In [1]:
```
# Importing pandas library
import pandas as pd

# Loading the CSV file
data = pd.read_csv("/Users/igormol/Downloads/vendas_20180102_20220826/traini

# Printing the structure of the data
print(data.info())

# Printing the content of the data
print(data)

training_data = data
```

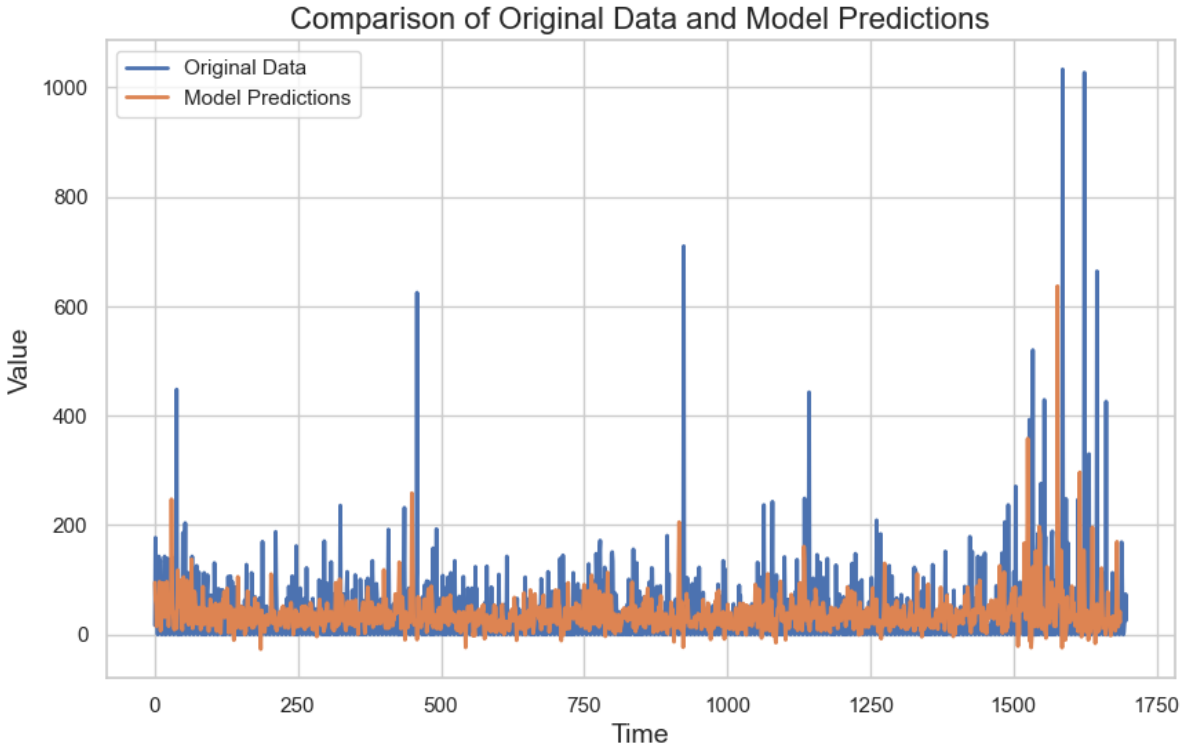
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1698 entries, 0 to 1697
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   time_scale      1698 non-null   int64  
 1   sum_quant_item  1698 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 26.7 KB
None
      time_scale  sum_quant_item
0              1            17.0
1              2           177.0
2              3           102.0
3              4            54.0
4              5            21.0
...          ...             ...
1693        1694             4.0
1694        1695            19.0
1695        1696            27.0
1696        1697            74.0
1697        1698            26.0

[1698 rows x 2 columns]
```

In [2]:
```
# Exponential_Smoothing_CNN_v2.py
#
# O programa a seguir consiste em uma implementação de um modelo híbrido de
# aprendizado profundo para a análise de séries temporais, combinando o méto
# de suavização exponencial com uma Rede Neural de Convolução.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Dense, Flatten
from sklearn.preprocessing import MinMaxScaler
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
import seaborn as sns

# Set the theme
sns.set_theme(style='whitegrid')

# Preparando os dados

# Nesta etapa, os dados são preparados para a Rede Neural Convolucional (CNN
# A função `preparar_dados` recebe uma série temporal e um parâmetro de jane
# O parâmetro de janela define o intervalo de etapas de tempo a serem consid
# na entrada para a modelagem. A função cria uma lista de padrões de entrada
# da série temporal. São retornados dois arrays Numpy: X como entrada e Y co
# para o modelo CNN. X é redimensionado para o formato tridimensional espera
# CNNs, ou seja, [amostras, timesteps, features].

def preparar_dados(series, window):
    X, y = [], []
    for i in range(len(series)-window):
        X.append(series[i:i+window])
        y.append(series[i+window])
    X = np.array(X)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1)) # Redimensionando para o
    y = np.array(y)
    return X, y

# Criando a CNN

# A função `criar_cnn` é utilizada para criar um modelo de CNN Sequencial. O
# consiste em várias camadas, incluindo uma camada Convolucional (Conv1D), u
# Flatten para transformar os tensores multidimensionais em 1D e duas camada
# onde a última camada contém apenas um neurônio como saída, uma vez que est
# problema de regressão. A função de ativação 'Relu' é utilizada para as cam
# ocultas e o modelo é compilado utilizando o otimizador Adam e a função de
# Erro Médio Quadrado.

def criar_cnn(window):
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_sha
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

# Aplicando suavização exponencial
```

```
# A função `aplicar_suavizacao_exponencial` aplica a suavização exponencial
# Este é um método simples de previsão para dados de séries temporais que cr
# de média móvel ponderada exponencialmente para os dados. O nível de suaviz
# (alpha) é definido como argumento com padrão 0.6. Valores de fator de suav
# (alpha) mais baixos dão mais peso a observações mais antigas, enquanto val
# fator de suavização mais altos dão mais peso a observações recentes.

def aplicar_suavizacao_exponencial(data, alpha=0.6):
    model = SimpleExpSmoothing(data)
    model_fit = model.fit(smoothing_level=alpha, optimized=False)
    return model_fit.fittedvalues

# Carregando dados e aplicando suavização exponencial
# training_data = pd.read_csv('nome_do_arquivo.csv')

training_data_smooth = aplicar_suavizacao_exponencial(training_data['sum_qua

# Normalizando os dados

# O MinMaxScaler do módulo de pré-processamento sklearn é utilizado para nor
# os dados de entrada para o intervalo [0,1]. Isso ajuda o modelo a convergi
# rapidamente e evita que valores de entrada baixos e altos viciem excessiva
# modelo.

scaler = MinMaxScaler(feature_range=(0, 1))
series = scaler.fit_transform(np.array(training_data_smooth).reshape(-1, 1))

# Preparando os dados para CNN

window = 10
X, y = preparar_dados(series, window)

# Criando e treinando a CNN

# O modelo CNN é criado usando a função `criar_cnn` e depois treinado nos da
# preparados por 200 épocas.

model = criar_cnn(window)
model.fit(X, y, epochs=200, verbose=0)

# Previsões

# O modelo faz previsões nos dados treinados e as previsões são transformada
# inversamente do range normalizado [0,1] para corresponder à escala dos dad

predictions = model.predict(X)
predictions = scaler.inverse_transform(predictions) # Retornando à escala or


# Plotando os dados originais e as previsões:
# A última parte do código é dedicada à visualização da série temporal origi
# com a predição, no mesmo gráfico, utilizando matplotlib e seaborn. O gráfi
# nos dados reais da série temporal e na série temporal prevista, para que o
# possa ver o quão bem as previsões correspondem aos dados originais. O rótu
# x 'Time', o rótulo do eixo y 'Values' e o título 'Comparison of Original D
# Model Predictions' são dados para o gráfico e uma legenda é exibida.

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(training_data['sum_quant_item'], label='Original Data', linewidth=2)
```

```
ax.plot(training_data['sum_quant_item'], label='Original Data', linewidth=2)
ax.plot(predictions, label='Model Predictions', linewidth=2)
plt.title('Comparison of Original Data and Model Predictions', fontsize=16)
plt.xlabel('Time', fontsize=14)
plt.ylabel('Value', fontsize=14)
plt.legend()
plt.show()
```

```
53/53 [==============================] - 2s 6ms/step
```



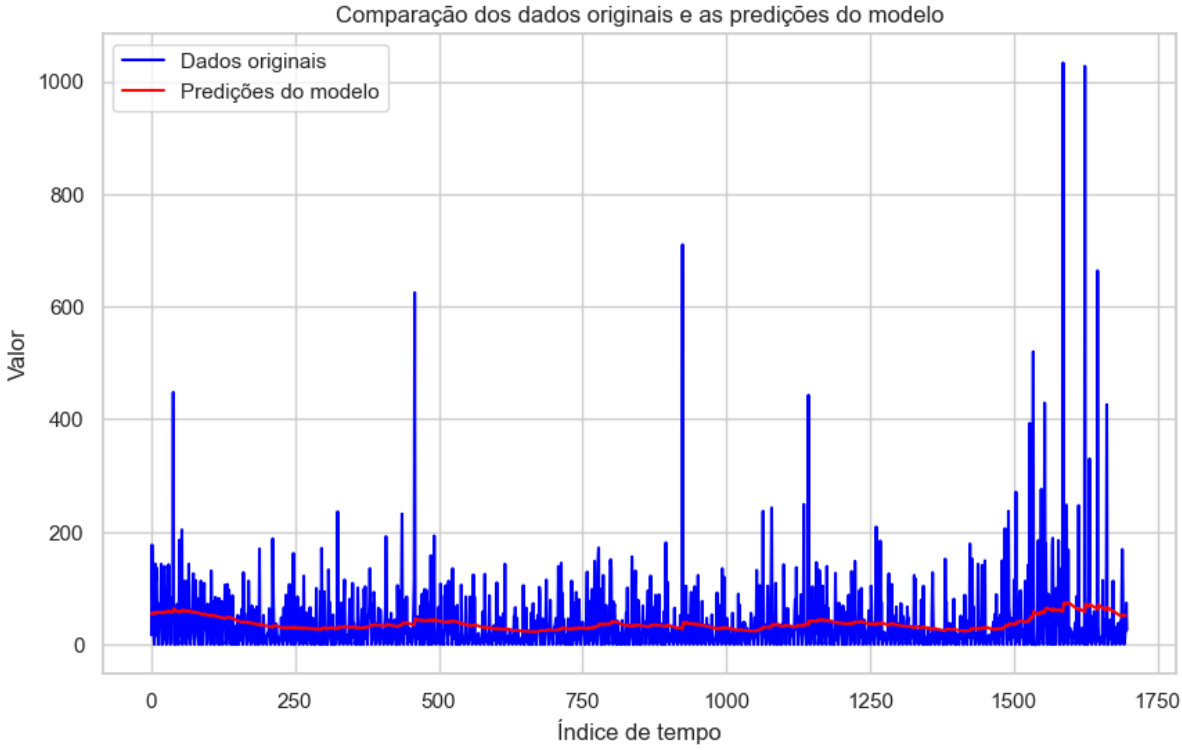
In [4]:
```
# Importar bibliotecas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# Suavização Exponencial
def aplicar_suavizacao_exponencial(data):
    modelo_suavizacao = ExponentialSmoothing(data)
    modelo_ajustado = modelo_suavizacao.fit()
    suavizado = modelo_ajustado.fittedvalues
    return suavizado

# Preparar dados para a RNN
def preparar_dados_rnn(data, look_back=1):
    dataX, dataY = [], []
    for i in range(len(data)-look_back-1):
        a = data[i:(i+look_back)]
        dataX.append(a)
        dataY.append(data[i + look_back])
    return np.array(dataX), np.array(dataY)

# Criação da RNN
def criar_rnn_model(look_back):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(4, input_shape=(1, look_back)))
    model.add(tf.keras.layers.Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Lendo os dados
# Para fim de exemplo, vou supor que foi um DataFrame do pandas
# training_data = pd.read_csv('seu_arquivo.csv')

# Aplicando suavização exponencial
training_data['suavizado'] = aplicar_suavizacao_exponencial(training_data['s

# Ajustando a escala dos dados
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(training_data['suavizado'].values.reshape(-1,

# Preparando os dados para a RNN
look_back = 1
trainX, trainY = preparar_dados_rnn(dataset, look_back)
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

# Criação e ajuste da RNN
model = criar_rnn_model(look_back)
model.fit(trainX, trainY, epochs=50, batch_size=1, verbose=2)

# Gerando predições
trainPredict = model.predict(trainX)
trainPredict = scaler.inverse_transform(trainPredict)

# Cálculo do erro do modelo
```

```
# Calculo do erro do modelo
trainScore = np.sqrt(mean_squared_error(trainY, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))

# Definindo o estilo dos gráficos
sns.set_style("whitegrid")

# Criação da figura e dos eixos
fig, ax = plt.subplots(figsize=(10, 6))

# Plotando os dados originais
ax.plot(training_data['sum_quant_item'].values, color='blue', label='Dados c

# Plotando as previsões do modelo
ax.plot(trainPredict, color='red', label='Predições do modelo')

# Adicionando a legenda
ax.legend(loc='upper left')

# Adicionando um título ao gráfico
ax.set_title('Comparação dos dados originais e as predições do modelo')

# Adicionando rótulos aos eixos
ax.set_xlabel('Índice de tempo')
ax.set_ylabel('Valor')

# Mostrando o gráfico
plt.show()
```

```
Epoch 1/50
1696/1696 - 27s - loss: 0.0302 - 27s/epoch - 16ms/step
Epoch 2/50
1696/1696 - 13s - loss: 0.0012 - 13s/epoch - 8ms/step
Epoch 3/50
1696/1696 - 11s - loss: 4.1363e-04 - 11s/epoch - 7ms/step
Epoch 4/50
1696/1696 - 10s - loss: 4.1764e-04 - 10s/epoch - 6ms/step
Epoch 5/50
1696/1696 - 11s - loss: 4.1015e-04 - 11s/epoch - 7ms/step
Epoch 6/50
1696/1696 - 10s - loss: 4.2341e-04 - 10s/epoch - 6ms/step
Epoch 7/50
1696/1696 - 9s - loss: 4.2027e-04 - 9s/epoch - 6ms/step
Epoch 8/50
1696/1696 - 8s - loss: 4.1442e-04 - 8s/epoch - 5ms/step
Epoch 9/50
1696/1696 - 8s - loss: 4.1084e-04 - 8s/epoch - 5ms/step
Epoch 10/50
1696/1696 - 11s - loss: 4.1280e-04 - 11s/epoch - 6ms/step
Epoch 11/50
1696/1696 - 10s - loss: 4.1090e-04 - 10s/epoch - 6ms/step
Epoch 12/50
1696/1696 - 9s - loss: 4.2630e-04 - 9s/epoch - 6ms/step
Epoch 13/50
1696/1696 - 10s - loss: 4.1784e-04 - 10s/epoch - 6ms/step
Epoch 14/50
1696/1696 - 9s - loss: 4.1251e-04 - 9s/epoch - 5ms/step
Epoch 15/50
1696/1696 - 8s - loss: 4.1993e-04 - 8s/epoch - 5ms/step
Epoch 16/50
```

```
1696/1696 - 9s - loss: 4.1867e-04 - 9s/epoch - 6ms/step
Epoch 17/50
1696/1696 - 9s - loss: 4.1576e-04 - 9s/epoch - 5ms/step
Epoch 18/50
1696/1696 - 9s - loss: 4.1513e-04 - 9s/epoch - 5ms/step
Epoch 19/50
1696/1696 - 10s - loss: 4.1009e-04 - 10s/epoch - 6ms/step
Epoch 20/50
1696/1696 - 10s - loss: 4.1418e-04 - 10s/epoch - 6ms/step
Epoch 21/50
1696/1696 - 8s - loss: 4.1533e-04 - 8s/epoch - 5ms/step
Epoch 22/50
1696/1696 - 9s - loss: 4.1223e-04 - 9s/epoch - 5ms/step
Epoch 23/50
1696/1696 - 10s - loss: 4.1863e-04 - 10s/epoch - 6ms/step
Epoch 24/50
1696/1696 - 11s - loss: 4.2063e-04 - 11s/epoch - 7ms/step
Epoch 25/50
1696/1696 - 10s - loss: 4.1038e-04 - 10s/epoch - 6ms/step
Epoch 26/50
1696/1696 - 10s - loss: 4.1040e-04 - 10s/epoch - 6ms/step
Epoch 27/50
1696/1696 - 10s - loss: 4.1613e-04 - 10s/epoch - 6ms/step
Epoch 28/50
1696/1696 - 8s - loss: 4.1168e-04 - 8s/epoch - 5ms/step
Epoch 29/50
1696/1696 - 11s - loss: 4.1487e-04 - 11s/epoch - 6ms/step
Epoch 30/50
1696/1696 - 9s - loss: 4.1390e-04 - 9s/epoch - 6ms/step
Epoch 31/50
1696/1696 - 8s - loss: 4.1662e-04 - 8s/epoch - 5ms/step
Epoch 32/50
1696/1696 - 8s - loss: 4.0789e-04 - 8s/epoch - 5ms/step
Epoch 33/50
1696/1696 - 9s - loss: 4.2163e-04 - 9s/epoch - 6ms/step
Epoch 34/50
1696/1696 - 11s - loss: 4.1067e-04 - 11s/epoch - 6ms/step
Epoch 35/50
1696/1696 - 9s - loss: 4.1444e-04 - 9s/epoch - 5ms/step
Epoch 36/50
1696/1696 - 9s - loss: 4.0764e-04 - 9s/epoch - 5ms/step
Epoch 37/50
1696/1696 - 8s - loss: 4.1850e-04 - 8s/epoch - 5ms/step
Epoch 38/50
1696/1696 - 9s - loss: 4.1602e-04 - 9s/epoch - 5ms/step
Epoch 39/50
1696/1696 - 8s - loss: 4.1096e-04 - 8s/epoch - 5ms/step
Epoch 40/50
1696/1696 - 8s - loss: 4.0272e-04 - 8s/epoch - 5ms/step
Epoch 41/50
1696/1696 - 9s - loss: 4.0664e-04 - 9s/epoch - 5ms/step
Epoch 42/50
1696/1696 - 8s - loss: 4.1321e-04 - 8s/epoch - 5ms/step
Epoch 43/50
1696/1696 - 10s - loss: 4.1318e-04 - 10s/epoch - 6ms/step
Epoch 44/50
1696/1696 - 12s - loss: 4.1198e-04 - 12s/epoch - 7ms/step
Epoch 45/50
1696/1696 - 14s - loss: 4.0970e-04 - 14s/epoch - 8ms/step
```

```
Epoch 46/50
1696/1696 - 12s - loss: 4.1254e-04 - 12s/epoch - 7ms/step
Epoch 47/50
1696/1696 - 10s - loss: 4.0294e-04 - 10s/epoch - 6ms/step
Epoch 48/50
1696/1696 - 11s - loss: 4.0734e-04 - 11s/epoch - 7ms/step
Epoch 49/50
1696/1696 - 10s - loss: 4.1026e-04 - 10s/epoch - 6ms/step
Epoch 50/50
1696/1696 - 10s - loss: 4.0482e-04 - 10s/epoch - 6ms/step
53/53 [==============================] - 2s 4ms/step
Train Score: 38.90 RMSE
```



In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf


# Normalizando os dados
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(training_data[['sum_quant_item']])

# Função para criar série temporal adequada para o Transformer
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.asarray(dataX), np.asarray(dataY)

# Função para a construção do modelo Transformer
def build_model(input_shape, head_size, num_heads, ff_dim, num_transformer_b
    inputs = tf.keras.Input(shape=input_shape)
    x = inputs
    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout_rat
    x = tf.keras.layers.GlobalAveragePooling1D(data_format="channels_first")
    for units in mlp_units:
        x = tf.keras.layers.Dense(units, activation=tf.keras.activations.rel
        x = tf.keras.layers.Dropout(mlp_dropout_rate)(x)
    outputs = tf.keras.layers.Dense(1, activation='linear')(x)
    return tf.keras.Model(inputs, outputs)

# Função para a construção do encoder do Transformer
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout_rate):
    x = tf.keras.layers.LayerNormalization(epsilon=1e-6)(inputs)
    x = tf.keras.layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout_rate
    )(x, x)
    x = tf.keras.layers.Dropout(dropout_rate)(x)
    res = x + inputs

    x = tf.keras.layers.LayerNormalization(epsilon=1e-6)(res)
    x = tf.keras.layers.Conv1D(filters=ff_dim, kernel_size=1, activation="re
    x = tf.keras.layers.Dropout(dropout_rate)(x)
    x = tf.keras.layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res

# Utilizando a função para criar um novo conjunto de dados
look_back = 10
X, y = create_dataset(data_scaled, look_back)

# Dividir os dados entre conjunto de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Expandir as dimensões
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Parâmetros para o modelo Transformer
head_size = 64
num_heads = 4
ff_dim = 32
dropout_rate = 0.1
num_transformer_blocks = 2

# Parâmetros para o modelo MLP
mlp_units = [128]
mlp_dropout_rate = 0.1

# Construir o modelo
model = build_model(X_train.shape[1:], head_size, num_heads, ff_dim, num_tra

# Compilar o modelo
model.compile(loss=tf.keras.losses.MeanSquaredError(), optimizer=tf.keras.op

# Treinando o modelo
history = model.fit(X_train, y_train, batch_size=64, epochs=100, validation_

# Avaliando o modelo
plt.figure(figsize=(10,6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Fazendo as previsões
predicted_quantity = model.predict(X_test)
predicted_quantity = scaler.inverse_transform(predicted_quantity)

# Plotando os resultados
plt.figure(figsize=(10,6))
plt.plot(scaler.inverse_transform(y_test.reshape(-1, 1)), color='blue', labe
plt.plot(predicted_quantity, color='red', label='Previsões')
plt.title('Previsão do modelo Transformer')
plt.xlabel('Escala de tempo')
plt.ylabel('sum_quant_item')
plt.legend()
plt.show()
```

```
Epoch 1/100
17/17 [==============================] - 11s 102ms/step - loss: 0.0046 - me
an_absolute_percentage_error: 5946996.5000 - val_loss: 0.0025 - val_mean_ab
solute_percentage_error: 7956494.0000
Epoch 2/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0039 - mean
_absolute_percentage_error: 5183410.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6899664.5000
Epoch 3/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0037 - mean
_absolute_percentage_error: 5397045.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6635875.5000
Epoch 4/100
17/17 [==============================] - 1s 34ms/step - loss: 0.0037 - mean
```

```
_absolute_percentage_error: 5342869.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5915128.5000
Epoch 5/100
17/17 [==============================] - 1s 39ms/step - loss: 0.0037 - mean
_absolute_percentage_error: 5324738.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5673208.0000
Epoch 6/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0037 - mean
_absolute_percentage_error: 4748406.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 7026825.0000
Epoch 7/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0037 - mean
_absolute_percentage_error: 5221034.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5080440.0000
Epoch 8/100
17/17 [==============================] - 1s 45ms/step - loss: 0.0037 - mean
_absolute_percentage_error: 4775158.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6223128.0000
Epoch 9/100
17/17 [==============================] - 1s 34ms/step - loss: 0.0038 - mean
_absolute_percentage_error: 4966952.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5402532.5000
Epoch 10/100
17/17 [==============================] - 1s 34ms/step - loss: 0.0037 - mean
_absolute_percentage_error: 5181588.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5936530.0000
Epoch 11/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0036 - mean
_absolute_percentage_error: 4875544.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6892411.0000
Epoch 12/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0036 - mean
_absolute_percentage_error: 4931606.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6648261.5000
Epoch 13/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0036 - mean
_absolute_percentage_error: 5046774.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6708686.0000
Epoch 14/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0036 - mean
_absolute_percentage_error: 4969614.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5514246.5000
Epoch 15/100
17/17 [==============================] - 1s 41ms/step - loss: 0.0036 - mean
_absolute_percentage_error: 4898180.5000 - val_loss: 0.0020 - val_mean_abso
lute_percentage_error: 6280894.5000
Epoch 16/100
17/17 [==============================] - 1s 45ms/step - loss: 0.0036 - mean
_absolute_percentage_error: 4462856.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6832866.0000
Epoch 17/100
17/17 [==============================] - 1s 46ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 5135021.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5059138.0000
Epoch 18/100
17/17 [==============================] - 1s 48ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4362532.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6646929.0000
Epoch 19/100
```

```
17/17 [==============================] - 1s 48ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4950838.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5751848.5000
Epoch 20/100
17/17 [==============================] - 1s 47ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4539238.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6538148.0000
Epoch 21/100
17/17 [==============================] - 1s 46ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4690479.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6603916.0000
Epoch 22/100
17/17 [==============================] - 1s 44ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4722627.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 7722394.5000
Epoch 23/100
17/17 [==============================] - 1s 31ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4711337.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 4947974.0000
Epoch 24/100
17/17 [==============================] - 1s 30ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4699384.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6278720.0000
Epoch 25/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4712161.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6339950.0000
Epoch 26/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4440973.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6683003.5000
Epoch 27/100
17/17 [==============================] - 1s 39ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4490665.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5700363.0000
Epoch 28/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4524800.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6432314.0000
Epoch 29/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4593752.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5652247.0000
Epoch 30/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4710864.5000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5637670.5000
Epoch 31/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4186294.2500 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 6534016.5000
Epoch 32/100
17/17 [==============================] - 1s 33ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4497980.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6150109.5000
Epoch 33/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 5026821.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 4582203.5000
```

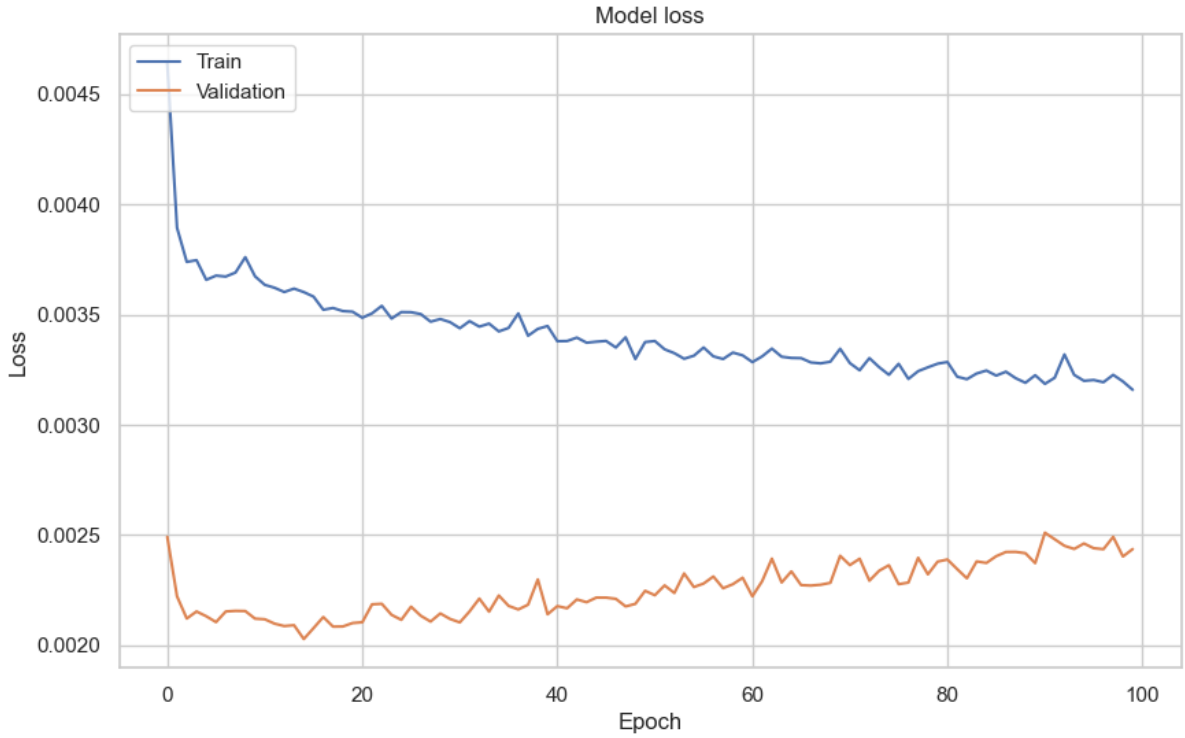
```
Epoch 34/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 3912537.2500 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6775986.5000
Epoch 35/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4519995.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 4619568.5000
Epoch 36/100
17/17 [==============================] - 1s 42ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4720316.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5749721.5000
Epoch 37/100
17/17 [==============================] - 1s 41ms/step - loss: 0.0035 - mean
_absolute_percentage_error: 4370073.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6952530.0000
Epoch 38/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4235614.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5748604.0000
Epoch 39/100
17/17 [==============================] - 1s 50ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4731376.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 4391836.5000
Epoch 40/100
17/17 [==============================] - 1s 47ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4275352.0000 - val_loss: 0.0021 - val_mean_abso
lute_percentage_error: 5333426.0000
Epoch 41/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4353188.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5715784.5000
Epoch 42/100
17/17 [==============================] - 1s 40ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4551020.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5236235.0000
Epoch 43/100
17/17 [==============================] - 1s 31ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4324449.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6734733.0000
Epoch 44/100
17/17 [==============================] - 1s 43ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4149063.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6274696.0000
Epoch 45/100
17/17 [==============================] - 1s 40ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4571242.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5365606.0000
Epoch 46/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4540338.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6524831.0000
Epoch 47/100
17/17 [==============================] - 1s 43ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4362728.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6544578.0000
Epoch 48/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4447976.0000 - val_loss: 0.0022 - val_mean_abso
```

```
lute_percentage_error: 5478003.5000
Epoch 49/100
17/17 [==============================] - 1s 32ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4373275.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5675456.5000
Epoch 50/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4332090.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5082747.5000
Epoch 51/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4180634.7500 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 6079438.0000
Epoch 52/100
17/17 [==============================] - 1s 41ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3912884.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5977676.0000
Epoch 53/100
17/17 [==============================] - 1s 40ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4211132.0000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5818883.0000
Epoch 54/100
17/17 [==============================] - 1s 33ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4613179.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5089513.5000
Epoch 55/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4026081.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6496352.0000
Epoch 56/100
17/17 [==============================] - 1s 32ms/step - loss: 0.0034 - mean
_absolute_percentage_error: 4145745.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5543958.5000
Epoch 57/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4426191.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 4252806.5000
Epoch 58/100
17/17 [==============================] - 1s 47ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3911343.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6639902.0000
Epoch 59/100
17/17 [==============================] - 1s 41ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4075991.7500 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5844252.5000
Epoch 60/100
17/17 [==============================] - 1s 53ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4192565.7500 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5631221.0000
Epoch 61/100
17/17 [==============================] - 1s 49ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4418135.5000 - val_loss: 0.0022 - val_mean_abso
lute_percentage_error: 5871602.5000
Epoch 62/100
17/17 [==============================] - 1s 54ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3972782.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6762390.0000
Epoch 63/100
17/17 [==============================] - 1s 51ms/step - loss: 0.0033 - mean
```

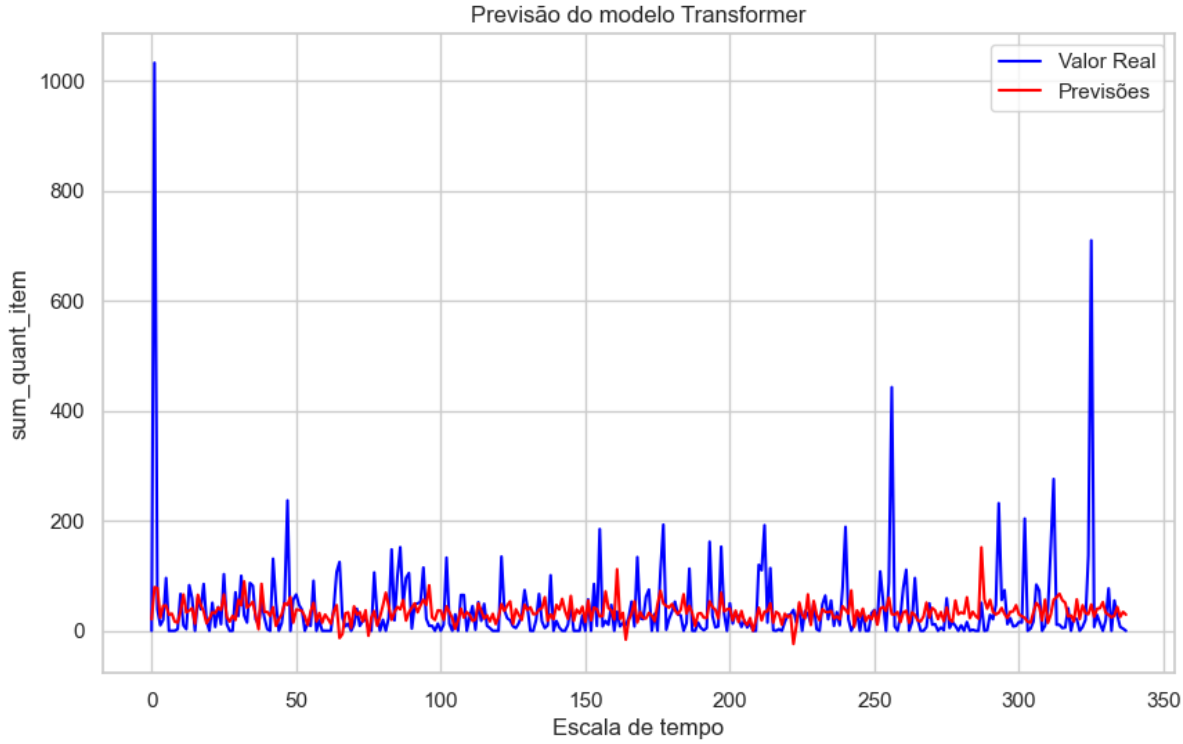
```
_absolute_percentage_error: 4196304.0000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5509053.0000
Epoch 64/100
17/17 [==============================] - 1s 49ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3863144.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6634191.5000
Epoch 65/100
17/17 [==============================] - 1s 51ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4536147.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5508915.5000
Epoch 66/100
17/17 [==============================] - 1s 47ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3758804.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6614845.0000
Epoch 67/100
17/17 [==============================] - 1s 40ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4172625.5000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5712144.0000
Epoch 68/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3966013.7500 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6206054.5000
Epoch 69/100
17/17 [==============================] - 1s 42ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4231601.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5744733.5000
Epoch 70/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4084672.5000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 4371237.5000
Epoch 71/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3867702.5000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5576486.0000
Epoch 72/100
17/17 [==============================] - 1s 32ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4415590.5000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 4787703.5000
Epoch 73/100
17/17 [==============================] - 1s 50ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4318374.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5632329.0000
Epoch 74/100
17/17 [==============================] - 1s 43ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3570760.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6187362.5000
Epoch 75/100
17/17 [==============================] - 1s 32ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4164973.7500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 6737997.5000
Epoch 76/100
17/17 [==============================] - 1s 36ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4117904.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 6353650.5000
Epoch 77/100
17/17 [==============================] - 1s 40ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3968024.2500 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5250760.5000
Epoch 78/100
```

```
17/17 [==============================] - 1s 38ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3988679.7500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5236574.0000
Epoch 79/100
17/17 [==============================] - 1s 40ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3972737.0000 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5476461.5000
Epoch 80/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4171343.2500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 4908862.0000
Epoch 81/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 3791515.2500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5054841.0000
Epoch 82/100
17/17 [==============================] - 1s 46ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3944418.7500 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 4701449.0000
Epoch 83/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4044606.7500 - val_loss: 0.0023 - val_mean_abso
lute_percentage_error: 5344011.0000
Epoch 84/100
17/17 [==============================] - 1s 34ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3987214.0000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 6323774.0000
Epoch 85/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4178651.2500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5544988.5000
Epoch 86/100
17/17 [==============================] - 1s 43ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3710823.5000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5882702.5000
Epoch 87/100
17/17 [==============================] - 1s 39ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4129686.2500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 6698925.0000
Epoch 88/100
17/17 [==============================] - 1s 40ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4052011.7500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 4794695.0000
Epoch 89/100
17/17 [==============================] - 1s 44ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3828986.2500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5377169.5000
Epoch 90/100
17/17 [==============================] - 1s 49ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3945478.7500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5744545.5000
Epoch 91/100
17/17 [==============================] - 1s 46ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3767981.2500 - val_loss: 0.0025 - val_mean_abso
lute_percentage_error: 4451386.0000
Epoch 92/100
17/17 [==============================] - 1s 45ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4093559.5000 - val_loss: 0.0025 - val_mean_abso
lute_percentage_error: 4901289.5000
```

```
Epoch 93/100
17/17 [==============================] - 1s 43ms/step - loss: 0.0033 - mean
_absolute_percentage_error: 4344350.0000 - val_loss: 0.0025 - val_mean_abso
lute_percentage_error: 5686213.0000
Epoch 94/100
17/17 [==============================] - 1s 42ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4019567.2500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5507497.0000
Epoch 95/100
17/17 [==============================] - 1s 38ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3874502.0000 - val_loss: 0.0025 - val_mean_abso
lute_percentage_error: 4401375.0000
Epoch 96/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3878647.7500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5403844.5000
Epoch 97/100
17/17 [==============================] - 1s 39ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3864042.5000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 4870589.0000
Epoch 98/100
17/17 [==============================] - 1s 35ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3787333.0000 - val_loss: 0.0025 - val_mean_abso
lute_percentage_error: 4651292.5000
Epoch 99/100
17/17 [==============================] - 1s 43ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 4047344.2500 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5592745.0000
Epoch 100/100
17/17 [==============================] - 1s 37ms/step - loss: 0.0032 - mean
_absolute_percentage_error: 3852054.0000 - val_loss: 0.0024 - val_mean_abso
lute_percentage_error: 5485260.5000
```



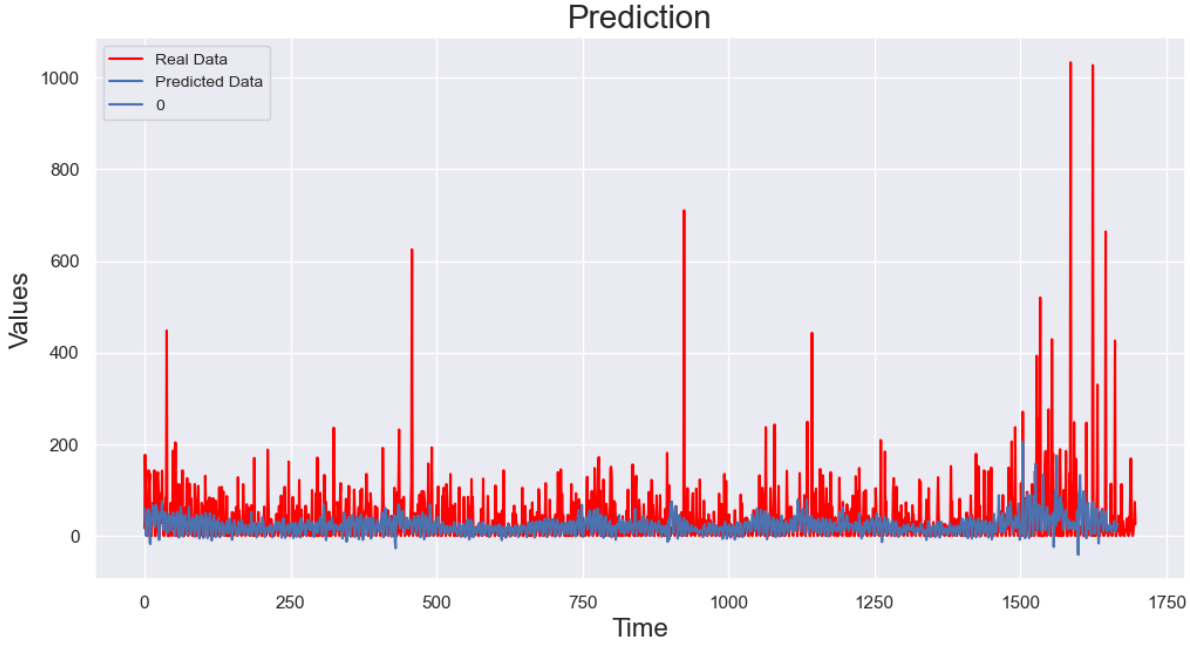

```
11/11 [==============================] - 1s 12ms/step
```


Previsão do modelo Transformer
Valor Real
Previsões
sum_quant_item
Escala de tempo
0
200
400
600
800
1000
0
50
100
150
200
250
300
350

In [6]:
```
# Hybrid_CNN_GRU.py
#
# O código-fonte a seguir apresenta uma implementação de modelagem de séries
# temporais utilizando uma abordagem híbrida, combinando Redes Neurais Convo
# cionais (CNN) e Unidades Recorrentes de Gated (GRU). Esta técnica é empreg
# para prever futuros valores em uma série temporal representada pelos dados
# em ``training_data''. Esta abordagem híbrida, combinando características
# aprendidas por meio de convoluções com mecanismos recorrentes, é particula
# mente útil para capturar padrões temporais complexos em séries temporais.

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, GRU, Dense
import seaborn as sns

# Extração e Normalização dos Dados:
# Extrai os dados de treinamento e realiza a normalização usando Min-Max Sca

training_set = training_data.iloc[:, 1:2].values
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)

# Criação de Conjunto de Dados Temporais:
# Define uma função para criar pares de entrada e saída para o treinamento d
# modelo, considerando o número de observações passadas (look_back), o horiz
# de previsão (forecast_horizon), e o tamanho do lote (batch_size).

def create_dataset(dataset, look_back=1, forecast_horizon=1, batch_size=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-forecast_horizon+1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back: i + look_back + forecast_horizon

    return np.array(dataX), np.array(dataY)

# Parâmetros e Criação do Conjunto de Dados Temporais:
# Especifica os parâmetros para a função e cria o conjunto de dados temporai

look_back = 30
forecast_horizon = 1
batch_size = 32
X_train, y_train = create_dataset(training_set_scaled, look_back, forecast_h
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# Função para Criação do Modelo:
# Define uma função para construir o modelo, incluindo uma camada de convolu
# (CNN) seguida por camadas GRU e uma camada densa de saída.

def create_model():
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_sha
    model.add(GRU(units=50, return_sequences=True))
    model.add(GRU(units=50))
    model.add(Dense(units = forecast_horizon))
```

```
    return model

# Criação e Compilação do Modelo:
# Cria o modelo usando a função definida e o compila com o otimizador 'adam'
# função de perda sendo o erro quadrático médio.

model = create_model()
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
history = model.fit(X_train, y_train, epochs = 100, batch_size = batch_size,

# Treinamento do Modelo:
# Treina o modelo com os dados de treino, utilizando 100 épocas e um tamanho
# lote de 32.

y_pred = model.predict(X_train)
training_predicted = sc.inverse_transform(y_pred)

# Exibição do Gráfico:

sns.set(style="darkgrid")
plt.figure(figsize=(12,6))
sns.lineplot(data=training_data["sum_quant_item"].values, color='red', label
sns.lineplot(data=training_predicted, color='blue', label='Predicted Data')
plt.title("Prediction", fontsize=20)
plt.xlabel("Time", fontsize=16)
plt.ylabel("Values", fontsize=16)
plt.legend(fontsize=10)
plt.show()
```

```
53/53 [==============================] - 2s 13ms/step
```



In [ ]: