SERIES DE TIEMPO

Data: https://www.kaggle.com/datasets/podsyp/time-series-starter-dataset

Acerca de la data:

- **Contexto:** El aprendizaje automático se puede aplicar a conjuntos de datos de series temporales.
- **Contenido:** Un problema al iniciarse en la predicción de series temporales con aprendizaje automático es encontrar conjuntos de datos estándar de buena calidad con los que practicar.

```
import pandas as pd
import numpy as np

df = pd.read_csv('Month_Value.csv')
df.info()
```

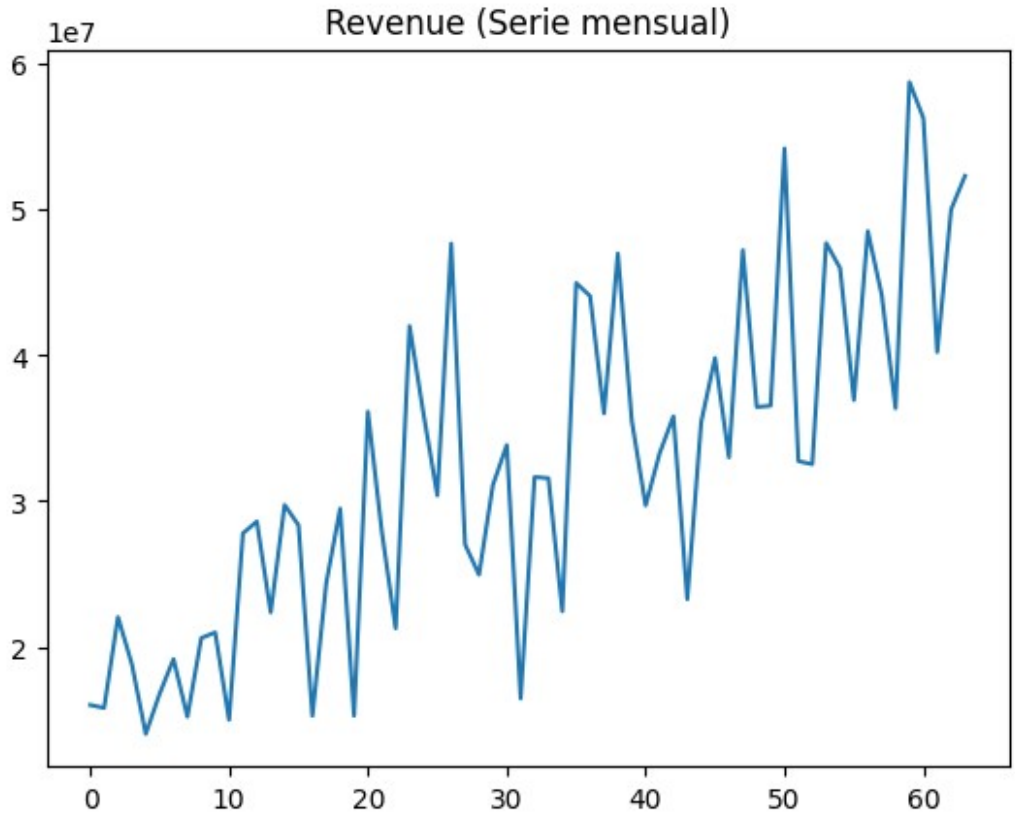
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 5 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   Period                                  96 non-null     object 
 1   Revenue                                 64 non-null     float64
 2   Sales_quantity                          64 non-null     float64
 3   Average_cost                            64 non-null     float64
 4   The_average_annual_payroll_of_the_region  64 non-null     float64
dtypes: float64(4), object(1)
memory usage: 3.9+ KB
```

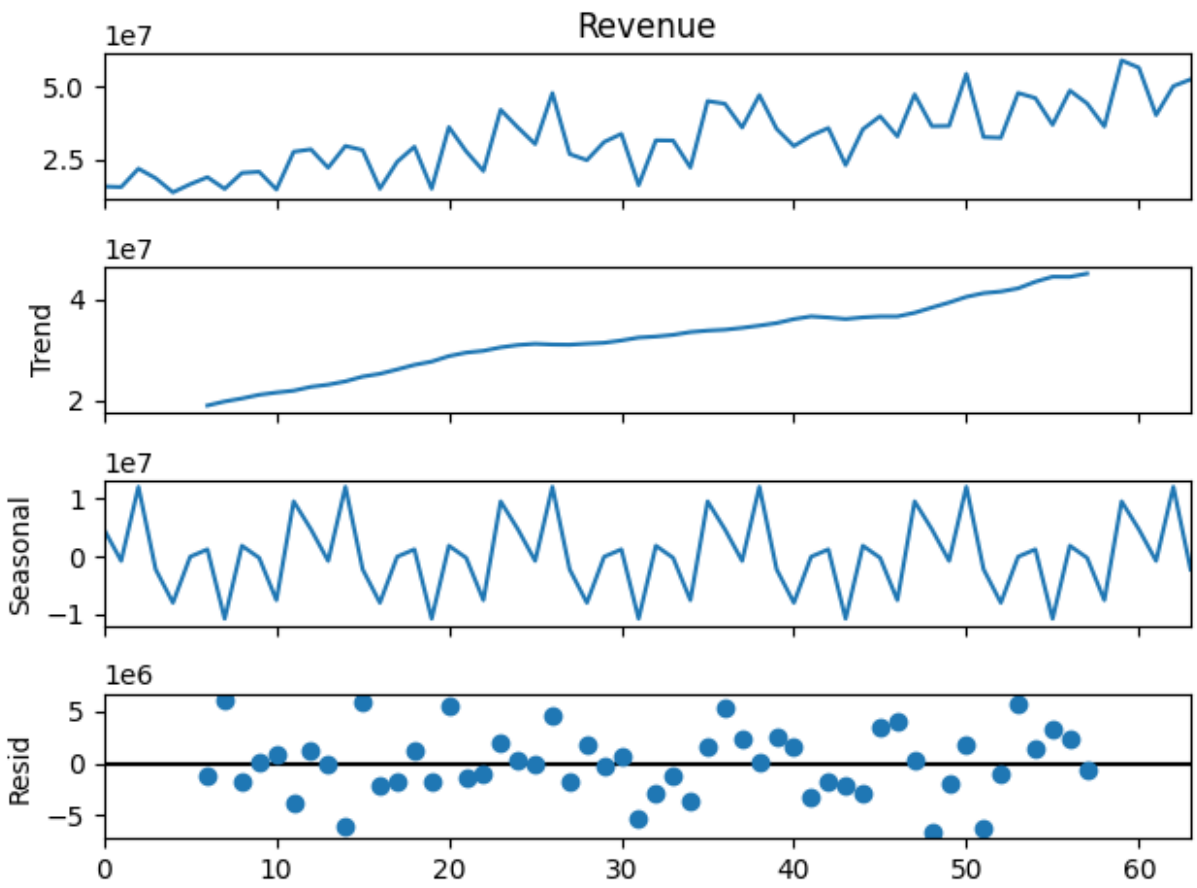
```
# 2) Explorar nulos
print(df.isna().mean().sort_values(ascending=False))

y = df['Revenue'].dropna()
```

```
Revenue                                   0.333333
Average_cost                              0.333333
Sales_quantity                            0.333333
The_average_annual_payroll_of_the_region    0.333333
Period                                    0.000000
dtype: float64
```

```
# 3) Visualización y descomposición
import matplotlib.pyplot as plt

y.plot(title='Revenue (Serie mensual)')
```

```
plt.show()

from statsmodels.tsa.seasonal import seasonal_decompose
decomp = seasonal_decompose(y, model='additive', period=12)
decomp.plot()
plt.show()
```

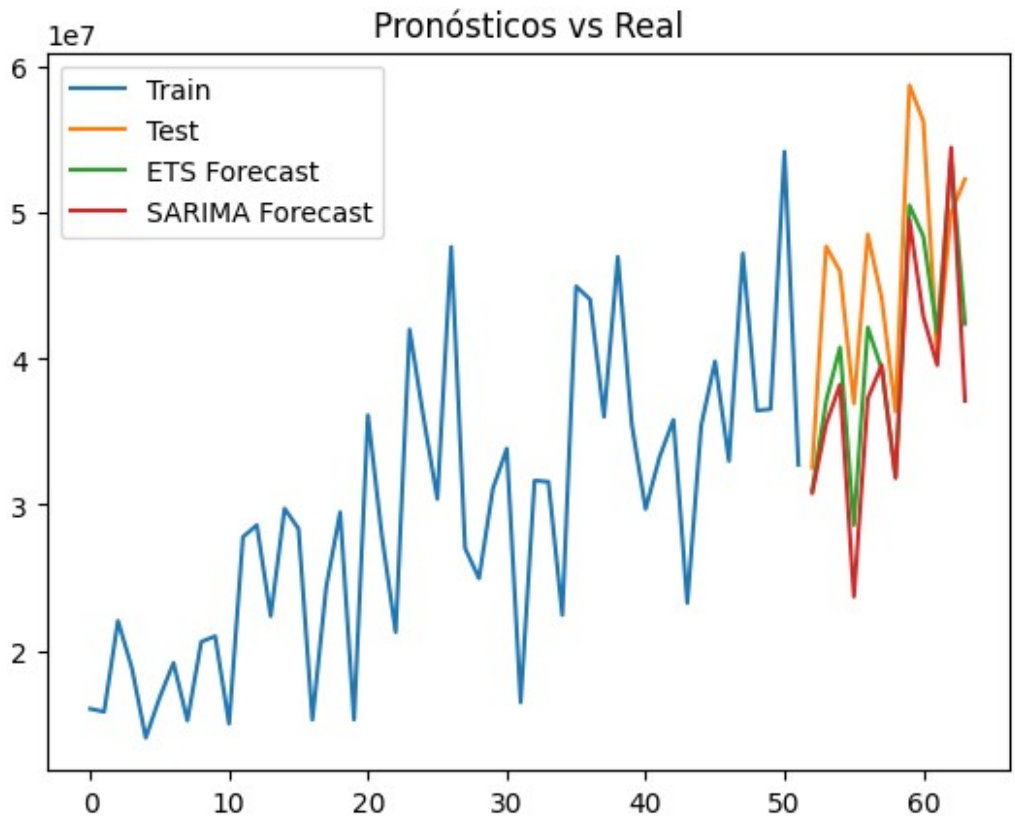





```
# 4) Split train/test (últimos 12 meses como prueba)
test_h = 12
y_train, y_test = y.iloc[:-test_h], y.iloc[-test_h:]

# 5) Modelos base (benchmarks)
yhat_naive = pd.Series(y_train.iloc[-1], index=y_test.index)
if len(y_train) >= 12:
    yhat_snaive = y.shift(12).reindex(y_test.index)
else:
    yhat_snaive = yhat_naive.copy()

from sklearn.metrics import mean_absolute_error, mean_squared_error
def mape(a, f):
    return (np.abs((a - f) / a).replace([np.inf, -np.inf],
np.nan)).dropna().mean()*100

def rmse(a, f):
    return np.sqrt(mean_squared_error(a, f))

print("\n=== Benchmarks ===")
print("Naive   -> MAE:", mean_absolute_error(y_test, yhat_naive),
      "RMSE:", rmse(y_test, yhat_naive),
```

```
      "MAPE:", mape(y_test, yhat_naive))

print("S-Naive -> MAE:", mean_absolute_error(y_test, yhat_snaive),
      "RMSE:", rmse(y_test, yhat_snaive),
      "MAPE:", mape(y_test, yhat_snaive))


=== Benchmarks ===
Naive   -> MAE: 13111538.618015 RMSE: 15216271.592097135 MAPE:
26.43194527056407
S-Naive -> MAE: 10059476.567023333 RMSE: 11722870.991977694 MAPE:
21.162121832657675

# 6) Modelo clásico: ETS (Holt-Winters) o SARIMA
from statsmodels.tsa.holtwinters import ExponentialSmoothing
ets = ExponentialSmoothing(
    y_train, trend='add', seasonal='add', seasonal_periods=12,
initialization_method='estimated'
).fit(optimized=True)
ets_fc = ets.forecast(test_h)
print("\n=== ETS ===")
print("MAE:", mean_absolute_error(y_test, ets_fc),
      "RMSE:", rmse(y_test, ets_fc),
      "MAPE:", mape(y_test, ets_fc))


=== ETS ===
MAE: 6048383.772116501 RMSE: 6704949.33948876 MAPE: 12.937811883651545

import warnings
warnings.filterwarnings("ignore")
import itertools
import statsmodels.api as sm

p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(i, j, k, 12) for i, j, k in pdq]

best_aic, best_order, best_seasonal = np.inf, None, None
for order in pdq:
    for seas in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y_train, order=order,
seasonal_order=seas,

enforce_stationarity=False, enforce_invertibility=False)
            res = mod.fit(disp=False)
            if res.aic < best_aic:
                best_aic, best_order, best_seasonal = res.aic, order,
seas
        except:
```

```
            pass

print("\nMejor SARIMA:", best_order, best_seasonal, "AIC:", best_aic)
sarima = sm.tsa.statespace.SARIMAX(y_train, order=best_order,
seasonal_order=best_seasonal,
                                   enforce_stationarity=False,
enforce_invertibility=False).fit(disp=False)
sarima_fc = sarima.forecast(test_h)

print("\n=== SARIMA ===")
print("MAE:", mean_absolute_error(y_test, sarima_fc),
      "RMSE:", rmse(y_test, sarima_fc),
      "MAPE:", mape(y_test, sarima_fc))


Mejor SARIMA: (0, 1, 1) (1, 1, 1, 12) AIC: 850.541184797142

=== SARIMA ===
MAE: 8183188.678398053 RMSE: 9443617.606850306 MAPE:
17.388751035539443

# 7) Visualizar comparativo
plt.plot(y_train.index, y_train, label='Train')
plt.plot(y_test.index, y_test, label='Test')
plt.plot(y_test.index, ets_fc, label='ETS Forecast')
plt.plot(y_test.index, sarima_fc, label='SARIMA Forecast')
plt.legend(); plt.title("Pronósticos vs Real"); plt.show()
```



Conclusión comparativa

- ETS: más estable, pero tiende a subestimar picos y caídas extremas.
- SARIMA: más sensible, captura mejor la variabilidad, aunque corre el riesgo de sobreajuste.