39_Sanskruti Nijai

EXPERIMENT NO 08

Design and implement RNN for classification of temporal data , sequence to sequence data modelling etc.

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import yfinance as yf
    from sklearn.preprocessing import MinMaxScaler
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
    from keras.callbacks import EarlyStopping
```

```
symbol = "GOOGL"  # Google's stock symbol
start_date = "2015-01-01"
end_date = "2021-12-31"

data = yf.download(symbol, start=start_date, end=end_date)
data.head()
```

```
[*********************100%%**********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2015-01-02	26.629999	26.790001	26.393999	26.477501	26.477501	26480000
2015-01-05	26.357500	26.399500	25.887501	25.973000	25.973000	41182000
2015-01-06	26.025000	26.060499	25.277500	25.332001	25.332001	54456000
2015-01-07	25.547501	25.574499	25.182501	25.257500	25.257500	46918000
2015-01-08	25.075500	25.375000	24.750999	25.345501	25.345501	73054000

```
[3] scaler = MinMaxScaler()
    data['Close_scaled'] = scaler.fit_transform(data['Close'].values.reshape(-1, 1))
```

```
[4] sequence_length = 10

    X, y = [], []

    for i in range(len(data) - sequence_length):
        X.append(data['Close_scaled'].values[i:i+sequence_length])
        y.append(data['Close_scaled'].values[i+sequence_length])

    X = np.array(X)
    y = np.array(y)
```

```
[5] train_size = int(0.8 * len(X))
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]
```

```
# Reshape and normalize data
X_train = X_train.reshape(-1, sequence_length, 1).astype(np.float32)
y_train = y_train.astype(np.float32)

# Build and compile the RNN model
model = Sequential([
    SimpleRNN(units=50, activation='relu', return_sequences=True, input_shape=(sequence_length, 1)),
    SimpleRNN(units=50, activation='relu', return_sequences=True),
    SimpleRNN(units=50, activation='relu'),
    Dense(units=1)
])

model.compile(optimizer='adam',loss='mse',metrics='mean_absolute_error')

# Train the model
callbacks = [EarlyStopping(monitor='loss',patience=10,restore_best_weights=True)]
model.fit(X_train, y_train, epochs=200, batch_size=32, callbacks=callbacks)
```

```
44/44 [==============================] - 1s 15ms/step - loss: 7.2777e-05 - mean_absolute_error: 0.0060
Epoch 14/200
44/44 [==============================] - 1s 17ms/step - loss: 6.9196e-05 - mean_absolute_error: 0.0058
Epoch 15/200
44/44 [==============================] - 1s 15ms/step - loss: 6.9237e-05 - mean_absolute_error: 0.0059
Epoch 16/200
44/44 [==============================] - 1s 12ms/step - loss: 6.9727e-05 - mean_absolute_error: 0.0059
Epoch 17/200
44/44 [==============================] - 0s 9ms/step - loss: 6.5833e-05 - mean_absolute_error: 0.0057
Epoch 18/200
44/44 [==============================] - 0s 8ms/step - loss: 7.0078e-05 - mean_absolute_error: 0.0059
Epoch 19/200
44/44 [==============================] - 0s 9ms/step - loss: 6.4460e-05 - mean_absolute_error: 0.0056
Epoch 20/200
44/44 [==============================] - 0s 9ms/step - loss: 6.4355e-05 - mean_absolute_error: 0.0057
Epoch 21/200
44/44 [==============================] - 0s 8ms/step - loss: 7.0397e-05 - mean_absolute_error: 0.0059
Epoch 22/200
44/44 [==============================] - 0s 9ms/step - loss: 6.2250e-05 - mean_absolute_error: 0.0056
Epoch 23/200
44/44 [==============================] - 0s 8ms/step - loss: 6.0961e-05 - mean_absolute_error: 0.0054
Epoch 24/200
44/44 [==============================] - 0s 8ms/step - loss: 6.0329e-05 - mean_absolute_error: 0.0054
Epoch 25/200
44/44 [==============================] - 0s 9ms/step - loss: 6.5765e-05 - mean_absolute_error: 0.0057
Epoch 26/200
44/44 [==============================] - 0s 9ms/step - loss: 5.6806e-05 - mean_absolute_error: 0.0051
Epoch 27/200
44/44 [==============================] - 0s 9ms/step - loss: 6.0378e-05 - mean_absolute_error: 0.0054
Epoch 28/200
44/44 [==============================] - 0s 10ms/step - loss: 6.2154e-05 - mean_absolute_error: 0.0055
Epoch 29/200
44/44 [==============================] - 0s 8ms/step - loss: 6.2913e-05 - mean_absolute_error: 0.0056
Epoch 30/200
44/44 [==============================] - 0s 9ms/step - loss: 5.3960e-05 - mean_absolute_error: 0.0049
Epoch 31/200
44/44 [==============================] - 0s 9ms/step - loss: 5.0250e-05 - mean_absolute_error: 0.0048
Epoch 32/200
44/44 [==============================] - 0s 8ms/step - loss: 5.0285e-05 - mean_absolute_error: 0.0047
Epoch 33/200
44/44 [==============================] - 0s 9ms/step - loss: 6.5087e-05 - mean_absolute_error: 0.0056
Epoch 34/200
44/44 [==============================] - 0s 8ms/step - loss: 5.9047e-05 - mean_absolute_error: 0.0053
```

```
[7] y_pred = model.predict(X_test)

11/11 [==============================] - 0s 3ms/step
```

```
[8] y_test = y_test.reshape(-1, 1)
    y_pred = y_pred.reshape(-1, 1)
    y_test_original = scaler.inverse_transform(y_test)
    y_pred_original = scaler.inverse_transform(y_pred)
```

```
plt.figure(figsize=(12, 6))
plt.plot(data.index[train_size+sequence_length:], y_test_original, label='Actual Prices', color='blue')
plt.plot(data.index[train_size+sequence_length:], y_pred_original, label='Predicted Prices', color='red')
plt.title('Google Stock Price Prediction with RNN')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```