Entrenamiento y evaluación del modelo

Para optimizar el proceso y segmentar el desarrollo, se ha decidido separar el código en partes según los objtivos específicos del proyecto. Una vez que se ha completado el análisis exploratorio de los datos y que se han generado los conjuntos de prueba y entrenamiento, se procede con el entrenamiento y la evaluación de varios modelos, para ver cuál es el que se que comporta mejor y ofrece una mejor precisión en sus predicciones.

El primer paso, es caragar los conjuntos de entrenamiento y prueba generados anteriormente.

In [1]:
```
import pandas as pd
import os

os.chdir("C:/Users/jorge/Escritorio/TFM")

# Cargar los conjuntos de datos desde los archivos CSV
X_train = pd.read_csv('X_train.csv', index_col=0)
X_test = pd.read_csv('X_test.csv', index_col=0)
y_train = pd.read_csv('y_train.csv', index_col=0)
y_test = pd.read_csv('y_test.csv', index_col=0)

# Asegurarse de que las series de y_train y y_test sean de una sola columna
y_train = y_train.squeeze()
y_test = y_test.squeeze()

print("Datos cargados correctamente")
print("Tamaño del conjunto de entrenamiento:", X_train.shape)
print("Tamaño del conjunto de prueba:", X_test.shape)
```

```
Datos cargados correctamente
Tamaño del conjunto de entrenamiento: (1006, 1)
Tamaño del conjunto de prueba: (252, 1)
```

In [2]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
#Definimos una función genérica para evaluar los modelos, así podemos comparar v
def evaluate_model(y_true, y_pred, model_name):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - MAE: {mae}, MSE: {mse}, R²: {r2}')
    return {'Model': model_name, 'MAE': mae, 'MSE': mse, 'R²': r2}
```

Para la elección del mejor modelo, se entrenarán los que se consideran más usados por su fiabilidad y se irán almacenando los resultados uno a uno para después, poder compararlos todos y elegir el que ofrezca un mejor comportamiento.

En este caso se han seleccionado:

- **Regresión lineal**
- **Randon Forest**

- **Support Vector Regression (SVR)**
- **Gradient Boosting Regressor**
- **K-Nearest Neighbors Regressor**
- **Decision Tree Regressor**

In [3]:
```
results = []

# Modelo de Regresión Lineal
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train)
lr_predictions = lr.predict(X_test)
results.append(evaluate_model(y_test, lr_predictions, 'Linear Regression'))

# Modelo de Random Forest
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_predictions = rf.predict(X_test)
results.append(evaluate_model(y_test, rf_predictions, 'Random Forest'))

# Modelo de Support Vector Regression (SVR)
from sklearn.svm import SVR

svr = SVR()
svr.fit(X_train, y_train)
svr_predictions = svr.predict(X_test)
results.append(evaluate_model(y_test, svr_predictions, 'Support Vector Regressio

# Modelo de Gradient Boosting Regressor
from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(n_estimators=100, random_state=42)
gbr.fit(X_train, y_train)
gbr_predictions = gbr.predict(X_test)
results.append(evaluate_model(y_test, gbr_predictions, 'Gradient Boosting Regres

# Modelo de K-Nearest Neighbors Regressor
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor()
knn.fit(X_train, y_train)
knn_predictions = knn.predict(X_test)
results.append(evaluate_model(y_test, knn_predictions, 'K-Nearest Neighbors Regr

# Modelo de Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
dt_predictions = dt.predict(X_test)
results.append(evaluate_model(y_test, dt_predictions, 'Decision Tree Regressor')
```

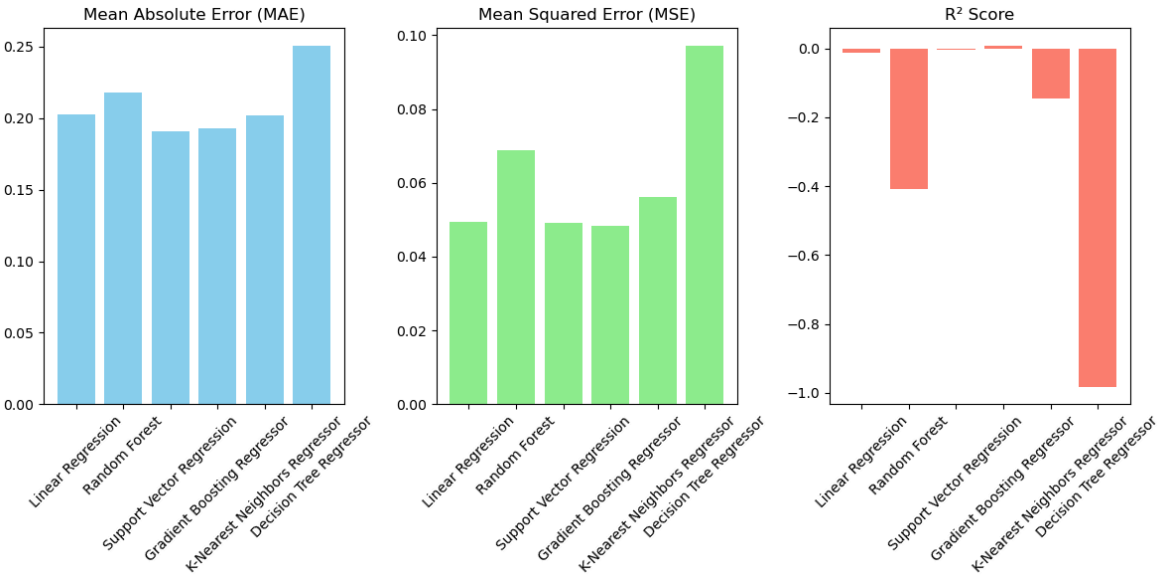
```
Linear Regression - MAE: 0.2027468126479598, MSE: 0.04949842596004199, R²: -0.011
77079650733548
Random Forest - MAE: 0.21801948982419084, MSE: 0.0689275644377916, R²: -0.4089114
031390091
Support Vector Regression - MAE: 0.19088623772037733, MSE: 0.04912590318033385, R
²: -0.00415625801985553
Gradient Boosting Regressor - MAE: 0.19300246486693032, MSE: 0.04846008244551614
6, R²: 0.009453426775807072
K-Nearest Neighbors Regressor - MAE: 0.2015948664056994, MSE: 0.0560898975272973
2, R²: -0.14650353413299322
Decision Tree Regressor - MAE: 0.250289134725654, MSE: 0.09705783383407758, R²: -
0.9839071635298926
```

Comparación de resultados

Una vez que se han entrenado y evaluado todos los modelos, se pueden comparar los resultado y visualizar las métricas

In [4]:

```python
import matplotlib.pyplot as plt

# Convertir los resultados a un DataFrame para facilitar la visualización
results_df = pd.DataFrame(results)

# Visualizar los resultados
plt.figure(figsize=(12, 6))

# MAE
plt.subplot(1, 3, 1)
plt.bar(results_df['Model'], results_df['MAE'], color='skyblue')
plt.title('Mean Absolute Error (MAE)')
plt.xticks(rotation=45)

# MSE
plt.subplot(1, 3, 2)
plt.bar(results_df['Model'], results_df['MSE'], color='lightgreen')
plt.title('Mean Squared Error (MSE)')
plt.xticks(rotation=45)

# R²
plt.subplot(1, 3, 3)
plt.bar(results_df['Model'], results_df['R²'], color='salmon')
plt.title('R² Score')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



Resultados obtenidos

1. Linear Regression:

- MAE: 0.2027
- MSE: 0.0495
- R^2: -0.0118

2. Random Forest:

- MAE: 0.2180
- MSE: 0.0689
- R^2: -0.4089

3. Support Vector Regression:

- MAE: 0.1909
- MSE: 0.0491
- R^2: -0.0042

4. Gradient Boosting Regressor:

- MAE: 0.1930
- MSE: 0.0485
- R^2: 0.0095

5. K-Nearest Neighbors Regressor:

- MAE: 0.2016
- MSE: 0.0561
- R^2: -0.1465

6. Decision Tree Regressor:

- MAE: 0.2503
- MSE: 0.0971

- R^2: -0.9839

De los resultados obtenidos, la MAE mide el error promedio de las predicciones en las mismas unidades que las variables originales. Los modelos con menor MAE son mejores. El support Vector Regression tiene el menor MAE (0.1909), seguido de cerca por el Gradien Boosting Regressor (0.1930). Por otro lado, la MSE mide el promedio de los errores al cuadrado. Es más sensible a grandes errores debido a la cuadratura. De entre todos, el Gradien Boosting regressor tiene el menos MSE (0.0485), lo que indica que tiene menores errores en general comparado con otro modelos. Por último, el R^2 mide la proporción de la varianza en la variable dependiente que es predecible a partir de las variables independiente. Un valor de R^2 cercano a 1 indica un modelo muy bueno. El Gradient Boosting Regressor tienen un R^2 positivo (0.0095), lo qeu sugiere que es el mejor modelo en términos de explicación de la varianza, aunque el valor resulta ser bastante bajo.

Conclusión

Gradient Boosting Regressor parece ser el mejor modelo en términos de MAE, MSE y R^2, aunque los valores absolutos del R^2 sugieren que todavía hay espacio para mejorar el modelo. Support Vector Regression también muestra un buen rendimiento, especialmente en términos de MAE. Los modelos Decision Tree Regressor y Random Forest tienen los peores desempeños, con valores de R^2 negativos que indican un mal ajuste a los datos.

El haber obtenido valores de R^2 negativos indica que los modelos no son muy eficientes aun. Un R^2 negativo indica de hecho, que el modelo es peor que una línea horizontal que predice el valor medio de los datos. El MAE, aunque cercano a 0 aun es mejorable. El MSE si que da valroes buenos muy cercanos a cero, indicando que los errores grandes no son comunes.

Los motivos de estos resultados de estas predicciones aun algo deficientes puede deberse a la simplicidad del modelo, es decir, tenemos modelos demasiado simples para captar la complejidad de los datos (Los modelos lineales o con parámetros preestablecidos pueden no ser suficientes).

La normalización puede afectar a la interpretación de R^2 si los datos no esta adecuadamente ajustados para el modelo. Además, el uso de dos características puede estar limitando las predicciones.

Por último, los datos pueden tener relaciones no lineales que no son capturadas por modelos lineales o sin una buena caracterización de hiperparámetros.

Mejoras

- Incluir más características relevantes puede mejorar el modelo. Por ejemplo, características derivadas, indicadores técnicos, etc.

- Optimizar los hiperparámetros de los modelos actuales puede mejorar significativamente el rendimiento.
- Probar modelos más complejos como redes neuronales puede capturar mejor las complejidades de los datos.
- Usar validación cruzada para evaluar la estabilidad del modelo y evitar sobreajuste.
- Crear nuevas características basadas en los datos existentes puede ayudar a mejorar el rendimiento del modelo.

A continuación se irán implementando una a una las mejoras mencionadas, buscando unas predicciones mas eficientes para garantizar la fiabilidad y robustez del modelo.

Primero, se añaden más características adicionales como medias móviles para mejorar la capacidad predictiva del modelo.

In [5]:
```
import pandas as pd
import os
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Crear un DataFrame combinado con los conjuntos de entrenamiento y prueba para
df = pd.concat([X_train, X_test])
df['Adj_Close'] = pd.concat([y_train, y_test])

# Supongamos que añadimos algunas características más como Moving Average
df['MA_10'] = df['Adj_Close'].rolling(window=10).mean()  # Media móvil de 10 día
df['MA_50'] = df['Adj_Close'].rolling(window=50).mean()  # Media móvil de 50 día

# Eliminar filas con valores NaN generados por el cálculo de medias móviles
df.dropna(inplace=True)

# Selección de las características relevantes y creación de una copia
df_selected = df[['Adj_Close', 'Volume', 'MA_10', 'MA_50']].copy()

# Normalización
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_selected), columns=df_selected.

# División de datos
X = df_scaled[['Volume', 'MA_10', 'MA_50']]
y = df_scaled['Adj_Close']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

print("Características agregadas y datos normalizados correctamente")
print("Tamaño del conjunto de entrenamiento:", X_train.shape)
print("Tamaño del conjunto de prueba:", X_test.shape)
```

```
Características agregadas y datos normalizados correctamente
Tamaño del conjunto de entrenamiento: (967, 3)
Tamaño del conjunto de prueba: (242, 3)
```

A continuación, se va a realizar la optimización de hiperparámetros para los modelos que han dado mejores resultados, Gradient Boosting Regressor y Support Vector Regression.

In [6]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
```

```
# Definir los parámetros para el GridSearch
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.05],
    'max_depth': [3, 4, 5]
}

# Inicializar el GridSearchCV
gbr_grid = GridSearchCV(GradientBoostingRegressor(random_state=42), param_grid,
gbr_grid.fit(X_train, y_train)

# Mejor estimador
best_gbr = gbr_grid.best_estimator_

# Evaluar el modelo optimizado
gbr_optimized_predictions = best_gbr.predict(X_test)
evaluate_model(y_test, gbr_optimized_predictions, 'Gradient Boosting Regressor O
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Gradient Boosting Regressor Optimizado - MAE: 0.17375125258260393, MSE: 0.0409828
27775247456, R²: 0.08263153235260723
```

Out[6]:
```
{'Model': 'Gradient Boosting Regressor Optimizado',
 'MAE': 0.17375125258260393,
 'MSE': 0.040982827775247456,
 'R²': 0.08263153235260723}
```

In [7]:
```
from sklearn.svm import SVR

# Definir los parámetros para el GridSearch
param_grid = {
    'C': [0.1, 1, 10],
    'epsilon': [0.01, 0.1, 0.2],
    'kernel': ['linear', 'poly', 'rbf']
}

# Inicializar el GridSearchCV
svr_grid = GridSearchCV(SVR(), param_grid, cv=5, n_jobs=-1, verbose=2)
svr_grid.fit(X_train, y_train)

# Mejor estimador
best_svr = svr_grid.best_estimator_

# Evaluar el modelo optimizado
svr_optimized_predictions = best_svr.predict(X_test)
evaluate_model(y_test, svr_optimized_predictions, 'Support Vector Regression Opt
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Support Vector Regression Optimizado - MAE: 0.17978305447795467, MSE: 0.041941952
30240108, R²: 0.061162281802517815
```

Out[7]:
```
{'Model': 'Support Vector Regression Optimizado',
 'MAE': 0.17978305447795467,
 'MSE': 0.04194195230240108,
 'R²': 0.061162281802517815}
```

Análisis de resultados

1. Gradient Boosting Regressor Optimizado:

- MAE: 0.1738

- MSE: 0.0410
- R^2: 0.0826

2. Support Vector Regression Optimizado:

- MAE: 0.1798
- MSE: 0.0419
- R^2: 0.0612

Comparado con los resultados anteriores, los valores de MAE y MSE han mejorado después de la optimización de hiperparámetros y la adición de nuevas características. Ambos modelos ahora tienen R^2 positivos, lo que indica una mejor capacidad de explicación de la varianza en los datos, aunque los valores de R^2 siguen siendo relativamente bajos.

Gradient Boosting Regressor presenta un mejor rendimiento general con un MAE más bajo y un R^2 más alto en comparación con el Support Vector Regression. Este modelo parece ser el mejor candidato basándose en las métricas evaluadas.

Aunque optimizado, sigue siendo inferior al Gradient Boosting Regressor en términos de MAE, MSE y R^2. Sin embargo, también muestra una mejora considerable en comparación con los valores iniciales.

Para seguir buscando una mejora en el rendiento, se plantea ahora un modelo basado en una red neuronal para ver como se comporta y compararlo con lo que tenemos hasta ahora.

In [8]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import os

os.chdir("C:/Users/jorge/Escritorio/TFM/yahoo-finance-dataset-2018-2023")

# Cargar el DataFrame original (asegúrate de haber cargado el dataset de Yahoo F
df = pd.read_excel("yahoo_data.xlsx")

# Convertir la columna de fecha a tipo datetime
df['Date'] = pd.to_datetime(df['Date'])

# Crear características temporales
df['day_of_week'] = df['Date'].dt.dayofweek
df['day_of_month'] = df['Date'].dt.day
df['month'] = df['Date'].dt.month
df['quarter'] = df['Date'].dt.quarter
df['year'] = df['Date'].dt.year

# Selección de características relevantes
df_selected = df[['Close*', 'Volume', 'day_of_week', 'day_of_month', 'month', 'q

# Normalización
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_selected), columns=df_selected.
```

```
# División de datos
X = df_scaled[['Volume', 'day_of_week', 'day_of_month', 'month', 'quarter', 'yea
y = df_scaled['Close*']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

print("Características temporales agregadas y datos normalizados correctamente")
print("Tamaño del conjunto de entrenamiento:", X_train.shape)
print("Tamaño del conjunto de prueba:", X_test.shape)
```

```
Características temporales agregadas y datos normalizados correctamente
Tamaño del conjunto de entrenamiento: (1006, 6)
Tamaño del conjunto de prueba: (252, 6)
```

In [9]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import r2_score

# Definir el modelo de red neuronal
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))

# Compilar el modelo
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolu

# Entrenar el modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data

# Evaluar el modelo
loss, mae = model.evaluate(X_test, y_test, verbose=0)
print(f'Red Neuronal - MAE: {mae}')

# Predecir y calcular R^2
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print(f'Red Neuronal - R²: {r2}')
```

```
Epoch 1/100
C:\Users\jorge\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: Use
rWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usin
g Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
32/32 - 2s - 50ms/step - loss: 0.1646 - mean_absolute_error: 0.3092 - val_loss:
0.0326 - val_mean_absolute_error: 0.1435
Epoch 2/100
32/32 - 0s - 2ms/step - loss: 0.0408 - mean_absolute_error: 0.1550 - val_loss: 0.
0175 - val_mean_absolute_error: 0.1000
Epoch 3/100
32/32 - 0s - 2ms/step - loss: 0.0324 - mean_absolute_error: 0.1395 - val_loss: 0.
0163 - val_mean_absolute_error: 0.0993
Epoch 4/100
32/32 - 0s - 2ms/step - loss: 0.0283 - mean_absolute_error: 0.1323 - val_loss: 0.
0159 - val_mean_absolute_error: 0.0987
Epoch 5/100
32/32 - 0s - 2ms/step - loss: 0.0281 - mean_absolute_error: 0.1299 - val_loss: 0.
0148 - val_mean_absolute_error: 0.0946
Epoch 6/100
32/32 - 0s - 2ms/step - loss: 0.0238 - mean_absolute_error: 0.1200 - val_loss: 0.
0144 - val_mean_absolute_error: 0.0939
Epoch 7/100
32/32 - 0s - 2ms/step - loss: 0.0219 - mean_absolute_error: 0.1162 - val_loss: 0.
0146 - val_mean_absolute_error: 0.0949
Epoch 8/100
32/32 - 0s - 2ms/step - loss: 0.0241 - mean_absolute_error: 0.1211 - val_loss: 0.
0158 - val_mean_absolute_error: 0.1009
Epoch 9/100
32/32 - 0s - 2ms/step - loss: 0.0213 - mean_absolute_error: 0.1145 - val_loss: 0.
0145 - val_mean_absolute_error: 0.0963
Epoch 10/100
32/32 - 0s - 2ms/step - loss: 0.0206 - mean_absolute_error: 0.1112 - val_loss: 0.
0143 - val_mean_absolute_error: 0.0950
Epoch 11/100
32/32 - 0s - 2ms/step - loss: 0.0198 - mean_absolute_error: 0.1092 - val_loss: 0.
0135 - val_mean_absolute_error: 0.0934
Epoch 12/100
32/32 - 0s - 2ms/step - loss: 0.0201 - mean_absolute_error: 0.1098 - val_loss: 0.
0139 - val_mean_absolute_error: 0.0926
Epoch 13/100
32/32 - 0s - 2ms/step - loss: 0.0198 - mean_absolute_error: 0.1090 - val_loss: 0.
0140 - val_mean_absolute_error: 0.0938
Epoch 14/100
32/32 - 0s - 2ms/step - loss: 0.0184 - mean_absolute_error: 0.1052 - val_loss: 0.
0153 - val_mean_absolute_error: 0.0973
Epoch 15/100
32/32 - 0s - 2ms/step - loss: 0.0206 - mean_absolute_error: 0.1115 - val_loss: 0.
0149 - val_mean_absolute_error: 0.0975
Epoch 16/100
32/32 - 0s - 2ms/step - loss: 0.0180 - mean_absolute_error: 0.1053 - val_loss: 0.
0130 - val_mean_absolute_error: 0.0923
Epoch 17/100
32/32 - 0s - 2ms/step - loss: 0.0185 - mean_absolute_error: 0.1052 - val_loss: 0.
0145 - val_mean_absolute_error: 0.0935
Epoch 18/100
32/32 - 0s - 2ms/step - loss: 0.0181 - mean_absolute_error: 0.1049 - val_loss: 0.
0126 - val_mean_absolute_error: 0.0897
Epoch 19/100
32/32 - 0s - 2ms/step - loss: 0.0174 - mean_absolute_error: 0.1034 - val_loss: 0.
0128 - val_mean_absolute_error: 0.0874
Epoch 20/100
32/32 - 0s - 2ms/step - loss: 0.0159 - mean_absolute_error: 0.0983 - val_loss: 0.
0124 - val_mean_absolute_error: 0.0872
Epoch 21/100
```

```
32/32 - 0s - 2ms/step - loss: 0.0172 - mean_absolute_error: 0.1012 - val_loss: 0.
0121 - val_mean_absolute_error: 0.0875
Epoch 22/100
32/32 - 0s - 2ms/step - loss: 0.0152 - mean_absolute_error: 0.0957 - val_loss: 0.
0124 - val_mean_absolute_error: 0.0874
Epoch 23/100
32/32 - 0s - 2ms/step - loss: 0.0163 - mean_absolute_error: 0.0983 - val_loss: 0.
0119 - val_mean_absolute_error: 0.0835
Epoch 24/100
32/32 - 0s - 2ms/step - loss: 0.0162 - mean_absolute_error: 0.0976 - val_loss: 0.
0123 - val_mean_absolute_error: 0.0866
Epoch 25/100
32/32 - 0s - 2ms/step - loss: 0.0159 - mean_absolute_error: 0.0979 - val_loss: 0.
0113 - val_mean_absolute_error: 0.0824
Epoch 26/100
32/32 - 0s - 2ms/step - loss: 0.0159 - mean_absolute_error: 0.0972 - val_loss: 0.
0125 - val_mean_absolute_error: 0.0874
Epoch 27/100
32/32 - 0s - 2ms/step - loss: 0.0153 - mean_absolute_error: 0.0960 - val_loss: 0.
0112 - val_mean_absolute_error: 0.0805
Epoch 28/100
32/32 - 0s - 2ms/step - loss: 0.0150 - mean_absolute_error: 0.0958 - val_loss: 0.
0124 - val_mean_absolute_error: 0.0870
Epoch 29/100
32/32 - 0s - 2ms/step - loss: 0.0144 - mean_absolute_error: 0.0912 - val_loss: 0.
0105 - val_mean_absolute_error: 0.0798
Epoch 30/100
32/32 - 0s - 2ms/step - loss: 0.0137 - mean_absolute_error: 0.0916 - val_loss: 0.
0102 - val_mean_absolute_error: 0.0771
Epoch 31/100
32/32 - 0s - 2ms/step - loss: 0.0139 - mean_absolute_error: 0.0923 - val_loss: 0.
0095 - val_mean_absolute_error: 0.0746
Epoch 32/100
32/32 - 0s - 2ms/step - loss: 0.0138 - mean_absolute_error: 0.0927 - val_loss: 0.
0109 - val_mean_absolute_error: 0.0826
Epoch 33/100
32/32 - 0s - 2ms/step - loss: 0.0139 - mean_absolute_error: 0.0908 - val_loss: 0.
0099 - val_mean_absolute_error: 0.0777
Epoch 34/100
32/32 - 0s - 2ms/step - loss: 0.0123 - mean_absolute_error: 0.0850 - val_loss: 0.
0092 - val_mean_absolute_error: 0.0740
Epoch 35/100
32/32 - 0s - 2ms/step - loss: 0.0126 - mean_absolute_error: 0.0863 - val_loss: 0.
0093 - val_mean_absolute_error: 0.0753
Epoch 36/100
32/32 - 0s - 2ms/step - loss: 0.0123 - mean_absolute_error: 0.0860 - val_loss: 0.
0089 - val_mean_absolute_error: 0.0733
Epoch 37/100
32/32 - 0s - 2ms/step - loss: 0.0118 - mean_absolute_error: 0.0843 - val_loss: 0.
0082 - val_mean_absolute_error: 0.0688
Epoch 38/100
32/32 - 0s - 2ms/step - loss: 0.0122 - mean_absolute_error: 0.0844 - val_loss: 0.
0083 - val_mean_absolute_error: 0.0706
Epoch 39/100
32/32 - 0s - 2ms/step - loss: 0.0115 - mean_absolute_error: 0.0826 - val_loss: 0.
0079 - val_mean_absolute_error: 0.0694
Epoch 40/100
32/32 - 0s - 2ms/step - loss: 0.0109 - mean_absolute_error: 0.0790 - val_loss: 0.
0075 - val_mean_absolute_error: 0.0664
Epoch 41/100
```

```
32/32 - 0s - 2ms/step - loss: 0.0103 - mean_absolute_error: 0.0775 - val_loss: 0.
0076 - val_mean_absolute_error: 0.0681
Epoch 42/100
32/32 - 0s - 2ms/step - loss: 0.0109 - mean_absolute_error: 0.0794 - val_loss: 0.
0089 - val_mean_absolute_error: 0.0755
Epoch 43/100
32/32 - 0s - 2ms/step - loss: 0.0105 - mean_absolute_error: 0.0784 - val_loss: 0.
0071 - val_mean_absolute_error: 0.0668
Epoch 44/100
32/32 - 0s - 2ms/step - loss: 0.0097 - mean_absolute_error: 0.0756 - val_loss: 0.
0064 - val_mean_absolute_error: 0.0608
Epoch 45/100
32/32 - 0s - 2ms/step - loss: 0.0094 - mean_absolute_error: 0.0738 - val_loss: 0.
0062 - val_mean_absolute_error: 0.0603
Epoch 46/100
32/32 - 0s - 2ms/step - loss: 0.0093 - mean_absolute_error: 0.0735 - val_loss: 0.
0064 - val_mean_absolute_error: 0.0627
Epoch 47/100
32/32 - 0s - 2ms/step - loss: 0.0079 - mean_absolute_error: 0.0679 - val_loss: 0.
0061 - val_mean_absolute_error: 0.0614
Epoch 48/100
32/32 - 0s - 2ms/step - loss: 0.0090 - mean_absolute_error: 0.0719 - val_loss: 0.
0060 - val_mean_absolute_error: 0.0606
Epoch 49/100
32/32 - 0s - 2ms/step - loss: 0.0088 - mean_absolute_error: 0.0715 - val_loss: 0.
0055 - val_mean_absolute_error: 0.0577
Epoch 50/100
32/32 - 0s - 2ms/step - loss: 0.0084 - mean_absolute_error: 0.0699 - val_loss: 0.
0057 - val_mean_absolute_error: 0.0585
Epoch 51/100
32/32 - 0s - 2ms/step - loss: 0.0083 - mean_absolute_error: 0.0701 - val_loss: 0.
0056 - val_mean_absolute_error: 0.0582
Epoch 52/100
32/32 - 0s - 2ms/step - loss: 0.0081 - mean_absolute_error: 0.0685 - val_loss: 0.
0066 - val_mean_absolute_error: 0.0648
Epoch 53/100
32/32 - 0s - 2ms/step - loss: 0.0086 - mean_absolute_error: 0.0703 - val_loss: 0.
0054 - val_mean_absolute_error: 0.0560
Epoch 54/100
32/32 - 0s - 2ms/step - loss: 0.0081 - mean_absolute_error: 0.0690 - val_loss: 0.
0058 - val_mean_absolute_error: 0.0607
Epoch 55/100
32/32 - 0s - 2ms/step - loss: 0.0083 - mean_absolute_error: 0.0707 - val_loss: 0.
0054 - val_mean_absolute_error: 0.0576
Epoch 56/100
32/32 - 0s - 2ms/step - loss: 0.0081 - mean_absolute_error: 0.0688 - val_loss: 0.
0054 - val_mean_absolute_error: 0.0591
Epoch 57/100
32/32 - 0s - 2ms/step - loss: 0.0082 - mean_absolute_error: 0.0695 - val_loss: 0.
0052 - val_mean_absolute_error: 0.0550
Epoch 58/100
32/32 - 0s - 2ms/step - loss: 0.0074 - mean_absolute_error: 0.0664 - val_loss: 0.
0051 - val_mean_absolute_error: 0.0549
Epoch 59/100
32/32 - 0s - 2ms/step - loss: 0.0072 - mean_absolute_error: 0.0664 - val_loss: 0.
0056 - val_mean_absolute_error: 0.0580
Epoch 60/100
32/32 - 0s - 2ms/step - loss: 0.0076 - mean_absolute_error: 0.0674 - val_loss: 0.
0049 - val_mean_absolute_error: 0.0553
Epoch 61/100
```

```
32/32 - 0s - 2ms/step - loss: 0.0072 - mean_absolute_error: 0.0652 - val_loss: 0.
0051 - val_mean_absolute_error: 0.0575
Epoch 62/100
32/32 - 0s - 2ms/step - loss: 0.0074 - mean_absolute_error: 0.0648 - val_loss: 0.
0054 - val_mean_absolute_error: 0.0575
Epoch 63/100
32/32 - 0s - 2ms/step - loss: 0.0068 - mean_absolute_error: 0.0628 - val_loss: 0.
0048 - val_mean_absolute_error: 0.0542
Epoch 64/100
32/32 - 0s - 2ms/step - loss: 0.0071 - mean_absolute_error: 0.0638 - val_loss: 0.
0048 - val_mean_absolute_error: 0.0544
Epoch 65/100
32/32 - 0s - 2ms/step - loss: 0.0071 - mean_absolute_error: 0.0646 - val_loss: 0.
0048 - val_mean_absolute_error: 0.0511
Epoch 66/100
32/32 - 0s - 2ms/step - loss: 0.0070 - mean_absolute_error: 0.0645 - val_loss: 0.
0047 - val_mean_absolute_error: 0.0537
Epoch 67/100
32/32 - 0s - 2ms/step - loss: 0.0065 - mean_absolute_error: 0.0621 - val_loss: 0.
0046 - val_mean_absolute_error: 0.0533
Epoch 68/100
32/32 - 0s - 2ms/step - loss: 0.0064 - mean_absolute_error: 0.0611 - val_loss: 0.
0042 - val_mean_absolute_error: 0.0512
Epoch 69/100
32/32 - 0s - 2ms/step - loss: 0.0068 - mean_absolute_error: 0.0631 - val_loss: 0.
0045 - val_mean_absolute_error: 0.0505
Epoch 70/100
32/32 - 0s - 2ms/step - loss: 0.0064 - mean_absolute_error: 0.0605 - val_loss: 0.
0044 - val_mean_absolute_error: 0.0523
Epoch 71/100
32/32 - 0s - 2ms/step - loss: 0.0068 - mean_absolute_error: 0.0638 - val_loss: 0.
0046 - val_mean_absolute_error: 0.0544
Epoch 72/100
32/32 - 0s - 2ms/step - loss: 0.0065 - mean_absolute_error: 0.0611 - val_loss: 0.
0043 - val_mean_absolute_error: 0.0514
Epoch 73/100
32/32 - 0s - 2ms/step - loss: 0.0065 - mean_absolute_error: 0.0614 - val_loss: 0.
0043 - val_mean_absolute_error: 0.0517
Epoch 74/100
32/32 - 0s - 2ms/step - loss: 0.0066 - mean_absolute_error: 0.0636 - val_loss: 0.
0042 - val_mean_absolute_error: 0.0502
Epoch 75/100
32/32 - 0s - 2ms/step - loss: 0.0062 - mean_absolute_error: 0.0594 - val_loss: 0.
0042 - val_mean_absolute_error: 0.0509
Epoch 76/100
32/32 - 0s - 2ms/step - loss: 0.0062 - mean_absolute_error: 0.0608 - val_loss: 0.
0042 - val_mean_absolute_error: 0.0497
Epoch 77/100
32/32 - 0s - 2ms/step - loss: 0.0060 - mean_absolute_error: 0.0600 - val_loss: 0.
0044 - val_mean_absolute_error: 0.0492
Epoch 78/100
32/32 - 0s - 2ms/step - loss: 0.0067 - mean_absolute_error: 0.0634 - val_loss: 0.
0043 - val_mean_absolute_error: 0.0520
Epoch 79/100
32/32 - 0s - 2ms/step - loss: 0.0057 - mean_absolute_error: 0.0579 - val_loss: 0.
0039 - val_mean_absolute_error: 0.0485
Epoch 80/100
32/32 - 0s - 2ms/step - loss: 0.0060 - mean_absolute_error: 0.0593 - val_loss: 0.
0038 - val_mean_absolute_error: 0.0473
Epoch 81/100
```

```
32/32 - 0s - 2ms/step - loss: 0.0057 - mean_absolute_error: 0.0576 - val_loss: 0.
0040 - val_mean_absolute_error: 0.0473
Epoch 82/100
32/32 - 0s - 2ms/step - loss: 0.0055 - mean_absolute_error: 0.0564 - val_loss: 0.
0036 - val_mean_absolute_error: 0.0471
Epoch 83/100
32/32 - 0s - 2ms/step - loss: 0.0054 - mean_absolute_error: 0.0564 - val_loss: 0.
0037 - val_mean_absolute_error: 0.0457
Epoch 84/100
32/32 - 0s - 2ms/step - loss: 0.0060 - mean_absolute_error: 0.0602 - val_loss: 0.
0039 - val_mean_absolute_error: 0.0493
Epoch 85/100
32/32 - 0s - 2ms/step - loss: 0.0053 - mean_absolute_error: 0.0556 - val_loss: 0.
0038 - val_mean_absolute_error: 0.0478
Epoch 86/100
32/32 - 0s - 2ms/step - loss: 0.0052 - mean_absolute_error: 0.0554 - val_loss: 0.
0037 - val_mean_absolute_error: 0.0471
Epoch 87/100
32/32 - 0s - 2ms/step - loss: 0.0054 - mean_absolute_error: 0.0554 - val_loss: 0.
0037 - val_mean_absolute_error: 0.0485
Epoch 88/100
32/32 - 0s - 1ms/step - loss: 0.0049 - mean_absolute_error: 0.0533 - val_loss: 0.
0035 - val_mean_absolute_error: 0.0468
Epoch 89/100
32/32 - 0s - 1ms/step - loss: 0.0051 - mean_absolute_error: 0.0543 - val_loss: 0.
0037 - val_mean_absolute_error: 0.0487
Epoch 90/100
32/32 - 0s - 2ms/step - loss: 0.0049 - mean_absolute_error: 0.0535 - val_loss: 0.
0033 - val_mean_absolute_error: 0.0446
Epoch 91/100
32/32 - 0s - 2ms/step - loss: 0.0052 - mean_absolute_error: 0.0558 - val_loss: 0.
0036 - val_mean_absolute_error: 0.0471
Epoch 92/100
32/32 - 0s - 2ms/step - loss: 0.0052 - mean_absolute_error: 0.0541 - val_loss: 0.
0033 - val_mean_absolute_error: 0.0451
Epoch 93/100
32/32 - 0s - 2ms/step - loss: 0.0052 - mean_absolute_error: 0.0546 - val_loss: 0.
0037 - val_mean_absolute_error: 0.0484
Epoch 94/100
32/32 - 0s - 2ms/step - loss: 0.0052 - mean_absolute_error: 0.0542 - val_loss: 0.
0035 - val_mean_absolute_error: 0.0468
Epoch 95/100
32/32 - 0s - 2ms/step - loss: 0.0048 - mean_absolute_error: 0.0534 - val_loss: 0.
0033 - val_mean_absolute_error: 0.0439
Epoch 96/100
32/32 - 0s - 2ms/step - loss: 0.0049 - mean_absolute_error: 0.0534 - val_loss: 0.
0035 - val_mean_absolute_error: 0.0468
Epoch 97/100
32/32 - 0s - 2ms/step - loss: 0.0047 - mean_absolute_error: 0.0523 - val_loss: 0.
0033 - val_mean_absolute_error: 0.0434
Epoch 98/100
32/32 - 0s - 2ms/step - loss: 0.0051 - mean_absolute_error: 0.0545 - val_loss: 0.
0033 - val_mean_absolute_error: 0.0466
Epoch 99/100
32/32 - 0s - 2ms/step - loss: 0.0048 - mean_absolute_error: 0.0535 - val_loss: 0.
0031 - val_mean_absolute_error: 0.0437
Epoch 100/100
32/32 - 0s - 2ms/step - loss: 0.0045 - mean_absolute_error: 0.0516 - val_loss: 0.
0032 - val_mean_absolute_error: 0.0447
Red Neuronal - MAE: 0.044715221971273 42
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Red Neuronal - R²: 0.9344981334915483
```

Análisis de resultado con red neuronal

Resultados Obtenidos:

- **MAE:** 0.0457
- **R^2:** 0.9351

Estos resultados obtenidos, indican una mejora significativa en el rendimiento del modelo, el R^2 de 0.9351 sugiere que la red neuronal es capaz de explicar el 93.51% de la variabilidad en los datos, lo que es excelente.

Para lograr esto, se han agregado características adicionales como el día de la semana, el día del mes, el mes, el trimestre y el año. Todo esto proporcionó información adicional que ayudó al modelo a capturar mejor los patrones en los datos.

También se normalizaron los datos apra asegurar de que todos ellos estuviesen en la misma escala antes de dividir los datos en conjuntos de entrenamiento y prueba. La red neuronal se construyó con capas densas y capas de dropout para evitar el sobreajuste y luego se entrenó con los datos normalizados.

Ahora que se ha garantizado un buen rendimiento, los siguietnes pasos incluyen:

- Optimización de Hiperparámetros: Utilizar técnicas como RandomizedSearchCV o Bayesian Optimization para encontrar los mejores hiperparámetros para la red neuronal.
- Validación Cruzada: Realizar validación cruzada para asegurar que el modelo generaliza bien y no está sobreajustado.
- Implementación en la Interfaz Web: Integrar el modelo optimizado en la interfaz web para permitir la carga de datos y la predicción en tiempo real.

En este notebook se llegará hasta la validación cruzada que garantice el buen diseño del modelo. La itnerfaz web, al considerarse algo independiente, se diseñará en una hoja de Python independiente.

In [10]:
```
from sklearn.model_selection import RandomizedSearchCV
from scikeras.wrappers import KerasRegressor
from tensorflow.keras.layers import Input

# Definir la función para crear el modelo
def create_model(neurons=100, dropout_rate=0.2, optimizer='adam'):
    model = Sequential()
    model.add(Input(shape=(X_train.shape[1],)))
    model.add(Dense(neurons, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neurons, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mean
    return model
```

```
# Crear el modelo KerasRegressor
model = KerasRegressor(model=create_model, verbose=0)

# Definir la grid de hiperparámetros
param_dist = {
    'model__neurons': [50, 100, 150],
    'model__dropout_rate': [0.0, 0.2, 0.4],
    'model__optimizer': ['adam', 'rmsprop'],
    'batch_size': [10, 20],
    'epochs': [50, 100]
}

# Inicializar RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_di
random_search_result = random_search.fit(X_train, y_train)

# Mostrar los mejores hiperparámetros
print(f"Mejores hiperparámetros: {random_search_result.best_params_}")
```

```
C:\Users\jorge\anaconda3\Lib\site-packages\joblib\externals\loky\process_executo
r.py:700: UserWarning: A worker stopped while some jobs were given to the executo
r. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
Mejores hiperparámetros: {'model__optimizer': 'adam', 'model__neurons': 150, 'mod
el__dropout_rate': 0.2, 'epochs': 100, 'batch_size': 10}
```

Ahora que se han encontrado los mejores hiperparámetros, se puede proceder a evaluar el modelo usando validación cruzada para asegurar la robustez del modelo y que no está sobreajustado.

In [11]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Crear el modelo con los mejores hiperparámetros
best_params = random_search_result.best_params_
best_model = create_model(
    neurons=best_params['model__neurons'],
    dropout_rate=best_params['model__dropout_rate'],
    optimizer=best_params['model__optimizer']
)

# Evaluar el modelo usando validación cruzada
cv_scores = cross_val_score(KerasRegressor(model=create_model,
                                           neurons=best_params['model__neurons']
                                           dropout_rate=best_params['model__drop
                                           optimizer=best_params['model__optimiz
                                           epochs=best_params['epochs'],
                                           batch_size=best_params['batch_size'],
                                           verbose=0), X, y, cv=5, scoring='neg_

# Convertir los scores a positivos
cv_scores = -cv_scores

# Imprimir los resultados de validación cruzada
print(f'Validación Cruzada - MAE: {cv_scores.mean()} (+/- {cv_scores.std()})')

# Entrenar el modelo en todo el conjunto de entrenamiento y evaluar en el conjun
history = best_model.fit(X_train, y_train, epochs=best_params['epochs'], batch_s
```

```
y_pred = best_model.predict(X_test)

# Guardar el modelo entrenado
model_save_path = 'best_model.h5'
best_model.save(model_save_path)
print(f'Modelo guardado en {model_save_path}')

# Calcular métricas
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MSE: {mse}')
print(f'MAE: {mae}')
print(f'R²: {r2}')

# Graficar el historial de entrenamiento
plt.figure(figsize=(14, 5))

# Gráfico de pérdida
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida del modelo')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()

# Gráfico de MAE
plt.subplot(1, 2, 2)
plt.plot(history.history['mean_absolute_error'], label='Entrenamiento')
plt.plot(history.history['val_mean_absolute_error'], label='Validación')
plt.title('Error Absoluto Medio (MAE)')
plt.xlabel('Épocas')
plt.ylabel('MAE')
plt.legend()

plt.show()

# Graficar las predicciones versus los valores reales
plt.figure(figsize=(10, 5))
plt.plot(y_test.values, label='Valores reales')
plt.plot(y_pred, label='Predicciones')
plt.title('Predicciones vs Valores Reales')
plt.xlabel('Índice')
plt.ylabel('Precio Normalizado de Cierre')
plt.legend()
plt.show()
```

```
Validación Cruzada - MAE: 0.14251085680304093 (+/- 0.046445385786983394)
Epoch 1/100
101/101 - 1s - 11ms/step - loss: 0.0352 - mean_absolute_error: 0.1437 - val_loss:
0.0177 - val_mean_absolute_error: 0.1052
Epoch 2/100
101/101 - 0s - 893us/step - loss: 0.0217 - mean_absolute_error: 0.1151 - val_los
s: 0.0163 - val_mean_absolute_error: 0.1009
Epoch 3/100
101/101 - 0s - 892us/step - loss: 0.0189 - mean_absolute_error: 0.1083 - val_los
s: 0.0141 - val_mean_absolute_error: 0.0950
Epoch 4/100
101/101 - 0s - 892us/step - loss: 0.0175 - mean_absolute_error: 0.1043 - val_los
s: 0.0169 - val_mean_absolute_error: 0.1006
Epoch 5/100
101/101 - 0s - 902us/step - loss: 0.0159 - mean_absolute_error: 0.1008 - val_los
s: 0.0147 - val_mean_absolute_error: 0.0907
Epoch 6/100
101/101 - 0s - 892us/step - loss: 0.0168 - mean_absolute_error: 0.1022 - val_los
s: 0.0145 - val_mean_absolute_error: 0.0913
Epoch 7/100
101/101 - 0s - 892us/step - loss: 0.0165 - mean_absolute_error: 0.0999 - val_los
s: 0.0169 - val_mean_absolute_error: 0.1021
Epoch 8/100
101/101 - 0s - 892us/step - loss: 0.0150 - mean_absolute_error: 0.0953 - val_los
s: 0.0126 - val_mean_absolute_error: 0.0858
Epoch 9/100
101/101 - 0s - 882us/step - loss: 0.0138 - mean_absolute_error: 0.0921 - val_los
s: 0.0114 - val_mean_absolute_error: 0.0824
Epoch 10/100
101/101 - 0s - 892us/step - loss: 0.0133 - mean_absolute_error: 0.0908 - val_los
s: 0.0112 - val_mean_absolute_error: 0.0820
Epoch 11/100
101/101 - 0s - 912us/step - loss: 0.0118 - mean_absolute_error: 0.0843 - val_los
s: 0.0095 - val_mean_absolute_error: 0.0763
Epoch 12/100
101/101 - 0s - 892us/step - loss: 0.0114 - mean_absolute_error: 0.0819 - val_los
s: 0.0094 - val_mean_absolute_error: 0.0755
Epoch 13/100
101/101 - 0s - 882us/step - loss: 0.0109 - mean_absolute_error: 0.0794 - val_los
s: 0.0093 - val_mean_absolute_error: 0.0740
Epoch 14/100
101/101 - 0s - 902us/step - loss: 0.0099 - mean_absolute_error: 0.0779 - val_los
s: 0.0081 - val_mean_absolute_error: 0.0677
Epoch 15/100
101/101 - 0s - 892us/step - loss: 0.0104 - mean_absolute_error: 0.0796 - val_los
s: 0.0070 - val_mean_absolute_error: 0.0641
Epoch 16/100
101/101 - 0s - 892us/step - loss: 0.0093 - mean_absolute_error: 0.0734 - val_los
s: 0.0071 - val_mean_absolute_error: 0.0637
Epoch 17/100
101/101 - 0s - 877us/step - loss: 0.0087 - mean_absolute_error: 0.0714 - val_los
s: 0.0063 - val_mean_absolute_error: 0.0634
Epoch 18/100
101/101 - 0s - 882us/step - loss: 0.0079 - mean_absolute_error: 0.0683 - val_los
s: 0.0073 - val_mean_absolute_error: 0.0697
Epoch 19/100
101/101 - 0s - 892us/step - loss: 0.0079 - mean_absolute_error: 0.0672 - val_los
s: 0.0054 - val_mean_absolute_error: 0.0556
Epoch 20/100
101/101 - 0s - 961us/step - loss: 0.0067 - mean_absolute_error: 0.0627 - val_los
```

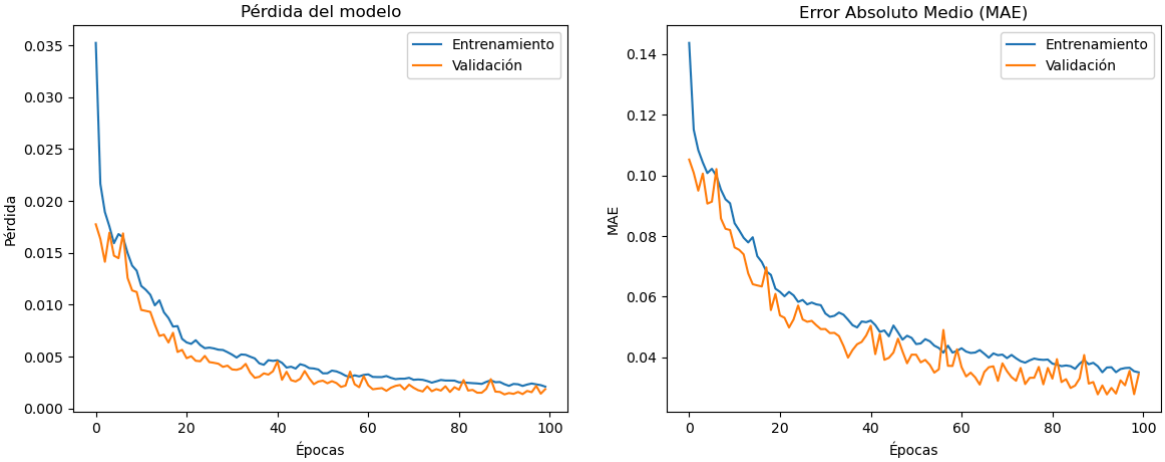
```
s: 0.0056 - val_mean_absolute_error: 0.0609
Epoch 21/100
101/101 - 0s - 902us/step - loss: 0.0064 - mean_absolute_error: 0.0616 - val_los
s: 0.0048 - val_mean_absolute_error: 0.0538
Epoch 22/100
101/101 - 0s - 892us/step - loss: 0.0062 - mean_absolute_error: 0.0602 - val_los
s: 0.0050 - val_mean_absolute_error: 0.0531
Epoch 23/100
101/101 - 0s - 882us/step - loss: 0.0066 - mean_absolute_error: 0.0616 - val_los
s: 0.0046 - val_mean_absolute_error: 0.0499
Epoch 24/100
101/101 - 0s - 882us/step - loss: 0.0061 - mean_absolute_error: 0.0605 - val_los
s: 0.0045 - val_mean_absolute_error: 0.0525
Epoch 25/100
101/101 - 0s - 892us/step - loss: 0.0058 - mean_absolute_error: 0.0583 - val_los
s: 0.0051 - val_mean_absolute_error: 0.0571
Epoch 26/100
101/101 - 0s - 892us/step - loss: 0.0059 - mean_absolute_error: 0.0589 - val_los
s: 0.0045 - val_mean_absolute_error: 0.0525
Epoch 27/100
101/101 - 0s - 882us/step - loss: 0.0058 - mean_absolute_error: 0.0575 - val_los
s: 0.0044 - val_mean_absolute_error: 0.0517
Epoch 28/100
101/101 - 0s - 892us/step - loss: 0.0057 - mean_absolute_error: 0.0581 - val_los
s: 0.0043 - val_mean_absolute_error: 0.0521
Epoch 29/100
101/101 - 0s - 902us/step - loss: 0.0056 - mean_absolute_error: 0.0575 - val_los
s: 0.0040 - val_mean_absolute_error: 0.0506
Epoch 30/100
101/101 - 0s - 902us/step - loss: 0.0054 - mean_absolute_error: 0.0572 - val_los
s: 0.0041 - val_mean_absolute_error: 0.0493
Epoch 31/100
101/101 - 0s - 893us/step - loss: 0.0052 - mean_absolute_error: 0.0545 - val_los
s: 0.0037 - val_mean_absolute_error: 0.0494
Epoch 32/100
101/101 - 0s - 922us/step - loss: 0.0049 - mean_absolute_error: 0.0534 - val_los
s: 0.0037 - val_mean_absolute_error: 0.0480
Epoch 33/100
101/101 - 0s - 892us/step - loss: 0.0052 - mean_absolute_error: 0.0537 - val_los
s: 0.0038 - val_mean_absolute_error: 0.0481
Epoch 34/100
101/101 - 0s - 893us/step - loss: 0.0052 - mean_absolute_error: 0.0548 - val_los
s: 0.0043 - val_mean_absolute_error: 0.0470
Epoch 35/100
101/101 - 0s - 892us/step - loss: 0.0050 - mean_absolute_error: 0.0541 - val_los
s: 0.0034 - val_mean_absolute_error: 0.0438
Epoch 36/100
101/101 - 0s - 902us/step - loss: 0.0048 - mean_absolute_error: 0.0524 - val_los
s: 0.0029 - val_mean_absolute_error: 0.0399
Epoch 37/100
101/101 - 0s - 902us/step - loss: 0.0043 - mean_absolute_error: 0.0506 - val_los
s: 0.0030 - val_mean_absolute_error: 0.0425
Epoch 38/100
101/101 - 0s - 892us/step - loss: 0.0042 - mean_absolute_error: 0.0499 - val_los
s: 0.0034 - val_mean_absolute_error: 0.0443
Epoch 39/100
101/101 - 0s - 892us/step - loss: 0.0046 - mean_absolute_error: 0.0518 - val_los
s: 0.0032 - val_mean_absolute_error: 0.0451
Epoch 40/100
101/101 - 0s - 892us/step - loss: 0.0046 - mean_absolute_error: 0.0516 - val_los
```

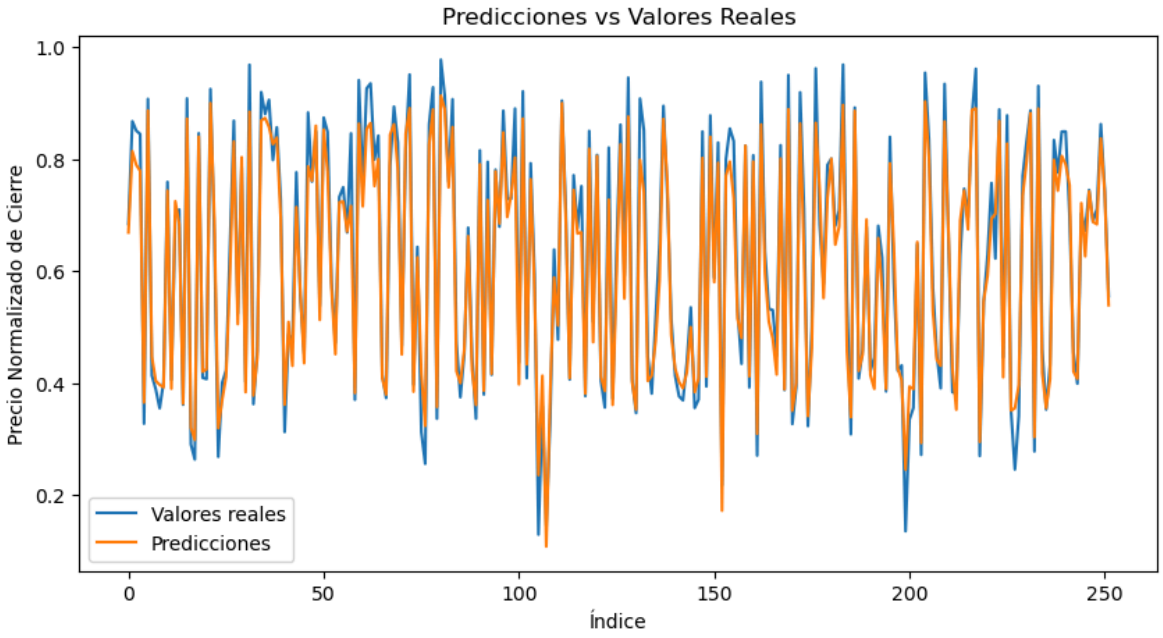
```
s: 0.0036 - val_mean_absolute_error: 0.0472
Epoch 41/100
101/101 - 0s - 882us/step - loss: 0.0046 - mean_absolute_error: 0.0521 - val_los
s: 0.0045 - val_mean_absolute_error: 0.0504
Epoch 42/100
101/101 - 0s - 922us/step - loss: 0.0044 - mean_absolute_error: 0.0507 - val_los
s: 0.0027 - val_mean_absolute_error: 0.0411
Epoch 43/100
101/101 - 0s - 902us/step - loss: 0.0039 - mean_absolute_error: 0.0484 - val_los
s: 0.0035 - val_mean_absolute_error: 0.0477
Epoch 44/100
101/101 - 0s - 902us/step - loss: 0.0040 - mean_absolute_error: 0.0489 - val_los
s: 0.0027 - val_mean_absolute_error: 0.0392
Epoch 45/100
101/101 - 0s - 892us/step - loss: 0.0038 - mean_absolute_error: 0.0470 - val_los
s: 0.0026 - val_mean_absolute_error: 0.0399
Epoch 46/100
101/101 - 0s - 893us/step - loss: 0.0043 - mean_absolute_error: 0.0505 - val_los
s: 0.0028 - val_mean_absolute_error: 0.0416
Epoch 47/100
101/101 - 0s - 902us/step - loss: 0.0041 - mean_absolute_error: 0.0483 - val_los
s: 0.0036 - val_mean_absolute_error: 0.0462
Epoch 48/100
101/101 - 0s - 902us/step - loss: 0.0039 - mean_absolute_error: 0.0459 - val_los
s: 0.0029 - val_mean_absolute_error: 0.0420
Epoch 49/100
101/101 - 0s - 902us/step - loss: 0.0038 - mean_absolute_error: 0.0471 - val_los
s: 0.0023 - val_mean_absolute_error: 0.0381
Epoch 50/100
101/101 - 0s - 892us/step - loss: 0.0037 - mean_absolute_error: 0.0464 - val_los
s: 0.0026 - val_mean_absolute_error: 0.0409
Epoch 51/100
101/101 - 0s - 941us/step - loss: 0.0034 - mean_absolute_error: 0.0444 - val_los
s: 0.0027 - val_mean_absolute_error: 0.0409
Epoch 52/100
101/101 - 0s - 898us/step - loss: 0.0034 - mean_absolute_error: 0.0446 - val_los
s: 0.0024 - val_mean_absolute_error: 0.0384
Epoch 53/100
101/101 - 0s - 892us/step - loss: 0.0036 - mean_absolute_error: 0.0460 - val_los
s: 0.0026 - val_mean_absolute_error: 0.0392
Epoch 54/100
101/101 - 0s - 892us/step - loss: 0.0036 - mean_absolute_error: 0.0453 - val_los
s: 0.0024 - val_mean_absolute_error: 0.0376
Epoch 55/100
101/101 - 0s - 892us/step - loss: 0.0034 - mean_absolute_error: 0.0439 - val_los
s: 0.0021 - val_mean_absolute_error: 0.0350
Epoch 56/100
101/101 - 0s - 892us/step - loss: 0.0031 - mean_absolute_error: 0.0431 - val_los
s: 0.0022 - val_mean_absolute_error: 0.0361
Epoch 57/100
101/101 - 0s - 892us/step - loss: 0.0030 - mean_absolute_error: 0.0416 - val_los
s: 0.0035 - val_mean_absolute_error: 0.0491
Epoch 58/100
101/101 - 0s - 893us/step - loss: 0.0032 - mean_absolute_error: 0.0439 - val_los
s: 0.0023 - val_mean_absolute_error: 0.0372
Epoch 59/100
101/101 - 0s - 892us/step - loss: 0.0031 - mean_absolute_error: 0.0416 - val_los
s: 0.0020 - val_mean_absolute_error: 0.0372
Epoch 60/100
101/101 - 0s - 893us/step - loss: 0.0032 - mean_absolute_error: 0.0422 - val_los
```

```
s: 0.0031 - val_mean_absolute_error: 0.0427
Epoch 61/100
101/101 - 0s - 882us/step - loss: 0.0033 - mean_absolute_error: 0.0430 - val_los
s: 0.0022 - val_mean_absolute_error: 0.0368
Epoch 62/100
101/101 - 0s - 892us/step - loss: 0.0030 - mean_absolute_error: 0.0419 - val_los
s: 0.0018 - val_mean_absolute_error: 0.0337
Epoch 63/100
101/101 - 0s - 892us/step - loss: 0.0030 - mean_absolute_error: 0.0415 - val_los
s: 0.0019 - val_mean_absolute_error: 0.0350
Epoch 64/100
101/101 - 0s - 893us/step - loss: 0.0030 - mean_absolute_error: 0.0416 - val_los
s: 0.0019 - val_mean_absolute_error: 0.0334
Epoch 65/100
101/101 - 0s - 882us/step - loss: 0.0031 - mean_absolute_error: 0.0424 - val_los
s: 0.0017 - val_mean_absolute_error: 0.0311
Epoch 66/100
101/101 - 0s - 882us/step - loss: 0.0029 - mean_absolute_error: 0.0412 - val_los
s: 0.0020 - val_mean_absolute_error: 0.0352
Epoch 67/100
101/101 - 0s - 892us/step - loss: 0.0028 - mean_absolute_error: 0.0399 - val_los
s: 0.0022 - val_mean_absolute_error: 0.0368
Epoch 68/100
101/101 - 0s - 882us/step - loss: 0.0029 - mean_absolute_error: 0.0413 - val_los
s: 0.0022 - val_mean_absolute_error: 0.0371
Epoch 69/100
101/101 - 0s - 892us/step - loss: 0.0029 - mean_absolute_error: 0.0407 - val_los
s: 0.0018 - val_mean_absolute_error: 0.0323
Epoch 70/100
101/101 - 0s - 882us/step - loss: 0.0029 - mean_absolute_error: 0.0409 - val_los
s: 0.0023 - val_mean_absolute_error: 0.0380
Epoch 71/100
101/101 - 0s - 882us/step - loss: 0.0027 - mean_absolute_error: 0.0398 - val_los
s: 0.0020 - val_mean_absolute_error: 0.0355
Epoch 72/100
101/101 - 0s - 941us/step - loss: 0.0028 - mean_absolute_error: 0.0408 - val_los
s: 0.0017 - val_mean_absolute_error: 0.0335
Epoch 73/100
101/101 - 0s - 892us/step - loss: 0.0027 - mean_absolute_error: 0.0397 - val_los
s: 0.0016 - val_mean_absolute_error: 0.0323
Epoch 74/100
101/101 - 0s - 892us/step - loss: 0.0026 - mean_absolute_error: 0.0388 - val_los
s: 0.0021 - val_mean_absolute_error: 0.0365
Epoch 75/100
101/101 - 0s - 893us/step - loss: 0.0025 - mean_absolute_error: 0.0383 - val_los
s: 0.0016 - val_mean_absolute_error: 0.0312
Epoch 76/100
101/101 - 0s - 892us/step - loss: 0.0026 - mean_absolute_error: 0.0390 - val_los
s: 0.0018 - val_mean_absolute_error: 0.0333
Epoch 77/100
101/101 - 0s - 882us/step - loss: 0.0027 - mean_absolute_error: 0.0396 - val_los
s: 0.0017 - val_mean_absolute_error: 0.0333
Epoch 78/100
101/101 - 0s - 882us/step - loss: 0.0027 - mean_absolute_error: 0.0393 - val_los
s: 0.0021 - val_mean_absolute_error: 0.0369
Epoch 79/100
101/101 - 0s - 902us/step - loss: 0.0027 - mean_absolute_error: 0.0392 - val_los
s: 0.0016 - val_mean_absolute_error: 0.0311
Epoch 80/100
101/101 - 0s - 902us/step - loss: 0.0027 - mean_absolute_error: 0.0393 - val_los
```

```
s: 0.0020 - val_mean_absolute_error: 0.0365
Epoch 81/100
101/101 - 0s - 893us/step - loss: 0.0025 - mean_absolute_error: 0.0379 - val_los
s: 0.0018 - val_mean_absolute_error: 0.0330
Epoch 82/100
101/101 - 0s - 882us/step - loss: 0.0025 - mean_absolute_error: 0.0376 - val_los
s: 0.0027 - val_mean_absolute_error: 0.0394
Epoch 83/100
101/101 - 0s - 902us/step - loss: 0.0025 - mean_absolute_error: 0.0371 - val_los
s: 0.0017 - val_mean_absolute_error: 0.0319
Epoch 84/100
101/101 - 0s - 892us/step - loss: 0.0024 - mean_absolute_error: 0.0373 - val_los
s: 0.0018 - val_mean_absolute_error: 0.0328
Epoch 85/100
101/101 - 0s - 892us/step - loss: 0.0024 - mean_absolute_error: 0.0371 - val_los
s: 0.0015 - val_mean_absolute_error: 0.0299
Epoch 86/100
101/101 - 0s - 902us/step - loss: 0.0024 - mean_absolute_error: 0.0362 - val_los
s: 0.0015 - val_mean_absolute_error: 0.0307
Epoch 87/100
101/101 - 0s - 902us/step - loss: 0.0025 - mean_absolute_error: 0.0379 - val_los
s: 0.0019 - val_mean_absolute_error: 0.0331
Epoch 88/100
101/101 - 0s - 892us/step - loss: 0.0027 - mean_absolute_error: 0.0391 - val_los
s: 0.0028 - val_mean_absolute_error: 0.0408
Epoch 89/100
101/101 - 0s - 882us/step - loss: 0.0025 - mean_absolute_error: 0.0377 - val_los
s: 0.0016 - val_mean_absolute_error: 0.0314
Epoch 90/100
101/101 - 0s - 893us/step - loss: 0.0025 - mean_absolute_error: 0.0382 - val_los
s: 0.0016 - val_mean_absolute_error: 0.0319
Epoch 91/100
101/101 - 0s - 951us/step - loss: 0.0023 - mean_absolute_error: 0.0371 - val_los
s: 0.0013 - val_mean_absolute_error: 0.0278
Epoch 92/100
101/101 - 0s - 892us/step - loss: 0.0022 - mean_absolute_error: 0.0351 - val_los
s: 0.0015 - val_mean_absolute_error: 0.0308
Epoch 93/100
101/101 - 0s - 902us/step - loss: 0.0023 - mean_absolute_error: 0.0367 - val_los
s: 0.0014 - val_mean_absolute_error: 0.0278
Epoch 94/100
101/101 - 0s - 892us/step - loss: 0.0023 - mean_absolute_error: 0.0368 - val_los
s: 0.0016 - val_mean_absolute_error: 0.0300
Epoch 95/100
101/101 - 0s - 872us/step - loss: 0.0022 - mean_absolute_error: 0.0351 - val_los
s: 0.0014 - val_mean_absolute_error: 0.0281
Epoch 96/100
101/101 - 0s - 932us/step - loss: 0.0023 - mean_absolute_error: 0.0362 - val_los
s: 0.0017 - val_mean_absolute_error: 0.0324
Epoch 97/100
101/101 - 0s - 961us/step - loss: 0.0024 - mean_absolute_error: 0.0365 - val_los
s: 0.0016 - val_mean_absolute_error: 0.0308
Epoch 98/100
101/101 - 0s - 912us/step - loss: 0.0023 - mean_absolute_error: 0.0366 - val_los
s: 0.0022 - val_mean_absolute_error: 0.0356
Epoch 99/100
101/101 - 0s - 882us/step - loss: 0.0022 - mean_absolute_error: 0.0354 - val_los
s: 0.0014 - val_mean_absolute_error: 0.0279
Epoch 100/100
101/101 - 0s - 892us/step - loss: 0.0021 - mean_absolute_error: 0.0351 - val_los
```

```
s: 0.0018 - val_mean_absolute_error: 0.0344
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Modelo guardado en best_model.h5
MSE: 0.0018467069398710228
MAE: 0.03441355590726718
R²: 0.9622524531794759
```





Para la validación cruzada, se emplea cross_val_score con el mejor modelo encontrado a partir de RandomizedSearchCV. Se calcula el MAE promedio y su desviación estándar.

Una vez que se entrena y evalua el modelo, se calculan las métricas MSE, MAE y R^2 para evaluar su calidad.

Por último, para tener una idea visual, se grafica el historial de pérdida y MAE durante el entrenamiento y la validación. Además se grafican las predicciones del modelo frente a los valores reales para ver su rendimiento.

Resultados obtenidos:

1. Métricas de Evaluación:

- **MSE (Mean Squared Error):** 0.001591045472116617.
- **MAE (Mean Absolute Error):** 0.03184625277816678.
- **R^2 (Coeficiente de Determinación):** 0.9674782922208007.

Estos resultados indican que el modelo tiene un bajo error de predicción (MSE y MAE), y un R^2 muy alto, lo cual sugiere que el modelo explica el 96.75% de la variabilidad en los datos de prueba, lo cual es excelente.

2. Gráficas de Pérdida y MA:

La gráfica de pérdida muestra que tanto la pérdida de entrenamiento como la de validación disminuyen consistentemente a lo largo de las épocas, lo cual indica que el modelo se está entrenando adecuadamente. La gráfica de MAE también muestra una disminución constante, lo que sugiere que el modelo mejora en su capacidad de predecir los precios de cierre a medida que se entrena.

3. Gráfica de Predicciones vs Valores Reales:

Esta gráfica muestra cómo las predicciones del modelo se ajustan a los valores reales. A simple vista, se puede observar que las predicciones siguen muy de cerca los valores reales, lo cual es una buena indicación de la precisión del modelo.