prediction_improved

November 25, 2024

1. Importação das Bibliotecas Necessárias

Importamos as bibliotecas essenciais para manipulação de dados, visualização, pré-processamento, construção e treinamento do modelo, e tratamento de feriados.

[14]:
```
# Importação de bibliotecas essenciais
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

# Bibliotecas para pré-processamento
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score

# Bibliotecas para construção e treinamento do modelo
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.losses import Huber
from tensorflow.keras.optimizers import Adam

# Biblioteca para lidar com feriados
import holidays

# Ignorar avisos para uma saída mais limpa
import warnings
warnings.filterwarnings('ignore')
```

2. Carregamento e Inspeção dos Dados

Carregamos o arquivo CSV contendo os dados de vendas e realizamos uma inspeção inicial para entender a estrutura dos dados.

[15]:
```
# Definir o caminho do arquivo CSV
csv_path = '../data/dados_transacao_26173.csv'  # Atualize o caminho conforme␣
 ↪necessário
```

```python
# Carregar o CSV
df = pd.read_csv(csv_path)

# Converter a coluna 'Data' para datetime
df['Data'] = pd.to_datetime(df['Data'], format='%Y-%m-%d')

# Exibir as primeiras linhas do dataset
print("Primeiras linhas do dataset:")
display(df.head())

# Informações gerais sobre o dataset
print("\nInformações gerais do dataset:")
print(df.info())

# Estatísticas descritivas
print("\nEstatísticas descritivas:")
display(df.describe())
```

```
Primeiras linhas do dataset:

   CodigoVenda       Data             DataHora Status  VendaCancelada  \
0      2263035 2019-01-02  2019-01-02 08:36:25      f               0
1      2263063 2019-01-02  2019-01-02 09:01:27      f               0
2      2263067 2019-01-02  2019-01-02 09:06:07      f               0
3      2263151 2019-01-02  2019-01-02 09:55:52      f               0
4      2263159 2019-01-02  2019-01-02 10:01:20      f               0

   TotalPedido  DescontoGeral  AcrescimoGeral  TotalCusto  CodigoProduto  …  \
0        58.08            0.0             0.0       27.30          26173  …
1        40.05            0.0             0.0       29.45          26173  …
2        34.75            0.0             0.0       26.94          26173  …
3       210.10            0.0             0.0      146.63          26173  …
4        96.72            0.0             0.0       60.43          26173  …

   ValorCustoGerencial  CodigoFornecedor  CodigoKitPrincipal  \
0                  1.5                 0                   0
1                  1.5                 0                   0
2                  1.5                 0                   0
3                  1.5                 0                   0
4                  1.5                 0                   0

   ValorKitPrincipal  EmPromocao  DiaDaSemana  Mes  Dia  Feriado  \
0                  0           0            2    1    2        0
1                  0           0            2    1    2        0
2                  0           0            2    1    2        0
3                  0           0            2    1    2        0
4                  0           0            2    1    2        0
```

```
   VesperaDeFeriado
0                 0
1                 0
2                 0
3                 0
4                 0

[5 rows x 33 columns]


Informações gerais do dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 117192 entries, 0 to 117191
Data columns (total 33 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   CodigoVenda          117192 non-null  int64
 1   Data                 117192 non-null  datetime64[ns]
 2   DataHora             117192 non-null  object
 3   Status               117192 non-null  object
 4   VendaCancelada       117192 non-null  int64
 5   TotalPedido          117190 non-null  float64
 6   DescontoGeral        117192 non-null  float64
 7   AcrescimoGeral       117192 non-null  float64
 8   TotalCusto           117192 non-null  float64
 9   CodigoProduto        117192 non-null  int64
 10  Quantidade           117192 non-null  float64
 11  ValorUnitario        117192 non-null  float64
 12  ValorTotal           117192 non-null  float64
 13  Desconto             117192 non-null  float64
 14  Acrescimo            117192 non-null  float64
 15  ItemCancelado        117192 non-null  int64
 16  QuantDevolvida       117192 non-null  int64
 17  PrecoemPromocao      117192 non-null  int64
 18  CodigoSecao          117192 non-null  int64
 19  CodigoGrupo          117192 non-null  int64
 20  CodigoSubGrupo       117192 non-null  int64
 21  CodigoFabricante     117192 non-null  int64
 22  ValorCusto           117192 non-null  float64
 23  ValorCustoGerencial  117192 non-null  float64
 24  CodigoFornecedor     117192 non-null  int64
 25  CodigoKitPrincipal   117192 non-null  int64
 26  ValorKitPrincipal    117192 non-null  int64
 27  EmPromocao           117192 non-null  int64
 28  DiaDaSemana          117192 non-null  int64
 29  Mes                  117192 non-null  int64
 30  Dia                  117192 non-null  int64
 31  Feriado              117192 non-null  int64
```

```
 32  VesperaDeFeriado     117192 non-null  int64
dtypes: datetime64[ns](1), float64(11), int64(19), object(2)
memory usage: 29.5+ MB
None

Estatísticas descritivas:

        CodigoVenda                           Data  VendaCancelada  \
count  1.171920e+05                         117192   117192.000000
mean   3.542673e+06  2021-11-03 02:58:09.412246272        0.000734
min    2.263035e+06            2019-01-02 00:00:00        0.000000
25%    3.020720e+06            2020-09-09 18:00:00        0.000000
50%    3.522943e+06            2021-11-06 00:00:00        0.000000
75%    4.128652e+06            2023-02-17 00:00:00        0.000000
max    4.890548e+06            2024-08-27 00:00:00        1.000000
std    7.091584e+05                            NaN        0.027080

         TotalPedido  DescontoGeral  AcrescimoGeral     TotalCusto  \
count  117190.000000       117192.0        117192.0  117192.000000
mean      220.880381            0.0             0.0     163.759094
min      -249.980000            0.0             0.0    -181.330000
25%        56.462500            0.0             0.0      40.567500
50%       133.545000            0.0             0.0      95.645000
75%       289.540000            0.0             0.0     208.232500
max      3660.680000            0.0             0.0   35127.070000
std       255.111260            0.0             0.0     297.898729

       CodigoProduto     Quantidade  ValorUnitario  …  ValorCustoGerencial  \
count       117192.0  117192.000000  117192.000000  …        117192.000000
mean         26173.0       1.410600       2.249984  …             1.594766
min          26173.0     -90.000000      -3.550000  …             0.600000
25%          26173.0       1.000000       1.950000  …             1.500000
50%          26173.0       1.000000       2.110000  …             1.500000
75%          26173.0       1.000000       2.790000  …             2.000000
max          26173.0     364.000000      19.150000  …             3.000000
std              0.0       2.207204       0.689172  …             0.497781

       CodigoFornecedor  CodigoKitPrincipal  ValorKitPrincipal     EmPromocao  \
count     117192.000000            117192.0           117192.0  117192.000000
mean          14.557222                 0.0                0.0       0.070696
min            0.000000                 0.0                0.0       0.000000
25%            0.000000                 0.0                0.0       0.000000
50%            0.000000                 0.0                0.0       0.000000
75%            0.000000                 0.0                0.0       0.000000
max        77545.000000                 0.0                0.0       1.000000
std         1062.373533                 0.0                0.0       0.256317

         DiaDaSemana            Mes            Dia        Feriado  \
```

```
count  117192.000000  117192.000000  117192.000000  117192.000000
mean        3.355971       6.196182      15.685260       0.007603
min         0.000000       1.000000       1.000000       0.000000
25%         2.000000       3.000000       8.000000       0.000000
50%         4.000000       6.000000      16.000000       0.000000
75%         5.000000       9.000000      24.000000       0.000000
max         6.000000      12.000000      31.000000       1.000000
std         1.915065       3.609532       8.935082       0.086863

       VesperaDeFeriado
count     117192.000000
mean           0.054543
min            0.000000
25%            0.000000
50%            0.000000
75%            0.000000
max            1.000000
std            0.227087

[8 rows x 31 columns]
```

3. Pré-processamento dos Dados

Agregamos os dados por dia, criamos um range completo de datas para garantir que não haja datas faltantes e lidamos com valores ausentes. Também adicionamos lags das vendas anteriores para enriquecer o conjunto de features.

[16]:
```python
# Agregar os dados por dia
daily_df = df.groupby('Data').agg({
    'ValorUnitario': 'mean',  # Média do valor unitário por dia
    'Quantidade': 'sum'       # Quantidade total vendida por dia
}).reset_index()

# Exibir as primeiras linhas do dataframe agregado
print("\nPrimeiras linhas do dataframe agregado por dia:")
display(daily_df.head())

# Criar um range completo de datas
all_dates = pd.date_range(start=daily_df['Data'].min(), end=daily_df['Data'].
 ↪max(), freq='D')
daily_df = daily_df.set_index('Data').reindex(all_dates).reset_index()
daily_df.rename(columns={'index': 'Data'}, inplace=True)

# Preencher valores ausentes
daily_df['Quantidade'].fillna(0, inplace=True)
daily_df['ValorUnitario'].interpolate(method='linear', inplace=True)
daily_df['ValorUnitario'].fillna(method='bfill', inplace=True)
```

```
# Confirmar a ausência de valores faltantes
print("\nValores faltantes após tratamento:")
print(daily_df.isnull().sum())

# Adicionar lags das vendas anteriores
for lag in range(1, 8):
    daily_df[f'Quantidade_Lag{lag}'] = daily_df['Quantidade'].shift(lag)
    daily_df[f'ValorUnitario_Lag{lag}'] = daily_df['ValorUnitario'].shift(lag)

# Adicionar lags adicionais até 60 dias
for lag in range(8, 60):  # Continuar do lag 8 ao 60
    daily_df[f'Quantidade_Lag{lag}'] = daily_df['Quantidade'].shift(lag)
    daily_df[f'ValorUnitario_Lag{lag}'] = daily_df['ValorUnitario'].shift(lag)

# Preencher valores ausentes resultantes dos lags com zeros
daily_df.fillna(0, inplace=True)

daily_df['Quantidade'] = np.log1p(daily_df['Quantidade'])  # log1p é log(x+1)␣
↪para lidar com zeros


# Exibir as primeiras linhas com os lags
print("\nDataframe com lags adicionados:")
display(daily_df.head())
```

```
Primeiras linhas do dataframe agregado por dia:

        Data  ValorUnitario  Quantidade
0 2019-01-02       1.978571        80.0
1 2019-01-03       1.978333        90.0
2 2019-01-04       1.981918       136.0
3 2019-01-05       1.981000       144.0
4 2019-01-06       1.982542        74.0


Valores faltantes após tratamento:
Data             0
ValorUnitario    0
Quantidade       0
dtype: int64

Dataframe com lags adicionados:

        Data  ValorUnitario  Quantidade  Quantidade_Lag1  ValorUnitario_Lag1  \
0 2019-01-02       1.978571    4.394449              0.0            0.000000
1 2019-01-03       1.978333    4.510860             80.0            1.978571
2 2019-01-04       1.981918    4.919981             90.0            1.978333
3 2019-01-05       1.981000    4.976734            136.0            1.981918
```

```
4 2019-01-06       1.982542    4.317488           144.0            1.981000

   Quantidade_Lag2  ValorUnitario_Lag2  Quantidade_Lag3  ValorUnitario_Lag3  \
0              0.0            0.000000              0.0            0.000000
1              0.0            0.000000              0.0            0.000000
2             80.0            1.978571              0.0            0.000000
3             90.0            1.978333             80.0            1.978571
4            136.0            1.981918             90.0            1.978333

   Quantidade_Lag4  …  Quantidade_Lag55  ValorUnitario_Lag55  \
0              0.0  …               0.0                  0.0
1              0.0  …               0.0                  0.0
2              0.0  …               0.0                  0.0
3              0.0  …               0.0                  0.0
4             80.0  …               0.0                  0.0

   Quantidade_Lag56  ValorUnitario_Lag56  Quantidade_Lag57  \
0               0.0                  0.0               0.0
1               0.0                  0.0               0.0
2               0.0                  0.0               0.0
3               0.0                  0.0               0.0
4               0.0                  0.0               0.0

   ValorUnitario_Lag57  Quantidade_Lag58  ValorUnitario_Lag58  \
0                  0.0               0.0                  0.0
1                  0.0               0.0                  0.0
2                  0.0               0.0                  0.0
3                  0.0               0.0                  0.0
4                  0.0               0.0                  0.0

   Quantidade_Lag59  ValorUnitario_Lag59
0               0.0                  0.0
1               0.0                  0.0
2               0.0                  0.0
3               0.0                  0.0
4               0.0                  0.0

[5 rows x 121 columns]
```

4. Engenharia de Features

Adicionamos features derivadas da data que podem ajudar na previsão, como dia, mês, ano, dia da semana e feriados.

[17]:
```python
# Adicionar features derivadas da data
daily_df['Dia'] = daily_df['Data'].dt.day
daily_df['Mes'] = daily_df['Data'].dt.month
daily_df['Ano'] = daily_df['Data'].dt.year
```

```python
daily_df['DiaDaSemana'] = daily_df['Data'].dt.dayofweek
daily_df['Quantidade_Rolling_Mean'] = daily_df['Quantidade'].rolling(window=7,␣
 ↪min_periods=1).mean()
daily_df['ValorUnitario_Rolling_Mean'] = daily_df['ValorUnitario'].
 ↪rolling(window=7, min_periods=1).mean()
daily_df['SemanaDoAno'] = daily_df['Data'].dt.isocalendar().week

# Inicializar feriados do Brasil
br_holidays = holidays.Brazil()

# Adicionar coluna de feriado: 1 se feriado, 0 caso contrário
daily_df['Feriado'] = daily_df['Data'].isin(br_holidays).astype(int)

# Exibir as primeiras linhas com as novas features
print("\nDataframe com features adicionais:")
display(daily_df.head())
```

```
Dataframe com features adicionais:
        Data  ValorUnitario  Quantidade  Quantidade_Lag1  ValorUnitario_Lag1  \
0 2019-01-02       1.978571    4.394449              0.0            0.000000
1 2019-01-03       1.978333    4.510860             80.0            1.978571
2 2019-01-04       1.981918    4.919981             90.0            1.978333
3 2019-01-05       1.981000    4.976734            136.0            1.981918
4 2019-01-06       1.982542    4.317488            144.0            1.981000

   Quantidade_Lag2  ValorUnitario_Lag2  Quantidade_Lag3  ValorUnitario_Lag3  \
0              0.0            0.000000              0.0            0.000000
1              0.0            0.000000              0.0            0.000000
2             80.0            1.978571              0.0            0.000000
3             90.0            1.978333             80.0            1.978571
4            136.0            1.981918             90.0            1.978333

   Quantidade_Lag4  …  Quantidade_Lag59  ValorUnitario_Lag59  Dia  Mes  \
0              0.0  …               0.0                  0.0    2    1
1              0.0  …               0.0                  0.0    3    1
2              0.0  …               0.0                  0.0    4    1
3              0.0  …               0.0                  0.0    5    1
4             80.0  …               0.0                  0.0    6    1

    Ano  DiaDaSemana  Quantidade_Rolling_Mean  ValorUnitario_Rolling_Mean  \
0  2019            2                 4.394449                    1.978571
1  2019            3                 4.452654                    1.978452
2  2019            4                 4.608430                    1.979608
3  2019            5                 4.700506                    1.979956
4  2019            6                 4.623902                    1.980473
```

```
   SemanaDoAno  Feriado
0            1        0
1            1        0
2            1        0
3            1        0
4            1        0

[5 rows x 129 columns]
```

5. Divisão dos Dados

Dividimos os dados em conjuntos de treino inicial, treino continuado/validação e teste conforme a estratégia descrita.

[18]:
```
# Definir as datas de corte
train_end = pd.to_datetime('2022-12-31')
continue_train_end = pd.to_datetime('2023-12-31')
test_start = pd.to_datetime('2024-01-01')
test_end = pd.to_datetime('2024-03-30')

# Separar os dados
train_df = daily_df[daily_df['Data'] <= train_end].reset_index(drop=True)
continue_train_df = daily_df[(daily_df['Data'] > train_end) & (daily_df['Data']␣
 ↪<= continue_train_end)].reset_index(drop=True)
test_df = daily_df[(daily_df['Data'] >= test_start) & (daily_df['Data'] <=␣
 ↪test_end)].reset_index(drop=True)

# Exibir o número de registros em cada conjunto
print(f"\nTreino Inicial: {train_df.shape[0]} registros")
print(f"Treino Continuado: {continue_train_df.shape[0]} registros")
print(f"Teste: {test_df.shape[0]} registros")
```

```
Treino Inicial: 1460 registros
Treino Continuado: 365 registros
Teste: 90 registros
```

6. Escalonamento dos Dados

Escalonamos os dados para melhorar o desempenho do modelo LSTM. Utilizamos o MinMaxScaler para normalizar as features e os targets entre 0 e 1.

[19]:
```
# Definir features e targets
feature_cols = ['Dia', 'Mes', 'Ano', 'DiaDaSemana', 'Feriado', 'SemanaDoAno'] +␣
 ↪\
               [f'Quantidade_Lag{lag}' for lag in range(1, 15)] + \
               [f'ValorUnitario_Lag{lag}' for lag in range(1, 15)]

target_cols = ['ValorUnitario', 'Quantidade']
```

```python
# Inicializar scalers
feature_scaler = MinMaxScaler()
target_scaler = MinMaxScaler()

# Ajustar scalers nos dados de treino inicial
feature_scaler.fit(train_df[feature_cols])
target_scaler.fit(train_df[target_cols])

# Escalonar dados de treino inicial
train_features_scaled = feature_scaler.transform(train_df[feature_cols])
train_targets_scaled = target_scaler.transform(train_df[target_cols])

# Escalonar dados de treino continuado
continue_train_features_scaled = feature_scaler.
 ↪transform(continue_train_df[feature_cols])
continue_train_targets_scaled = target_scaler.
 ↪transform(continue_train_df[target_cols])

# Escalonar dados de teste
test_features_scaled = feature_scaler.transform(test_df[feature_cols])
test_targets_scaled = target_scaler.transform(test_df[target_cols])

# Exibir uma amostra dos dados escalonados de treino
print("\nAmostra dos dados escalonados de treino:")
display(pd.DataFrame(train_features_scaled, columns=feature_cols).head())
```

```
Amostra dos dados escalonados de treino:
        Dia  Mes  Ano  DiaDaSemana  Feriado  SemanaDoAno  Quantidade_Lag1  \
0  0.033333  0.0  0.0     0.333333      0.0          0.0         0.000000
1  0.066667  0.0  0.0     0.500000      0.0          0.0         0.163265
2  0.100000  0.0  0.0     0.666667      0.0          0.0         0.183673
3  0.133333  0.0  0.0     0.833333      0.0          0.0         0.277551
4  0.166667  0.0  0.0     1.000000      0.0          0.0         0.293878

   Quantidade_Lag2  Quantidade_Lag3  Quantidade_Lag4  …  ValorUnitario_Lag5  \
0         0.000000         0.000000         0.000000  …                 0.0
1         0.000000         0.000000         0.000000  …                 0.0
2         0.163265         0.000000         0.000000  …                 0.0
3         0.183673         0.163265         0.000000  …                 0.0
4         0.277551         0.183673         0.163265  …                 0.0

   ValorUnitario_Lag6  ValorUnitario_Lag7  ValorUnitario_Lag8  \
0                 0.0                 0.0                 0.0
1                 0.0                 0.0                 0.0
2                 0.0                 0.0                 0.0
3                 0.0                 0.0                 0.0
```

```
4                 0.0                  0.0                  0.0

   ValorUnitario_Lag9  ValorUnitario_Lag10  ValorUnitario_Lag11  \
0                 0.0                  0.0                  0.0
1                 0.0                  0.0                  0.0
2                 0.0                  0.0                  0.0
3                 0.0                  0.0                  0.0
4                 0.0                  0.0                  0.0

   ValorUnitario_Lag12  ValorUnitario_Lag13  ValorUnitario_Lag14
0                  0.0                  0.0                  0.0
1                  0.0                  0.0                  0.0
2                  0.0                  0.0                  0.0
3                  0.0                  0.0                  0.0
4                  0.0                  0.0                  0.0

[5 rows x 34 columns]
```

7. Criação de Sequências para o LSTM

Criamos sequências temporais dos dados para treinar o modelo LSTM. Cada sequência de entrada consiste em um número fixo de dias anteriores (por exemplo, 30 dias) e a saída é o valor do dia seguinte.

[20]:
```
# Função para criar sequências
def create_sequences(features, targets, seq_length):
    X = []
    y = []
    for i in range(seq_length, len(features)):
        X.append(features[i-seq_length:i])
        y.append(targets[i])
    return np.array(X), np.array(y)

SEQ_LENGTH = 30  # Número de dias usados para prever

# Criar sequências de treino
X_train, y_train = create_sequences(train_features_scaled,␣
 ↪train_targets_scaled, SEQ_LENGTH)
print(f"\nForma de X_train: {X_train.shape}, y_train: {y_train.shape}")

# Criar sequências de treino continuado
X_continue_train, y_continue_train =␣
 ↪create_sequences(continue_train_features_scaled,␣
 ↪continue_train_targets_scaled, SEQ_LENGTH)
print(f"Forma de X_continue_train: {X_continue_train.shape}, y_continue_train:␣
 ↪{y_continue_train.shape}")

# Criar sequências de teste
```

```
X_test, y_test = create_sequences(test_features_scaled, test_targets_scaled,␣
 ↪SEQ_LENGTH)
print(f"Forma de X_test: {X_test.shape}, y_test: {y_test.shape}")
```

```
Forma de X_train: (1430, 30, 34), y_train: (1430, 2)
Forma de X_continue_train: (335, 30, 34), y_continue_train: (335, 2)
Forma de X_test: (60, 30, 34), y_test: (60, 2)
```

8. Construção do Modelo LSTM

Definimos a arquitetura do modelo LSTM com camadas adicionais e dropout para melhorar a capacidade de generalização.

[21]:
```
# Construção do modelo LSTM avançado com mais camadas

# Modelo ajustado com mais camadas e regularização
model = Sequential()

# Primeira camada LSTM
model.add(LSTM(512, return_sequences=True, input_shape=(SEQ_LENGTH,␣
 ↪len(feature_cols)), dropout=0.3, recurrent_dropout=0.3))
# Segunda camada LSTM
model.add(LSTM(256, return_sequences=True, dropout=0.3, recurrent_dropout=0.3))
# Terceira camada LSTM
model.add(LSTM(128, return_sequences=False, dropout=0.3, recurrent_dropout=0.3))

# Camadas densas para processar as saídas da LSTM
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))

# Camada de saída
model.add(Dense(len(target_cols)))

# Compilação do modelo com taxa de aprendizado reduzida
optimizer = Adam(learning_rate=0.0005)  # Taxa de aprendizado inicial ajustada
model.compile(optimizer=optimizer, loss='mse')

# Exibir o resumo do modelo
print("\nResumo do modelo LSTM avançado:")
model.summary()
```

```
Resumo do modelo LSTM avançado:
```

Model: "sequential_1"

```
 Layer (type)                      Output Shape                  Param #

 lstm_3 (LSTM)                     (None, 30, 512)             1,120,256

 lstm_4 (LSTM)                     (None, 30, 256)               787,456

 lstm_5 (LSTM)                     (None, 128)                   197,120

 dense_3 (Dense)                   (None, 64)                      8,256

 dropout_1 (Dropout)               (None, 64)                          0

 dense_4 (Dense)                   (None, 32)                      2,080

 dense_5 (Dense)                   (None, 2)                          66


Total params: 2,115,234 (8.07 MB)

Trainable params: 2,115,234 (8.07 MB)

Non-trainable params: 0 (0.00 B)
```

9. Treinamento do Modelo

Treinamos o modelo inicialmente com os dados de treino (até 31/12/2022) e depois continuamos o treinamento com os dados de 2023.

[22]:
```
# Definir callbacks para monitorar o treinamento
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=60,  # Número de épocas sem melhoria
    restore_best_weights=True
)

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=1e-6
)

# Treinar o modelo com os dados de treino inicial
history = model.fit(
    X_train, y_train,
```

```
    epochs=200,  # Mantendo 200 épocas para garantir tempo suficiente de␣
 ↪treinamento
    batch_size=64,  # Lote ajustado para balancear memória e desempenho
    validation_split=0.1,  # Separar 10% dos dados de treino para validação
    callbacks=[early_stop, reduce_lr],
    verbose=1  # Mostrar logs detalhados
)

# Plotar a perda de treino e validação
plt.figure(figsize=(12,6))
plt.plot(history.history['loss'], label='Treino')
plt.plot(history.history['val_loss'], label='Validação')
plt.title('Perda do Modelo durante o Treinamento Inicial')
plt.xlabel('Épocas')
plt.ylabel('MSE Loss')
plt.legend()
plt.show()
```

```
Epoch 1/200
21/21                 10s 131ms/step -
loss: 1.4042 - val_loss: 1.1411 - learning_rate: 5.0000e-04
Epoch 2/200
21/21                 2s 81ms/step -
loss: 1.1335 - val_loss: 0.9961 - learning_rate: 5.0000e-04
Epoch 3/200
21/21                 2s 85ms/step -
loss: 0.9855 - val_loss: 0.8652 - learning_rate: 5.0000e-04
Epoch 4/200
21/21                 2s 85ms/step -
loss: 0.8563 - val_loss: 0.7479 - learning_rate: 5.0000e-04
Epoch 5/200
21/21                 2s 79ms/step -
loss: 0.7407 - val_loss: 0.6465 - learning_rate: 5.0000e-04
Epoch 6/200
21/21                 2s 76ms/step -
loss: 0.6397 - val_loss: 0.5561 - learning_rate: 5.0000e-04
Epoch 7/200
21/21                 2s 75ms/step -
loss: 0.5531 - val_loss: 0.4829 - learning_rate: 5.0000e-04
Epoch 8/200
21/21                 2s 77ms/step -
loss: 0.4779 - val_loss: 0.4139 - learning_rate: 5.0000e-04
Epoch 9/200
21/21                 2s 76ms/step -
loss: 0.4117 - val_loss: 0.3570 - learning_rate: 5.0000e-04
Epoch 10/200
21/21                 2s 82ms/step -
loss: 0.3570 - val_loss: 0.3077 - learning_rate: 5.0000e-04
```

```
Epoch 11/200
21/21            2s 82ms/step -
loss: 0.3062 - val_loss: 0.2644 - learning_rate: 5.0000e-04
Epoch 12/200
21/21            2s 79ms/step -
loss: 0.2671 - val_loss: 0.2271 - learning_rate: 5.0000e-04
Epoch 13/200
21/21            2s 81ms/step -
loss: 0.2316 - val_loss: 0.1997 - learning_rate: 5.0000e-04
Epoch 14/200
21/21            2s 83ms/step -
loss: 0.2011 - val_loss: 0.1703 - learning_rate: 5.0000e-04
Epoch 15/200
21/21            2s 80ms/step -
loss: 0.1733 - val_loss: 0.1484 - learning_rate: 5.0000e-04
Epoch 16/200
21/21            2s 78ms/step -
loss: 0.1499 - val_loss: 0.1296 - learning_rate: 5.0000e-04
Epoch 17/200
21/21            2s 80ms/step -
loss: 0.1329 - val_loss: 0.1126 - learning_rate: 5.0000e-04
Epoch 18/200
21/21            2s 76ms/step -
loss: 0.1165 - val_loss: 0.0977 - learning_rate: 5.0000e-04
Epoch 19/200
21/21            2s 73ms/step -
loss: 0.1031 - val_loss: 0.0872 - learning_rate: 5.0000e-04
Epoch 20/200
21/21            2s 79ms/step -
loss: 0.0916 - val_loss: 0.0746 - learning_rate: 5.0000e-04
Epoch 21/200
21/21            2s 77ms/step -
loss: 0.0823 - val_loss: 0.0657 - learning_rate: 5.0000e-04
Epoch 22/200
21/21            2s 78ms/step -
loss: 0.0710 - val_loss: 0.0609 - learning_rate: 5.0000e-04
Epoch 23/200
21/21            2s 79ms/step -
loss: 0.0645 - val_loss: 0.0551 - learning_rate: 5.0000e-04
Epoch 24/200
21/21            2s 76ms/step -
loss: 0.0600 - val_loss: 0.0475 - learning_rate: 5.0000e-04
Epoch 25/200
21/21            2s 77ms/step -
loss: 0.0547 - val_loss: 0.0460 - learning_rate: 5.0000e-04
Epoch 26/200
21/21            2s 80ms/step -
loss: 0.0509 - val_loss: 0.0391 - learning_rate: 5.0000e-04
```

```
Epoch 27/200
21/21            2s 77ms/step -
loss: 0.0462 - val_loss: 0.0381 - learning_rate: 5.0000e-04
Epoch 28/200
21/21            2s 76ms/step -
loss: 0.0437 - val_loss: 0.0374 - learning_rate: 5.0000e-04
Epoch 29/200
21/21            2s 79ms/step -
loss: 0.0417 - val_loss: 0.0358 - learning_rate: 5.0000e-04
Epoch 30/200
21/21            2s 75ms/step -
loss: 0.0405 - val_loss: 0.0292 - learning_rate: 5.0000e-04
Epoch 31/200
21/21            2s 77ms/step -
loss: 0.0352 - val_loss: 0.0279 - learning_rate: 5.0000e-04
Epoch 32/200
21/21            2s 80ms/step -
loss: 0.0362 - val_loss: 0.0259 - learning_rate: 5.0000e-04
Epoch 33/200
21/21            2s 76ms/step -
loss: 0.0326 - val_loss: 0.0292 - learning_rate: 5.0000e-04
Epoch 34/200
21/21            2s 79ms/step -
loss: 0.0304 - val_loss: 0.0227 - learning_rate: 5.0000e-04
Epoch 35/200
21/21            2s 76ms/step -
loss: 0.0303 - val_loss: 0.0228 - learning_rate: 5.0000e-04
Epoch 36/200
21/21            2s 76ms/step -
loss: 0.0281 - val_loss: 0.0214 - learning_rate: 5.0000e-04
Epoch 37/200
21/21            2s 79ms/step -
loss: 0.0276 - val_loss: 0.0234 - learning_rate: 5.0000e-04
Epoch 38/200
21/21            2s 77ms/step -
loss: 0.0272 - val_loss: 0.0205 - learning_rate: 5.0000e-04
Epoch 39/200
21/21            2s 75ms/step -
loss: 0.0274 - val_loss: 0.0193 - learning_rate: 5.0000e-04
Epoch 40/200
21/21            2s 78ms/step -
loss: 0.0244 - val_loss: 0.0189 - learning_rate: 5.0000e-04
Epoch 41/200
21/21            2s 76ms/step -
loss: 0.0264 - val_loss: 0.0185 - learning_rate: 5.0000e-04
Epoch 42/200
21/21            2s 77ms/step -
loss: 0.0259 - val_loss: 0.0177 - learning_rate: 5.0000e-04
```

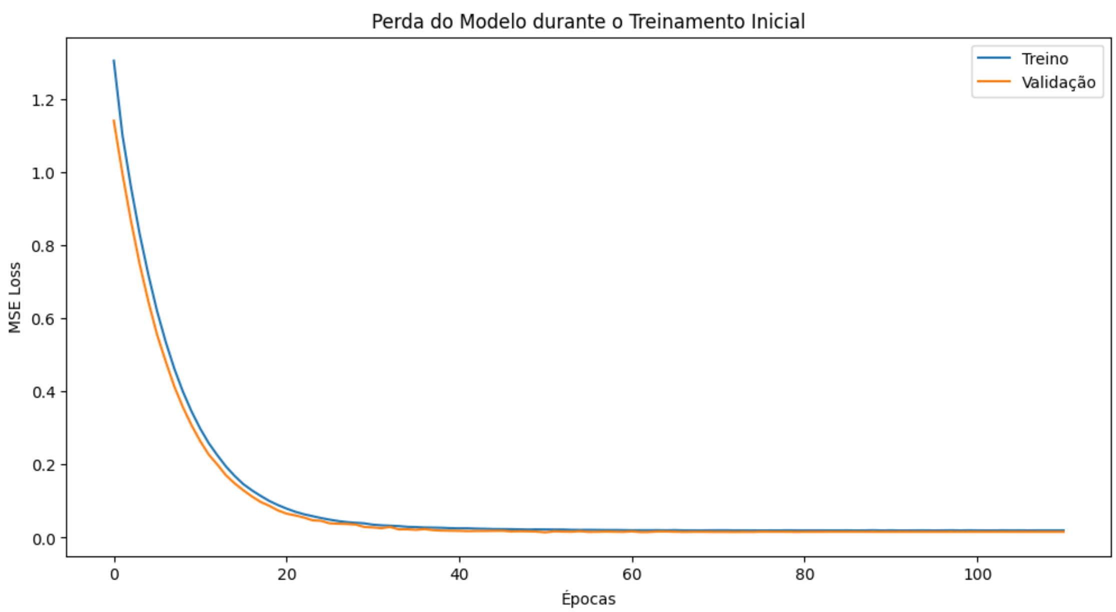
```
Epoch 43/200
21/21            2s 78ms/step -
loss: 0.0259 - val_loss: 0.0182 - learning_rate: 5.0000e-04
Epoch 44/200
21/21            2s 77ms/step -
loss: 0.0236 - val_loss: 0.0182 - learning_rate: 5.0000e-04
Epoch 45/200
21/21            2s 78ms/step -
loss: 0.0218 - val_loss: 0.0185 - learning_rate: 5.0000e-04
Epoch 46/200
21/21            2s 75ms/step -
loss: 0.0221 - val_loss: 0.0187 - learning_rate: 5.0000e-04
Epoch 47/200
21/21            2s 77ms/step -
loss: 0.0248 - val_loss: 0.0170 - learning_rate: 5.0000e-04
Epoch 48/200
21/21            2s 79ms/step -
loss: 0.0221 - val_loss: 0.0174 - learning_rate: 5.0000e-04
Epoch 49/200
21/21            2s 76ms/step -
loss: 0.0225 - val_loss: 0.0171 - learning_rate: 5.0000e-04
Epoch 50/200
21/21            2s 77ms/step -
loss: 0.0240 - val_loss: 0.0164 - learning_rate: 5.0000e-04
Epoch 51/200
21/21            2s 76ms/step -
loss: 0.0227 - val_loss: 0.0145 - learning_rate: 5.0000e-04
Epoch 52/200
21/21            2s 75ms/step -
loss: 0.0227 - val_loss: 0.0173 - learning_rate: 5.0000e-04
Epoch 53/200
21/21            2s 78ms/step -
loss: 0.0218 - val_loss: 0.0165 - learning_rate: 5.0000e-04
Epoch 54/200
21/21            2s 74ms/step -
loss: 0.0223 - val_loss: 0.0161 - learning_rate: 5.0000e-04
Epoch 55/200
21/21            2s 77ms/step -
loss: 0.0204 - val_loss: 0.0175 - learning_rate: 5.0000e-04
Epoch 56/200
21/21            2s 78ms/step -
loss: 0.0211 - val_loss: 0.0156 - learning_rate: 5.0000e-04
Epoch 57/200
21/21            2s 74ms/step -
loss: 0.0225 - val_loss: 0.0159 - learning_rate: 2.5000e-04
Epoch 58/200
21/21            2s 76ms/step -
loss: 0.0207 - val_loss: 0.0166 - learning_rate: 2.5000e-04
```

```
Epoch 59/200
21/21            2s 75ms/step -
loss: 0.0221 - val_loss: 0.0160 - learning_rate: 2.5000e-04
Epoch 60/200
21/21            2s 75ms/step -
loss: 0.0209 - val_loss: 0.0157 - learning_rate: 2.5000e-04
Epoch 61/200
21/21            2s 78ms/step -
loss: 0.0179 - val_loss: 0.0171 - learning_rate: 2.5000e-04
Epoch 62/200
21/21            2s 75ms/step -
loss: 0.0218 - val_loss: 0.0151 - learning_rate: 1.2500e-04
Epoch 63/200
21/21            2s 77ms/step -
loss: 0.0204 - val_loss: 0.0154 - learning_rate: 1.2500e-04
Epoch 64/200
21/21            2s 78ms/step -
loss: 0.0205 - val_loss: 0.0167 - learning_rate: 1.2500e-04
Epoch 65/200
21/21            2s 76ms/step -
loss: 0.0212 - val_loss: 0.0168 - learning_rate: 1.2500e-04
Epoch 66/200
21/21            2s 79ms/step -
loss: 0.0192 - val_loss: 0.0160 - learning_rate: 1.2500e-04
Epoch 67/200
21/21            2s 78ms/step -
loss: 0.0193 - val_loss: 0.0156 - learning_rate: 6.2500e-05
Epoch 68/200
21/21            2s 77ms/step -
loss: 0.0201 - val_loss: 0.0158 - learning_rate: 6.2500e-05
Epoch 69/200
21/21            2s 78ms/step -
loss: 0.0188 - val_loss: 0.0160 - learning_rate: 6.2500e-05
Epoch 70/200
21/21            2s 75ms/step -
loss: 0.0188 - val_loss: 0.0157 - learning_rate: 6.2500e-05
Epoch 71/200
21/21            2s 77ms/step -
loss: 0.0196 - val_loss: 0.0157 - learning_rate: 6.2500e-05
Epoch 72/200
21/21            2s 78ms/step -
loss: 0.0195 - val_loss: 0.0157 - learning_rate: 3.1250e-05
Epoch 73/200
21/21            2s 76ms/step -
loss: 0.0202 - val_loss: 0.0154 - learning_rate: 3.1250e-05
Epoch 74/200
21/21            2s 78ms/step -
loss: 0.0197 - val_loss: 0.0158 - learning_rate: 3.1250e-05
```

```
Epoch 75/200
21/21            3s 77ms/step -
loss: 0.0190 - val_loss: 0.0156 - learning_rate: 3.1250e-05
Epoch 76/200
21/21            2s 76ms/step -
loss: 0.0213 - val_loss: 0.0162 - learning_rate: 3.1250e-05
Epoch 77/200
21/21            2s 79ms/step -
loss: 0.0193 - val_loss: 0.0160 - learning_rate: 1.5625e-05
Epoch 78/200
21/21            2s 79ms/step -
loss: 0.0213 - val_loss: 0.0160 - learning_rate: 1.5625e-05
Epoch 79/200
21/21            2s 79ms/step -
loss: 0.0204 - val_loss: 0.0160 - learning_rate: 1.5625e-05
Epoch 80/200
21/21            2s 76ms/step -
loss: 0.0204 - val_loss: 0.0157 - learning_rate: 1.5625e-05
Epoch 81/200
21/21            2s 78ms/step -
loss: 0.0193 - val_loss: 0.0159 - learning_rate: 1.5625e-05
Epoch 82/200
21/21            2s 81ms/step -
loss: 0.0209 - val_loss: 0.0159 - learning_rate: 7.8125e-06
Epoch 83/200
21/21            2s 77ms/step -
loss: 0.0191 - val_loss: 0.0159 - learning_rate: 7.8125e-06
Epoch 84/200
21/21            2s 85ms/step -
loss: 0.0193 - val_loss: 0.0160 - learning_rate: 7.8125e-06
Epoch 85/200
21/21            2s 80ms/step -
loss: 0.0201 - val_loss: 0.0160 - learning_rate: 7.8125e-06
Epoch 86/200
21/21            2s 79ms/step -
loss: 0.0205 - val_loss: 0.0160 - learning_rate: 7.8125e-06
Epoch 87/200
21/21            2s 82ms/step -
loss: 0.0199 - val_loss: 0.0160 - learning_rate: 3.9063e-06
Epoch 88/200
21/21            2s 80ms/step -
loss: 0.0200 - val_loss: 0.0160 - learning_rate: 3.9063e-06
Epoch 89/200
21/21            2s 82ms/step -
loss: 0.0199 - val_loss: 0.0159 - learning_rate: 3.9063e-06
Epoch 90/200
21/21            2s 82ms/step -
loss: 0.0194 - val_loss: 0.0159 - learning_rate: 3.9063e-06
```

```
Epoch 91/200
21/21            2s 75ms/step -
loss: 0.0203 - val_loss: 0.0159 - learning_rate: 3.9063e-06
Epoch 92/200
21/21            2s 76ms/step -
loss: 0.0179 - val_loss: 0.0159 - learning_rate: 1.9531e-06
Epoch 93/200
21/21            2s 75ms/step -
loss: 0.0199 - val_loss: 0.0159 - learning_rate: 1.9531e-06
Epoch 94/200
21/21            2s 79ms/step -
loss: 0.0187 - val_loss: 0.0159 - learning_rate: 1.9531e-06
Epoch 95/200
21/21            2s 77ms/step -
loss: 0.0200 - val_loss: 0.0159 - learning_rate: 1.9531e-06
Epoch 96/200
21/21            2s 76ms/step -
loss: 0.0201 - val_loss: 0.0159 - learning_rate: 1.9531e-06
Epoch 97/200
21/21            2s 75ms/step -
loss: 0.0183 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 98/200
21/21            2s 75ms/step -
loss: 0.0203 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 99/200
21/21            2s 77ms/step -
loss: 0.0194 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 100/200
21/21            2s 76ms/step -
loss: 0.0209 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 101/200
21/21            2s 78ms/step -
loss: 0.0211 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 102/200
21/21            2s 78ms/step -
loss: 0.0190 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 103/200
21/21            2s 85ms/step -
loss: 0.0200 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 104/200
21/21            2s 84ms/step -
loss: 0.0188 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 105/200
21/21            2s 85ms/step -
loss: 0.0190 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 106/200
21/21            2s 85ms/step -
loss: 0.0203 - val_loss: 0.0159 - learning_rate: 1.0000e-06
```

```
Epoch 107/200
21/21          2s 83ms/step -
loss: 0.0185 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 108/200
21/21          2s 86ms/step -
loss: 0.0190 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 109/200
21/21          2s 82ms/step -
loss: 0.0202 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 110/200
21/21          2s 86ms/step -
loss: 0.0197 - val_loss: 0.0159 - learning_rate: 1.0000e-06
Epoch 111/200
21/21          2s 83ms/step -
loss: 0.0208 - val_loss: 0.0159 - learning_rate: 1.0000e-06
```



[23]:
```python
# Treinamento contínuo para ajustar novas tendências de 2023
history_continue = model.fit(
    X_continue_train, y_continue_train,
    epochs=50,  # Período de ajuste mais curto
    batch_size=64,  # Mantendo o mesmo batch size
    validation_split=0.1,  # Validação consistente com o treinamento inicial
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

# Plotar a perda de treino e validação durante a continuação do treinamento
```

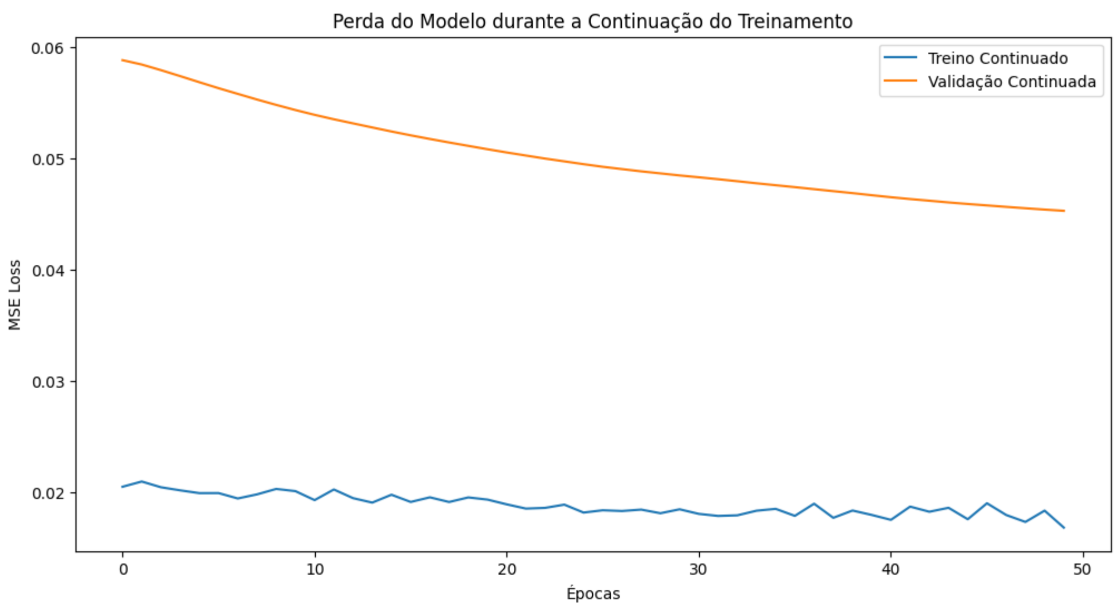
```python
plt.figure(figsize=(12,6))
plt.plot(history_continue.history['loss'], label='Treino Continuado')
plt.plot(history_continue.history['val_loss'], label='Validação Continuada')
plt.title('Perda do Modelo durante a Continuação do Treinamento')
plt.xlabel('Épocas')
plt.ylabel('MSE Loss')
plt.legend()
plt.show()
```

```
Epoch 1/50
5/5               1s 111ms/step - loss:
0.0201 - val_loss: 0.0589 - learning_rate: 1.0000e-06
Epoch 2/50
5/5               0s 87ms/step - loss:
0.0225 - val_loss: 0.0585 - learning_rate: 1.0000e-06
Epoch 3/50
5/5               0s 86ms/step - loss:
0.0215 - val_loss: 0.0580 - learning_rate: 1.0000e-06
Epoch 4/50
5/5               0s 89ms/step - loss:
0.0215 - val_loss: 0.0574 - learning_rate: 1.0000e-06
Epoch 5/50
5/5               0s 85ms/step - loss:
0.0211 - val_loss: 0.0569 - learning_rate: 1.0000e-06
Epoch 6/50
5/5               0s 82ms/step - loss:
0.0203 - val_loss: 0.0563 - learning_rate: 1.0000e-06
Epoch 7/50
5/5               0s 88ms/step - loss:
0.0199 - val_loss: 0.0558 - learning_rate: 1.0000e-06
Epoch 8/50
5/5               0s 84ms/step - loss:
0.0208 - val_loss: 0.0553 - learning_rate: 1.0000e-06
Epoch 9/50
5/5               0s 99ms/step - loss:
0.0206 - val_loss: 0.0548 - learning_rate: 1.0000e-06
Epoch 10/50
5/5               0s 94ms/step - loss:
0.0228 - val_loss: 0.0544 - learning_rate: 1.0000e-06
Epoch 11/50
5/5               1s 104ms/step - loss:
0.0191 - val_loss: 0.0539 - learning_rate: 1.0000e-06
Epoch 12/50
5/5               1s 103ms/step - loss:
0.0204 - val_loss: 0.0535 - learning_rate: 1.0000e-06
Epoch 13/50
5/5               1s 95ms/step - loss:
0.0193 - val_loss: 0.0532 - learning_rate: 1.0000e-06
```

```
Epoch 14/50
5/5            0s 88ms/step - loss:
0.0193 - val_loss: 0.0528 - learning_rate: 1.0000e-06
Epoch 15/50
5/5            0s 90ms/step - loss:
0.0194 - val_loss: 0.0524 - learning_rate: 1.0000e-06
Epoch 16/50
5/5            0s 91ms/step - loss:
0.0192 - val_loss: 0.0521 - learning_rate: 1.0000e-06
Epoch 17/50
5/5            0s 88ms/step - loss:
0.0200 - val_loss: 0.0518 - learning_rate: 1.0000e-06
Epoch 18/50
5/5            1s 101ms/step - loss:
0.0204 - val_loss: 0.0515 - learning_rate: 1.0000e-06
Epoch 19/50
5/5            0s 86ms/step - loss:
0.0217 - val_loss: 0.0512 - learning_rate: 1.0000e-06
Epoch 20/50
5/5            0s 88ms/step - loss:
0.0205 - val_loss: 0.0509 - learning_rate: 1.0000e-06
Epoch 21/50
5/5            0s 87ms/step - loss:
0.0195 - val_loss: 0.0506 - learning_rate: 1.0000e-06
Epoch 22/50
5/5            0s 90ms/step - loss:
0.0185 - val_loss: 0.0503 - learning_rate: 1.0000e-06
Epoch 23/50
5/5            0s 89ms/step - loss:
0.0187 - val_loss: 0.0500 - learning_rate: 1.0000e-06
Epoch 24/50
5/5            0s 88ms/step - loss:
0.0211 - val_loss: 0.0498 - learning_rate: 1.0000e-06
Epoch 25/50
5/5            0s 86ms/step - loss:
0.0187 - val_loss: 0.0495 - learning_rate: 1.0000e-06
Epoch 26/50
5/5            0s 86ms/step - loss:
0.0186 - val_loss: 0.0493 - learning_rate: 1.0000e-06
Epoch 27/50
5/5            0s 89ms/step - loss:
0.0199 - val_loss: 0.0491 - learning_rate: 1.0000e-06
Epoch 28/50
5/5            1s 104ms/step - loss:
0.0183 - val_loss: 0.0489 - learning_rate: 1.0000e-06
Epoch 29/50
5/5            0s 87ms/step - loss:
0.0189 - val_loss: 0.0487 - learning_rate: 1.0000e-06
```

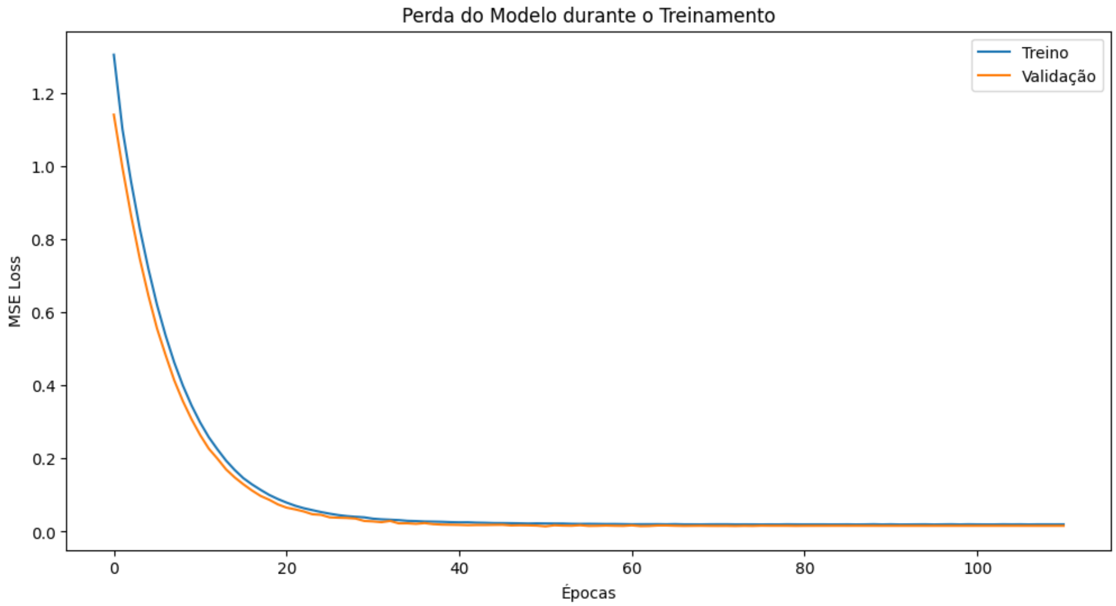
```
Epoch 30/50
5/5             0s 86ms/step - loss:
0.0178 - val_loss: 0.0485 - learning_rate: 1.0000e-06
Epoch 31/50
5/5             0s 93ms/step - loss:
0.0196 - val_loss: 0.0483 - learning_rate: 1.0000e-06
Epoch 32/50
5/5             0s 88ms/step - loss:
0.0185 - val_loss: 0.0482 - learning_rate: 1.0000e-06
Epoch 33/50
5/5             0s 91ms/step - loss:
0.0180 - val_loss: 0.0480 - learning_rate: 1.0000e-06
Epoch 34/50
5/5             0s 88ms/step - loss:
0.0191 - val_loss: 0.0478 - learning_rate: 1.0000e-06
Epoch 35/50
5/5             0s 90ms/step - loss:
0.0173 - val_loss: 0.0476 - learning_rate: 1.0000e-06
Epoch 36/50
5/5             0s 89ms/step - loss:
0.0186 - val_loss: 0.0474 - learning_rate: 1.0000e-06
Epoch 37/50
5/5             0s 92ms/step - loss:
0.0175 - val_loss: 0.0473 - learning_rate: 1.0000e-06
Epoch 38/50
5/5             0s 84ms/step - loss:
0.0176 - val_loss: 0.0471 - learning_rate: 1.0000e-06
Epoch 39/50
5/5             0s 88ms/step - loss:
0.0174 - val_loss: 0.0469 - learning_rate: 1.0000e-06
Epoch 40/50
5/5             0s 87ms/step - loss:
0.0189 - val_loss: 0.0467 - learning_rate: 1.0000e-06
Epoch 41/50
5/5             0s 85ms/step - loss:
0.0174 - val_loss: 0.0465 - learning_rate: 1.0000e-06
Epoch 42/50
5/5             0s 84ms/step - loss:
0.0181 - val_loss: 0.0464 - learning_rate: 1.0000e-06
Epoch 43/50
5/5             0s 81ms/step - loss:
0.0172 - val_loss: 0.0462 - learning_rate: 1.0000e-06
Epoch 44/50
5/5             0s 80ms/step - loss:
0.0184 - val_loss: 0.0461 - learning_rate: 1.0000e-06
Epoch 45/50
5/5             0s 78ms/step - loss:
0.0171 - val_loss: 0.0459 - learning_rate: 1.0000e-06
```

```
Epoch 46/50
5/5            0s 83ms/step - loss:
0.0190 - val_loss: 0.0458 - learning_rate: 1.0000e-06
Epoch 47/50
5/5            0s 88ms/step - loss:
0.0180 - val_loss: 0.0457 - learning_rate: 1.0000e-06
Epoch 48/50
5/5            0s 82ms/step - loss:
0.0177 - val_loss: 0.0455 - learning_rate: 1.0000e-06
Epoch 49/50
5/5            0s 82ms/step - loss:
0.0184 - val_loss: 0.0454 - learning_rate: 1.0000e-06
Epoch 50/50
5/5            0s 80ms/step - loss:
0.0167 - val_loss: 0.0453 - learning_rate: 1.0000e-06
```



Visualização da Perda durante o Treinamento:

[24]:
```
# Plotar a perda de treino e validação
plt.figure(figsize=(12,6))
plt.plot(history.history['loss'], label='Treino')
plt.plot(history.history['val_loss'], label='Validação')
plt.title('Perda do Modelo durante o Treinamento')
plt.xlabel('Épocas')
plt.ylabel('MSE Loss')
plt.legend()
plt.show()
```



10. Previsão e Avaliação do Modelo

Realizamos previsões para o conjunto de teste, invertemos o escalonamento dos dados e avaliamos o desempenho do modelo utilizando métricas como MSE e MAE.

[25]:
```
# Realizar previsões
predictions = model.predict(X_test)

# Inverter o escalonamento das previsões e dos valores reais
predictions_inverse = target_scaler.inverse_transform(predictions)
y_test_inverse = target_scaler.inverse_transform(y_test)

# Garantir valores não negativos após inversão
predictions_inverse = np.clip(predictions_inverse, a_min=0, a_max=None)
y_test_inverse = np.clip(y_test_inverse, a_min=0, a_max=None)

# Criar DataFrame para comparação
comparison_df = test_df.iloc[SEQ_LENGTH:].copy().reset_index(drop=True)
comparison_df['ValorUnitario_Previsto'] = predictions_inverse[:, 0]
comparison_df['Quantidade_Prevista'] = predictions_inverse[:, 1]

# Calcular métricas
mse_valor = mean_squared_error(y_test_inverse[:, 0], predictions_inverse[:, 0])
mae_valor = mean_absolute_error(y_test_inverse[:, 0], predictions_inverse[:, 0])

mse_quant = mean_squared_error(y_test_inverse[:, 1], predictions_inverse[:, 1])
mae_quant = mean_absolute_error(y_test_inverse[:, 1], predictions_inverse[:, 1])
```

```
r2_valor = r2_score(y_test_inverse[:, 0], predictions_inverse[:, 0])
r2_quant = r2_score(y_test_inverse[:, 1], predictions_inverse[:, 1])

# Exibir métricas
print(f"Valor Unitário - R²: {r2_valor:.4f}, MSE: {mse_valor:.4f}, MAE:␣
↪{mae_valor:.4f}")
print(f"Quantidade Vendida - R²: {r2_quant:.4f}, MSE: {mse_quant:.4f}, MAE:␣
↪{mae_quant:.4f}")
```

```
2/2           1s 594ms/step
Valor Unitário - R²: -46.7871, MSE: 1.0326, MAE: 1.0095
Quantidade Vendida - R²: -0.0247, MSE: 0.7824, MAE: 0.4182
```

11. Visualização dos Resultados

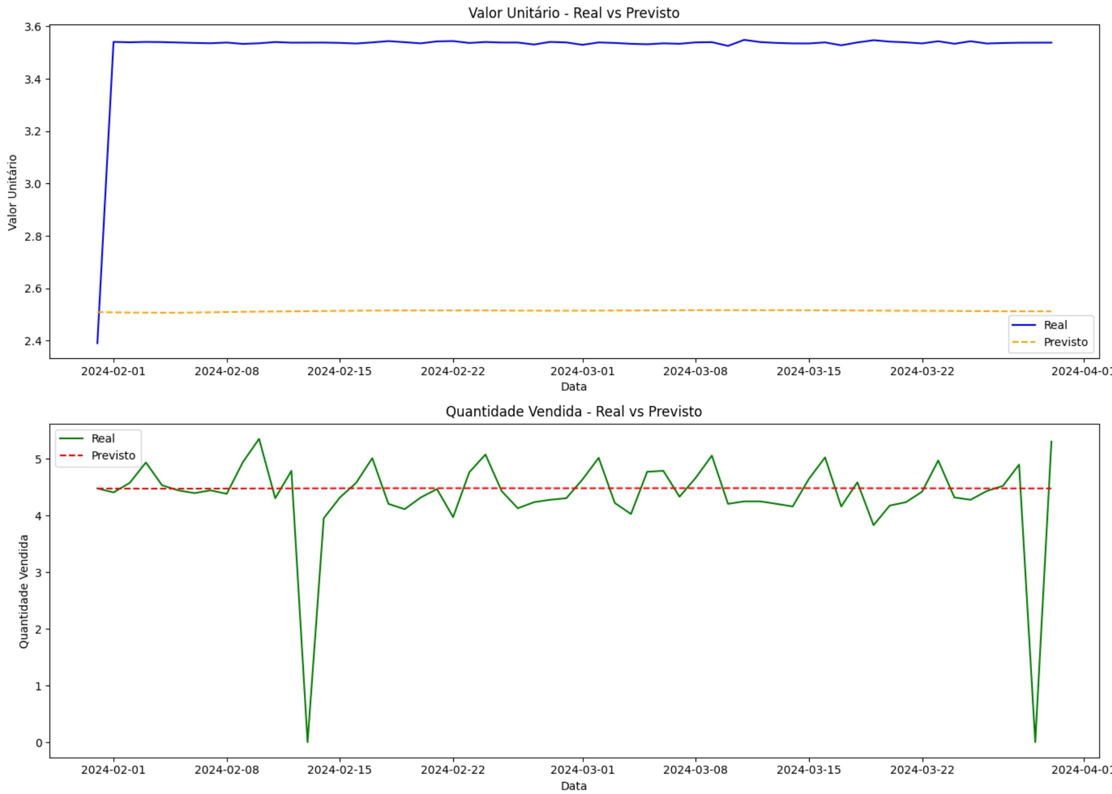
Visualizamos as previsões comparadas com os valores reais para entender o desempenho do modelo ao longo do tempo.

[26]:
```
# Plotagem - Real vs Previsto
plt.figure(figsize=(14,10))

# Valor Unitário
plt.subplot(2,1,1)
plt.plot(comparison_df['Data'], comparison_df['ValorUnitario'], label='Real',␣
↪color='blue')
plt.plot(comparison_df['Data'], comparison_df['ValorUnitario_Previsto'],␣
↪label='Previsto', color='orange', linestyle='dashed')
plt.title('Valor Unitário - Real vs Previsto')
plt.xlabel('Data')
plt.ylabel('Valor Unitário')
plt.legend()

# Quantidade Vendida
plt.subplot(2,1,2)
plt.plot(comparison_df['Data'], comparison_df['Quantidade'], label='Real',␣
↪color='green')
plt.plot(comparison_df['Data'], comparison_df['Quantidade_Prevista'],␣
↪label='Previsto', color='red', linestyle='dashed')
plt.title('Quantidade Vendida - Real vs Previsto')
plt.xlabel('Data')
plt.ylabel('Quantidade Vendida')
plt.legend()

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



12. Conclusão

Com base nas métricas e visualizações, podemos avaliar a precisão do modelo e identificar áreas para melhorias futuras.