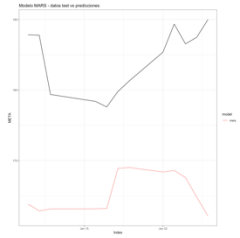
Informe evaluación del módulo 2

1. Modelos implementados

En el presente informe se describe la implementación de cuatro modelos de pronóstico de series de tiempo aplicados a los precios de las acciones de META en el periodo comprendido entre el 1 de mayo de 2016 y el 28 de enero de 2018. Los modelos evaluados fueron: MARS, LSTM, Random Forest y SSA.

1.1 Modelo MARS: El modelo MARS es un enfoque flexible de regresión no lineal que divide los datos en intervalos y ajusta funciones spline para capturar relaciones no lineales.

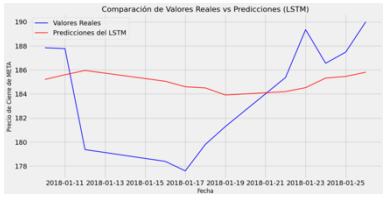
Ventajas: Permite modelar relaciones complejas sin necesidad de definir explícitamente la forma de la no linealidad. Es interpretable, al mostrar las relaciones mediante las bases spline. Tiene una buena capacidad para manejar variables predictoras continuas.
Desventajas: Su rendimiento disminuye en presencia de datos con alta dependencia temporal o series altamente autocorrelacionadas. Puede sobreajustar si no se seleccionan adecuadamente los parámetros de regularización.



En este caso, MARS capturó parte de la tendencia de los precios de META, pero mostró limitaciones para reflejar la dinámica temporal completa, lo cual se refleja en su **RMSE de 19.44** y **MAPE de 10.13%**, siendo el desempeño más bajo entre los modelos evaluados. Podemos evidenciar al graficar los valores del set de Test que las predicciones de nuestro modelo MARS no son tranprecisas. En los últimos valores se puede evidenciar que aunque los valores tendían a crecer, las predicciones del modelo eran valores en disminución

1.2 Modelo LSTM: La red LSTM es una arquitectura de redes neuronales recurrentes diseñada para aprender dependencias a largo plazo en series de tiempo.

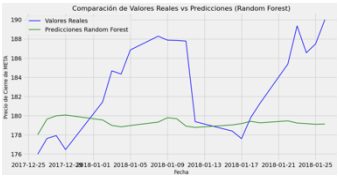
Ventajas: Excelente capacidad para capturar patrones secuenciales y dependencias temporales. Puede modelar relaciones no lineales y dinámicas complejas. Se adapta bien a datos con estacionalidad o tendencia.
Desventajas: Requiere mayor tiempo de cómputo y recursos computacionales para el entrenamiento. Necesita mayor ajuste de hiperparámetros y experimentación.Difícil de interpretar respecto al impacto de cada variable.



Para el caso de los precios de META, el LSTM mostró **el mejor desempeño global** con un **RMSE de 4.3**, **MAE de 3.81** y **MAPE de 2.09%**, capturando de manera adecuada las tendencias y fluctuaciones del mercado durante el periodo analizado. Como se observa en la grafica, los valores pronosticados por el modelo LSTM para el set de datos de test, es muy cercano al valor real.

1.3 Modelo RF: Es un modelo de ensamble basado en múltiples árboles de decisión, utilizado principalmente para problemas de clasificación y regresión.
Ventajas: Robusto frente al sobreajuste y valores atípicos. Fácil de ajustar y con buen rendimiento sin necesidad de mucha parametrización. Permite estimar la importancia de las variables.

Desventajas: No está diseñado específicamente para datos secuenciales o autocorrelacionados.No captura directamente la dependencia temporal sin ingeniería de características adicional.Puede presentar limitaciones al predecir valores extremos en series de tiempo.



En este proyecto, Random Forest ofreció un buen desempeño general (**RMSE de 6.15**, **MAPE de 2.75%**), aunque ligeramente inferior al LSTM y SSA, sugiriendo que, aunque robusto, no logra capturar toda la dinámica secuencial de la serie. Podemos observar que aunque los valores de las predicciones están cerca de los valores reales el modelo no logra capturar de mejor forma el comportamiento creciente de la serie.

1.4 Modelo SSA: (Singular Spectrum Analysis): Es un método de descomposición espectral que permite descomponer una serie en componentes como tendencia, estacionalidad y ruido.

Ventajas: Permite identificar y aislar componentes subyacentes de la serie. Es no paramétrico, no requiere suposiciones sobre la estructura del modelo.Útil para análisis exploratorio y reducción de ruido.

Desventajas: No es explícitamente predictivo, requiere métodos adicionales para proyectar hacia adelante. Puede ser sensible a la selección de la ventana de embebido.Limitado para capturar shocks o cambios abruptos.

En este análisis, el SSA logró un desempeño similar al LSTM, con **RMSE de 5.47**, **MAE de 4.69** y **MAPE de 2.59%**, mostrando capacidad para capturar las principales componentes del comportamiento del precio. La descomposición de la serie de tiempo mediante el método SSA (Análisis Espectral Singular) permite separar la serie en componentes clave como la tendencia, la estacionalidad y los componentes de ruido. La tendencia, que refleja el comportamiento general y el crecimiento de los datos, se identifica claramente, mientras que la estacionalidad captura patrones recurrentes que ocurren en intervalos regulares. Estos componentes se combinan de manera eficiente para proporcionar una reconstrucción precisa de la serie original. Al hacer esta separación, SSA permite al modelo centrarse en las características más relevantes, mejorando su capacidad de pronóstico y optimizando el ajuste a los datos. De esta manera, la descomposición de la serie mejora el desempeño del modelo, facilitando la captura de las relaciones subyacentes de los datos.

2. Selección del modelo con mejor desempeño

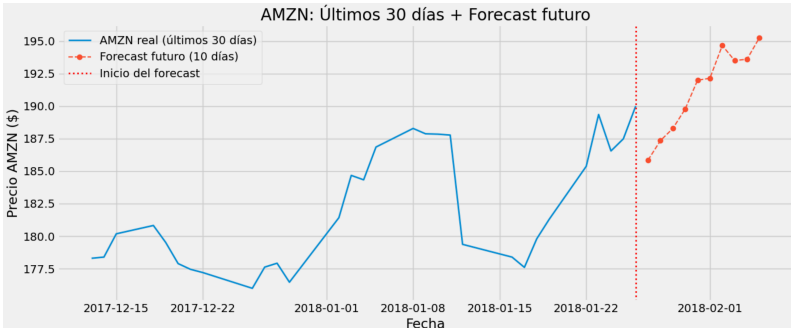
Tras la comparación de los cuatro modelos, podemos observar que el modelo LSTM obtuvo las mejores métricas de desempeño, con el menor RMSE (4.33), MAE (3.82) y un MAPE del 2.09%. Estos valores reflejan una mayor precisión en las predicciones frente al resto de modelos, especialmente comparado con MARS que presentó errores más altos. El modelo SSA también mostró un desempeño competitivo, con métricas similares al LSTM (MAPE de 2.59%), lo que sugiere que es una alternativa viable para pronósticos de corto plazo o series con patrones bien definidos. Sin embargo, el LSTM demostró mayor capacidad para capturar las dinámicas

complejas y secuenciales del precio de las acciones, por lo que se selecciona como el modelo con mejor desempeño. Esta selección se justifica no solo por los valores métricos, sino también por su capacidad para adaptarse a la naturaleza secuencial y no lineal de los precios bursátiles, lo cual es esencial en contextos de mercado volátiles como el de META.

Modelo	RMSE	MAE	MAPE (%)
MARS	19.45	18.77	10.14
LSTM	4.33	3.82	2.09
Random Forest	6.15	5.12	2.75
SSA	5.48	4.69	2.59

En conclusión, el modelo **LSTM** se posiciona como la mejor opción para el pronóstico de los precios de las acciones de META, al demostrar una capacidad superior para capturar las dinámicas no lineales y las dependencias temporales inherentes a las series de tiempo financieras. Su arquitectura basada en memorias de largo plazo le permite adaptarse de manera más eficaz a las fluctuaciones y tendencias del mercado, logrando las mejores métricas de desempeño entre los modelos evaluados (RMSE de 4.33, MAE de 3.82 y MAPE de 2.09%). Comparado con MARS, Random Forest y SSA, el LSTM ofrece ventajas clave como su habilidad para modelar secuencias sin necesidad de ingeniería de características adicional, su flexibilidad para aprender patrones complejos directamente de los datos, y su adaptabilidad ante cambios en la dinámica del mercado, lo que lo convierte en una herramienta poderosa y precisa para pronósticos financieros en entornos volátiles y altamente dinámicos.

En la siguiente gráfica se presenta un pronóstico a 10 días del precio de la acción de **META** utilizando el modelo **LSTM**. Se puede apreciar que el modelo ha logrado capturar de manera efectiva la tendencia creciente observada en la serie histórica, manteniendo la coherencia con el comportamiento previo de los datos. El pronóstico sigue este patrón ascendente sin presentar valores atípicos o extremos que pudieran generar dudas sobre su fiabilidad, lo cual sugiere que el modelo ha aprendido adecuadamente las dinámicas subyacentes de la serie. Además, la suavidad y continuidad de la curva proyectada reflejan la capacidad del LSTM para procesar la secuencia temporal de manera robusta, evitando fluctuaciones abruptas que no estén respaldadas por el comportamiento histórico. Este resultado respalda la elección del LSTM como el modelo más adecuado para generar proyecciones confiables y realistas en el contexto de los precios bursátiles de META.



ANEXOS

En este primer ANEXO se incluye los modelos LSTM y Random Forest

Librerias

```
# ------------------------------
# Configuración de warnings
# ------------------------------
import warnings
warnings.filterwarnings('ignore')

# ------------------------------
# TensorFlow y Keras
# ------------------------------
import tensorflow as tf
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.sequence import
TimeseriesGenerator
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping

# ------------------------------
# Manipulación de datos
# ------------------------------
import numpy as np
import pandas as pd
from numpy import array, asarray
from pandas import DataFrame, concat

# ------------------------------
# Descarga de datos financieros
# ------------------------------
import yfinance as yf
from pandas_datareader import data as pdr

# ------------------------------
# Visualización
# ------------------------------
import matplotlib.pyplot as plt
import plotly.graph_objects as go
plt.style.use('fivethirtyeight')
plt.rcParams['lines.linewidth'] = 1.5
# %matplotlib inline  # <-- Solo si estás en Jupyter
```

```
# ------------------------------
# Modelado y pronóstico
# ------------------------------
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_absolute_percentage_error
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline

from skforecast.ForecasterAutoreg import ForecasterAutoreg
from skforecast.ForecasterAutoregCustom import ForecasterAutoregCustom
from skforecast.ForecasterAutoregDirect import ForecasterAutoregDirect
from skforecast.model_selection import grid_search_forecaster,
backtesting_forecaster
from skforecast.utils import save_forecaster, load_forecaster

from joblib import dump, load
from datetime import datetime
from math import sqrt
```

PASO 1:Aplicación de los modelos

1. Descarga de Yahoo Finance la serie de precios de cierre del activo META en las fechas: 01-05-2016 al 28-01-2018.

```
import pandas as pd

# Leer los datos desde el archivo CSV
meta = pd.read_csv('META_datos_completos.csv')[['Close', 'Date']]
dt = meta
dt['Date'] = pd.to_datetime(dt['Date'])

dt = dt.set_index('Date')

dt.columns = ['META']

# Ahora agregamos el encabezado 'Ticker' para que quede el formato que
buscas
dt.columns.name = 'Ticker'
```

2. Grafica la serie

```
#Una vez llamada la librería, graficamos la serie dt.
import matplotlib.pyplot as plt
plt.plot(dt)
```

```
[<matplotlib.lines.Line2D at 0x178027f10>]
```

3. Prepara los datos a manera que cuentes con un formato adecuado para la fecha.

```
#Ahora, pasamos el data frame a un objeto de tiempo de pandas.
dataindex= pd.to_datetime(dt.index)
#Ahora, a numpy array:
dt2= dt.to_numpy()
#Y generamos un rearreglo de la dimensión del array a 1D, en los
renglones (el -1 signigica que numpy calcula este número por
nosotros).
close_data = dt2.reshape((-1,1))
```

4. Parte la serie a un 5% de prueba y el restante para el entrenamiento.

```
#Especificamos primeramente el porcentje del corte.
split_percent = 0.95
split = int(split_percent*len(close_data))

#Aplicamos dicha proporción a la definición de la parte de
entrenamiento y de prueba.
```

```
close_train = close_data[:split]
close_test = close_data[split:]

date_train = dataindex[:split]
date_test = dataindex[split:]
```

5. Realiza el preprocesamiento necesario para aplicar cada uno de los modelos

5.1 Preprocesamiento LSTM

```
#Especificamos la longitud o length, es el número de observaciones
tipo lags a emplear en la prción de entrada de cada muestra, el
ejemplo anterior era de 3.
n_back = 10   #la longitud de la ventana
train_generator = TimeseriesGenerator(close_train, close_train,
length=n_back, batch_size=25)
test_generator = TimeseriesGenerator(close_test, close_test,
length=n_back, batch_size=1)
```

5.1 Preprocesamiento RF

```
## Imputamos algún valor faltante con el valor anterior.
dt.fillna(method='bfill', inplace=True)
dt= dt.rename(columns={'META': 'y'})

# Split data into train-test
#
================================================================================
========
split_percent = 0.052
steps = int(split_percent*len(close_data))
#steps = 100
data_train = dt[:-steps]
data_test  = dt[-steps:]
print(f"Train dates : {data_train.index.min()} ---
{data_train.index.max()}  (n={len(data_train)})")
print(f"Test dates  : {data_test.index.min()} ---
{data_test.index.max()}  (n={len(data_test)})")

#Así también para las fechas:
date_train = dataindex[:-steps]
date_test = dataindex[-steps:]
```

```
Train dates : 2016-05-02 00:00:00 --- 2017-12-22 00:00:00  (n=417)
Test dates  : 2017-12-26 00:00:00 --- 2018-01-26 00:00:00  (n=22)
```

6. Aplica cada uno de los modelos MARS, LSTM, RF y SSA a la serie de META

6.1 LSTM

```
##Neural Net
#Importamos primero los módulos requeridos:
from keras.models import Sequential
from keras.layers import LSTM, Dense

#Configuramos la de red.
model = Sequential()
model.add(
    LSTM(200,  ## de nodos, dependerá de su poder de cómputo.
        activation='relu',
        input_shape=(n_back,1))   ##la longitud de la secuencia
)
model.add(Dense(1))  #capa de salida de predicción
model.compile(optimizer='adam', loss='mse')

num_epochs = 30
# Use model.fit instead of model.fit_generator
model.fit(train_generator, epochs=num_epochs, verbose=0)

##Ajuste del modelo a los datos:
history = model.fit(train_generator, epochs=num_epochs, verbose=0)

#Imprimimos gráfico de funión de pérdida acorde los epochs
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

plot_data = [
    go.Scatter(
        x=hist['epoch'],
        y=hist['loss'],
        name='loss'
    )
]

plot_layout = go.Layout(
        title='Training loss')
fig = go.Figure(data=plot_data, layout=plot_layout)
fig.show()
```


```
{"config":{"plotlyServerURL":"https://plot.ly"},"data":
[{"name":"loss","type":"scatter","x":
{"bdata":"AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwd","dtype":"i1"},"y":
{"bdata":"AAAAADqaL0AAAABgyfoqQAAAAOApdTBAAAAAYNNdJUAAAACgBpkfQAAAAGCL
kiFAAAAAAC0PIkAAAACg1osfQAAAAADAfCJAAAAAoGkTKkAAAACgdgonQAAAAOAxuiNAAA
AAYJVBJ0AAAABA8dotQAAAAADt0yhAAAAAYICwKUAAAADA3MYgQAAAAODHQxxAAAAAYK/
```

jI0AAAACgHu0jQAAAACAvSCpAAAAAYAIBIkAAAACAA74fQAAAAIAI+yNAAAAA4ALBJkAAA
ADAZecfQAAAAOA1kyZAAAAAID1MJkAAAADgTbkfQAAAAOAQuBxA","dtype":"f8"}}],"
layout":{"template":{"data":{"bar":[{"error_x":
{"color":"#2a3f5f"},"error_y":{"color":"#2a3f5f"},"marker":{"line":
{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"bar"}],"barpo
lar":[{"marker":{"line":{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"barpolar"}],"
carpet":[{"aaxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"baxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"type":"carpet"}],"ch
oropleth":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"contour"}],"contourcarpet":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"heatmap":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmap"}],"histogram":[{"marker":{"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"histogram"}],
"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2d"}],"histogram2dcontour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2dcontour"}],"mesh3d":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"parcoords":[{"line":
{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"pie":
[{"automargin":true,"type":"pie"}],"scatter":[{"fillpattern":
{"fillmode":"overlay","size":10,"solidity":0.2},"type":"scatter"}],"sc

```
atter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":
{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}],"scattercarpet":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"scattergeo":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scattergl":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergl"}],"scattermap":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattermap"}],"scattermapbox":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scatterpolar"
:[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterpolargl
":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"scatterterna
ry":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"surface":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"surface"}],"table":[{"cells":{"fill":
{"color":"#EBF0F8"},"line":{"color":"white"}},"header":{"fill":
{"color":"#C8D4E3"},"line":
{"color":"white"}},"type":"table"}]},"layout":{"annotationdefaults":
{"arrowcolor":"#2a3f5f","arrowhead":0,"arrowwidth":1},"autotypenumbers
":"strict","coloraxis":{"colorbar":
{"outlinewidth":0,"ticks":""}},"colorscale":{"diverging":
[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],
[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],
[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]],"sequentialminus":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]]},"colorway":
["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692"
,"#B6E880","#FF97FF","#FECB52"],"font":{"color":"#2a3f5f"},"geo":
{"bgcolor":"white","lakecolor":"white","landcolor":"#E5ECF6","showlake
s":true,"showland":true,"subunitcolor":"white"},"hoverlabel":
```

```
{"align":"left"},"hovermode":"closest","mapbox":
{"style":"light"},"paper_bgcolor":"white","plot_bgcolor":"#E5ECF6","po
lar":{"angularaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","radialaxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"scene":
{"xaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"yaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"zaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
},"shapedefaults":{"line":{"color":"#2a3f5f"}},"ternary":{"aaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"baxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","caxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"title":
{"x":5.0e-2},"xaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2},"yaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2}}},"title":
{"text":"Training loss"}}}
```


6.1 RF

```
# Creamos y entrenamos el modelo de Random Forest con la función de
estructura de datos generada a través de la función Forecaster, en
este caso, creando ventanas de 10 días:
#
================================================================
========
forecaster = ForecasterAutoreg(
                regressor = RandomForestRegressor(random_state=123),
                lags = 15
                )

forecaster.fit(y=data_train['y'])
```

7. Empleando tus resultados, calcula las métricas para la medición de asertividad de un pronóstico, al menos debescalcular dos métricos de error de pronóstico, ejemplo: MAPE, RMSE, MAE, entre otros.

Validacion desempeño del modelo

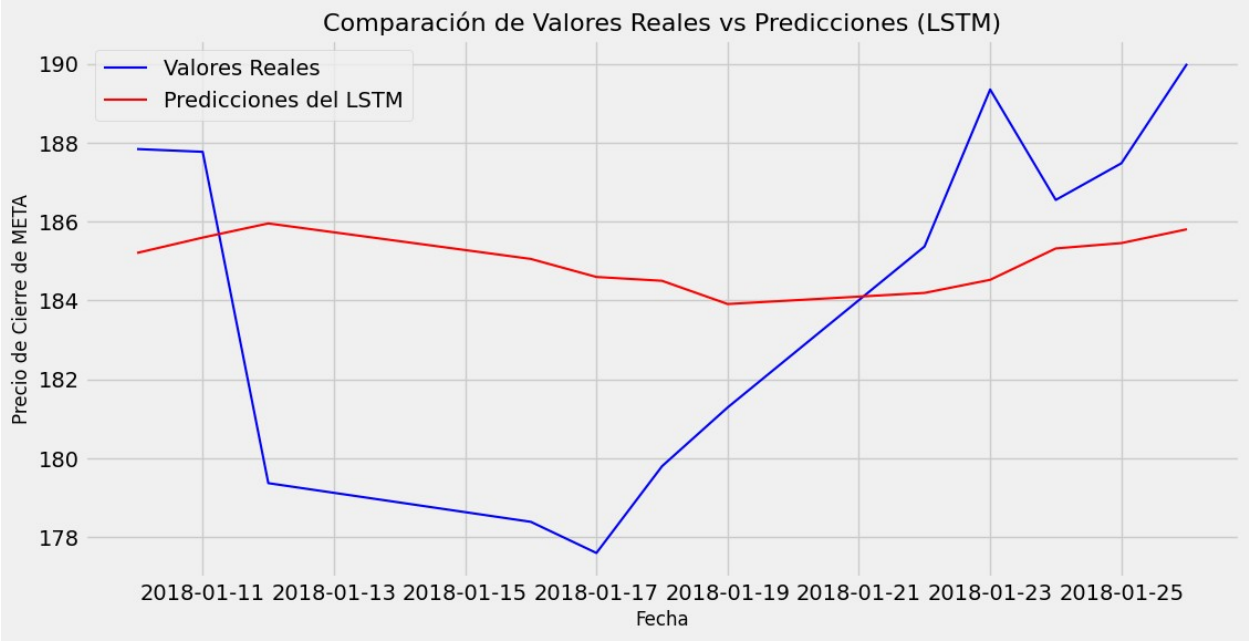
7.1 Validacion en conjunto de test con LSTM

```
# Extraer los valores reales del test set
input_data = close_data
output_data = close_data

y_true = output_data[split + n_back:]  # Para alinear con la longitud
de las predicciones

# Hacer las predicciones
y_pred = model.predict(test_generator)

# Calcular métricas
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
mae = mean_absolute_error(y_true, y_pred)
mape = mean_absolute_percentage_error(y_true, y_pred)

print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"MAPE: {mape * 100:.2f}%")
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
RMSE: 4.332173022692807
MAE: 3.818679504394533
MAPE: 2.09%
```

```
# Grafica los valores reales y las predicciones
plt.figure(figsize=(12, 6))
plt.plot(date_test[n_back:], y_true, label='Valores Reales',
color='blue')  # Valores reales
plt.plot(date_test[n_back:], y_pred, label='Predicciones del LSTM',
color='red')  # Predicciones

# Añadir título y etiquetas
plt.title('Comparación de Valores Reales vs Predicciones (LSTM)',
fontsize=16)
plt.xlabel('Fecha', fontsize=12)
plt.ylabel('Precio de Cierre de META', fontsize=12)
plt.legend()

# Mostrar la gráfica
```

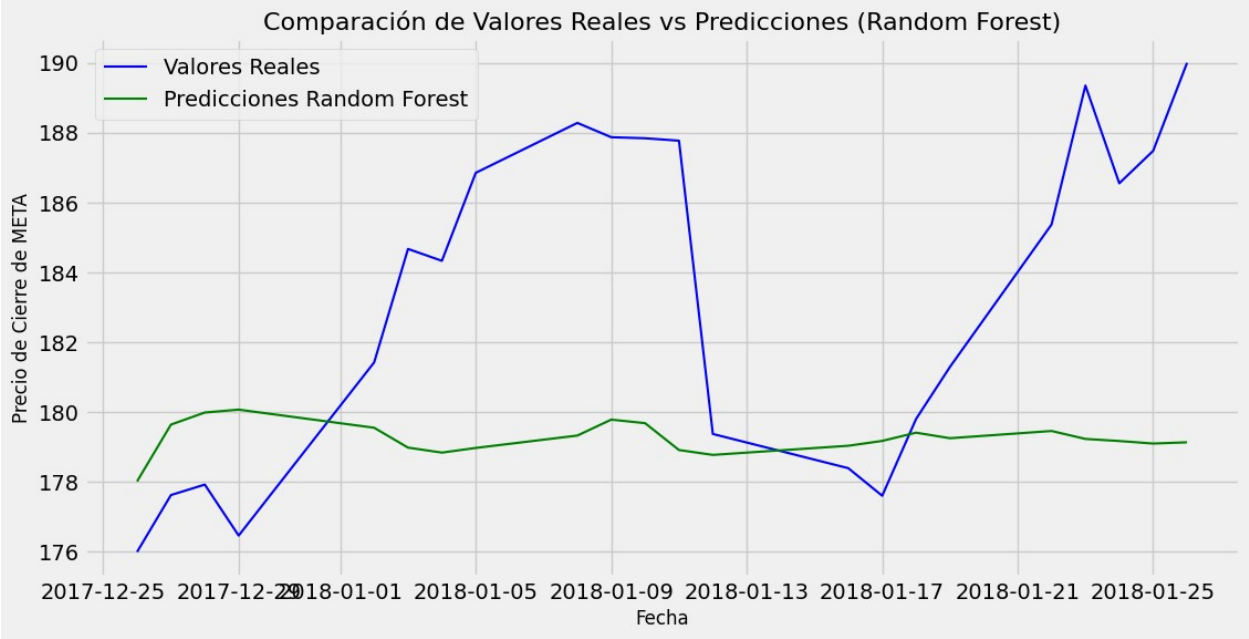
```
plt.grid(True)
plt.show()
```



7.1 Validacion en conjunto de test con RF

```
# Determinamos el error de pronóstico:
# Pronóstico, en este caso elegimos una ventana de n-test días.
#
# ============================================================================
# ========
steps = steps  ## el mismo que la ventana de test
predictions = forecaster.predict(steps=steps)

#
# ============================================================================
# ========
# Calcula métricas
rmse = sqrt(mean_squared_error(data_test['y'], predictions))
mae = mean_absolute_error(data_test['y'], predictions)
mape = mean_absolute_percentage_error(data_test['y'], predictions) *
100  # en porcentaje

# Muestra resultados
print(f"Test error (RMSE): {rmse}")
print(f"Test error (MAE): {mae}")
print(f"Test error (MAPE): {mape}%")

Test error (RMSE): 6.15054444921891
Test error (MAE): 5.119609090909063
Test error (MAPE): 2.751584137094789%
```

```
# Grafica los valores reales y las predicciones
plt.figure(figsize=(12, 6))

# Graficar los valores reales de la serie de tiempo
plt.plot(date_test, data_test['y'], label='Valores Reales',
color='blue')

# Graficar las predicciones del modelo
plt.plot(date_test, predictions, label='Predicciones Random Forest',
color='green')

# Añadir título y etiquetas
plt.title('Comparación de Valores Reales vs Predicciones (Random
Forest)', fontsize=16)
plt.xlabel('Fecha', fontsize=12)
plt.ylabel('Precio de Cierre de META', fontsize=12)

# Mostrar la leyenda
plt.legend()

# Mostrar la gráfica
plt.grid(True)
plt.show()
```



PASO 2:Comparación de los modelos y selección del mejor

1. Compara y sustenta qué modelo presentó un mejor desempeño de acuerdo a los criterios de selección pertinentes.

Modelo	RMSE	MAE	MAPE (%)
MARS	19.45	18.77	10.14
LSTM	4.33	3.82	2.09
Random Forest	6.15	5.12	2.75
SSA	5.48	4.69	2.59

En base a las métricas obtenidas, el modelo LSTM muestra el mejor desempeño en términos de RMSE, MAE y MAPE, superando a los demás modelos con un RMSE de 4.33, MAE de 3.82 y un MAPE de 2.09%. Este desempeño es notablemente mejor en comparación con el modelo MARS (RMSE: 19.45, MAPE: 10.14%) y Random Forest (RMSE: 6.15, MAPE: 2.75%). Sin embargo, el modelo SSA también tiene buenos resultados con un MAPE de 2.59%, aunque sigue siendo menos preciso que el LSTM. Estos resultados sugieren que los modelos basados en redes neuronales, como el LSTM, pueden capturar mejor las complejas relaciones temporales en los datos de precios de acciones, mientras que los modelos más simples, como MARS y Random Forest, no logran igualar la precisión

2. Genera un pronóstico de 10 días hacia adelante y una visualización del mismo

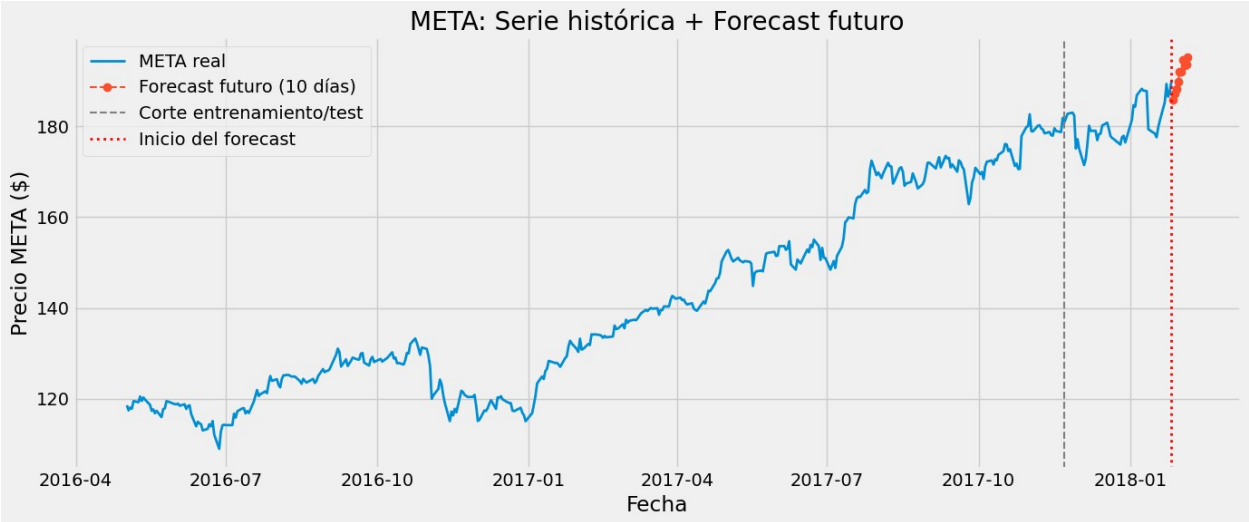
```
input_data = close_data
output_data = close_data

def optimize_lstm_univariate(input_data, output_data, dataindex,
                             n_back=10, split_percent=0.9,
                             batch_size=32, max_trials=10, epochs=25,
                             fast_mode=False):

    if fast_mode:
        max_trials = 2
        epochs = 15
        print(" MODO RÁPIDO ACTIVADO: max_trials=2, epochs=15")

    split = int(split_percent * len(output_data))

    # Generadores
    train_generator = TimeseriesGenerator(input_data[:split],
output_data[:split],
                                          length=n_back,
batch_size=batch_size)
    test_generator = TimeseriesGenerator(input_data[split:],
output_data[split:],
                                         length=n_back, batch_size=1)
```

```
    def build_model(hp):
        model = Sequential()
        units = hp.Int('units', min_value=32, max_value=300 if
fast_mode else 256, step=32)
        dropout_rate = hp.Float('dropout', min_value=0.1,
max_value=0.3 if fast_mode else 0.5, step=0.1)

        model.add(LSTM(units, activation='relu',
return_sequences=True, input_shape=(n_back, 1)))
        model.add(Dropout(dropout_rate))
        model.add(LSTM(units // 2, activation='relu'))
        model.add(Dropout(dropout_rate))
        model.add(Dense(1))  # Predicción univariada

        optimizer_choice = hp.Choice('optimizer', ['adam', 'rmsprop'])
        optimizer = Adam() if optimizer_choice == 'adam' else
RMSprop()

        model.compile(optimizer=optimizer, loss='mse')
        return model

    tuner = kt.RandomSearch(
        build_model,
        objective='val_loss',
        max_trials=max_trials,
        executions_per_trial=1,
        directory='lstm_tuning',
        project_name='amzn_univariate'
    )

    tuner.search(train_generator, validation_data=test_generator,
epochs=epochs, verbose=1)

    best_model = tuner.get_best_models(num_models=1)[0]
    best_hp = tuner.get_best_hyperparameters(1)[0]

    print(" Mejores hiperparámetros encontrados:")
    for k, v in best_hp.values.items():
        print(f"  {k}: {v}")

    early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
    history = best_model.fit(
        train_generator,
        validation_data=test_generator,
        epochs=50 if not fast_mode else 10,
        callbacks=[early_stop],
        verbose=1
    )
```

```
    return best_model, best_hp

best_model, best_hp = optimize_lstm_univariate(
    input_data=input_data,
    output_data=output_data,
    dataindex=dataindex,
    n_back=10,
    batch_size=50,  ## si no hay tiempo, pueden bajar este parámetro
    max_trials=10,
    epochs=10,
    fast_mode=True  # Puedes cambiar a True para búsqueda completa
)

close_data = dt2.reshape(-1)  # <--- esto es clave, si no, el
pronóstico queda achatado, en otras dimensiones que le dió el reshape.

# Reconstruir el modelo optimizado en el modo FAST--
rebuilt_model = build_model(best_hp)
rebuilt_model.set_weights(best_model.get_weights())
_ = rebuilt_model.predict(np.zeros((1, n_back, 1), dtype=np.float32))


# 3] Asegurar que close_data esté en 1D ---
close_data = dt2.reshape(-1)  # ¡aquí va lo que preguntaste!

# 4] Definir predict() ---
def predict(num_prediction, model):
    prediction_list = close_data[-n_back:]
    for _ in range(num_prediction):
        x = prediction_list[-n_back:]
        x = x.reshape((1, n_back, 1)).astype(np.float32)
        out = model.predict(x, verbose=0)[0][0]
        prediction_list = np.append(prediction_list, out)
    return prediction_list[n_back-1:]

# --- [5] Generar forecast y graficar ---
forecast = predict(10, rebuilt_model)

def predict_dates(num_prediction):
    last_date = dataindex[-1]
    return pd.date_range(start=last_date + pd.Timedelta(days=1),
periods=num_prediction)

forecast_dates = predict_dates(10)

#Fechas, 10 días pegado al histórico:
def predict(num_prediction, model):
    prediction_list = close_data[-n_back:]
```

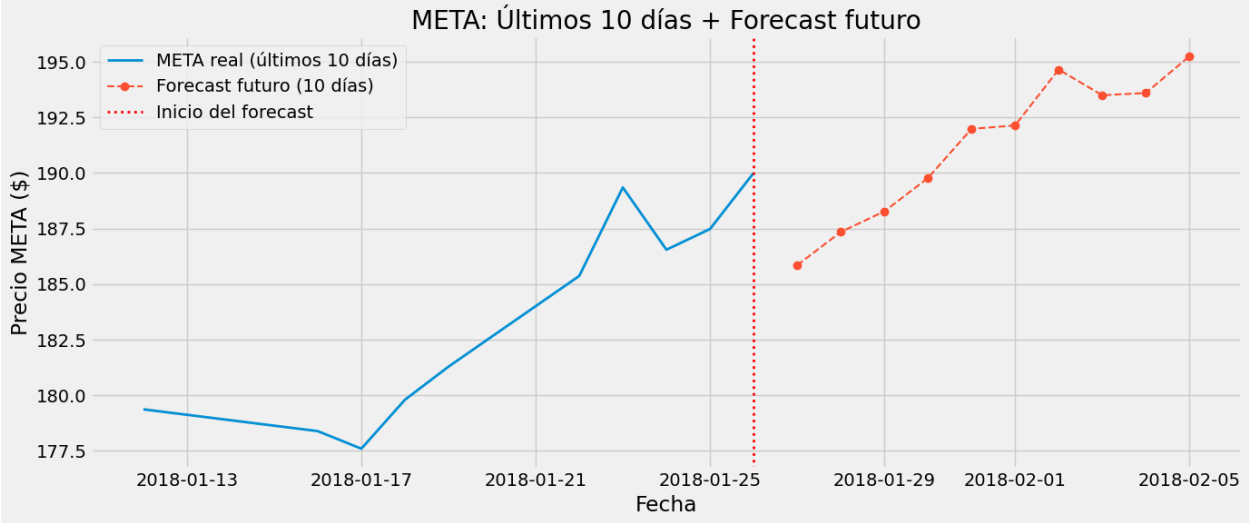
```
    for _ in range(num_prediction):
        x = prediction_list[-n_back:]
        x = x.reshape((1, n_back, 1)).astype(np.float32)
        out = model.predict(x, verbose=0)[0][0]
        prediction_list = np.append(prediction_list, out)

    return prediction_list[n_back-1:]  # solo forecast (sin la parte
usada como entrada)

num_prediction = 10
forecast = predict(num_prediction, rebuilt_model)


 MODO RÁPIDO ACTIVADO: max_trials=2, epochs=15
Reloading Tuner from lstm_tuning/amzn_univariate/tuner0.json
 Mejores hiperparámetros encontrados:
  units: 64
  dropout: 0.1
  optimizer: adam
Epoch 1/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 1052.0535 - val_loss:
232.3014
Epoch 2/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1013.2159 - val_loss:
36.4303
Epoch 3/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 886.8623 - val_loss:
144.0780
Epoch 4/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 723.8525 - val_loss:
39.5540
Epoch 5/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 669.3236 - val_loss:
50.2292
Epoch 6/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 703.8141 - val_loss:
479.4458
Epoch 7/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 783.4880 - val_loss:
71.2362
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step

#Fecha de corte (80%)
split = int(0.9 * len(close_data))

# Asegurar que `forecast` tenga la longitud correcta (10 en este caso)
forecast = forecast[-10:]  # Ajustar la longitud de forecast

# Gráfico
plt.figure(figsize=(14, 6))
```

```python
# 1. Serie histórica
plt.plot(dataindex, close_data, label='META real', linewidth=2)

# 2. Forecast futuro
plt.plot(forecast_dates, forecast, label='Forecast futuro (10 días)',
linestyle='--', marker='o')

# 3. Línea vertical: corte entre entrenamiento y test
plt.axvline(x=dataindex[split], color='gray', linestyle='--',
linewidth=1.5, label='Corte entrenamiento/test')

# 4. Línea vertical: inicio del forecast (último día conocido)
plt.axvline(x=dataindex[-1], color='red', linestyle=':', linewidth=2,
label='Inicio del forecast')

# Estética
plt.title('META: Serie histórica + Forecast futuro')
plt.xlabel('Fecha')
plt.ylabel('Precio META ($)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```python
# Últimos 10 días históricos
n_recent = 10
recent_dates = dataindex[-n_recent:]
recent_values = close_data[-n_recent:]

# Asegurar forecast tenga longitud 10
forecast = forecast[-10:]  # por si tiene más
```

```
forecast_dates = forecast_dates[:10]  # asegurar que coincide

# Gráfico
plt.figure(figsize=(14, 6))

# 1. Últimos 30 días reales
plt.plot(recent_dates, recent_values, label='META real (últimos 10
días)', linewidth=2)

# 2. Forecast futuro
plt.plot(forecast_dates, forecast, label='Forecast futuro (10 días)',
linestyle='--', marker='o')

# 3. Línea vertical: inicio del forecast
plt.axvline(x=dataindex[-1], color='red', linestyle=':', linewidth=2,
label='Inicio del forecast')

# Estética
plt.title('META: Últimos 10 días + Forecast futuro')
plt.xlabel('Fecha')
plt.ylabel('Precio META ($)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



ANEXOS

En este primer ANEXO se incluye los modelos MARS y SSA

Librerias

```
# Manipulación y análisis de datos
library(tidyverse)  # Incluye dplyr, ggplot2, readr, etc.
library(dplyr)      # Incluido en tidyverse, pero lo puedes dejar explícito si quieres
library(fields)      # Manipulación de datos espaciales

# Modelado
library(earth)       # Modelos MARS
library(mgcv)        # Modelos GAM
library(caret)       # Machine learning

# Series de tiempo
library(tseries)     # Análisis de series de tiempo
library(tsfeatures)  # Extracción de features de series de tiempo
library(forecast)    # Modelado y pronóstico de series de tiempo
library(quantmod)    # Datos financieros
library(xts)         # Objetos de series de tiempo extensibles
library(Rssa)        # Singular Spectrum Analysis
library(TTR)         # Technical Trading Rules
# library(TSA)       # Si necesitas TSA, descomenta

# Visualización
library(ggplot2)     # Gráficos (ya incluido en tidyverse)
library(dygraphs)    # Visualización interactiva de series de tiempo

# Métricas y rendimiento
library(Metrics)               # Métricas de error
library(yardstick)

# Configuración global
theme_set(theme_bw())
options(warn = -1)

# Exploración inicial
list.files(path = "../input")
```

```
Warning message:
“package ‘ggplot2’ was built under R version 4.2.3”
Warning message:
“package ‘tidyr’ was built under R version 4.2.3”
```

```
Warning message:
“package ‘readr’ was built under R version 4.2.3”
Warning message:
“package ‘dplyr’ was built under R version 4.2.3”
Warning message:
“package ‘stringr’ was built under R version 4.2.3”
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.1
✔ purrr     1.0.4
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
Loading required package: spam

Spam version 2.10-0 (2023-10-23) is loaded.
Type 'help( Spam)' or 'demo( spam)' for a short introduction
and overview of this package.
Help for individual functions is also obtained by adding the
suffix '.spam' to the function name, e.g. 'help( chol.spam)'.


Attaching package: ‘spam’


The following objects are masked from ‘package:base’:

    backsolve, forwardsolve


Loading required package: viridisLite


Try help(fields) to get started.

Warning message:
“package ‘earth’ was built under R version 4.2.3”
Loading required package: Formula

Loading required package: plotmo

Warning message:
“package ‘plotmo’ was built under R version 4.2.3”
Loading required package: plotrix
```

```
Attaching package: ‘plotrix’

The following object is masked from ‘package:fields’:

    color.scale

Loading required package: nlme

Attaching package: ‘nlme’

The following object is masked from ‘package:dplyr’:

    collapse

This is mgcv 1.8-41. For overview type 'help("mgcv-package")'.

Loading required package: lattice

Attaching package: ‘caret’

The following object is masked from ‘package:purrr’:

    lift


Warning message:
“package ‘tseries’ was built under R version 4.2.3”
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo

Warning message:
“package ‘forecast’ was built under R version 4.2.3”

Attaching package: ‘forecast’


The following object is masked from ‘package:nlme’:

    getResponse


Warning message:
“package ‘quantmod’ was built under R version 4.2.3”
```

```
Loading required package: xts

Loading required package: zoo


Attaching package: ‘zoo’


The following objects are masked from ‘package:base’:

    as.Date, as.Date.numeric



######################### Warning from 'xts' package
##########################
#
#
# The dplyr lag() function breaks how base R's lag() function is
supposed to  #
# work, which breaks lag(my_xts). Calls to lag(my_xts) that you type
or       #
# source() into this session won't work correctly.
#
#
#
# Use stats::lag() to make sure you're not using dplyr::lag(), or you
can add #
# conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop
#
# dplyr from breaking base R's lag() function.
#
#
#
# Code in packages is not affected. It's protected by R's namespace
mechanism #
# Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this
warning.  #
#
#
##############################################################################
#########


Attaching package: ‘xts’


The following object is masked from ‘package:fields’:

    addLegend
```

```
The following objects are masked from ‘package:dplyr’:

    first, last


Loading required package: TTR

Warning message:
“package ‘TTR’ was built under R version 4.2.3”
Loading required package: svd

Warning message:
“package ‘svd’ was built under R version 4.2.3”

Attaching package: ‘Rssa’


The following object is masked from ‘package:plotrix’:

    clplot


The following object is masked from ‘package:spam’:

    cleanup


The following object is masked from ‘package:stats’:

    decompose



Attaching package: ‘Metrics’


The following object is masked from ‘package:forecast’:

    accuracy


The following objects are masked from ‘package:caret’:

    precision, recall



Attaching package: ‘yardstick’


The following objects are masked from ‘package:Metrics’:
```

```
    accuracy, mae, mape, mase, precision, recall, rmse, smape

The following object is masked from 'package:forecast':

    accuracy

The following objects are masked from 'package:caret':

    precision, recall, sensitivity, specificity

The following object is masked from 'package:readr':

    spec


character(0)
```

PASO 1:Aplicación de los modelos

1. Descarga de Yahoo Finance la serie de precios de cierre del activo META en las fechas: 01-05-2016 al 28-01-2018.

```
###Función para obtener datos:
start<-format(as.Date("2016-05-01"),"%Y-%m-%d")
end<-format(as.Date("2018-01-28"),"%Y-%m-%d")

precios <-function(simbolo)
{
  ##Obtener precios stocks de Yahoo FInance
  datos <- getSymbols(simbolo, auto.assign = FALSE, from=start,
to=end)
  ## Elimar faltantes:
  datos<-na.omit(datos)
  ##mantener columnas con precios cierre 4:
  datos <- datos[,4]
  ##Para hacerlo datos accesibles en el global environment:
  assign(simbolo, datos, envir = .GlobalEnv)
}
##Llamar el activo de interés, pueden ser varios:
precios("META")
```

2. Grafica la serie

```
##Juntar los datos y renombrarlos:
Pr<-merge.xts(META) %>% na.omit()
colnames(Pr) <- c("META")
##Serie tiempo, grafica interactiva
Precios<- dygraph(Pr[,c(1)], main="Precios ") %>%
  dyAxis("y", label = "Precios") %>%
  dyRangeSelector(dateWindow = c("2016-05-01", "2018-01-28"))%>%
  dyOptions(colors = RColorBrewer::brewer.pal(4,"Set1"))
Precios
```

```
HTML widgets cannot be represented in plain text (need html)
```

3. Prepara los datos a manera que cuentes con un formato adecuado para la fecha.

```
##Index format.
xpr <- as.xts(Pr, dateFormat = "Date")
Pr_index <- fortify.zoo(xpr)
nrow(Pr)
```

```
[1] 439
```

4. Parte la serie a un 5% de prueba y el restante para el entrenamiento.

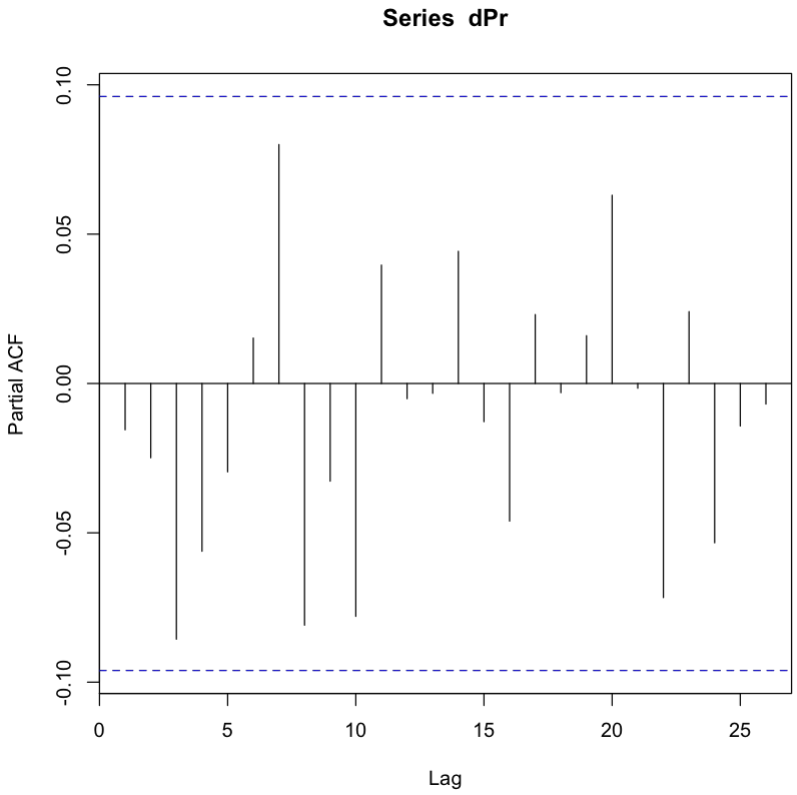
```
#Partimos serie para train & test
h <- round(length(Pr)*0.05, digits = 0 )
Pr_tra <- Pr[1:(nrow(Pr) - h), ]
Pr_tes<- Pr[(nrow(Pr) - h + 1):nrow(Pr), ]

#Pasamos a formato data frame para poder generar los retrasos
Pr_df_tra<-as.data.frame(Pr_tra)
```

5. Realiza el preprocesamiento necesario para aplicar cada uno de los modelos

5.1 Preprocesamiento MARS

```
#Generación de features o rezagos y obtención base datos
#Obteneomos features:
tsfeatures(Pr_df_tra$META)
pacf_features(Pr_df_tra$META)
#Visualiamos los rezagos autorregresivos, haciendo estacionaria la
serie:
dPr<-diff(Pr_df_tra$META)%>% na.omit() # Train section.
pacf(dPr)
```

```
  frequency nperiods seasonal_period trend     spike        linearity
curvature
1 1         0        1               0.9901344 1.737658e-09 19.22551
4.38829
  e_acf1    e_acf10  entropy    x_acf1    x_acf10  diff1_acf1
diff1_acf10
1 0.7349023 1.041272 0.09337241 0.9927452 9.236512 -0.01550977
0.03439839
  diff2_acf1 diff2_acf10
1 -0.4949165 0.2821558

     x_pacf5 diff1x_pacf5 diff2x_pacf5
  0.98585140   0.01221966   0.48969533
```




```
#Generamos retrasos
lag_pr = lag(Pr_df_tra,n=3L)
lag2_pr = lag(Pr_df_tra,n=7L)
lag3_pr = lag(Pr_df_tra,n=8L)
lag4_pr = lag(Pr_df_tra,n=10L)

#Unimos los nuevos features o retrasos en una base
Pr_all_train<-cbind(Pr_df_tra, lag_pr, lag2_pr, lag3_pr, lag4_pr) %>
%na.omit()
```

```
colnames(Pr_all_train)<-c("META","l3","l7","l8","l10")
head(Pr_all_train)
```

```
           META   l3     l7     l8     l10
2016-05-16 118.67 119.52 117.81 118.06 118.57
2016-05-17 117.35 120.28 119.49 117.81 117.43
2016-05-18 117.65 119.81 119.24 119.49 118.06
2016-05-19 116.81 118.67 120.50 119.24 117.81
2016-05-20 117.35 117.35 119.52 120.50 119.49
2016-05-23 115.97 117.65 120.28 119.52 119.24
```

```
Pr_all_train$trend = 1:nrow(Pr_all_train)
#Separamos objetos para las secciones de entrenamiento con todos los
features X_train:
x_train <- Pr_all_train %>%
  select(starts_with(c("l", "t")))
#Igual para la respuesta y_train:
y_train <- Pr_all_train %>%
  select(META)
```

5.2 Preprocesamiento SSA

```
# Un paso importante que hay que hacer, es pasar los datos de formato
xts a ts, que es el formato que emplea SSA. para ello, pasamos primero
a data frame
# y luego a ts.
META <- data.frame(date=index(META), coredata(META))
META1<-META$META
##pasamos a ts
META.ts<-ts(META1, start=c(2016,122), frequency = 365)

## Preprocesamiento: Partición de datos en conjunto de prueba y de
entrenamiento.
## Podemos partir la serie en a manera de obtener 36 datos para la
prueba.
h <- round(length(META.ts)*0.05, digits = 0)

#Una vez creada la ventana h, generamos los conjuntos de entrenamiento
y prueba.
train<-META.ts[1:(length(META.ts)-h)]
test<-META.ts[(length(META.ts)-h+1):length(META.ts)]
train.ts=ts(train,start=c(2016,05), frequency = 365)
```

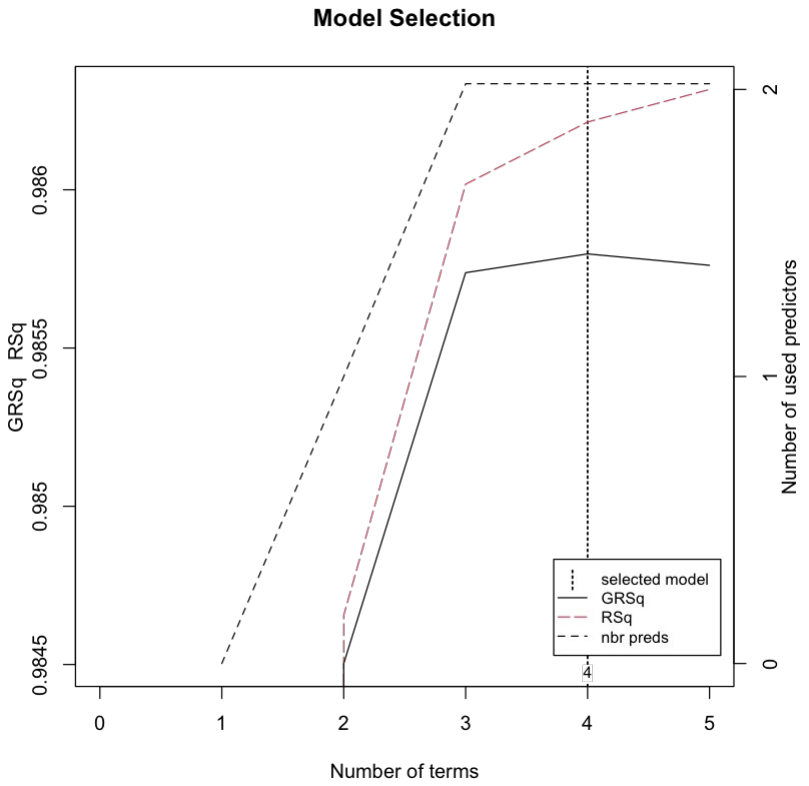
6. Aplica cada uno de los modelos MARS, LSTM, RF y SSA a la serie de META

6.1 MARS

```
mars_mod <- earth(
  META ~ ., data=Pr_all_train, pmethod="backward" )
# Podemos ver el resultado del modelo:
print(mars_mod)
```

```
Selected 4 of 5 terms, and 2 of 5 predictors
Termination condition: RSq changed by less than 0.001 at 5 terms
Importance: l3, trend, l7-unused, l8-unused, l10-unused
Number of terms at each degree of interaction: 1 3 (additive model)
GCV 6.754156    RSS 2655.196    GRSq 0.9857974    RSq 0.9862141
```

```
plot(mars_mod, which = 1)
```



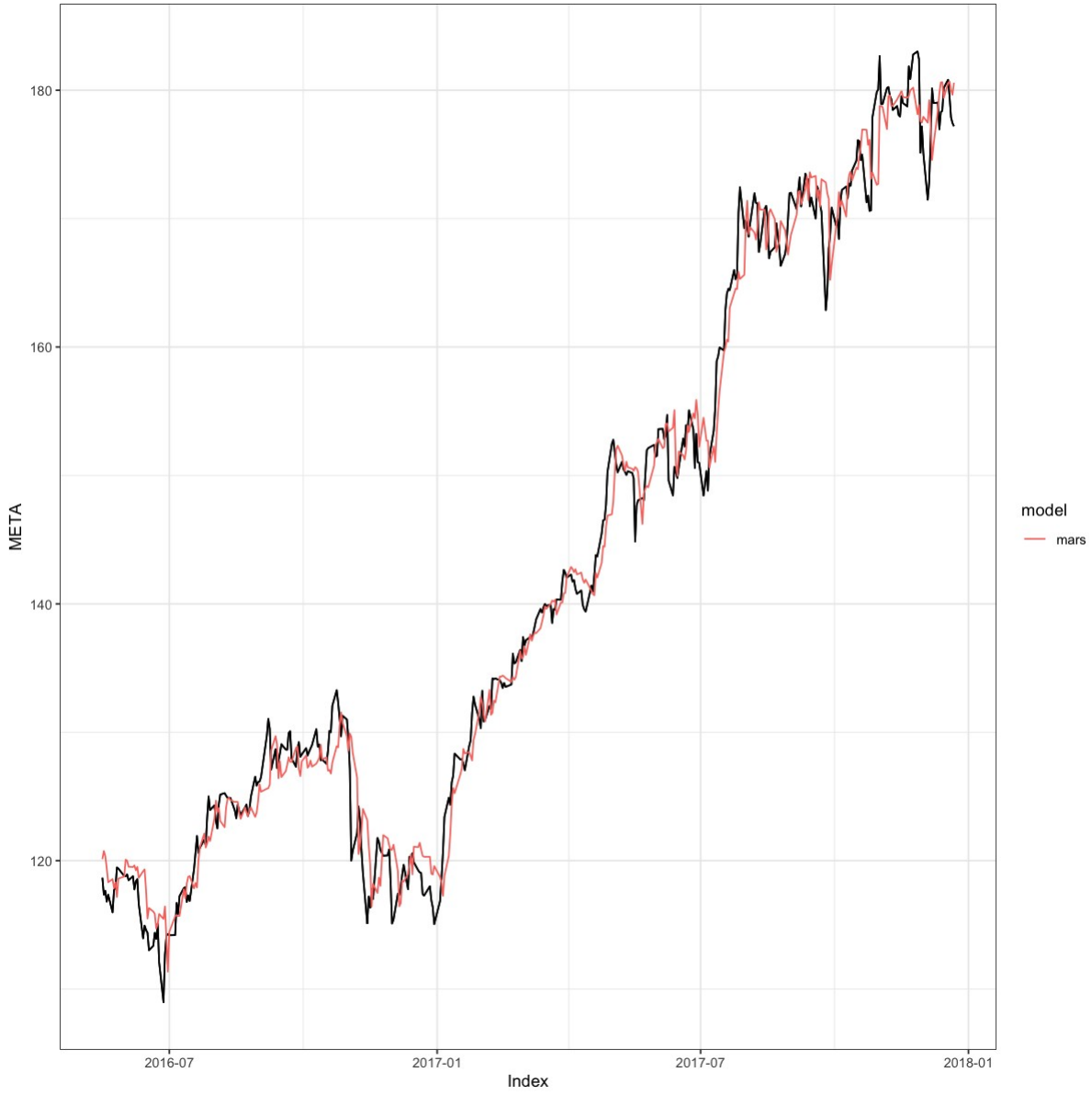

```
# Unimos las base junto con los resultados de ajuste
Pr_train <- Pr_all_train %>%
  mutate(
    fitted_mars = fitted(mars_mod)[,1]
  )

#Damos formato de xts para poder manipular los datos y graficarlos con ggplot más adelante
x <- as.xts(Pr_train, dateFormat = "Date")
Pr_train2 <- fortify.zoo(x)
```

```
# Podemos dar formato para que el etiquetado en la visualización sea
más sencilla.
Pr_train3 <- Pr_train2 %>%
  select(Index, META, starts_with("fitted")) %>%
  gather("model", "value", -Index, -META) %>%
  mutate(model = str_remove(model, "fitted_"))
tail(Pr_train3)
```

```
    Index      META   model value
402 2017-12-15 180.18 mars  179.3978
403 2017-12-18 180.82 mars  180.5558
404 2017-12-19 179.51 mars  180.6754
405 2017-12-20 177.89 mars  180.0363
406 2017-12-21 177.45 mars  179.6480
407 2017-12-22 177.20 mars  180.5794
```

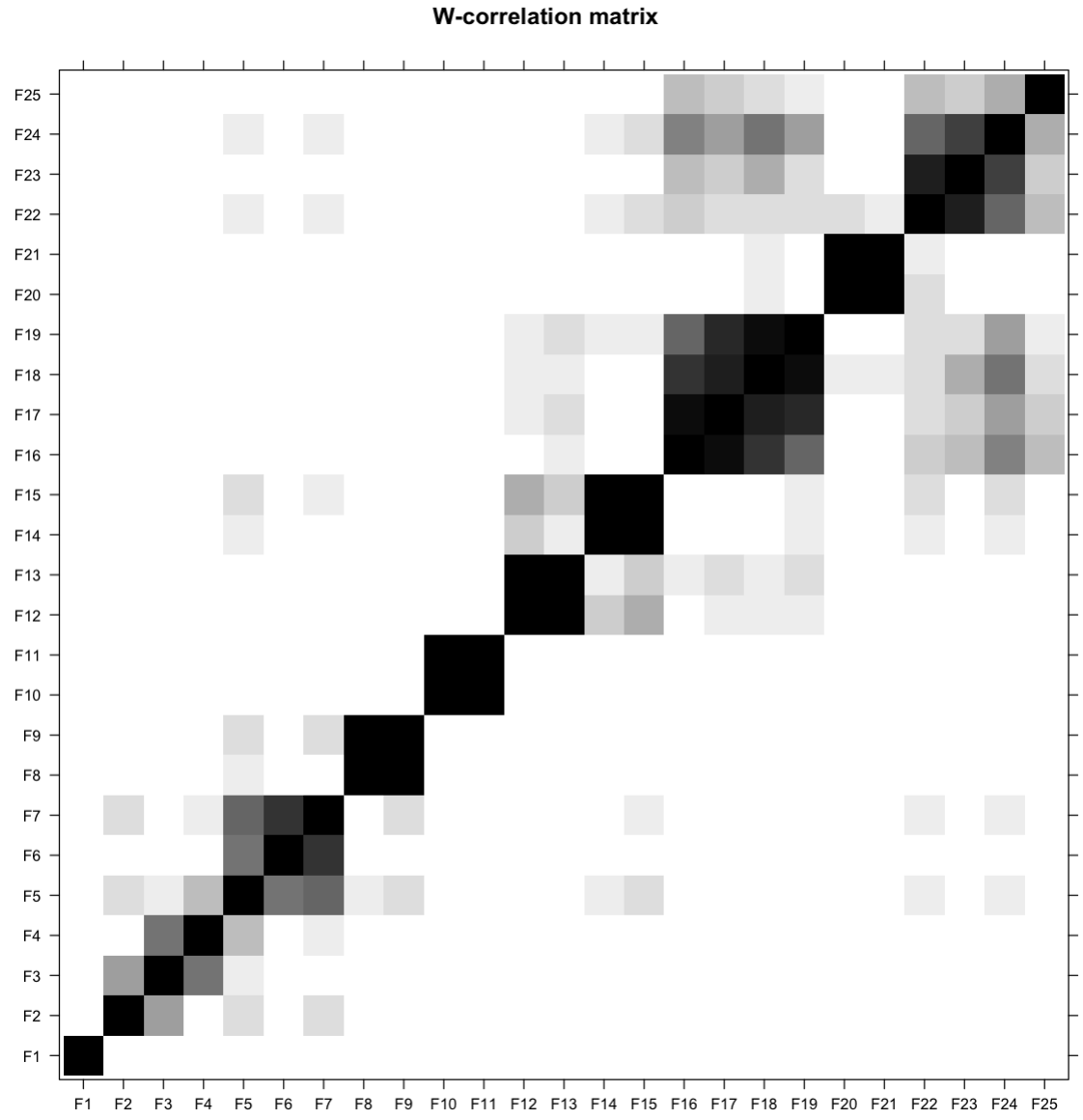
Graficas del modelo MARS

```
## Graficamos:
library(ggplot2)
options(repr.plot.width = 10, repr.plot.height = 10)
ggplot(Pr_train3,aes(Index,META))+geom_line() +
geom_line()+
geom_line(aes(y=value, color=model))
#theme(plot.title = element_text(size=30), legend.position="upper",
cex=100)
```



6.2 SSA

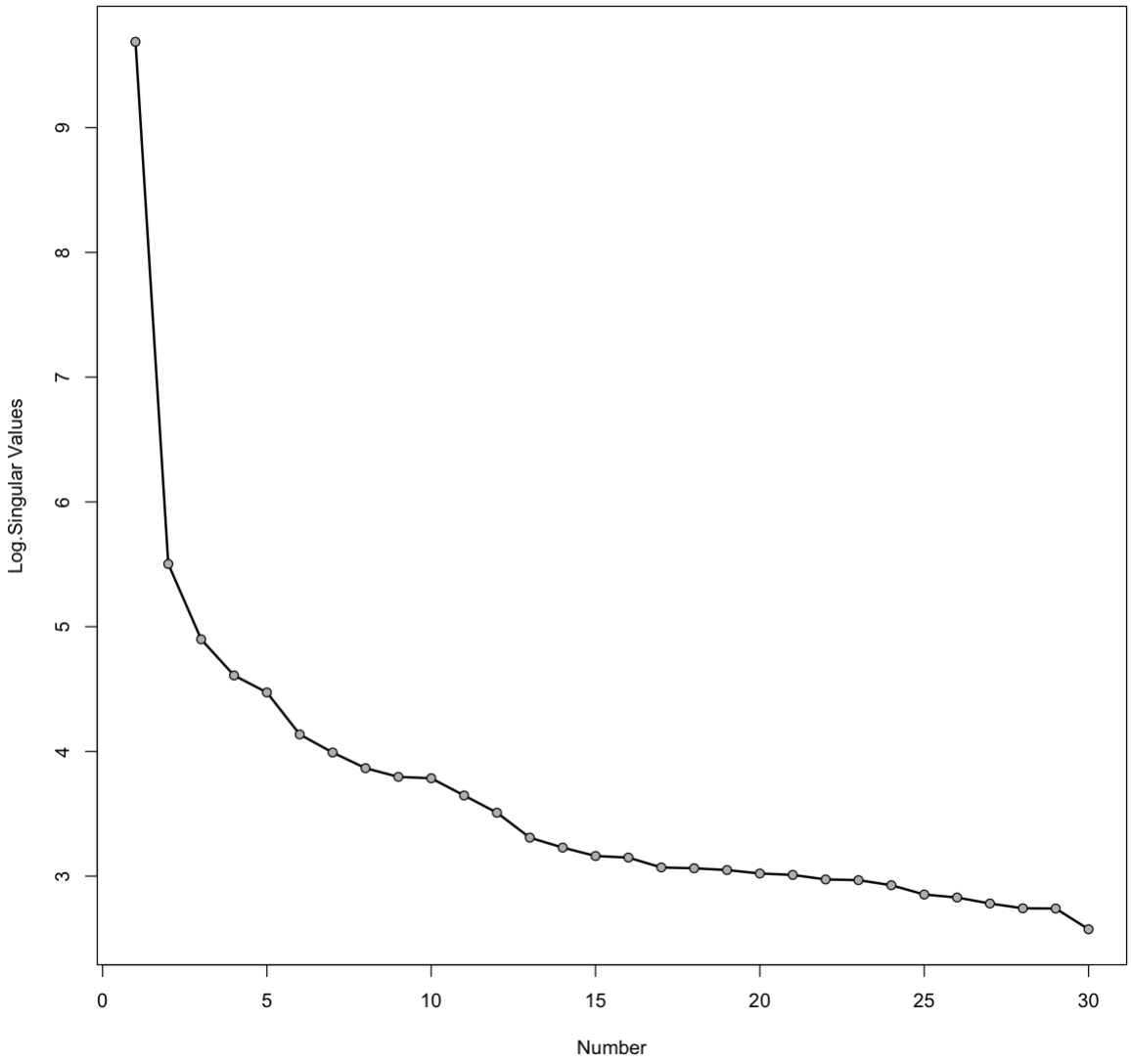
```
#En estos pasos, aplicamos la función de ssa() a la serie en formato
ts y procedemos a la selección para la recosntrucción.
##SSA embeding y separación:
s1<-ssa(train.ts)  #Hay otros métodos como el "topelitz", para series
estacionarias. En este caso no aplica.
##Reconstrucción: empleamos las herramientas visuales para la
selección.
wplot<-wcor(s1, group=1:25)
plot(wplot)
```




```
##Hankelización:
UniHankel=function(Y,L){
  k<-length(Y)-L+1
  outer((1:L), (1:k), function(x,y) Y[(x+y-1)])
}

##Función SVD
SVD<- function(Y,L){
  X<-UniHankel(Y,L)
  svd(X)
}

###W-correlation
W.corr<-function(Yt,L, groups){
```

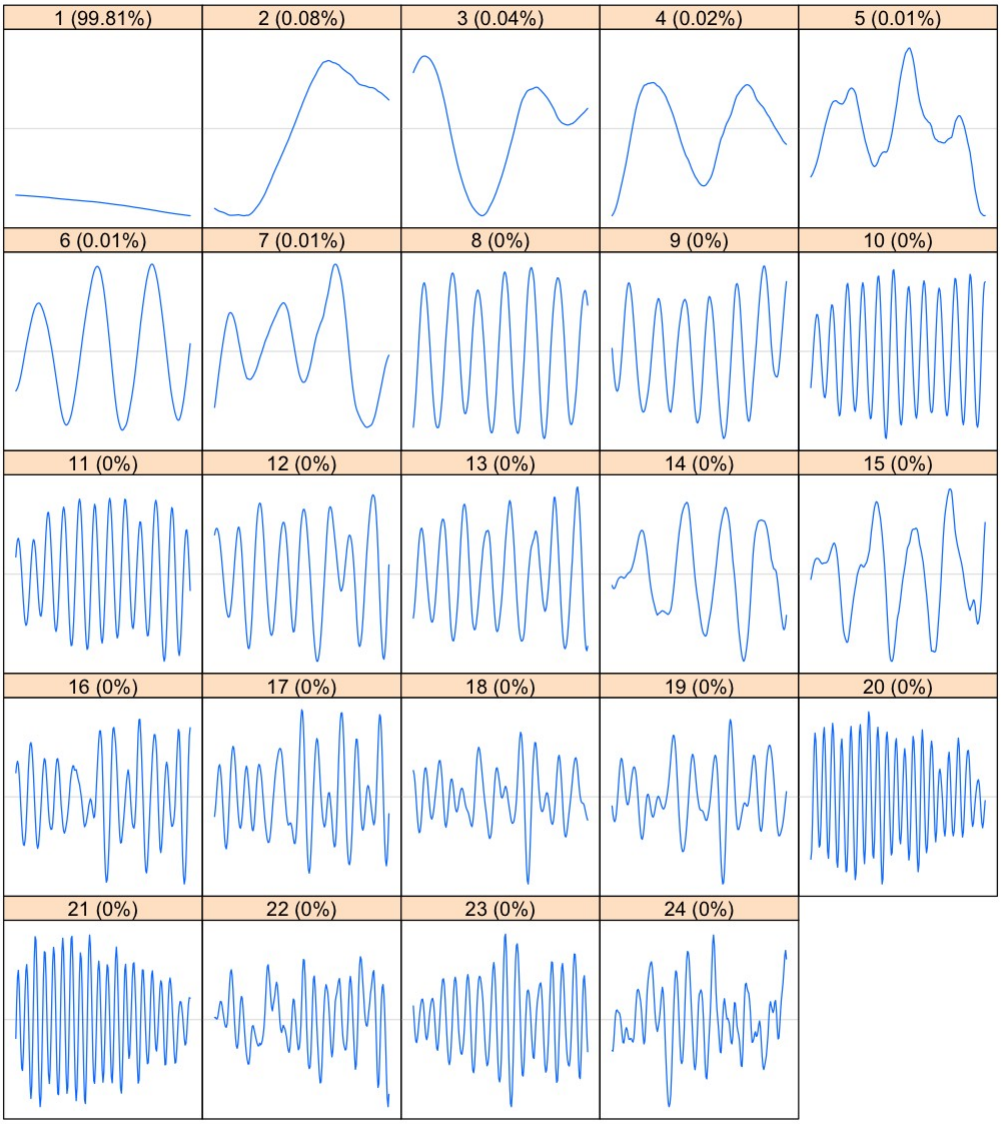
```
  m<-length(groups); w.corr<-diag(m)
  N<-length(Yt)
  w<-((N+1)-abs((N+1)/2-L)-abs((N+1)/2-1:N)-
    abs(abs((N+1)/2-L)-abs((N+1)/2-1:N)))/2
wcorr<-function(i,j){
  Y1<-SSA.Rec(Yt,L,groups[[i]])$Approximation
  Y2<-SSA.Rec(Yt,L,groups[[j]])$Approximation
  sum(w*Y1*Y2)/sqrt(sum(w*Y1^2)*sum(w*Y2^2))}
for (i in 1:(m-1)){
  for (j in (i+1):m){
    w.corr[i,j]=w.corr[j,i]=wcorr(i,j)}}
rownames(w.corr)<-colnames(w.corr)<-groups
w.corr
}

###Plotting Images w.corr
Plt.Img<-function(x){
  min<-min(x)
  max<-max(x)
  yLabels<-rownames(x)
  xLabels<-colnames(x)
  if( is.null (xLabels)){
    xLabels <- c(1:ncol(x))
  }
  if (is.null(yLabels)){
    yLabels <- c(1:nrow(x))
  }
  layout(matrix(data=c(1,2), nrow=1, ncol=2),
         widths=c(4,1), heights=c(1,1))
  ColorRamp<-gray( seq(1,0, length=20))
  ColorLevels<-seq(min, max, length=length(ColorRamp))
                  par(mar=c(3,5,2.5,2))
                  image(1:length(xLabels), 1:length(yLabels),
                        t(x), col=colorRamp, xlab="",
                        ylab = "", axes=FALSE, zlim=c(min, max))
                  title(main=c("Image Plot"))
                  axis(BELOW<-1, at=1:length(xLabels),
                       labels=xLabels, cex.axis=0.7)
                  axis(LEFT<-2, at=1:length(yLabels),
                       labels=yLabels, las=HORIZONTAL<-1,
                       cex.axis=0.7)
                  box()
                  par(mar=c(3,2.5, 2.5,2))
                  image(1, ColorLevels,
                        matrix(data=ColorLevels,
ncol=length(ColorLevels), nrow=1),
                        col=colorRamp,
                        xlab="", ylab="",
                        xaxt="n")
```

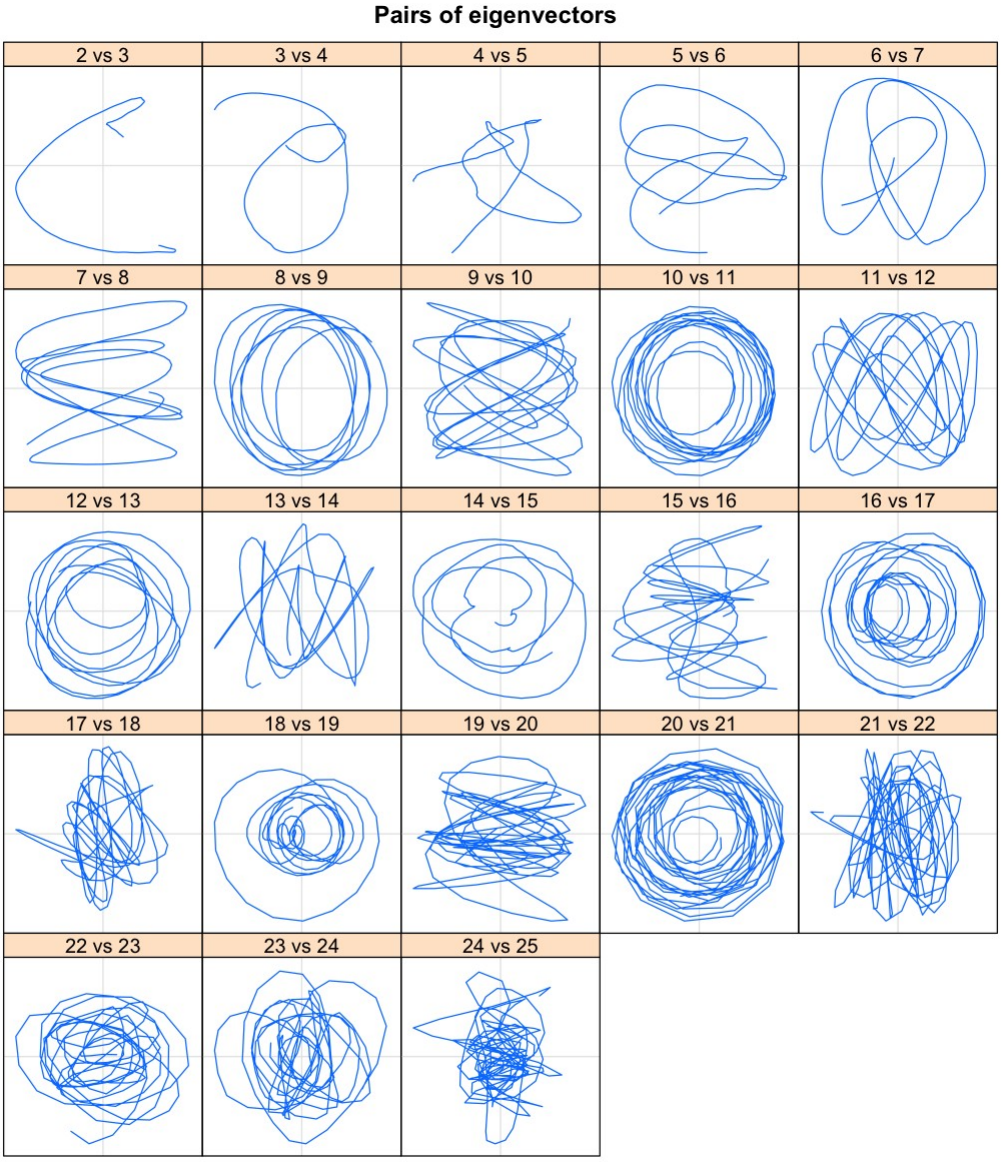
```
                  layout(1)
}

##Singular Values graph
Sing.plt<-function(Y,L){
  lambda<-log(SVD(Y,L)$d)
  d<-length(lambda)
  win<-1:d
  plot.new()
  plot.window(xlim=range(win), ylim=range(lambda))
  usr=par("usr")
  rect(usr[1], usr[3], usr[2], usr[4])
  lines(win, lambda, lwd=2)
  points(win, lambda, pch=21, bg="gray")
  axis(1)
  axis(2)
  box()
  title(xlab="Number")
  title(ylab="Log.Singular Values")}


#Tambien generamos la scree-plot.
Sing.plt(META.ts, 30)
```



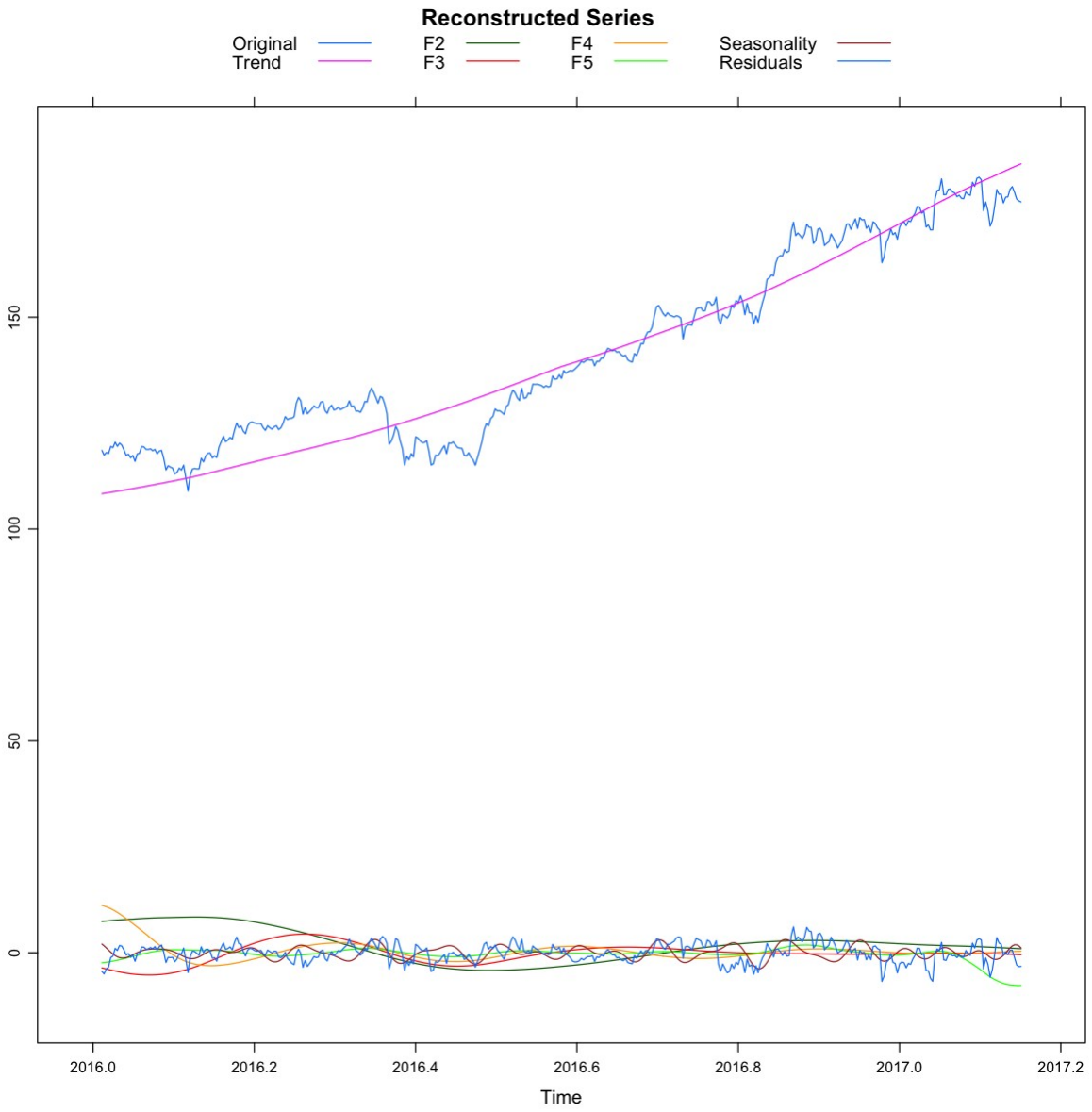

```
###Gráficas componentes 1D (compoente por componente) y la 2D (pares formen poligonos regulares : estables)
plot(s1, type="vectors", idx=1:24)
plot(s1, type= "paired", idx=2:24, plot.contrib = FALSE)
```

Eigenvectors






```
##Fase recontrucción:
r=reconstruct(s1, groups = list(Trend=1,2,3,4,5, Seasonality=c(8:9,
10:11, 12:13)))
plot(r, add.residuals=TRUE, add.original=TRUE, plot.method="xyplot",
     superpose=TRUE, auto.key=list(columns=4))
```

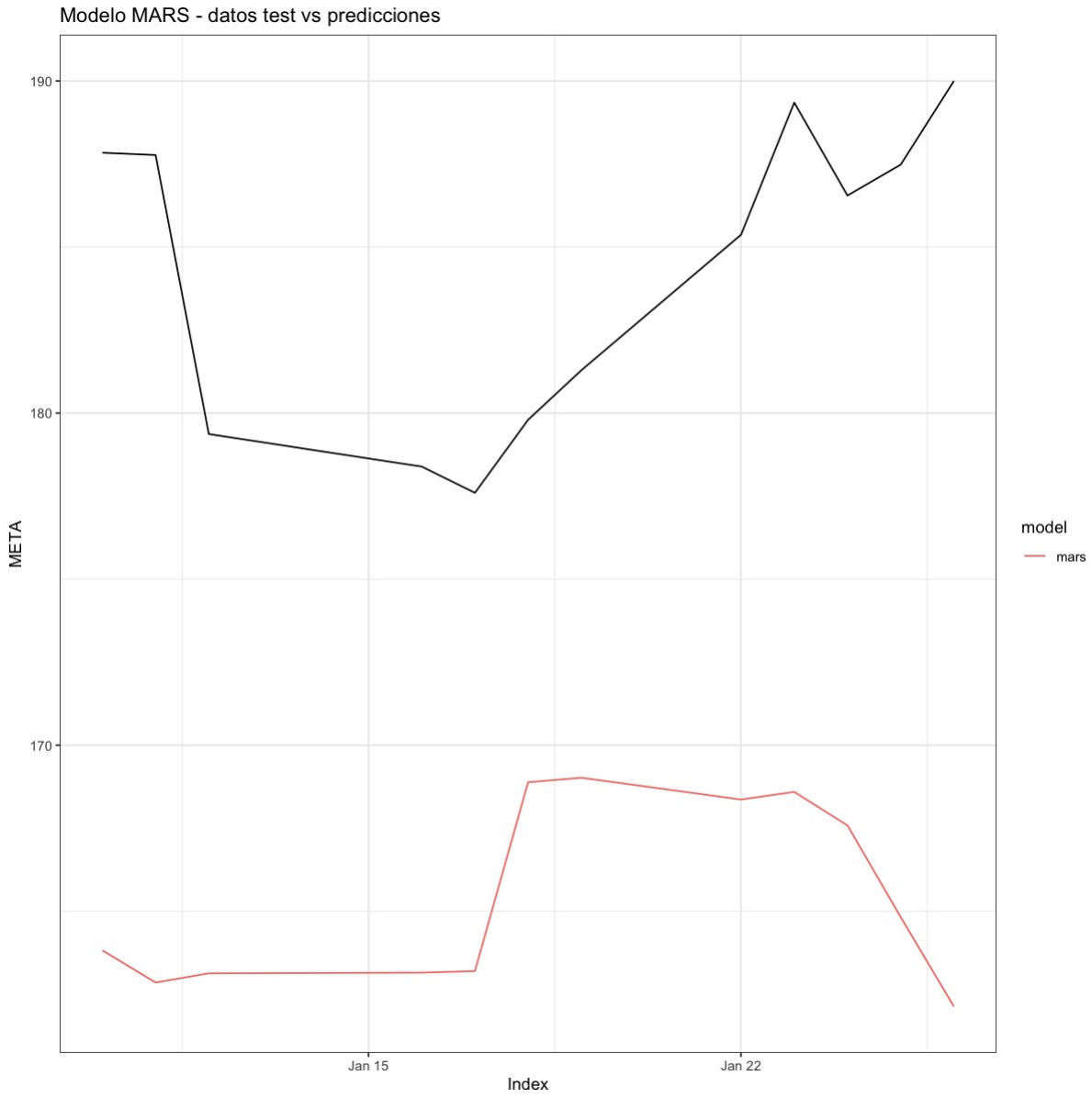


7. Empleando tus resultados, calcula las métricas para la medición de asertividad de un pronóstico, al menos debescalcular dos métricos de error de pronóstico, ejemplo: MAPE, RMSE, MAE, entre otros.

7.1 Validacion en conjunto de test con MARS

```
#Procedemos con pasar los datos de un formato xts, a un formato
data.frame para facilitar la generación de rezagos.
Pr_test2<-as.data.frame(Pr_tes)
#Generación de rezagos o features: debemos generar los mismos tipos de
features que en la parte de entrenamiento.
```

```r
lag1_pr_test = lag(Pr_test2,n=3L)
lag2_pr_test = lag(Pr_test2,n=7L)
lag3_pr_test = lag(Pr_test2,n=8L)
lag4_pr_test = lag(Pr_test2,n=10L)

Pr_all_test<-cbind(Pr_test2, lag1_pr_test, lag2_pr_test, lag3_pr_test,
lag4_pr_test) %>%na.omit()
colnames(Pr_all_test)<-c("META","l3","l7","l8","l10")
#Generamos la tendencia:
Pr_all_test$trend = 1:nrow(Pr_all_test)
#Renombramos:
colnames(Pr_all_test)<-c("META","l3","l7","l8","l10", "trend")

# Tambien, damos formato indexado a la fecha.
yt <- as.xts(Pr_all_test, dateFormat = "Date")
Pr_test2 <- fortify.zoo(yt)

#Neuvamente, podemos crear un objeto que contemple sólamente los
rezagos o features de la parte de prueba, este sería el x_test.
x_test <- Pr_test2 %>%
  select(starts_with(c("l", "t")))

##Llamamos de manera similar, a la respuesta de la parte de prueba.
y_test <- Pr_test2%>%
  select(META)

#Modelo MARS
pred_mars = predict(mars_mod, x_test)[,1]
str(pred_mars)

#Podemos unir los resultados a manera que se generen columnas con los
resultados de ajuste de los modelos:
Pr_test3 <- Pr_test2 %>%
  mutate(
    pred_mars = predict(mars_mod, x_test)[,1]
  )

##Podemos graficar sólo el segmento de pronóstico:
Pr_test3 %>%
  select(Index, META, starts_with("pred")) %>%
  gather("model", "forecast", -Index, -META) %>%
  mutate(model = str_remove(model, "pred_")) %>%
  ggplot(aes(Index, META)) +
  geom_line() +
  geom_line(aes(y = forecast, color = model)) +
  ggtitle("Modelo MARS - datos test vs predicciones")  # Título del
gráfico


 num [1:12] 164 163 163 163 163 ...
```




```
# Metricas de desempeño modelo MARS
metrics_table <- bind_rows(
  yardstick::rmse(Pr_test3, truth = META, estimate = pred_mars),
  yardstick::mae(Pr_test3, truth = META, estimate = pred_mars),
  yardstick::mape(Pr_test3, truth = META, estimate = pred_mars)
)
metrics_table
```

```
  .metric .estimator .estimate
1 rmse    standard   19.44984
2 mae     standard   18.76902
3 mape    standard   10.13680
```

7.2 Validacion en conjunto de test con SSA

```r
#Generamos el pronóstico señalando los tripletes de la reconstrucción.
pronostico1<-forecast(s1, groups = list(c(1:5,10:13)), len=h,
method="recurrent",
                                  interval = "prediction",
level=c(0.8, 0.99),only.new=TRUE )
# Graficamos el pronóstico.
autoplot(pronostico1, type="l", main="Prediction Intervals",
ylab="Precio")

pred<-pronostico1$mean
```


```r
###Cálculo error MAPE, RMSE
RMSE1 <- Metrics::rmse(test, pred)
```

```
MAPE1 <- Metrics::mape(test, pred)
MAE1  <- Metrics::mae(test, pred)

RMSE1
MAE1
MAPE1_porcentaje <- MAPE1 * 100
MAPE1_porcentaje
```

```
[1] 5.476903
```

```
[1] 4.69289
```

```
[1] 2.594655
```

PASO 2:Comparación de los modelos y selección del mejor

1.Compara y sustenta qué modelo presentó un mejor desempeño de acuerdo a los criterios de selección pertinentes.

Modelo	RMSE	MAE	MAPE (%)
MARS	19.45	18.77	10.14
LSTM	4.33	3.82	2.09
Random Forest	6.15	5.12	2.75
SSA	5.48	4.69	2.59

En base a las métricas obtenidas, el modelo LSTM muestra el mejor desempeño en términos de RMSE, MAE y MAPE, superando a los demás modelos con un RMSE de 4.33, MAE de 3.82 y un MAPE de 2.09%. Este desempeño es notablemente mejor en comparación con el modelo MARS (RMSE: 19.45, MAPE: 10.14%) y Random Forest (RMSE: 6.15, MAPE: 2.75%). Sin embargo, el modelo SSA también tiene buenos resultados con un MAPE de 2.59%, aunque sigue siendo menos preciso que el LSTM. Estos resultados sugieren que los modelos basados en redes neuronales, como el LSTM, pueden capturar mejor las complejas relaciones temporales en los datos de precios de acciones, mientras que los modelos más simples, como MARS y Random Forest, no logran igualar la precisión

```
```