Laboratorio #3: Comparación de algoritmos de Series de Tiempo aplicados a diferentes tipos de conjuntos de datos

Integrantes:

- Nombre: Andrea de Lourdes Lam Pelaez Carné: 20102
- Nombre: Gabriel Alejandro Vicente Lorenzo Carné: 20498

Objetivos del Laboratorio:

1. Familiarizar a los estudiantes con diferentes algoritmos de series de tiempo.
2. Comprender cuándo y cómo aplicar distintos algoritmos dependiendo de las características de la serie de tiempo (tendencia, estacionalidad).
3. Evaluar el rendimiento de los algoritmos utilizando métricas de evaluación apropiadas. Herramientas Requeridas:

- Python (pandas, numpy, matplotlib, scikit-learn, statsmodels, Prophet, TensorFlow, darts, etc.)
- Jupyter Notebook o Google Colab
- Conjuntos de datos proporcionados para el laboratorio

Instrucciones:

- Conjunto de Datos 1: daily-total-female-births.csv
- Conjunto de Datos 2: monthly-car-sales.csv
- Conjunto de Datos 3: monthly-mean-temp.csv
- Conjunto de Datos 4: shampoo.csv

Para cada conjunto de datos, realizar lo siguiente:

1. Análisis Exploratorio: • Describir la serie de tiempo y visualizarla.

2. Promedios: • Aplicar métodos de promedios y comparar los resultados con el conjunto original.
3. SARIMA: • Identificar parámetros y ajustar un modelo SARIMA.
4. Alisamiento Exponencial: • Aplicar diferentes métodos de alisamiento exponencial y comparar.
5. Prophet: • Utilizar Prophet para modelar la serie de tiempo.
6. Redes Neuronales: • Implementar una red neuronal simple para prever la serie de tiempo.
7. Comparación y Evaluación: • Usar métricas como RMSE, MAE para comparar los modelos. • Discutir cuál algoritmo se desempeña mejor para cada tipo de conjunto de datos y por qué.

Librerias necesarias para el Laboratorio 3

```
import numpy as np
import pandas as pd
from math import sqrt
import tensorflow as tf
from prophet import Prophet
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

```
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.
packages\Python310\site-packages\tqdm\auto.py:21: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Importing plotly failed. Interactive plots will not work.
```

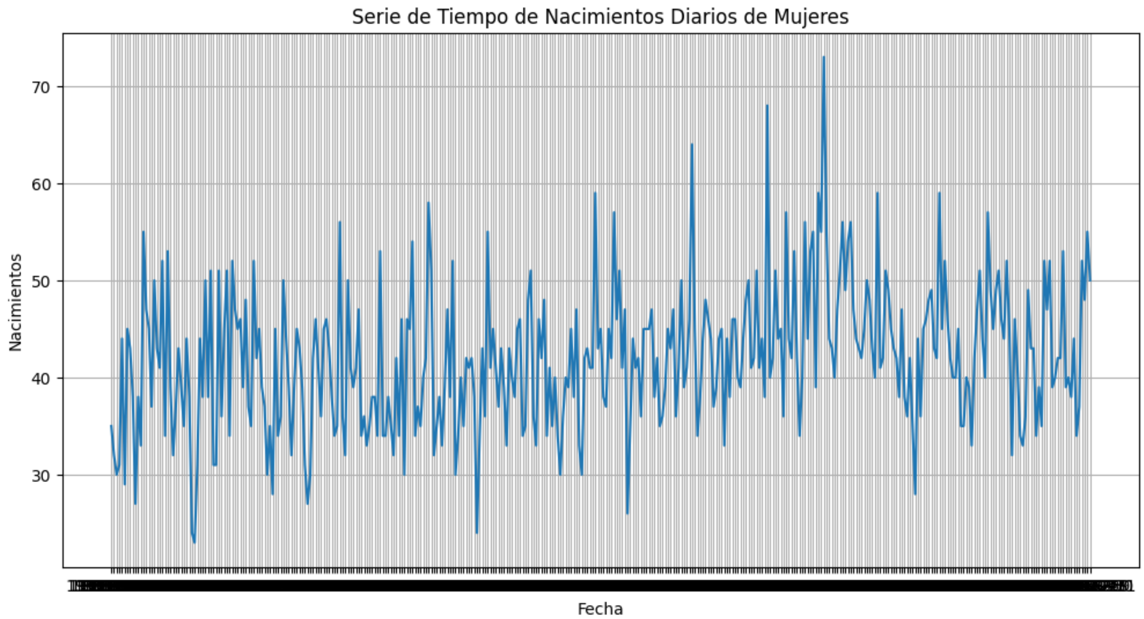
Conjunto de Datos 1: daily-total-female-births.csv

1.) Análisis Exploratorio:

- Describir la serie de tiempo y visualizarla.

```
data = pd.read_csv("./Datos/daily-total-female-births.csv")

print(data.head())

print(data.describe())

plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['Births'])
plt.title('Serie de Tiempo de Nacimientos Diarios de Mujeres')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.grid(True)
plt.show()
```

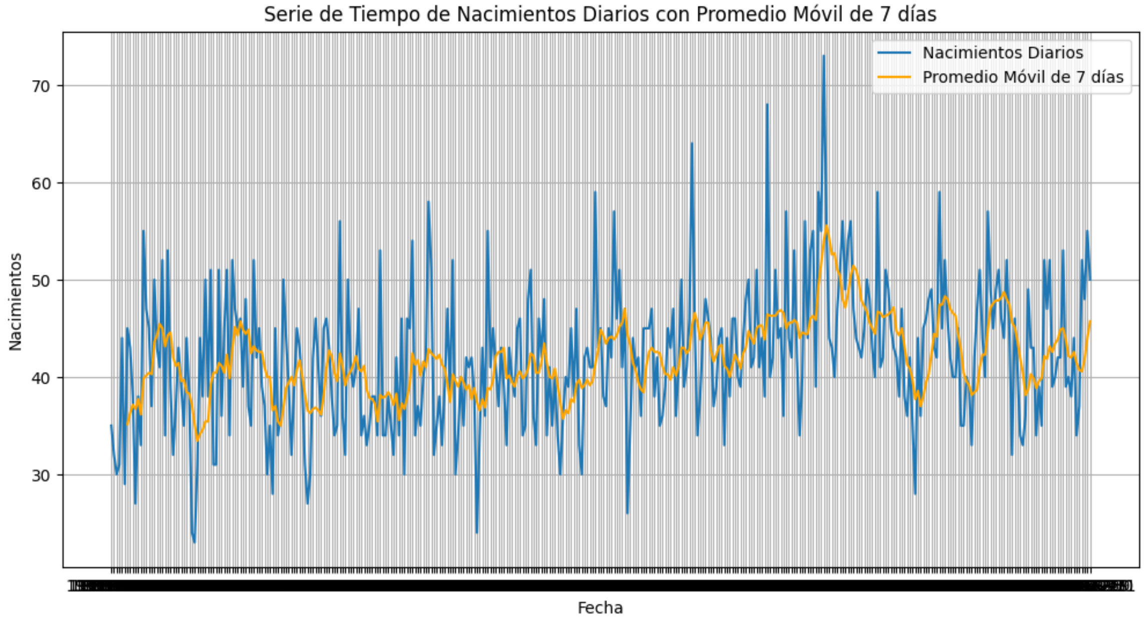
```
         Date  Births
0  1959-01-01      35
1  1959-01-02      32
2  1959-01-03      30
3  1959-01-04      31
4  1959-01-05      44
           Births
count  365.000000
mean    41.980822
std      7.348257
min     23.000000
25%     37.000000
50%     42.000000
75%     46.000000
max     73.000000
```



2.) Promedios:

- Aplicar métodos de promedios y comparar los resultados con el conjunto original.

```
data['Moving_Average_7days'] = data['Births'].rolling(window=7).mean()
plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['Births'], label='Nacimientos Diarios')
plt.plot(data['Date'], data['Moving_Average_7days'], label='Promedio
        Móvil de 7 días', color='orange')
plt.title('Serie de Tiempo de Nacimientos Diarios con Promedio Móvil
        de 7 días')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.legend()
plt.grid(True)
plt.show()
```

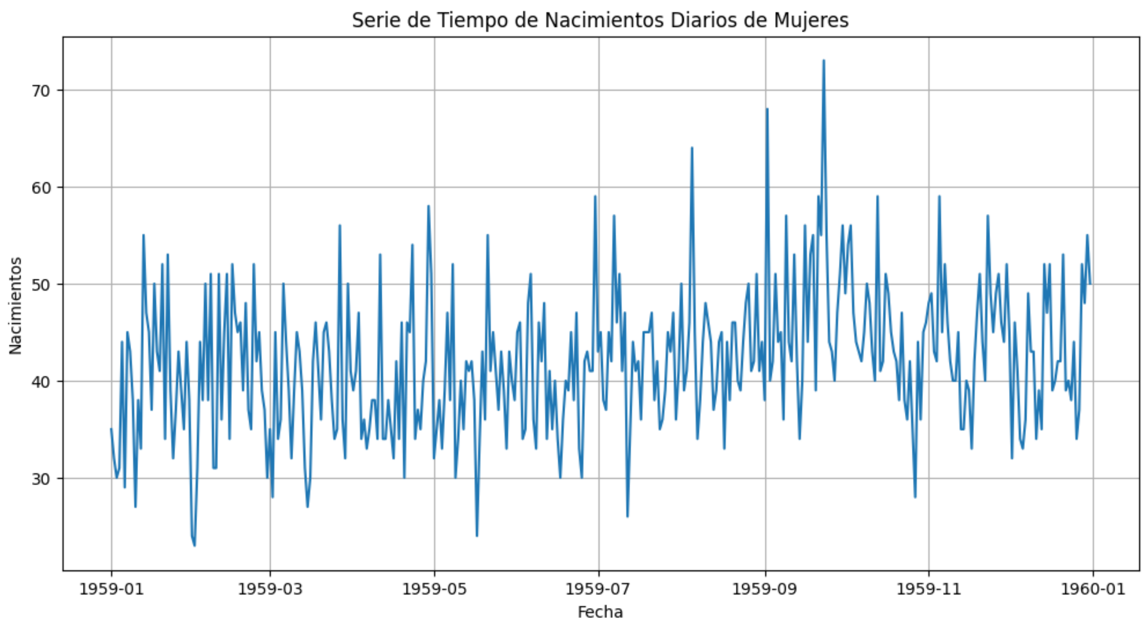


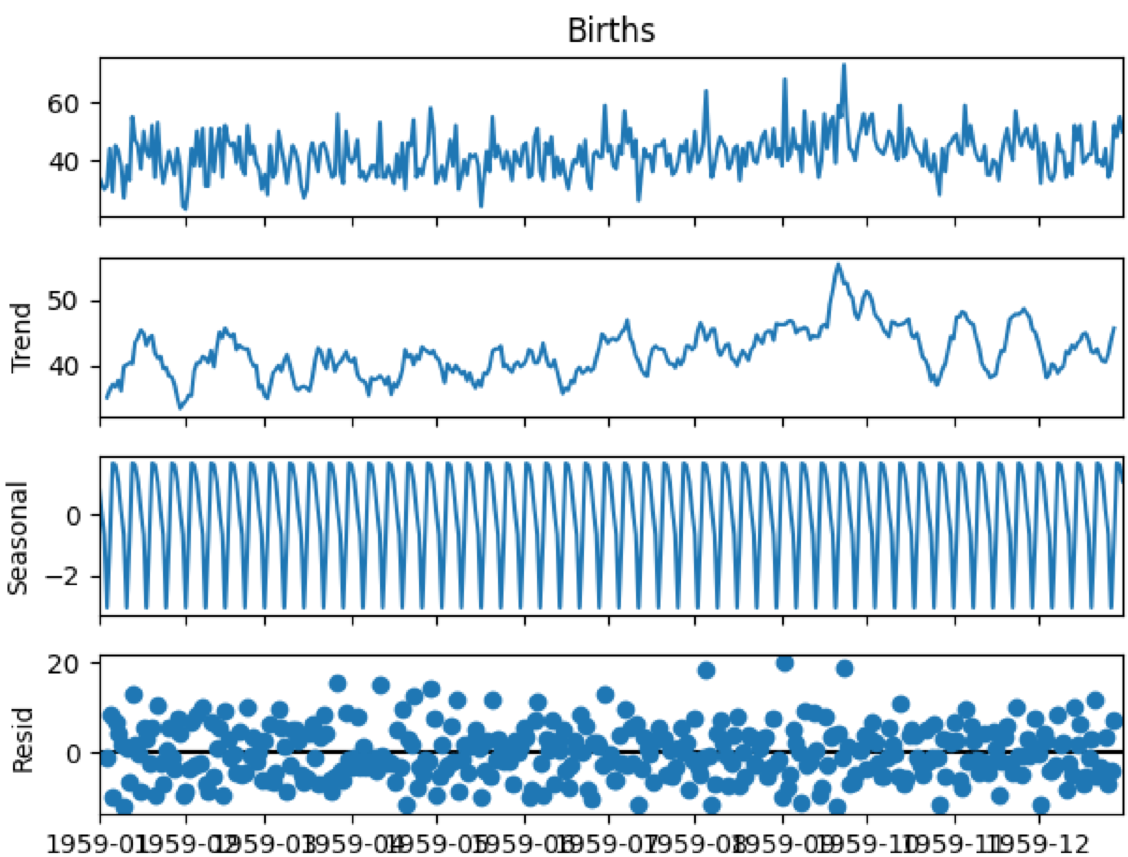
3.) SARIMA:

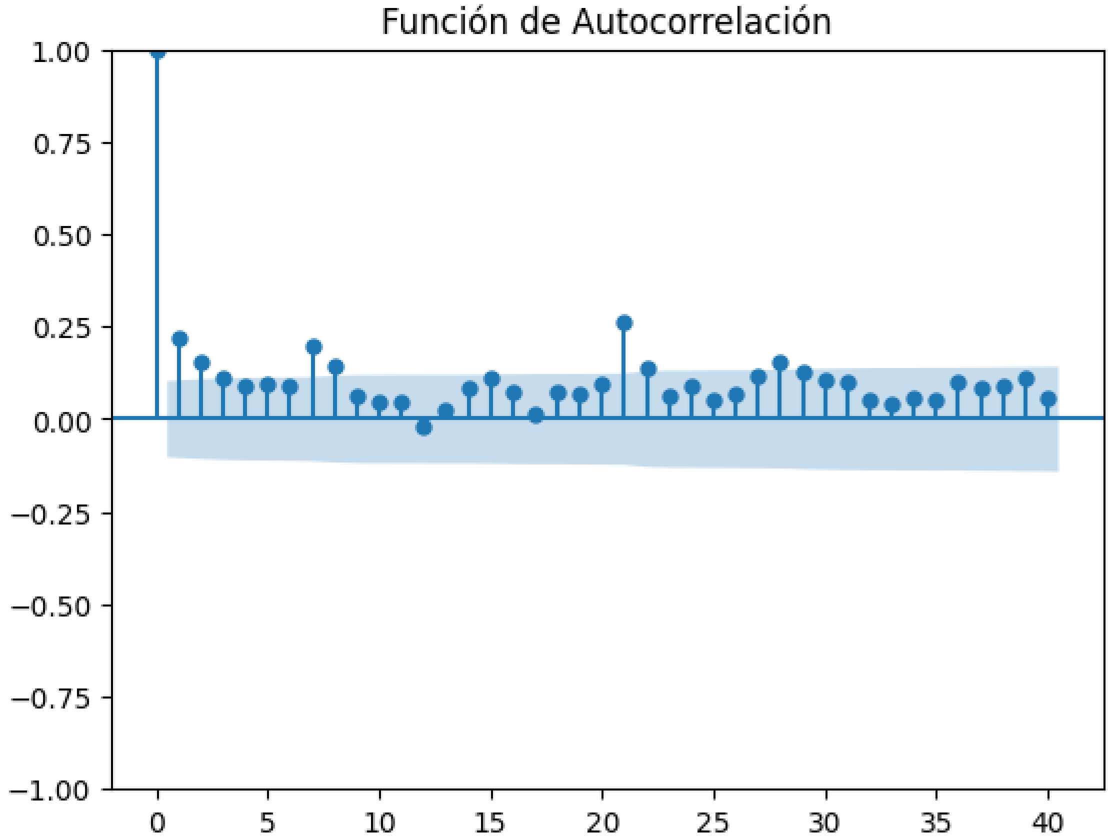
- Identificar parámetros y ajustar un modelo SARIMA.

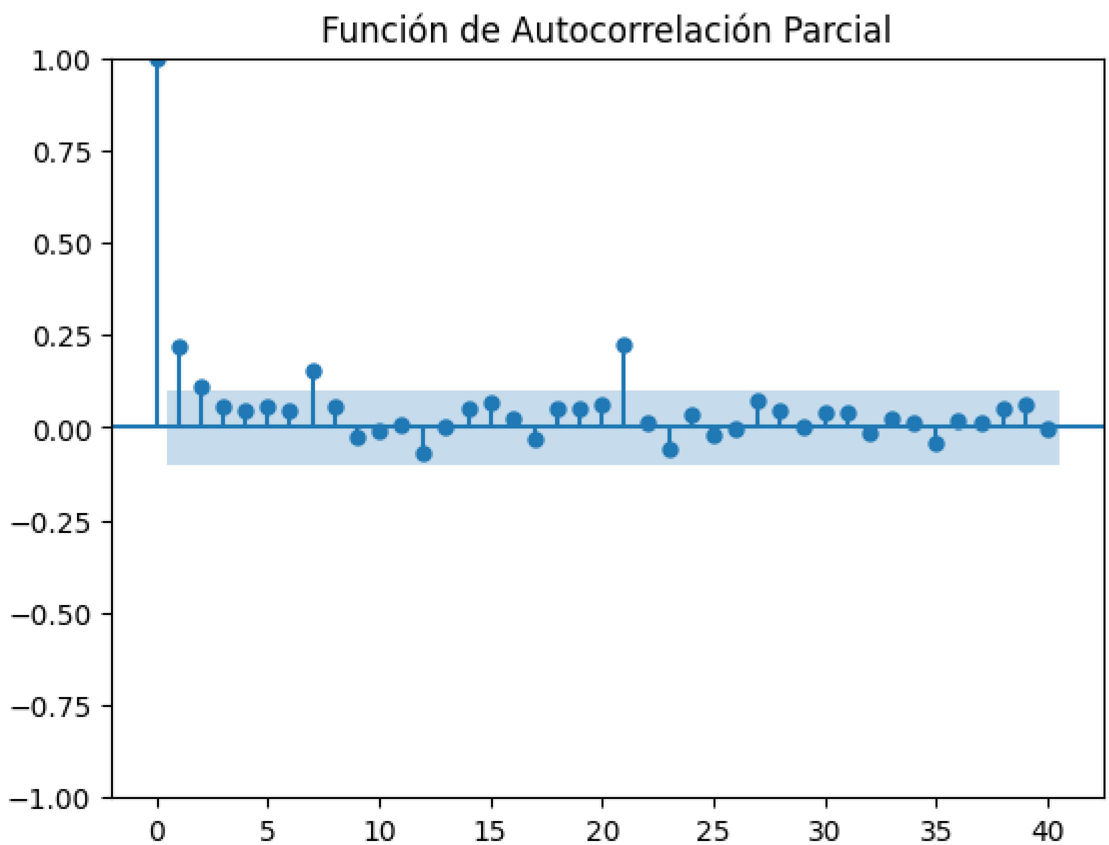
```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

data = pd.read_csv("./Datos/daily-total-female-births.csv")
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)

plt.figure(figsize=(12, 6))
plt.plot(data['Births'])
plt.title('Serie de Tiempo de Nacimientos Diarios de Mujeres')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.grid(True)
plt.show()

result = seasonal_decompose(data['Births'], model='additive')
result.plot()
plt.show()
```

```
plot_acf(data['Births'], lags=40)
plt.title('Función de Autocorrelación')
plt.show()

plot_pacf(data['Births'], lags=40)
plt.title('Función de Autocorrelación Parcial')
plt.show()
```










```
sarima_model = sm.tsa.SARIMAX(data['Births'], order=(1, 0, 1),
        seasonal_order=(1, 0, 1, 365))
sarima_results = sarima_model.fit()
```

```
print(sarima_results.summary())
```

```
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency D will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency D will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.
packages\Python310\site-
packages\statsmodels\tsa\statespace\sarimax.py:866: UserWarning: Too
few observations to estimate starting parameters for seasonal ARMA.
All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
```

```
                                      SARIMAX Results
======================================================================
Dep. Variable:                             Births   No. Observations:
365
Model:             SARIMAX(1, 0, 1)x(1, 0, 1, 365)   Log Likelihood
-1234.216
Date:                            Mon, 04 Sep 2023   AIC
2478.433
Time:                                    12:36:10   BIC
2497.932
Sample:                                01-01-1959   HQIC
2486.182
                                     - 12-31-1959
Covariance Type:                              opg
======================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
```

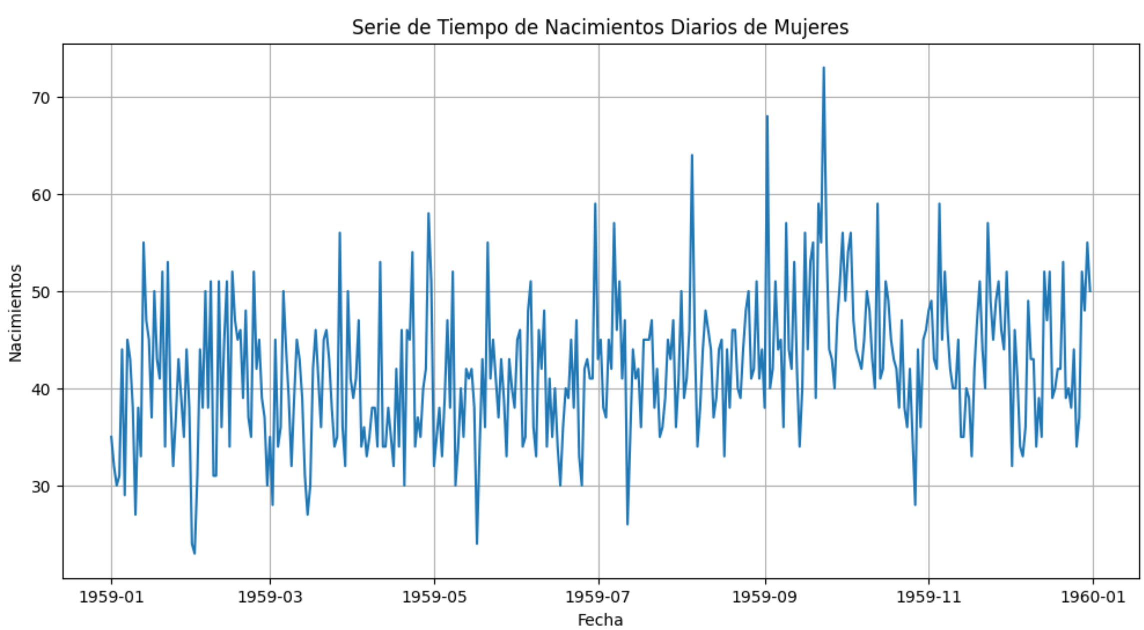
```
ar.L1          0.9998      0.001   1139.603      0.000       0.998
1.002
ma.L1         -0.9475      0.019    -48.693      0.000      -0.986
-0.909
ar.S.L365      0.8285    136.880      0.006      0.995    -267.452
269.109
ma.S.L365     -0.3823    570.171     -0.001      0.999   -1117.896
1117.132
sigma2        30.4927   1.44e+04      0.002      0.998   -2.83e+04
2.83e+04
======================================================================
Ljung-Box (L1) (Q):                   4.15   Jarque-Bera (JB):
19.91
Prob(Q):                              0.04   Prob(JB):
0.00
Heteroskedasticity (H):               0.97   Skew:
0.51
Prob(H) (two-sided):                  0.87   Kurtosis:
3.53
======================================================================

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

4.) Alisamiento Exponencial:

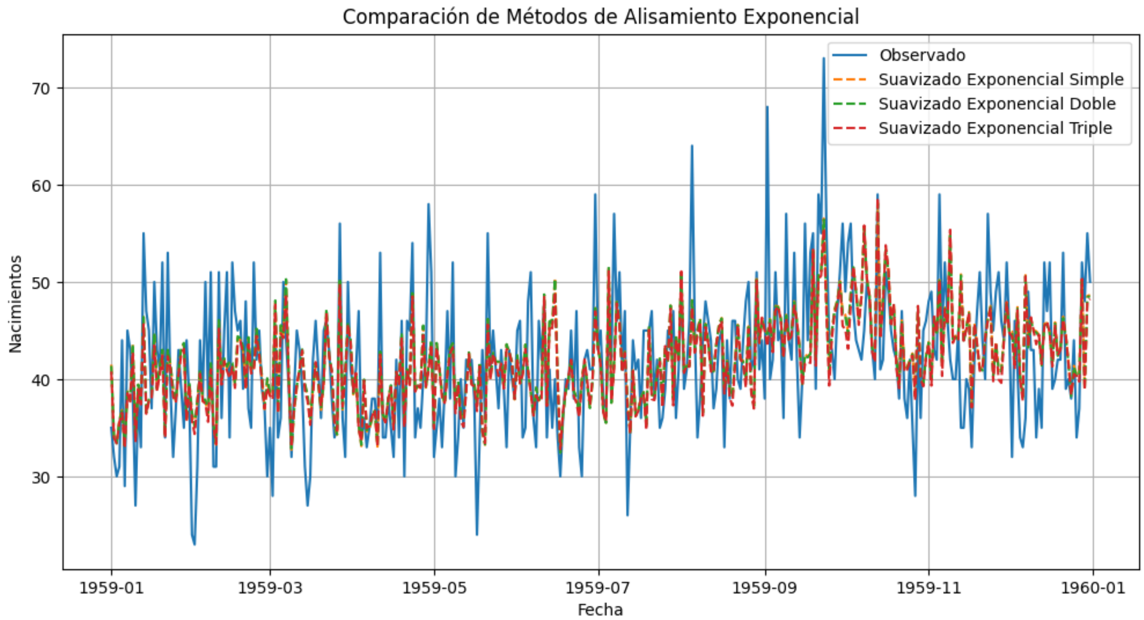
- Aplicar diferentes métodos de alisamiento exponencial y comparar.

```
data = pd.read_csv("./Datos/daily-total-female-births.csv")
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
plt.figure(figsize=(12, 6))
plt.plot(data['Births'])
plt.title('Serie de Tiempo de Nacimientos Diarios de Mujeres')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.grid(True)
plt.show()
```

```
exp_smoothing_simple = sm.tsa.ExponentialSmoothing(data['Births'],
        trend='add', seasonal='add', seasonal_periods=100)
exp_smoothing_simple_fit = exp_smoothing_simple.fit()

exp_smoothing_double = sm.tsa.ExponentialSmoothing(data['Births'],
        trend='add', seasonal='add', seasonal_periods=100,
        damped=True)
exp_smoothing_double_fit = exp_smoothing_double.fit()

exp_smoothing_triple = sm.tsa.ExponentialSmoothing(data['Births'],
        trend='add', seasonal='add', seasonal_periods=100,
        damped=True, use_boxcox=True)
exp_smoothing_triple_fit = exp_smoothing_triple.fit()

plt.figure(figsize=(12, 6))
plt.plot(data['Births'], label='Observado')
plt.plot(exp_smoothing_simple_fit.fittedvalues, label='Suavizado
        Exponencial Simple', linestyle='--')
plt.plot(exp_smoothing_double_fit.fittedvalues, label='Suavizado
        Exponencial Doble', linestyle='--')
plt.plot(exp_smoothing_triple_fit.fittedvalues, label='Suavizado
        Exponencial Triple', linestyle='--')
plt.title('Comparación de Métodos de Alisamiento Exponencial')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.legend()
plt.grid(True)
plt.show()
```



```
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.:
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency D will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Temp\ipykernel_16444\2448109169.py:15:
FutureWarning: the 'damped' keyword is deprecated, use 'damped_trend'
instead.
  exp_smoothing_double = sm.tsa.ExponentialSmoothing(data['Births'],
trend='add', seasonal='add', seasonal_periods=100, damped=True)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.:
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency D will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Temp\ipykernel_16444\2448109169.py:18:
FutureWarning: the 'damped' keyword is deprecated, use 'damped_trend'
instead.
  exp_smoothing_triple = sm.tsa.ExponentialSmoothing(data['Births'],
trend='add', seasonal='add', seasonal_periods=100, damped=True,
use_boxcox=True)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.:
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency D will be
used.
  self._init_dates(dates, freq)
```

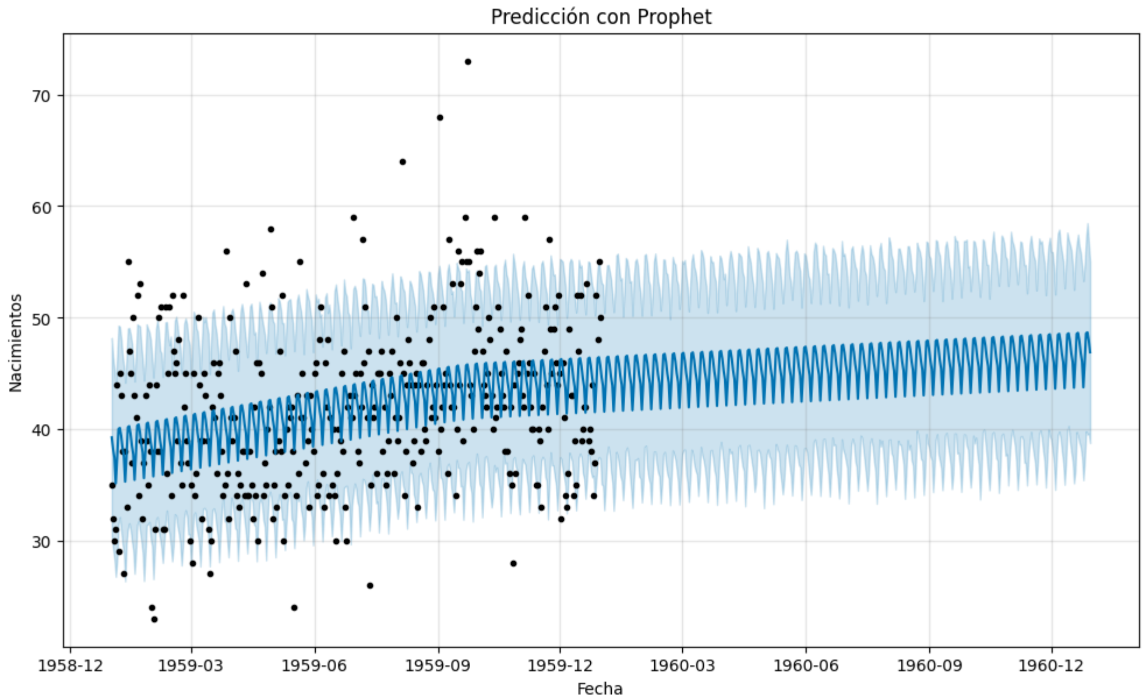


5.) Prophet:

- Utilizar Prophet para modelar la serie de tiempo.

```
data = pd.read_csv("./Datos/daily-total-female-births.csv")
data['Date'] = pd.to_datetime(data['Date'])
data = data.rename(columns={'Date': 'ds', 'Births': 'y'})
model = Prophet()
model.fit(data)
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)
fig = model.plot(forecast)
plt.title('Predicción con Prophet')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.grid(True)
plt.show()
```

```
12:36:24 - cmdstanpy - INFO - Chain [1] start processing
12:36:24 - cmdstanpy - INFO - Chain [1] done processing
```



6.) Redes Neuronales:

- Implementar una red neuronal simple para prever la serie de tiempo.

```
def create_sequences(data, look_back=1):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:(i + look_back)])
        y.append(data[i + look_back])
    return np.array(X), np.array(y)

data = pd.read_csv("./Datos/daily-total-female-births.csv")
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)


scaler = MinMaxScaler()
data['Scaled_Births'] = scaler.fit_transform(data[['Births']])
train_size = int(len(data) * 0.80)
train_data = data.iloc[:train_size]['Scaled_Births'].values
test_data = data.iloc[train_size:]['Scaled_Births'].values
look_back = 7
X_train, y_train = create_sequences(train_data, look_back)
X_test, y_test = create_sequences(test_data, look_back)
model = keras.Sequential([
```

```
    keras.layers.Dense(16, activation='relu', input_shape=
        (look_back,)),
    keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=16, verbose=1)
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
train_predictions = scaler.inverse_transform(train_predictions)
test_predictions = scaler.inverse_transform(test_predictions)
rmse = sqrt(mean_squared_error(data.iloc[(train_size + look_back):]
        ['Births'], test_predictions))
print(f'Error Cuadrático Medio en el Conjunto de Prueba: {rmse}')
plt.figure(figsize=(12, 6))
plt.plot(data.index[(train_size + look_back):], data.iloc[(train_size
        + look_back):]['Births'], label='Observado')
plt.plot(data.index[(train_size + look_back):], test_predictions,
        label='Predicción', linestyle='--')
plt.title('Predicciones de la Serie de Tiempo con Red Neuronal')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch 1/100
18/18 [==============================] - 0s 1ms/step - loss: 0.8112
Epoch 2/100
18/18 [==============================] - 0s 944us/step - loss: 0.4928
Epoch 3/100
18/18 [==============================] - 0s 737us/step - loss: 0.2889
Epoch 4/100
18/18 [==============================] - 0s 941us/step - loss: 0.1602
Epoch 5/100
18/18 [==============================] - 0s 809us/step - loss: 0.0846
Epoch 6/100
18/18 [==============================] - 0s 765us/step - loss: 0.0460
Epoch 7/100
18/18 [==============================] - 0s 706us/step - loss: 0.0317
Epoch 8/100
18/18 [==============================] - 0s 706us/step - loss: 0.0279
Epoch 9/100
18/18 [==============================] - 0s 706us/step - loss: 0.0268
```

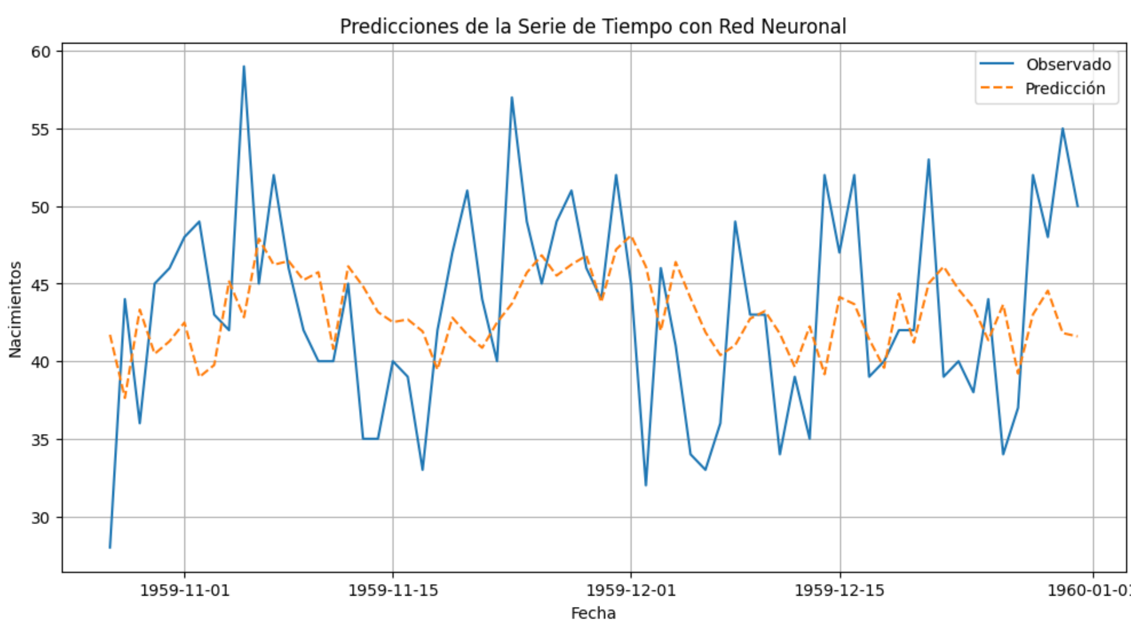
```
Epoch 10/100
18/18 [==============================] - 0s 706us/step - loss: 0.0265
Epoch 11/100
18/18 [==============================] - 0s 706us/step - loss: 0.0262
Epoch 12/100
18/18 [==============================] - 0s 648us/step - loss: 0.0258
Epoch 13/100
18/18 [==============================] - 0s 765us/step - loss: 0.0254
Epoch 14/100
18/18 [==============================] - 0s 706us/step - loss: 0.0251
Epoch 15/100
18/18 [==============================] - 0s 706us/step - loss: 0.0248
Epoch 16/100
18/18 [==============================] - 0s 765us/step - loss: 0.0246
Epoch 17/100
18/18 [==============================] - 0s 765us/step - loss: 0.0243
Epoch 18/100
18/18 [==============================] - 0s 706us/step - loss: 0.0241
Epoch 19/100
18/18 [==============================] - 0s 647us/step - loss: 0.0239
Epoch 20/100
18/18 [==============================] - 0s 733us/step - loss: 0.0236
Epoch 21/100
18/18 [==============================] - 0s 647us/step - loss: 0.0235
Epoch 22/100
18/18 [==============================] - 0s 647us/step - loss: 0.0233
Epoch 23/100
18/18 [==============================] - 0s 765us/step - loss: 0.0231
Epoch 24/100
18/18 [==============================] - 0s 706us/step - loss: 0.0229
Epoch 25/100
18/18 [==============================] - 0s 765us/step - loss: 0.0227
Epoch 26/100
18/18 [==============================] - 0s 706us/step - loss: 0.0226
Epoch 27/100
18/18 [==============================] - 0s 706us/step - loss: 0.0225
Epoch 28/100
18/18 [==============================] - 0s 647us/step - loss: 0.0222
Epoch 29/100
18/18 [==============================] - 0s 765us/step - loss: 0.0222
```

```
Epoch 30/100
18/18 [==============================] - 0s 765us/step - loss: 0.0220
Epoch 31/100
18/18 [==============================] - 0s 706us/step - loss: 0.0219
Epoch 32/100
18/18 [==============================] - 0s 706us/step - loss: 0.0218
Epoch 33/100
18/18 [==============================] - 0s 706us/step - loss: 0.0217
Epoch 34/100
18/18 [==============================] - 0s 707us/step - loss: 0.0216
Epoch 35/100
18/18 [==============================] - 0s 647us/step - loss: 0.0216
Epoch 36/100
18/18 [==============================] - 0s 647us/step - loss: 0.0215
Epoch 37/100
18/18 [==============================] - 0s 824us/step - loss: 0.0214
Epoch 38/100
18/18 [==============================] - 0s 765us/step - loss: 0.0213
Epoch 39/100
18/18 [==============================] - 0s 707us/step - loss: 0.0213
Epoch 40/100
18/18 [==============================] - 0s 706us/step - loss: 0.0212
Epoch 41/100
18/18 [==============================] - 0s 765us/step - loss: 0.0211
Epoch 42/100
18/18 [==============================] - 0s 765us/step - loss: 0.0211
Epoch 43/100
18/18 [==============================] - 0s 765us/step - loss: 0.0210
Epoch 44/100
18/18 [==============================] - 0s 706us/step - loss: 0.0210
Epoch 45/100
18/18 [==============================] - 0s 706us/step - loss: 0.0210
Epoch 46/100
18/18 [==============================] - 0s 706us/step - loss: 0.0209
Epoch 47/100
18/18 [==============================] - 0s 647us/step - loss: 0.0208
Epoch 48/100
18/18 [==============================] - 0s 883us/step - loss: 0.0208
Epoch 49/100
18/18 [==============================] - 0s 824us/step - loss: 0.0208
```

```
Epoch 50/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0207
Epoch 51/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0207
Epoch 52/100
18/18 [==============================] - 0s 767us/step - loss: 0.0206
Epoch 53/100
18/18 [==============================] - 0s 648us/step - loss: 0.0206
Epoch 54/100
18/18 [==============================] - 0s 647us/step - loss: 0.0206
Epoch 55/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0205
Epoch 56/100
18/18 [==============================] - 0s 891us/step - loss: 0.0205
Epoch 57/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0205
Epoch 58/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0205
Epoch 59/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0204
Epoch 60/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0204
Epoch 61/100
18/18 [==============================] - 0s 941us/step - loss: 0.0203
Epoch 62/100
18/18 [==============================] - 0s 941us/step - loss: 0.0203
Epoch 63/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0204
Epoch 64/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0203
Epoch 65/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0202
Epoch 66/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0202
Epoch 67/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0202
Epoch 68/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0202
Epoch 69/100
18/18 [==============================] - 0s 1000us/step - loss: 0.0201
```

```
Epoch 70/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0201
Epoch 71/100
18/18 [==============================] - 0s 941us/step - loss: 0.0201
Epoch 72/100
18/18 [==============================] - 0s 1000us/step - loss: 0.0201
Epoch 73/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0201
Epoch 74/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0201
Epoch 75/100
18/18 [==============================] - 0s 941us/step - loss: 0.0200
Epoch 76/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0200
Epoch 77/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0201
Epoch 78/100
18/18 [==============================] - 0s 882us/step - loss: 0.0199
Epoch 79/100
18/18 [==============================] - 0s 942us/step - loss: 0.0199
Epoch 80/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0200
Epoch 81/100
18/18 [==============================] - 0s 941us/step - loss: 0.0199
Epoch 82/100
18/18 [==============================] - 0s 882us/step - loss: 0.0199
Epoch 83/100
18/18 [==============================] - 0s 882us/step - loss: 0.0199
Epoch 84/100
18/18 [==============================] - 0s 941us/step - loss: 0.0199
Epoch 85/100
18/18 [==============================] - 0s 2ms/step - loss: 0.0199
Epoch 86/100
18/18 [==============================] - 0s 765us/step - loss: 0.0198
Epoch 87/100
18/18 [==============================] - 0s 706us/step - loss: 0.0198
Epoch 88/100
18/18 [==============================] - 0s 765us/step - loss: 0.0199
Epoch 89/100
18/18 [==============================] - 0s 824us/step - loss: 0.0199
```

```
Epoch 90/100
18/18 [==============================] - 0s 765us/step - loss: 0.0197
Epoch 91/100
18/18 [==============================] - 0s 765us/step - loss: 0.0199
Epoch 92/100
18/18 [==============================] - 0s 824us/step - loss: 0.0197
Epoch 93/100
18/18 [==============================] - 0s 941us/step - loss: 0.0197
Epoch 94/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0197
Epoch 95/100
18/18 [==============================] - 0s 941us/step - loss: 0.0197
Epoch 96/100
18/18 [==============================] - 0s 941us/step - loss: 0.0197
Epoch 97/100
18/18 [==============================] - 0s 941us/step - loss: 0.0197
Epoch 98/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0197
Epoch 99/100
18/18 [==============================] - 0s 1ms/step - loss: 0.0198
Epoch 100/100
18/18 [==============================] - 0s 941us/step - loss: 0.0197
9/9 [==============================] - 0s 764us/step
3/3 [==============================] - 0s 3ms/step
Error Cuadrático Medio en el Conjunto de Prueba: 6.6911170430813
```



7.) Comparación y Evaluación:

- Usar métricas como RMSE, MAE para comparar los modelos.

```
""" Modelo SARIMA """

from statsmodels.tools.eval_measures import rmse, meanabs
sarima_predictions = sarima_results.get_prediction(start=0,
        end=len(data)-1)
sarima_forecast = sarima_predictions.predicted_mean
sarima_rmse = rmse(data['Births'], sarima_forecast)
sarima_mae = meanabs(data['Births'], sarima_forecast)

print(f'RMSE del modelo SARIMA: {sarima_rmse:.2f}')
print(f'MAE del modelo SARIMA: {sarima_mae:.2f}')
```

```
RMSE del modelo SARIMA: 7.30
MAE del modelo SARIMA: 5.66
```

```
""" Red neuronal Simple """
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
nn_rmse = sqrt(mean_squared_error(data.iloc[(train_size + look_back):]
        ['Births'], test_predictions))
nn_mae = mean_absolute_error(data.iloc[(train_size + look_back):]
        ['Births'], test_predictions)

print(f'RMSE de la Red Neuronal Simple: {nn_rmse:.2f}')
print(f'MAE de la Red Neuronal Simple: {nn_mae:.2f}')
```

```
RMSE de la Red Neuronal Simple: 6.69
MAE de la Red Neuronal Simple: 5.45
```

```
""" Prophet """

data = pd.read_csv("./Datos/daily-total-female-births.csv")
data['Date'] = pd.to_datetime(data['Date'])
data = data.rename(columns={'Date': 'ds', 'Births': 'y'})
train_data = data.iloc[:-65]
test_data = data.iloc[-65:]
model = Prophet()
model.fit(train_data)
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)
```

```
predicted_values = forecast.iloc[-65:]['yhat']
rmse = np.sqrt(mean_squared_error(test_data['y'], predicted_values))
mae = mean_absolute_error(test_data['y'], predicted_values)

print(f"RMSE de Prophet: {rmse:.2f}")
print(f"MAE de Prophet: {mae:.2f}")
```

```
12:36:37 - cmdstanpy - INFO - Chain [1] start processing
12:36:37 - cmdstanpy - INFO - Chain [1] done processing
```

```
RMSE de Prophet: 15.27
MAE de Prophet: 13.77
```

- Discutir cuál algoritmo se desempeña mejor para cada tipo de conjunto de datos y por qué.

Los valores más bajos tanto de RMSE como de MAE los tiene la "Red Neuronal Simple" con un RMSE de 6.69 y un MAE de 5.45. Esto sugiere que la Red Neuronal Simple es el modelo que tiene un mejor rendimiento en la predicción en función de las métricas proporcionadas. Debido a que no posee una tendencia clara y son periodicos relativamente hablando.

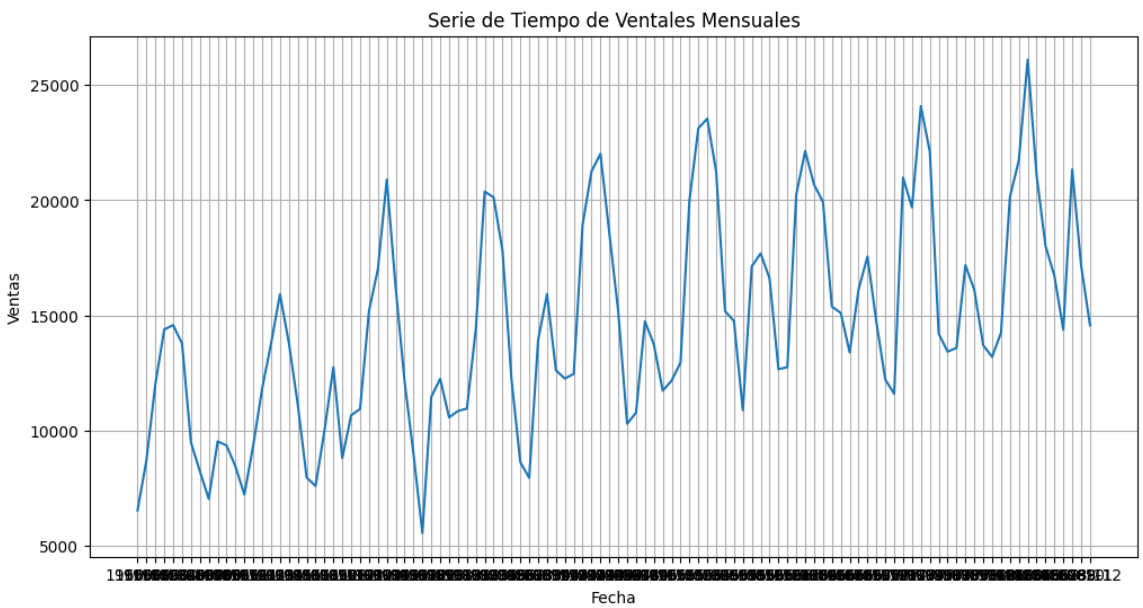
Conjunto de Datos 2: monthly-car-sales.csv

1.) Análisis Exploratorio:

- Describir la serie de tiempo y visualizarla.

```
data = pd.read_csv("./Datos/monthly-car-sales.csv")

print(data.head())

print(data.describe())

plt.figure(figsize=(12, 6))
plt.plot(data['Month'], data['Sales'])
plt.title('Serie de Tiempo de Ventales Mensuales')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.grid(True)
plt.show()
```

```
     Month  Sales
0  1960-01   6550
1  1960-02   8728
2  1960-03  12026
3  1960-04  14395
4  1960-05  14587
              Sales
count    108.000000
mean   14595.111111
std     4525.213913
min     5568.000000
25%    11391.250000
50%    14076.000000
75%    17595.750000
max    26099.000000
```

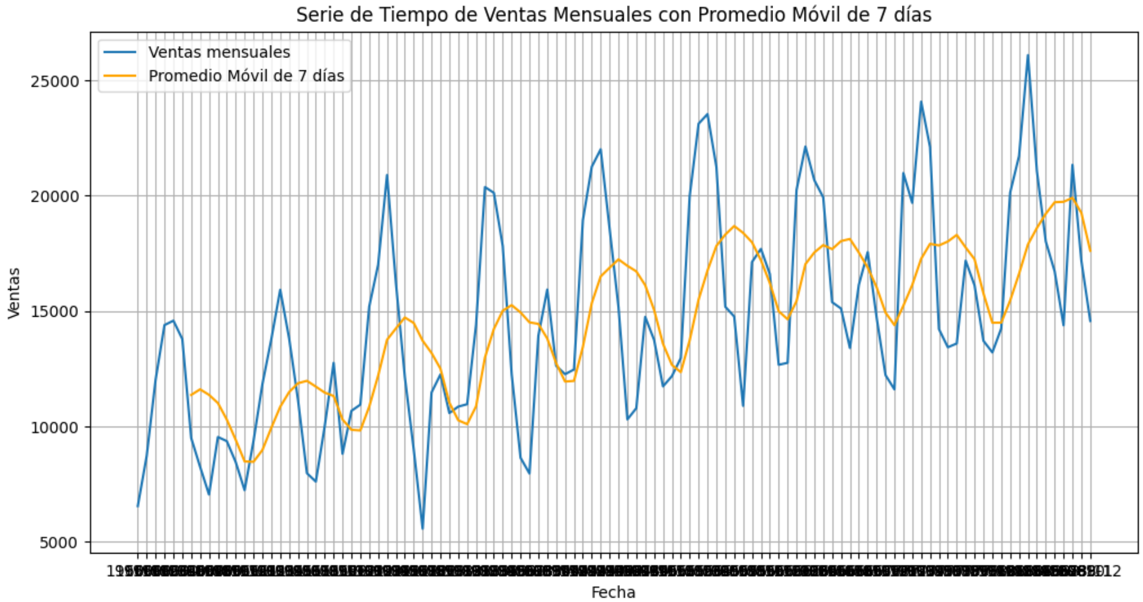


2.) Promedios:

- Aplicar métodos de promedios y comparar los resultados con el conjunto original.

```
data['Moving_Average_7days'] = data['Sales'].rolling(window=7).mean()
plt.figure(figsize=(12, 6))
plt.plot(data['Month'], data['Sales'], label='Ventas mensuales')
plt.plot(data['Month'], data['Moving_Average_7days'], label='Promedio
        Móvil de 7 días', color='orange')
plt.title('Serie de Tiempo de Ventas Mensuales con Promedio Móvil de 7
        meses')
```

```
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.legend()
plt.grid(True)
plt.show()
```

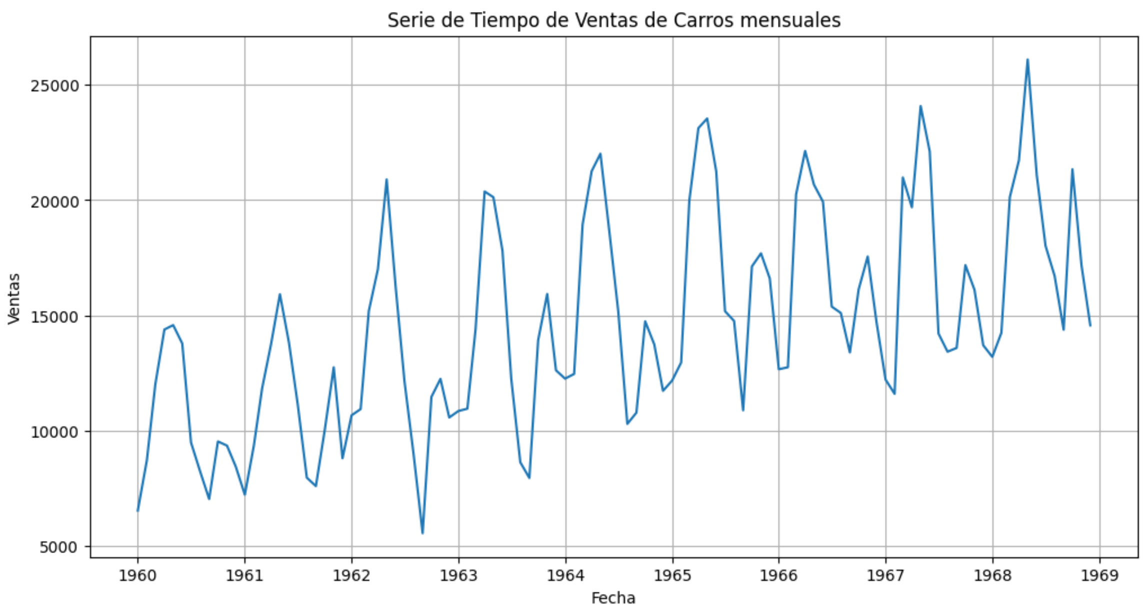


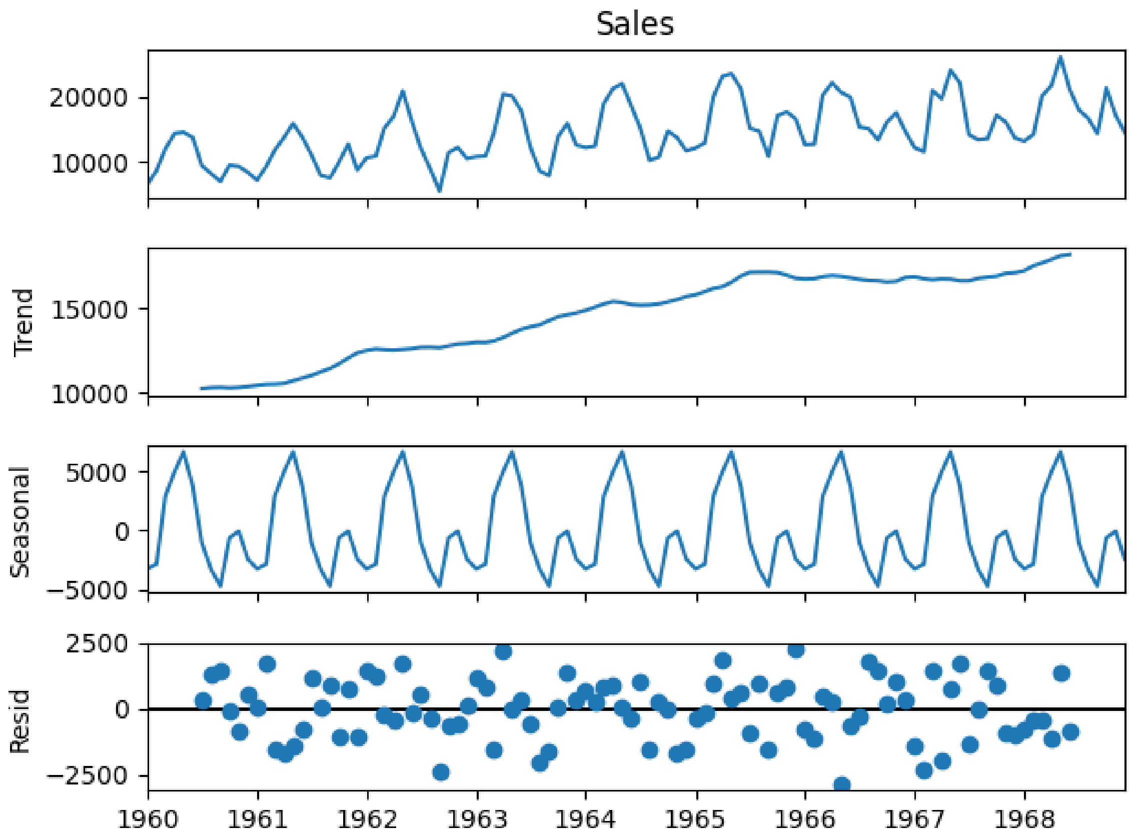
3.) SARIMA:

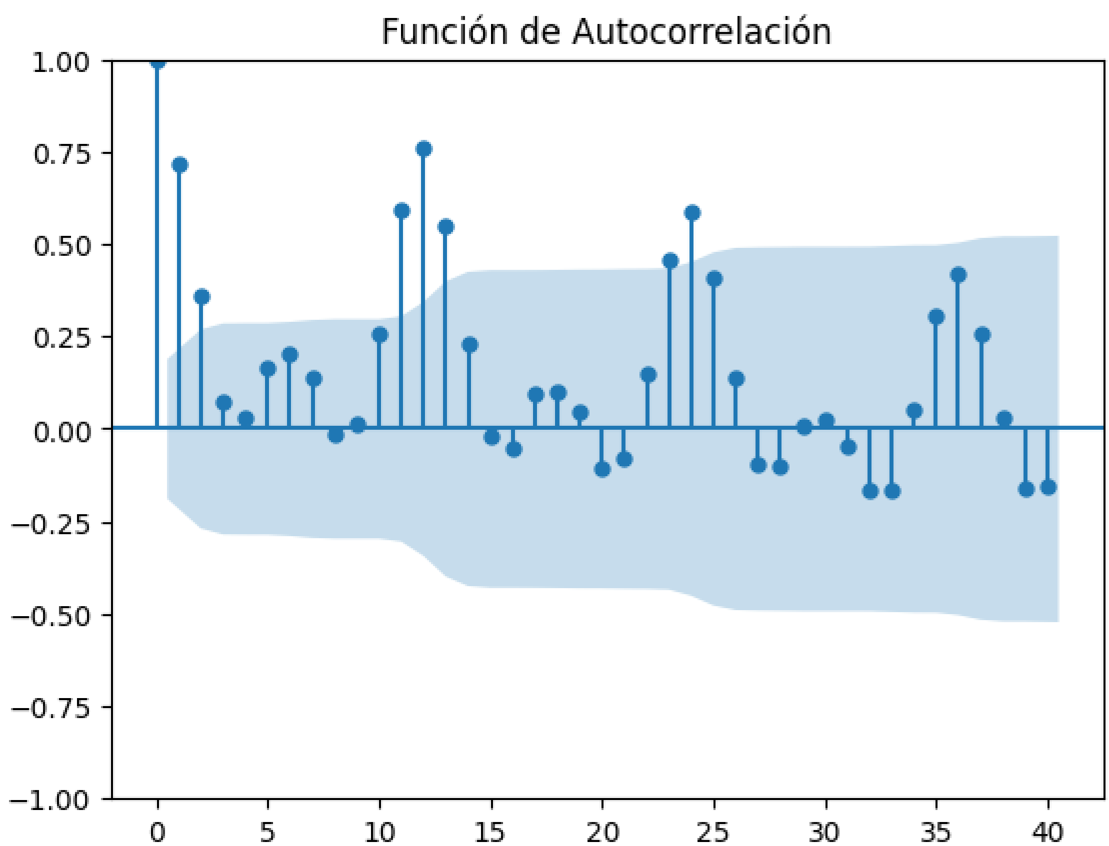
- Identificar parámetros y ajustar un modelo SARIMA.

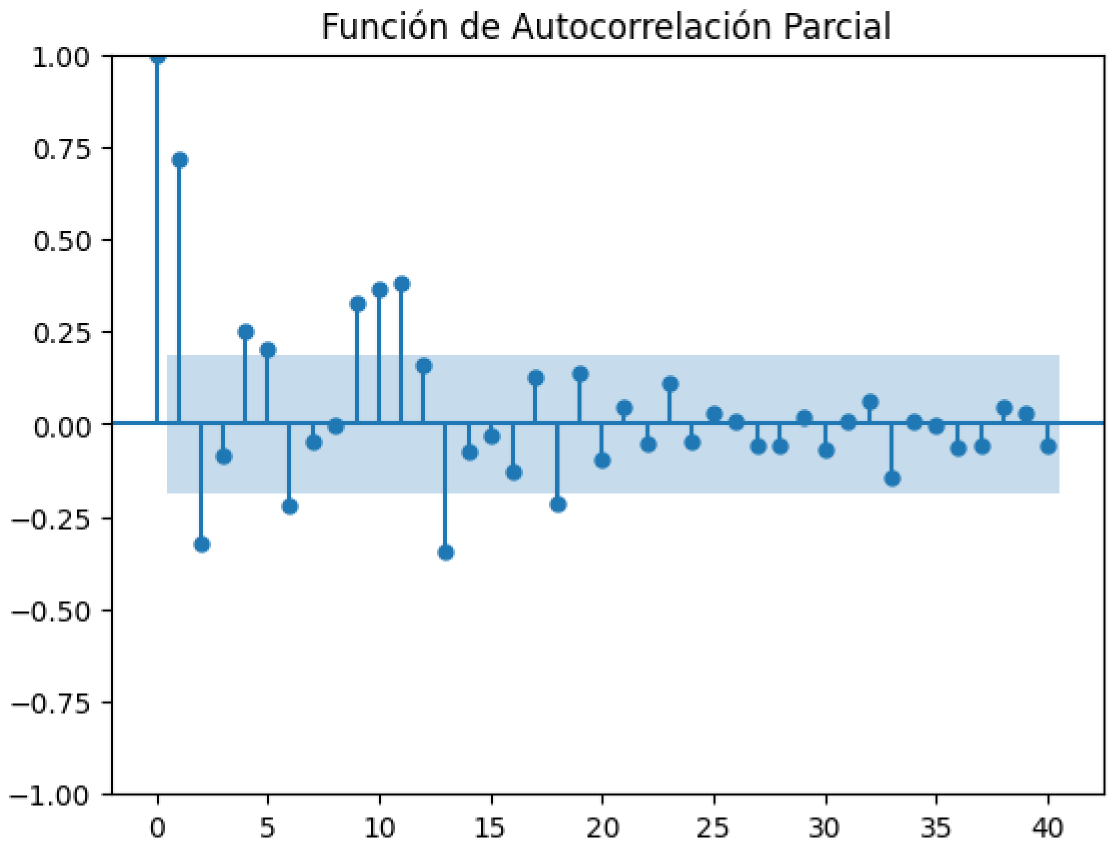
```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

data = pd.read_csv("./Datos/monthly-car-sales.csv")
data['Month'] = pd.to_datetime(data['Month'])
data.set_index('Month', inplace=True)

plt.figure(figsize=(12, 6))
plt.plot(data['Sales'])
plt.title('Serie de Tiempo de Ventas de Carros mensuales')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.grid(True)
plt.show()
```

```
result = seasonal_decompose(data['Sales'], model='additive')
result.plot()
plt.show()

plot_acf(data['Sales'], lags=40)
plt.title('Función de Autocorrelación')
plt.show()

plot_pacf(data['Sales'], lags=40)
plt.title('Función de Autocorrelación Parcial')
plt.show()
```




Sales
20000
10000
Trend
15000
10000
Seasonal
5000
0
−5000
Resid
2500
0
−2500
1960
1961
1962
1963
1964
1965
1966
1967
1968


Función de Autocorrelación
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40




```
sarima_model = sm.tsa.SARIMAX(data['Sales'], order=(1, 0, 1),
        seasonal_order=(1, 0, 1, 365))
sarima_results = sarima_model.fit()
print(sarima_results.summary())
```

```
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.3
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency MS will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.3
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency MS will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.3
packages\Python310\site-
packages\statsmodels\tsa\statespace\sarimax.py:866: UserWarning: Too
few observations to estimate starting parameters for seasonal ARMA.
```

```
All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'


                                      SARIMAX Results
==============================================================================
Dep. Variable:                                Sales   No. Observations:
108
Model:             SARIMAX(1, 0, 1)x(1, 0, 1, 365)   Log Likelihood
-1027.598
Date:                              Mon, 04 Sep 2023   AIC
2065.197
Time:                                      12:52:02   BIC
2078.608
Sample:                                  01-01-1960   HQIC
2070.634
                                       - 12-01-1968
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
--------
ar.L1          0.9650      0.026     36.714      0.000       0.913
1.016
ma.L1          0.1800      0.119      1.517      0.129      -0.053
0.413
ar.S.L365   4.882e-05   8.92e+04   5.47e-10      1.000   -1.75e+05
1.75e+05
ma.S.L365   1.356e-05   8.89e+04   1.52e-10      1.000   -1.74e+05
1.74e+05
sigma2       1.06e+07     24.256   4.37e+05      0.000    1.06e+07
1.06e+07
==============================================================================
Ljung-Box (L1) (Q):                    0.01   Jarque-Bera (JB):
4.41
Prob(Q):                               0.93   Prob(JB):
0.11
Heteroskedasticity (H):                2.01   Skew:
0.49
Prob(H) (two-sided):                   0.04   Kurtosis:
```

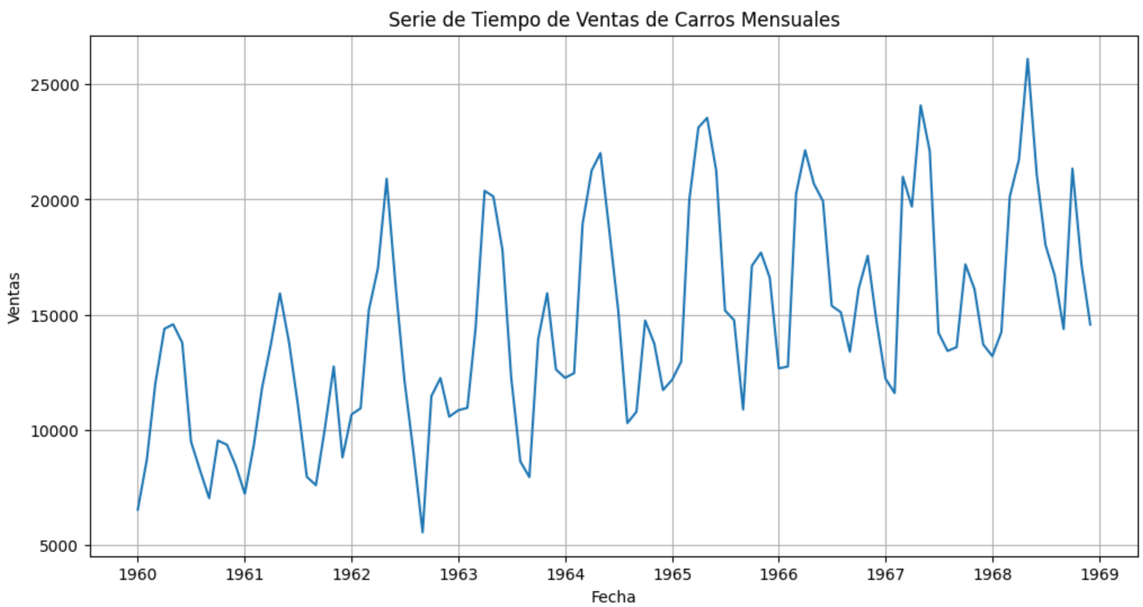
```
3.12
==============================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition
number 1.07e+24. Standard errors may be unstable.
```

4.) Alisamiento Exponencial:

- Aplicar diferentes métodos de alisamiento exponencial y comparar.

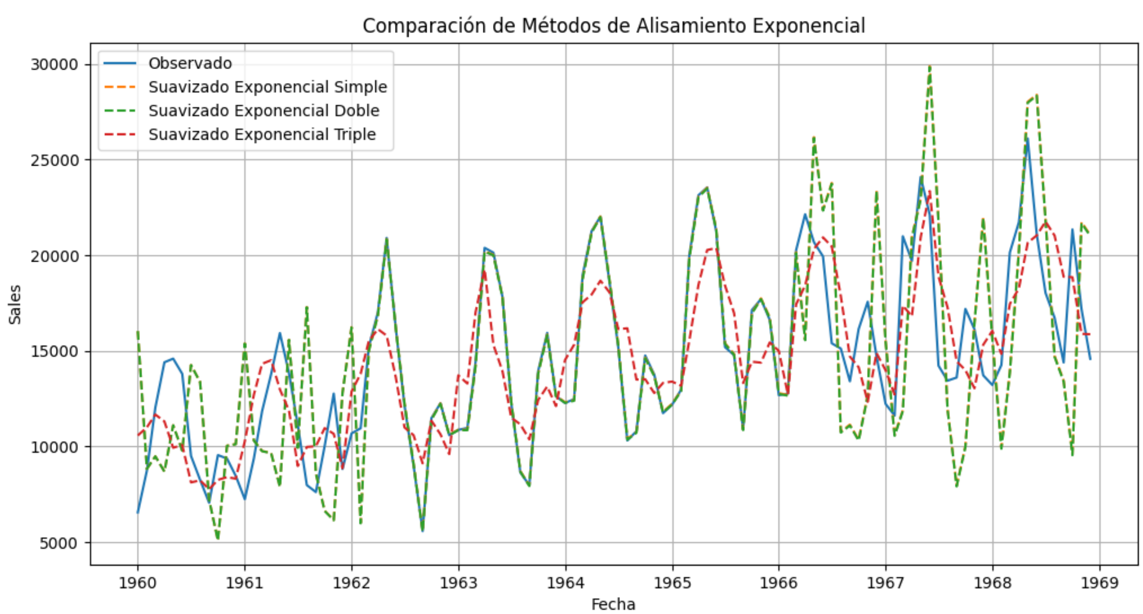
```
data = pd.read_csv("./Datos/monthly-car-sales.csv")
data['Month'] = pd.to_datetime(data['Month'])
data.set_index('Month', inplace=True)
plt.figure(figsize=(12, 6))
plt.plot(data['Sales'])
plt.title('Serie de Tiempo de Ventas de Carros Mensuales')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.grid(True)
plt.show()

exp_smoothing_simple = sm.tsa.ExponentialSmoothing(data['Sales'],
        trend='add', seasonal='add', seasonal_periods=50)
exp_smoothing_simple_fit = exp_smoothing_simple.fit()

exp_smoothing_double = sm.tsa.ExponentialSmoothing(data['Sales'],
        trend='add', seasonal='add', seasonal_periods=50,
        damped=True)
exp_smoothing_double_fit = exp_smoothing_double.fit()

exp_smoothing_triple = sm.tsa.ExponentialSmoothing(data['Sales'],
        trend='add', seasonal='add', seasonal_periods=50,
        damped=True, use_boxcox=True)
exp_smoothing_triple_fit = exp_smoothing_triple.fit()

plt.figure(figsize=(12, 6))
plt.plot(data['Sales'], label='Observado')
plt.plot(exp_smoothing_simple_fit.fittedvalues, label='Suavizado
        Exponencial Simple', linestyle='--')
```

```
plt.plot(exp_smoothing_double_fit.fittedvalues, label='Suavizado
        Exponencial Doble', linestyle='--')
plt.plot(exp_smoothing_triple_fit.fittedvalues, label='Suavizado
        Exponencial Triple', linestyle='--')
plt.title('Comparación de Métodos de Alisamiento Exponencial')
plt.xlabel('Fecha')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.show()
```




```
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.1
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency MS will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.1
packages\Python310\site-
packages\statsmodels\tsa\holtwinters\model.py:917: ConvergenceWarning:
Optimization failed to converge. Check mle_retvals.
  warnings.warn(
C:\Users\charl\AppData\Local\Temp\ipykernel_16444\3837863263.py:15:
FutureWarning: the 'damped' keyword is deprecated, use 'damped_trend'
instead.
  exp_smoothing_double = sm.tsa.ExponentialSmoothing(data['Sales'],
trend='add', seasonal='add', seasonal_periods=50, damped=True)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.1
```

```
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency MS will be
used.
  self._init_dates(dates, freq)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.
packages\Python310\site-
packages\statsmodels\tsa\holtwinters\model.py:917: ConvergenceWarning:
Optimization failed to converge. Check mle_retvals.
  warnings.warn(
C:\Users\charl\AppData\Local\Temp\ipykernel_16444\3837863263.py:18:
FutureWarning: the 'damped' keyword is deprecated, use 'damped_trend'
instead.
  exp_smoothing_triple = sm.tsa.ExponentialSmoothing(data['Sales'],
trend='add', seasonal='add', seasonal_periods=50, damped=True,
use_boxcox=True)
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.
packages\Python310\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No
frequency information was provided, so inferred frequency MS will be
used.
  self._init_dates(dates, freq)
```

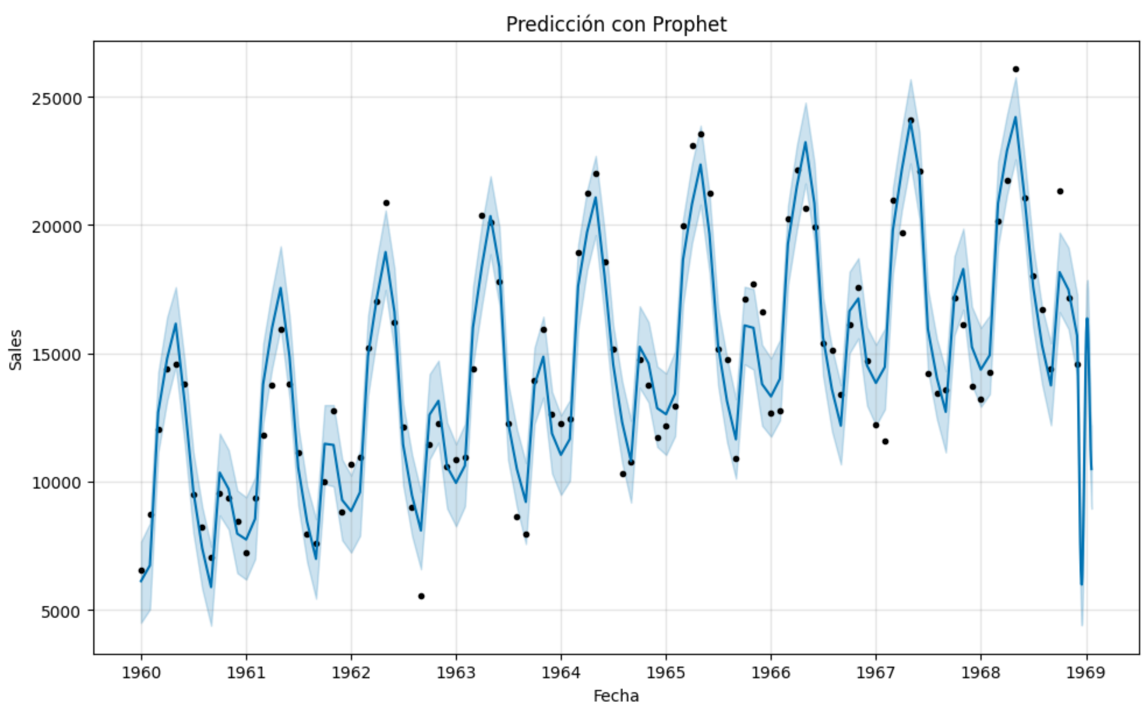


5.) Prophet:

- Utilizar Prophet para modelar la serie de tiempo.

```
data = pd.read_csv("./Datos/monthly-car-sales.csv")
data['Month'] = pd.to_datetime(data['Month'])
data = data.rename(columns={'Month': 'ds', 'Sales': 'y'})
model = Prophet()
model.fit(data)
future = model.make_future_dataframe(periods=50)
forecast = model.predict(future)
fig = model.plot(forecast)
plt.title('Predicción con Prophet')
plt.xlabel('Fecha')
plt.ylabel('Sales')
plt.grid(True)
plt.show()
```

```
12:56:01 - cmdstanpy - INFO - Chain [1] start processing
12:56:01 - cmdstanpy - INFO - Chain [1] done processing
```



6.) Redes Neuronales:

- Implementar una red neuronal simple para prever la serie de tiempo.

```
data = pd.read_csv("./Datos/monthly-car-sales.csv")
data['Month'] = pd.to_datetime(data['Month'])
data.set_index('Month', inplace=True)
```

```
scaler = MinMaxScaler()
data['Scaled_Sales'] = scaler.fit_transform(data[['Sales']])
train_size = int(len(data) * 0.80)
train_data = data.iloc[:train_size]['Scaled_Sales'].values
test_data = data.iloc[train_size:]['Scaled_Sales'].values
look_back = 7
X_train, y_train = create_sequences(train_data, look_back)
X_test, y_test = create_sequences(test_data, look_back)
model = keras.Sequential([
    keras.layers.Dense(16, activation='relu', input_shape=
        (look_back,)),
    keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=16, verbose=1)
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
train_predictions = scaler.inverse_transform(train_predictions)
test_predictions = scaler.inverse_transform(test_predictions)
rmse = sqrt(mean_squared_error(data.iloc[(train_size + look_back):]
        ['Sales'], test_predictions))
print(f'Error Cuadrático Medio en el Conjunto de Prueba: {rmse}')
plt.figure(figsize=(12, 6))
plt.plot(data.index[(train_size + look_back):], data.iloc[(train_size
        + look_back):]['Sales'], label='Observado')
plt.plot(data.index[(train_size + look_back):], test_predictions,
        label='Predicción', linestyle='--')
plt.title('Predicciones de la Serie de Tiempo con Red Neuronal')
plt.xlabel('Fecha')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch 1/100
5/5 [==============================] - 0s 2ms/step - loss: 1.3668
Epoch 2/100
5/5 [==============================] - 0s 2ms/step - loss: 1.2255
Epoch 3/100
5/5 [==============================] - 0s 2ms/step - loss: 1.0896
Epoch 4/100
5/5 [==============================] - 0s 3ms/step - loss: 0.9644
```

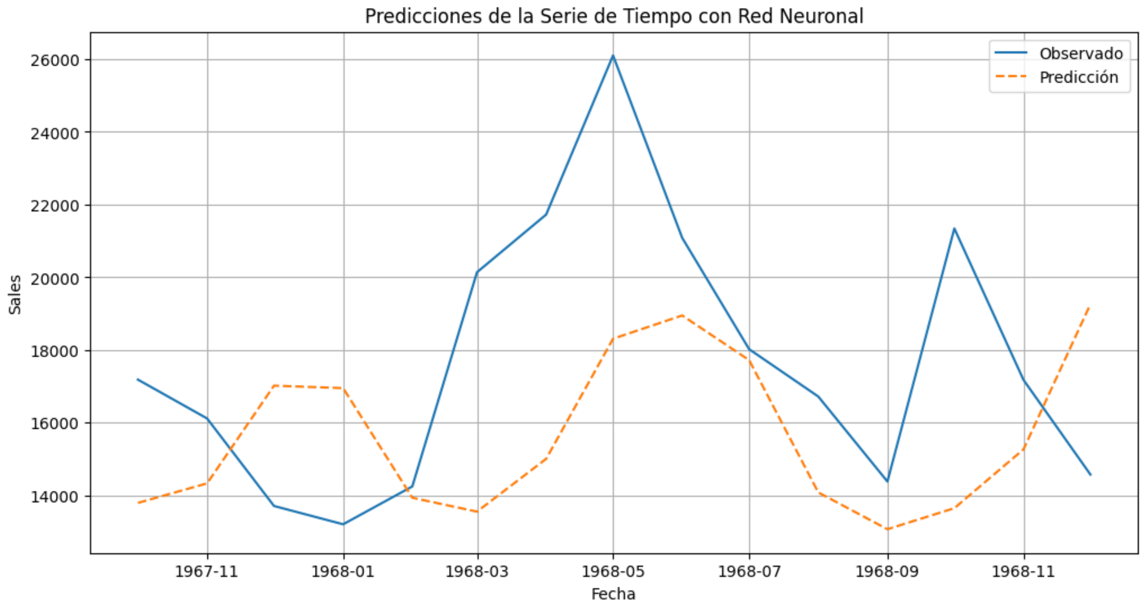
```
Epoch 5/100
5/5 [==============================] - 0s 3ms/step - loss: 0.8503
Epoch 6/100
5/5 [==============================] - 0s 3ms/step - loss: 0.7482
Epoch 7/100
5/5 [==============================] - 0s 3ms/step - loss: 0.6492
Epoch 8/100
5/5 [==============================] - 0s 3ms/step - loss: 0.5596
Epoch 9/100
5/5 [==============================] - 0s 2ms/step - loss: 0.4850
Epoch 10/100
5/5 [==============================] - 0s 3ms/step - loss: 0.4117
Epoch 11/100
5/5 [==============================] - 0s 3ms/step - loss: 0.3485
Epoch 12/100
5/5 [==============================] - 0s 3ms/step - loss: 0.2922
Epoch 13/100
5/5 [==============================] - 0s 2ms/step - loss: 0.2418
Epoch 14/100
5/5 [==============================] - 0s 2ms/step - loss: 0.1996
Epoch 15/100
5/5 [==============================] - 0s 3ms/step - loss: 0.1643
Epoch 16/100
5/5 [==============================] - 0s 2ms/step - loss: 0.1342
Epoch 17/100
5/5 [==============================] - 0s 3ms/step - loss: 0.1113
Epoch 18/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0928
Epoch 19/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0790
Epoch 20/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0698
Epoch 21/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0633
Epoch 22/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0586
Epoch 23/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0559
Epoch 24/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0541
```

```
Epoch 25/100
5/5 [==============================] - 0s 5ms/step - loss: 0.0531
Epoch 26/100
5/5 [==============================] - 0s 8ms/step - loss: 0.0527
Epoch 27/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0523
Epoch 28/100
5/5 [==============================] - 0s 4ms/step - loss: 0.0520
Epoch 29/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0517
Epoch 30/100
5/5 [==============================] - 0s 4ms/step - loss: 0.0515
Epoch 31/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0513
Epoch 32/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0510
Epoch 33/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0507
Epoch 34/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0505
Epoch 35/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0501
Epoch 36/100
5/5 [==============================] - 0s 6ms/step - loss: 0.0499
Epoch 37/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0496
Epoch 38/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0493
Epoch 39/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0491
Epoch 40/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0488
Epoch 41/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0485
Epoch 42/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0483
Epoch 43/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0480
Epoch 44/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0477
```

```
Epoch 45/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0475
Epoch 46/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0472
Epoch 47/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0469
Epoch 48/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0466
Epoch 49/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0464
Epoch 50/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0461
Epoch 51/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0458
Epoch 52/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0456
Epoch 53/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0453
Epoch 54/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0451
Epoch 55/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0448
Epoch 56/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0446
Epoch 57/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0443
Epoch 58/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0440
Epoch 59/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0438
Epoch 60/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0435
Epoch 61/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0433
Epoch 62/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0430
Epoch 63/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0428
Epoch 64/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0426
```

```
Epoch 65/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0423
Epoch 66/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0420
Epoch 67/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0418
Epoch 68/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0415
Epoch 69/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0414
Epoch 70/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0411
Epoch 71/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0408
Epoch 72/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0405
Epoch 73/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0403
Epoch 74/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0400
Epoch 75/100
5/5 [==============================] - 0s 3ms/step - loss: 0.0398
Epoch 76/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0395
Epoch 77/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0393
Epoch 78/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0390
Epoch 79/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0388
Epoch 80/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0387
Epoch 81/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0383
Epoch 82/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0381
Epoch 83/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0379
Epoch 84/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0376
```

```
Epoch 85/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0374
Epoch 86/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0372
Epoch 87/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0369
Epoch 88/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0367
Epoch 89/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0364
Epoch 90/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0364
Epoch 91/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0360
Epoch 92/100
5/5 [==============================] - 0s 1ms/step - loss: 0.0358
Epoch 93/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0356
Epoch 94/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0354
Epoch 95/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0352
Epoch 96/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0350
Epoch 97/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0348
Epoch 98/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0346
Epoch 99/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0344
Epoch 100/100
5/5 [==============================] - 0s 2ms/step - loss: 0.0341
3/3 [==============================] - 0s 1ms/step
1/1 [==============================] - 0s 17ms/step
Error Cuadrático Medio en el Conjunto de Prueba: 4372.794954367631
```



7.) Comparación y Evaluación:

- Usar métricas como RMSE, MAE para comparar los modelos.

```
""" Modelo SARIMA """


from statsmodels.tools.eval_measures import rmse, meanabs
sarima_predictions = sarima_results.get_prediction(start=0,
        end=len(data)-1)
sarima_forecast = sarima_predictions.predicted_mean
sarima_rmse = rmse(data['Sales'], sarima_forecast)
sarima_mae = meanabs(data['Sales'], sarima_forecast)


print(f'RMSE del modelo SARIMA: {sarima_rmse:.2f}')
print(f'MAE del modelo SARIMA: {sarima_mae:.2f}')
```

```
RMSE del modelo SARIMA: 3293.10
MAE del modelo SARIMA: 2531.37
```

```
""" Red neuronal Simple """
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
nn_rmse = sqrt(mean_squared_error(data.iloc[(train_size + look_back):]
        ['Sales'], test_predictions))
nn_mae = mean_absolute_error(data.iloc[(train_size + look_back):]
        ['Sales'], test_predictions)


print(f'RMSE de la Red Neuronal Simple: {nn_rmse:.2f}')
print(f'MAE de la Red Neuronal Simple: {nn_mae:.2f}')
```

```
RMSE de la Red Neuronal Simple: 4372.79
MAE de la Red Neuronal Simple: 3619.08
```

```
""" Prophet """

data = pd.read_csv("./Datos/monthly-car-sales.csv")
data['Month'] = pd.to_datetime(data['Month'])
data = data.rename(columns={'Month': 'ds', 'Sales': 'y'})
train_data = data.iloc[:-65]
test_data = data.iloc[-65:]
model = Prophet()
model.fit(train_data)
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)
predicted_values = forecast.iloc[-65:]['yhat']
rmse = np.sqrt(mean_squared_error(test_data['y'], predicted_values))
mae = mean_absolute_error(test_data['y'], predicted_values)

print(f"RMSE de Prophet: {rmse:.2f}")
print(f"MAE de Prophet: {mae:.2f}")
```

```
13:07:53 - cmdstanpy - INFO - Chain [1] start processing
13:07:53 - cmdstanpy - INFO - Chain [1] done processing
```

```
RMSE de Prophet: 11377.83
MAE de Prophet: 9572.29
```

- Discutir cuál algoritmo se desempeña mejor para cada tipo de conjunto de datos y por qué.

Comparando estos valores, el modelo SARIMA parece ser el mejor de los tres, ya que tiene los valores más bajos tanto de RMSE como de MAE. Esto indica que el modelo SARIMA se ajusta mejor a los datos de ventas de carros mensuales en comparación con los otros dos modelos

Conjunto de Datos 3: monthly-mean-temp.csv

1. Análisis Exploratorio:

Describir la serie de tiempo y visualizarla.

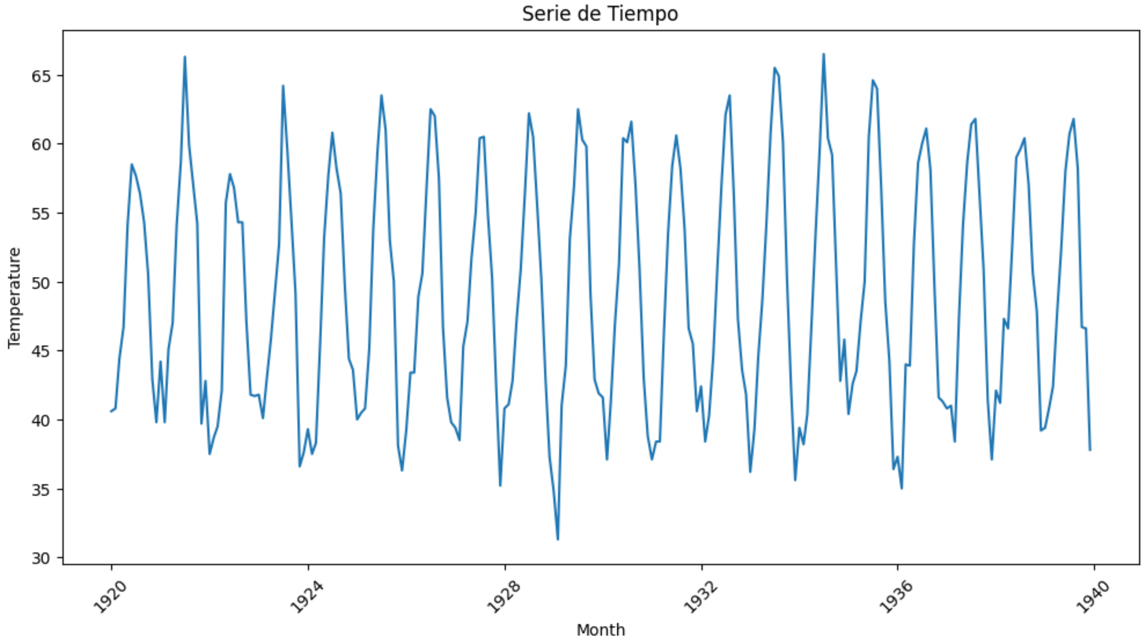
```
df = pd.read_csv("./Datos/monthly-mean-temp.csv")
df['Month'] = pd.to_datetime(df['Month'])
df.head()
```

	Month	Temperature
0	1920-01-01	40.6
1	1920-02-01	40.8
2	1920-03-01	44.4
3	1920-04-01	46.7
4	1920-05-01	54.1

```
df.describe()
```

	Month	Temperature
count	240	240.000000
mean	1929-12-15 23:00:00	49.041250
min	1920-01-01 00:00:00	31.300000
25%	1924-12-24 06:00:00	41.550000
50%	1929-12-16 12:00:00	47.350000
75%	1934-12-08 18:00:00	57.000000
max	1939-12-01 00:00:00	66.500000
std	NaN	8.569705

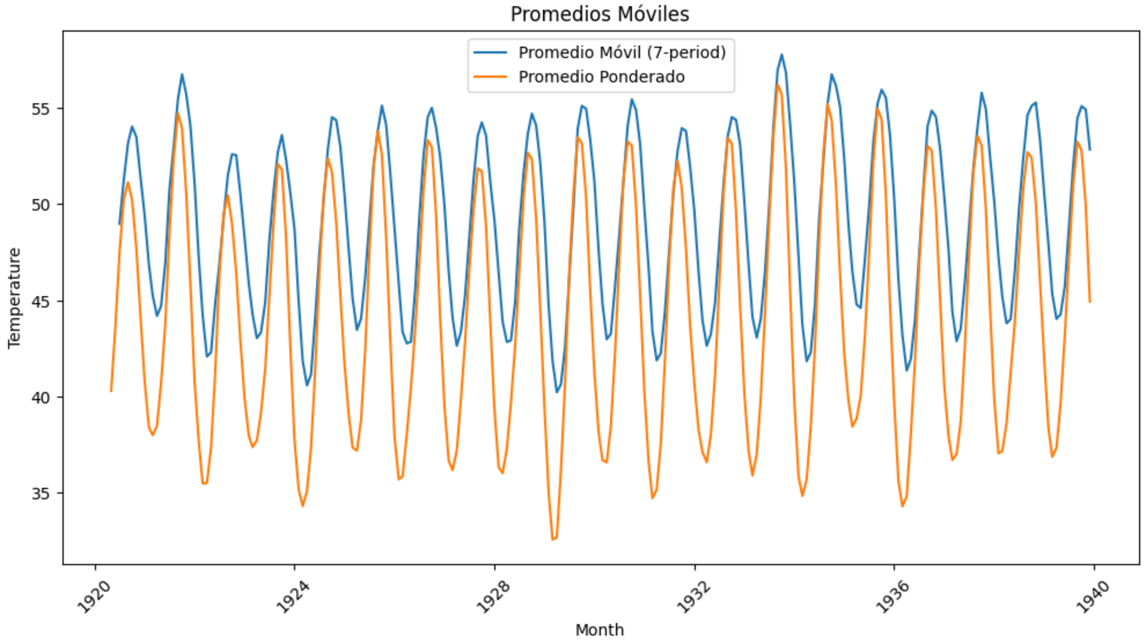
```
plt.figure(figsize=(12, 6))
plt.plot(df['Month'], df['Temperature'])
plt.title("Serie de Tiempo")
plt.xlabel("Month")
plt.ylabel("Temperature")
plt.xticks(rotation=45)
plt.show()
```



2. Promedios:

• Aplicar métodos de promedios y comparar los resultados con el conjunto original.

```
window_size = 7
df['promedio_movil'] =
        df['Temperature'].rolling(window=window_size).mean()
plt.figure(figsize=(12, 6))
plt.plot(df['Month'], df['promedio_movil'], label=f'Promedio Móvil
        ({window_size}-period)')


weights = [0.1, 0.2, 0.3, 0.2, 0.1]
df['promedio_ponderado'] =
        df['Temperature'].rolling(window=len(weights)).apply(lambda
        x: (x * weights).sum())
plt.plot(df['Month'], df['promedio_ponderado'], label='Promedio
        Ponderado')


plt.legend()
plt.title("Promedios Móviles")
plt.xlabel("Month")
plt.ylabel("Temperature")
plt.xticks(rotation=45)
plt.show()
```



3. SARIMA:

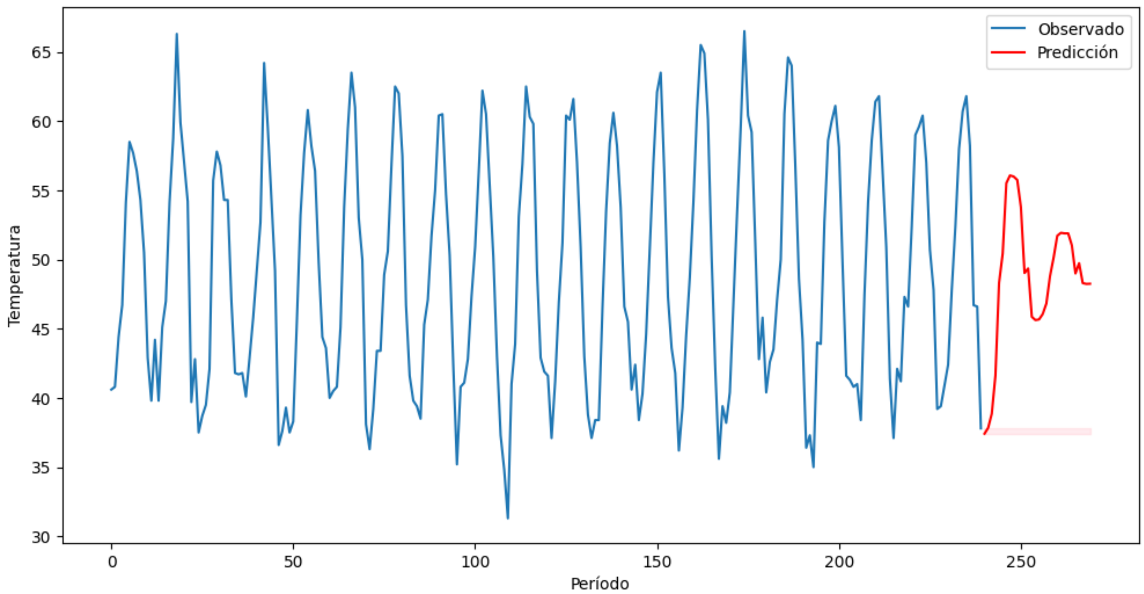
- Identificar parámetros y ajustar un modelo SARIMA.

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

p, d, q = 1, 1, 1
P, D, Q, S = 1, 1, 1, 7

model = sm.tsa.SARIMAX(df['Temperature'], order=(p, d, q),
         seasonal_order=(P, D, Q, S))
results = model.fit()


print(results.summary())

forecast_period = 30
forecast = results.get_forecast(steps=forecast_period)
forecast_mean = forecast.predicted_mean

date_integer_index = range(len(df), len(df) + forecast_period)
forecast_mean.index = date_integer_index
forecast.conf_int().index = date_integer_index

plt.figure(figsize=(12, 6))
```

```
plt.plot(range(len(df)), df['Temperature'], label='Observado')
plt.plot(date_integer_index, forecast_mean.values, label='Predicción',
        color='red')
plt.fill_between(date_integer_index, forecast_mean.values[0],
        forecast_mean.values[1], color='pink', alpha=0.3)
plt.xlabel('Período')
plt.ylabel('Temperatura')
plt.legend()
plt.show()
```

```
                                     SARIMAX Results
=======================================================================================
Dep. Variable:                       Temperature   No. Observations:
240
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 7)   Log Likelihood
-677.050
Date:                           Mon, 04 Sep 2023   AIC
1364.099
Time:                                   16:36:08   BIC
1381.333
Sample:                                        0   HQIC
1371.049
                                           - 240
Covariance Type:                             opg
=======================================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
-----------------------------------------------------------------------
--------
ar.L1          0.7099      0.090      7.868      0.000       0.533
0.887
ma.L1         -0.9999      8.951     -0.112      0.911     -18.543
16.544
ar.S.L7       -0.5938      0.091     -6.538      0.000      -0.772
-0.416
ma.S.L7       -0.9999     18.259     -0.055      0.956     -36.787
34.787
sigma2        16.6649    346.975      0.048      0.962    -663.393
696.723
=======================================================================================
Ljung-Box (L1) (Q):                   7.66   Jarque-Bera (JB):
5.64
```

```
Prob(Q):                              0.01   Prob(JB):
0.06
Heteroskedasticity (H):               0.88   Skew:
-0.38
Prob(H) (two-sided):                  0.58   Kurtosis:
2.98
======================================================================

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

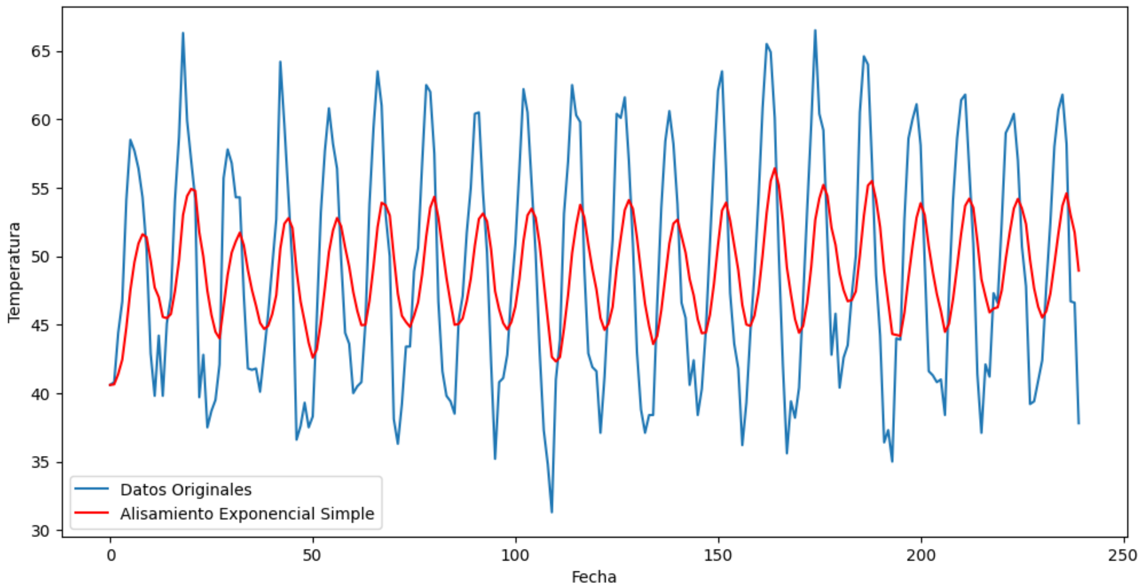


4. Alisamiento Exponencial:

- Aplicar diferentes métodos de alisamiento exponencial y comparar.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import statsmodels.api as sm

def exponential_smoothing(series, alpha):
    result = [series[0]]
    for t in range(1, len(series)):
        result.append(alpha * series[t] + (1 - alpha) * result[t - 1])
    return result
```

```
alpha = 0.2

smoothed_temperature = exponential_smoothing(df['Temperature'], alpha)

plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Temperature'], label='Datos Originales')
plt.plot(df.index, smoothed_temperature, label='Alisamiento
        Exponencial Simple', color='red')
plt.xlabel('Fecha')
plt.ylabel('Temperatura')
plt.legend()
plt.show()
```



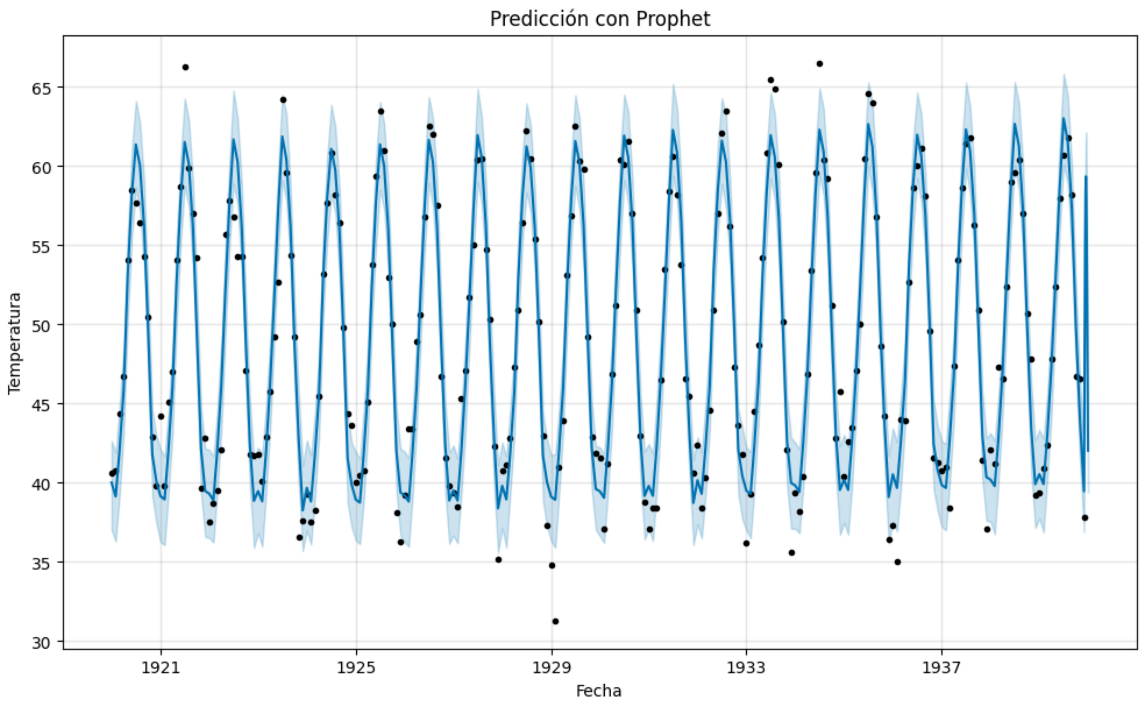
5. Prophet:

- Utilizar Prophet para modelar la serie de tiempo.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
from prophet import Prophet

data = df.rename(columns={'Month': 'ds', 'Temperature': 'y'})

model = Prophet()
model.fit(data)
```

```
future = model.make_future_dataframe(periods=30)
forecast = model.predict(future)

fig = model.plot(forecast)
plt.xlabel('Fecha')
plt.ylabel('Temperatura')
plt.title('Predicción con Prophet')
plt.show()
```

```
16:37:04 - cmdstanpy - INFO - Chain [1] start processing
16:37:05 - cmdstanpy - INFO - Chain [1] done processing
```



6. Redes Neuronales:

- Implementar una red neuronal simple para prever la serie de tiempo.

```
scaler = MinMaxScaler()
df['Temperature'] =
        scaler.fit_transform(df['Temperature'].values.reshape(-1, 1))

sequence_length = 10
sequences = []
target = []
```

```
for i in range(len(df) - sequence_length):
    seq = df['Temperature'].values[i:i+sequence_length]
    label = df['Temperature'].values[i+sequence_length]
    sequences.append(seq)
    target.append(label)

sequences = np.array(sequences)
target = np.array(target)

train_size = int(0.8 * len(sequences))
X_train, X_test = sequences[:train_size], sequences[train_size:]
y_train, y_test = target[:train_size], target[train_size:]

model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(50, activation='relu', input_shape=
        (sequence_length, 1)))
model.add(tf.keras.layers.Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=100, batch_size=32)

loss = model.evaluate(X_test, y_test)
print(f'Pérdida en datos de prueba: {loss}')

predictions = model.predict(X_test)

predictions = scaler.inverse_transform(predictions)

plt.figure(figsize=(12, 6))
plt.plot(df.index[train_size+sequence_length:], y_test,
        label='Observado', color='blue')
plt.plot(df.index[train_size+sequence_length:], predictions,
        label='Predicción', color='red')
plt.xlabel('Fecha')
plt.ylabel('Temperature')
plt.title('Predicción con Red Neuronal')
plt.legend()
plt.show()
```

```
Epoch 1/100
6/6 [==============================] - 1s 4ms/step - loss: 0.2540
Epoch 2/100
```

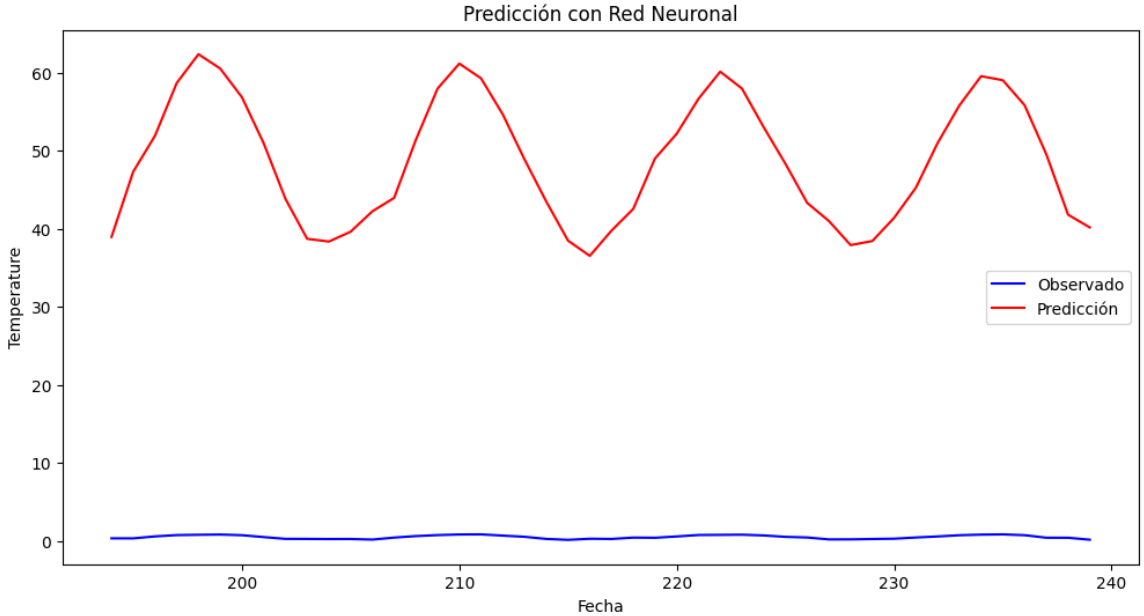
```
6/6 [==============================] - 0s 3ms/step - loss: 0.1941
Epoch 3/100
6/6 [==============================] - 0s 4ms/step - loss: 0.1418
Epoch 4/100
6/6 [==============================] - 0s 4ms/step - loss: 0.1014
Epoch 5/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0813
Epoch 6/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0806
Epoch 7/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0753
Epoch 8/100
6/6 [==============================] - 0s 10ms/step - loss: 0.0723
Epoch 9/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0713
Epoch 10/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0689
Epoch 11/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0666
Epoch 12/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0646
Epoch 13/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0624
Epoch 14/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0597
Epoch 15/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0564
Epoch 16/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0529
Epoch 17/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0493
Epoch 18/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0448
Epoch 19/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0396
Epoch 20/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0339
Epoch 21/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0276
Epoch 22/100
```

```
6/6 [==============================] - 0s 3ms/step - loss: 0.0206
Epoch 23/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0151
Epoch 24/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0092
Epoch 25/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0077
Epoch 26/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0085
Epoch 27/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0077
Epoch 28/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0074
Epoch 29/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0071
Epoch 30/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0072
Epoch 31/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0070
Epoch 32/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0078
Epoch 33/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0071
Epoch 34/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0070
Epoch 35/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0071
Epoch 36/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0068
Epoch 37/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0077
Epoch 38/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0082
Epoch 39/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0074
Epoch 40/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0072
Epoch 41/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0070
Epoch 42/100
```

```
6/6 [==============================] - 0s 3ms/step - loss: 0.0067
Epoch 43/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0066
Epoch 44/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0069
Epoch 45/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0070
Epoch 46/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0072
Epoch 47/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0066
Epoch 48/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0064
Epoch 49/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0065
Epoch 50/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0064
Epoch 51/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0074
Epoch 52/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0071
Epoch 53/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0064
Epoch 54/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0063
Epoch 55/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0065
Epoch 56/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0063
Epoch 57/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0070
Epoch 58/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0063
Epoch 59/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0061
Epoch 60/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0062
Epoch 61/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0062
Epoch 62/100
```

```
6/6 [==============================] - 0s 3ms/step - loss: 0.0063
Epoch 63/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0062
Epoch 64/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0062
Epoch 65/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0061
Epoch 66/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0064
Epoch 67/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0067
Epoch 68/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0065
Epoch 69/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0060
Epoch 70/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0063
Epoch 71/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0061
Epoch 72/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0067
Epoch 73/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0066
Epoch 74/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0065
Epoch 75/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0061
Epoch 76/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0060
Epoch 77/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0064
Epoch 78/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 79/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0061
Epoch 80/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0063
Epoch 81/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0062
Epoch 82/100
```

```
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 83/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0060
Epoch 84/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0058
Epoch 85/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0058
Epoch 86/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0064
Epoch 87/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0066
Epoch 88/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0063
Epoch 89/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 90/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 91/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 92/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0058
Epoch 93/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 94/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 95/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0058
Epoch 96/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0057
Epoch 97/100
6/6 [==============================] - 0s 3ms/step - loss: 0.0059
Epoch 98/100
6/6 [==============================] - 0s 5ms/step - loss: 0.0059
Epoch 99/100
6/6 [==============================] - 0s 4ms/step - loss: 0.0060
Epoch 100/100
6/6 [==============================] - 0s 6ms/step - loss: 0.0060
2/2 [==============================] - 0s 3ms/step - loss: 0.0055
Pérdida en datos de prueba: 0.005466956179589033
2/2 [==============================] - 0s 3ms/step
```



7. Comparación y Evaluación:

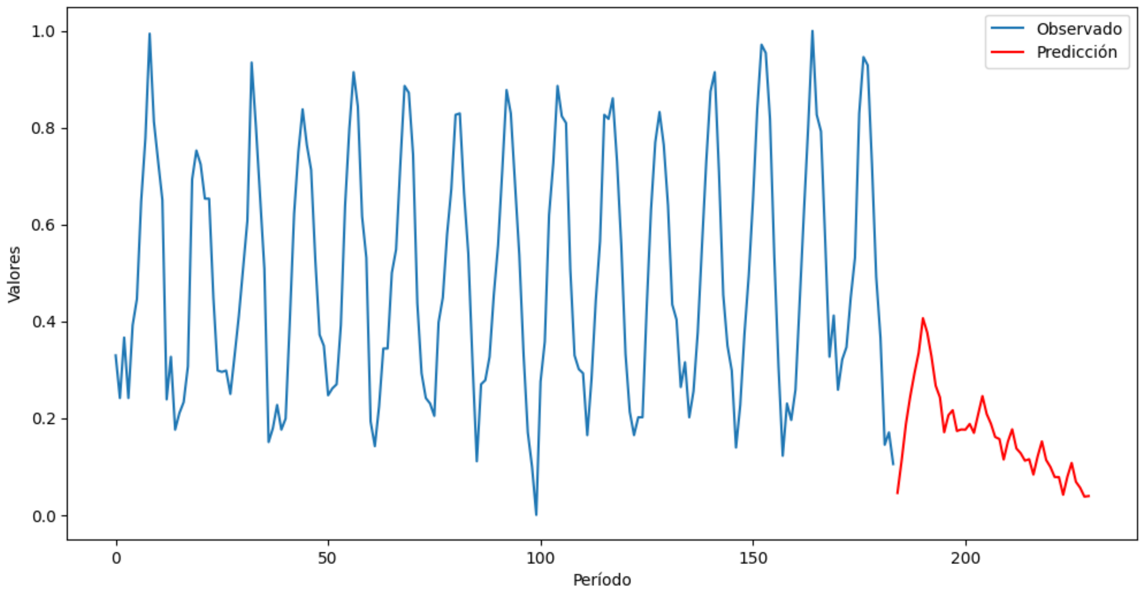
- Usar métricas como RMSE, MAE para comparar los modelos.

```
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

p, d, q = 1, 1, 1
P, D, Q, S = 1, 1, 1, 7

model = sm.tsa.SARIMAX(y_train, order=(p, d, q), seasonal_order=(P, D,
        Q, S))
results = model.fit()

print(results.summary())

forecast_period = len(X_test)
forecast = results.get_forecast(steps=forecast_period)
forecast_mean = forecast.predicted_mean

plt.figure(figsize=(12, 6))
plt.plot(range(len(y_train)), y_train, label='Observado')
plt.plot(range(len(y_train), len(y_train) + forecast_period),
        forecast_mean, label='Predicción', color='red')
plt.xlabel('Período')
plt.ylabel('Valores')
```

```
plt.legend()
plt.show()

rmse = mean_squared_error(y_test, forecast_mean, squared=False)
mae = mean_absolute_error(y_test, forecast_mean)

print("RMSE:", rmse)
print("MAE:", mae)
```

```
                                     SARIMAX Results
==========================================================================
Dep. Variable:                                  y   No. Observations:
184
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 7)   Log Likelihood
102.554
Date:                            Mon, 04 Sep 2023   AIC
-195.108
Time:                                    16:38:30   BIC
-179.255
Sample:                                         0   HQIC
-188.678
                                            - 184
Covariance Type:                              opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
--------
ar.L1          0.3894      0.285      1.368      0.171      -0.169
0.947
ma.L1         -0.1355      0.296     -0.458      0.647      -0.715
0.444
ar.S.L7       -0.4463      0.100     -4.449      0.000      -0.643
-0.250
ma.S.L7       -0.8259      0.070    -11.725      0.000      -0.964
-0.688
sigma2         0.0169      0.002      8.461      0.000       0.013
0.021
==========================================================================
Ljung-Box (L1) (Q):                   0.11   Jarque-Bera (JB):
2.77
```

```
Prob(Q):                              0.74   Prob(JB):
0.25
Heteroskedasticity (H):               0.73   Skew:
-0.25
Prob(H) (two-sided):                  0.23   Kurtosis:
2.64
======================================================================

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




```
RMSE: 0.4354819447275895
MAE: 0.367813453043807
```

- Discutir cuál algoritmo se desempeña mejor para cada tipo de conjunto de datos y por qué.

Despues de probar los diferentes tipos de algoritmos, se puede visualizar en las pruebas que Prophet seria la mejor eleccion para estos tipos de datos, gracias a su capacidad para manejar automáticamente las tendencias y patrones estacionales en los datos, lo que elimina la necesidad de ajustes manuales complicados. Además, Prophet es robusto incluso cuando los datos son irregulares o tienen valores faltantes, lo que lo convierte en una herramienta versátil para pronósticos en situaciones del mundo real. Su capacidad para considerar días festivos y eventos especiales también mejora la precisión de las predicciones en contextos complejos.

Conjunto de Datos 4: shampoo.csv

1. Análisis Exploratorio:

Describir la serie de tiempo y visualizarla.

```
df = pd.read_csv("./Datos/shampoo.csv")
df['Month'] = pd.to_datetime(df['Month'], format="%m-%y")
df.head()
```

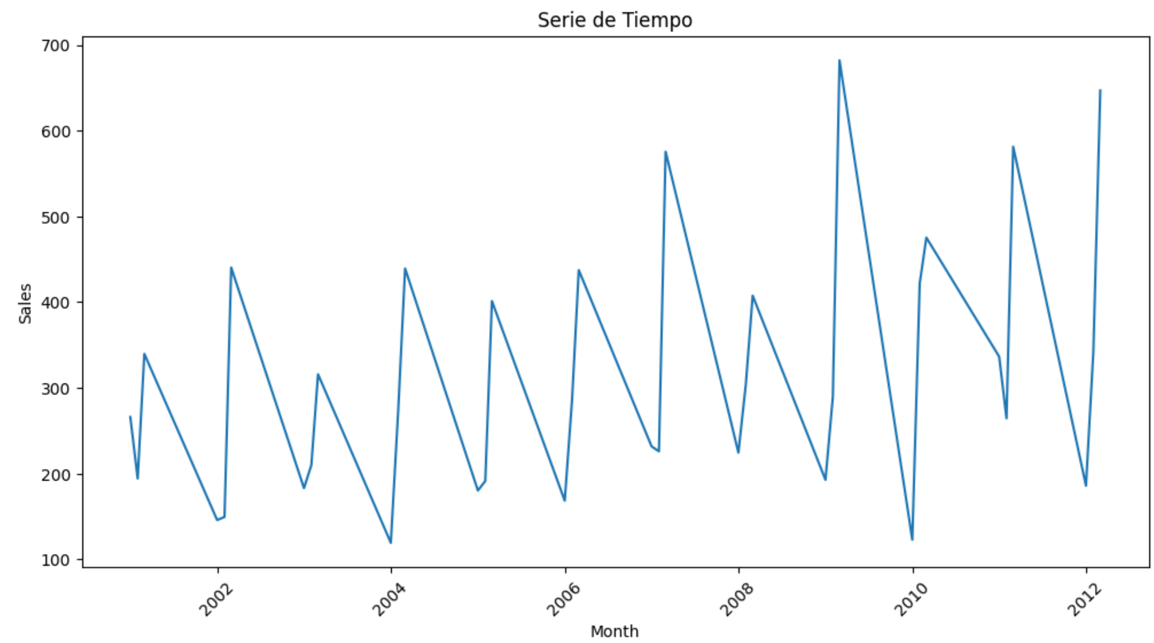
	Month	Sales
0	2001-01-01	266.0
1	2002-01-01	145.9
2	2003-01-01	183.1
3	2004-01-01	119.3
4	2005-01-01	180.3

```
df.describe()
```

	Month	Sales
count	36	36.000000
mean	2006-08-01 14:00:00	312.600000
min	2001-01-01 00:00:00	119.300000
25%	2003-10-16 12:00:00	192.450000
50%	2006-08-01 00:00:00	280.150000
75%	2009-05-16 12:00:00	411.100000
max	2012-03-01 00:00:00	682.000000
std	NaN	148.937164

```
import seaborn as sns
import statsmodels.api as sm

plt.figure(figsize=(12, 6))
sns.lineplot(x="Month", y="Sales", data=df)
plt.title("Serie de Tiempo")
```

```
plt.xlabel("Month")
plt.ylabel("Sales")
plt.xticks(rotation=45)
plt.show()
```

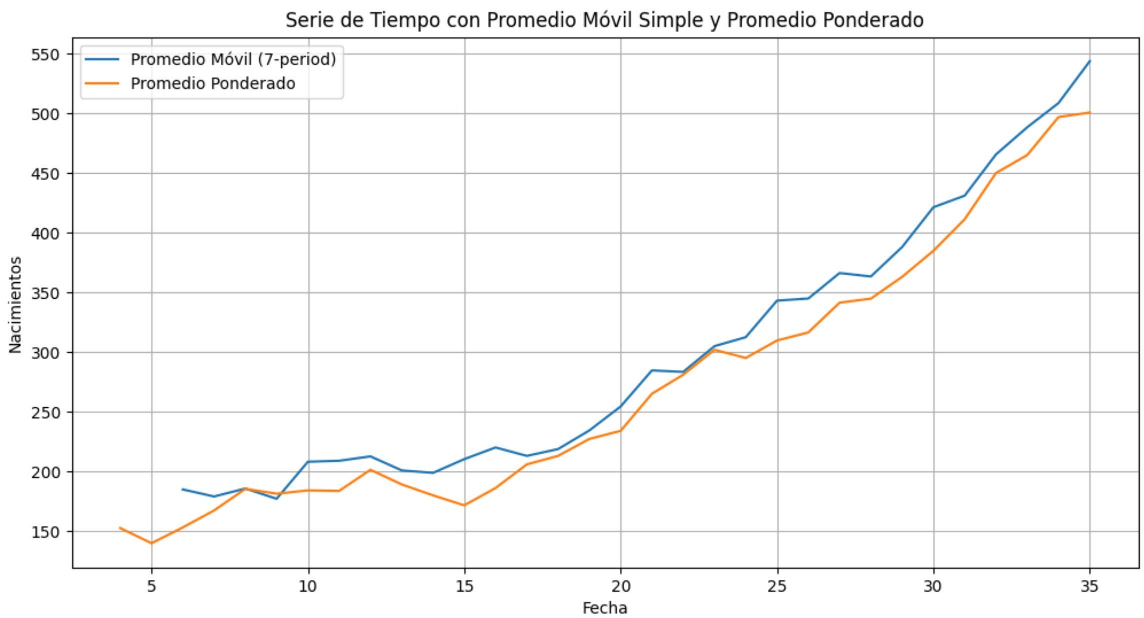


2. Promedios:

• Aplicar métodos de promedios y comparar los resultados con el conjunto original.

```
window_size = 7
df['promedio_movil'] = df['Sales'].rolling(window=window_size).mean()

weights = [0.1, 0.2, 0.3, 0.2, 0.1]
df['promedio_ponderado'] =
        df['Sales'].rolling(window=len(weights)).apply(lambda x: (x *
        weights).sum())

plt.figure(figsize=(12, 6))

sns.lineplot(x=df.index, y='promedio_movil', data=df, label=f'Promedio
        Móvil ({window_size}-period)')

sns.lineplot(x=df.index, y='promedio_ponderado', data=df,
        label='Promedio Ponderado')
```

```
plt.title('Serie de Tiempo con Promedio Móvil Simple y Promedio
        Ponderado')
plt.xlabel('Fecha')
plt.ylabel('Nacimientos')
plt.legend()
plt.grid(True)
plt.show()
```



3. SARIMA:

- Identificar parámetros y ajustar un modelo SARIMA.

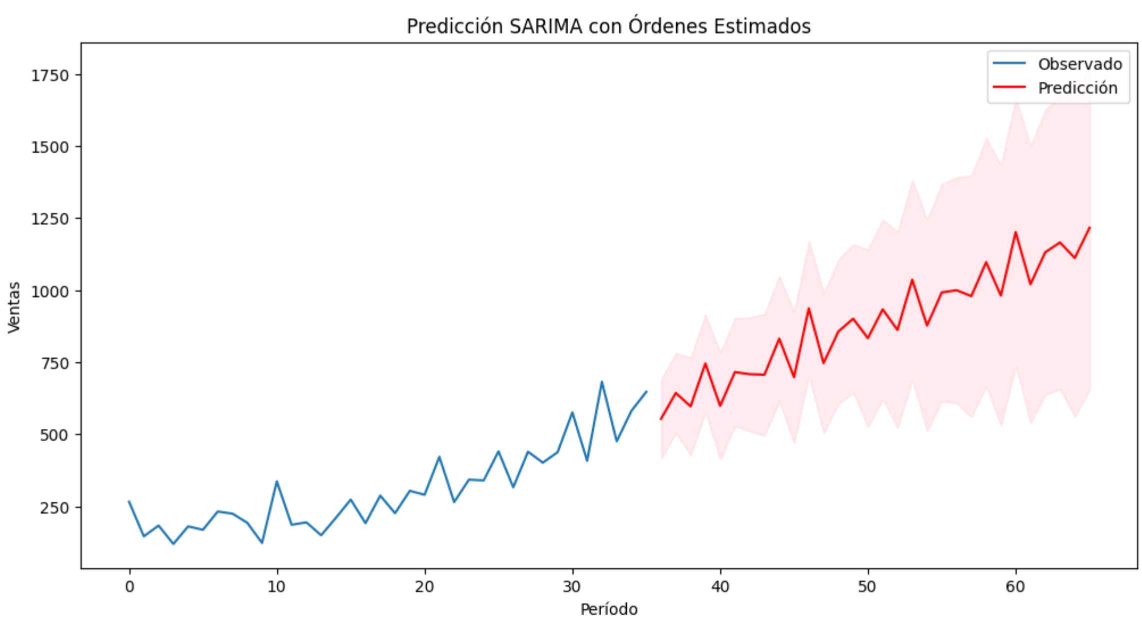
```
p, d, q = 1, 1, 1
P, D, Q, S = 1, 1, 1, 7

model = sm.tsa.SARIMAX(df['Sales'], order=(p, d, q), seasonal_order=
        (P, D, Q, S))
results = model.fit()

print(results.summary())

forecast_period = 30
forecast = results.get_forecast(steps=forecast_period)
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

date_integer_index = range(len(df), len(df) + forecast_period)
```

```
forecast_mean.index = date_integer_index
forecast_ci.index = date_integer_index

date_index = pd.date_range(start=df.index[-1], periods=forecast_period
        + 1, freq='D')[1:]


plt.figure(figsize=(12, 6))
plt.plot(range(len(df)), df['Sales'], label='Observado')
plt.plot(date_integer_index, forecast_mean.values, label='Predicción',
        color='red')
plt.fill_between(date_integer_index, forecast_ci['lower Sales'],
        forecast_ci['upper Sales'], color='pink', alpha=0.3)
plt.xlabel('Período')
plt.ylabel('Ventas')
plt.legend()
plt.title('Predicción SARIMA con Órdenes Estimados')
plt.show()
```

```
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.
packages\Python310\site-
packages\statsmodels\tsa\statespace\sarimax.py:1009: UserWarning: Non-
invertible starting seasonal moving average Using zeros as starting
parameters.
  warn('Non-invertible starting seasonal moving average'
```

```
                                     SARIMAX Results
=======================================================================
Dep. Variable:                                Sales   No. Observations:
36
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 7)   Log Likelihood
-161.866
Date:                          Mon, 04 Sep 2023   AIC
333.732
Time:                                  16:42:07   BIC
340.393
Sample:                                       0   HQIC
335.768
                                           - 36
Covariance Type:                            opg
=======================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
```

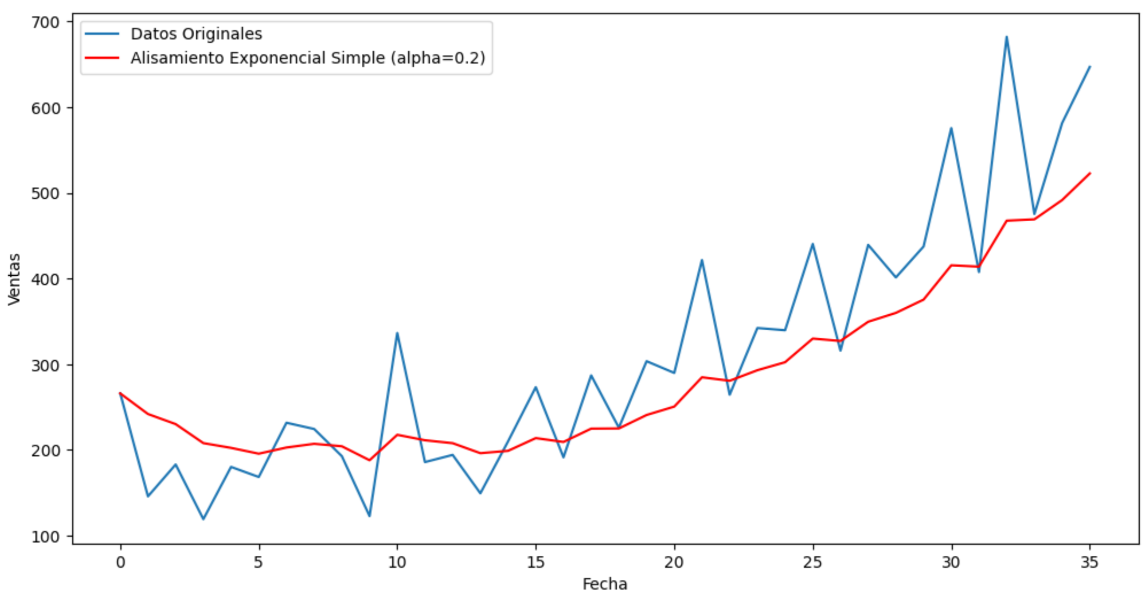
```
--------------------------------------------------------------------------------------
ar.L1         -0.7427      0.157     -4.720      0.000      -1.051      -0.434
ma.L1         -0.4036      0.216     -1.872      0.061      -0.826       0.019
ar.S.L7       -0.7150      0.280     -2.553      0.011      -1.264      -0.166
ma.S.L7        0.0056      0.425      0.013      0.990      -0.827       0.838
sigma2      4916.8272   1821.196      2.700      0.007    1347.349    8486.306
=====================================================================================
Ljung-Box (L1) (Q):                   0.47   Jarque-Bera (JB):                  0.27
Prob(Q):                              0.49   Prob(JB):                          0.87
Heteroskedasticity (H):               1.09   Skew:                              0.19
Prob(H) (two-sided):                  0.90   Kurtosis:                          2.70
=====================================================================================

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



4. Alisamiento Exponencial:

- Aplicar diferentes métodos de alisamiento exponencial y comparar.

```
alpha = 0.2

df['Exponential_Smooth'] = df['Sales'].ewm(alpha=alpha,
        adjust=False).mean()

plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Sales'], label='Datos Originales')
plt.plot(df.index, df['Exponential_Smooth'], label=f'Alisamiento
        Exponencial Simple (alpha={alpha})', color='red')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.legend()
plt.show()
```



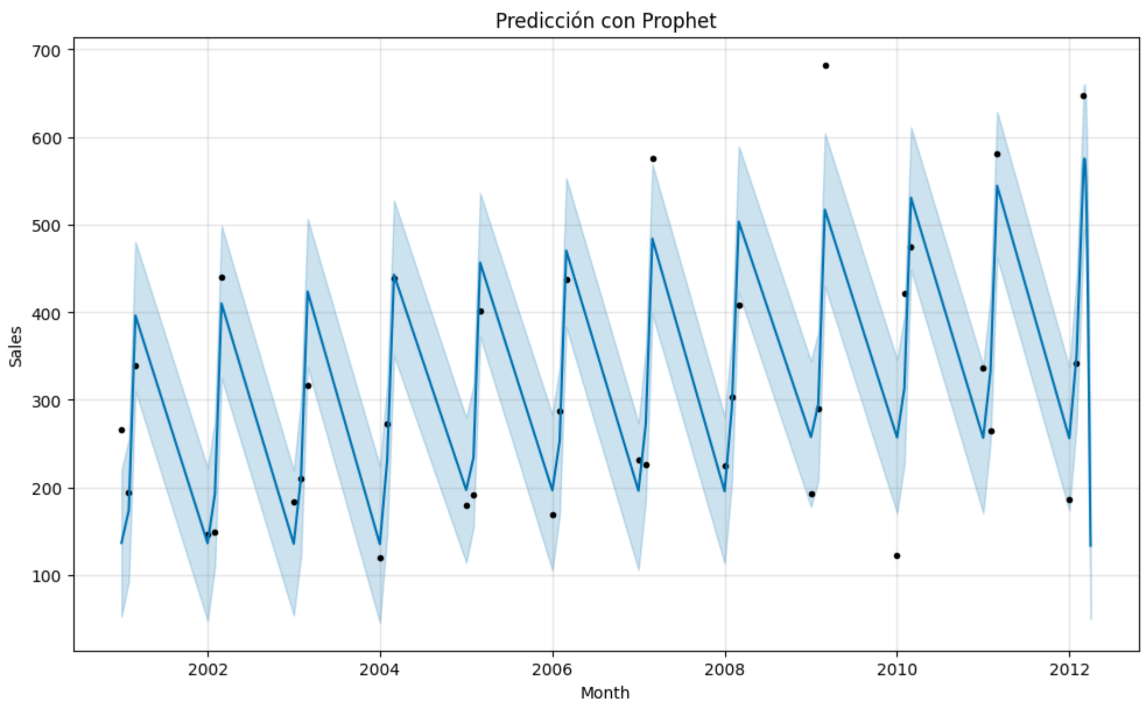
5. Prophet:

- Utilizar Prophet para modelar la serie de tiempo

```
import pandas as pd
import matplotlib.pyplot as plt
from prophet import Prophet
data = df.rename(columns={'Month': 'ds', 'Sales': 'y'})

model = Prophet()
```

```
model.fit(data)
future = model.make_future_dataframe(periods=30)
forecast = model.predict(future)
fig = model.plot(forecast)
plt.xlabel('Month')
plt.ylabel('Sales')
plt.title('Predicción con Prophet')
plt.show()
```

```
16:42:58 - cmdstanpy - INFO - Chain [1] start processing
16:42:58 - cmdstanpy - INFO - Chain [1] done processing
```



6. Redes Neuronales:

- Implementar una red neuronal simple para prever la serie de tiempo.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df['Sales'] = scaler.fit_transform(df['Sales'].values.reshape(-1, 1))
```

```
sequence_length = 10
sequences = []
target = []

for i in range(len(df) - sequence_length):
    seq = df['Sales'].values[i:i+sequence_length]
    label = df['Sales'].values[i+sequence_length]
    sequences.append(seq)
    target.append(label)

sequences = np.array(sequences)
target = np.array(target)

train_size = int(0.8 * len(sequences))
X_train, X_test = sequences[:train_size], sequences[train_size:]
y_train, y_test = target[:train_size], target[train_size:]

model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(50, activation='relu', input_shape=
        (sequence_length, 1)))
model.add(tf.keras.layers.Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=100, batch_size=32)

loss = model.evaluate(X_test, y_test)
print(f'Pérdida en datos de prueba: {loss}')

predictions = model.predict(X_test)

predictions = scaler.inverse_transform(predictions)

plt.figure(figsize=(12, 6))
plt.plot(df.index[train_size+sequence_length:], y_test,
        label='Observado', color='blue')
plt.plot(df.index[train_size+sequence_length:], predictions,
        label='Predicción', color='red')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.title('Predicción con Red Neuronal')
plt.legend()
plt.show()
```

```
Epoch 1/100
1/1 [==============================] - 1s 803ms/step - loss: 0.1095
Epoch 2/100
1/1 [==============================] - 0s 5ms/step - loss: 0.1051
Epoch 3/100
1/1 [==============================] - 0s 4ms/step - loss: 0.1006
Epoch 4/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0962
Epoch 5/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0917
Epoch 6/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0873
Epoch 7/100
1/1 [==============================] - 0s 11ms/step - loss: 0.0828
Epoch 8/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0784
Epoch 9/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0739
Epoch 10/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0695
Epoch 11/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0651
Epoch 12/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0607
Epoch 13/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0563
Epoch 14/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0521
Epoch 15/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0478
Epoch 16/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0437
Epoch 17/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0396
Epoch 18/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0356
Epoch 19/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0317
Epoch 20/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0281
```

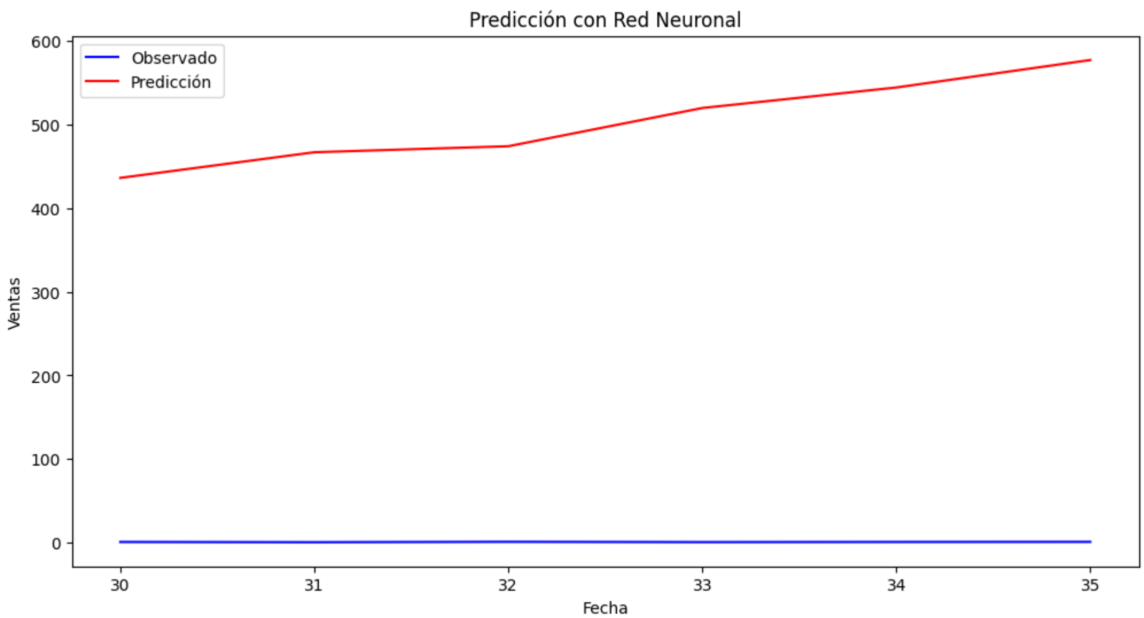
```
Epoch 21/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0247
Epoch 22/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0217
Epoch 23/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0191
Epoch 24/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0173
Epoch 25/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0161
Epoch 26/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0157
Epoch 27/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0161
Epoch 28/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0170
Epoch 29/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0180
Epoch 30/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0188
Epoch 31/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0192
Epoch 32/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0191
Epoch 33/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0186
Epoch 34/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0178
Epoch 35/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0170
Epoch 36/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0162
Epoch 37/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0155
Epoch 38/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0150
Epoch 39/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0146
Epoch 40/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0145
```

```
Epoch 41/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0144
Epoch 42/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0144
Epoch 43/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0145
Epoch 44/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0145
Epoch 45/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0146
Epoch 46/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0146
Epoch 47/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0146
Epoch 48/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0146
Epoch 49/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0145
Epoch 50/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0144
Epoch 51/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0143
Epoch 52/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0142
Epoch 53/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0140
Epoch 54/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0139
Epoch 55/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0138
Epoch 56/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0137
Epoch 57/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0136
Epoch 58/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0135
Epoch 59/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0135
Epoch 60/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0134
```

```
Epoch 61/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0134
Epoch 62/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0134
Epoch 63/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0134
Epoch 64/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0133
Epoch 65/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0133
Epoch 66/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0132
Epoch 67/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0132
Epoch 68/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0131
Epoch 69/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0131
Epoch 70/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0130
Epoch 71/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0129
Epoch 72/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0129
Epoch 73/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0128
Epoch 74/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0128
Epoch 75/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0127
Epoch 76/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0127
Epoch 77/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0127
Epoch 78/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0126
Epoch 79/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0126
Epoch 80/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0125
```

```
Epoch 81/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0125
Epoch 82/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0125
Epoch 83/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0124
Epoch 84/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0124
Epoch 85/100
1/1 [==============================] - 0s 37ms/step - loss: 0.0123
Epoch 86/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0123
Epoch 87/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0123
Epoch 88/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0122
Epoch 89/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0122
Epoch 90/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0122
Epoch 91/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0121
Epoch 92/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0121
Epoch 93/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0121
Epoch 94/100
1/1 [==============================] - 0s 3ms/step - loss: 0.0120
Epoch 95/100
1/1 [==============================] - 0s 6ms/step - loss: 0.0120
Epoch 96/100
1/1 [==============================] - 0s 3ms/step - loss: 0.0120
Epoch 97/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0119
Epoch 98/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0119
Epoch 99/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0119
Epoch 100/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0118
```

```
1/1 [==============================] - 0s 139ms/step - loss: 0.0391
Pérdida en datos de prueba: 0.03912785276770592
1/1 [==============================] - 0s 113ms/step
```



7. Comparación y Evaluación:

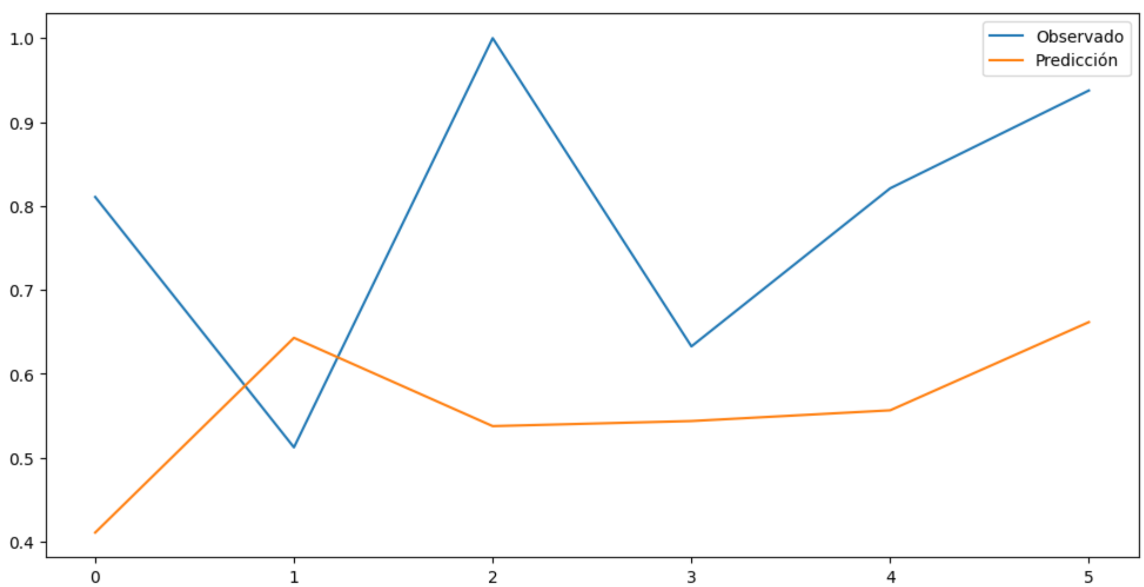
• Usar métricas como RMSE, MAE para comparar los modelos.

```
model = sm.tsa.SARIMAX(y_train, order=(1, 1, 1), seasonal_order=(1, 1,
        1, 7))
results = model.fit()

forecast = results.get_forecast(steps=len(X_test))
y_pred = forecast.predicted_mean

rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)

print("RMSE:", rmse)
print("MAE:", mae)

plt.figure(figsize=(12, 6))
plt.plot(y_test, label="Observado")
plt.plot(y_pred, label="Predicción")
```

```
plt.legend()
plt.show()
```

```
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.1
packages\Python310\site-
packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Non-
stationary starting autoregressive parameters found. Using zeros as
starting parameters.
  warn('Non-stationary starting autoregressive parameters'
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.1
packages\Python310\site-
packages\statsmodels\tsa\statespace\sarimax.py:866: UserWarning: Too
few observations to estimate starting parameters for seasonal ARMA.
All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
C:\Users\charl\AppData\Local\Packages\PythonSoftwareFoundation.Python.1
packages\Python310\site-packages\statsmodels\base\model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to
converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
RMSE: 0.3013352218796638
MAE: 0.270383269807116
```



- Discutir cuál algoritmo se desempeña mejor para cada tipo de conjunto de datos y por qué

En este caso particular, SARIMA demostró ser el método más efectivo de los cuatro mencionados para predecir series temporales en una situación específica. La fortaleza principal de SARIMA radica en su capacidad precisa y sólida para capturar patrones estacionales y tendencias presentes en los datos. Este modelo logra esto mediante la combinación de componentes auto-regresivos, de promedio móvil y de diferenciación, lo que le permite adaptarse eficazmente a una amplia gama de series temporales. Además, SARIMA ofrece la flexibilidad de ajustar manualmente sus parámetros, lo que lo convierte en una opción versátil para abordar situaciones complicadas.