Aim: To implement a Recurrent Neural Network (RNN) for predicating time series data

Run the code to unzip the dataset.zip file after uploding

```
#@title Run the code to unzip the dataset.zip file after uploding
import zipfile
import io
# Replace 'your_zip_file.zip' with the name of your uploaded zip file
with zipfile.ZipFile('dataset.zip', 'r') as zip_ref:
    zip_ref.extractall()


import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Load the weather dataset
weather_data = pd.read_csv('/content/weather_features.csv')

# Select relevant columns for analysis
selected_features = ['temp', 'temp_min', 'temp_max', 'pressure', 'humidity', 'wind_speed', 'rain_1h', 'rain_3h', 'snow_3h', 'clouds_all']
X = weather_data[selected_features].values

# Normalize the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Define the input sequence length
seq_length = 10

# Prepare the data for training
X_train = []
y_train = []
for i in range(len(X_scaled) - seq_length):
    X_train.append(X_scaled[i:i + seq_length])
    y_train.append(X_scaled[i + seq_length])

X_train, y_train = np.array(X_train), np.array(y_train)

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Define the RNN model
model = Sequential([
    SimpleRNN(32, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True),
    Dropout(0.2),
    SimpleRNN(32),
    Dropout(0.2),
    Dense(1)
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Define early stopping
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=5, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_data=(X_val, y_val), callbacks=[early_stop])

# Evaluate the model on both training and validation data
train_loss = model.evaluate(X_train, y_train)
val_loss = model.evaluate(X_val, y_val)
print("Training Loss:", train_loss)
print("Validation Loss:", val_loss)

# Visualize the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```