REPRESENTACIÓN DE SERIES TEMPORALES

```
display(furniture)
```

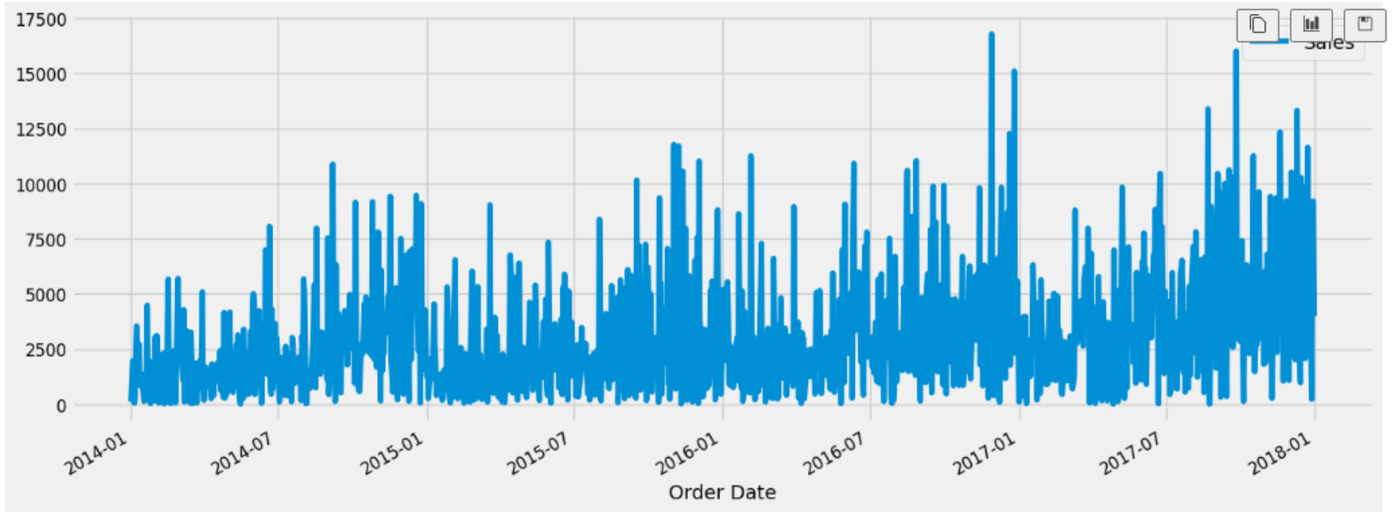
✓ 0.0s

Order Date	Sales
2014-01-01	113.6700
2014-01-03	1960.8225
2014-01-04	1686.7000
2014-01-05	48.7800
2014-01-06	324.4500
...	...
2017-12-27	2815.0026
2017-12-28	224.7000
2017-12-29	9109.4028
2017-12-30	9210.8869
2017-12-31	3995.2371

1331 rows × 1 columns

```
furniture.plot(figsize=(15, 6))
plt.show()
```

✓ 0.1s Pythor



- La representación más sencilla de una serie consiste en representarla como el grafo de una función del tiempo

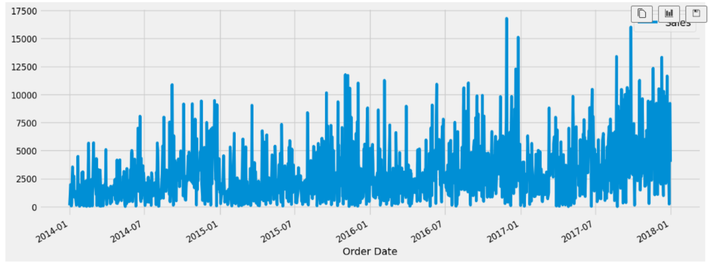
REPRESENTACIÓN DE SERIES TEMPORALES

```
display(furniture)
```
✓ 0.0s

Order Date	Sales
2014-01-01	113.6700
2014-01-03	1960.8225
2014-01-04	1686.7000
2014-01-05	48.7800
2014-01-06	324.4500
...	...
2017-12-27	2815.0026
2017-12-28	224.7000
2017-12-29	9109.4028
2017-12-30	9210.8869
2017-12-31	3995.2371

1331 rows × 1 columns

```
furniture.plot(figsize=(15, 6))
plt.show()
```
✓ 0.1s

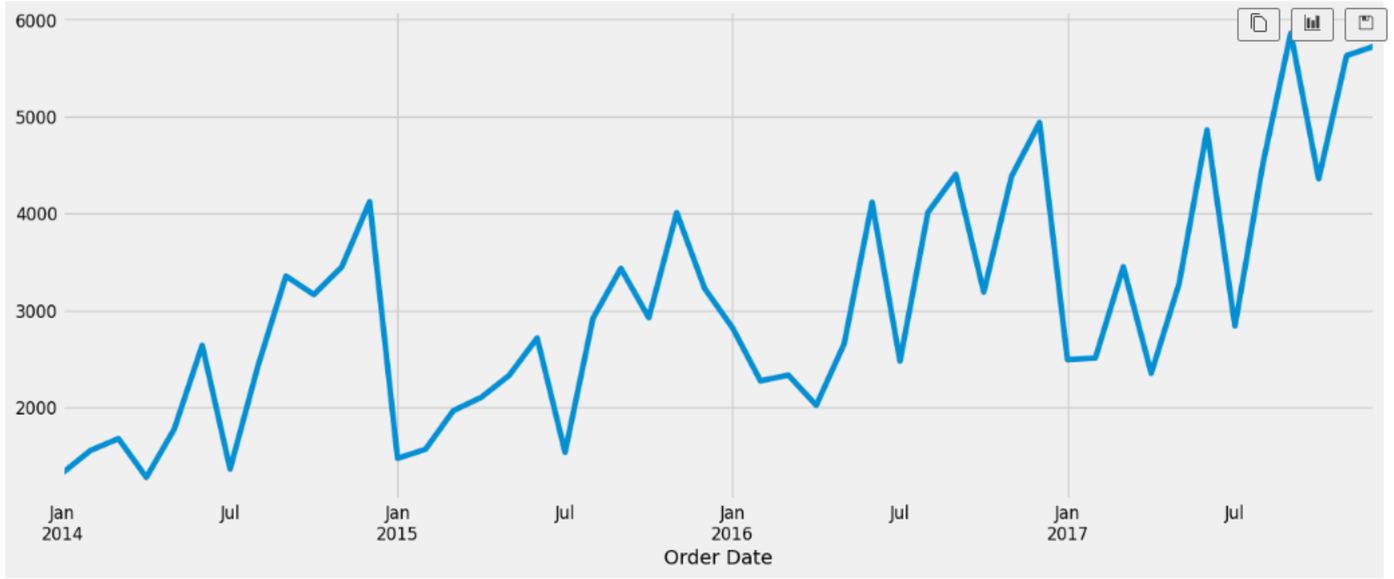



```
# Datos mensuales, fecha del último día del mes
y = furniture['Sales'].resample('M').mean()
```

```
display(y)
```
✓ 0.0s

```
Order Date
2014-01-31    1325.528962
2014-02-28    1556.495109
2014-03-31    1678.219783
2014-04-30    1278.731046
2014-05-31    1777.452571
2014-06-30    2640.850083
2014-07-31    1364.476639
```

```
y.plot(figsize=(15, 6))
plt.show()
```
✓ 0.1s



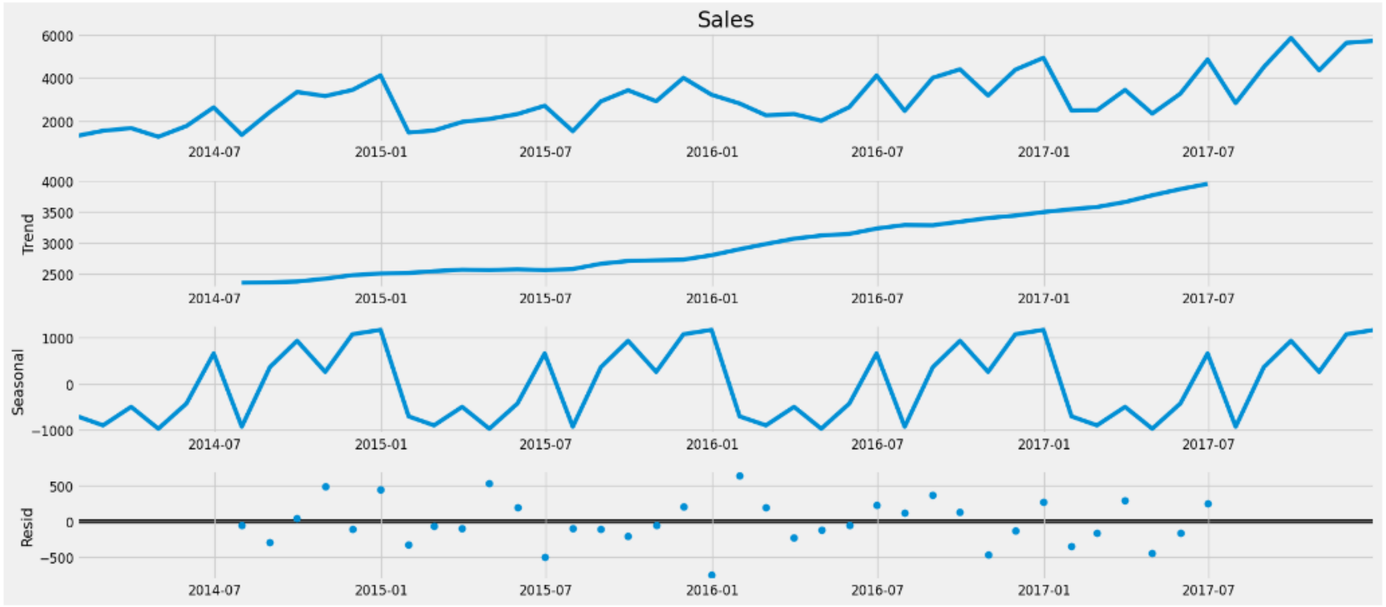
- La representación directa de los valores no siempre es informativa; generalmente hay que preprocesar la serie hasta que el periodo de muestreo es adecuado

DESCOMPOSICIÓN ESTACIONAL

- Un análisis exploratorio sencillo consiste en descomponer la serie en suma (o producto) de una parte periódica y de una tendencia, lo cual permite extrapolar la serie mediante la suma de la extrapolación de la tendencia y de la parte periódica

```python
# Descomposición estacional
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
plt.show()
```

Python

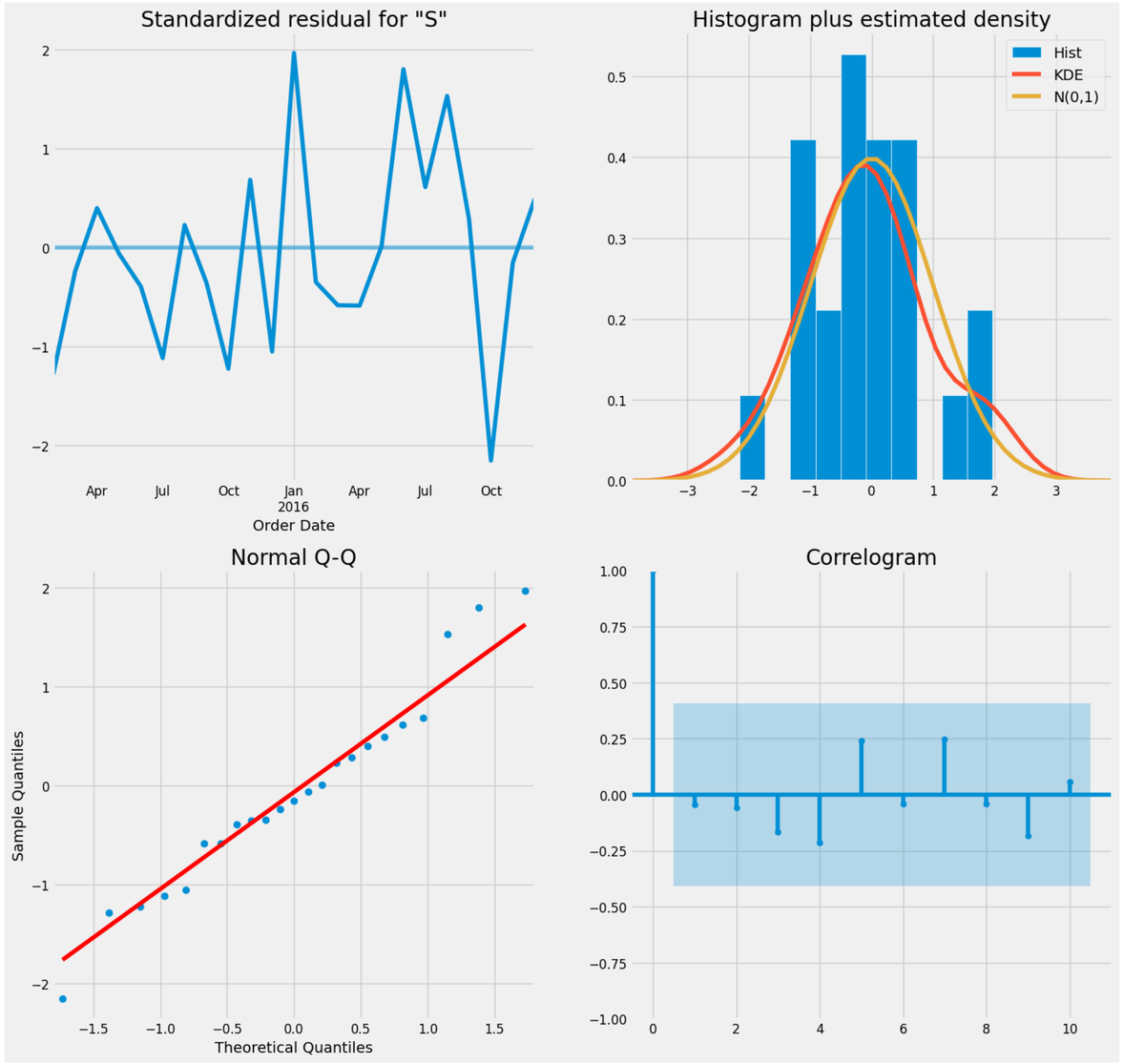


MODELOS ARIMA + VARIABLES EXÓGENAS

- Los modelos ARIMA ya han sido vistos en la carrera (combinan un término autoregresivo con una media móvil y una diferenciación, que puede mejorar la estacionareidad
- Los modelos SARIMA (Seasonal ARIMA) añaden una pareja de términos AR y MA sobre periodos a múltiplos de la periodicidad de la serie
- Los modelos SARIMAX añaden una variable exógena
- El orden de cada uno de los términos puede determinarse mediante el criterio de información de Akaike (penaliza a los modelos excesivamente complejos)

SARIMAX

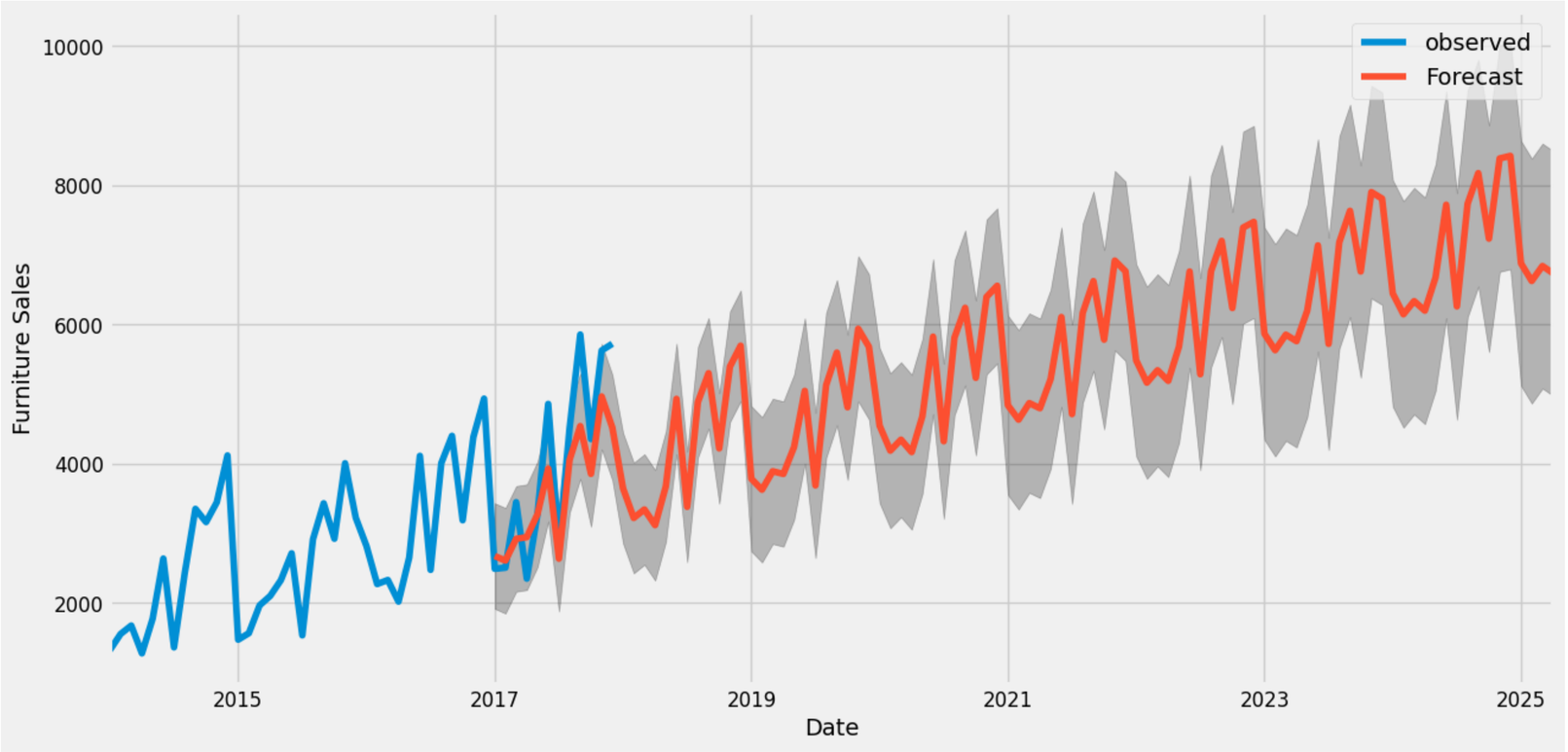
- El ajuste de los datos al modelo se valida gráficamente mediante el residuo estandar, el gráfico Q-Q , el histograma de los residuos y el correlograma



PREDICCIÓN E INCERTIDUMBRE

- La predicción y su incertidumbre asociada pueden mostrarse rellenando el espacio entre los intervalos superiores e inferiores de la predicción

```
# Predicción a múltiples pasos
pred_uc = results.get_forecast(steps=100)
pred_ci = pred_uc.conf_int()
ax = y.plot(label='observed', figsize=(14, 7))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Furniture Sales')
plt.legend()
plt.show()
```



HOLT-WINTERS

1. **Componente de nivel (suavizado)**: Estima el nivel actual de la serie.

$$S_t = \alpha \times (Y_t - I_{t-L}) + (1 - \alpha) \times (S_{t-1} + B_{t-1})$$

2. **Componente de tendencia**: Estima la tendencia de la serie.

$$B_t = \beta \times (S_t - S_{t-1}) + (1 - \beta) \times B_{t-1}$$

3. **Componente estacional**: Estima la estacionalidad de la serie.

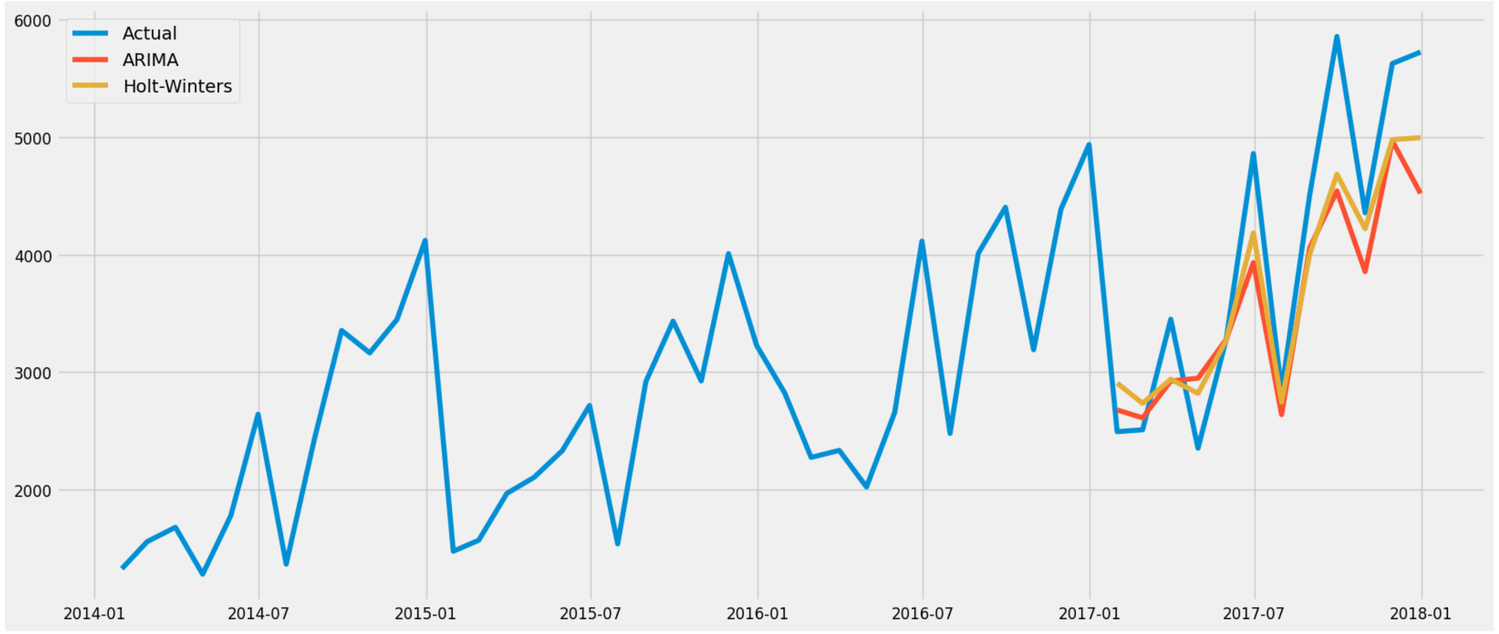
$$I_t = \gamma \times (Y_t - S_t) + (1 - \gamma) \times I_{t-L}$$

4. **Predicción para m periodos adelante**:

$$\hat{Y}_{t+m} = S_t + m \times B_t + I_{t-L+1+(m-1) \mod L}$$

Donde:

- Y_t es el valor observado en el tiempo t.
- S_t es el componente de nivel en el tiempo t.
- B_t es el componente de tendencia en el tiempo t.
- I_t es el componente estacional en el tiempo t.
- L es la longitud del ciclo estacional.
- α, β, y γ son coeficientes de suavizado para el nivel, la tendencia y la estacionalidad, respectivamente, y están en el rango [0,1].




```python
from statsmodels.tsa.api import ExponentialSmoothing
hw_model = ExponentialSmoothing(
    y_train, trend='add', seasonal='add', seasonal_periods=12).fit()
predicciones_hw = hw_model.forecast(12)
```

PROPHET (META)

- Prophet es una herramienta desarrollada por Facebook para la predicción de series temporales, que tiene varias ventajas frente a los métodos estadísticos:
  - **Manejo de Estacionalidades:** Prophet puede detectar automáticamente estacionalidades diarias, semanales y anuales en los datos. También permite a los usuarios definir estacionalidades personalizadas.
  - **Días Festivos y Eventos Especiales:** Prophet permite incorporar información sobre días festivos o eventos que pueden afectar las series temporales, como el "Black Friday" o Navidad.
  - **Componentes Aditivos**: El modelo que Prophet utiliza es aditivo, lo que significa que está compuesto por varias componentes (tendencia, estacionalidad, días festivos) que se suman para producir el pronóstico final.
  - **Implementación en Python y R:** Prophet está disponible tanto para Python como para R, lo que lo hace accesible para una amplia gama de usuarios y aplicaciones.
  - **Datos perdidos:** Prophet es robusto frente a datos faltantes y cambios bruscos en las tendencias.

PROPHET (META)

- El modelo de Prophet se basa en tres componentes principales:
  - **Tendencia (Trend):** Prophet intenta capturar tendencias no lineales con un modelo de crecimiento que puede ser saturado (logístico) o no saturado (lineal). El modelo logístico es útil cuando se espera que la serie alcance una saturación o un punto de inflexión. El usuario puede especificar puntos de cambio potenciales o dejar que Prophet los detecte automáticamente.
  - **Estacionalidad (Seasonality):** Prophet incluye componentes de estacionalidad anual y semanal. También permite a los usuarios agregar sus propias estacionalidades. Estas estacionalidades se modelan utilizando submodelos con Fourier, lo que permite capturar patrones cíclicos.
  - **Días festivos y eventos especiales:** El usuario puede proporcionar una lista de fechas que representen eventos especiales, como días festivos, y Prophet intentará estimar su efecto.
- El modelo final es una suma de estos componentes. Matemáticamente, el modelo tiene similitudes con los modelos aditivos generalizados (GAMs), aunque con estructuras específicas para la estacionalidad y la tendencia.

PROPHET (META)

- Para hacer una predicción con prophet el índice debe nombrarse "ds" y el valor de la serie "y"

```
# ------------------------------------
# Prophet
# ------------------------------------


import pandas as pd
from datetime import datetime
datelist = pd.date_range(datetime.today(), periods=100).tolist()
valores = np.arange(100)+np.random.normal(loc=0, scale=10, size=100)+15*np.sin(2*np.pi*np.arange(100)/7)

ejemplo = pd.DataFrame(
    {"ds": datelist, "y": valores}
)

display(ejemplo)
```

5] ✓ 0.0s Python

	ds	y
0	2023-10-17 22:55:02.172150	0.128795
1	2023-10-18 22:55:02.172150	10.542799
2	2023-10-19 22:55:02.172150	2.751019
3	2023-10-20 22:55:02.172150	28.326793
4	2023-10-21 22:55:02.172150	-1.850171
...	...	...
95	2024-01-20 22:55:02.172150	88.896484
96	2024-01-21 22:55:02.172150	81.989146
97	2024-01-22 22:55:02.172150	79.750928
98	2024-01-23 22:55:02.172150	108.070114
99	2024-01-24 22:55:02.172150	104.015624

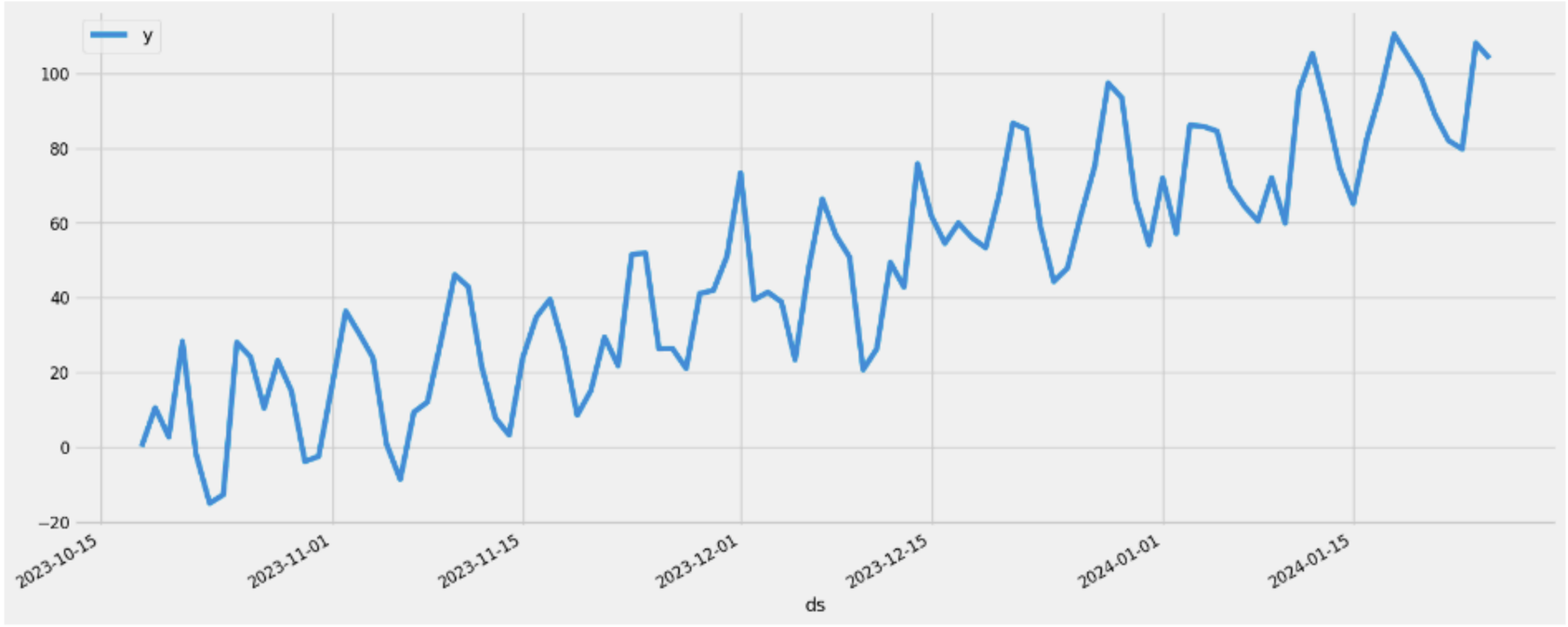
100 rows × 2 columns

PROPHET (META)

```
ejemplo.plot(x="ds",y="y")
```

✓ 0.1s Python

```
<Axes: xlabel='ds'>
```



PROPHET (META)

- Para realizar predicciones se construye un dataframe con los valores de índice correspondientes
- Con la orden "predict" se obtienen predicciones junto con los intervalos de predicción al valor seleccionado

```
from prophet import Prophet
m = Prophet()
m.fit(ejemplo)
```

✓ 0.0s Python

```
22:55:08 - cmdstanpy - INFO - Chain [1] start processing
22:55:08 - cmdstanpy - INFO - Chain [1] done processing

<prophet.forecaster.Prophet at 0x16dc2a910>
```

```
future = m.make_future_dataframe(periods=100)
future.tail()
```

✓ 0.0s Python

	ds
195	2024-04-29 22:55:02.172150
196	2024-04-30 22:55:02.172150
197	2024-05-01 22:55:02.172150
198	2024-05-02 22:55:02.172150
199	2024-05-03 22:55:02.172150

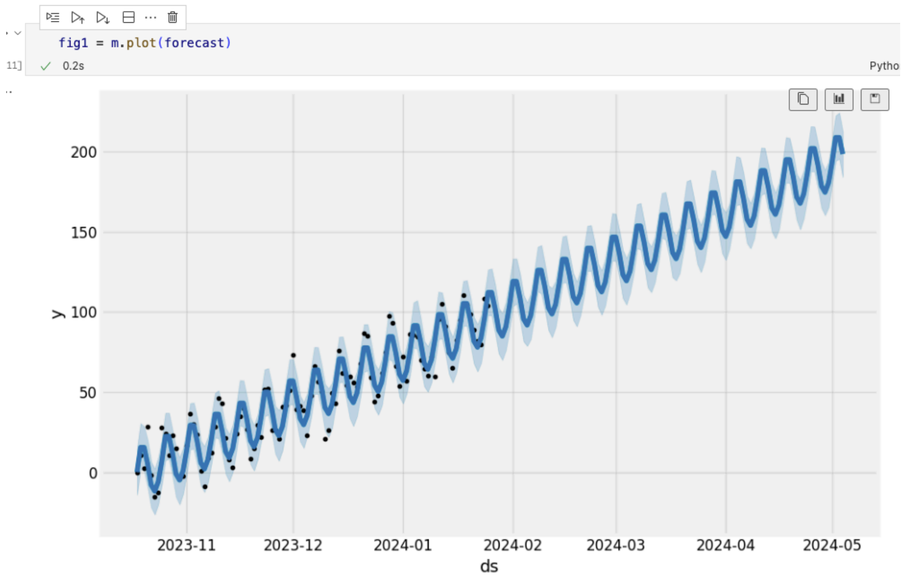
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

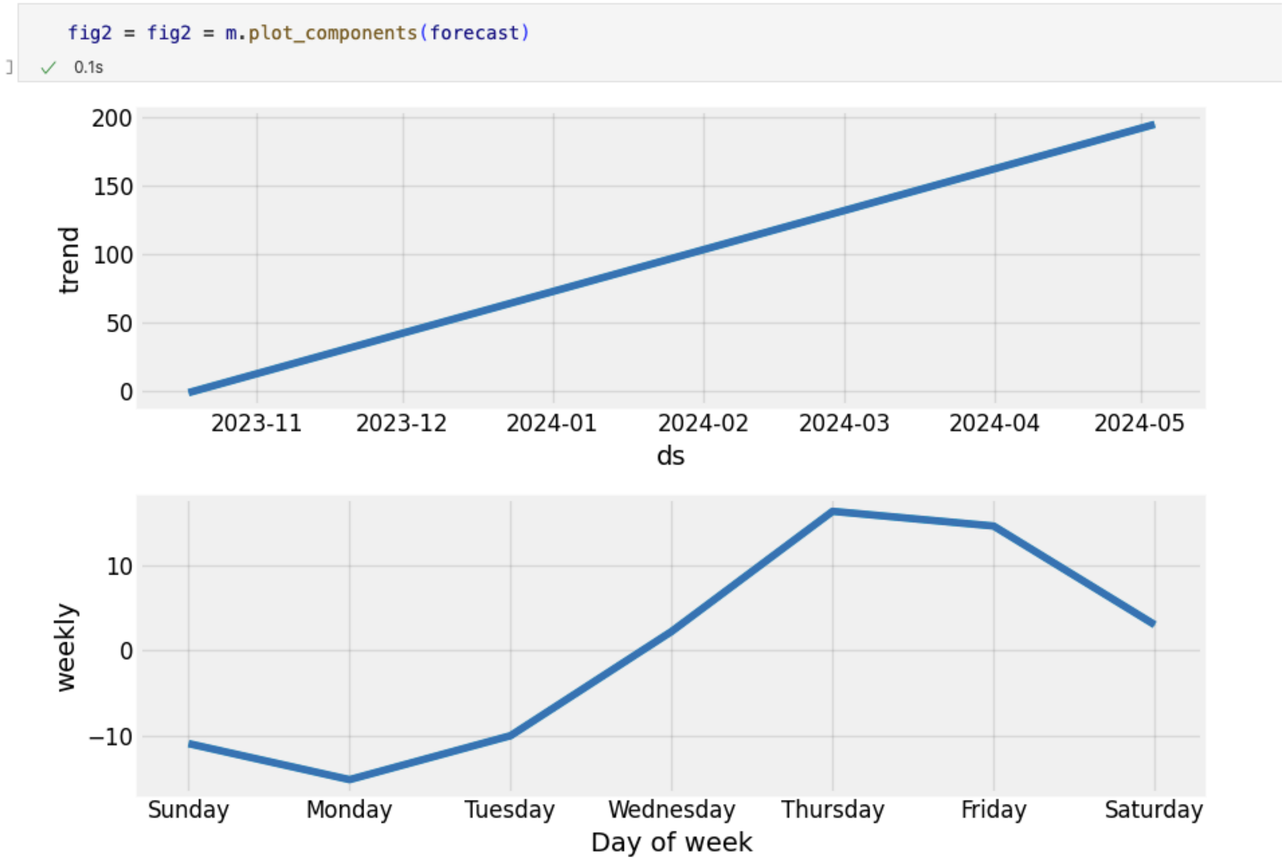
✓ 0.0s Python

	ds	yhat	yhat_lower	yhat_upper
195	2024-04-29 22:55:02.172150	180.425869	168.963986	191.792596
196	2024-04-30 22:55:02.172150	193.243774	182.248583	204.166448
197	2024-05-01 22:55:02.172150	208.724888	197.590873	220.015874
198	2024-05-02 22:55:02.172150	208.710422	197.250171	220.011124
199	2024-05-03 22:55:02.172150	198.352855	187.439939	209.709265

PROPHET (META)

- Pueden obtenerse gráficos con los intervalos de predicción mediante las órdenes plot / plot_components



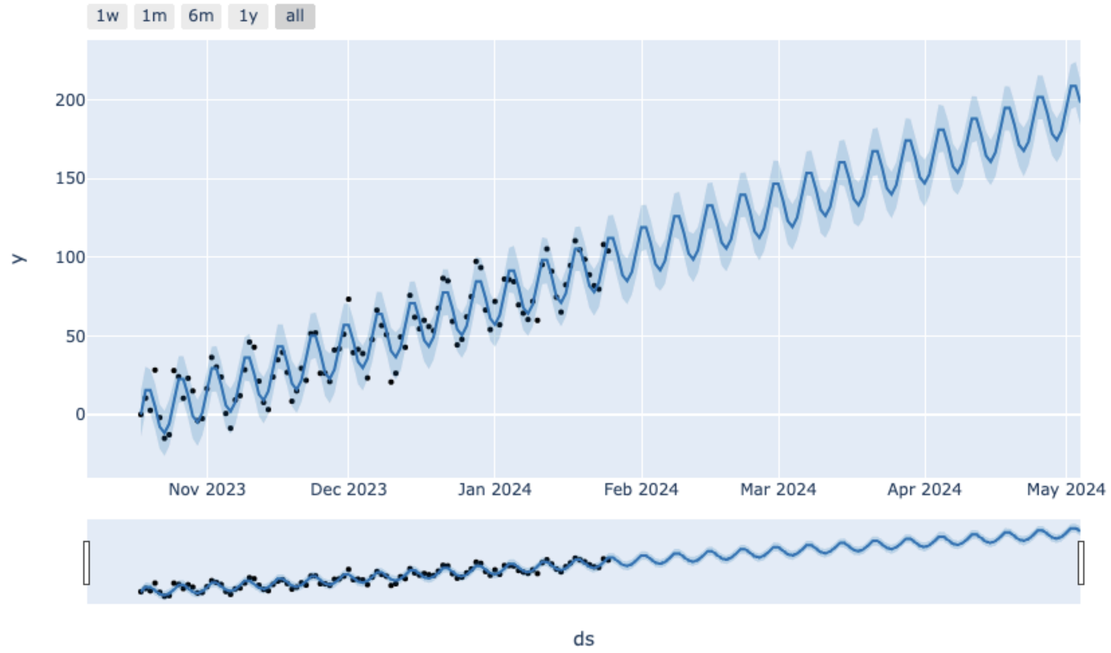


PROPHET (META)

- También está integrado con plotly para obtener gráficos interactivos

```
from prophet.plot import plot_plotly, plot_components_plotly

plot_plotly(m, forecast)
```

13] ✓ 0.0s



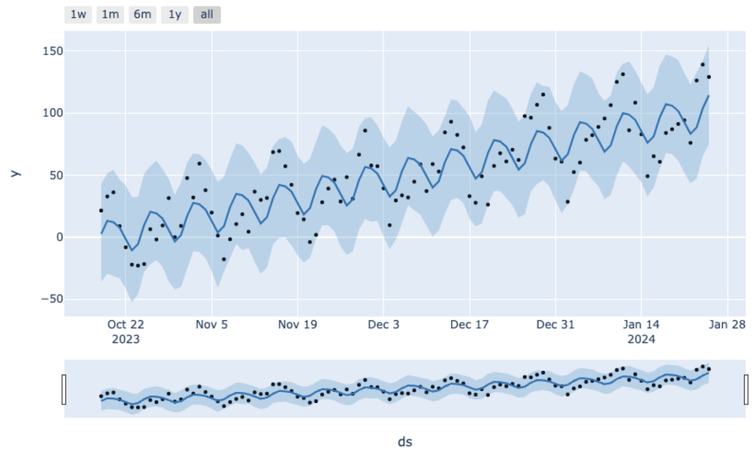
PROPHET (META)

- Pueden añadirse variables exógenas a la serie para mejorar las predicciones (ver notebook de ejemplo)

```
combinado = Prophet(interval_width=0.95)
combinado.fit(ejemplo)
forecast = combinado.make_future_dataframe(periods=36, freq='M')
forecast = combinado.predict(ejemplo)
plot_plotly(combinado, forecast)
```

[285] ✓ 0.1s

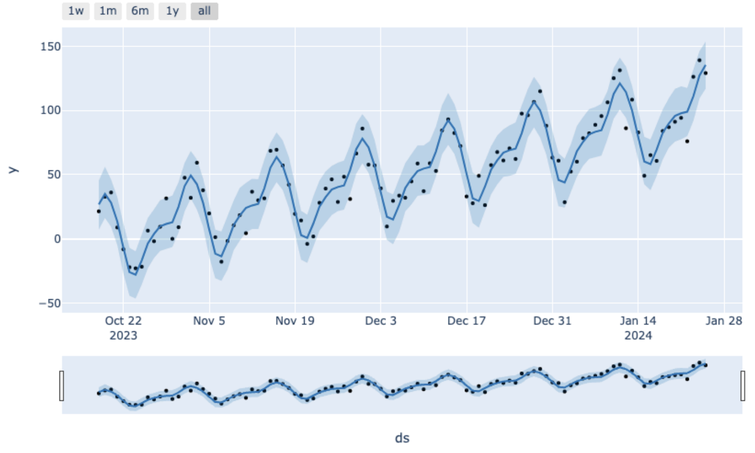
```
23:29:28 - cmdstanpy - INFO - Chain [1] start processing
23:29:28 - cmdstanpy - INFO - Chain [1] done processing
```



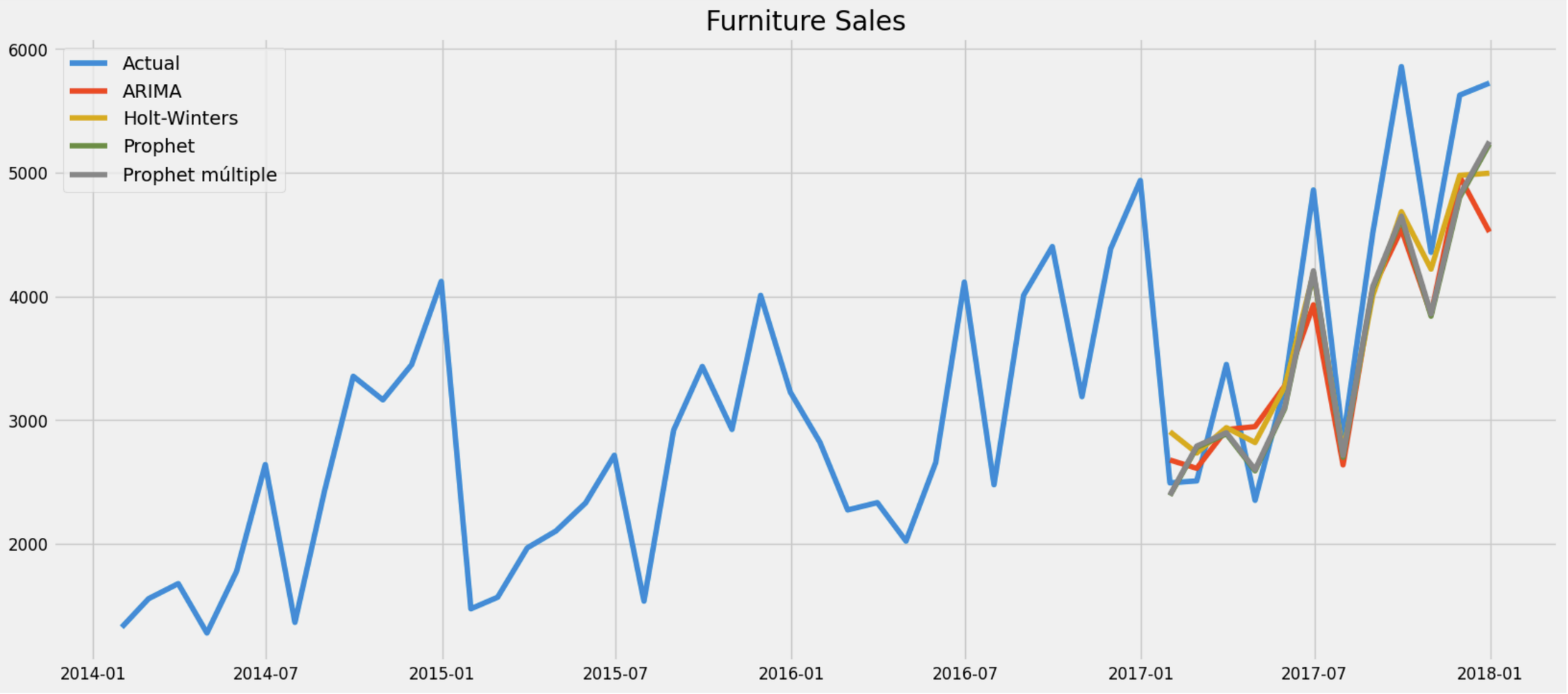

```
combinado = Prophet(interval_width=0.95)
combinado.add_regressor("entrada", standardize=False)
combinado.fit(ejemplo)
forecast = combinado.make_future_dataframe(periods=36, freq='M')
forecast = combinado.predict(ejemplo)
plot_plotly(combinado, forecast)
```

✓ 0.0s

```
23:29:36 - cmdstanpy - INFO - Chain [1] start processing
23:29:36 - cmdstanpy - INFO - Chain [1] done processing
```



PROPHET


Furniture Sales
Actual
ARIMA
Holt-Winters
Prophet
Prophet múltiple
6000
5000
4000
3000
2000
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01

DEEPAR (AMAZON)

- DeepAR es un método de predicción de series temporales desarrollado por Amazon para su servicio Amazon SageMaker.
- Es un modelo basado en redes neuronales recurrentes (RNN), específicamente en las células LSTM (Long Short-Term Memory)
- Está diseñado para manejar múltiples series temporales con la capacidad de aprovechar patrones aprendidos en una serie para mejorar la predicción en otras.

DEEPAR (AMAZON)

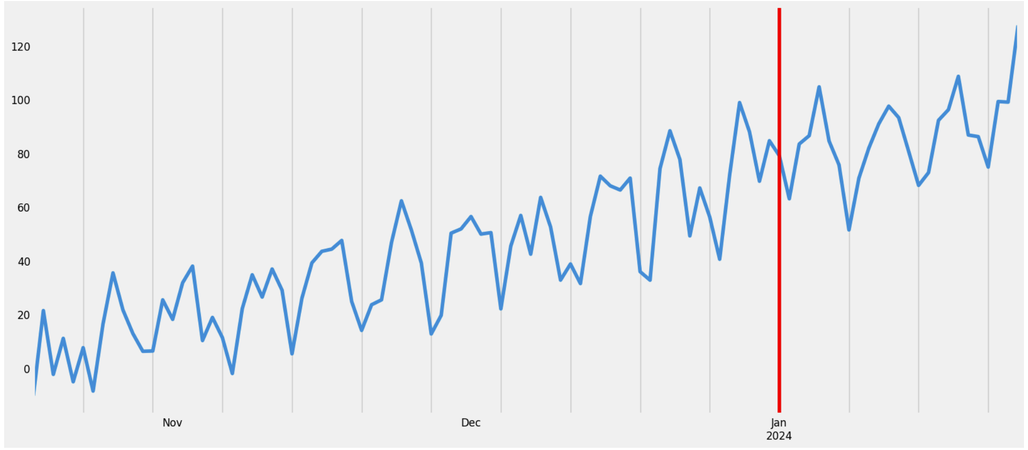
- **Uso de Series Múltiples:** se puede entrenar en múltiples series temporales simultáneamente. Esto es particularmente útil cuando se tienen muchas series temporales relacionadas, ya que el modelo puede aprender patrones de una serie y usar ese aprendizaje para mejorar las predicciones en otra.
- **Modelo Probabilístico:** produce predicciones en forma de distribuciones de probabilidad, no solo puntos de predicción.
- **Manejo de Valores Perdidos:** capacidad para manejar valores faltantes en las series temporales
- **Inclusión de Características Exógenas:** permite la inclusión de características adicionales (exógenas) que pueden influir en las predicciones, lo que aumenta la flexibilidad del modelo para diferentes aplicaciones.
- **Escalabilidad:** puede utilizarse en combinación con Amazon SageMaker. Está diseñado para ser escalable y manejar grandes cantidades de datos.

DEEPAR (AMAZON)

- DeepAR está pensado para entrenar una red y validarla de forma simultánea en varios datasets al mismo tiempo
- La estructura de datos de entrenamiento es una lista de datasets. Los datos de set son una lista en la que cada elemento contiene tanto los datos de entrenamiento como los de test

```
# Una variable "target", una fecha "start" y una frecuencia
training_data = ListDataset(
    [{"start": ejemplo.index[0], "target": ejemplo.loc[:endTrain,"variable"], }],
    freq="D"
)
# Train + test
test_data = Lis
    [{"start": ejemplo.index[0], "target": ejemplo.loc[:,"variable"], }],
    freq="D"
)

to_pandas(test_data[0]).plot()
plt.axvline(endTrain, color='r')
plt.grid(which="both")
plt.show()
```

(variable) ejemplo: DataFrame



DEEPAR (AMAZON)

- El estimador indica el horizonte de predicción y los parámetros de la red neuronal

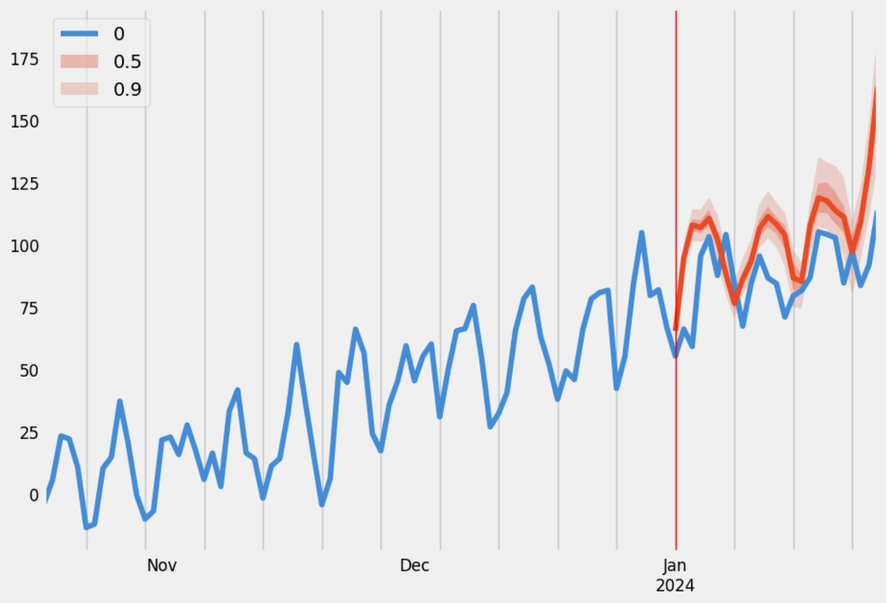
```python
estimator = DeepAREstimator(freq="D",
                            prediction_length=25,
                            trainer_kwargs={"max_epochs": 100, "accelerator": "cpu"})

predictor = estimator.train(training_data)
```

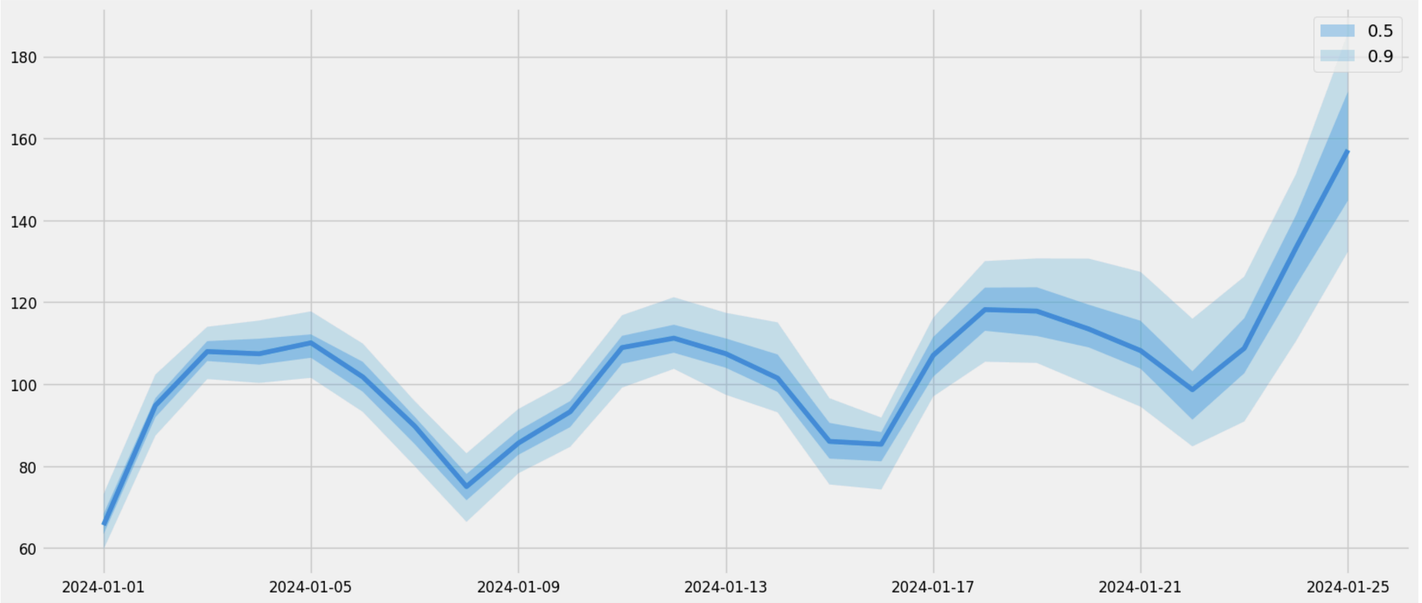
✓ 2m 3.5s Python

```python
forecast_it, ts_it = make_evaluation_predictions(
    dataset=test_data,  # test dataset
    predictor=predictor,  # predictor
    num_samples=100,  # number of sample paths we want for evaluation
)
forecasts = list(forecast_it)
tss = list(ts_it)

print(f"Number of sample paths: {forecasts[0].num_samples}")
print(f"Dimension of samples: {forecasts[0].samples.shape}")
print(f"Start date of the forecast window: {forecasts[0].start_date}")
print(f"Frequency of the time series: {forecasts[0].freq}")

plot_prob_forecasts(tss[0], forecasts[0], 100)
plt.show()

forecasts[0].plot(show_label=True)
plt.legend()
```

✓ 0.3s Python

DEEPAR (AMAZON)



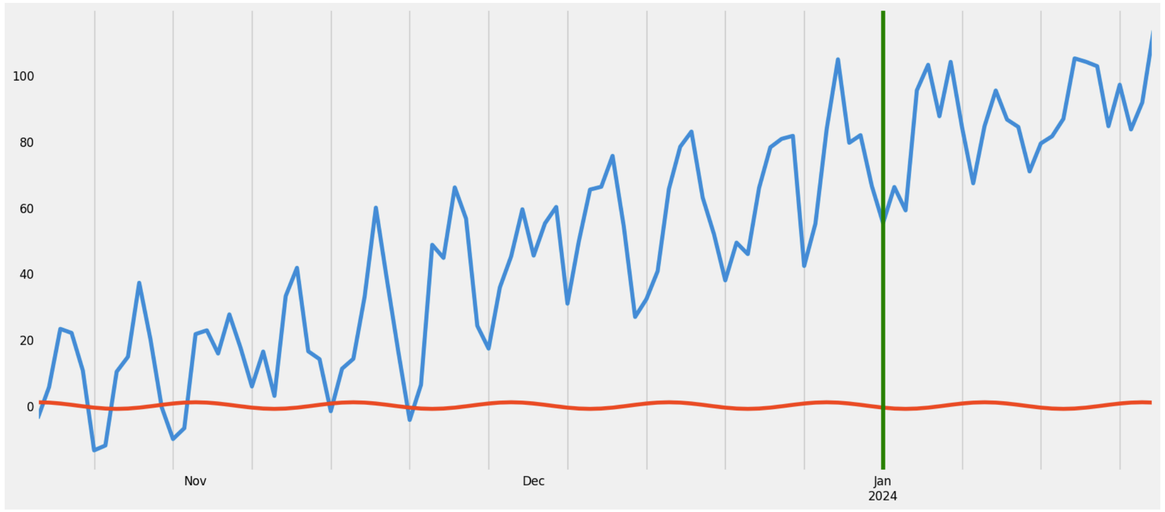


DEEPAR (AMAZON)

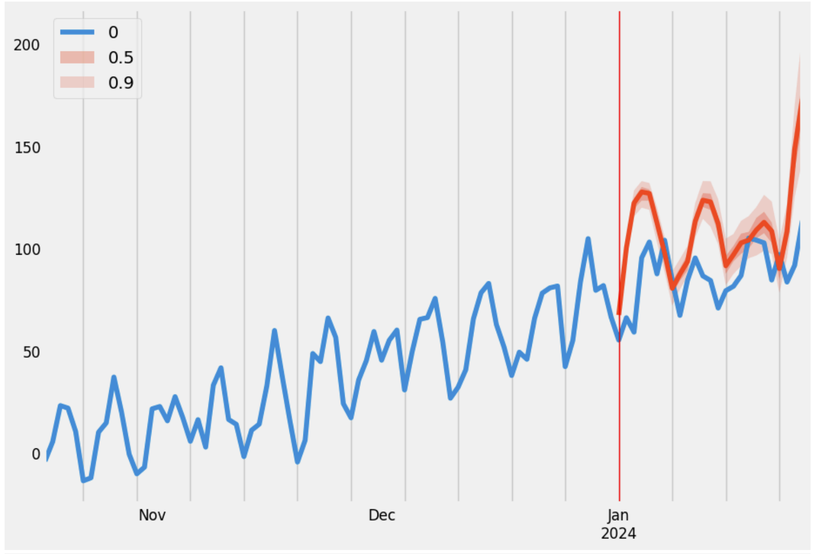
- En predicciones para múltiples series se entrena una única red neuronal y se define su error como la media de los errores sobre cada serie. Esto es diferente de los modelos para series multivaluadas (VARIMA) o de serie + variables exógenas (ARIMAX/ SARIMAX/Prophet)

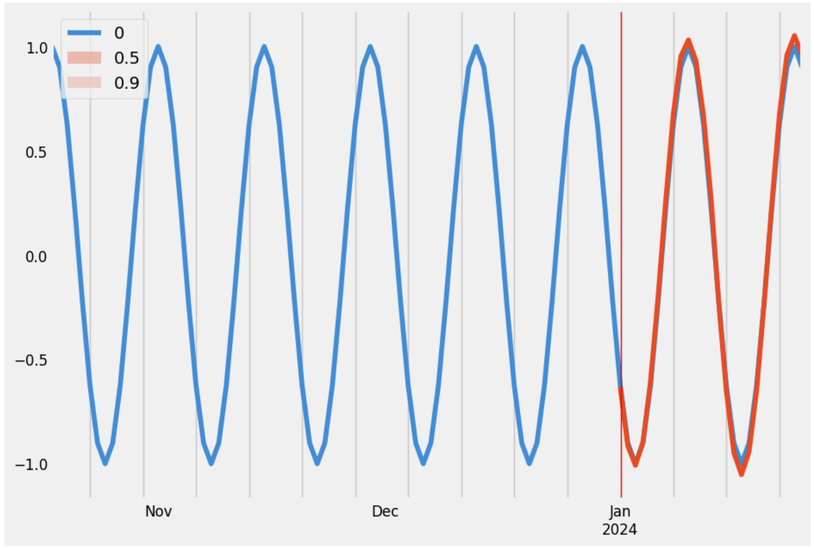
```
# Una variable "target", una fecha "start" y una frecuencia
training_data = ListDataset(
    [{"start": ejemplo.index[0], "target": ejemplo.loc[:endTrain,"variable"]},
     {"start": ejemplo.index[0], "target": ejemplo.loc[:endTrain,"entrada"]}],
    freq="D"
)
# Train + test
test_data = ListDataset(
    [{"start": ejemplo.index[0], "target": ejemplo.loc[:,"variable"]},
     {"start": ejemplo.index[0], "target": ejemplo.loc[:,"entrada"]}],
    freq="D"
)

to_pandas(test_data[0]).plot()
to_pandas(test_data[1]).plot()
plt.axvline(endTrain, color='g')
plt.grid(which="both")
plt.show()
```

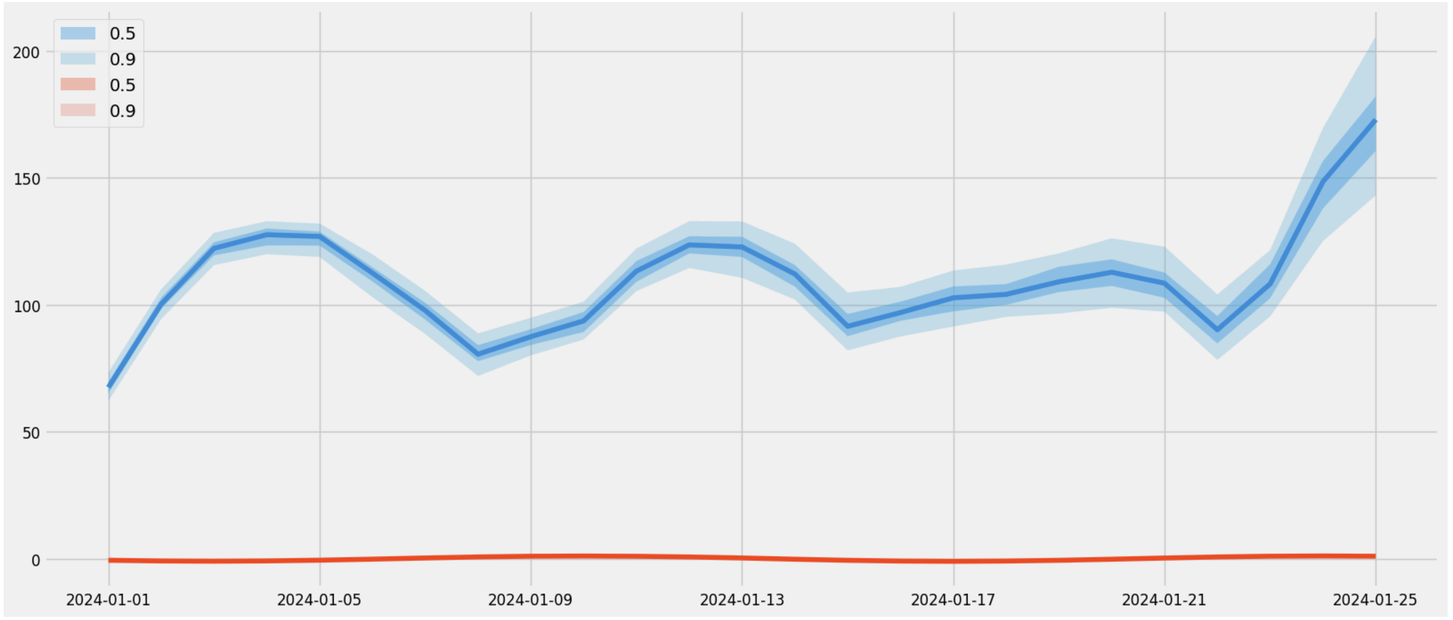
✓ 0.1s Python



DEEPAR (AMAZON)







DEEPAR (AMAZON)