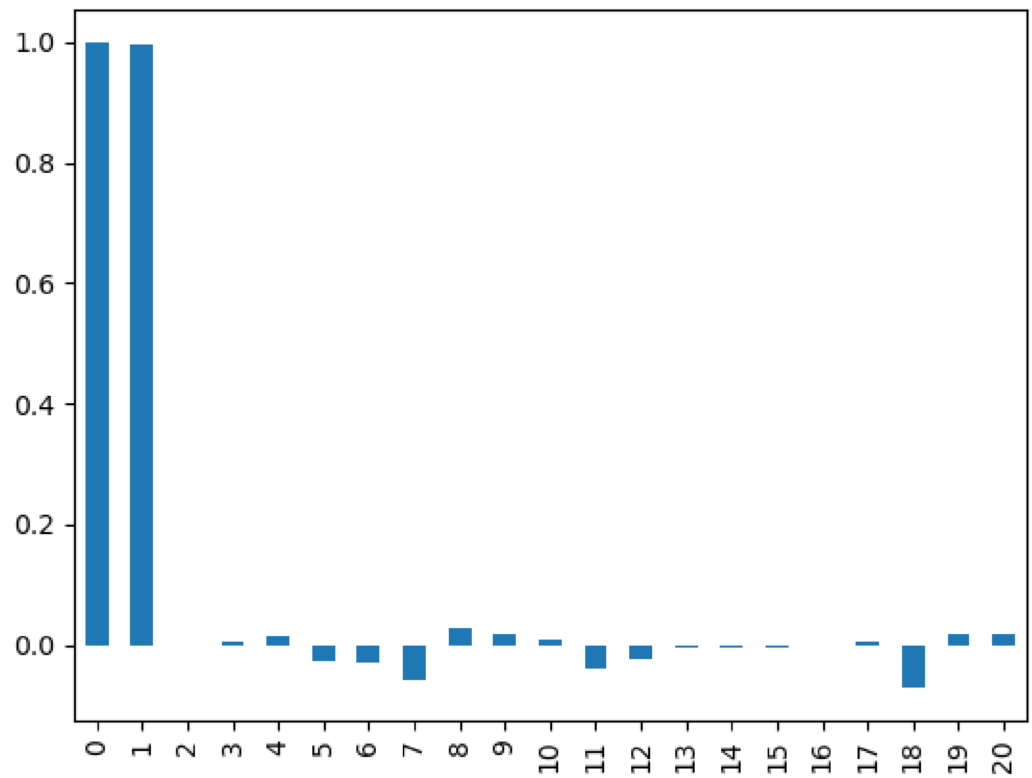
[76]:
```
ACF = pd.Series(lag_acf)
ACF.plot(kind = "bar");
```



Se repiten los pasos anteriores pero con PACF:

[77]:
```
lag_pacf = pacf(res_log_est, nlags=20, method='ols');
```

[78]:
```
PACF = pd.Series(lag_pacf)
PACF.plot(kind = "bar");
```



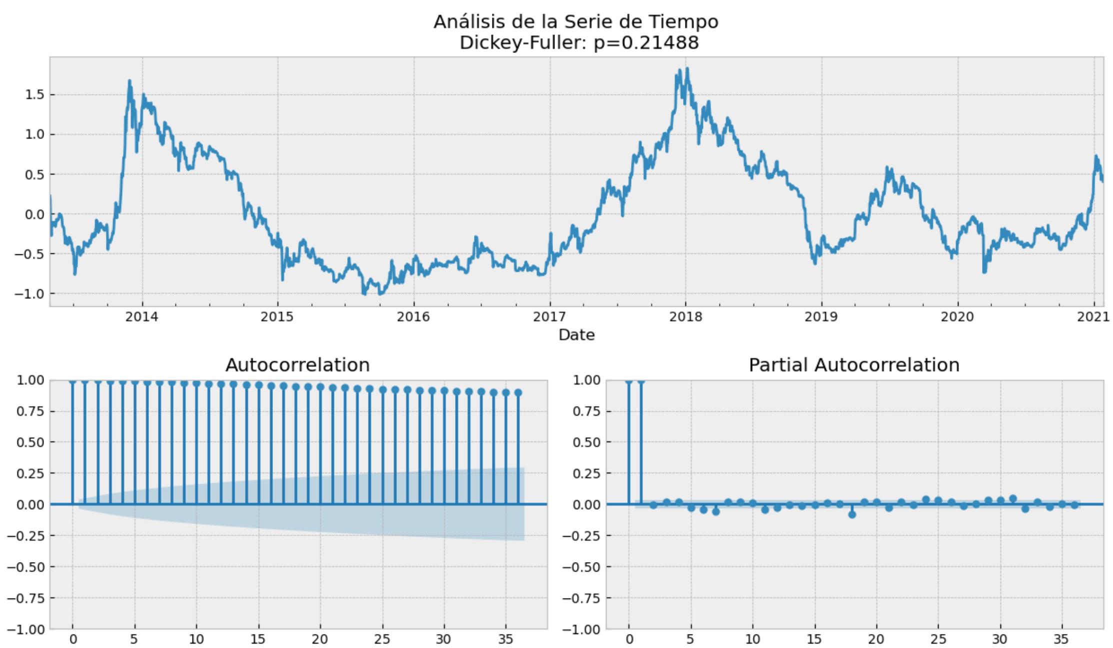
Se puede concluir de este análisis que la correlación indirecta es alta, pero la parcial que considera solo influencia directa de cada período es absoluta en tan solo un mes antes del momento a analizar: esto habla de la alta volatilidad del caso a analizar, ya que se deduce de la herramienta y los datos que es de poca utilidad para la predicción de valores en las criptomonedas la utilización de información de más de un mes atrás

Se crea una función para plotear una serie con información sobre los ACF y PACF y su estacionalidad:

[79]:
```
def tsplot(y, lags=None, figsize=(12, 7), style='bmh'):
    """
        Plotea la serie de tiempo, el ACF y PACF y el test de Dickey-Fuller

        y - serie de tiempo
        lags - cuántos lags incluir para el cálculo de la ACF y PACF

    """
    if not isinstance(y, pd.Series):
        y = pd.Series(y)

    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        layout = (2, 2)
```

```
    # Se definen ejes
    ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
    acf_ax = plt.subplot2grid(layout, (1, 0))
    pacf_ax = plt.subplot2grid(layout, (1, 1))

    y.plot(ax=ts_ax)

    # Se obtiene el p-value con H0: raiz unitaria presente
    p_value = sm.tsa.stattools.adfuller(y)[1]

    ts_ax.set_title('Análisis de la Serie de Tiempo\n Dickey-Fuller: p={0:.
↪5f}'\
                    .format(p_value))

    # Plot de autocorrelacion
    smt.graphics.plot_acf(y, lags=lags, ax=acf_ax)
    # Plot de autocorrelacion parcial
    smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax)
    plt.tight_layout()
```

[80]:
```
# Aplicación de la función con la serie res_log:
tsplot(res_log_est, lags=36)
```



2.8 h) ARIMA

Se aplica el auto_arima sobre res_log_est

[81]:
```
stepwise_fit = auto_arima(res_log_est, trace=True, suppress_warnings=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-9444.242, Time=1.04 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-9452.140, Time=0.38 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-9450.140, Time=0.55 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-9450.140, Time=1.20 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-9454.135, Time=0.23 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-9448.140, Time=0.55 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 3.955 seconds
```

Como se obtuvo mejores resultados con p=2 y q=1, s aplica:

[82]:
```
model_ARIMA = ARIMA(res_log_est, order=(2,0,1))

# Estimación del modelo:
results_ARIMA = model_ARIMA.fit()
results_ARIMA.fittedvalues.head()
```

[82]:
```
Date
2013-04-29    0.037648
2013-04-30    0.221391
2013-05-01    0.181000
2013-05-02   -0.091703
2013-05-03   -0.202132
Freq: D, dtype: float64
```

Se observa el summary:

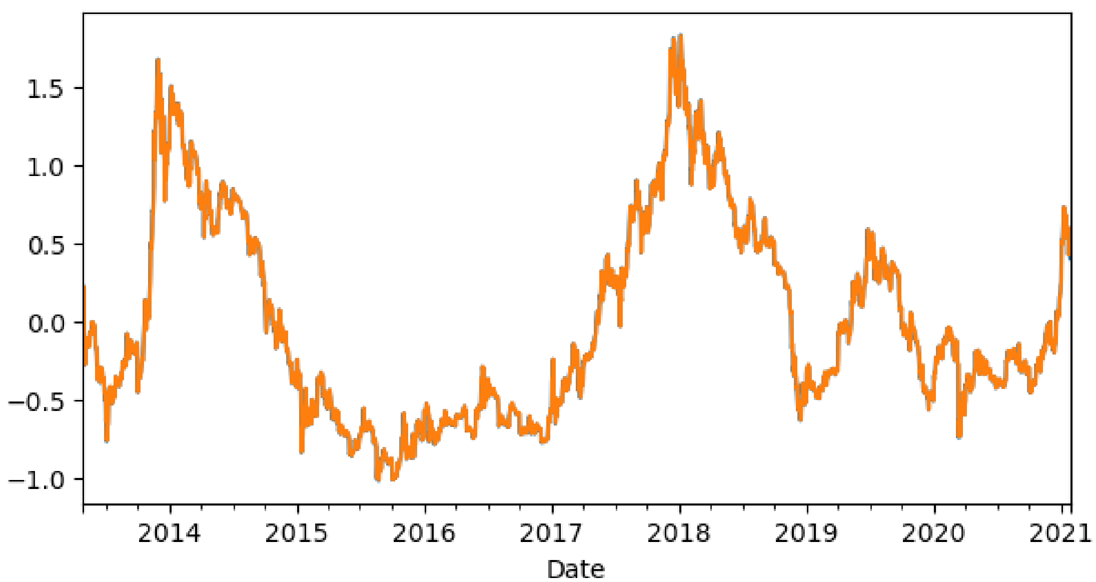
[83]:
```
print(results_ARIMA.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 2831
Model:                 ARIMA(2, 0, 1)   Log Likelihood                4729.821
Date:                Tue, 08 Aug 2023   AIC                          -9449.641
Time:                        21:29:45   BIC                          -9419.899
Sample:                    04-29-2013   HQIC                         -9438.911
                         - 01-27-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0376      0.308      0.122      0.903      -0.567       0.642
ar.L1          0.0797      0.115      0.694      0.488      -0.145       0.305
```

```
ar.L2          0.9147      0.114      7.997      0.000      0.691      1.139
ma.L1          0.9094      0.119      7.652      0.000      0.677      1.142
sigma2         0.0021   2.26e-05     91.558      0.000      0.002      0.002
================================================================================
===
Ljung-Box (L1) (Q):                   0.30   Jarque-Bera (JB):
12186.52
Prob(Q):                              0.59   Prob(JB):
0.00
Heteroskedasticity (H):               0.65   Skew:
-0.45
Prob(H) (two-sided):                  0.00   Kurtosis:
13.12
================================================================================
===

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

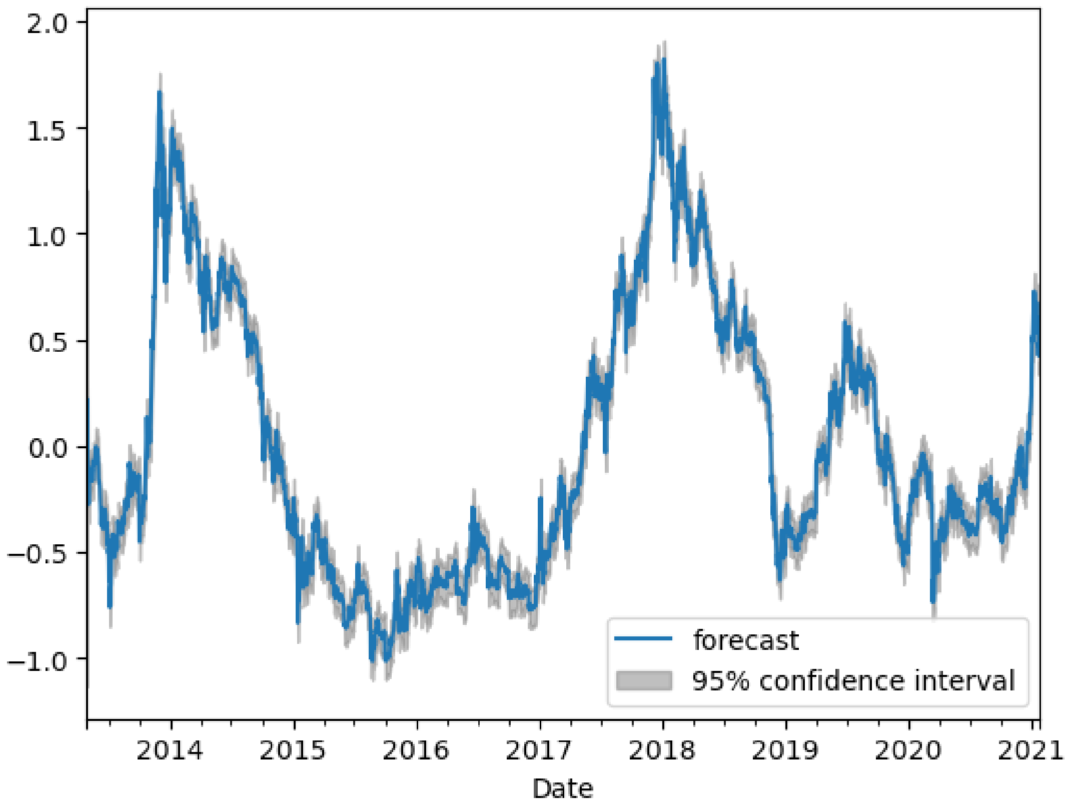
Ploteo de resultados:

[84]:
```
plt.figure(figsize=(7,3.5))
res_log_est.plot()
results_ARIMA.fittedvalues.plot();
```

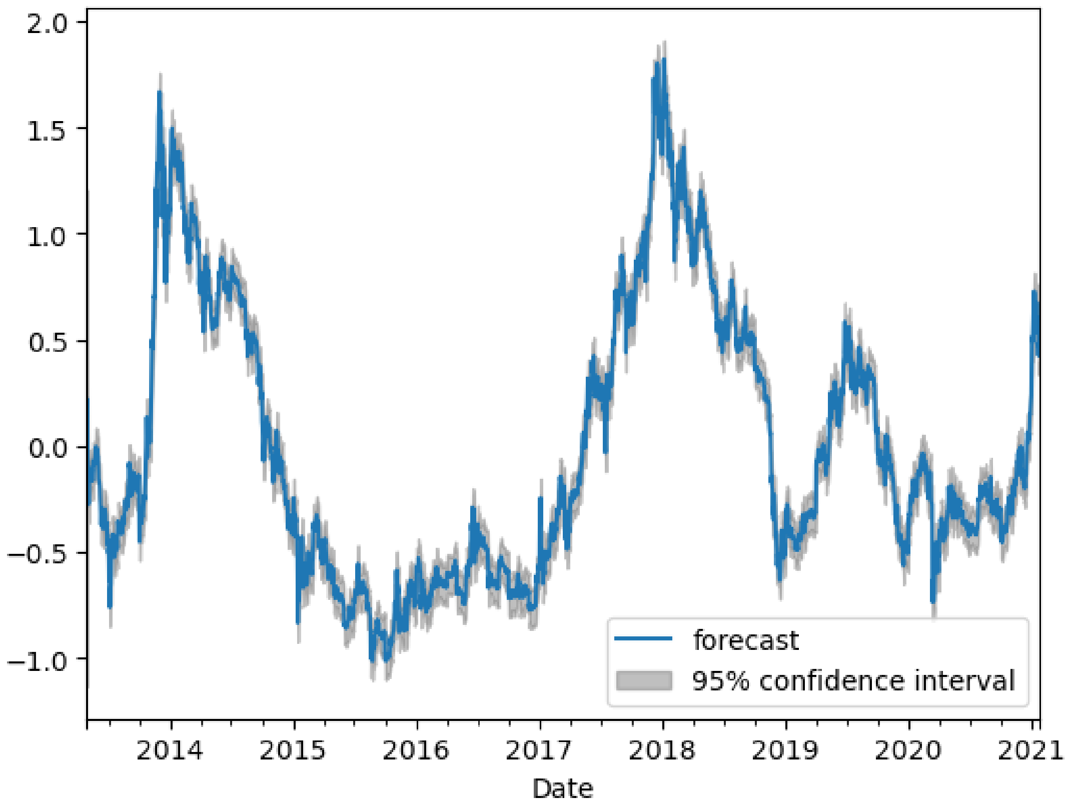


[85]:
```
plot_predict(results_ARIMA)
```

[85]:



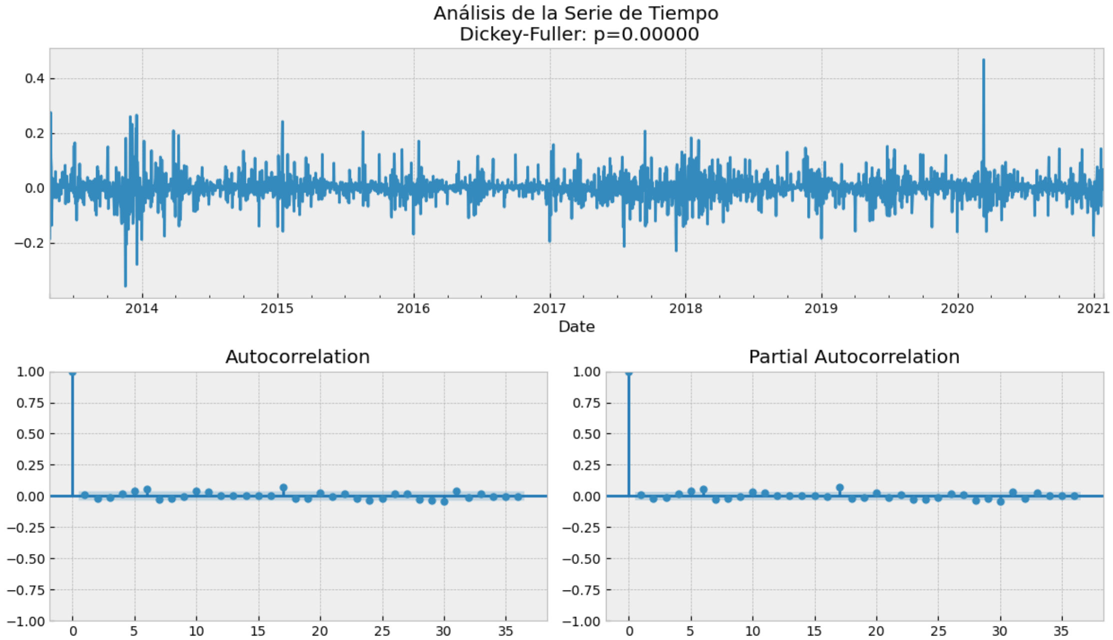
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
forecast
95% confidence interval
2014
2015
2016
2017
2018
2019
2020
2021
Date



Análisis de los residuos del modelo ARIMA:

[86]:
```
res_ARIMA =  results_ARIMA.fittedvalues - res_log_est
```

[87]:
```
tsplot(res_ARIMA, lags=36)
```



Aplicación del método Forecast:

[88]:
```
forecast = results_ARIMA.forecast(len(df_test['Close']), alpha=0.05)
print(forecast)
```

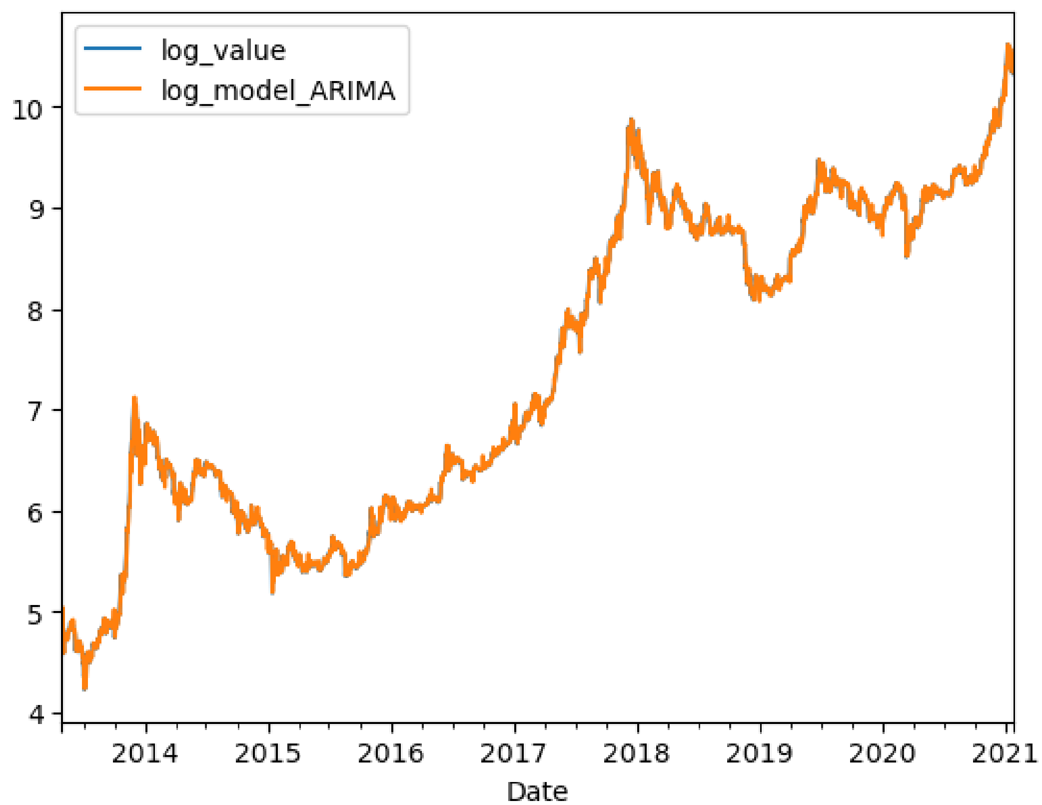
```
2021-01-28    0.401940
2021-01-29    0.399350
2021-01-30    0.399719
2021-01-31    0.397379
2021-02-01    0.397529
2021-02-02    0.395401
2021-02-03    0.395369
2021-02-04    0.393420
2021-02-05    0.393235
2021-02-06    0.391437
2021-02-07    0.391125
2021-02-08    0.389455
2021-02-09    0.389037
2021-02-10    0.387476
2021-02-11    0.386969
2021-02-12    0.385501
2021-02-13    0.384919
2021-02-14    0.383530
2021-02-15    0.382888
2021-02-16    0.381566
2021-02-17    0.380873
2021-02-18    0.379608
```

```
2021-02-19    0.378874
2021-02-20    0.377658
2021-02-21    0.376889
2021-02-22    0.375716
2021-02-23    0.374919
2021-02-24    0.373783
2021-02-25    0.372963
2021-02-26    0.371858
2021-02-27    0.371021
Freq: D, Name: predicted_mean, dtype: float64
```

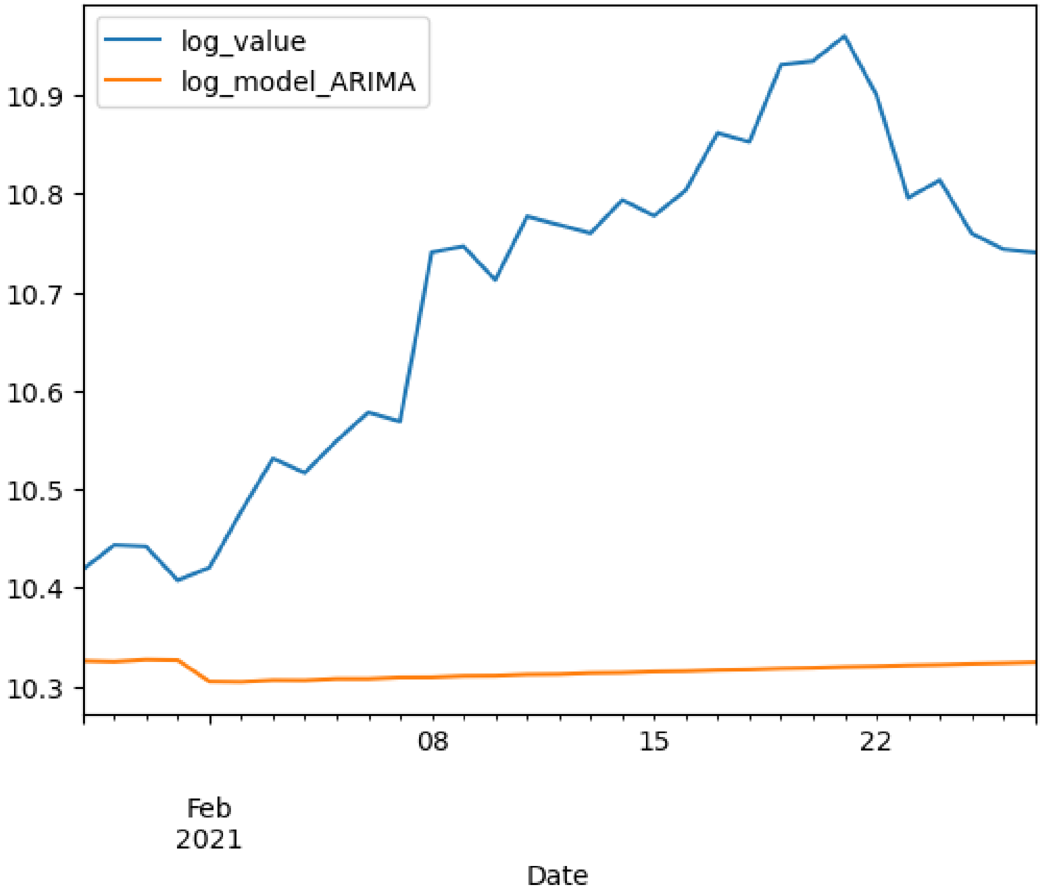
[89]:
```
# Se crea una variable para el train y otra para el test:

df_train['log_model_ARIMA'] = df_train['model_log_est'] + results_ARIMA.
 ↪fittedvalues
df_test['log_model_ARIMA'] = df_test['model_log_est'] + forecast
```

Ploteo de las predicciones vs las series reales, en train y test:

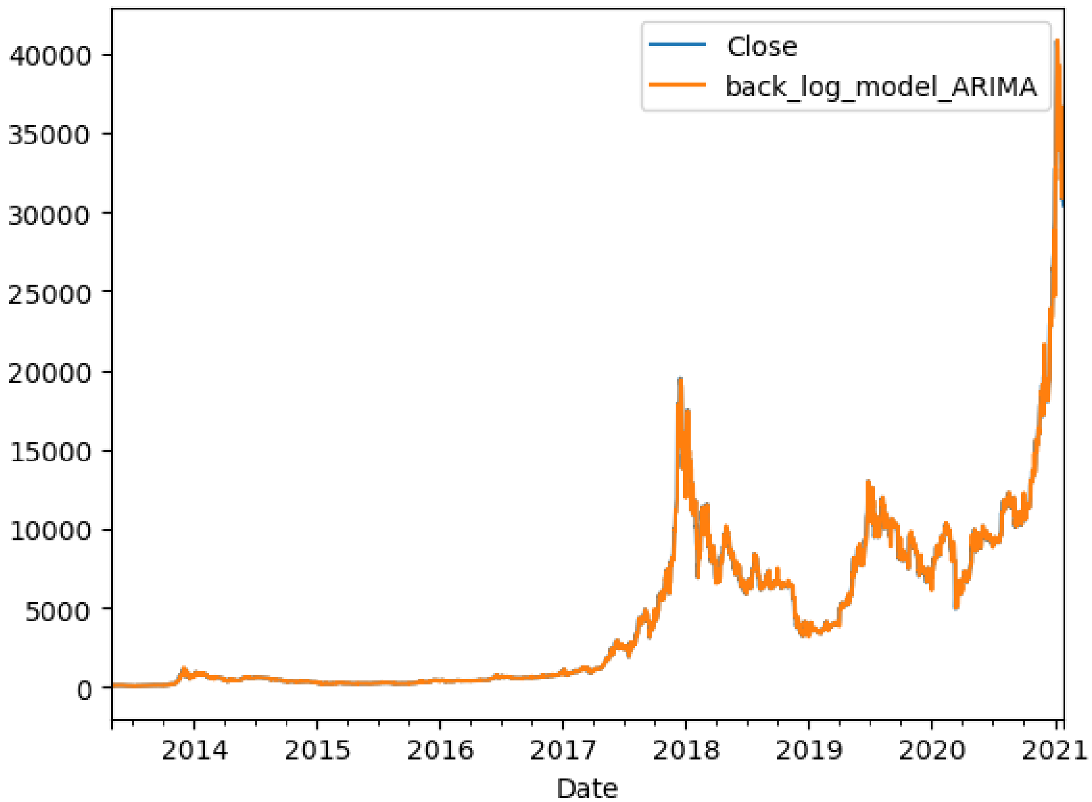
[90]:
```
df_train.plot(kind = "line", y = ['log_value', 'log_model_ARIMA']);
```



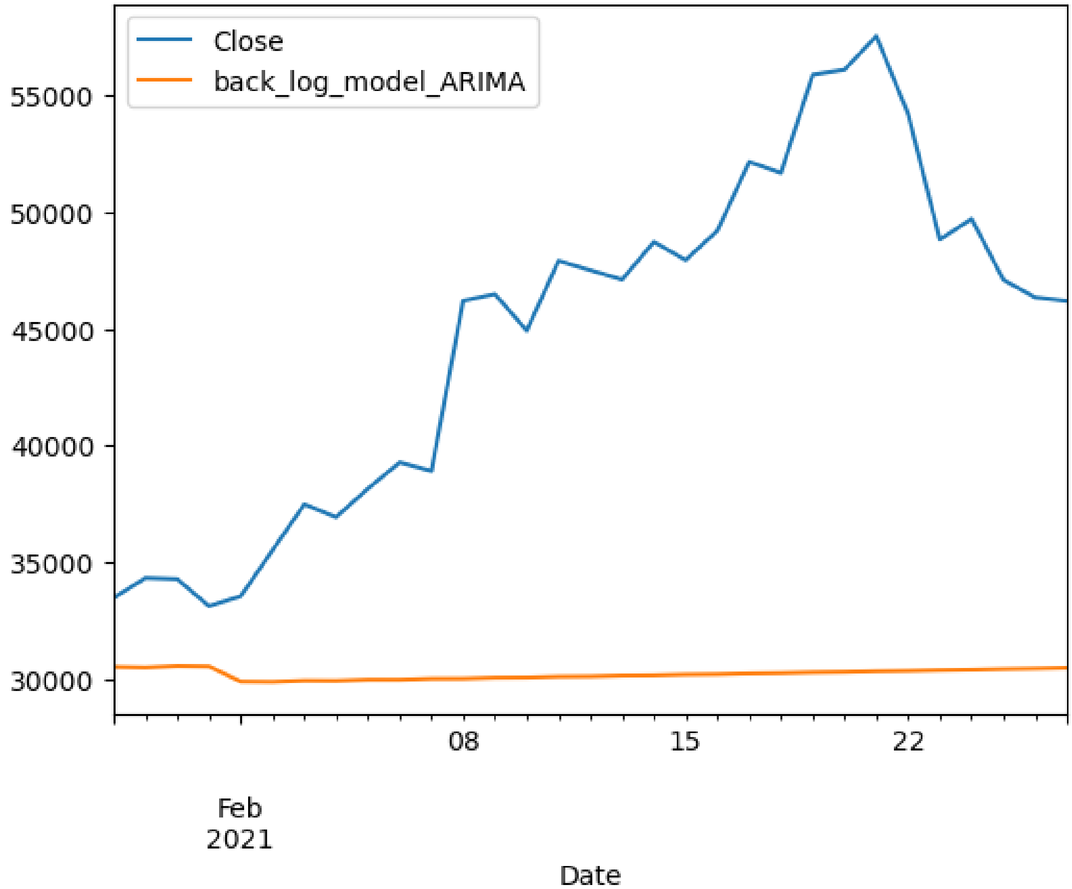
[91]:
```
df_test.plot(kind = "line", y = ['log_value', 'log_model_ARIMA']);
```



[92]:
```
# Se repite el proceso con back transformation del modelo log:

df_train['back_log_model_ARIMA'] = np.exp(df_train['log_model_ARIMA'])
df_test['back_log_model_ARIMA'] = np.exp(df_test['log_model_ARIMA'])

df_train.plot(kind = "line", y = ['Close', 'back_log_model_ARIMA']);
df_test.plot(kind = "line", y = ['Close', 'back_log_model_ARIMA']);
```



Close
back_log_model_ARIMA
40000
35000
30000
25000
20000
15000
10000
5000
0
2014
2015
2016
2017
2018
2019
2020
2021
Date



Se calcula el MAPE + RMSE y se almacena

[93]:
```
model_MAPE = mean_absolute_percentage_error (df_test.Close , df_test.
    ↪back_log_model_ARIMA)
```

[94]:
```
df_Results.loc[6, "Model"] = "Log Model + est + ARIMA"
df_Results.loc[6, "RMSE"] = RMSE(df_test['back_log_model_ARIMA'],␣
    ↪df_test['Close'])
df_Results.loc[6, "MAPE"] = model_MAPE
```

2.9 I) Prophet

En la 3ra y última notebook se prueba una herramienta no vista en el curso. Se separó para ser planteada como un anexo, debido a nuestra incertidumbre con respecto a sus resultados.

[95]:
```
# Como en esta notebook no se analiza esa herramienta, los resultados son␣
    ↪ingresados manualmente

df_Results.loc[7, "Model"] = "Prophet"
```

```python
df_Results.loc[7, "RMSE"] = 6997.16
df_Results.loc[7, "MAPE"] = 15.54

df_Results
```

[95]:

	Model	RMSE	MAPE
0	Mean	40968.433969	89.845161
1	Random Walk	16049.403917	30.009555
2	LinearTrend	33666.922165	72.755041
3	Transf Log	25190.683145	52.878589
4	Transf Log + est	25219.962947	52.844384
5	Simple Smoothing	13517.463589	23.350291
6	Log Model + est + ARIMA	16242.023902	30.550652
7	Prophet	6997.16	15.54

3 3) Comparación de resultados

Análisis de RMSE y MAPE visualizado

[96]:
```python
df_Results
```

[96]:

	Model	RMSE	MAPE
0	Mean	40968.433969	89.845161
1	Random Walk	16049.403917	30.009555
2	LinearTrend	33666.922165	72.755041
3	Transf Log	25190.683145	52.878589
4	Transf Log + est	25219.962947	52.844384
5	Simple Smoothing	13517.463589	23.350291
6	Log Model + est + ARIMA	16242.023902	30.550652
7	Prophet	6997.16	15.54

[97]:
```python
fig, ax1 = plt.subplots(figsize=(22, 13))
ax1.set_xlabel('Modelos', fontsize=22)
ax1.set_ylabel('MAPE', fontsize=20, color='b')
ax1.bar(df_Results.index - 0.2, df_Results.MAPE, width=0.4, color='b',␣
 ↪linewidth=2, label = "MAPE")
ax1.tick_params(axis='y', labelcolor='b', labelsize = 17)
ax1.tick_params(axis='x', labelsize = 15)
ax1.set_ylim([0, 99])

ax2 = ax1.twinx()
ax2.set_ylabel('RMSE', fontsize=20, color='r')
ax2.bar(df_Results.index + 0.2, df_Results.RMSE, width=0.4, color='r',␣
 ↪linewidth=2, label = "RMSE")
ax2.tick_params(axis='y', labelcolor='r', labelsize = 17)
ax2.set_ylim([0, 50000])

plt.axvline(x='Means', color="grey", linestyle="--", lw=1.3)
```

```
plt.axvline(x='Random Walk',color="grey", linestyle="--", lw=1.3)
plt.axvline(x='LinearTrend', color="grey", linestyle="--", lw=1.3)
plt.axvline(x='Transf Log' , color="grey", linestyle="--", lw=1.3)
plt.axvline(x='Transf Log + est', color="grey", linestyle="--", lw=1.3)
plt.axvline(x='S. Smoothing', color="grey", linestyle="--", lw=1.3)
plt.axvline(x='Log_est_ARIMA', color="grey", linestyle="--", lw=1.3)
plt.axvline(x='Prophet', color="grey", linestyle="--", lw=1.3)

plt.grid(which='major', axis='y', color='black', lw=0.4, alpha=0.6)
plt.suptitle("Comparación de resultados", fontsize=24, y=0.94)
legend1 = ax1.legend(loc=(0.86, 0.9), fontsize = 18)
legend2 = ax2.legend(loc=(0.86, 0.82), fontsize = 18)

plt.show()
```