In [ ]:
```
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split

data = pd.read_csv('state_consumption.csv')
data
```

Out[ ]:

	Goa	Gujarat	Maharashtra	DNH	Rajasthan
0	12.8	319.5	428.6	18.6	234.1
1	13.7	316.7	419.6	18.2	240.2
2	12.6	301.9	395.8	16.7	239.8
3	13.0	313.2	411.1	17.6	239.1
4	12.9	320.7	408.6	18.6	240.4
...	...	...	...	...	...
454	12.3	312.0	421.6	18.2	219.8
455	8.8	289.7	369.2	17.4	203.2
456	8.8	297.9	395.5	18.3	212.6
457	8.8	296.8	395.5	18.7	225.5
458	10.2	288.7	399.6	18.5	225.7

459 rows × 5 columns

In [ ]:
```
model_filename = "my_table_model.h5"
if os.path.exists(model_filename):
    # Load the existing model
    model = tf.keras.models.load_model(model_filename)
    print("Existing model loaded.")
else:
    model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(5,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # No activation function for regression
])
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
    print("New model created.")
```

```
Existing model loaded.
```

In [ ]:
```
# Split data into features and target
X = data[['Goa', 'Gujarat', 'Maharashtra', 'DNH', 'Rajasthan']]
data['total_electricity_consumption'] = data['Goa'] + data['Gujarat'] + \
    data['Maharashtra'] + data['DNH'] + data['Rajasthan']
y = data['total_electricity_consumption']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
data
```

Out[ ]:

	Goa	Gujarat	Maharashtra	DNH	Rajasthan	total_electricity_consumption
0	12.8	319.5	428.6	18.6	234.1	1013.6
1	13.7	316.7	419.6	18.2	240.2	1008.4
2	12.6	301.9	395.8	16.7	239.8	966.8
3	13.0	313.2	411.1	17.6	239.1	994.0
4	12.9	320.7	408.6	18.6	240.4	1001.2
...	...	...	...	...	...	...
454	12.3	312.0	421.6	18.2	219.8	983.9
455	8.8	289.7	369.2	17.4	203.2	888.3
456	8.8	297.9	395.5	18.3	212.6	933.1
457	8.8	296.8	395.5	18.7	225.5	945.3
458	10.2	288.7	399.6	18.5	225.7	942.7

459 rows × 6 columns

In [ ]:

```
# Check if model needs retraining
if not os.path.exists(model_filename):
    # Train the new model
    model.fit(X_train, y_train, epochs=1000, validation_data=(X_test, y_test))
else:
    # Retrain the existing model
    model.fit(X_train, y_train, epochs=500, validation_data=(X_test, y_test))
```

```
Epoch 1/500
12/12 [==============================] - 0s 15ms/step - loss: 0.0494 - mae: 0.1952 -
val_loss: 0.0156 - val_mae: 0.1209
Epoch 2/500
12/12 [==============================] - 0s 6ms/step - loss: 0.0197 - mae: 0.1256 -
val_loss: 0.0355 - val_mae: 0.1863
Epoch 3/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0445 - mae: 0.1813 -
val_loss: 0.0566 - val_mae: 0.2364
Epoch 4/500
12/12 [==============================] - 0s 5ms/step - loss: 0.0570 - mae: 0.2114 -
val_loss: 0.0591 - val_mae: 0.2418
Epoch 5/500
12/12 [==============================] - 0s 3ms/step - loss: 0.0203 - mae: 0.1156 -
val_loss: 0.0017 - val_mae: 0.0313
Epoch 6/500
12/12 [==============================] - 0s 5ms/step - loss: 0.0086 - mae: 0.0795 -
val_loss: 0.0179 - val_mae: 0.1260
Epoch 7/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0164 - mae: 0.1103 -
val_loss: 0.0382 - val_mae: 0.1888
Epoch 8/500
12/12 [==============================] - 0s 3ms/step - loss: 0.0221 - mae: 0.1268 -
val_loss: 0.0113 - val_mae: 0.0976
Epoch 9/500
12/12 [==============================] - 0s 5ms/step - loss: 0.0287 - mae: 0.1481 -
val_loss: 0.0016 - val_mae: 0.0307
Epoch 10/500
12/12 [==============================] - 0s 5ms/step - loss: 0.0092 - mae: 0.0850 -
val_loss: 0.0076 - val_mae: 0.0808
Epoch 11/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0396 - mae: 0.1672 -
val_loss: 0.1983 - val_mae: 0.4446
Epoch 12/500
12/12 [==============================] - 0s 3ms/step - loss: 0.1455 - mae: 0.3528 -
val_loss: 0.0141 - val_mae: 0.1102
Epoch 13/500
12/12 [==============================] - 0s 5ms/step - loss: 0.0133 - mae: 0.1051 -
val_loss: 0.0192 - val_mae: 0.1353
Epoch 14/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0109 - mae: 0.0922 -
val_loss: 0.0348 - val_mae: 0.1797
Epoch 15/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0472 - mae: 0.1876 -
val_loss: 0.0685 - val_mae: 0.2606
Epoch 16/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0554 - mae: 0.1994 -
val_loss: 0.0079 - val_mae: 0.0791
Epoch 17/500
12/12 [==============================] - 0s 5ms/step - loss: 0.1114 - mae: 0.2986 -
val_loss: 0.1108 - val_mae: 0.3272
Epoch 18/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0314 - mae: 0.1439 -
val_loss: 0.0256 - val_mae: 0.1523
Epoch 19/500
12/12 [==============================] - 0s 5ms/step - loss: 0.3044 - mae: 0.4046 -
```

```
12/12 [==============================] - 0s 4ms/step - loss: 0.0549 - mae: 0.2074 -
val_loss: 0.0936 - val_mae: 0.3056
Epoch 487/500
12/12 [==============================] - 0s 3ms/step - loss: 0.0929 - mae: 0.2797 -
val_loss: 0.0195 - val_mae: 0.1345
Epoch 488/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0799 - mae: 0.2521 -
val_loss: 0.0173 - val_mae: 0.1301
Epoch 489/500
12/12 [==============================] - 0s 3ms/step - loss: 0.0306 - mae: 0.1606 -
val_loss: 8.0297e-04 - val_mae: 0.0214
Epoch 490/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0598 - mae: 0.2194 -
val_loss: 0.1484 - val_mae: 0.3849
Epoch 491/500
12/12 [==============================] - 0s 4ms/step - loss: 0.4746 - mae: 0.6553 -
val_loss: 0.0057 - val_mae: 0.0685
Epoch 492/500
12/12 [==============================] - 0s 3ms/step - loss: 0.2939 - mae: 0.5160 -
val_loss: 0.3419 - val_mae: 0.5805
Epoch 493/500
12/12 [==============================] - 0s 4ms/step - loss: 0.1199 - mae: 0.3063 -
val_loss: 0.1760 - val_mae: 0.4187
Epoch 494/500
12/12 [==============================] - 0s 3ms/step - loss: 0.0658 - mae: 0.2039 -
val_loss: 0.0066 - val_mae: 0.0770
Epoch 495/500
12/12 [==============================] - 0s 4ms/step - loss: 0.0295 - mae: 0.1526 -
val_loss: 0.2056 - val_mae: 0.4528
Epoch 496/500
12/12 [==============================] - 0s 5ms/step - loss: 1.4286 - mae: 1.1228 -
val_loss: 0.7272 - val_mae: 0.8513
Epoch 497/500
12/12 [==============================] - 0s 3ms/step - loss: 0.3831 - mae: 0.5283 -
val_loss: 0.3339 - val_mae: 0.5766
Epoch 498/500
12/12 [==============================] - 0s 2ms/step - loss: 0.3799 - mae: 0.5406 -
val_loss: 0.3698 - val_mae: 0.6042
Epoch 499/500
12/12 [==============================] - 0s 3ms/step - loss: 0.1610 - mae: 0.3494 -
val_loss: 0.0524 - val_mae: 0.2279
Epoch 500/500
12/12 [==============================] - 0s 3ms/step - loss: 0.0187 - mae: 0.1240 -
val_loss: 0.0017 - val_mae: 0.0362
```

In [ ]:
```
model.save(model_filename)
print("Model saved.")
```

```
Model saved.
c:\Users\rohit\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\e
ngine\training.py:3000: UserWarning: You are saving your model as an HDF5 file via `
model.save()`. This file format is considered legacy. We recommend using instead the
native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

In [ ]:
```
last_5_rows = data.tail(5)
X_test = last_5_rows[['Goa', 'Gujarat', 'Maharashtra', 'DNH', 'Rajasthan']]

y_true = last_5_rows['total_electricity_consumption']

# Use the model to predict electricity consumption for the last 5 rows
y_pred = model.predict(X_test)

for i in range(5):
    print(f"True Consumption: {y_true.iloc[i]} kWh")
    print(f"Predicted Consumption: {y_pred[i][0]} kWh")
    print()
```

```
1/1 [==============================] - 0s 49ms/step
True Consumption: 983.9000000000001 kWh
Predicted Consumption: 983.869384765625 kWh

True Consumption: 888.3 kWh
Predicted Consumption: 888.3291625976562 kWh

True Consumption: 933.1 kWh
Predicted Consumption: 933.113525390625 kWh

True Consumption: 945.3000000000001 kWh
Predicted Consumption: 945.2976684570312 kWh

True Consumption: 942.7 kWh
Predicted Consumption: 942.672607421875 kWh
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_true, y_pred)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_true, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

```
Mean Absolute Error (MAE): 0.02
Mean Squared Error (MSE): 0.00
Root Mean Squared Error (RMSE): 0.02
```