# CPE 313 : Advanced Machine Learning with Deep Learning

## HOA 1.1 Using Tensorflow with a Real Dataset

- Name: Corpuz, Micki Laurren

---

# Linear Regression with a Real Dataset

This Colab uses a real dataset to predict the prices of houses in California.

## Learning Objectives:

After doing this Colab, you'll know how to do the following:

- Read a .csv file into a pandas DataFrame.
- Examine a dataset.
- Experiment with different features in building a model.
- Tune the model's hyperparameters.

## The Dataset

The dataset for this exercise is based on 1990 census data from California. The dataset is old but still provides a great opportunity to learn about machine learning programming.

## Import relevant modules

The following hidden code cell imports the necessary code to run the code in the rest of this Colaboratory.

```
#@title Import relevant modules
import pandas as pd
import tensorflow as tf
from matplotlib import pyplot as plt

# The following lines adjust the granularity of reporting.
pd.options.display.max_rows = 10
pd.options.display.float_format = "{:.1f}".format
```

## The dataset

Datasets are often stored on disk or at a URL in .csv format.

A well-formed .csv file contains column names in the first row, followed by many rows of data. A comma divides each value in each row. For example, here are the first five rows of the .csv file holding the California Housing Dataset:

```
"longitude","latitude","housing_median_age","total_rooms","total_bedro
oms","population","households","median_income","median_house_value"
-
114.310000,34.190000,15.000000,5612.000000,1283.000000,1015.000000,472
.000000,1.493600,66900.000000
-
114.470000,34.400000,19.000000,7650.000000,1901.000000,1129.000000,463
.000000,1.820000,80100.000000
-
114.560000,33.690000,17.000000,720.000000,174.000000,333.000000,117.00
0000,1.650900,85700.000000
-
114.570000,33.640000,14.000000,1501.000000,337.000000,515.000000,226.0
00000,3.191700,73400.000000
```

## Load the .csv file into a pandas DataFrame

This Colab, like many machine learning programs, gathers the .csv file and stores the data in memory as a pandas Dataframe. Pandas is an open source Python library. The primary datatype in pandas is a DataFrame. You can imagine a pandas DataFrame as a spreadsheet in which each row is identified by a number and each column by a name. Pandas is itself built on another open source Python library called NumPy. If you aren't familiar with these technologies, please view these two quick tutorials:

- NumPy
- Pandas DataFrames

The following code cell imports the .csv file into a pandas DataFrame and scales the values in the label (`median_house_value`):

```
# Import the dataset.
training_df =
pd.read_csv(filepath_or_buffer="https://download.mlcc.google.com/mledu
-datasets/california_housing_train.csv")

# Scale the label.
training_df["median_house_value"] /= 1000.0

# Print the first rows of the pandas DataFrame.
training_df.head()
```

```
   longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
0     -114.3      34.2                15.0       5612.0
1283.0
1     -114.5      34.4                19.0       7650.0
```

```
1901.0
2     -114.6      33.7                17.0         720.0
174.0
3     -114.6      33.6                14.0        1501.0
337.0
4     -114.6      33.6                20.0        1454.0
326.0

   population  households  median_income  median_house_value
0      1015.0       472.0            1.5                66.9
1      1129.0       463.0            1.8                80.1
2       333.0       117.0            1.7                85.7
3       515.0       226.0            3.2                73.4
4       624.0       262.0            1.9                65.5
```

Scaling `median_house_value` puts the value of each house in units of thousands. Scaling will keep loss values and learning rates in a friendlier range.

Although scaling a label is usually *not* essential, scaling features in a multi-feature model usually *is* essential.

## Examine the dataset

A large part of most machine learning projects is getting to know your data. The pandas API provides a `describe` function that outputs the following statistics about every column in the DataFrame:

- `count`, which is the number of rows in that column. Ideally, `count` contains the same value for every column.
- `mean` and `std`, which contain the mean and standard deviation of the values in each column.
- `min` and `max`, which contain the lowest and highest values in each column.
- `25%`, `50%`, `75%`, which contain various quantiles.

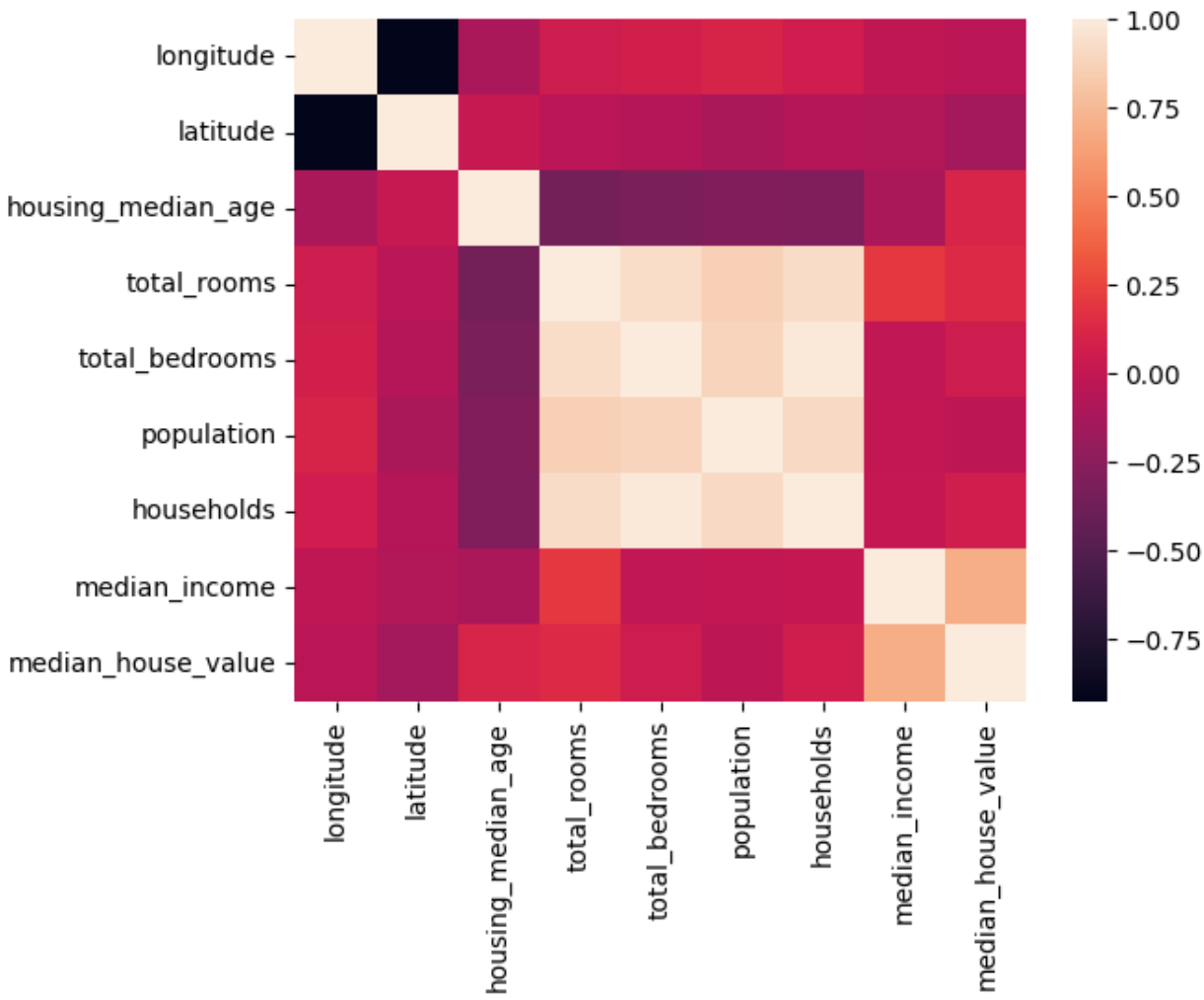
```
# Get statistics on the dataset.
training_df.describe()

       longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
count    17000.0   17000.0             17000.0      17000.0
17000.0
mean      -119.6      35.6                28.6       2643.7
539.4
std          2.0       2.1                12.6       2179.9
421.5
min       -124.3      32.5                 1.0          2.0
1.0
```

```
25%       -121.8      33.9                 18.0       1462.0
297.0
50%       -118.5      34.2                 29.0       2127.0
434.0
75%       -118.0      37.7                 37.0       3151.2
648.2
max       -114.3      42.0                 52.0      37937.0
6445.0
```

| | population | households | median_income | median_house_value |
|---|---|---|---|---|
| count | 17000.0 | 17000.0 | 17000.0 | 17000.0 |
| mean | 1429.6 | 501.2 | 3.9 | 207.3 |
| std | 1147.9 | 384.5 | 1.9 | 116.0 |
| min | 3.0 | 1.0 | 0.5 | 15.0 |
| 25% | 790.0 | 282.0 | 2.6 | 119.4 |
| 50% | 1167.0 | 409.0 | 3.5 | 180.4 |
| 75% | 1721.0 | 605.2 | 4.8 | 265.0 |
| max | 35682.0 | 6082.0 | 15.0 | 500.0 |

```
import seaborn as sns

corr = training_df.corr()

sns.heatmap(corr)
```

```
<Axes: >
```




```
corr

                    longitude  latitude  housing_median_age
total_rooms  \
longitude                 1.0      -0.9                -0.1
0.0
latitude                 -0.9       1.0                 0.0          -
0.0
housing_median_age       -0.1       0.0                 1.0          -
0.4
total_rooms               0.0      -0.0                -0.4
1.0
total_bedrooms            0.1      -0.1                -0.3
0.9
population                0.1      -0.1                -0.3
0.9
households                0.1      -0.1                -0.3
0.9
median_income            -0.0      -0.1                -0.1
```

```
0.2
median_house_value       -0.0      -0.1                 0.1
0.1

                     total_bedrooms  population  households
median_income  \
longitude                       0.1         0.1         0.1           -
0.0
latitude                       -0.1        -0.1        -0.1           -
0.1
housing_median_age             -0.3        -0.3        -0.3           -
0.1
total_rooms                     0.9         0.9         0.9
0.2
total_bedrooms                  1.0         0.9         1.0           -
0.0
population                      0.9         1.0         0.9           -
0.0
households                      1.0         0.9         1.0
0.0
median_income                  -0.0        -0.0         0.0
1.0
median_house_value              0.0        -0.0         0.1
0.7

                     median_house_value
longitude                          -0.0
latitude                           -0.1
housing_median_age                  0.1
total_rooms                         0.1
total_bedrooms                      0.0
population                         -0.0
households                          0.1
median_income                       0.7
median_house_value                  1.0
```

## Task 1: Identify anomalies in the dataset

Do you see any anomalies (strange values) in the data?

> Some strange values stand out, especially in total_rooms, total_bedrooms, population, and households, where the maximum values are much higher than what most of the data shows. Median_income also has a few unusually high values compared to the rest. On the other hand, longitude, latitude, and housing_median_age look fairly normal and don't show extreme outliers.

# Define functions that build and train a model

The following code defines two functions:

- `build_model(my_learning_rate)`, which builds a randomly-initialized model.
- `train_model(model, feature, label, epochs)`, which trains the model from the examples (feature and label) you pass.

Since you don't need to understand model building code right now, we've hidden this code cell. You may optionally double-click the following headline to see the code that builds and trains a model.

```
#@title Define the functions that build and train a model
def build_model(my_learning_rate):
  """Create and compile a simple linear regression model."""
  # Most simple tf.keras models are sequential.
  model = tf.keras.models.Sequential()

  # Describe the topography of the model.
  # The topography of a simple linear regression model
  # is a single node in a single layer.
  model.add(tf.keras.layers.Dense(units=1,
                                  input_shape=(1,)))

  # Compile the model topography into code that TensorFlow can
efficiently
  # execute. Configure training to minimize the model's mean squared
error.

model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=my_l
earning_rate),
                loss="mean_squared_error",
                metrics=[tf.keras.metrics.RootMeanSquaredError()])

  return model


def train_model(model, df, feature, label, epochs, batch_size):
  """Train the model by feeding it data."""

  # Feed the model the feature and the label.
  # The model will train for the specified number of epochs.
  history = model.fit(x=df[feature],
                      y=df[label],
                      batch_size=batch_size,
                      epochs=epochs)

  # Gather the trained model's weight and bias.
  trained_weight = model.get_weights()[0]
  trained_bias = model.get_weights()[1]

  # The list of epochs is stored separately from the rest of history.
  epochs = history.epoch
```

```
  # Isolate the error for each epoch.
  hist = pd.DataFrame(history.history)

  # To track the progression of training, we're going to take a
snapshot
  # of the model's root mean squared error at each epoch.
  rmse = hist["root_mean_squared_error"]

  return trained_weight, trained_bias, epochs, rmse

print("Defined the build_model and train_model functions.")
```

```
Defined the build_model and train_model functions.
```

## Define plotting functions

The following matplotlib functions create the following plots:

- a scatter plot of the feature vs. the label, and a line showing the output of the trained model
- a loss curve

You may optionally double-click the headline to see the matplotlib code, but note that writing matplotlib code is not an important part of learning ML programming.

```
#@title Define the plotting functions
def plot_the_model(trained_weight, trained_bias, feature, label):
  """Plot the trained model against 200 random training examples."""

  # Label the axes.
  plt.xlabel(feature)
  plt.ylabel(label)

  # Create a scatter plot from 200 random points of the dataset.
  random_examples = training_df.sample(n=200)
  plt.scatter(random_examples[feature], random_examples[label])

  # Weight and bias are scalars
  w = float(trained_weight)
  b = float(trained_bias)

  # Create a red line representing the model. The red line starts
  # at coordinates (x0, y0) and ends at coordinates (x1, y1).
  x0 = 0
  y0 = b
  x1 = random_examples[feature].max()
  y1 = b + (w * x1)
  plt.plot([x0, x1], [y0, y1], c='r')

  # Render the scatter plot and the red line.
```

```
  plt.show()


def plot_the_loss_curve(epochs, rmse):
  """Plot a curve of loss vs. epoch."""

  plt.figure()
  plt.xlabel("Epoch")
  plt.ylabel("Root Mean Squared Error")

  plt.plot(epochs, rmse, label="Loss")
  plt.legend()
  plt.ylim([rmse.min()*0.97, rmse.max()])
  plt.show()

print("Defined the plot_the_model and plot_the_loss_curve functions.")
```

```
Defined the plot_the_model and plot_the_loss_curve functions.
```

## Call the model functions

An important part of machine learning is determining which features correlate with the label. For example, real-life home-value prediction models typically rely on hundreds of features and synthetic features. However, this model relies on only one feature. For now, you'll arbitrarily use `total_rooms` as that feature.

```
# The following variables are the hyperparameters.
learning_rate = 0.01
epochs = 256
batch_size = 32

# Specify the feature and the label.
my_feature = "total_rooms"  # the total number of rooms on a specific
city block.
my_label="median_house_value" # the median value of a house on a
specific city block.
# That is, you're going to create a model that predicts house value
based
# solely on total_rooms.

# Discard any pre-existing version of the model.
my_model = None

# Invoke the functions.
my_model = build_model(learning_rate)
weight, bias, epochs, rmse = train_model(my_model, training_df,
                                         my_feature, my_label,
                                         epochs, batch_size)
```

```
print("\nThe learned weight for your model is %.4f" % weight)
print("The learned bias for your model is %.4f\n" % bias )

plot_the_model(weight, bias, my_feature, my_label)
plot_the_loss_curve(epochs, rmse)
```

```
Epoch 1/256

c:\Users\micki\anaconda3\envs\tf_env\lib\site-packages\keras\src\
layers\core\dense.py:92: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 836us/step - loss: 29036.0273 -
root_mean_squared_error: 170.3996
Epoch 2/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 888us/step - loss: 27966.2520 -
root_mean_squared_error: 167.2311
Epoch 3/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step - loss: 27141.5586 -
root_mean_squared_error: 164.7469
Epoch 4/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 26505.4355 -
root_mean_squared_error: 162.8049
Epoch 5/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 25793.4277 -
root_mean_squared_error: 160.6033
Epoch 6/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 25039.3828 -
root_mean_squared_error: 158.2384
Epoch 7/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 24430.6758 -
root_mean_squared_error: 156.3031
Epoch 8/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 833us/step - loss: 23885.0234 -
root_mean_squared_error: 154.5478
Epoch 9/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - loss: 23215.8750 -
root_mean_squared_error: 152.3676
Epoch 10/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 22664.4160 -
root_mean_squared_error: 150.5471
Epoch 11/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 914us/step - loss: 22170.8633 -
root_mean_squared_error: 148.8988
Epoch 12/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 21662.6504 -
```

```
root_mean_squared_error: 147.1824
Epoch 13/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 21101.1094 -
root_mean_squared_error: 145.2622
Epoch 14/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 874us/step - loss: 20833.0645 -
root_mean_squared_error: 144.3366
Epoch 15/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 20135.6230 -
root_mean_squared_error: 141.9000
Epoch 16/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 19878.8242 -
root_mean_squared_error: 140.9923
Epoch 17/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 19461.2227 -
root_mean_squared_error: 139.5035
Epoch 18/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 18773.3125 -
root_mean_squared_error: 137.0157
Epoch 19/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 870us/step - loss: 18637.6543 -
root_mean_squared_error: 136.5198
Epoch 20/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 18019.2344 -
root_mean_squared_error: 134.2357
Epoch 21/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 17930.0078 -
root_mean_squared_error: 133.9030
Epoch 22/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 17564.8418 -
root_mean_squared_error: 132.5324
Epoch 23/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 17388.8047 -
root_mean_squared_error: 131.8666
Epoch 24/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 16878.5508 -
root_mean_squared_error: 129.9175
Epoch 25/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 837us/step - loss: 16748.9609 -
root_mean_squared_error: 129.4178
Epoch 26/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 16368.7451 -
root_mean_squared_error: 127.9404
Epoch 27/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 824us/step - loss: 16109.0205 -
root_mean_squared_error: 126.9213
Epoch 28/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - loss: 16010.8799 -
root_mean_squared_error: 126.5341
```

```
Epoch 29/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 15684.5439 -
root_mean_squared_error: 125.2379
Epoch 30/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 855us/step - loss: 15609.9375 -
root_mean_squared_error: 124.9397
Epoch 31/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 15479.0635 -
root_mean_squared_error: 124.4149
Epoch 32/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 15165.0518 -
root_mean_squared_error: 123.1465
Epoch 33/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 14962.0977 -
root_mean_squared_error: 122.3197
Epoch 34/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 14971.7539 -
root_mean_squared_error: 122.3591
Epoch 35/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 805us/step - loss: 14783.5566 -
root_mean_squared_error: 121.5876
Epoch 36/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 14758.0244 -
root_mean_squared_error: 121.4826
Epoch 37/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 14685.6240 -
root_mean_squared_error: 121.1843
Epoch 38/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 14493.5732 -
root_mean_squared_error: 120.3893
Epoch 39/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 14407.0039 -
root_mean_squared_error: 120.0292
Epoch 40/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 14363.4746 -
root_mean_squared_error: 119.8477
Epoch 41/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 14286.6494 -
root_mean_squared_error: 119.5268
Epoch 42/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 14185.5176 -
root_mean_squared_error: 119.1030
Epoch 43/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 14194.3662 -
root_mean_squared_error: 119.1401
Epoch 44/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 14157.8115 -
root_mean_squared_error: 118.9866
Epoch 45/256
```

```
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 853us/step - loss: 14047.2344 -
root_mean_squared_error: 118.5210
Epoch 46/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 14089.3066 -
root_mean_squared_error: 118.6984
Epoch 47/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 905us/step - loss: 13949.0576 -
root_mean_squared_error: 118.1061
Epoch 48/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 13959.9297 -
root_mean_squared_error: 118.1521
Epoch 49/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 13934.0908 -
root_mean_squared_error: 118.0427
Epoch 50/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 835us/step - loss: 13930.3047 -
root_mean_squared_error: 118.0267
Epoch 51/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 13876.6230 -
root_mean_squared_error: 117.7991
Epoch 52/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - loss: 13956.8057 -
root_mean_squared_error: 118.1389
Epoch 53/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 860us/step - loss: 13874.7461 -
root_mean_squared_error: 117.7911
Epoch 54/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 883us/step - loss: 13802.8965 -
root_mean_squared_error: 117.4857
Epoch 55/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - loss: 13857.4648 -
root_mean_squared_error: 117.7177
Epoch 56/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 916us/step - loss: 13843.1855 -
root_mean_squared_error: 117.6571
Epoch 57/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 887us/step - loss: 13865.7910 -
root_mean_squared_error: 117.7531
Epoch 58/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 927us/step - loss: 13825.2578 -
root_mean_squared_error: 117.5809
Epoch 59/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 13779.6318 -
root_mean_squared_error: 117.3867
Epoch 60/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 13837.2432 -
root_mean_squared_error: 117.6318
Epoch 61/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 13716.6367 -
```

```
root_mean_squared_error: 117.1180
Epoch 62/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 13814.3086 -
root_mean_squared_error: 117.5343
Epoch 63/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 13813.8125 -
root_mean_squared_error: 117.5322
Epoch 64/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 773us/step - loss: 13750.4824 -
root_mean_squared_error: 117.2625
Epoch 65/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 13805.3809 -
root_mean_squared_error: 117.4963
Epoch 66/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 13775.7363 -
root_mean_squared_error: 117.3701
Epoch 67/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - loss: 13716.6221 -
root_mean_squared_error: 117.1180
Epoch 68/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 13788.8760 -
root_mean_squared_error: 117.4260
Epoch 69/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 13759.2725 -
root_mean_squared_error: 117.2999
Epoch 70/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 13770.9355 -
root_mean_squared_error: 117.3496
Epoch 71/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - loss: 13725.9131 -
root_mean_squared_error: 117.1576
Epoch 72/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - loss: 13776.0713 -
root_mean_squared_error: 117.3715
Epoch 73/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 13774.4531 -
root_mean_squared_error: 117.3646
Epoch 74/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 13786.8496 -
root_mean_squared_error: 117.4174
Epoch 75/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step - loss: 13763.6074 -
root_mean_squared_error: 117.3184
Epoch 76/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 769us/step - loss: 13814.7578 -
root_mean_squared_error: 117.5362
Epoch 77/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 848us/step - loss: 13761.7832 -
root_mean_squared_error: 117.3106
```

```
Epoch 78/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - loss: 13727.6260 -
root_mean_squared_error: 117.1650
Epoch 79/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 835us/step - loss: 13794.2148 -
root_mean_squared_error: 117.4488
Epoch 80/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 829us/step - loss: 13831.1396 -
root_mean_squared_error: 117.6059
Epoch 81/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 13770.4033 -
root_mean_squared_error: 117.3474
Epoch 82/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 13702.4727 -
root_mean_squared_error: 117.0576
Epoch 83/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 13765.2002 -
root_mean_squared_error: 117.3252
Epoch 84/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 833us/step - loss: 13784.0762 -
root_mean_squared_error: 117.4056
Epoch 85/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 13782.6777 -
root_mean_squared_error: 117.3997
Epoch 86/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 847us/step - loss: 13697.0996 -
root_mean_squared_error: 117.0346
Epoch 87/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 13752.7549 -
root_mean_squared_error: 117.2721
Epoch 88/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 762us/step - loss: 13755.3506 -
root_mean_squared_error: 117.2832
Epoch 89/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 13777.3809 -
root_mean_squared_error: 117.3771
Epoch 90/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 13771.1494 -
root_mean_squared_error: 117.3505
Epoch 91/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - loss: 13744.3457 -
root_mean_squared_error: 117.2363
Epoch 92/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 13771.6055 -
root_mean_squared_error: 117.3525
Epoch 93/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - loss: 13808.8486 -
root_mean_squared_error: 117.5111
Epoch 94/256
```

```
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - loss: 13764.4727 -
root_mean_squared_error: 117.3221
Epoch 95/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 13742.0469 -
root_mean_squared_error: 117.2265
Epoch 96/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 13719.3730 -
root_mean_squared_error: 117.1297
Epoch 97/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 13682.3037 -
root_mean_squared_error: 116.9714
Epoch 98/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 13734.5596 -
root_mean_squared_error: 117.1945
Epoch 99/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 13705.6885 -
root_mean_squared_error: 117.0713
Epoch 100/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 13792.9268 -
root_mean_squared_error: 117.4433
Epoch 101/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 853us/step - loss: 13790.1035 -
root_mean_squared_error: 117.4313
Epoch 102/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 13742.2988 -
root_mean_squared_error: 117.2276
Epoch 103/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 13780.4629 -
root_mean_squared_error: 117.3902
Epoch 104/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 13771.4609 -
root_mean_squared_error: 117.3519
Epoch 105/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - loss: 13755.3076 -
root_mean_squared_error: 117.2830
Epoch 106/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 900us/step - loss: 13760.7432 -
root_mean_squared_error: 117.3062
Epoch 107/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 983us/step - loss: 13776.4971 -
root_mean_squared_error: 117.3733
Epoch 108/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - loss: 13735.2822 -
root_mean_squared_error: 117.1976
Epoch 109/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 13827.3105 -
root_mean_squared_error: 117.5896
Epoch 110/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 13745.3223 -
```

```
root_mean_squared_error: 117.2404
Epoch 111/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 13795.6924 -
root_mean_squared_error: 117.4551
Epoch 112/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 896us/step - loss: 13756.3975 -
root_mean_squared_error: 117.2877
Epoch 113/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 917us/step - loss: 13719.8203 -
root_mean_squared_error: 117.1316
Epoch 114/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 932us/step - loss: 13794.4141 -
root_mean_squared_error: 117.4496
Epoch 115/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 860us/step - loss: 13754.9023 -
root_mean_squared_error: 117.2813
Epoch 116/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - loss: 13787.3213 -
root_mean_squared_error: 117.4194
Epoch 117/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 13767.5908 -
root_mean_squared_error: 117.3354
Epoch 118/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 907us/step - loss: 13693.2891 -
root_mean_squared_error: 117.0183
Epoch 119/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 13704.8369 -
root_mean_squared_error: 117.0677
Epoch 120/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 13781.2148 -
root_mean_squared_error: 117.3934
Epoch 121/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - loss: 13753.1250 -
root_mean_squared_error: 117.2737
Epoch 122/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - loss: 13715.7676 -
root_mean_squared_error: 117.1143
Epoch 123/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 771us/step - loss: 13741.4102 -
root_mean_squared_error: 117.2238
Epoch 124/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 13786.0176 -
root_mean_squared_error: 117.4139
Epoch 125/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 13763.9785 -
root_mean_squared_error: 117.3200
Epoch 126/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 13759.2783 -
root_mean_squared_error: 117.2999
```

```
Epoch 127/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 904us/step - loss: 13710.3389 -
root_mean_squared_error: 117.0912
Epoch 128/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 13770.9678 -
root_mean_squared_error: 117.3498
Epoch 129/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 922us/step - loss: 13742.6104 -
root_mean_squared_error: 117.2289
Epoch 130/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 893us/step - loss: 13725.5498 -
root_mean_squared_error: 117.1561
Epoch 131/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 871us/step - loss: 13785.6992 -
root_mean_squared_error: 117.4125
Epoch 132/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 870us/step - loss: 13744.5371 -
root_mean_squared_error: 117.2371
Epoch 133/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 895us/step - loss: 13710.9277 -
root_mean_squared_error: 117.0937
Epoch 134/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 13805.6758 -
root_mean_squared_error: 117.4976
Epoch 135/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 849us/step - loss: 13708.3232 -
root_mean_squared_error: 117.0826
Epoch 136/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 876us/step - loss: 13733.9580 -
root_mean_squared_error: 117.1920
Epoch 137/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 973us/step - loss: 13750.8682 -
root_mean_squared_error: 117.2641
Epoch 138/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 864us/step - loss: 13736.5908 -
root_mean_squared_error: 117.2032
Epoch 139/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 842us/step - loss: 13799.0664 -
root_mean_squared_error: 117.4694
Epoch 140/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 13720.9785 -
root_mean_squared_error: 117.1366
Epoch 141/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 850us/step - loss: 13746.4053 -
root_mean_squared_error: 117.2451
Epoch 142/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 893us/step - loss: 13832.2969 -
root_mean_squared_error: 117.6108
Epoch 143/256
```

```
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 903us/step - loss: 13735.0586 -
root_mean_squared_error: 117.1967
Epoch 144/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 13714.3301 -
root_mean_squared_error: 117.1082
Epoch 145/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - loss: 13779.6885 -
root_mean_squared_error: 117.3869
Epoch 146/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 829us/step - loss: 13769.8008 -
root_mean_squared_error: 117.3448
Epoch 147/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - loss: 13734.0469 -
root_mean_squared_error: 117.1924
Epoch 148/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 13782.8701 -
root_mean_squared_error: 117.4005
Epoch 149/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - loss: 13830.1377 -
root_mean_squared_error: 117.6016
Epoch 150/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 13792.2646 -
root_mean_squared_error: 117.4405
Epoch 151/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 855us/step - loss: 13779.2061 -
root_mean_squared_error: 117.3849
Epoch 152/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 13806.2354 -
root_mean_squared_error: 117.4999
Epoch 153/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 13807.2236 -
root_mean_squared_error: 117.5041
Epoch 154/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 13734.2471 -
root_mean_squared_error: 117.1932
Epoch 155/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 861us/step - loss: 13862.6689 -
root_mean_squared_error: 117.7398
Epoch 156/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 13782.1494 -
root_mean_squared_error: 117.3974
Epoch 157/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 13779.2656 -
root_mean_squared_error: 117.3851
Epoch 158/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 13726.2822 -
root_mean_squared_error: 117.1592
Epoch 159/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 849us/step - loss: 13719.3193 -
```

```
root_mean_squared_error: 117.1295
Epoch 160/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 13722.4512 -
root_mean_squared_error: 117.1429
Epoch 161/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 13748.1631 -
root_mean_squared_error: 117.2526
Epoch 162/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 13728.8877 -
root_mean_squared_error: 117.1703
Epoch 163/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - loss: 13797.8252 -
root_mean_squared_error: 117.4641
Epoch 164/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 13726.9609 -
root_mean_squared_error: 117.1621
Epoch 165/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 13773.9854 -
root_mean_squared_error: 117.3626
Epoch 166/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 837us/step - loss: 13743.8398 -
root_mean_squared_error: 117.2341
Epoch 167/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 13781.2598 -
root_mean_squared_error: 117.3936
Epoch 168/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 13779.9111 -
root_mean_squared_error: 117.3879
Epoch 169/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 13815.1230 -
root_mean_squared_error: 117.5378
Epoch 170/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 13796.6104 -
root_mean_squared_error: 117.4590
Epoch 171/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 13773.4004 -
root_mean_squared_error: 117.3601
Epoch 172/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 13754.6064 -
root_mean_squared_error: 117.2800
Epoch 173/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 13783.9688 -
root_mean_squared_error: 117.4052
Epoch 174/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 847us/step - loss: 13768.2539 -
root_mean_squared_error: 117.3382
Epoch 175/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 13783.1738 -
root_mean_squared_error: 117.4018
Epoch 176/256
```

```
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 13769.3789 -
root_mean_squared_error: 117.3430
Epoch 177/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 13813.4102 -
root_mean_squared_error: 117.5305
Epoch 178/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - loss: 13759.7422 -
root_mean_squared_error: 117.3019
Epoch 179/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - loss: 13807.5752 -
root_mean_squared_error: 117.5056
Epoch 180/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 13791.0996 -
root_mean_squared_error: 117.4355
Epoch 181/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 13773.2305 -
root_mean_squared_error: 117.3594
Epoch 182/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 13737.0898 -
root_mean_squared_error: 117.2053
Epoch 183/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 13772.8496 -
root_mean_squared_error: 117.3578
Epoch 184/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 13763.5459 -
root_mean_squared_error: 117.3181
Epoch 185/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 13773.1777 -
root_mean_squared_error: 117.3592
Epoch 186/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 13780.6846 -
root_mean_squared_error: 117.3912
Epoch 187/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 13772.8330 -
root_mean_squared_error: 117.3577
Epoch 188/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 13772.6104 -
root_mean_squared_error: 117.3568
Epoch 189/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 13725.1592 -
root_mean_squared_error: 117.1544
Epoch 190/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - loss: 13754.1152 -
root_mean_squared_error: 117.2779
Epoch 191/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 13719.6406 -
root_mean_squared_error: 117.1309
Epoch 192/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 13778.3613 -
```

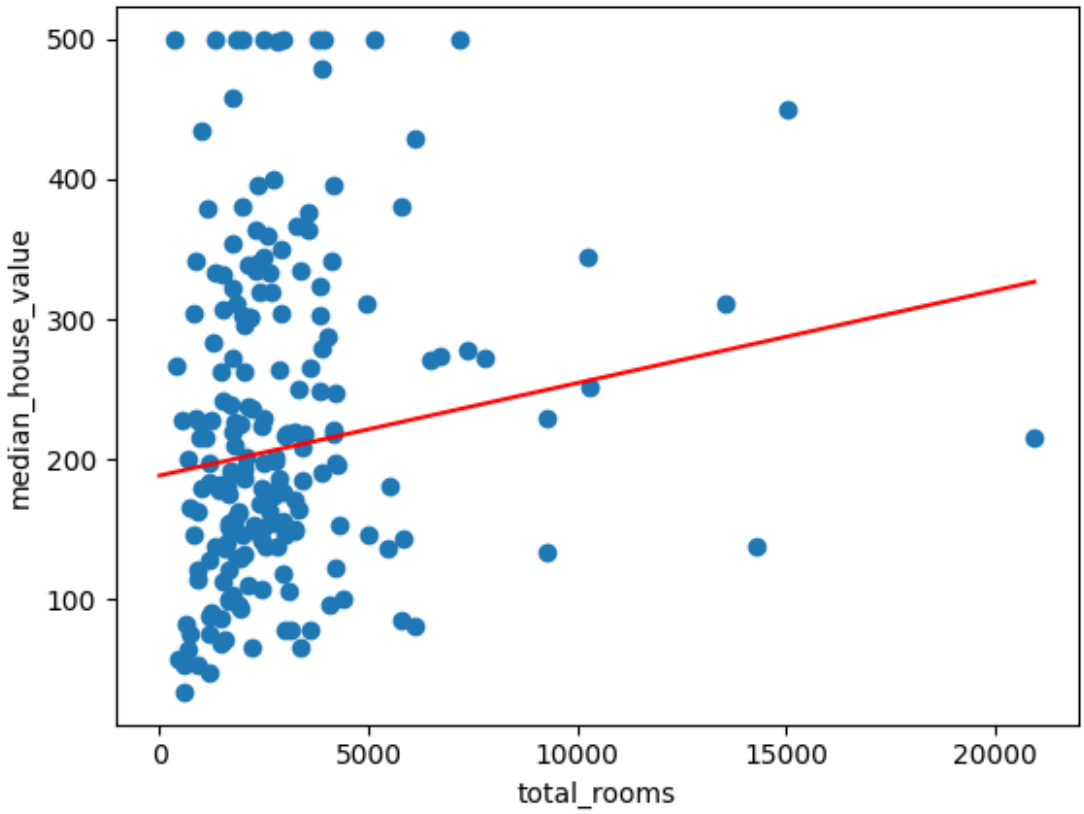
```
root_mean_squared_error: 117.3813
Epoch 193/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 13781.7920 -
root_mean_squared_error: 117.3959
Epoch 194/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - loss: 13759.7490 -
root_mean_squared_error: 117.3020
Epoch 195/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 13724.3545 -
root_mean_squared_error: 117.1510
Epoch 196/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 13749.5449 -
root_mean_squared_error: 117.2585
Epoch 197/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 13783.6494 -
root_mean_squared_error: 117.4038
Epoch 198/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step - loss: 13800.8467 -
root_mean_squared_error: 117.4770
Epoch 199/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 13787.8154 -
root_mean_squared_error: 117.4215
Epoch 200/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 13775.8223 -
root_mean_squared_error: 117.3704
Epoch 201/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - loss: 13795.2646 -
root_mean_squared_error: 117.4532
Epoch 202/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 13766.5850 -
root_mean_squared_error: 117.3311
Epoch 203/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 13766.4590 -
root_mean_squared_error: 117.3306
Epoch 204/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 13747.1240 -
root_mean_squared_error: 117.2481
Epoch 205/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - loss: 13779.2598 -
root_mean_squared_error: 117.3851
Epoch 206/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 848us/step - loss: 13757.9424 -
root_mean_squared_error: 117.2943
Epoch 207/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 13752.0176 -
root_mean_squared_error: 117.2690
Epoch 208/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 13743.4180 -
root_mean_squared_error: 117.2323
```

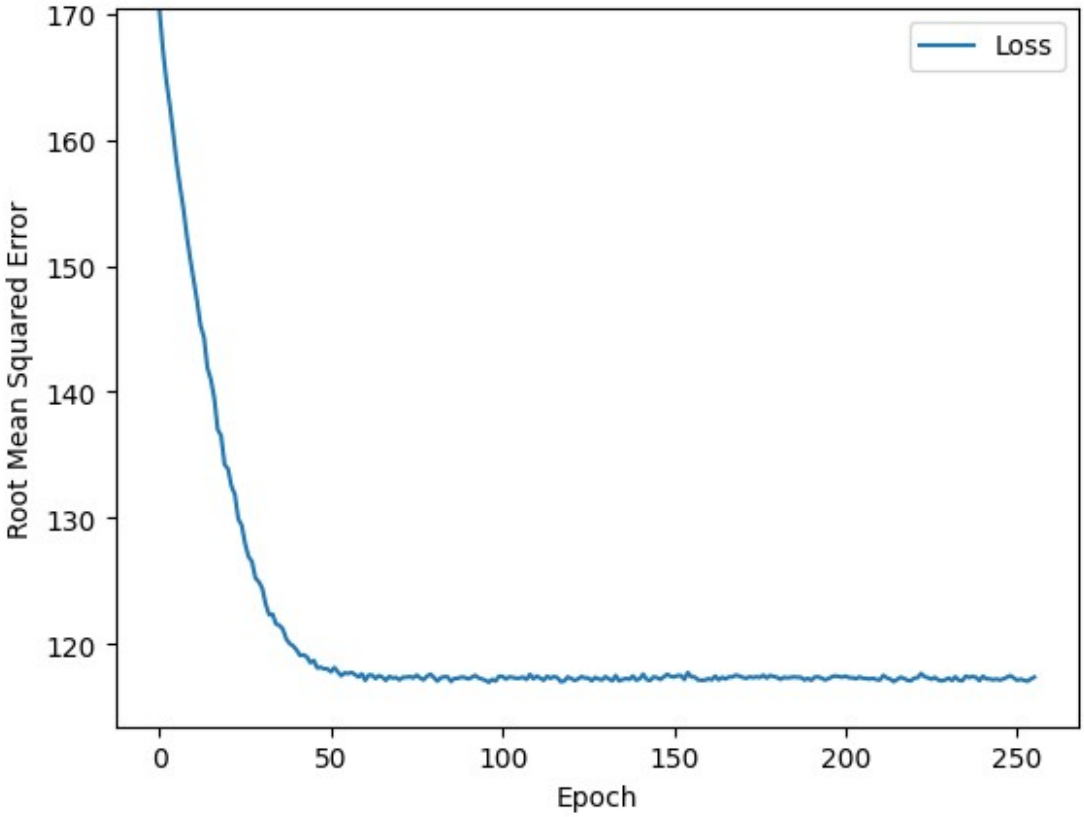
```
Epoch 209/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - loss: 13755.1621 -
root_mean_squared_error: 117.2824
Epoch 210/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 13732.5557 -
root_mean_squared_error: 117.1860
Epoch 211/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 13724.5537 -
root_mean_squared_error: 117.1518
Epoch 212/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 13811.0322 -
root_mean_squared_error: 117.5203
Epoch 213/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 881us/step - loss: 13764.2979 -
root_mean_squared_error: 117.3213
Epoch 214/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 13740.9590 -
root_mean_squared_error: 117.2218
Epoch 215/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 13694.7051 -
root_mean_squared_error: 117.0244
Epoch 216/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 856us/step - loss: 13731.5615 -
root_mean_squared_error: 117.1817
Epoch 217/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 13775.7012 -
root_mean_squared_error: 117.3699
Epoch 218/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 13754.2412 -
root_mean_squared_error: 117.2785
Epoch 219/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 13719.0283 -
root_mean_squared_error: 117.1283
Epoch 220/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 869us/step - loss: 13708.7256 -
root_mean_squared_error: 117.0843
Epoch 221/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - loss: 13758.2578 -
root_mean_squared_error: 117.2956
Epoch 222/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 13767.4248 -
root_mean_squared_error: 117.3347
Epoch 223/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - loss: 13844.0479 -
root_mean_squared_error: 117.6607
Epoch 224/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 13785.7998 -
root_mean_squared_error: 117.4129
Epoch 225/256
```

```
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 805us/step - loss: 13766.0332 -
root_mean_squared_error: 117.3287
Epoch 226/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 13737.6396 -
root_mean_squared_error: 117.2077
Epoch 227/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 13773.7588 -
root_mean_squared_error: 117.3617
Epoch 228/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 13725.5049 -
root_mean_squared_error: 117.1559
Epoch 229/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 13708.3291 -
root_mean_squared_error: 117.0826
Epoch 230/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 13726.3027 -
root_mean_squared_error: 117.1593
Epoch 231/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - loss: 13759.2803 -
root_mean_squared_error: 117.3000
Epoch 232/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - loss: 13715.3779 -
root_mean_squared_error: 117.1127
Epoch 233/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 853us/step - loss: 13783.7744 -
root_mean_squared_error: 117.4043
Epoch 234/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 769us/step - loss: 13718.4912 -
root_mean_squared_error: 117.1260
Epoch 235/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - loss: 13710.2061 -
root_mean_squared_error: 117.0906
Epoch 236/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 13795.7783 -
root_mean_squared_error: 117.4554
Epoch 237/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 13769.4443 -
root_mean_squared_error: 117.3433
Epoch 238/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 773us/step - loss: 13788.2959 -
root_mean_squared_error: 117.4236
Epoch 239/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 13772.9697 -
root_mean_squared_error: 117.3583
Epoch 240/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 13712.5410 -
root_mean_squared_error: 117.1006
Epoch 241/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - loss: 13793.9805 -
```

```
root_mean_squared_error: 117.4478
Epoch 242/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 13746.8223 -
root_mean_squared_error: 117.2468
Epoch 243/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 771us/step - loss: 13742.7734 -
root_mean_squared_error: 117.2296
Epoch 244/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - loss: 13727.5850 -
root_mean_squared_error: 117.1648
Epoch 245/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 13742.4805 -
root_mean_squared_error: 117.2283
Epoch 246/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 13722.8164 -
root_mean_squared_error: 117.1444
Epoch 247/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 13745.1816 -
root_mean_squared_error: 117.2398
Epoch 248/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 13773.4111 -
root_mean_squared_error: 117.3602
Epoch 249/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 13804.1689 -
root_mean_squared_error: 117.4911
Epoch 250/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 13763.9062 -
root_mean_squared_error: 117.3197
Epoch 251/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 13719.7373 -
root_mean_squared_error: 117.1313
Epoch 252/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 13739.2598 -
root_mean_squared_error: 117.2146
Epoch 253/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 13720.6885 -
root_mean_squared_error: 117.1353
Epoch 254/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 13699.5703 -
root_mean_squared_error: 117.0452
Epoch 255/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 13745.4736 -
root_mean_squared_error: 117.2411
Epoch 256/256
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 13775.7480 -
root_mean_squared_error: 117.3701

The learned weight for your model is 0.0066
```

```
The learned bias for your model is 188.3858


C:\Users\micki\AppData\Local\Temp\ipykernel_22432\62330301.py:21:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  print("\nThe learned weight for your model is %.4f" % weight)
C:\Users\micki\AppData\Local\Temp\ipykernel_22432\62330301.py:22:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  print("The learned bias for your model is %.4f\n" % bias )
C:\Users\micki\AppData\Local\Temp\ipykernel_22432\2255290996.py:14:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  w = float(trained_weight)
C:\Users\micki\AppData\Local\Temp\ipykernel_22432\2255290996.py:15:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  b = float(trained_bias)
```


500
400
300
200
100
median_house_value
0
5000
10000
15000
20000
total_rooms


170
160
150
140
130
120
Root Mean Squared Error
Loss
0
50
100
150
200
250
Epoch

A certain amount of randomness plays into training a model. Consequently, you'll get different results each time you train the model. That said, given the dataset and the hyperparameters, the trained model will generally do a poor job describing the feature's relation to the label.

## Use the model to make predictions

You can use the trained model to make predictions. In practice, you should make predictions on examples that are not used in training. However, for this exercise, you'll just work with a subset of the same training dataset. A later Colab exercise will explore ways to make predictions on examples not used in training.

First, run the following code to define the house prediction function:

```
def predict_house_values(n, feature, label):
  """Predict house values based on a feature."""

  batch = training_df[feature][10000:10000 + n]
  predicted_values = my_model.predict_on_batch(x=batch)

  print("feature   label          predicted")
  print("  value   value          value")
  print("          in thousand$   in thousand$")
  print("------------------------------------")
  for i in range(n):
    print ("%5.0f %6.0f %15.0f" % (training_df[feature][10000 + i],
                                   training_df[label][10000 + i],
                                   predicted_values[i][0] ))
```

Now, invoke the house prediction function on 10 examples:

```
predict_house_values(10, my_feature, my_label)
```

```
feature   label          predicted
  value   value          value
          in thousand$   in thousand$
------------------------------------
 1960     53             177
 3400     92             210
 3677     69             217
 2202     62             183
 2403     80             188
 5652    295             262
 3318    500             209
 2552    342             191
 1364    118             164
 3468    128             212
```

## Task 2: Judge the predictive power of the model

Look at the preceding table. How close is the predicted value to the label value? In other words, does your model accurately predict house values?

> Using percent error, predictions start off extremely inaccurate—spiking around 226% above the actual values—and then gradually improve, dropping to about 65% error by the tenth iteration. This suggests the model struggles to capture the true relationship between the chosen feature (total_rooms) and label (median house values. Still, these first few cases may not fully represent the dataset, so the model's overall predictive power could differ when evaluated more broadly

# Task 3: Try a different feature

The `total_rooms` feature had only a little predictive power. Would a different feature have greater predictive power? Try using `population` as the feature instead of `total_rooms`.

Note: When you change features, you might also need to change the hyperparameters.

```
training_df.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

```
my_feature = "population"   # Replace the ? with population or
possibly
                  # a different column name.

# Experiment with the hyperparameters.
learning_rate = 0.001
epochs = 128
batch_size = 32

# Don't change anything below this line.
my_model = build_model(learning_rate)
weight, bias, epochs, rmse = train_model(my_model, training_df,
                                         my_feature, my_label,
                                         epochs, batch_size)
plot_the_model(weight, bias, my_feature, my_label)
plot_the_loss_curve(epochs, rmse)

predict_house_values(15, my_feature, my_label)
```

```
Epoch 1/128
```

```
c:\Users\micki\anaconda3\envs\tf_env\lib\site-packages\keras\src\
layers\core\dense.py:92: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
```

```
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 806us/step - loss: 815981.0000 -
root_mean_squared_error: 903.3167
Epoch 2/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 842us/step - loss: 58412.1797 -
root_mean_squared_error: 241.6861
Epoch 3/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 837us/step - loss: 30832.9863 -
root_mean_squared_error: 175.5932
Epoch 4/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 875us/step - loss: 30769.1523 -
root_mean_squared_error: 175.4114
Epoch 5/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 30730.8496 -
root_mean_squared_error: 175.3022
Epoch 6/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 30519.0430 -
root_mean_squared_error: 174.6970
Epoch 7/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 30451.1953 -
root_mean_squared_error: 174.5027
Epoch 8/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 30442.6973 -
root_mean_squared_error: 174.4784
Epoch 9/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 30263.0996 -
root_mean_squared_error: 173.9629
Epoch 10/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 30242.5371 -
root_mean_squared_error: 173.9038
Epoch 11/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - loss: 30164.0703 -
root_mean_squared_error: 173.6781
Epoch 12/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 30063.0332 -
root_mean_squared_error: 173.3869
Epoch 13/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 867us/step - loss: 29996.5332 -
root_mean_squared_error: 173.1951
Epoch 14/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - loss: 29929.6367 -
root_mean_squared_error: 173.0018
Epoch 15/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29866.6074 -
root_mean_squared_error: 172.8196
Epoch 16/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29861.5156 -
```

```
root_mean_squared_error: 172.8049
Epoch 17/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 29737.1445 -
root_mean_squared_error: 172.4446
Epoch 18/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29688.5898 -
root_mean_squared_error: 172.3038
Epoch 19/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 991us/step - loss: 29559.5156 -
root_mean_squared_error: 171.9288
Epoch 20/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29468.7676 -
root_mean_squared_error: 171.6647
Epoch 21/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29411.9375 -
root_mean_squared_error: 171.4991
Epoch 22/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29296.1504 -
root_mean_squared_error: 171.1612
Epoch 23/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29290.2812 -
root_mean_squared_error: 171.1440
Epoch 24/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29139.6211 -
root_mean_squared_error: 170.7033
Epoch 25/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29139.5215 -
root_mean_squared_error: 170.7030
Epoch 26/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 29040.6113 -
root_mean_squared_error: 170.4131
Epoch 27/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 28941.3203 -
root_mean_squared_error: 170.1215
Epoch 28/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 963us/step - loss: 28907.1152 -
root_mean_squared_error: 170.0209
Epoch 29/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 876us/step - loss: 28924.6289 -
root_mean_squared_error: 170.0724
Epoch 30/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 947us/step - loss: 28703.0957 -
root_mean_squared_error: 169.4199
Epoch 31/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 885us/step - loss: 28644.3770 -
root_mean_squared_error: 169.2465
Epoch 32/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 28599.8691 -
root_mean_squared_error: 169.1150
```

```
Epoch 33/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 28651.2031 -
root_mean_squared_error: 169.2667
Epoch 34/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 28378.7832 -
root_mean_squared_error: 168.4600
Epoch 35/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 891us/step - loss: 28355.4160 -
root_mean_squared_error: 168.3907
Epoch 36/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 28249.5000 -
root_mean_squared_error: 168.0759
Epoch 37/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - loss: 28085.7031 -
root_mean_squared_error: 167.5879
Epoch 38/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 842us/step - loss: 28109.7070 -
root_mean_squared_error: 167.6595
Epoch 39/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 831us/step - loss: 28033.1328 -
root_mean_squared_error: 167.4310
Epoch 40/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - loss: 27924.6660 -
root_mean_squared_error: 167.1068
Epoch 41/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 27897.7480 -
root_mean_squared_error: 167.0262
Epoch 42/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 27740.0234 -
root_mean_squared_error: 166.5534
Epoch 43/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 27705.7129 -
root_mean_squared_error: 166.4503
Epoch 44/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 831us/step - loss: 27680.6191 -
root_mean_squared_error: 166.3749
Epoch 45/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 27521.1680 -
root_mean_squared_error: 165.8951
Epoch 46/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 27534.2598 -
root_mean_squared_error: 165.9345
Epoch 47/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 27542.6719 -
root_mean_squared_error: 165.9599
Epoch 48/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 27330.6758 -
root_mean_squared_error: 165.3199
Epoch 49/128
```

```
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 873us/step - loss: 27382.0195 -
root_mean_squared_error: 165.4751
Epoch 50/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 890us/step - loss: 27198.6602 -
root_mean_squared_error: 164.9202
Epoch 51/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 925us/step - loss: 27147.2129 -
root_mean_squared_error: 164.7641
Epoch 52/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 898us/step - loss: 27017.8301 -
root_mean_squared_error: 164.3710
Epoch 53/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 27038.0352 -
root_mean_squared_error: 164.4325
Epoch 54/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 920us/step - loss: 26946.1094 -
root_mean_squared_error: 164.1527
Epoch 55/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 993us/step - loss: 26824.3066 -
root_mean_squared_error: 163.7813
Epoch 56/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 960us/step - loss: 26847.9238 -
root_mean_squared_error: 163.8534
Epoch 57/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 26741.8438 -
root_mean_squared_error: 163.5293
Epoch 58/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 854us/step - loss: 26705.3398 -
root_mean_squared_error: 163.4177
Epoch 59/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 26499.2363 -
root_mean_squared_error: 162.7859
Epoch 60/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 26548.2461 -
root_mean_squared_error: 162.9363
Epoch 61/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 872us/step - loss: 26461.0488 -
root_mean_squared_error: 162.6685
Epoch 62/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 870us/step - loss: 26232.8008 -
root_mean_squared_error: 161.9654
Epoch 63/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 882us/step - loss: 26269.7070 -
root_mean_squared_error: 162.0793
Epoch 64/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 913us/step - loss: 26241.0977 -
root_mean_squared_error: 161.9910
Epoch 65/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 900us/step - loss: 26125.3418 -
```

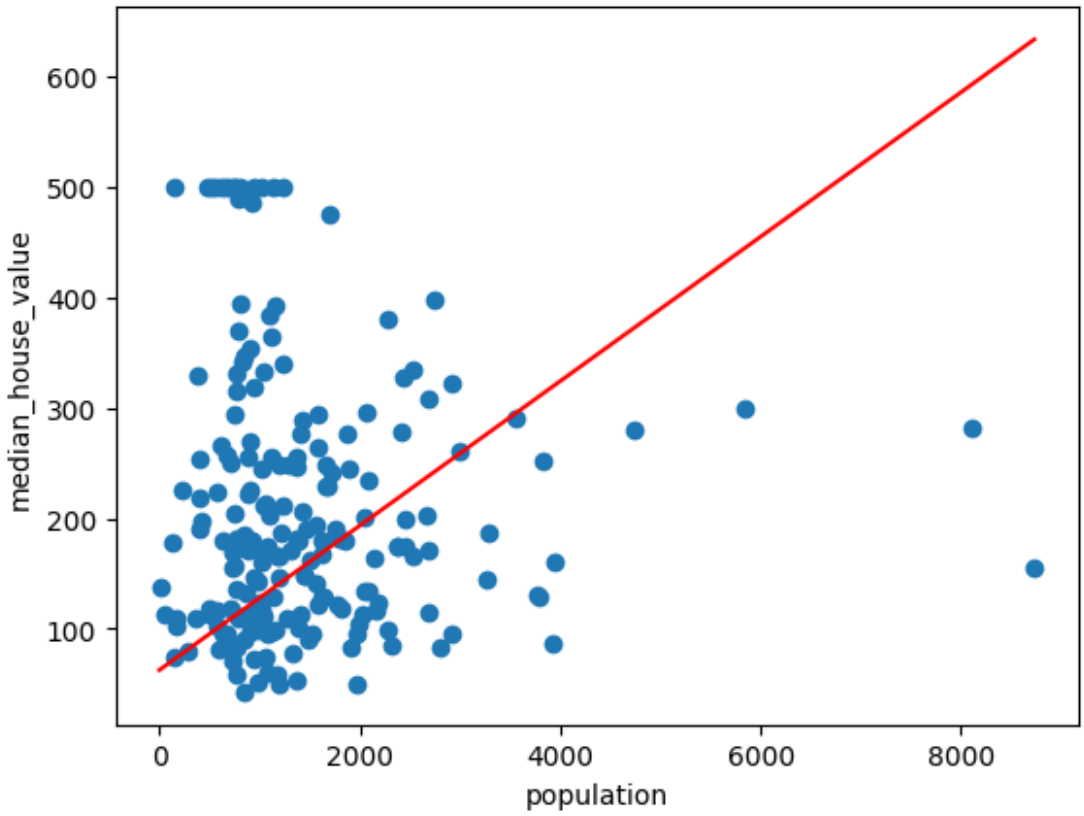
```
root_mean_squared_error: 161.6334
Epoch 66/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 866us/step - loss: 26131.7246 -
root_mean_squared_error: 161.6531
Epoch 67/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 842us/step - loss: 25946.0215 -
root_mean_squared_error: 161.0777
Epoch 68/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 26027.9551 -
root_mean_squared_error: 161.3318
Epoch 69/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 841us/step - loss: 25887.2500 -
root_mean_squared_error: 160.8952
Epoch 70/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 891us/step - loss: 25809.7793 -
root_mean_squared_error: 160.6542
Epoch 71/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 25754.8926 -
root_mean_squared_error: 160.4833
Epoch 72/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 25666.3789 -
root_mean_squared_error: 160.2073
Epoch 73/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - loss: 25609.1777 -
root_mean_squared_error: 160.0287
Epoch 74/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 883us/step - loss: 25589.9102 -
root_mean_squared_error: 159.9685
Epoch 75/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 25456.4219 -
root_mean_squared_error: 159.5507
Epoch 76/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 25397.4844 -
root_mean_squared_error: 159.3659
Epoch 77/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 25332.9473 -
root_mean_squared_error: 159.1633
Epoch 78/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 25276.6543 -
root_mean_squared_error: 158.9863
Epoch 79/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 849us/step - loss: 25169.0508 -
root_mean_squared_error: 158.6476
Epoch 80/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 25108.9414 -
root_mean_squared_error: 158.4580
Epoch 81/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 25047.2578 -
root_mean_squared_error: 158.2633
```

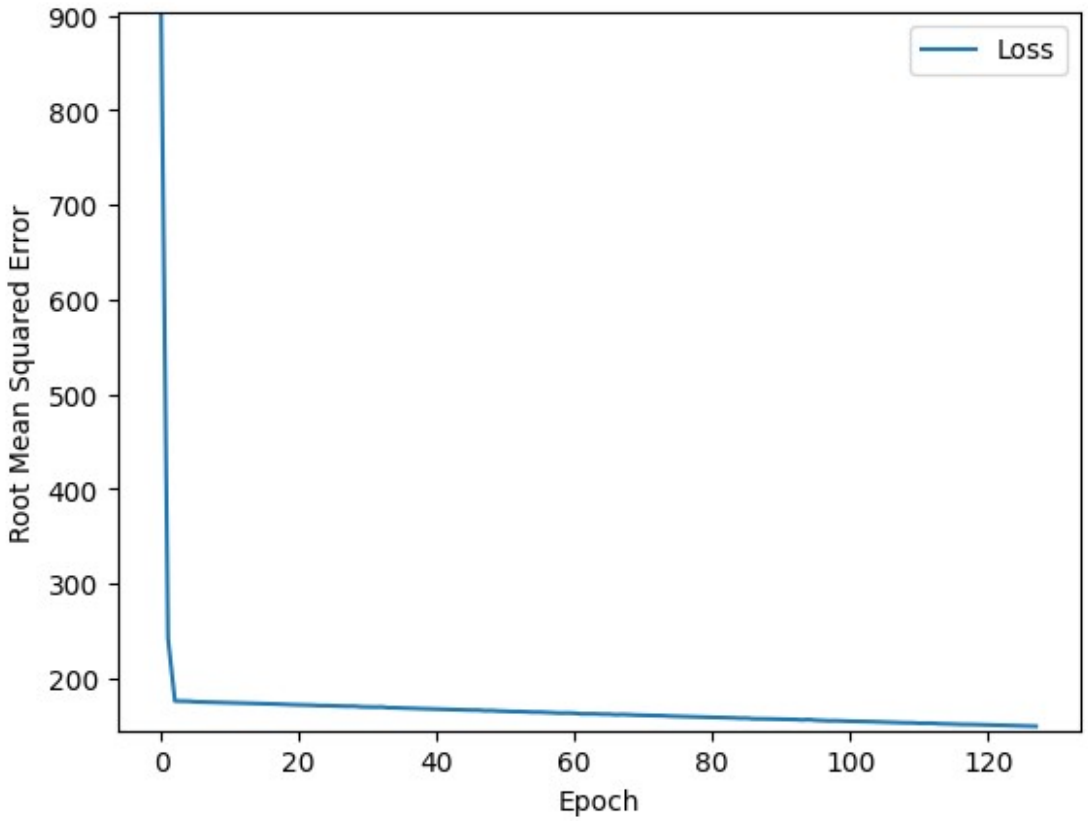
```
Epoch 82/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 894us/step - loss: 24976.8809 -
root_mean_squared_error: 158.0408
Epoch 83/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - loss: 24946.6484 -
root_mean_squared_error: 157.9451
Epoch 84/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 873us/step - loss: 24867.2305 -
root_mean_squared_error: 157.6935
Epoch 85/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 844us/step - loss: 24763.1309 -
root_mean_squared_error: 157.3631
Epoch 86/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 24781.0684 -
root_mean_squared_error: 157.4200
Epoch 87/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step - loss: 24576.7559 -
root_mean_squared_error: 156.7698
Epoch 88/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 936us/step - loss: 24577.4805 -
root_mean_squared_error: 156.7721
Epoch 89/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 986us/step - loss: 24541.6504 -
root_mean_squared_error: 156.6577
Epoch 90/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 24496.8613 -
root_mean_squared_error: 156.5147
Epoch 91/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 841us/step - loss: 24452.5781 -
root_mean_squared_error: 156.3732
Epoch 92/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 934us/step - loss: 24353.4961 -
root_mean_squared_error: 156.0561
Epoch 93/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 901us/step - loss: 24346.0684 -
root_mean_squared_error: 156.0323
Epoch 94/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 890us/step - loss: 24209.9297 -
root_mean_squared_error: 155.5954
Epoch 95/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 24327.7656 -
root_mean_squared_error: 155.9736
Epoch 96/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 24098.3066 -
root_mean_squared_error: 155.2363
Epoch 97/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 23974.4551 -
root_mean_squared_error: 154.8369
Epoch 98/128
```

```
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - loss: 23917.1777 -
root_mean_squared_error: 154.6518
Epoch 99/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 23938.2812 -
root_mean_squared_error: 154.7200
Epoch 100/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 23821.2109 -
root_mean_squared_error: 154.3412
Epoch 101/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 23767.3945 -
root_mean_squared_error: 154.1668
Epoch 102/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - loss: 23734.5234 -
root_mean_squared_error: 154.0601
Epoch 103/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 23709.4180 -
root_mean_squared_error: 153.9786
Epoch 104/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 23637.2051 -
root_mean_squared_error: 153.7440
Epoch 105/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 879us/step - loss: 23546.0820 -
root_mean_squared_error: 153.4473
Epoch 106/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 23472.4551 -
root_mean_squared_error: 153.2072
Epoch 107/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 23427.3027 -
root_mean_squared_error: 153.0598
Epoch 108/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 23342.7773 -
root_mean_squared_error: 152.7834
Epoch 109/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step - loss: 23265.5879 -
root_mean_squared_error: 152.5306
Epoch 110/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 23265.5684 -
root_mean_squared_error: 152.5305
Epoch 111/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 23114.7988 -
root_mean_squared_error: 152.0355
Epoch 112/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 23104.2188 -
root_mean_squared_error: 152.0007
Epoch 113/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 872us/step - loss: 23123.3770 -
root_mean_squared_error: 152.0637
Epoch 114/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 22968.3574 -
```

```
root_mean_squared_error: 151.5531
Epoch 115/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 22937.9102 -
root_mean_squared_error: 151.4527
Epoch 116/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 22870.7168 -
root_mean_squared_error: 151.2307
Epoch 117/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 866us/step - loss: 22769.6660 -
root_mean_squared_error: 150.8962
Epoch 118/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 22758.4492 -
root_mean_squared_error: 150.8590
Epoch 119/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 1s 956us/step - loss: 22719.0703 -
root_mean_squared_error: 150.7285
Epoch 120/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 22658.7383 -
root_mean_squared_error: 150.5282
Epoch 121/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 22537.7969 -
root_mean_squared_error: 150.1259
Epoch 122/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 877us/step - loss: 22553.8047 -
root_mean_squared_error: 150.1792
Epoch 123/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 22481.9531 -
root_mean_squared_error: 149.9398
Epoch 124/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 22403.4258 -
root_mean_squared_error: 149.6777
Epoch 125/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 22384.2676 -
root_mean_squared_error: 149.6137
Epoch 126/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 22308.9004 -
root_mean_squared_error: 149.3616
Epoch 127/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 22271.1719 -
root_mean_squared_error: 149.2353
Epoch 128/128
532/532 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 22161.7773 -
root_mean_squared_error: 148.8683

C:\Users\micki\AppData\Local\Temp\ipykernel_22432\2255290996.py:14:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  w = float(trained_weight)
```

```
C:\Users\micki\AppData\Local\Temp\ipykernel_22432\2255290996.py:15:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  b = float(trained_bias)
```






```
WARNING:tensorflow:5 out of the last 5 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x00000171DB08EC20> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors
with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
feature   label          predicted
  value   value          value
          in thousand$   in thousand$
-------------------------------------
 1286     53             147
 1867     92             185
 2191     69             206
 1052     62             131
 1647     80             170
 2312    295             214
 1604    500             168
 1066    342             132
```

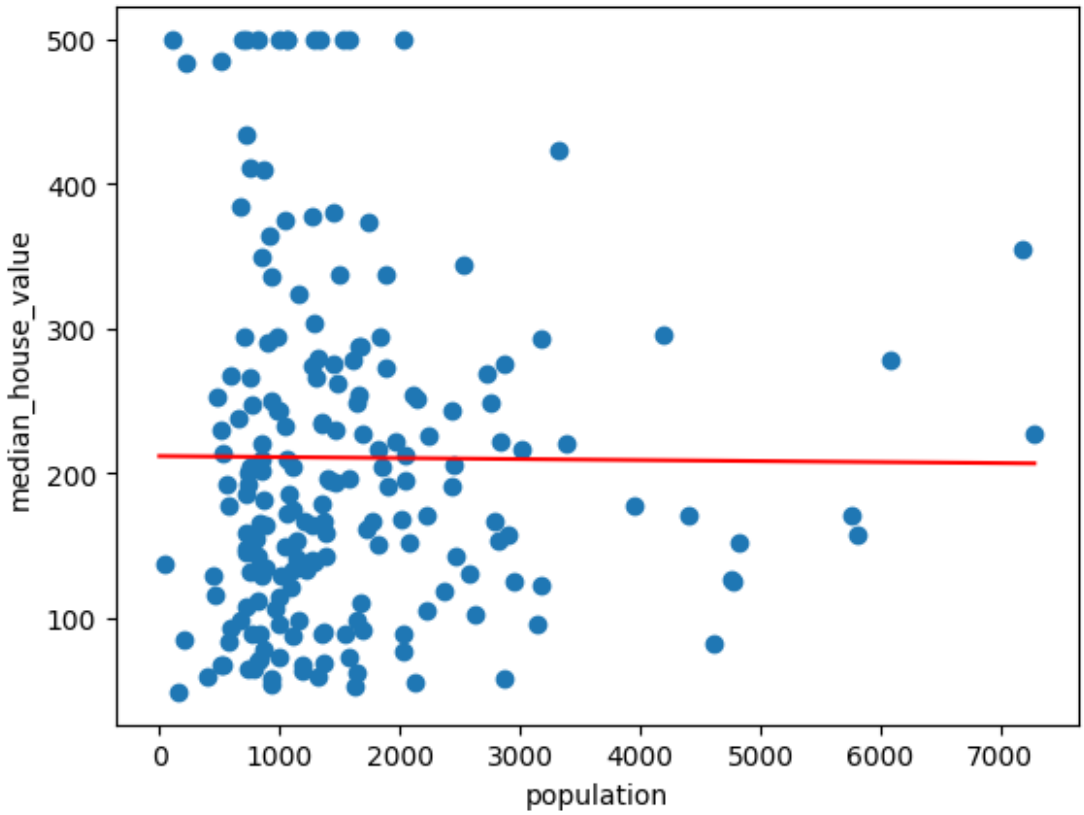
```
  338    118              85
 1604    128             168
 1200    187             141
  292     80              82
 2014    112             194
 1817     95             181
 1328     69             150
```

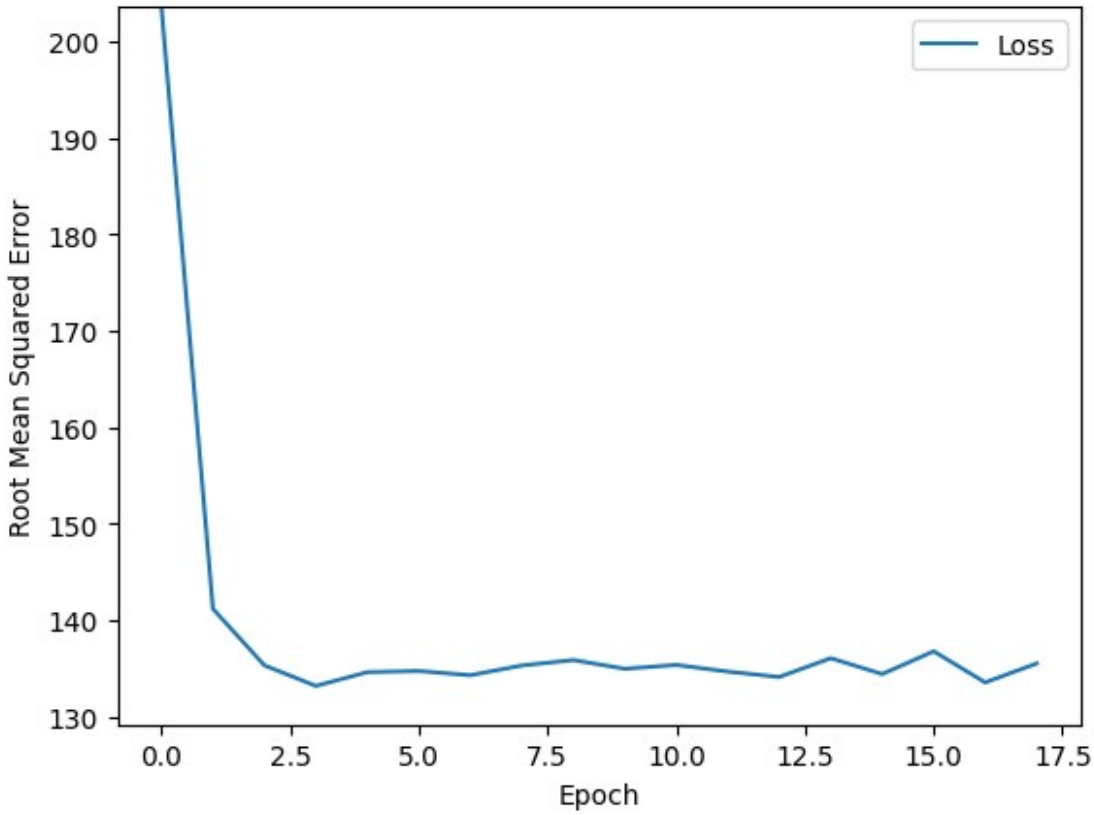
```
my_feature = "population" # Pick a feature other than "total_rooms"

# Possibly, experiment with the hyperparameters.
learning_rate = 0.05
epochs = 18
batch_size = 3

# Don't change anything below.
my_model = build_model(learning_rate)
weight, bias, epochs, rmse = train_model(my_model, training_df,
                                         my_feature, my_label,
                                         epochs, batch_size)

plot_the_model(weight, bias, my_feature, my_label)
plot_the_loss_curve(epochs, rmse)

predict_house_values(10, my_feature, my_label)
```

```
Epoch 1/18

c:\Users\micki\anaconda3\envs\tf_env\lib\site-packages\keras\src\
layers\core\dense.py:92: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 802us/step - loss: 41442.5352 -
root_mean_squared_error: 203.5744
Epoch 2/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 817us/step - loss: 19937.4512 -
root_mean_squared_error: 141.2000
Epoch 3/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 944us/step - loss: 18317.3809 -
root_mean_squared_error: 135.3417
Epoch 4/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - loss: 17744.2500 -
root_mean_squared_error: 133.2076
Epoch 5/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 842us/step - loss: 18127.0254 -
root_mean_squared_error: 134.6366
Epoch 6/18
```

```
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 861us/step - loss: 18158.4492 -
root_mean_squared_error: 134.7533
Epoch 7/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 802us/step - loss: 18041.1641 -
root_mean_squared_error: 134.3174
Epoch 8/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 4s 785us/step - loss: 18312.9805 -
root_mean_squared_error: 135.3255
Epoch 9/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 833us/step - loss: 18465.4902 -
root_mean_squared_error: 135.8878
Epoch 10/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 847us/step - loss: 18221.2480 -
root_mean_squared_error: 134.9861
Epoch 11/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 6s 990us/step - loss: 18328.3906 -
root_mean_squared_error: 135.3824
Epoch 12/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 915us/step - loss: 18142.0938 -
root_mean_squared_error: 134.6926
Epoch 13/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 904us/step - loss: 17991.9414 -
root_mean_squared_error: 134.1340
Epoch 14/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 823us/step - loss: 18517.8164 -
root_mean_squared_error: 136.0802
Epoch 15/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 816us/step - loss: 18073.5000 -
root_mean_squared_error: 134.4377
Epoch 16/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 812us/step - loss: 18713.5137 -
root_mean_squared_error: 136.7973
Epoch 17/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 827us/step - loss: 17840.4727 -
root_mean_squared_error: 133.5682
Epoch 18/18
5667/5667 ━━━━━━━━━━━━━━━━━━━━ 5s 848us/step - loss: 18369.8418 -
root_mean_squared_error: 135.5354

C:\Users\micki\AppData\Local\Temp\ipykernel_22432\2255290996.py:14:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
NumPy 1.25.)
  w = float(trained_weight)
C:\Users\micki\AppData\Local\Temp\ipykernel_22432\2255290996.py:15:
DeprecationWarning: Conversion of an array with ndim > 0 to a scalar
is deprecated, and will error in future. Ensure you extract a single
element from your array before performing this operation. (Deprecated
```

```
NumPy 1.25.)
  b = float(trained_bias)
```






```
feature   label          predicted
  value   value          value
          in thousand$   in thousand$
--------------------------------------
 1286     53             211
 1867     92             211
 2191     69             210
 1052     62             211
 1647     80             211
 2312    295             210
 1604    500             211
 1066    342             211
  338    118             212
 1604    128             211
```

Did `population` produce better predictions than `total_rooms`?

> Using population usually ended up with a a higher RMSE than total_rooms. This suggests that population performs a bit worse than total_rooms when it comes to making predictions.

# Task 4: Define a synthetic feature

You have determined that `total_rooms` and `population` were not useful features. That is, neither the total number of rooms in a neighborhood nor the neighborhood's population successfully predicted the median house price of that neighborhood. Perhaps though, the *ratio* of `total_rooms` to `population` might have some predictive power. That is, perhaps block density relates to median house value.

To explore this hypothesis, do the following:

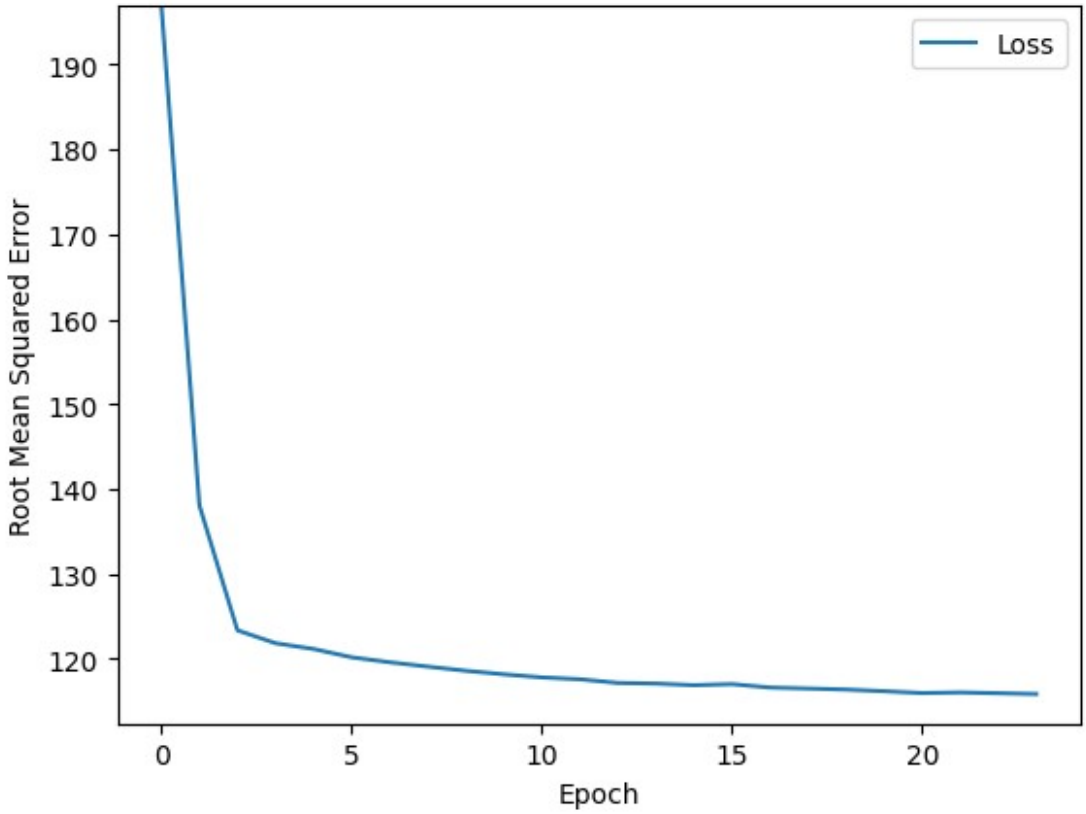
1. Create a synthetic feature that's a ratio of `total_rooms` to `population`. (If you are new to pandas DataFrames, please study the Pandas DataFrame Ultraquick Tutorial.)
2. Tune the three hyperparameters.
3. Determine whether this synthetic feature produces a lower loss value than any of the single features you tried earlier in this exercise.

```
# Define a synthetic feature
training_df["rooms_per_person"] = training_df["total_rooms"] /
training_df["population"]
my_feature = "rooms_per_person"

# Tune the hyperparameters.
learning_rate = 0.06
epochs = 24
batch_size = 30

# Don't change anything below this line.
my_model = build_model(learning_rate)
weight, bias, epochs, mae = train_model(my_model, training_df,
                                        my_feature, my_label,
                                        epochs, batch_size)

plot_the_loss_curve(epochs, mae)
predict_house_values(15, my_feature, my_label)
```

```
Epoch 1/24

c:\Users\micki\anaconda3\envs\tf_env\lib\site-packages\keras\src\
layers\core\dense.py:92: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 857us/step - loss: 38772.3594 -
root_mean_squared_error: 196.9070
Epoch 2/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 19060.9531 -
root_mean_squared_error: 138.0614
Epoch 3/24
```

```
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 884us/step - loss: 15217.3223 -
root_mean_squared_error: 123.3585
Epoch 4/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 924us/step - loss: 14845.8965 -
root_mean_squared_error: 121.8437
Epoch 5/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 14681.0723 -
root_mean_squared_error: 121.1655
Epoch 6/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 14444.9307 -
root_mean_squared_error: 120.1871
Epoch 7/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step - loss: 14304.4619 -
root_mean_squared_error: 119.6013
Epoch 8/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 14182.6895 -
root_mean_squared_error: 119.0911
Epoch 9/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 14065.9951 -
root_mean_squared_error: 118.6002
Epoch 10/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 13967.2627 -
root_mean_squared_error: 118.1832
Epoch 11/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 866us/step - loss: 13880.8789 -
root_mean_squared_error: 117.8171
Epoch 12/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 13829.8428 -
root_mean_squared_error: 117.6003
Epoch 13/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 845us/step - loss: 13728.5078 -
root_mean_squared_error: 117.1687
Epoch 14/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 13710.8496 -
root_mean_squared_error: 117.0933
Epoch 15/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 13665.8115 -
root_mean_squared_error: 116.9009
Epoch 16/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 13695.6846 -
root_mean_squared_error: 117.0286
Epoch 17/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 13600.8125 -
root_mean_squared_error: 116.6225
Epoch 18/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 853us/step - loss: 13574.3486 -
root_mean_squared_error: 116.5090
Epoch 19/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - loss: 13546.3359 -
```

```
root_mean_squared_error: 116.3887
Epoch 20/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 13500.7686 -
root_mean_squared_error: 116.1928
Epoch 21/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 13450.1309 -
root_mean_squared_error: 115.9747
Epoch 22/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 862us/step - loss: 13466.7939 -
root_mean_squared_error: 116.0465
Epoch 23/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 1s 861us/step - loss: 13446.0537 -
root_mean_squared_error: 115.9571
Epoch 24/24
567/567 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 13424.3984 -
root_mean_squared_error: 115.8637
```




```
WARNING:tensorflow:6 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x00000171DB107F40> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors
with different shapes, (3) passing Python objects instead of tensors.
```

```
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
feature   label          predicted
  value   value          value
          in thousand$   in thousand$
------------------------------------
    2      53            190
    2      92            202
    2      69            196
    2      62            213
    1      80            187
    2     295            227
    2     500            212
    2     342            225
    4     118            292
    2     128            216
    2     187            226
    3      80            237
    2     112            227
    2      95            221
    2      69            212
```

Based on the loss values, this synthetic feature produces a better model than the individual features you tried in Task 2 and Task 3. However, the model still isn't creating great predictions.

## Task 5. Find feature(s) whose raw values correlate with the label

So far, we've relied on trial-and-error to identify possible features for the model. Let's rely on statistics instead.

A **correlation matrix** indicates how each attribute's raw values relate to the other attributes' raw values. Correlation values have the following meanings:

- `1.0`: perfect positive correlation; that is, when one attribute rises, the other attribute rises.
- `-1.0`: perfect negative correlation; that is, when one attribute rises, the other attribute falls.
- `0.0`: no correlation; the two columns are not linearly related.

In general, the higher the absolute value of a correlation value, the greater its predictive power. For example, a correlation value of -0.8 implies far more predictive power than a correlation of -0.2.

The following code cell generates the correlation matrix for attributes of the California Housing Dataset:

```
# Generate a correlation matrix.
training_df.corr()
```

```
                     longitude  latitude  housing_median_age
total_rooms  \
longitude                  1.0      -0.9                -0.1
0.0
latitude                  -0.9       1.0                 0.0          -
0.0
housing_median_age        -0.1       0.0                 1.0          -
0.4
total_rooms                0.0      -0.0                -0.4
1.0
total_bedrooms             0.1      -0.1                -0.3
0.9
population                 0.1      -0.1                -0.3
0.9
households                 0.1      -0.1                -0.3
0.9
median_income             -0.0      -0.1                -0.1
0.2
median_house_value        -0.0      -0.1                 0.1
0.1
rooms_per_person          -0.1       0.1                -0.1
0.1

                     total_bedrooms  population  households
median_income  \
longitude                       0.1         0.1         0.1           -
0.0
latitude                       -0.1        -0.1        -0.1           -
0.1
housing_median_age             -0.3        -0.3        -0.3           -
0.1
total_rooms                     0.9         0.9         0.9
0.2
total_bedrooms                  1.0         0.9         1.0           -
0.0
population                      0.9         1.0         0.9           -
0.0
households                      1.0         0.9         1.0
0.0
median_income                  -0.0        -0.0         0.0
1.0
median_house_value              0.0        -0.0         0.1
0.7
rooms_per_person                0.0        -0.1        -0.0
```

```
0.2
```

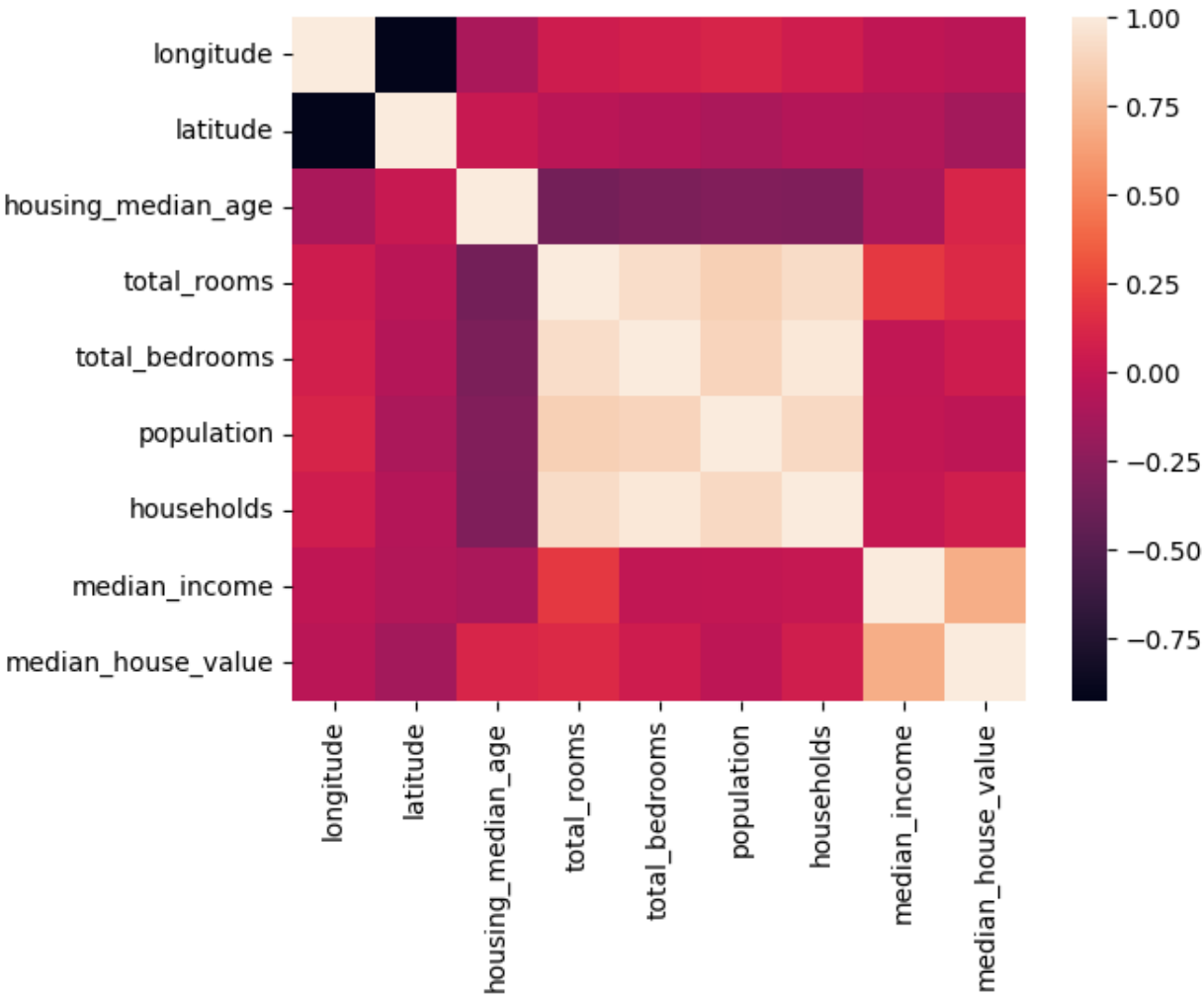
| | median_house_value | rooms_per_person |
|---|---|---|
| longitude | -0.0 | -0.1 |
| latitude | -0.1 | 0.1 |
| housing_median_age | 0.1 | -0.1 |
| total_rooms | 0.1 | 0.1 |
| total_bedrooms | 0.0 | 0.0 |
| population | -0.0 | -0.1 |
| households | 0.1 | -0.0 |
| median_income | 0.7 | 0.2 |
| median_house_value | 1.0 | 0.2 |
| rooms_per_person | 0.2 | 1.0 |

The correlation matrix shows nine potential features (including a synthetic feature) and one label (`median_house_value`). A strong negative correlation or strong positive correlation with the label suggests a potentially good feature.

**Your Task:** Determine which of the nine potential features appears to be the best candidate for a feature?

```
import seaborn as sns
sns.heatmap(corr)
```

```
<Axes: >
```




Median_income shows a strong correlation of about 0.7 with the median_house_value, which suggests it could be a good feature to use. The other features have correlations close to zero, so they don't seem very helpful for prediction. If I have time, I can try using median_income as the feature to check if it improves the model's performance.

Correlation matrices don't tell the entire story. In later exercises, you'll find additional ways to unlock predictive power from potential features.

**Note:** Using `median_income` as a feature may raise some ethical and fairness issues. Towards the end of the course, we'll explore ethical and fairness issues.