Minería de datos G.571

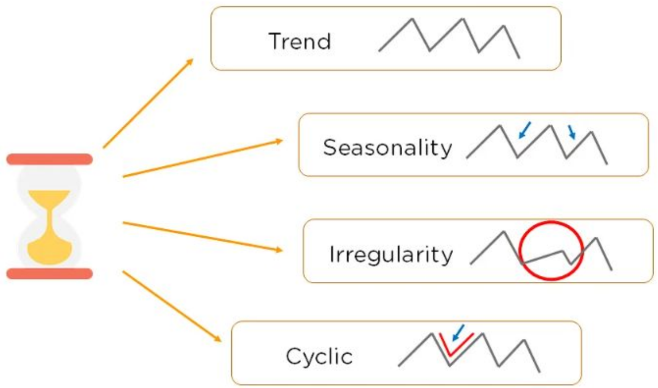
Pronóstico: ARFIMA

Minería de datos

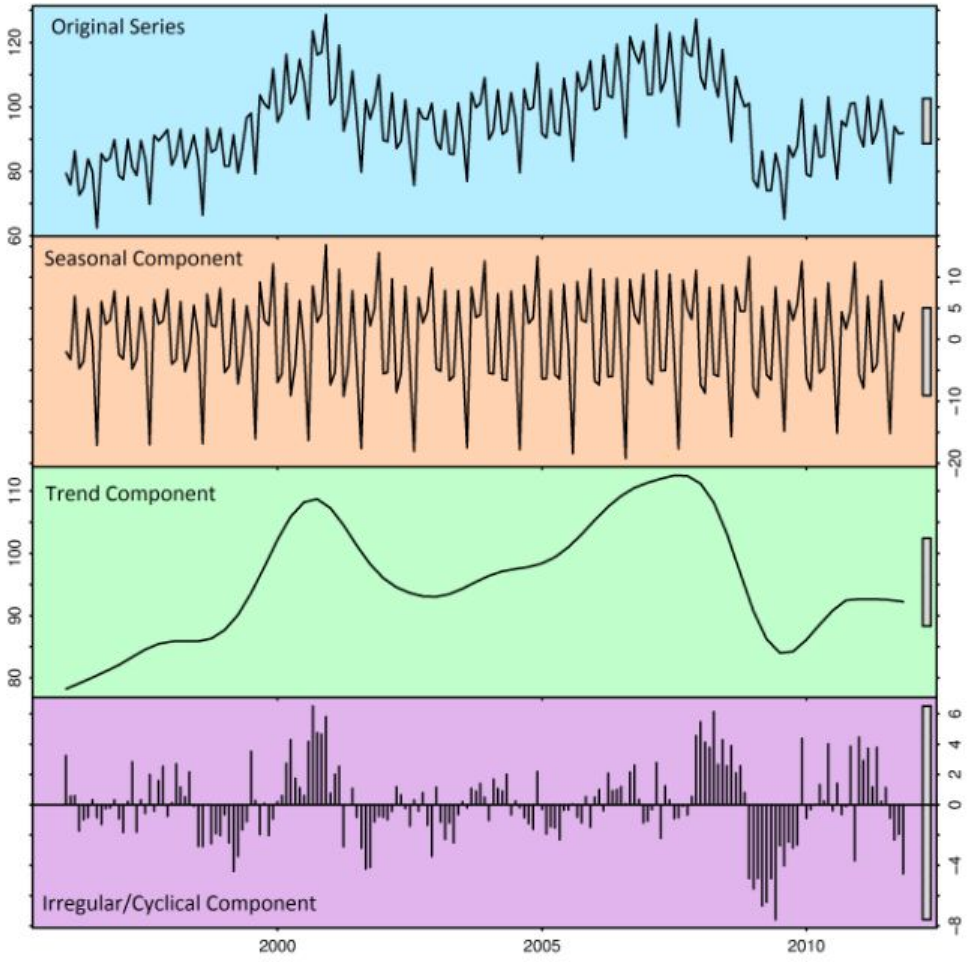
Zavala Roman Irvin Eduardo 1270771

Series de tiempo

- **Tendencia**: Movimiento vertical de los datos en tiempos largos
- **Estacionalidad**: Indican comportamientos que suceden en lapsos regulares
- **Irregularidad**: Comportamiento que no pertenece a los 2 anteriores
- **Cíclico**: Oscilaciones que duran más de un año, puede no ser periodico
- **Estacionario**: Tiene las mismas propiedades estadísticas en todo el tiempo (promedio, varianza y covarianza constantes)



Series de tiempo



Autoregressive Model (AR)

Dice que los valores anteriores afectan los valores actuales.

$$\hat{y}_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t$$

$$y_t = a_0 + \sum_{n=1}^{p} a_n y_{t-n} + \epsilon_t$$

Moving Average Model (MA)

Dice que el valor de la variable dependiente actual depende de los términos de error de los días anteriores.

$$y_t = \varepsilon_t + \alpha_1 \varepsilon_{t-1} + \alpha_2 \varepsilon_{t-2} + \cdots + \alpha_q \varepsilon_{t-q}$$

$$r_t = b_0 + \sum_{n=1}^{q} b_n r_{t-n} + \epsilon_t$$

$$r_t = \hat{y}_t - y_t$$

Auto Regressive Moving Average (ARMA)

Combina los modelos AR y MA.

$$y_t = \beta_1 y_{t-1} + \beta_2 y_{t-2} + \beta_3 y_{t-3} + \cdots + \beta_p y_{t-p} + \varepsilon_t + \alpha_1 \varepsilon_{t-1} + \alpha_2 \varepsilon_{t-2} + \cdots + \alpha_q \varepsilon_{t-q}$$

Integrated (I)

Fuerza la estacionalidad de la serie de tiempo por medio de un proceso de “diferenciación”.

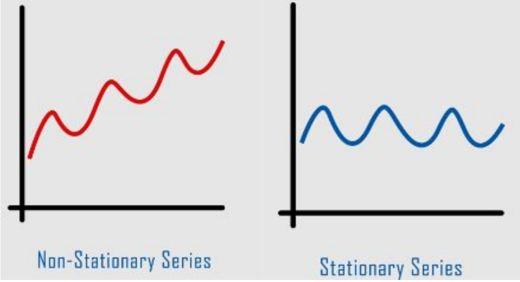
$By_t = y_{t-1}$ where B is called a backshift operator

Thus, a first order difference is written as

$$y'_t = y_t - y_{t-1} = (1 - B)y_t$$

In general, a d th-order difference can be written as

$$y'_t = (1 - B)^d y_t$$




```
dataset = pd.read_csv("MG_dataset.csv", on_bad_lines='skip')
dataset = dataset.shift(1)
dataset['1.2'][0] = 1.2
dataset = dataset.rename(columns={"1.2": "data"})
print(dataset)
print(dataset.diff())
```

```
              data
0     1.2000000000
1     1.0858000000
2     0.9824800000
3     0.8889800000
4     0.8043800000
...            ...
1195 0.9211900000
1196 0.9514500000
1197 0.9716200000
1198 0.9828300000
1199 0.9864000000

[1200 rows x 1 columns]
                data
0                NaN
1      -0.1142000000
2      -0.1033200000
3      -0.0935000000
4      -0.0846000000
...              ...
1195  0.0411800000
1196  0.0302600000
1197  0.0201700000
1198  0.0112100000
1199  0.0035700000

[1200 rows x 1 columns]
```

AutoRegressive Integrated Moving Average (ARIMA)

Junta todo lo anterior, generalmente tiene 3 parámetros:

p: Número de términos autorregresivos (AR)

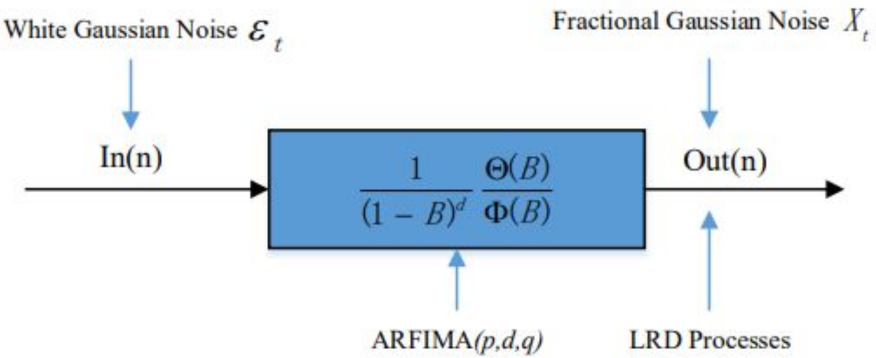
d: Número de diferencias no estacionarias (I)

q: Número de términos de moving-average (MA)

AutoRegressive Fractional Integrated Moving Average (ARFIMA)

Agrega diferenciación fraccionaria a para ser utilizado en series de tiempo donde se necesita predecir a largo plazo.

$$\Phi(B)(1-B)^d X_t = \Theta(B)\varepsilon_t,$$



ARFIMA en Python

En Python no existe la implementación de ARFIMA con ninguna librería, pero lo que se puede hacer es aplicar la parte de "Fractional" de AR***F***IMA antes y teniendo el dataset diferenciado mandarlo al ARIMA de sklearn con el parámetro de diferenciación en 0.

```
[60] """Computes the weights for our fractionally differenced features up to a given threshold
        requirement for fixed-window fractional differencing.
         Args:
             d: A float representing the differencing factor
             length: An int length of series to be differenced
             threshold: A float representing the minimum threshold to include weights for
         Returns:
             A numpy array containing the weights to be applied to our time series
     """
     def findWeights_FFD(d, length, threshold):
```

ARFIMA en Python

```
"""Computes fractionally differenced series
    Args:
        d: A float representing the differencing factor (any positive fractional)
        series: A pandas dataframe with one or more columns of time-series values to be differenced
        threshold: Threshold value past which we ignore weights
            (cutoff weight for window)
    Returns:
        diff_series: A numpy array of differenced series by d.
"""
def fracDiff(series, d, threshold = 1e-5):
```

Ejemplo de uso:

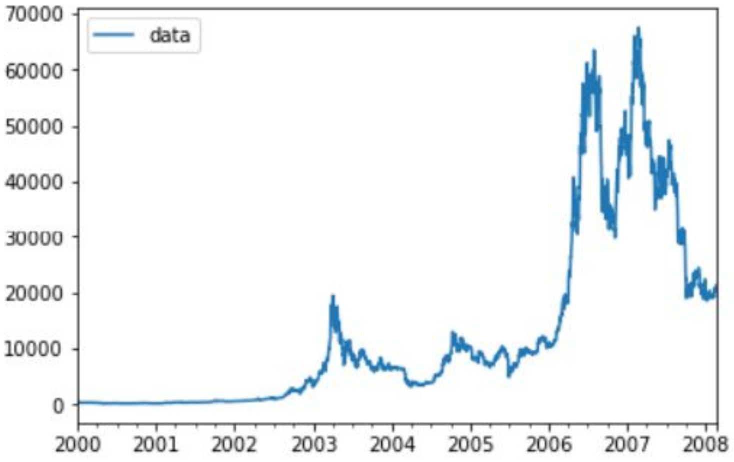
```
df_result = fracDiff(test_series, 3.9, 1e-5)
```

```
model = ARIMA(df_train, order=(2,0,1))
```

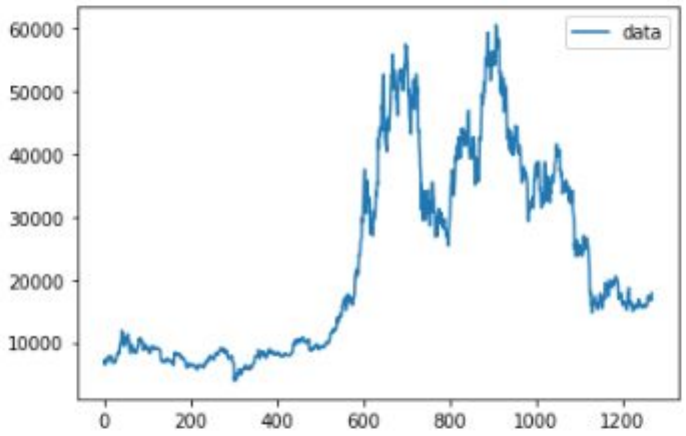
Créditos: hongwai1920/Using-ARIMA-model-to-forecast-returns

Probando ARFIMA

Original dataset Bitcoin Price

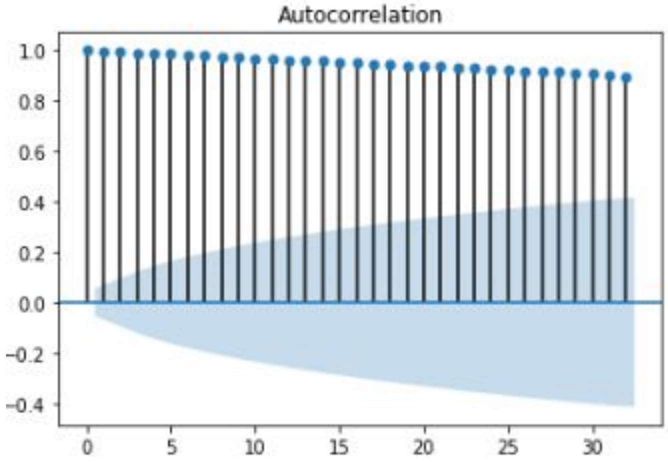


Después de diferenciación fraccionaria (0.02)

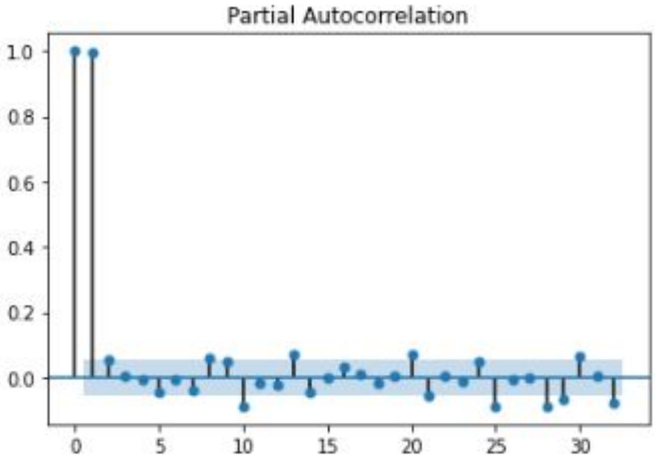


(Las fechas no corresponden al dataset)

ACF y PACF



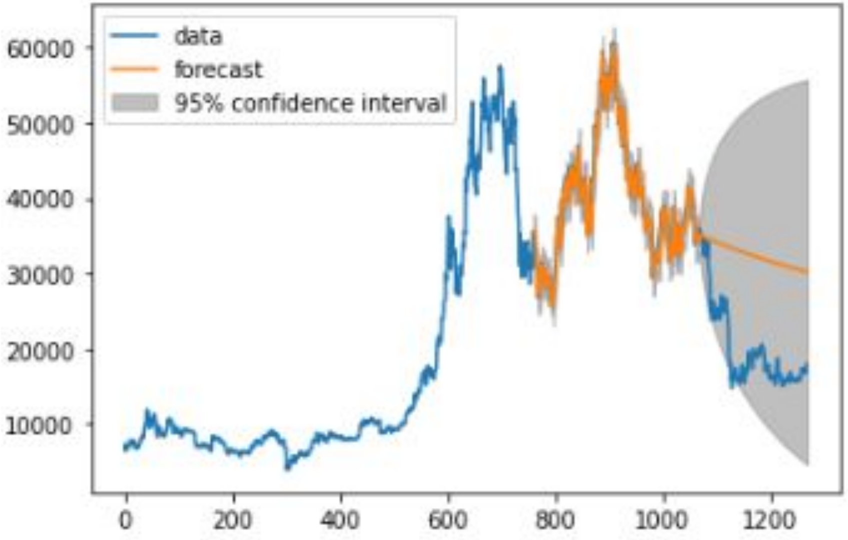
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
5
10
15
20
25
30



Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

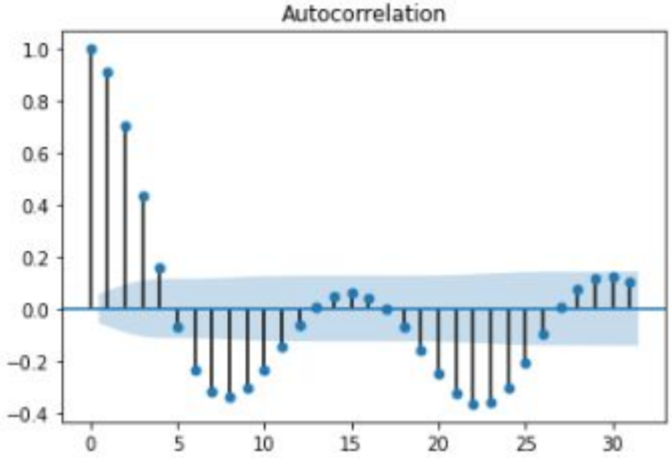
Predicción

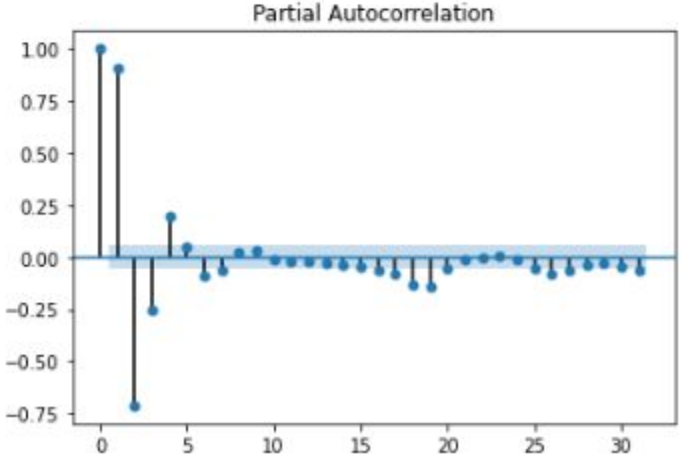
```python
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df_train, order=(1,0,1))
model_fit = model.fit()
print(model_fit.summary())
perc = 40
from statsmodels.graphics.tsaplots import plot_predict
fig, ax = plt.subplots()
s = pd.DataFrame(list(range(len(df))))
ax = df.loc[:].plot(ax=ax)

plot_predict(model_fit, len(df)-round(len(df) * perc/100), len(df), ax=ax)
plt.show()
```



Nota! En ARIMA al dar argumento en *d* se aplica la diferenciación y luego se deshace para aplicar el modelo al dataset original, en este caso no supe cómo revertir este método :(

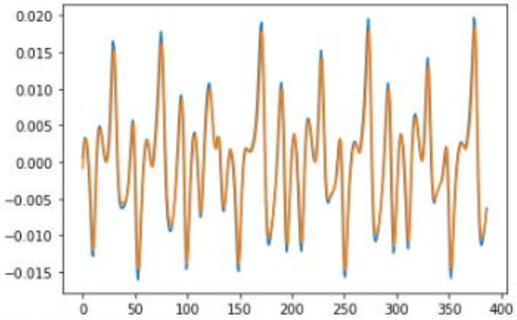
ARFIMA con MG dataset





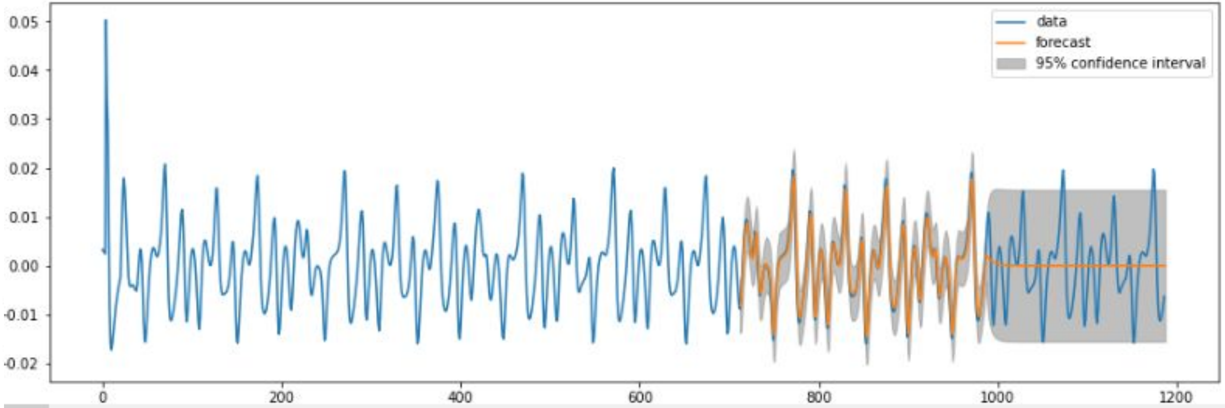

```
df_result = fracDiff(test_series, 1.99, 1e-5)
```

ARFIMA con MG dataset




```
anteriores = list(df['data'][:800])
y_true = []
y_pred = []
print(anteriores)
for nuevo in df['data'][800:]:
    model = ARIMA(anteriores, order=(1,0,1))
    model_fit = model.fit()
    forecast = model_fit.forecast(alpha=0.05)[0]

    y_true.append(nuevo)
    y_pred.append(forecast)
    anteriores.append(nuevo)
```

<matplotlib.legend.Legend at 0x7f4aeee2cbd0>




```
SSE 0.0015276978127738456
 MSE 3.9475395678910735e-06
 RMSE 0.001986841606140528
 MAE 0.0015451718698812444
 R2 0.9289024206687991
```

Pronóstico ventanas

TIEMPO	T1	T2	T3	T4	T5	T6	T7	T8	T9	...	TN
VENTANA 1	T-3		T-1	T+1							
VENTANA 2	T-5	T-4		T-2	T-1	T+1					
VENTANA 3	T-3		T-1	T+1	T+2						
VENTANA 4	T-5	T-4		T-2	T-1	T+1	T+2				

Bitcoin

TIEMPO	T1	T2	T3	T4	T5	T6	T7	T8	T9	...	TN
VENTANA 1	T-3	T-2		T+1							
VENTANA 2	T-5		T-3	T-2	T-1	T+1					
VENTANA 3	T-3	T-2		T+1	T+2						
VENTANA 4	T-5		T-3	T-2	T-1	T+1	T+2				

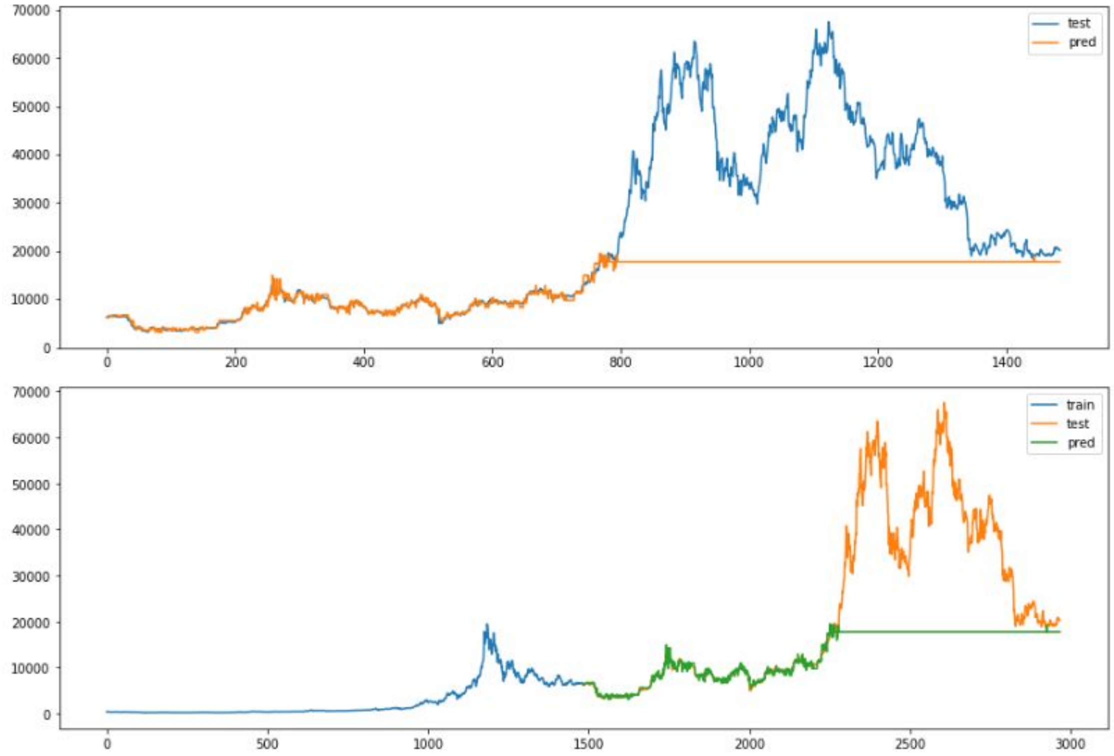
MG

Implementación ventanas

Solo devuelve el dataset creado dependiendo de la ventana

```
def predict_uno_2(dataset):
    ventana = 3
    predict_size = 1
    df = pd.DataFrame(dataset)
    cols = list()

    '''
    La tabla tiene (t-n) ... (t-1) | (t) ... (t+n)
    '''
    tabla = [list() for i in range(ventana-1+predict_size)]

    for i in range(ventana, len(df)-predict_size+1):
        tabla[0].append(df['data'][i-ventana])
        tabla[1].append(df['data'][i-1])
        tabla[2].append(df['data'][i])
    for array in tabla:
        cols.append(pd.Series(array))
    df = pd.concat(cols, axis=1)

    print(df)
    return df
```

Predicción Bitcoin: 2 entradas/1 salida 50% train

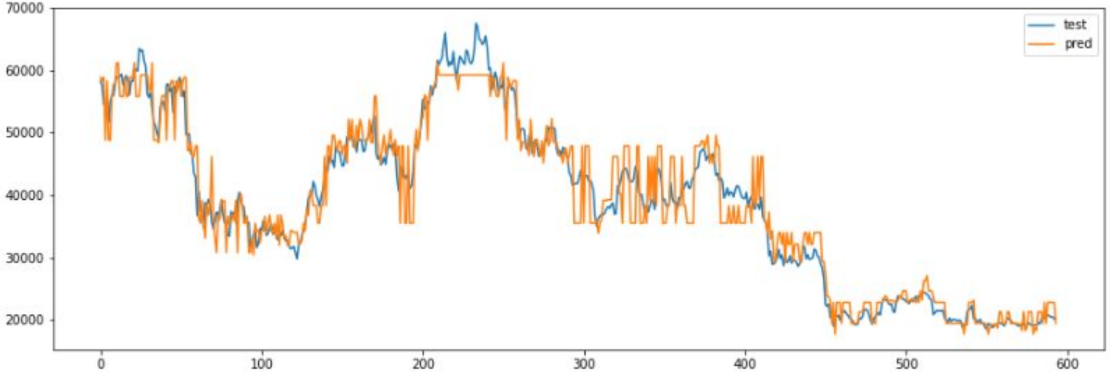


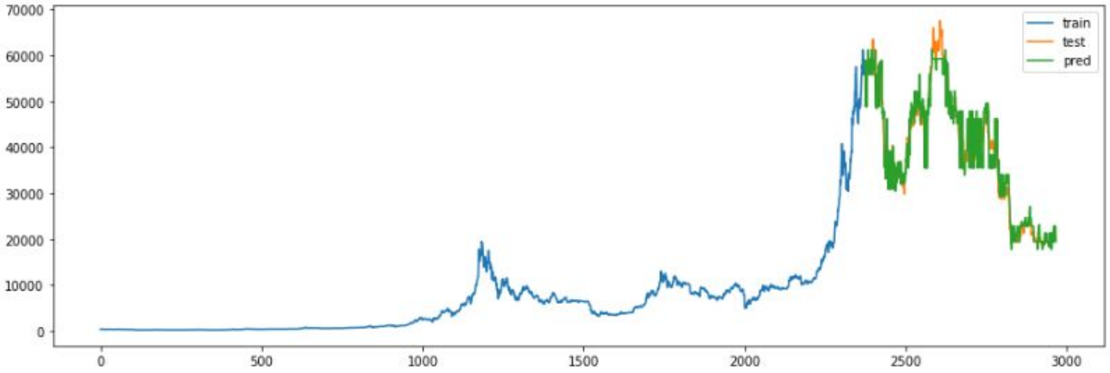

```
SSE [4.35365862e+11]
 MSE [2.93373223e+08]
 RMSE [17128.14124989]
 MAE [10208.27335948]
 R2 0.08315962115372233
```

	0	1	2
0	465.864014	424.102997	394.673004
1	456.859985	394.673004	408.084991
2	424.102997	408.084991	399.100006
3	394.673004	399.100006	402.092010
4	408.084991	402.092010	435.751007
...	...	...	...
2963	20287.957030	20817.982420	20633.695310
2964	20595.103520	20633.695310	20494.898440
2965	20817.982420	20494.898440	20482.958980
2966	20633.695310	20482.958980	20162.689450
2967	20494.898440	20162.689450	20208.769530

Mejor resultado para
bitcoin dataset!

Predicción Bitcoin: 2 entradas/1 salida 80% train

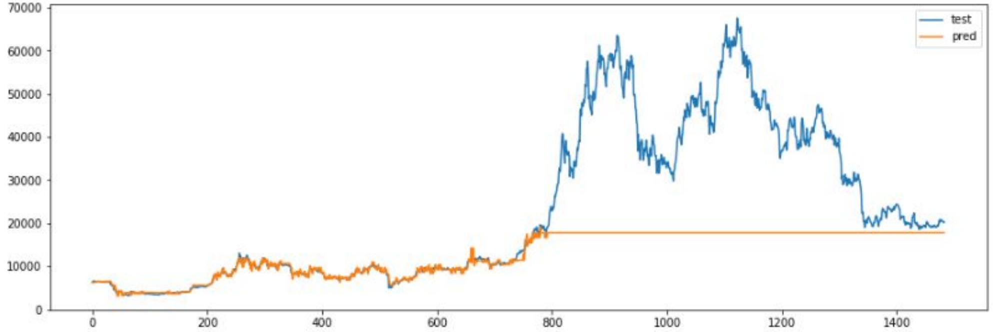


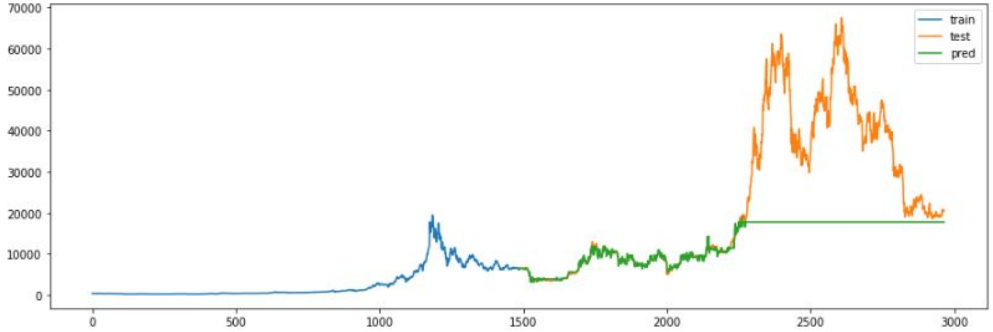



```
SSE [6.63582433e+09]
 MSE [11171421.43703903]
 RMSE [3342.36763942]
 MAE [2501.04320549]
 R2 0.937012889532168
```

	0	1	2
0	465.864014	424.102997	394.673004
1	456.859985	394.673004	408.084991
2	424.102997	408.084991	399.100006
3	394.673004	399.100006	402.092010
4	408.084991	402.092010	435.751007
...	...	...	...
2963	20287.957030	20817.982420	20633.695310
2964	20595.103520	20633.695310	20494.898440
2965	20817.982420	20494.898440	20482.958980
2966	20633.695310	20482.958980	20162.689450
2967	20494.898440	20162.689450	20208.769530

Predicción Bitcoin: 4 entradas/1 salida 50% train

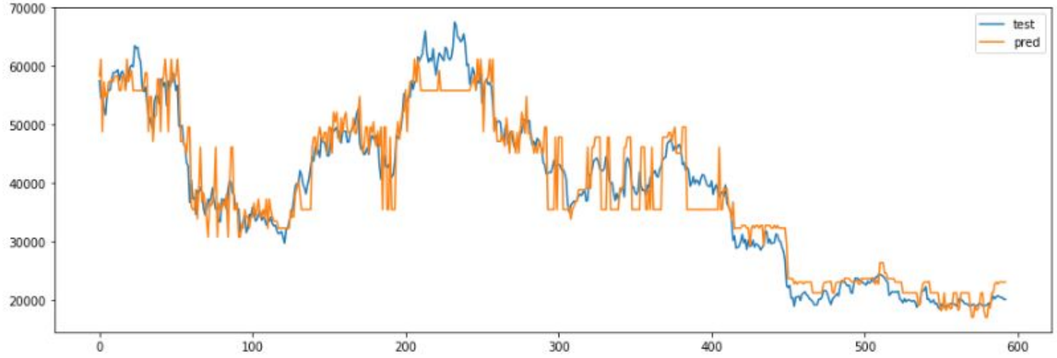


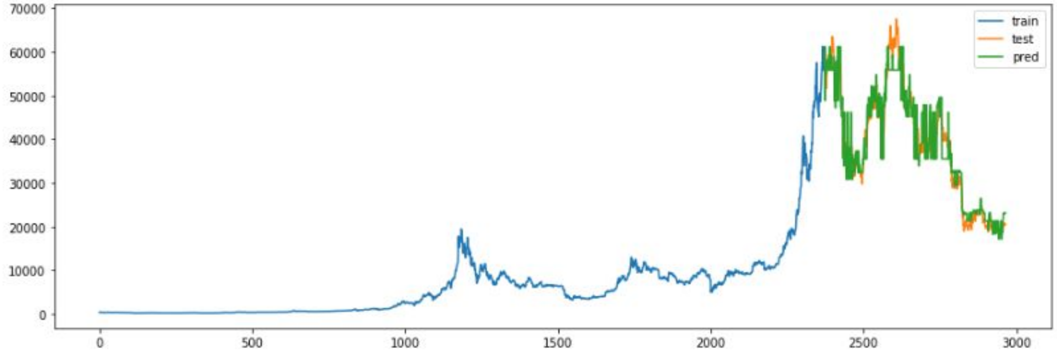



```
SSE [4.35365794e+11]
 MSE [2.93571001e+08]
 RMSE [17133.9137552]
 MAE [10219.60313943]
 R2 0.08263389822819689
```

	0	1	2	3	4
0	465.864014	456.859985	394.673004	408.084991	399.100006
1	456.859985	424.102997	408.084991	399.100006	402.092010
2	424.102997	394.673004	399.100006	402.092010	435.751007
3	394.673004	408.084991	402.092010	435.751007	423.156006
4	408.084991	399.100006	435.751007	423.156006	411.428986
...	...	...	...	...	...
2961	20092.236330	20772.802730	20595.103520	20817.982420	20633.695310
2962	20772.802730	20287.957030	20817.982420	20633.695310	20494.898440
2963	20287.957030	20595.103520	20633.695310	20494.898440	20482.958980
2964	20595.103520	20817.982420	20494.898440	20482.958980	20162.689450
2965	20817.982420	20633.695310	20482.958980	20162.689450	20208.769530

Predicción Bitcoin: 4 entradas/1 salida 80% train

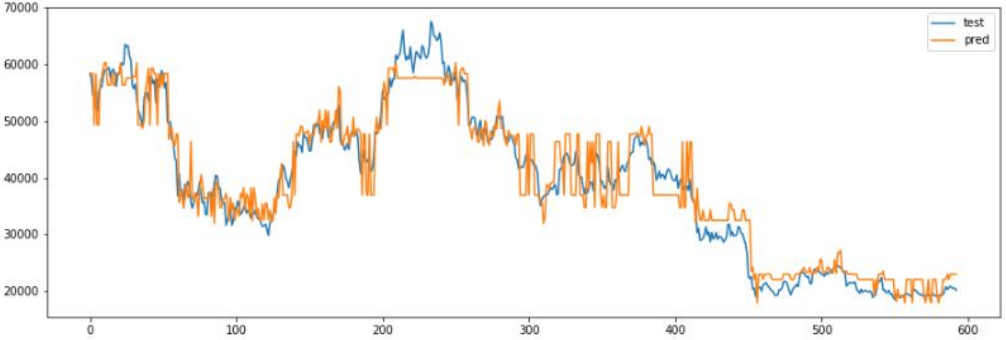


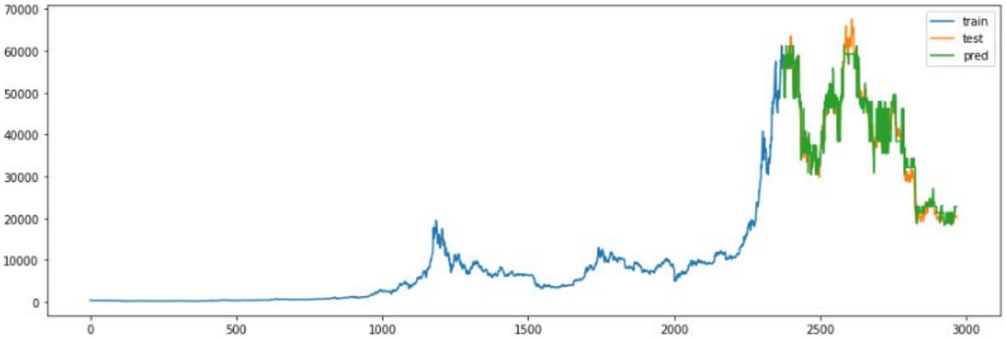


SSE [8.17872952e+09]
MSE [13792123.97263254]
RMSE [3713.77489526]
MAE [2862.30439012]
R2 0.9220936294015181

	0	1	2	3	4
0	465.864014	456.859985	394.673004	408.084991	399.100006
1	456.859985	424.102997	408.084991	399.100006	402.092010
2	424.102997	394.673004	399.100006	402.092010	435.751007
3	394.673004	408.084991	402.092010	435.751007	423.156006
4	408.084991	399.100006	435.751007	423.156006	411.428986
...	...	...	...	...	...
2961	20092.236330	20772.802730	20595.103520	20817.982420	20633.695310
2962	20772.802730	20287.957030	20817.982420	20633.695310	20494.898440
2963	20287.957030	20595.103520	20633.695310	20494.898440	20482.958980
2964	20595.103520	20817.982420	20494.898440	20482.958980	20162.689450
2965	20817.982420	20633.695310	20482.958980	20162.689450	20208.769530

Predicción Bitcoin: 2 entradas/2 salidas 80% dataset

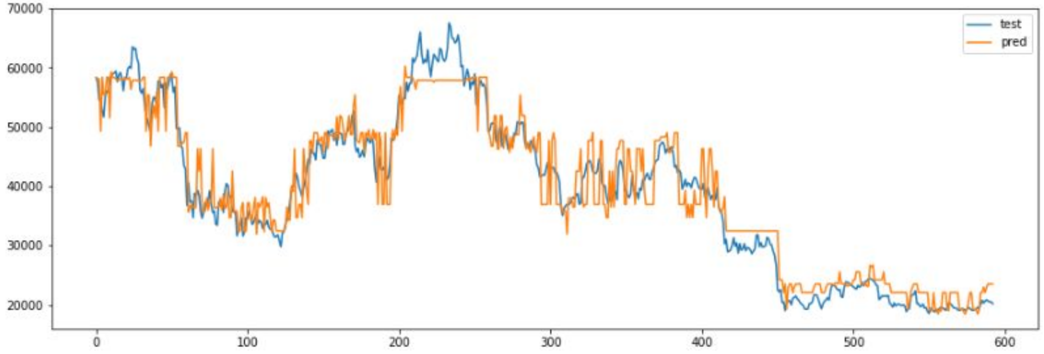


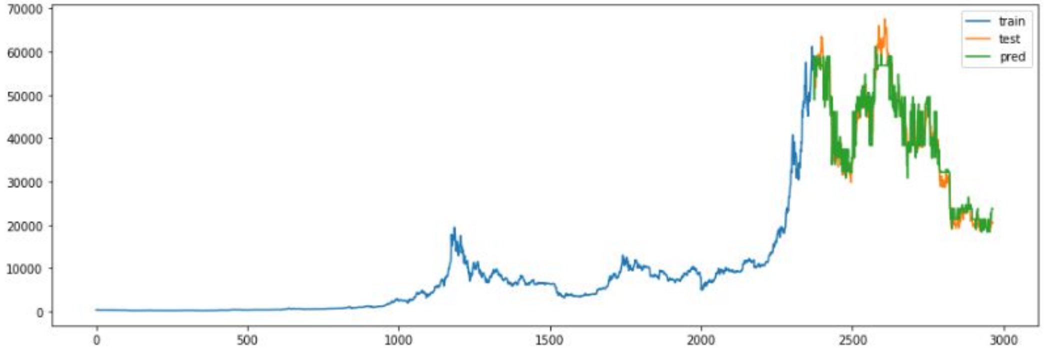


SSE 16823619228.14617
MSE 28370352.82992609
RMSE 5326.382715307462
MAE 5879.370046290042
R2 0.9198771904482907

	0	1	2	3
0	465.864014	424.102997	394.673004	408.084991
1	456.859985	394.673004	408.084991	399.100006
2	424.102997	408.084991	399.100006	402.092010
3	394.673004	399.100006	402.092010	435.751007
4	408.084991	402.092010	435.751007	423.156006
...	...	...	...	...
2962	20772.802730	20595.103520	20817.982420	20633.695310
2963	20287.957030	20817.982420	20633.695310	20494.898440
2964	20595.103520	20633.695310	20494.898440	20482.958980
2965	20817.982420	20494.898440	20482.958980	20162.689450
2966	20633.695310	20482.958980	20162.689450	20208.769530

Predicción Bitcoin: 4 entradas/2 salidas 80% train

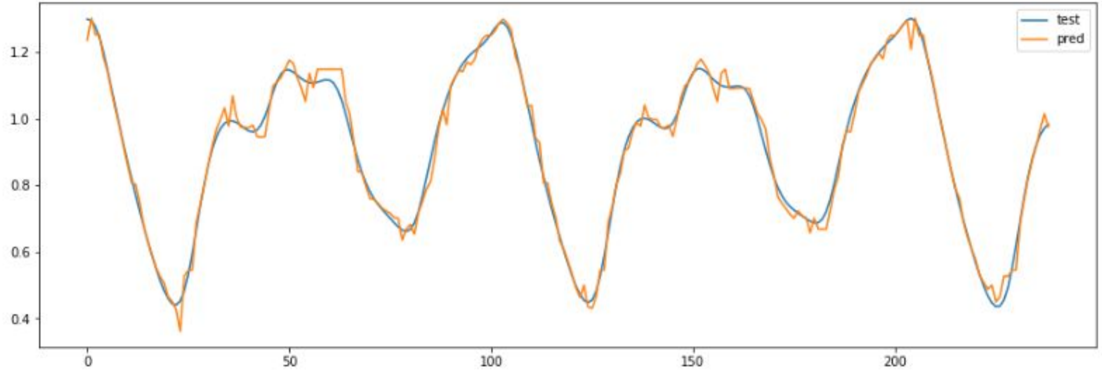


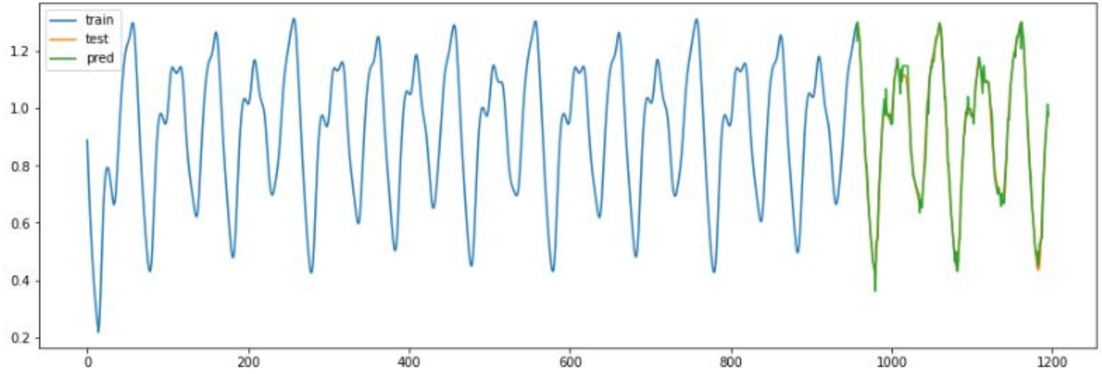



```
SSE 17019592511.656761
MSE 28700830.54242287
RMSE 5357.315609745507
MAE 5906.161006104553
R2 0.9189444614679174
```

```
                0            1            2            3            4  \
0      465.864014   456.859985   394.673004   408.084991   399.100006
1      456.859985   424.102997   408.084991   399.100006   402.092010
2      424.102997   394.673004   399.100006   402.092010   435.751007
3      394.673004   408.084991   402.092010   435.751007   423.156006
4      408.084991   399.100006   435.751007   423.156006   411.428986
...           ...          ...          ...          ...          ...
2960  19344.964840 20092.236330 20287.957030 20595.103520 20817.982420
2961  20092.236330 20772.802730 20595.103520 20817.982420 20633.695310
2962  20772.802730 20287.957030 20817.982420 20633.695310 20494.898440
2963  20287.957030 20595.103520 20633.695310 20494.898440 20482.958980
2964  20595.103520 20817.982420 20494.898440 20482.958980 20162.689450

                5
0      402.092010
1      435.751007
2      423.156006
3      411.428986
4      403.556000
...           ...
2960  20633.695310
2961  20494.898440
2962  20482.958980
2963  20162.689450
2964  20208.769530
```

Predicción MG: 2 entradas/1 salida



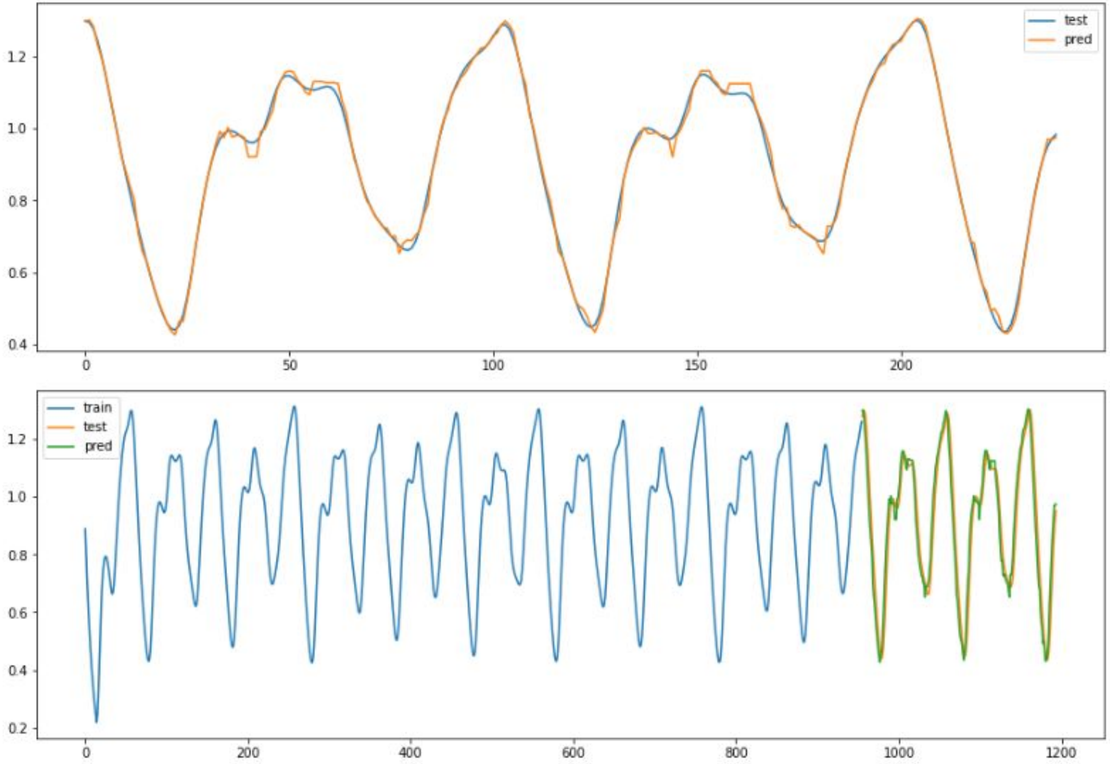



```
SSE [0.17424055]
 MSE [0.00072904]
 RMSE [0.02700074]
 MAE [0.01921778]
 R2 0.9872111979967401
```

	0	1	2
0	1.20000	1.08580	0.88898
1	1.08580	0.98248	0.80438
2	0.98248	0.88898	0.72784
3	0.88898	0.80438	0.65857
4	0.80438	0.72784	0.59590
...	...	...	...
1191	0.69324	0.76448	0.88001
1192	0.76448	0.82756	0.92119
1193	0.82756	0.88001	0.95145
1194	0.88001	0.92119	0.97162
1195	0.92119	0.95145	0.98283

Predicción MG: 4 entradas/1 salida

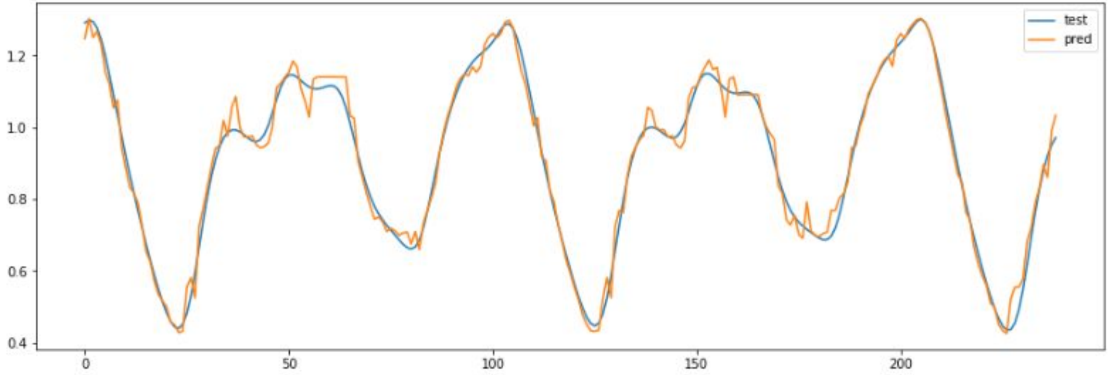
Mejor resultado para MG dataset!

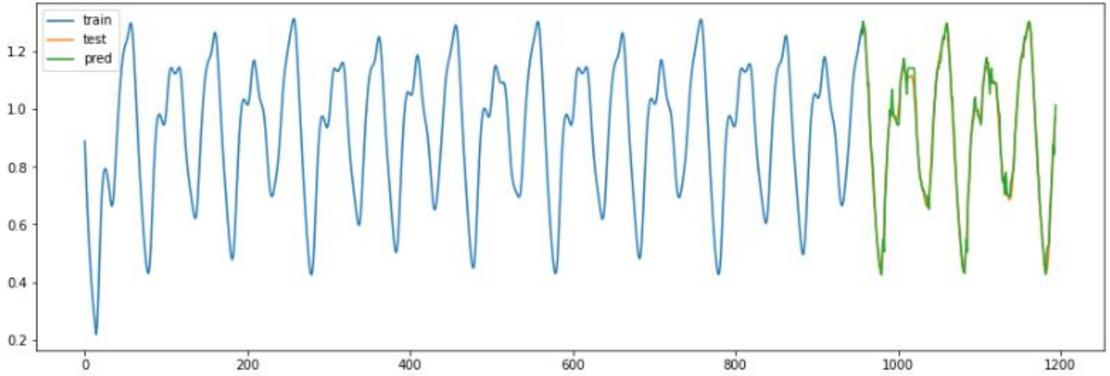



```
SSE [0.05452576]
 MSE [0.00022814]
 RMSE [0.01510435]
 MAE [0.01091619]
 R2 0.9959979516151715
```

	0	1	2	3	4
0	1.20000	0.98248	0.88898	0.80438	0.72784
1	1.08580	0.88898	0.80438	0.72784	0.65857
2	0.98248	0.80438	0.72784	0.65857	0.59590
3	0.88898	0.72784	0.65857	0.59590	0.53919
4	0.80438	0.65857	0.59590	0.53919	0.48788
...	...	...	...	...	...
1189	0.54945	0.69324	0.76448	0.82756	0.88001
1190	0.61897	0.76448	0.82756	0.88001	0.92119
1191	0.69324	0.82756	0.88001	0.92119	0.95145
1192	0.76448	0.88001	0.92119	0.95145	0.97162
1193	0.82756	0.92119	0.95145	0.97162	0.98283

Predicción MG: 2 entradas/2 salidas

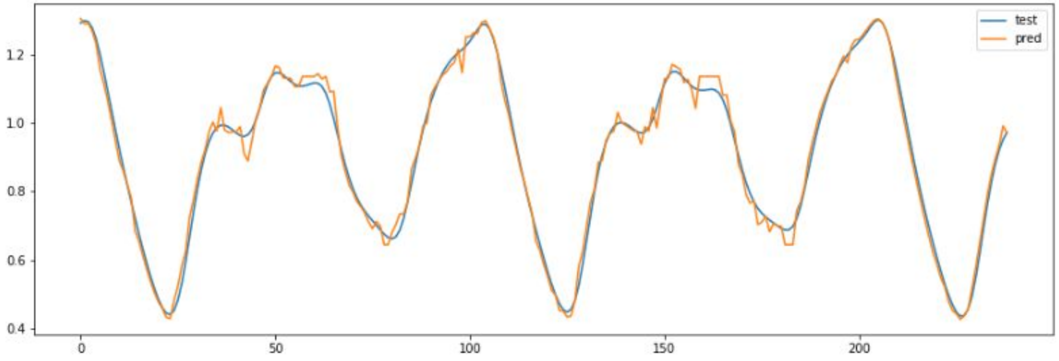


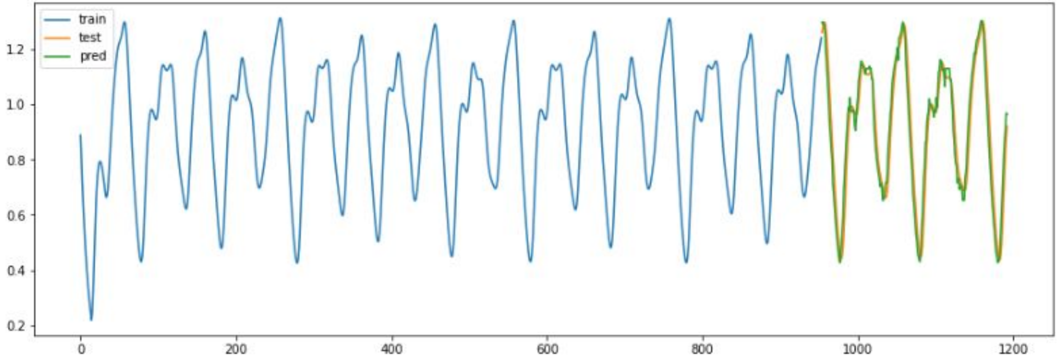



```
SSE 0.5242577047999997
 MSE 0.0021935468820083667
 RMSE 0.04683531661052764
 MAE 0.04692543933054397
 R2 0.9808206975185996
```

	0	1	2	3
0	1.20000	1.08580	0.88898	0.80438
1	1.08580	0.98248	0.80438	0.72784
2	0.98248	0.88898	0.72784	0.65857
3	0.88898	0.80438	0.65857	0.59590
4	0.80438	0.72784	0.59590	0.53919
...	...	...	...	...
1190	0.61897	0.69324	0.82756	0.88001
1191	0.69324	0.76448	0.88001	0.92119
1192	0.76448	0.82756	0.92119	0.95145
1193	0.82756	0.88001	0.95145	0.97162
1194	0.88001	0.92119	0.97162	0.98283

Predicción MG: 4 entradas/2 salidas





SSE 0.2717791786
MSE 0.0011371513748953976
RMSE 0.03372167514960367
MAE 0.0314919665271967
R2 0.9900516731271898

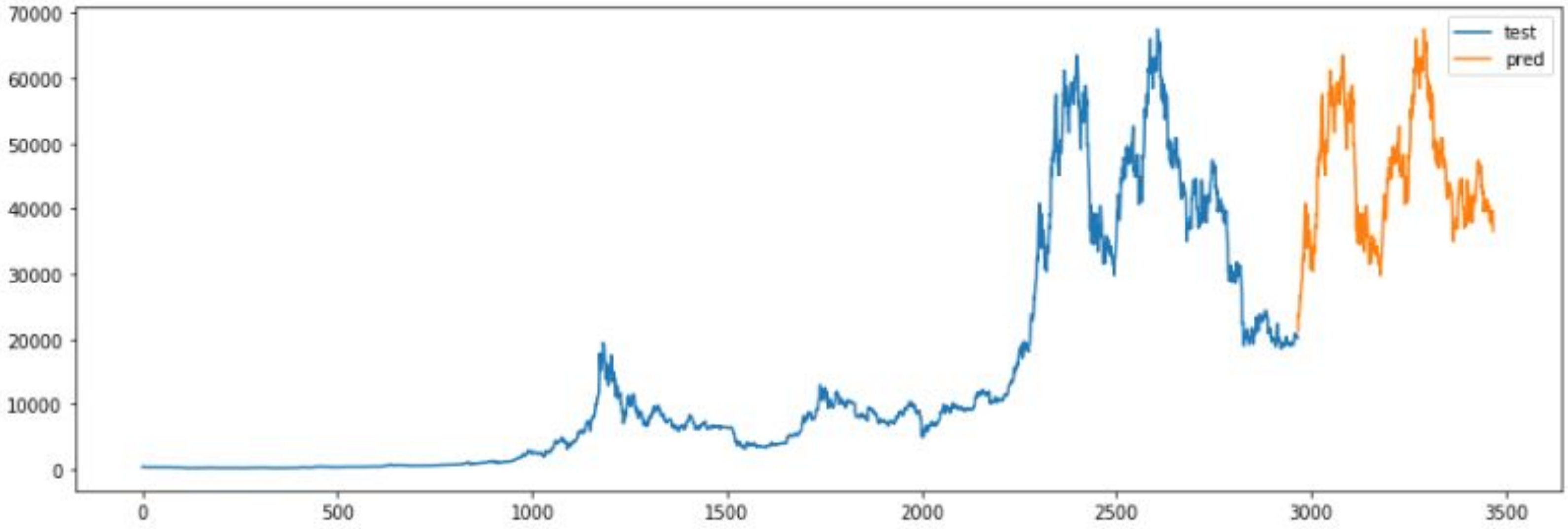
	0	1	2	3	4	5
0	1.20000	0.98248	0.88898	0.80438	0.72784	0.65857
1	1.08580	0.88898	0.80438	0.72784	0.65857	0.59590
2	0.98248	0.80438	0.72784	0.65857	0.59590	0.53919
3	0.88898	0.72784	0.65857	0.59590	0.53919	0.48788
4	0.80438	0.65857	0.59590	0.53919	0.48788	0.44145
...	...	...	...	...	...	...
1188	0.49305	0.61897	0.69324	0.76448	0.82756	0.88001
1189	0.54945	0.69324	0.76448	0.82756	0.88001	0.92119
1190	0.61897	0.76448	0.82756	0.88001	0.92119	0.95145
1191	0.69324	0.82756	0.88001	0.92119	0.95145	0.97162
1192	0.76448	0.88001	0.92119	0.95145	0.97162	0.98283

Predicciones después del dataset

Con las configuraciones de 2 y 4 entradas no es suficiente para hacer un modelo que prediga más allá, por lo que para predecir a futuro se amplió la ventana en ambos dataset.

MG dataset con 10 entradas y 1 salida

Predicciones después del dataset



Bitcoin dataset con 5 entradas y 1 salida

Referencias

Bora, N. (2021). *Understanding ARIMA models for machine learning*. Capital One. https://www.capitalone.com/tech/machine-learning/understanding-arima-models/

Fernandez, J. (2022). *Creating an ARIMA model for time series forecasting*. Towards Data Science. https://towardsdatascience.com/creating-an-arima-model-for-time-series-forecasting-ff3b619b848d

Liu, K., Chen, Y., & Zhang, X. (2017). An evaluation of ARFIMA (autoregressive fractional integral moving average) programs. *Axioms*, *6*(4), 16. https://doi.org/10.3390/axioms6020016

Maklin, C. (2019). *ARIMA model Python example — time series forecasting*. Towards Data Science. https://towardsdatascience.com/machine-learning-part-19-time-series-and-autoregressive-integrated-moving-average-model-arima-c1005347b0d7

Simplilearn. (2021). *Understanding time series analysis in python*. Simplilearn.com; Simplilearn. https://www.simplilearn.com/tutorials/python-tutorial/time-series-analysis-in-python

Dataset bitcoin:

https://finance.yahoo.com/quote/BTC-USD/history?period1=1410825600&period2=1667692800&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true