Energías Renovables - Index

May 8, 2024

1 Energías Renovables - Análisis del Índice Bursátil

Creado por:

- **V. D. Betancourt**

1.1 Introducción

1.1.1 Descripción

El presente proyecto pertenece al ámbito de las **Energías Renovables**. En particular, está enfocado en datos del **Índice Bursátil** de una **Empresa**.

1.1.2 Objetivo

El presente proyecto tiene como finalidad estudiar la **Índice Bursátil de la Empresa**, por medio de:

- Generar un *Análisis Exploratorio de Datos*, incluyendo diversas visualizaciones diseñadas en Seaborn.
- Generar *Predicciones* con diferentes modelos, tales como: Redes Neuronales y Series Temporales.

1.2 Settings

[ ]:
```
# Importar
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

[ ]:
```
# Yahoo Finance
#!pip install yfinance
import yfinance as yf
```

1.3 Descargar Datos

[ ]:
```
import yfinance as yf

# Crea un objeto Ticker para Audax
audax = yf.Ticker("ADX.MC")

# Definir período
fecha_inicio = "2016-01-01"
fecha_fin = "2024-04-30"

# Extraer los datos históricos
data = audax.history(start=fecha_inicio, end=fecha_fin)

# Muestra los datos descargados
data
```

[ ]:
```
                                 Open      High       Low     Close  Volume  \
Date
2016-01-04 00:00:00+01:00  0.375549  0.375549  0.360725  0.365666  210161
2016-01-05 00:00:00+01:00  0.365666  0.385432  0.365666  0.375549  163781
2016-01-06 00:00:00+01:00  0.385432  0.385432  0.365666  0.365666  152999
2016-01-07 00:00:00+01:00  0.370608  0.370608  0.360725  0.370608  134482
2016-01-08 00:00:00+01:00  0.370608  0.380490  0.365666  0.365666  182313
…                               …         …         …         …       …
2024-04-23 00:00:00+02:00  1.776000  1.780000  1.754000  1.774000  375601
2024-04-24 00:00:00+02:00  1.772000  1.806000  1.762000  1.804000  575267
2024-04-25 00:00:00+02:00  1.796000  1.830000  1.768000  1.784000  710361
2024-04-26 00:00:00+02:00  1.786000  1.798000  1.770000  1.782000  379534
2024-04-29 00:00:00+02:00  1.776000  1.788000  1.746000  1.770000  623360

                           Dividends  Stock Splits
Date
2016-01-04 00:00:00+01:00        0.0           0.0
2016-01-05 00:00:00+01:00        0.0           0.0
2016-01-06 00:00:00+01:00        0.0           0.0
2016-01-07 00:00:00+01:00        0.0           0.0
2016-01-08 00:00:00+01:00        0.0           0.0
…                                …             …
2024-04-23 00:00:00+02:00        0.0           0.0
2024-04-24 00:00:00+02:00        0.0           0.0
2024-04-25 00:00:00+02:00        0.0           0.0
2024-04-26 00:00:00+02:00        0.0           0.0
2024-04-29 00:00:00+02:00        0.0           0.0

[2130 rows x 7 columns]
```

Warning!:

- Nótese que `'data'` ya es un DataFrame.

1.4 Análisis Exploratorio de Datos

1.4.1 Información Básica

[ ]:
```
# Información general del DataFrame
print("Información de Valores No Nulos por Columna␣
  ↪==============================")
data.info()
```

```
Información de Valores No Nulos por Columna ==============================
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2130 entries, 2016-01-04 00:00:00+01:00 to 2024-04-29
00:00:00+02:00
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Open          2130 non-null   float64
 1   High          2130 non-null   float64
 2   Low           2130 non-null   float64
 3   Close         2130 non-null   float64
 4   Volume        2130 non-null   int64
 5   Dividends     2130 non-null   float64
 6   Stock Splits  2130 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 133.1 KB
```

[ ]:
```
# Estadísticas descriptivas
print("\nEstadística Descriptiva␣
  ↪================================================")
data.describe()

# Visualización: Histograma del precio de cierre ajustado
#print("\nHistograma del Precio de Cierre Ajustado␣
  ↪==============================")
#data['Close'].plot(kind='hist')
```

```
Estadística Descriptiva ================================================
```

[ ]:

	Open	High	Low	Close	Volume	\
count	2130.000000	2130.000000	2130.000000	2130.000000	2.130000e+03	
mean	1.335875	1.365992	1.300913	1.330905	9.426072e+05	
std	0.635222	0.652666	0.612189	0.631536	2.721767e+06	
min	0.345900	0.355783	0.340959	0.345900	0.000000e+00	
25%	0.630527	0.642386	0.617926	0.627562	1.668440e+05	
50%	1.308740	1.329623	1.283500	1.301000	3.849175e+05	

```
75%       1.902863     1.937042     1.859709     1.899872  9.498588e+05
max       3.172400     3.290995     3.043923     3.162518  9.922906e+07

          Dividends  Stock Splits
count  2130.000000        2130.0
mean      0.000011           0.0
std       0.000492           0.0
min       0.000000           0.0
25%       0.000000           0.0
50%       0.000000           0.0
75%       0.000000           0.0
max       0.022712           0.0
```

[ ]:
```
# Filas y Columnas
print("Filas y Columnas en el DataFrame")
data.shape
```

```
Filas y Columnas en el DataFrame
```

[ ]:
```
(2130, 7)
```

[ ]:
```
# Nombres de Columnas
print("Columnas")
data.columns
```

```
Columnas
```

[ ]:
```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Dividends', 'Stock Splits'],
dtype='object')
```

Warning!:

- Nótese que no se tiene una columna para la fecha.

1.4.2 Missing Values

[ ]:
```
# Missing Values
print("Total de Missing Values por Columna:")
print(data.isnull().sum())
```

```
Total de Missing Values por Columna:
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
```

```
Stock Splits    0
dtype: int64
```

1.4.3 Data Wrangling

Fechas

[ ]:
```
# Convertir Índice a Columna tipo Fecha
# Resetear el índice para convertir la fecha de índice a columna
data_reset = data.reset_index()

# Convertir la columna de fecha a formato 'YYYY-MM-DD'
data_reset['Date'] = data_reset['Date'].dt.strftime('%Y-%m-%d')
data_reset
```

[ ]:

	Date	Open	High	Low	Close	Volume	Dividends	\
0	2016-01-04	0.375549	0.375549	0.360725	0.365666	210161	0.0	
1	2016-01-05	0.365666	0.385432	0.365666	0.375549	163781	0.0	
2	2016-01-06	0.385432	0.385432	0.365666	0.365666	152999	0.0	
3	2016-01-07	0.370608	0.370608	0.360725	0.370608	134482	0.0	
4	2016-01-08	0.370608	0.380490	0.365666	0.365666	182313	0.0	
…	…	…	…	…	…	…		
2125	2024-04-23	1.776000	1.780000	1.754000	1.774000	375601	0.0	
2126	2024-04-24	1.772000	1.806000	1.762000	1.804000	575267	0.0	
2127	2024-04-25	1.796000	1.830000	1.768000	1.784000	710361	0.0	
2128	2024-04-26	1.786000	1.798000	1.770000	1.782000	379534	0.0	
2129	2024-04-29	1.776000	1.788000	1.746000	1.770000	623360	0.0	

	Stock Splits
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
…	…
2125	0.0
2126	0.0
2127	0.0
2128	0.0
2129	0.0

```
[2130 rows x 8 columns]
```

1.4.4 Exportar Datos a CSV

Este paso permitirá abrir y visualizar los datos en formato **.csv** desde otro tipo de programas.

[ ]:
```
# Exportar a CSV, asegurándote de que la columna de fecha esté incluida
data_reset.to_csv('datos_hist_adxmc.csv', index=False)
```

1.4.5 Cargar CSV

Este paso sirve para simular que ya se tienen los datos históricos en un **.csv**. Evidentemente, asume que las columnas son las mismas que el fichero que se crea con los datos extraídos desde Yahoo Finance.

[ ]:
```
import pandas as pd

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Convertir la columna 'Date' de nuevo a tipo datetime si es necesario
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])

# Mostrar DataFrame
data_loaded.head()
```

[ ]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2016-01-04	0.375549	0.375549	0.360725	0.365666	210161	0.0	0.0
1	2016-01-05	0.365666	0.385432	0.365666	0.375549	163781	0.0	0.0
2	2016-01-06	0.385432	0.385432	0.365666	0.365666	152999	0.0	0.0
3	2016-01-07	0.370608	0.370608	0.360725	0.370608	134482	0.0	0.0
4	2016-01-08	0.370608	0.380490	0.365666	0.365666	182313	0.0	0.0

1.4.6 Estadística Descriptiva del 'Close' por Año

[ ]:
```
import pandas as pd

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Asumiendo que data_reset es tu DataFrame y que 'Date' está en formato de␣
↪cadena
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded['Year'] = data_loaded['Date'].dt.year
```

[ ]:
```
# Ahora puedes agrupar por el año extraído
grouped = data_loaded.groupby('Year')

# Aplicar describe solo a la columna 'Close'
yearly_describe_close = grouped['Close'].describe()
```

```
yearly_describe_close
```

[ ]:

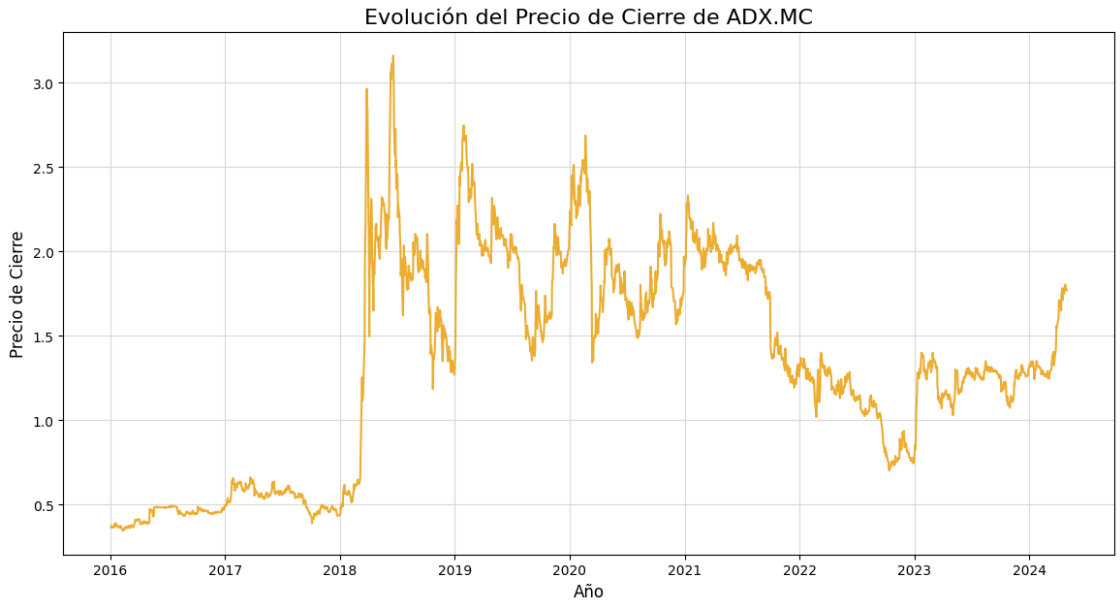
Year	count	mean	std	min	25%	50%	75%	max
2016	257.0	0.436211	0.044223	0.345900	0.390373	0.449670	0.469436	0.494143
2017	255.0	0.543383	0.061941	0.390373	0.484260	0.558382	0.583089	0.662152
2018	254.0	1.662322	0.658168	0.444729	1.349011	1.835743	2.075402	3.162518
2019	256.0	1.967609	0.313887	1.353953	1.709983	2.004246	2.105051	2.747437
2020	257.0	1.883809	0.289572	1.342093	1.664275	1.804612	2.045753	2.688140
2021	256.0	1.816454	0.296675	1.193000	1.518000	1.923000	2.016105	2.332357
2022	257.0	1.074863	0.195170	0.703000	0.880500	1.121000	1.241000	1.400000
2023	255.0	1.235635	0.092328	0.820000	1.181000	1.264000	1.294000	1.401000
2024	83.0	1.418578	0.185868	1.244000	1.290000	1.328000	1.555000	1.804000

[ ]:
```
# Exportar a CSV
yearly_describe_close.to_csv('datos_stat_anuales_adxmc.csv')
```

1.4.7 Lineplot

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Convertir la columna 'Date' de nuevo a tipo datetime si es necesario
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded.set_index('Date', inplace=True)

# Crear un gráfico de líneas para visualizar la evolución del precio de cierre
plt.figure(figsize=(14, 7))  # Configurar el tamaño de la figura
sns.lineplot(x='Date', y='Close', data=data_loaded, color='#ECAD30')  # 'o'␣
 ↪para mostrar puntos en cada dato
```

```
# Personalización adicional para mejorar la legibilidad
plt.title('Evolución del Precio de Cierre de ADX.MC', fontsize=16)  # Título del gráfico
plt.xlabel('Año', fontsize=12)  # Etiqueta para el eje X
plt.ylabel('Precio de Cierre', fontsize=12)  # Etiqueta para el eje Y
plt.grid(True, color='#D9D9D7')  # Añadir una cuadrícula para facilitar la lectura
plt.xticks(rotation=0)  # Rotar las etiquetas del eje X para mejorar la legibilidad

# Mostrar el gráfico
plt.show()
```



1.4.8 Velas Japonesas

[ ]:
```
!pip install mplfinance
```

```
Requirement already satisfied: mplfinance in /usr/local/lib/python3.10/dist-
packages (0.12.10b0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from mplfinance) (3.7.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from mplfinance) (2.0.3)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->mplfinance) (0.12.1)
```

```
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->mplfinance) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->mplfinance) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->mplfinance) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->mplfinance) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib->mplfinance) (1.16.0)
```

Visualización con Resampling

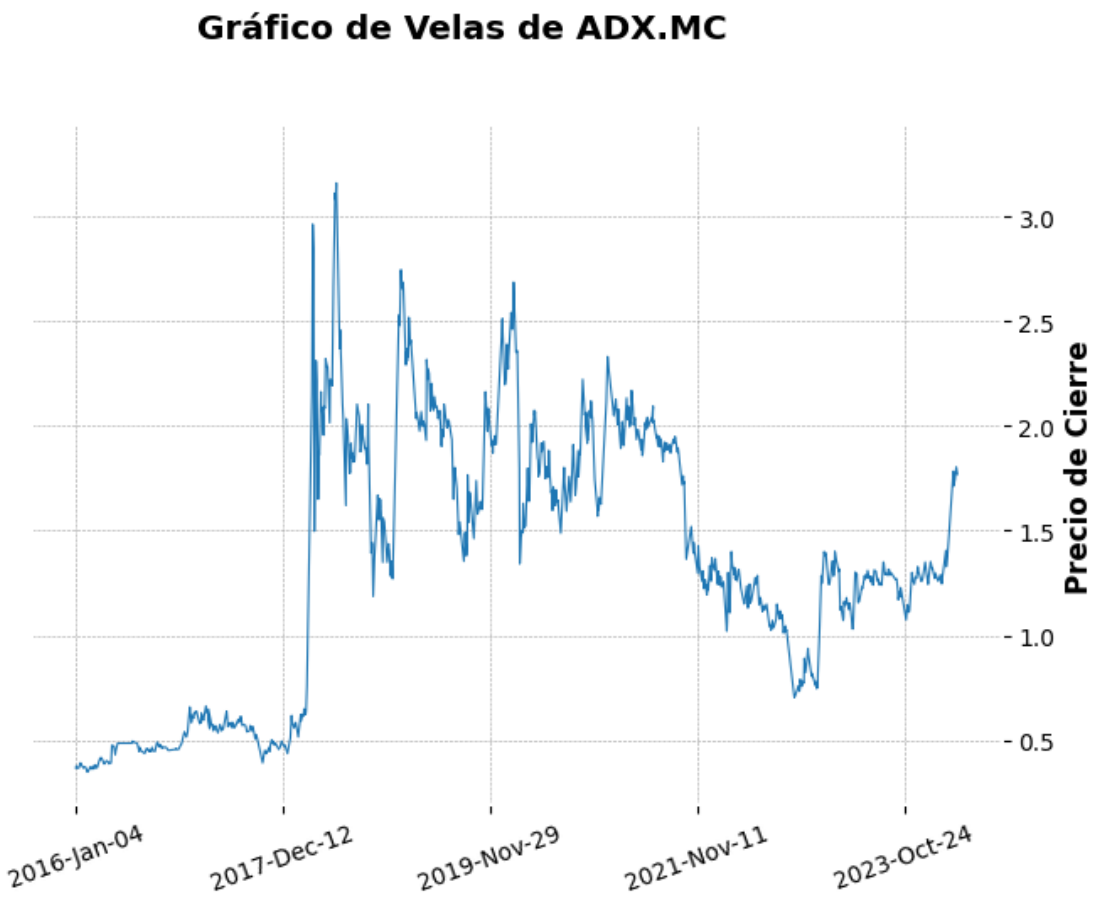
[ ]:
```python
import mplfinance as mpf
import pandas as pd

# Resamplear datos a frecuencia semanal y tomar el open, high, low y close de␣
 ↪cada semana
weekly_data = data_loaded.resample('W').agg({'Open': 'first',
                                             'High': 'max',
                                             'Low': 'min',
                                             'Close': 'last'})

# Crear el gráfico de velas con los datos resampleados
mpf.plot(weekly_data, type='candle', style='charles',
         title='Gráfico de Velas Semanal de ADX.MC',
         ylabel='Precio de Cierre')
```

Visualización de un Período Específico

[ ]:
```
import mplfinance as mpf
import pandas as pd

# Filtrar datos por un período específico
filtered_data = data_loaded.loc['2024-01-01':'2024-04-30']

# Crear el gráfico de velas con los datos filtrados
mpf.plot(filtered_data, type='candle', style='charles',
         title='Gráfico de Velas de ADX.MC - Últimos meses',
         ylabel='Precio de Cierre')
```

Visualizar Muchos Datos Cuando se tienen demasiados datos, arrojará un **Warning** que sugiere cambiar el `type='candle'` por `type='line'`.

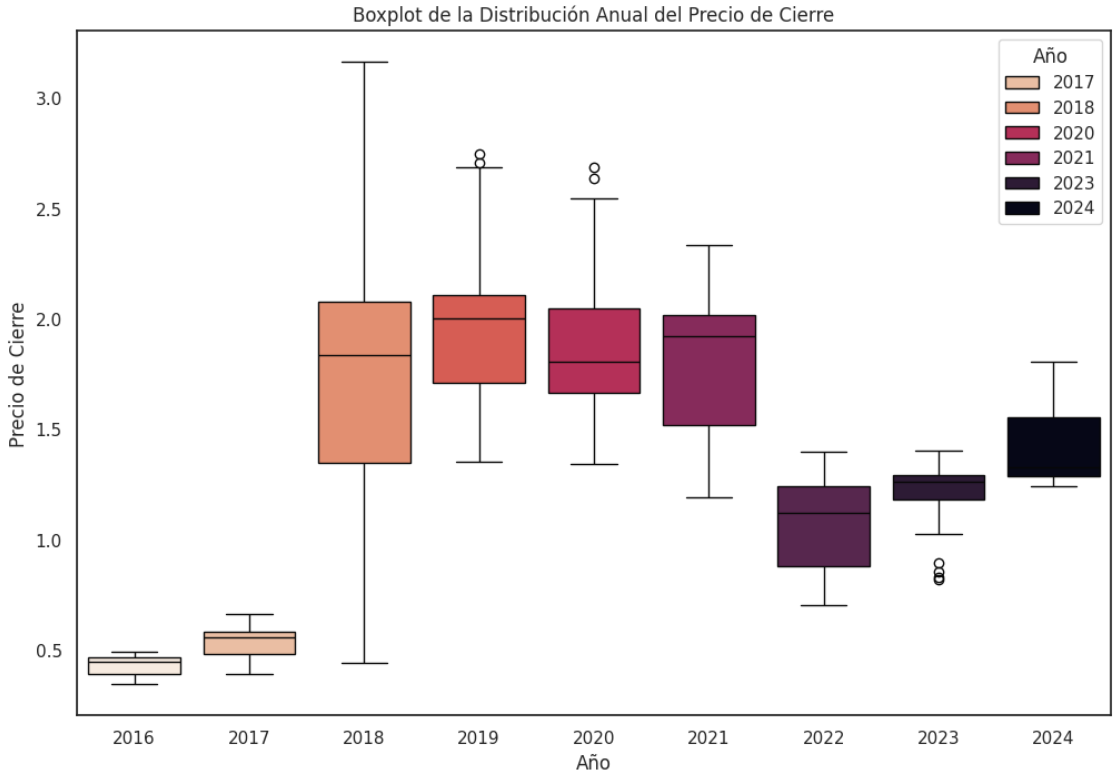
[ ]:
```
import mplfinance as mpf
import pandas as pd

# Ajustar el estilo visual del gráfico para permitir una mejor visualización de␣
 ↪muchos datos
mpf.plot(data_loaded, type='line', style='charles',
         title='Gráfico de Velas de ADX.MC',
         ylabel='Precio de Cierre',
         xrotation=20,  # Rota las etiquetas del eje x para mejor visualización
         scale_width_adjustment=dict(candle=0.8) # Ajustar la anchura de las␣
 ↪velas
         )
```



1.4.9 Boxplot

[16]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Convertir 'Date' a formato de Fecha
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded.set_index('Date', inplace=True)

# Extraer el año de la fecha y crear una nueva columna 'Year'
data_loaded['Año'] = data_loaded.index.year

# Configurar el estilo estético de los gráficos
sns.set(style="white")
```

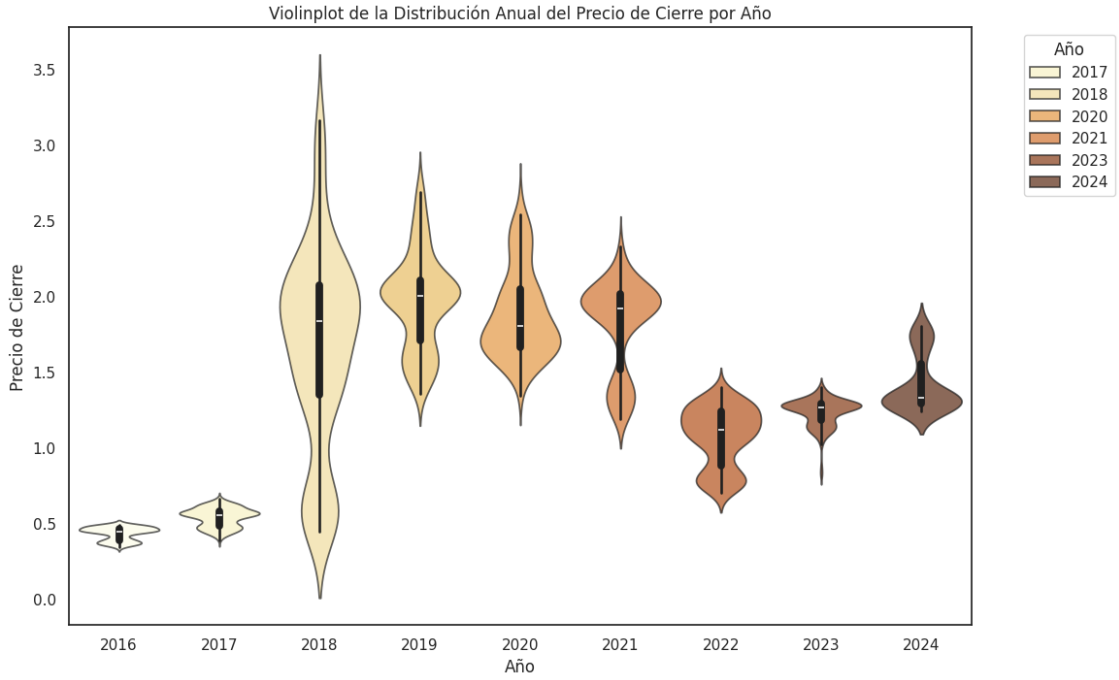
```
# Crear un gráfico de boxplot con años en el eje X y precios de cierre en el␣
 ↪eje Y
plt.figure(figsize=(12, 8))
boxplot = sns.boxplot(x='Año', y='Close', data=data_loaded, hue='Año',␣
 ↪palette='rocket_r')

# Configuración adicional para mejorar la visualización
boxplot.set_title('Boxplot de la Distribución Anual del Precio de Cierre')
boxplot.set_xlabel('Año')
boxplot.set_ylabel('Precio de Cierre')
plt.xticks(rotation=0)

# Mostrar el gráfico
plt.show()
```



1.4.10 Violinplot

[15]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Convertir 'Date' a formato de Fecha
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded.set_index('Date', inplace=True)

# Extraer el año de la fecha y crear una nueva columna 'Year'
data_loaded['Año'] = data_loaded.index.year

# Configurar el estilo estético de los gráficos
sns.set(style="white")

# Crear un gráfico de violinplot con años en el eje X, precios de cierre en el␣
 ↪eje Y, y una paleta de colores para cada año
plt.figure(figsize=(12, 8))
violinplot = sns.violinplot(x='Año', y='Close', data=data_loaded, hue='Año',␣
 ↪palette='YlOrBr', split=False, alpha=0.5)

# Configuración adicional para mejorar la visualización
violinplot.set_title('Violinplot de la Distribución Anual del Precio de Cierre␣
 ↪por Año')
violinplot.set_xlabel('Año')
violinplot.set_ylabel('Precio de Cierre')
plt.xticks(rotation=0)

# Añadir leyenda y ajustar la posición si es necesario
plt.legend(title='Año', bbox_to_anchor=(1.05, 1), loc='upper left')

# Mostrar el gráfico
plt.show()
```



1.5 Análisis de Riesgo de Mercado

1.5.1 Value-at-Risk (VaR)

El **Value-at-Risk (VaR)** es una técnica ampliamente utilizada para medir el riesgo de pérdida en inversiones. Estima cuánto podría perder una inversión en un periodo de tiempo determinado bajo condiciones normales de mercado, en un nivel de confianza específico.

1.5.2 VaR Histórico

El cálculo del **VaR por Metodología Histórica** (abreviado **VaR Hist**)utiliza datos históricos para simular posibles pérdidas futuras y determinar el umbral de pérdida correspondiente al nivel de confianza seleccionado.

Preparación

[ ]:
```
import pandas as pd

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Asegurarse de que la fecha está en formato adecuado y como índice
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded.set_index('Date', inplace=True)
```

Calcular Retornos Diarios

[ ]:
```
# Calcular retornos diarios
data_loaded['Returns'] = data_loaded['Close'].pct_change()
```

Ordenar los Retornos

[ ]:
```
# Eliminar cualquier NaN que pueda haber surgido en el cálculo de los retornos
data_loaded.dropna(inplace=True)

# Ordenar los retornos
sorted_returns = data_loaded['Returns'].sort_values()
```

Parámetros Comúnmente, el VaR se calcula para niveles de confianza del 95% o 99%. Esto implica que estamos buscando el peor retorno esperado en el 5% o 1% de los casos, respectivamente.

[ ]:
```
# Definir el Nivel de Confianza
confidence_level = 0.95
```

Calcular VaR Hist

[ ]:
```
# Calcular el índice del retorno que corresponde al VaR
var_index = int((1 - confidence_level) * len(sorted_returns))

# Obtener el VaR
var_value = sorted_returns.iloc[var_index]

# Convertir el VaR a Porcentaje
total_investment = 100000
var_percent = var_value * 100
var_amount = abs(var_value * total_investment)

print(f"El VaR histórico al {confidence_level*100}% de nivel de confianza es:␣
↪{var_percent:.2f}%")
print(f"\nEsto equivale a una pérdida potencial de €{var_amount:.2f} por cada␣
↪€{total_investment:.0f} invertidos.")
```

```
El VaR histórico al 95.0% de nivel de confianza es: -4.03%

Esto equivale a una pérdida potencial de €4026.84 por cada €100000 invertidos.
```

[ ]:
```
# Interpretación extendida
print(f"Interpretación: \n\nBajo condiciones normales de mercado, \nexiste un␣
↪{confidence_level*100:.0f}% de probabilidad " \
        f"de que \nla pérdida no sea mayor que €{var_amount:.2f}. \nEsto es␣
↪el {abs(var_percent):.2f}% del total invertido (€{total_investment:.0f}). " \
        f"\n\nPor lo tanto, existe un {(1-confidence_level)*100:.0f}% de␣
↪probabilidad \nde que, en un día, la pérdida exceda este monto: €{var_amount:
↪.2f}.")
```

```
Interpretación:

Bajo condiciones normales de mercado,
existe un 95% de probabilidad de que
la pérdida no sea mayor que €4026.84.
Esto es el 4.03% del total invertido (€100000).

Por lo tanto, existe un 5% de probabilidad
de que, en un día, la pérdida exceda este monto: €4026.84.
```

Automatizar

[ ]:
```python
def calcula_VaR_hist(data, total_investment, confidence_level=0.95):
    # Calcular retornos diarios
    data['Returns'] = data['Close'].pct_change()
    data.dropna(inplace=True)  # Eliminar NaNs

    # Ordenar los retornos
    sorted_returns = data['Returns'].sort_values()

    # Calcular el índice del retorno que corresponde al VaR
    var_index = int((1 - confidence_level) * len(sorted_returns))

    # Obtener el VaR
    var_value = sorted_returns.iloc[var_index]

    # Convertir el VaR a un formato más entendible (por ejemplo, como un␣
↪porcentaje)
    var_percent = var_value * 100
    var_amount = abs(var_value * total_investment)

    # Imprimir el VaR
    print(f"El VaR histórico al {confidence_level*100}% de nivel de confianza␣
↪es: {var_percent:.2f}%")
    print(f"Esto equivale a una pérdida potencial de €{var_amount:.2f} por cada␣
↪€{total_investment:.0f} invertidos.")

    # Interpretación extendida
    print(f"\nInterpretación: \n\nBajo condiciones normales de mercado,␣
↪\nexiste un {confidence_level*100:.0f}% de probabilidad " \
          f"de que \nla pérdida no sea mayor que €{var_amount:.2f}. \nEsto es␣
↪el {abs(var_percent):.2f}% del total invertido (€{total_investment:.0f}). " \
          f"\n\nPor lo tanto, existe un {(1-confidence_level)*100:.0f}% de␣
↪probabilidad \nde que, en un día, la pérdida exceda este monto: €{var_amount:
↪.2f}.")
```

[ ]:
```
# Uso de la función con los datos cargados y un valor total de inversión de␣
↪100,000 euros
monto_invertido = 100000

# Calcular VaR Histórico
calcula_VaR_hist(data_loaded, monto_invertido)
```

```
El VaR histórico al 95.0% de nivel de confianza es: -4.03%
Esto equivale a una pérdida potencial de €4026.84 por cada €100000 invertidos.

Interpretación:

Bajo condiciones normales de mercado,
existe un 95% de probabilidad de que
la pérdida no sea mayor que €4026.84.
Esto es el 4.03% del total invertido (€100000).

Por lo tanto, existe un 5% de probabilidad
de que, en un día, la pérdida exceda este monto: €4026.84.
```

1.5.3 VaR por Simulación Monte Carlo

El cálculo del Value-at-Risk (VaR) usando **Simulación Monte Carlo** (abreviado **VaR SMC**) es un método más complejo que la metodología histórica, pero ofrece la ventaja de poder modelar escenarios futuros basados en estimaciones estadísticas.

Este enfoque utiliza la simulación aleatoria para generar posibles resultados futuros basados en los retornos históricos, permitiendo calcular el VaR bajo diversas condiciones de mercado.

Preparación

[ ]:
```
import pandas as pd
import numpy as np

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Asumiendo que `data_loaded` es tu DataFrame y ya tiene una columna 'Close'␣
↪con precios de cierre
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded.set_index('Date', inplace=True)
```

Calcular Retornos Diarios

[ ]:
```
# Calcular retornos diarios
data_loaded['Returns'] = data_loaded['Close'].pct_change().dropna()
```

Parámetros La elección de la **semilla** en sí no es crítica, siempre que sea constante para lograr reproducibilidad. Puede elegirse cualquier número entero.

[ ]:
```
# Establecer una semilla para reproducibilidad
np.random.seed(42)
```

[ ]:
```
# Definir Parámetros
num_simulations = 1000
time_horizon = 1  # VaR de un día
confidence_level = 0.95
```

Simulación Monte Carlo

[ ]:
```
# Calcular la media y desviación estándar de los retornos diarios
mean_return = data_loaded['Returns'].mean()
std_dev_return = data_loaded['Returns'].std()

# Simular los posibles retornos futuros
simulated_returns = np.random.normal(mean_return, std_dev_return,␣
 ↪(num_simulations, time_horizon))
```

Calcular VaR SMC

[ ]:
```
# Calcular el valor final de la inversión para cada simulación
initial_investment = 100000  # Ejemplo de inversión inicial
simulated_end_values = initial_investment * (1 + simulated_returns).prod(axis=1)

# Ordenar los resultados simulados
sorted_simulated_values = np.sort(simulated_end_values)

# Calcular el VaR como el percentil que corresponde al nivel de confianza
var_index = int((1 - confidence_level) * num_simulations)
var_value_mc = sorted_simulated_values[var_index]
var_amount_mc = initial_investment - var_value_mc
```

[ ]:
```
# Resultados
print(f"El VaR por Simulación Monte Carlo al {confidence_level*100}% de nivel␣
 ↪de confianza es: €{var_amount_mc:.2f}")
print(f"\nEsto significa que bajo condiciones normales, hay un␣
 ↪{confidence_level*100:.0f}% de probabilidad " \
      f"\nde que la pérdida no exceda €{var_amount_mc:.2f} en el próximo día.")
```

```
El VaR por Simulación Monte Carlo al 95.0% de nivel de confianza es: €5426.28

Esto significa que bajo condiciones normales, hay un 95% de probabilidad
de que la pérdida no exceda €5426.28 en el próximo día.
```

1.6 Predicciones

1.6.1 Generar Predicciones Pasadas

Preparación

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Convertir 'Date' a formato de Fecha (por si no está así)
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded.set_index('Date', inplace=True)
```

[ ]:
```
# Escalar los datos
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data_loaded[['Close']])
```

[ ]:
```
# Definir la variable independiente
X = data_scaled

# Definir la variable dependiente
#(los mismos datos de 'Close' porque estamos haciendo predicción un paso␣
 ↪adelante)
y = data_scaled

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, shuffle=False)
```

Definición del Modelo con DropOut

[ ]:
```
# Definir el Modelo
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Definición del modelo
model = Sequential([
    Dense(64, input_dim=1, activation='relu'),  # Capa de entrada
    Dropout(0.2),  # Dropout para reducir el overfitting
    Dense(64, activation='relu'),  # Capa oculta
    Dense(1)  # Capa de salida
])

# Ver el resumen del modelo
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense (Dense)               (None, 64)                128

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 4353 (17.00 KB)
Trainable params: 4353 (17.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compilar el Modelo

[ ]:
```
# Compilar el Modelo
model.compile(optimizer='adam', loss='mean_squared_error')
```

Entrenamiento con EarlyStopping

[ ]:
```
# Entrenamiento con EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping

# Early stopping para evitar overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=50, validation_split=0.2,␣
 ↪callbacks=[early_stopping])
```

```
Epoch 1/50
43/43 [==============================] - 4s 11ms/step - loss: 0.0413 - val_loss:
9.1980e-04
Epoch 2/50
43/43 [==============================] - 0s 6ms/step - loss: 0.0036 - val_loss:
2.9869e-05
Epoch 3/50
43/43 [==============================] - 0s 4ms/step - loss: 0.0022 - val_loss:
3.6595e-05
Epoch 4/50
43/43 [==============================] - 0s 4ms/step - loss: 0.0019 - val_loss:
7.9609e-05
Epoch 5/50
43/43 [==============================] - 0s 4ms/step - loss: 0.0018 - val_loss:
1.2208e-04
Epoch 6/50
43/43 [==============================] - 0s 5ms/step - loss: 0.0017 - val_loss:
1.6364e-04
```

```
Epoch 7/50
43/43 [==============================] - 0s 4ms/step - loss: 0.0016 - val_loss:
1.9978e-04
```

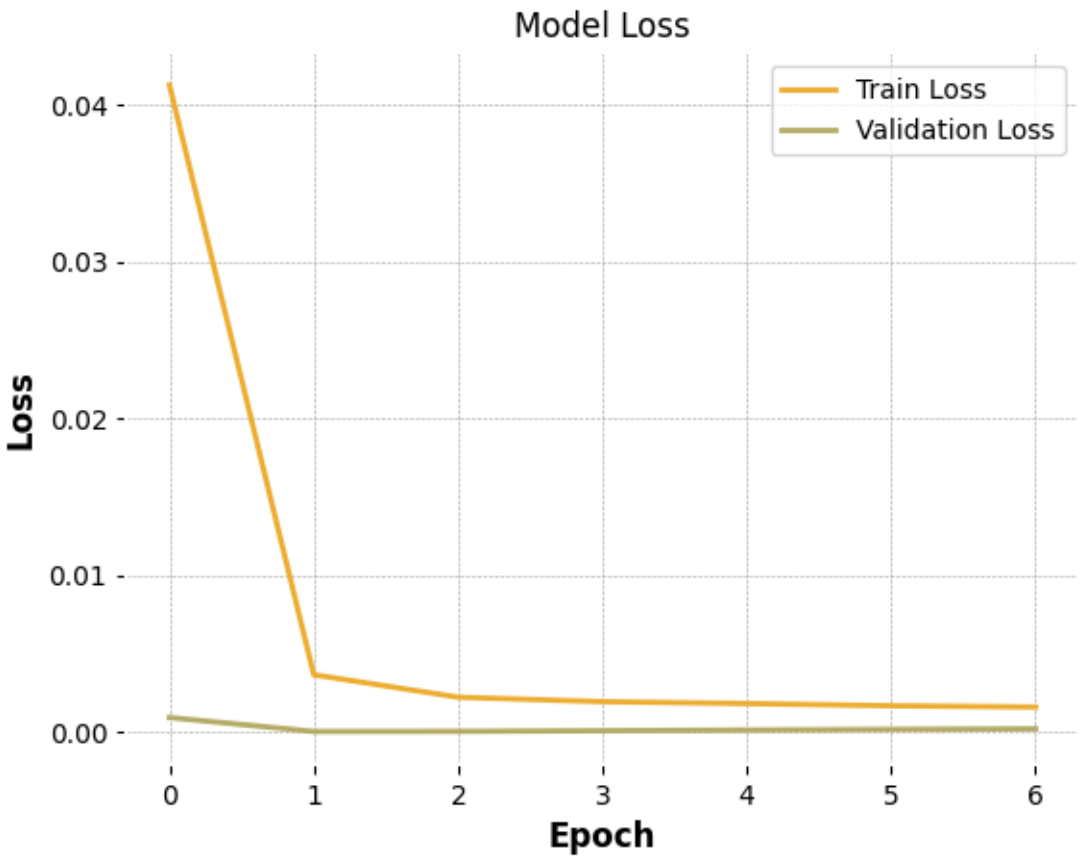
Evaluación del Modelo

[ ]:
```python
# Evaluación del modelo
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
```

```
14/14 [==============================] - 0s 5ms/step - loss: 1.1810e-04
Test Loss: 0.00011809503484982997
```

Visualizar el Historial del Entrenamiento

[ ]:
```python
# Visualizar Historial del Train y Validation
import matplotlib.pyplot as plt

# Visualizar la historia de entrenamiento
plt.plot(history.history['loss'], label='Train Loss', color='#ECAD30')
plt.plot(history.history['val_loss'], label='Validation Loss', color='#B5AB66')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



Generar Predicciones

[ ]:
```
# Generar predicciones
predictions = model.predict(X_test)
```

```
14/14 [==============================] - 0s 2ms/step
```

[ ]:
```
# Crear DataFrame de predicciones
predictions_df = pd.DataFrame(predictions, columns=['Predicted_Close'])
```

[ ]:
```
# Mostrar Predicciones
predictions_df
```

[ ]:
```
     Predicted_Close
0           0.278319
1           0.284397
2           0.283118
3           0.271280
4           0.257900
..               …
```

```
421         0.495622
422         0.505105
423         0.498783
424         0.498151
425         0.494358

[426 rows x 1 columns]
```

[ ]:
```
# Obtener las fechas correspondientes a X_test desde el DataFrame original
test_dates = data_loaded.index[len(data_loaded) - len(predictions):]
```

[ ]:
```
# Asignar estas fechas al DataFrame de predicciones
predictions_df.index = test_dates
```

[ ]:
```
# Mostrar Predicciones (debe mostrar las fechas)
predictions_df
```

[ ]:
```
            Predicted_Close
Date
2022-08-30         0.278319
2022-08-31         0.284397
2022-09-01         0.283118
2022-09-02         0.271280
2022-09-05         0.257900
…                       …
2024-04-23         0.495622
2024-04-24         0.505105
2024-04-25         0.498783
2024-04-26         0.498151
2024-04-29         0.494358

[426 rows x 1 columns]
```

Exportar Predicciones

[ ]:
```
# Exportar Predicciones
predictions_df.to_csv('datos_predicciones_pasadas_adxmc.csv')
```

Combinar Histórico con Predicciones

[ ]:
```
# Combinar los datos originales con las predicciones
combined_df = pd.concat([data_loaded['Close'], predictions_df], axis=1)

# Puede que desees rellenar NaN en los valores 'Actual_Close' para las fechas␣
 ↪de predicciones y viceversa
combined_df.columns = ['Actual_Close', 'Predicted_Close']
```

[ ]:
```
# Mostrar Combinado
combined_df.tail()
```

[ ]:

Date	Actual_Close	Predicted_Close
2024-04-23	1.774	0.495622
2024-04-24	1.804	0.505105
2024-04-25	1.784	0.498783
2024-04-26	1.782	0.498151
2024-04-29	1.770	0.494358

Exportar Combinado

[ ]:
```
# Exportar DataFrame combinado
combined_df.to_csv('datos_hist_con_predicciones_pasadas_adxmc.csv')
```

Visualizar Histórico y Predicciones del Histórico

[ ]:
```
# Visualizar los valores históricos y las predicciones

plt.figure(figsize=(10, 5))
plt.plot(combined_df['Actual_Close'], label='Real (Histórico)', color='#ECAD30')
plt.plot(combined_df['Predicted_Close'], label='Predicción (Pasada)',␣
↪color='#B5AB66', linestyle='--')
plt.title('Real vs Predicciones (Pasadas)')
plt.xlabel('Tiempo')
plt.ylabel('Precio de Cierre (Close)')
plt.legend()
plt.show()
```

1.6.2 Generar Predicciones Futuras

Se usará el mismo Modelo de Redes Neuronales creado para las Predicciones Pasadas, con la diferencia de que ahora se harán predicciones de los valores del Precio 'Close' en fechas futuras.

Preparación

[ ]:
```
import pandas as pd
import numpy as np
from datetime import timedelta
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Cargar el CSV en un DataFrame
data_loaded = pd.read_csv('datos_hist_adxmc.csv')

# Convertir 'Date' a formato de Fecha
data_loaded['Date'] = pd.to_datetime(data_loaded['Date'])
data_loaded.set_index('Date', inplace=True)
```

[ ]:
```
# Asumimos que 'data_loaded' ya está cargado y que 'scaler' fue entrenado con␣
 ↪una columna DataFrame
last_date = data_loaded.index.max()
future_dates = pd.date_range(start=last_date + timedelta(days=1), periods=30,␣
 ↪freq='D')

# Como hemos usado el último valor del histórico para predecir el futuro,␣
 ↪escalaremos este valor
# Aseguramos que estamos usando DataFrame para mantener la consistencia con el␣
 ↪entrenamiento de scaler
last_value = pd.DataFrame([data_loaded['Close'].iloc[-1]], columns=['Close'])

# Escalar usando DataFrame para evitar el warning
last_value_scaled = scaler.transform(last_value)
```

[ ]:
```
# Preparar el último valor como punto de partida
last_value_scaled = scaler.transform([[data_loaded['Close'].iloc[-1]]])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but MinMaxScaler was fitted with feature names
  warnings.warn(
```

Generar Predicciones

[ ]:
```
# Generar Predicciones Futuras en Cascada (Bucle)
```

```python
# Inicializar la lista de entradas para las predicciones
input_value = last_value_scaled
predicted_values = []

# Usar el modelo para hacer predicciones en cascada para los próximos 30 días
for _ in range(30):
    # Predecir el siguiente valor
    next_day_prediction_scaled = model.predict(input_value)

    # Guardar la predicción desescalada para el registro
    next_day_prediction = scaler.inverse_transform(next_day_prediction_scaled)
    predicted_values.append(next_day_prediction[0][0])

    # Usar la predicción como entrada para el próximo día
    input_value = next_day_prediction_scaled
```

```
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
```

Guardar Predicciones

[ ]:
```
# Crear DataFrame de predicciones futuras
future_predictions_df = pd.DataFrame(predicted_values, index=future_dates,␣
 ↪columns=['Predicted_Close'])
future_predictions_df
```

[ ]:

	Predicted_Close
2024-04-30	1.738318
2024-05-01	1.710112
2024-05-02	1.685082
2024-05-03	1.662886
2024-05-04	1.643203
2024-05-05	1.625748
2024-05-06	1.610269
2024-05-07	1.596543
2024-05-08	1.584403
2024-05-09	1.573674
2024-05-10	1.564190
2024-05-11	1.555808
2024-05-12	1.548400
2024-05-13	1.541852
2024-05-14	1.536064
2024-05-15	1.530949
2024-05-16	1.526427
2024-05-17	1.522431
2024-05-18	1.518899
2024-05-19	1.515777
2024-05-20	1.513018
2024-05-21	1.510579
2024-05-22	1.508424
2024-05-23	1.506519
2024-05-24	1.504838
2024-05-25	1.503369
2024-05-26	1.502085
2024-05-27	1.500964
2024-05-28	1.499985
2024-05-29	1.499129

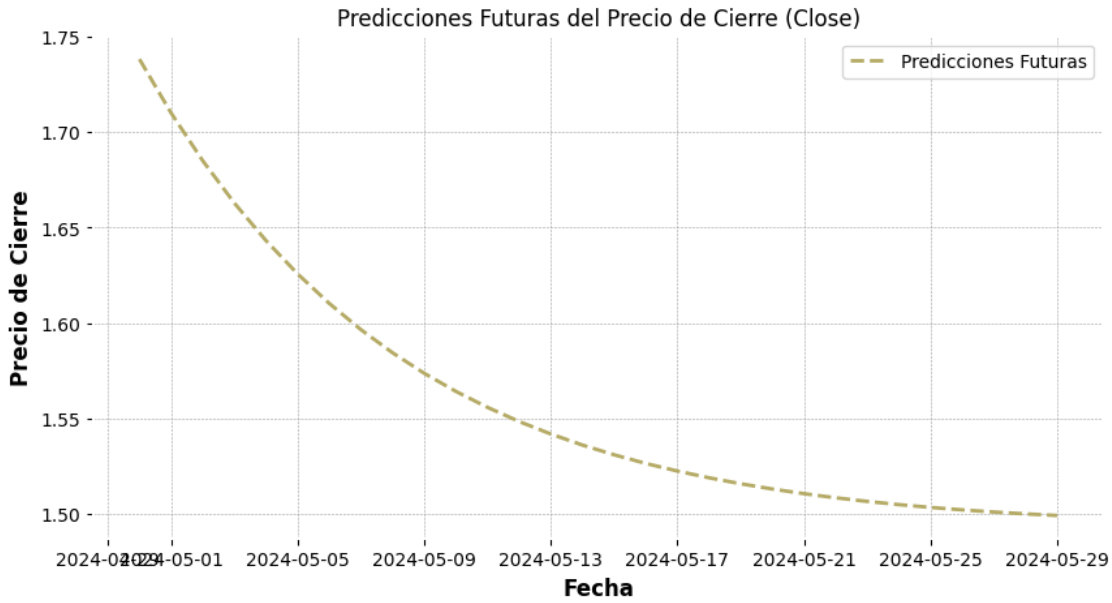
Exportar Predicciones a CSV

[ ]:
```
# Exportar Predicciones a CSV
future_predictions_df.to_csv('datos_predicciones_futuras_adxmc.csv')
```

Visualizar Predicciones Futuras

[ ]:
```
import matplotlib.pyplot as plt

# Visualizar las predicciones futuras
```

```
plt.figure(figsize=(10, 5))
#plt.plot(data_loaded['Close'], label='Histórico', color='#ECAD30')
plt.plot(future_predictions_df['Predicted_Close'], label='Predicciones␣
 ↪Futuras', color='#B5AB66', linestyle='--')
plt.title('Predicciones Futuras del Precio de Cierre (Close)')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre')
plt.legend()
plt.show()
```



[ ]:
```
import matplotlib.pyplot as plt

# Visualizar las predicciones futuras
plt.figure(figsize=(10, 5))
plt.plot(data_loaded['Close'], label='Histórico', color='#ECAD30')
plt.plot(future_predictions_df['Predicted_Close'], label='Predicciones␣
 ↪Futuras', color='#B5AB66', linestyle='--')
plt.title('Histórico y Predicciones Futuras del Precio de Cierre (Close)')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre')
plt.legend()
plt.show()
```

Combinar Histórico con Predicciones

[ ]:
```
# Combinar los datos históricos con las predicciones futuras
combined_df = pd.concat([data_loaded['Close'],␣
 ↪future_predictions_df['Predicted_Close']])
combined_df.to_csv('datos_hist_con_predicciones_futuras_adxmc.csv')
```

1.7 Fin