```
!wget https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
!unzip jena_climate_2009_2016.csv.zip
```

```
--2024-04-07 21:25:13--  https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.56.224, 54.231.200.48, 52.216.24.54, ...
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.56.224|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13565642 (13M) [application/zip]
Saving to: ‘jena_climate_2009_2016.csv.zip’

jena_climate_2009_2 100%[===================>]  12.94M  23.6MB/s    in 0.5s

2024-04-07 21:25:14 (23.6 MB/s) - ‘jena_climate_2009_2016.csv.zip’ saved [13565642/13565642]

Archive:  jena_climate_2009_2016.csv.zip
  inflating: jena_climate_2009_2016.csv
  inflating: __MACOSX/._jena_climate_2009_2016.csv
```

Double-click (or enter) to edit

Data Downloading and Preparation: First, the notebook downloads and unzips the "Jena climate dataset."

```
import os
fname = os.path.join("jena_climate_2009_2016.csv")

with open(fname) as f:
    data = f.read()

lines = data.split("\n")
header = lines[0].split(",")
lines = lines[1:]
print(header)
print(len(lines))
```

```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh (%)"', '"VPmax (mbar)"', '"VPact (mbar)"
420451
```

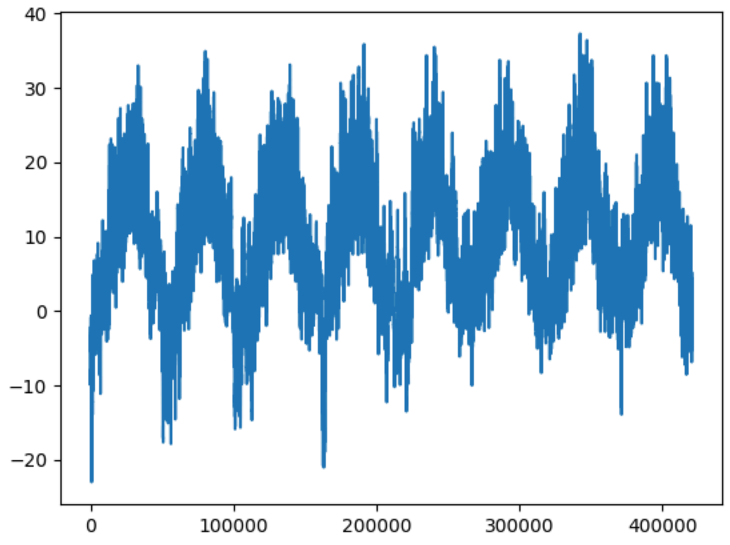
Data Inspection and Parsing: It reads the header and the number of lines to inspect the dataset. After that, the data is parsed into raw data and temperature values.

```
import numpy as np
temperature = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:]]
    temperature[i] = values[1]
    raw_data[i, :] = values[:]
```

Data Visualisation: The temperature time series is displayed.

```
from matplotlib import pyplot as plt
plt.plot(range(len(temperature)), temperature)
```

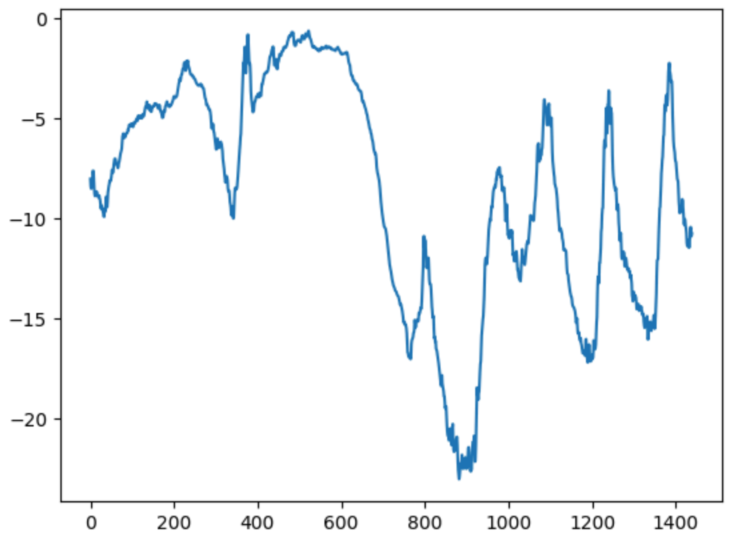
```
[<matplotlib.lines.Line2D at 0x7da35a72ebc0>]
```



Data Preparation: Training, Validation, and Test Sets are created from the data. By deducting the mean and dividing by the standard deviation, the data are normalised.

```
plt.plot(range(1440), temperature[:1440])
```

```
[<matplotlib.lines.Line2D at 0x7da35914f880>]
```



Creating Datasets: The TensorFlow timeseries_dataset_from_array function is used to create datasets.

```
num_train_samples = int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - num_train_samples - num_val_samples
print("num_train_samples:", num_train_samples)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)
```

```
num_train_samples: 210225
num_val_samples: 105112
num_test_samples: 105114
```

Baseline Model: It computes a baseline Mean Absolute Error (MAE) using a simple naive approach.

```
mean = raw_data[:num_train_samples].mean(axis=0)
raw_data -= mean
std = raw_data[:num_train_samples].std(axis=0)
raw_data /= std
```

Creating Datasets: The TensorFlow timeseries_dataset_from_array function is used to create datasets.

```
import numpy as np
from tensorflow import keras
int_sequence = np.arange(10)
dummy_dataset = keras.utils.timeseries_dataset_from_array(
    data=int_sequence[:-3],
    targets=int_sequence[3:],
    sequence_length=3,
    batch_size=2,
)

for inputs, targets in dummy_dataset:
    for i in range(inputs.shape[0]):
        print([int(x) for x in inputs[i]], int(targets[i]))
```

```
[0, 1, 2] 3
[1, 2, 3] 4
[2, 3, 4] 5
[3, 4, 5] 6
[4, 5, 6] 7
```

1D Convolutional Model: A 1D convolutional neural network model is trained and assessed.

```
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256

train_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
    end_index=num_train_samples)

val_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)

test_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)


for samples, targets in train_dataset:
    print("samples shape:", samples.shape)
    print("targets shape:", targets.shape)
    break
```

```
samples shape: (256, 120, 14)
targets shape: (256,)
```

Models of Recurrent Neural Networks (RNNs): It trains and assesses a range of RNN models, including a basic model based on LSTM.

```
def evaluate_naive_method(dataset):
    total_abs_err = 0.
    samples_seen = 0
    for samples, targets in dataset:
        preds = samples[:, -1, 1] * std[1] + mean[1]
        total_abs_err += np.sum(np.abs(preds - targets))
        samples_seen += samples.shape[0]
    return total_abs_err / samples_seen

print(f"Validation MAE: {evaluate_naive_method(val_dataset):.2f}")
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}")
```

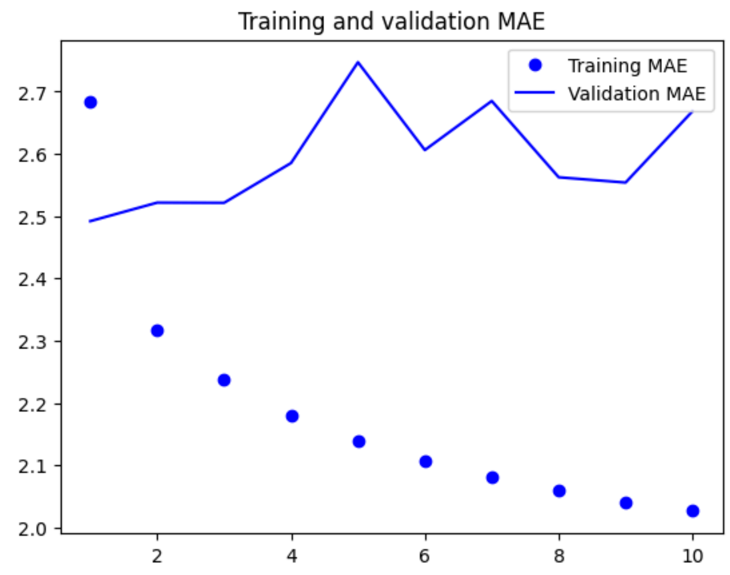
```
Validation MAE: 2.44
Test MAE: 2.62
```

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Flatten()(inputs)
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_dense.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 51s 61ms/step - loss: 12.0572 - mae: 2.6841 - val_loss: 9.9675 - val_mae: 2.4
Epoch 2/10
819/819 [==============================] - 51s 62ms/step - loss: 8.6757 - mae: 2.3161 - val_loss: 10.2533 - val_mae: 2.5
Epoch 3/10
819/819 [==============================] - 51s 62ms/step - loss: 8.0868 - mae: 2.2367 - val_loss: 10.1872 - val_mae: 2.5
Epoch 4/10
819/819 [==============================] - 44s 53ms/step - loss: 7.6622 - mae: 2.1796 - val_loss: 10.6972 - val_mae: 2.5
Epoch 5/10
819/819 [==============================] - 44s 53ms/step - loss: 7.3817 - mae: 2.1389 - val_loss: 11.9682 - val_mae: 2.7
Epoch 6/10
819/819 [==============================] - 43s 52ms/step - loss: 7.1584 - mae: 2.1066 - val_loss: 10.8210 - val_mae: 2.6
Epoch 7/10
819/819 [==============================] - 44s 53ms/step - loss: 6.9831 - mae: 2.0818 - val_loss: 11.4653 - val_mae: 2.6
Epoch 8/10
819/819 [==============================] - 42s 51ms/step - loss: 6.8343 - mae: 2.0595 - val_loss: 10.5323 - val_mae: 2.5
Epoch 9/10
819/819 [==============================] - 51s 62ms/step - loss: 6.7153 - mae: 2.0406 - val_loss: 10.4677 - val_mae: 2.5
Epoch 10/10
819/819 [==============================] - 43s 53ms/step - loss: 6.6164 - mae: 2.0264 - val_loss: 11.3226 - val_mae: 2.6
405/405 [==============================] - 14s 34ms/step - loss: 10.8573 - mae: 2.5893
Test MAE: 2.59
```

RNN comprehension: It offers explanations and code samples, along with an RNN implementation in NumPy and Keras layers.

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, "bo", label="Training MAE")
plt.plot(epochs, val_loss, "b", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.show()
```



Advanced RNN Usage: It delves into more complex RNN methods like bidirectional RNNs, stacked RNN layers, and dropout regularisation.

```
Start coding or generate with AI.
```

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_lstm.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 116s 138ms/step - loss: 39.9562 - mae: 4.5958 - val_loss: 11.9788 - val_mae:
Epoch 2/10
819/819 [==============================] - 91s 111ms/step - loss: 10.9919 - mae: 2.5666 - val_loss: 9.3321 - val_mae: 2.
Epoch 3/10
819/819 [==============================] - 113s 137ms/step - loss: 9.7797 - mae: 2.4322 - val_loss: 9.2785 - val_mae: 2.
Epoch 4/10
819/819 [==============================] - 92s 112ms/step - loss: 9.2800 - mae: 2.3722 - val_loss: 9.2809 - val_mae: 2.3
Epoch 5/10
819/819 [==============================] - 112s 136ms/step - loss: 9.0465 - mae: 2.3413 - val_loss: 9.2961 - val_mae: 2.
Epoch 6/10
819/819 [==============================] - 111s 136ms/step - loss: 8.8246 - mae: 2.3124 - val_loss: 9.4033 - val_mae: 2.
Epoch 7/10
819/819 [==============================] - 92s 112ms/step - loss: 8.6207 - mae: 2.2856 - val_loss: 9.3545 - val_mae: 2.3
Epoch 8/10
819/819 [==============================] - 111s 135ms/step - loss: 8.3901 - mae: 2.2543 - val_loss: 9.4326 - val_mae: 2.
Epoch 9/10
819/819 [==============================] - 91s 111ms/step - loss: 8.2394 - mae: 2.2328 - val_loss: 9.3929 - val_mae: 2.4
Epoch 10/10
819/819 [==============================] - 110s 134ms/step - loss: 8.0805 - mae: 2.2102 - val_loss: 9.5689 - val_mae: 2.
405/405 [==============================] - 23s 54ms/step - loss: 10.9062 - mae: 2.5794
Test MAE: 2.58
```

```
import numpy as np
timesteps = 100
input_features = 32
output_features = 64
inputs = np.random.random((timesteps, input_features))
state_t = np.zeros((output_features,))
W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))
successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
    successive_outputs.append(output_t)
    state_t = output_t
final_output_sequence = np.stack(successive_outputs, axis=0)
```

```
num_features = 14
inputs = keras.Input(shape=(None, num_features))
outputs = layers.SimpleRNN(16)(inputs)
```

```
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=False)(inputs)
print(outputs.shape)
```

```
    (None, 16)
```

```
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=True)(inputs)
print(outputs.shape)
```

```
    (None, 120, 16)
```

```
inputs = keras.Input(shape=(steps, num_features))
x = layers.SimpleRNN(16, return_sequences=True)(inputs)
x = layers.SimpleRNN(16, return_sequences=True)(x)
outputs = layers.SimpleRNN(16)(x)
```

Advanced use of recurrent neural networks Using recurrent dropout to fight overfitting Training and evaluating a dropout-regularized LSTM

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(32, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=50,
                    validation_data=val_dataset,
                    callbacks=callbacks)
    819/819 [==============================] - 196s 239ms/step - loss: 9.3399 - mae: 2.3747 - val_loss: 10.2115 - val_mae: 2
    Epoch 32/50
    819/819 [==============================] - 197s 240ms/step - loss: 9.2965 - mae: 2.3686 - val_loss: 10.2365 - val_mae: 2
    Epoch 33/50
    819/819 [==============================] - 196s 239ms/step - loss: 9.2605 - mae: 2.3628 - val_loss: 10.1643 - val_mae: 2
    Epoch 34/50
    819/819 [==============================] - 179s 218ms/step - loss: 9.2133 - mae: 2.3570 - val_loss: 10.0563 - val_mae: 2
    Epoch 35/50
    819/819 [==============================] - 197s 241ms/step - loss: 9.1842 - mae: 2.3507 - val_loss: 10.1230 - val_mae: 2
    Epoch 36/50
    819/819 [==============================] - 198s 242ms/step - loss: 9.1372 - mae: 2.3471 - val_loss: 10.2373 - val_mae: 2
    Epoch 37/50
    819/819 [==============================] - 182s 222ms/step - loss: 9.0035 - mae: 2.3301 - val_loss: 10.5679 - val_mae: 2
    Epoch 38/50
    819/819 [==============================] - 195s 238ms/step - loss: 9.0445 - mae: 2.3355 - val_loss: 10.0386 - val_mae: 2
    Epoch 39/50
    819/819 [==============================] - 194s 237ms/step - loss: 9.0090 - mae: 2.3276 - val_loss: 10.1474 - val_mae: 2
    Epoch 40/50
    819/819 [==============================] - 180s 219ms/step - loss: 8.9628 - mae: 2.3218 - val_loss: 10.6862 - val_mae: 2
    Epoch 41/50
    819/819 [==============================] - 182s 221ms/step - loss: 8.9527 - mae: 2.3223 - val_loss: 10.4236 - val_mae: 2
    Epoch 42/50
    819/819 [==============================] - 180s 219ms/step - loss: 8.8810 - mae: 2.3137 - val_loss: 10.5406 - val_mae: 2
    Epoch 43/50
    819/819 [==============================] - 180s 220ms/step - loss: 8.8508 - mae: 2.3097 - val_loss: 10.3268 - val_mae: 2
    Epoch 44/50
    819/819 [==============================] - 181s 221ms/step - loss: 8.8602 - mae: 2.3075 - val_loss: 10.6044 - val_mae: 2
    Epoch 45/50
    819/819 [==============================] - 180s 219ms/step - loss: 8.8116 - mae: 2.3005 - val_loss: 10.5876 - val_mae: 2
    Epoch 46/50
    819/819 [==============================] - 196s 239ms/step - loss: 8.8098 - mae: 2.3004 - val_loss: 10.6591 - val_mae: 2
    Epoch 47/50
    819/819 [==============================] - 196s 239ms/step - loss: 8.7721 - mae: 2.2959 - val_loss: 10.5801 - val_mae: 2
    Epoch 48/50
    819/819 [==============================] - 196s 240ms/step - loss: 8.7144 - mae: 2.2876 - val_loss: 10.4753 - val_mae: 2
    Epoch 49/50
    819/819 [==============================] - 197s 240ms/step - loss: 8.6984 - mae: 2.2871 - val_loss: 10.7922 - val_mae: 2
    Epoch 50/50
    819/819 [==============================] - 196s 239ms/step - loss: 8.6723 - mae: 2.2845 - val_loss: 10.7449 - val_mae: 2
```

```python
inputs = keras.Input(shape=(sequence_length, num_features))
x = layers.LSTM(32, recurrent_dropout=0.2, unroll=True)(inputs)
```

```python
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=50,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_stacked_gru_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
    Epoch 1/50
    819/819 [==============================] - 319s 376ms/step - loss: 24.9305 - mae: 3.6921 - val_loss: 9.5983 - val_mae: 2
    Epoch 2/50
    819/819 [==============================] - 306s 373ms/step - loss: 13.9984 - mae: 2.8971 - val_loss: 9.3122 - val_mae: 2
    Epoch 3/50
    819/819 [==============================] - 299s 365ms/step - loss: 13.1369 - mae: 2.8071 - val_loss: 8.9947 - val_mae: 2
    Epoch 4/50
    819/819 [==============================] - 325s 396ms/step - loss: 12.5602 - mae: 2.7474 - val_loss: 8.7998 - val_mae: 2
    Epoch 5/50
    819/819 [==============================] - 316s 386ms/step - loss: 12.0884 - mae: 2.6965 - val_loss: 8.9928 - val_mae: 2
    Epoch 6/50
    819/819 [==============================] - 305s 372ms/step - loss: 11.6980 - mae: 2.6536 - val_loss: 8.5860 - val_mae: 2
    Epoch 7/50
    819/819 [==============================] - 304s 371ms/step - loss: 11.2997 - mae: 2.6084 - val_loss: 8.9626 - val_mae: 2
    Epoch 8/50
    819/819 [==============================] - 299s 365ms/step - loss: 10.9309 - mae: 2.5642 - val_loss: 9.2563 - val_mae: 2
    Epoch 9/50
    819/819 [==============================] - 306s 374ms/step - loss: 10.6253 - mae: 2.5305 - val_loss: 9.0249 - val_mae: 2
    Epoch 10/50
    819/819 [==============================] - 306s 373ms/step - loss: 10.3121 - mae: 2.4928 - val_loss: 9.1574 - val_mae: 2
    Epoch 11/50
    819/819 [==============================] - 299s 364ms/step - loss: 10.0378 - mae: 2.4594 - val_loss: 9.8402 - val_mae: 2
    Epoch 12/50
    819/819 [==============================] - 300s 366ms/step - loss: 9.7743 - mae: 2.4265 - val_loss: 9.7600 - val_mae: 2.
    Epoch 13/50
```

```
819/819 [==============================] - 299s 365ms/step - loss: 9.5212 - mae: 2.3961 - val_loss: 9.6662 - val_mae: 2.
Epoch 14/50
819/819 [==============================] - 305s 372ms/step - loss: 9.3449 - mae: 2.3743 - val_loss: 10.0826 - val_mae: 2
Epoch 15/50
819/819 [==============================] - 305s 372ms/step - loss: 9.1847 - mae: 2.3538 - val_loss: 10.0355 - val_mae: 2
Epoch 16/50
819/819 [==============================] - 305s 372ms/step - loss: 9.0420 - mae: 2.3351 - val_loss: 10.3692 - val_mae: 2
Epoch 17/50
819/819 [==============================] - 301s 367ms/step - loss: 8.9020 - mae: 2.3178 - val_loss: 10.5940 - val_mae: 2
Epoch 18/50
819/819 [==============================] - 301s 367ms/step - loss: 8.7965 - mae: 2.3033 - val_loss: 10.6666 - val_mae: 2
Epoch 19/50
819/819 [==============================] - 305s 372ms/step - loss: 8.6430 - mae: 2.2853 - val_loss: 10.7138 - val_mae: 2
Epoch 20/50
819/819 [==============================] - 305s 371ms/step - loss: 8.5292 - mae: 2.2716 - val_loss: 11.1058 - val_mae: 2
Epoch 21/50
819/819 [==============================] - 305s 372ms/step - loss: 8.4270 - mae: 2.2573 - val_loss: 11.2249 - val_mae: 2
Epoch 22/50
819/819 [==============================] - 307s 374ms/step - loss: 8.3194 - mae: 2.2435 - val_loss: 10.7126 - val_mae: 2
Epoch 23/50
819/819 [==============================] - 305s 372ms/step - loss: 8.2584 - mae: 2.2323 - val_loss: 10.9685 - val_mae: 2
Epoch 24/50
819/819 [==============================] - 300s 367ms/step - loss: 8.2131 - mae: 2.2257 - val_loss: 11.4196 - val_mae: 2
Epoch 25/50
819/819 [==============================] - 307s 374ms/step - loss: 8.0901 - mae: 2.2105 - val_loss: 11.0274 - val_mae: 2
Epoch 26/50
819/819 [==============================] - 304s 372ms/step - loss: 8.0504 - mae: 2.2054 - val_loss: 10.9589 - val_mae: 2
Epoch 27/50
819/819 [==============================] - 300s 366ms/step - loss: 7.9842 - mae: 2.1938 - val_loss: 11.5435 - val_mae: 2
Epoch 28/50
```