prac_linear

April 29, 2025

[1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.datasets import boston_housing
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
```

```
2025-04-29 07:14:28.590578: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-04-29 07:14:28.594512: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-04-29 07:14:28.607916: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1745925268.630763   10212 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1745925268.637633   10212 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2025-04-29 07:14:28.660683: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

[2]:
```
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()
```

[3]:
```
X = np.vstack((X_train, X_test))
y = np.hstack((y_train, y_test))
```

[4]:
```
feature_names = [
    'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
    'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'
]
```

[5]:
```
df = pd.DataFrame(X, columns=feature_names)
df['Price'] = y
```

[6]:
```
print(df.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS   RAD    TAX  \
0  1.23247   0.0   8.14   0.0  0.538  6.142   91.7  3.9769   4.0  307.0
1  0.02177  82.5   2.03   0.0  0.415  7.610   15.7  6.2700   2.0  348.0
2  4.89822   0.0  18.10   0.0  0.631  4.970  100.0  1.3325  24.0  666.0
3  0.03961   0.0   5.19   0.0  0.515  6.037   34.5  5.9853   5.0  224.0
4  3.69311   0.0  18.10   0.0  0.713  6.376   88.4  2.5671  24.0  666.0

   PTRATIO       B  LSTAT  Price
0     21.0  396.90  18.72   15.2
1     14.7  395.38   3.11   42.3
2     20.2  375.52   3.26   50.0
3     20.2  396.90   8.01   21.1
4     20.2  391.43  14.65   17.7
```

[7]:
```
# Features and target variable
X = df.drop('Price', axis=1).values
y = df['Price'].values
```

[8]:
```
# Standardize the data (important for neural networks)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[9]:
```
# Build the Deep Neural Network model for Linear Regression
model = models.Sequential([
    layers.Dense(64, activation='relu', input_dim=X_train.shape[1]),
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)  # Output layer with 1 neuron for regression output (no␣
 ↪activation function)
])
```

```
/home/anil/.local/lib/python3.10/site-
packages/keras/src/layers/core/dense.py:87: 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)
```

```
W0000 00:00:1745925271.798467   10212 gpu_device.cc:2344] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices…
```

[10]:
```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

[11]:
```
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32,␣
 ↪validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/10
13/13             1s 14ms/step -
loss: 603.2776 - val_loss: 604.9301
Epoch 2/10
13/13             0s 4ms/step - loss:
565.3013 - val_loss: 568.8455
Epoch 3/10
13/13             0s 4ms/step - loss:
505.8569 - val_loss: 499.5744
Epoch 4/10
13/13             0s 3ms/step - loss:
446.8285 - val_loss: 370.1459
Epoch 5/10
13/13             0s 3ms/step - loss:
306.5861 - val_loss: 182.2021
Epoch 6/10
13/13             0s 4ms/step - loss:
130.7779 - val_loss: 62.5338
Epoch 7/10
13/13             0s 4ms/step - loss:
56.3639 - val_loss: 48.4146
Epoch 8/10
13/13             0s 4ms/step - loss:
39.7146 - val_loss: 36.6643
Epoch 9/10
13/13             0s 4ms/step - loss:
29.9657 - val_loss: 30.5572
Epoch 10/10
13/13             0s 3ms/step - loss:
24.9629 - val_loss: 27.6493
```

[12]:
```
# Evaluate the model on the test data
test_loss = model.evaluate(X_test, y_test)
```

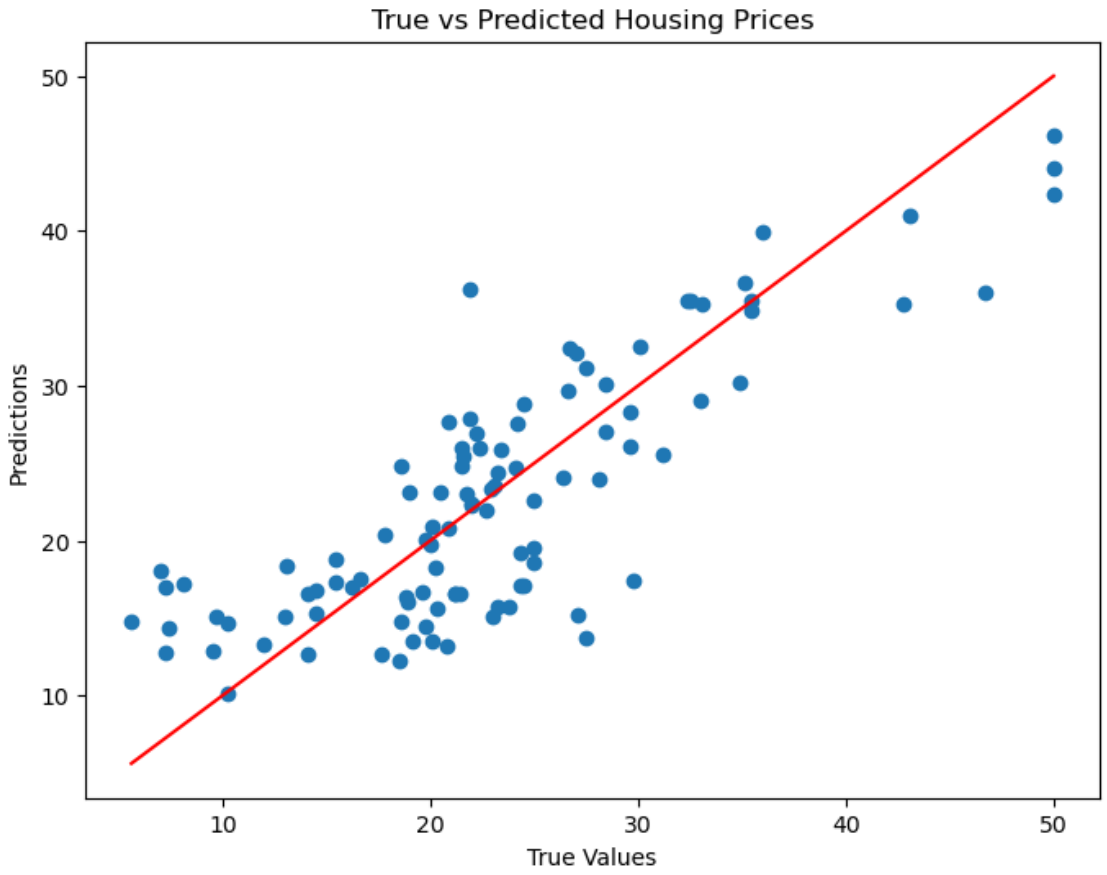
```
print(f"Test Loss (Mean Squared Error): {test_loss}")
```

```
4/4             0s 2ms/step - loss:
25.3579
Test Loss (Mean Squared Error): 27.649261474609375
```

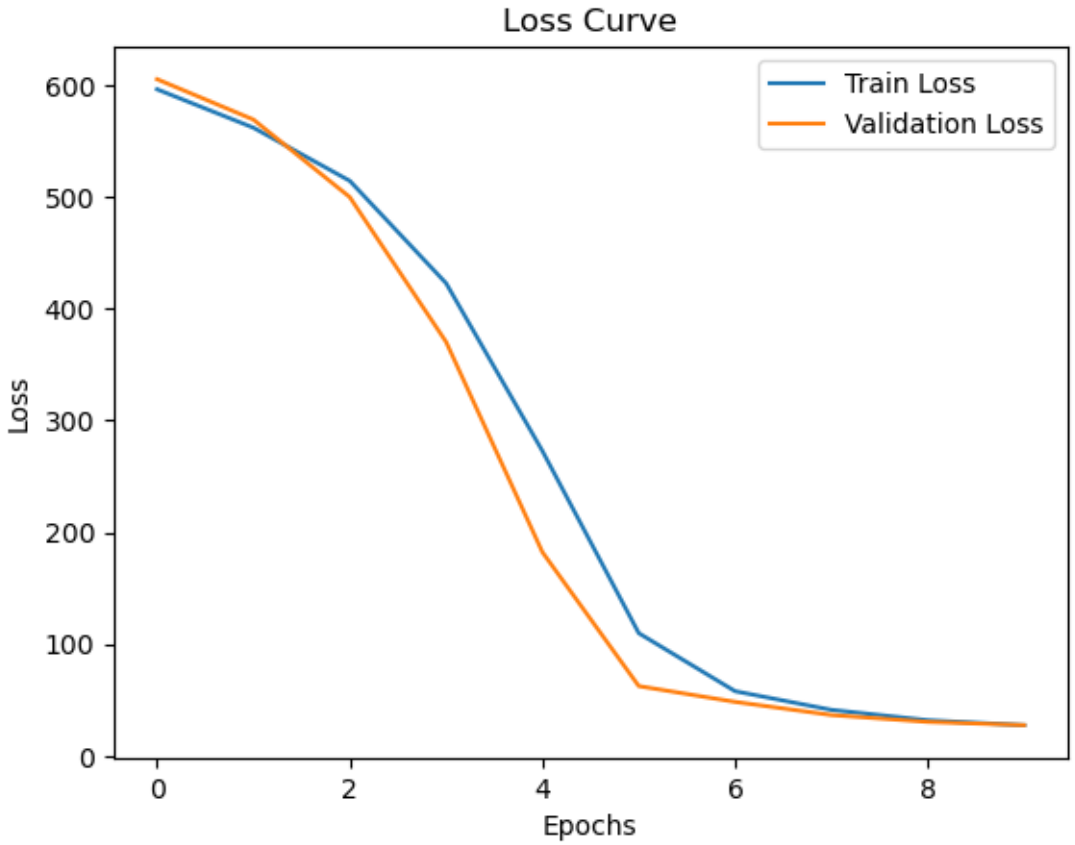
[13]:
```
# Make predictions
predictions = model.predict(X_test)
```

```
4/4             0s 14ms/step
```

[14]:
```
# Compare the predictions to the actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red') ␣
 ↪# 45-degree line
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True vs Predicted Housing Prices')
plt.show()
```



[15]:
```
# Optionally, plot the loss curve
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

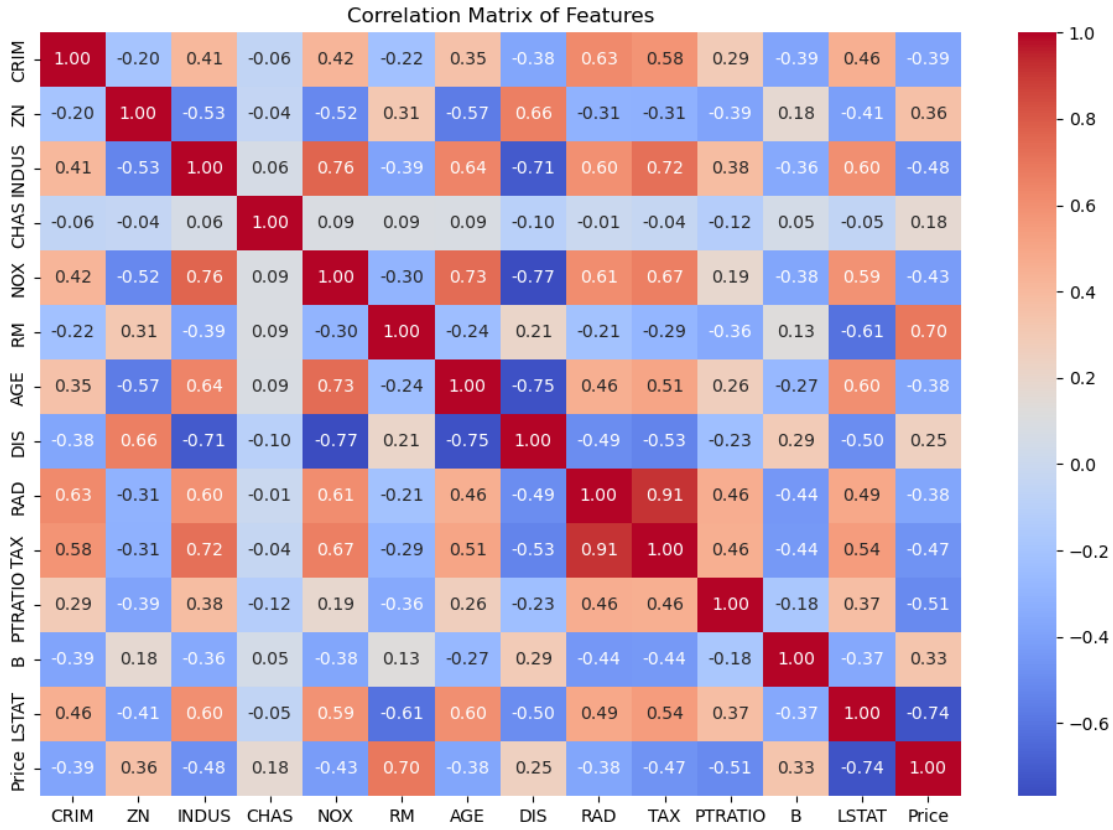


[16]:
```
import seaborn as sns
print("\nCorrelation matrix:")
correlation_matrix = df.corr()
print(correlation_matrix)

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Features')
plt.show()
```

Correlation matrix:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955

	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
CRIM	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621	-0.388305
ZN	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995	0.360445
INDUS	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603800	-0.483725
CHAS	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053929	0.175260
NOX	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590879	-0.427321
RM	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613808	0.695360
AGE	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602339	-0.376955
DIS	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496996	0.249929
RAD	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488676	-0.381626
TAX	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543993	-0.468536
PTRATIO	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374044	-0.507787
B	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.366087	0.333461
LSTAT	-0.496996	0.488676	0.543993	0.374044	-0.366087	1.000000	-0.737663
Price	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.737663	1.000000



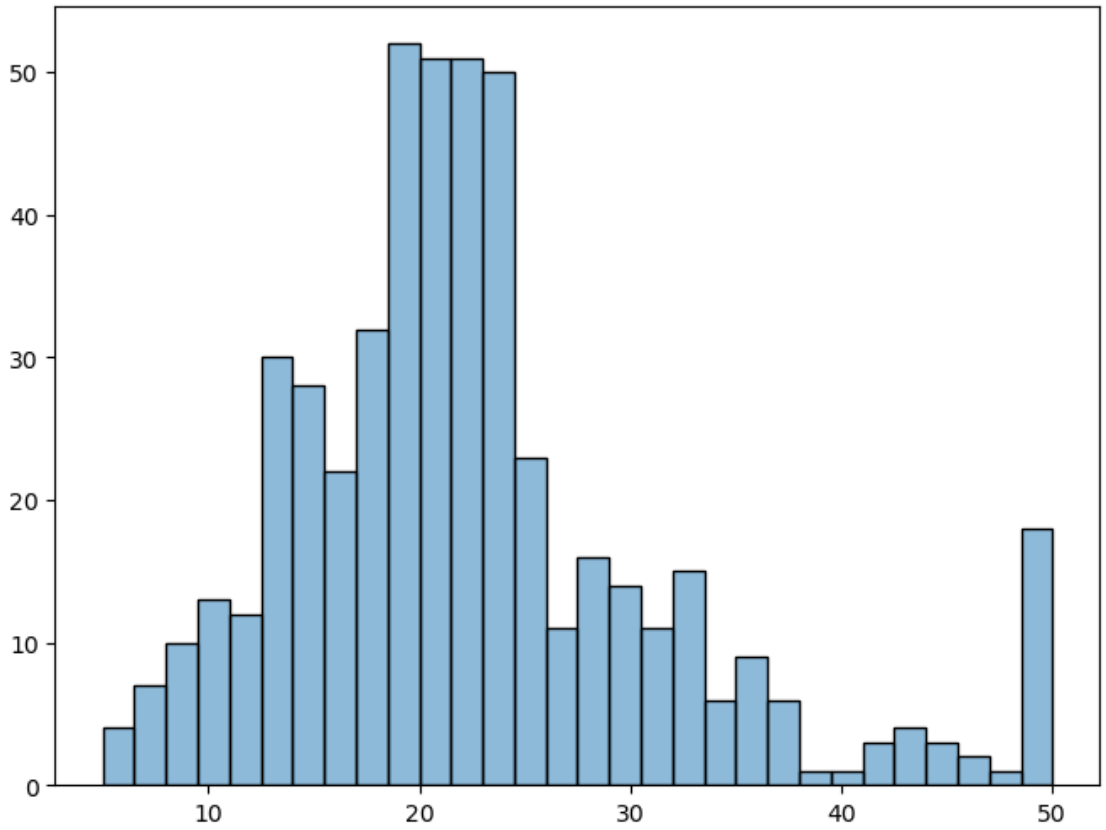
[17]:
```
# Visualizing the distribution of the target variable (Price)
plt.figure(figsize=(8, 6))
sns.histplot(df['Price'], kde=True, bins=30)
plt.title('Distribution of Housing Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[17], line 3
      1 # Visualizing the distribution of the target variable (Price)
      2 plt.figure(figsize=(8, 6))
----> 3 sns.histplot(df['Price'], kde=True, bins=30)
      4 plt.title('Distribution of Housing Prices')
      5 plt.xlabel('Price')

File ~/.local/lib/python3.10/site-packages/seaborn/distributions.py:1416, in␣
 ↪histplot(data, x, y, hue, weights, stat, bins, binwidth, binrange, discrete,␣
 ↪cumulative, common_bins, common_norm, multiple, element, fill, shrink, kde,␣
 ↪kde_kws, line_kws, thresh, pthresh, pmax, cbar, cbar_ax, cbar_kws, palette,␣
 ↪hue_order, hue_norm, color, log_scale, legend, ax, **kwargs)
```

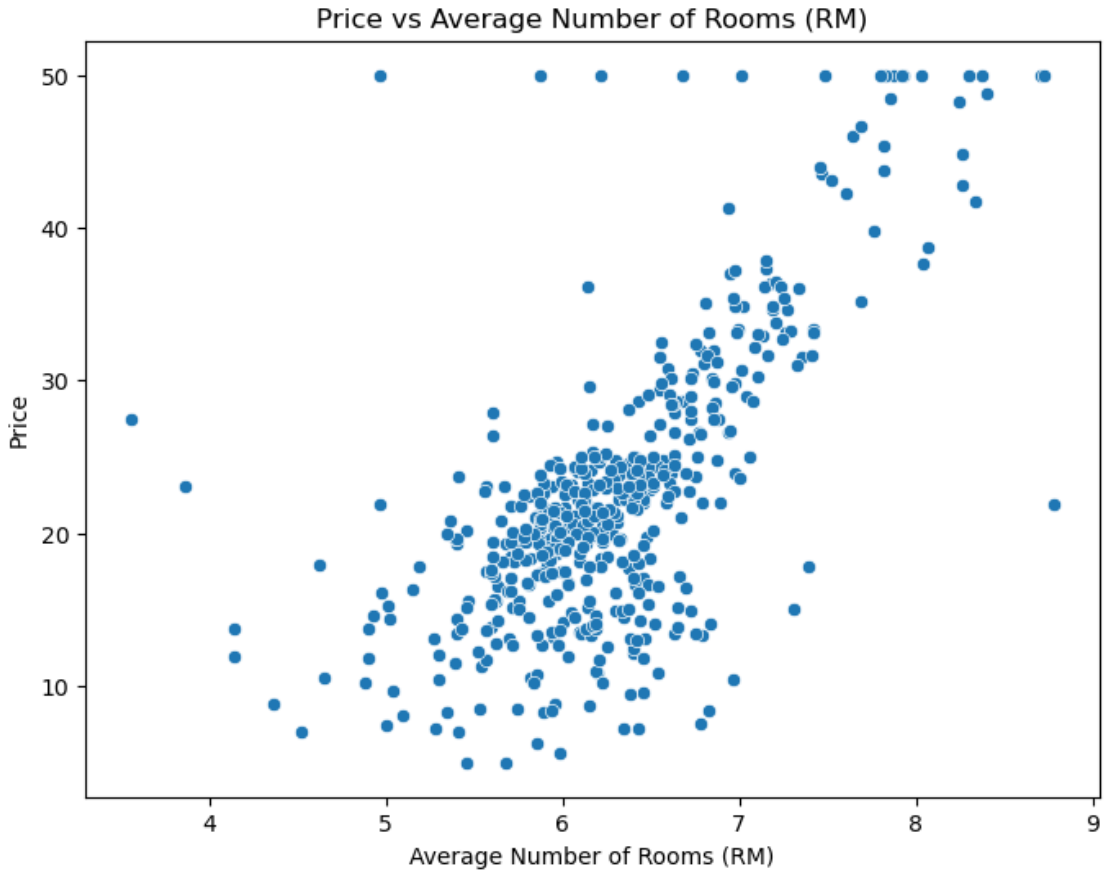
```
   1405 estimate_kws = dict(
   1406     stat=stat,
   1407     bins=bins,
   (…)
   1411     cumulative=cumulative,
   1412 )
   1414 if p.univariate:
-> 1416     p.plot_univariate_histogram(
   1417         multiple=multiple,
   1418         element=element,
   1419         fill=fill,
   1420         shrink=shrink,
   1421         common_norm=common_norm,
   1422         common_bins=common_bins,
   1423         kde=kde,
   1424         kde_kws=kde_kws,
   1425         color=color,
   1426         legend=legend,
   1427         estimate_kws=estimate_kws,
   1428         line_kws=line_kws,
   1429         **kwargs,
   1430     )
   1432 else:
   1434     p.plot_bivariate_histogram(
   1435         common_bins=common_bins,
   1436         common_norm=common_norm,
   (…)
   1446         **kwargs,
   1447     )

File ~/.local/lib/python3.10/site-packages/seaborn/distributions.py:651, in␣
 ↪_DistributionPlotter.plot_univariate_histogram(self, multiple, element, fill,
 ↪common_norm, common_bins, shrink, kde, kde_kws, color, legend, line_kws,␣
 ↪estimate_kws, **plot_kws)
    648     sticky_x, sticky_y = (0, np.inf), None
    650 line_kws["color"] = to_rgba(sub_color, 1)
--> 651 line, = ax.plot(
    652     *line_args, **line_kws,
    653 )
    655 if sticky_x is not None:
    656     line.sticky_edges.x[:] = sticky_x

File /usr/lib/python3/dist-packages/matplotlib/axes/_axes.py:1632, in Axes.
 ↪plot(self, scalex, scaley, data, *args, **kwargs)
   1390 """
   1391 Plot y versus x as lines and/or markers.
   1392
   (…)
```

```
   1629 (``'green'``) or hex strings (``'#008000'``).
   1630 """
   1631 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1632 lines = [*self._get_lines(*args, data=data, **kwargs)]
   1633 for line in lines:
   1634     self.add_line(line)

File /usr/lib/python3/dist-packages/matplotlib/axes/_base.py:312, in␣
 ↪_process_plot_var_args.__call__(self, data, *args, **kwargs)
    310     this += args[0],
    311     args = args[1:]
--> 312 yield from self._plot_args(this, kwargs)

File /usr/lib/python3/dist-packages/matplotlib/axes/_base.py:487, in␣
 ↪_process_plot_var_args._plot_args(self, tup, kwargs, return_kwargs)
    484         kw[prop_name] = val
    486 if len(xy) == 2:
--> 487     x = _check_1d(xy[0])
    488     y = _check_1d(xy[1])
    489 else:

File /usr/lib/python3/dist-packages/matplotlib/cbook/__init__.py:1327, in␣
 ↪_check_1d(x)
   1321 with warnings.catch_warnings(record=True) as w:
   1322     warnings.filterwarnings(
   1323         "always",
   1324         category=Warning,
   1325         message='Support for multi-dimensional indexing')
-> 1327     ndim = x[:, None].ndim
   1328     # we have definitely hit a pandas index or series object
   1329     # cast to a numpy array.
   1330     if len(w) > 0:

File ~/.local/lib/python3.10/site-packages/pandas/core/indexes/base.py:5419, in␣
 ↪Index.__getitem__(self, key)
   5417 # Because we ruled out integer above, we always get an arraylike here
   5418 if result.ndim > 1:
-> 5419     disallow_ndim_indexing(result)
   5421 # NB: Using _constructor._simple_new would break if MultiIndex
   5422 #  didn't override __getitem__
   5423 return self._constructor._simple_new(result, name=self._name)

File ~/.local/lib/python3.10/site-packages/pandas/core/indexers/utils.py:341, i␣
 ↪disallow_ndim_indexing(result)
    333 """
    334 Helper function to disallow multi-dimensional indexing on 1D Series/
 ↪Index.
    335
```

```
    (…)
    338 in GH#30588.
    339 """
    340 if np.ndim(result) > 1:
--> 341     raise ValueError(
    342         "Multi-dimensional indexing (e.g. `obj[:, None]`) is no longer
    343         "supported. Convert to a numpy array before indexing instead."
    344     )

ValueError: Multi-dimensional indexing (e.g. `obj[:, None]`) is no longer␣
 ↪supported. Convert to a numpy array before indexing instead.
```



[18]:
```
# Visualizing the relationship between the most correlated feature and Price
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['RM'], y=df['Price'])
plt.title('Price vs Average Number of Rooms (RM)')
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('Price')
plt.show()
```


Price vs Average Number of Rooms (RM)
Price
Average Number of Rooms (RM)
50
40
30
20
10
4
5
6
7
8
9

[ ]: