MODELO ENSAMBLADO

Preprocesamiento de Datos

```
# Importamos las librerías
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV, TimeSeriesSplit
from sklearn.feature_selection import mutual_info_regression, SelectKBest, f_regression
from sklearn.svm import SVR
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import mean_absolute_percentage_error, mean_squared_error
import plotly.graph_objects as go


# Limpieza de Datos
def clean_data(data):
    data = data.dropna()
    return data


# Normalización de las Variables
def normalize_data(data):
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data)
    return pd.DataFrame(data_scaled, columns=data.columns), scaler


# Selección de Variables utilizando diferentes métodos
def select_features(X, y, num_features):
    mutual_info = mutual_info_regression(X, y)
    k_best = SelectKBest(score_func=f_regression, k=num_features).fit(X, y)
    features = X.columns[k_best.get_support(indices=True)]
    return features.tolist()


# Cargar datos
data = pd.read_csv('https://query1.finance.yahoo.com/v7/finance/download/FSM?period1=1597123200&period2=1628659200&interval=1d&events=histor


# Mantener la columna de fechas para las gráficas
dates = data['Date']
data = data.drop(columns=['Date'])


# Limpiar y Normalizar
data = clean_data(data)
data, scaler = normalize_data(data)


# Seleccionar Variables
target_column = 'Close'
num_features = 5  # Número de características a seleccionar
selected_features = select_features(data.drop(columns=[target_column]), data[target_column], num_features)
selected_features.append(target_column)
data = data[selected_features]


# Separar características y objetivo
X = data.drop(columns=[target_column])
y = data[target_column]


# Dividir los datos en conjuntos de entrenamiento y prueba
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
dates_train, dates_test = dates[:train_size], dates[train_size:]


print(f'Características seleccionadas: {selected_features}')
```

```
Características seleccionadas: ['Open', 'High', 'Low', 'Adj Close', 'Volume', 'Close']
```

Entrenamiento y Validación Corregido

```
# Optimización del Modelo SVM
def optimize_svm(X_train, y_train):
    param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'kernel': ['rbf']}
    grid = GridSearchCV(SVR(), param_grid, refit=True, verbose=3, cv=TimeSeriesSplit(n_splits=5))
    grid.fit(X_train, y_train)
    return grid.best_estimator_


# Entrenamiento del Modelo LSTM
def train_lstm(X_train, y_train, input_shape):
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(1))

    model.compile(optimizer='adam', loss='mean_squared_error')
    model.fit(X_train, y_train, epochs=500, batch_size=32, validation_split=0.2)
    return model


# Preparar datos para LSTM
X_train_lstm = X_train.values.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = X_test.values.reshape((X_test.shape[0], 1, X_test.shape[1]))


# Optimizar y Entrenar Modelos
svm_model = optimize_svm(X_train, y_train)
lstm_model = train_lstm(X_train_lstm, y_train, (1, X_train.shape[1]))
```

```
5/5 [==============================] - 0s 85ms/step - loss: 0.0010 - val_loss: 5.0781e-05
Epoch 494/500
5/5 [==============================] - 0s 86ms/step - loss: 0.0010 - val_loss: 5.5009e-05
Epoch 495/500
5/5 [==============================] - 0s 96ms/step - loss: 8.3429e-04 - val_loss: 1.2871e-04
Epoch 496/500
5/5 [==============================] - 0s 69ms/step - loss: 0.0012 - val_loss: 7.2246e-05
Epoch 497/500
5/5 [==============================] - 0s 39ms/step - loss: 0.0011 - val_loss: 8.1324e-05
Epoch 498/500
5/5 [==============================] - 0s 54ms/step - loss: 0.0011 - val_loss: 4.2684e-05
Epoch 499/500
5/5 [==============================] - 0s 63ms/step - loss: 0.0010 - val_loss: 3.7981e-05
Epoch 500/500
5/5 [==============================] - 0s 50ms/step - loss: 9.9876e-04 - val_loss: 4.2124e-05
```

```
# Predicciones
svm_predictions = pd.Series(svm_model.predict(X_test), index=X_test.index)
lstm_predictions = pd.Series(lstm_model.predict(X_test_lstm).flatten(), index=X_test.index)
combined_predictions = pd.Series(np.median([svm_predictions, lstm_predictions], axis=0), index=X_test.index)
```

```
2/2 [==============================] - 2s 7ms/step
```

```
# Métricas de Validación
mape_svm = mean_absolute_percentage_error(y_test, svm_predictions)
mape_lstm = mean_absolute_percentage_error(y_test, lstm_predictions)
mape_combined = mean_absolute_percentage_error(y_test, combined_predictions)


print(f'MAPE SVM: {mape_svm}')
print(f'MAPE LSTM: {mape_lstm}')
print(f'MAPE Combined: {mape_combined}')
```

```
MAPE SVM: 25332400566781.773
MAPE LSTM: 16568517959841.182
MAPE Combined: 20950459263311.477
```

```
rmse_svm = np.sqrt(mean_squared_error(y_test, svm_predictions))
rmse_lstm = np.sqrt(mean_squared_error(y_test, lstm_predictions))
rmse_combined = np.sqrt(mean_squared_error(y_test, combined_predictions))


print(f'RMSE SVM: {rmse_svm}')
print(f'RMSE LSTM: {rmse_lstm}')
print(f'RMSE Combined: {rmse_combined}')
```

```
RMSE SVM: 0.09621961941190023
RMSE LSTM: 0.04793653618362173
RMSE Combined: 0.0712167931799812
```

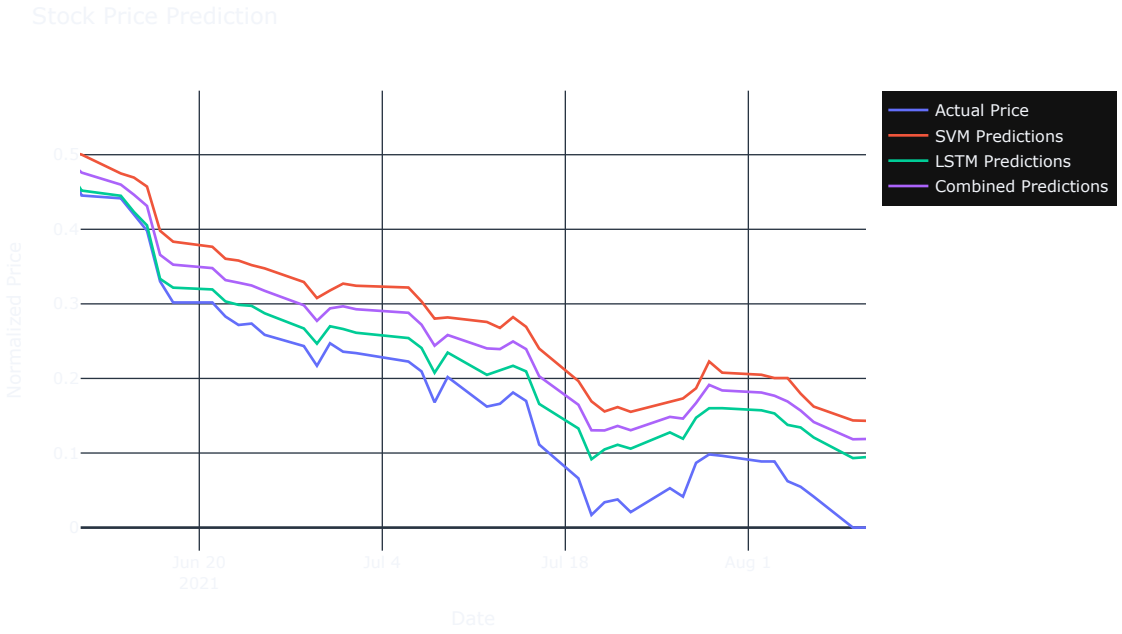
```
# Visualización con Plotly
def plot_forecast(dates_test, y_test, svm_predictions, lstm_predictions, combined_predictions):
    fig = go.Figure()

    fig.add_trace(go.Scatter(x=dates_test, y=y_test, mode='lines', name='Actual Price'))
    fig.add_trace(go.Scatter(x=dates_test, y=svm_predictions, mode='lines', name='SVM Predictions'))
    fig.add_trace(go.Scatter(x=dates_test, y=lstm_predictions, mode='lines', name='LSTM Predictions'))
    fig.add_trace(go.Scatter(x=dates_test, y=combined_predictions, mode='lines', name='Combined Predictions'))

    fig.update_layout(title='Stock Price Prediction',
                      xaxis_title='Date',
                      yaxis_title='Normalized Price',
                      template='plotly_dark')

    fig.show()


plot_forecast(dates_test, y_test, svm_predictions, lstm_predictions, combined_predictions)
```


Stock Price Prediction
Actual Price
SVM Predictions
LSTM Predictions
Combined Predictions
Normalized Price
Jun 20
2021
Jul 4
Jul 18
Aug 1
Date