mlp_hw_cmpe188

April 27, 2025

Worked with:

- Trevor Mathisen
- Viet Nguyen

1 Redo MLR HW with Boston dataset

[1]:

```
from pandas import read_csv, DataFrame, Series
from pandas.plotting import scatter_matrix
from numpy import set_printoptions, argmax, isnan, nan, mean, random
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, Normalizer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.feature_selection import RFE
import tensorflow as tf
import numpy as np
import pandas as pd
print(tf.__version__)
```

```
2025-04-27 12:45:48.873044: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-04-27 12:45:48.883223: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] 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 00:00:1745783148.895329  324049 cuda_dnn.cc:8579] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
```

```
E0000 00:00:1745783148.898866  324049 cuda_blas.cc:1407] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
W0000 00:00:1745783148.907495  324049 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1745783148.907507  324049 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1745783148.907508  324049 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
W0000 00:00:1745783148.907509  324049 computation_placer.cc:177] computation
placer already registered. Please check linkage and avoid linking the same
target more than once.
2025-04-27 12:45:48.910276: 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 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

```
2.19.0
```

[2]:
```python
filename = 'boston.csv'
data = read_csv(filename)
set_printoptions(precision=3)
data = data.drop('index', axis=1)
print(data.head(5))
print(data.isnull().sum())
print(data.shape)
# Display unique values in each column
for col in data.columns:
    unique_values = data[col].unique()
    print(f"Unique values in '{col}': {unique_values}")
```

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

    black  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
crim       0
```

```
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
(506, 14)
Unique values in 'crim': [6.320e-03 2.731e-02 2.729e-02 3.237e-02 6.905e-02
2.985e-02 8.829e-02
 1.446e-01 2.112e-01 1.700e-01 2.249e-01 1.175e-01 9.378e-02 6.298e-01
 6.380e-01 6.274e-01 1.054e+00 7.842e-01 8.027e-01 7.258e-01 1.252e+00
 8.520e-01 1.232e+00 9.884e-01 7.503e-01 8.405e-01 6.719e-01 9.558e-01
 7.730e-01 1.002e+00 1.131e+00 1.355e+00 1.388e+00 1.152e+00 1.613e+00
 6.417e-02 9.744e-02 8.014e-02 1.751e-01 2.763e-02 3.359e-02 1.274e-01
 1.415e-01 1.594e-01 1.227e-01 1.714e-01 1.884e-01 2.293e-01 2.539e-01
 2.198e-01 8.873e-02 4.337e-02 5.360e-02 4.981e-02 1.360e-02 1.311e-02
 2.055e-02 1.432e-02 1.545e-01 1.033e-01 1.493e-01 1.717e-01 1.103e-01
 1.265e-01 1.951e-02 3.584e-02 4.379e-02 5.789e-02 1.355e-01 1.282e-01
 8.826e-02 1.588e-01 9.164e-02 1.954e-01 7.896e-02 9.512e-02 1.015e-01
 8.707e-02 5.646e-02 8.387e-02 4.113e-02 4.462e-02 3.659e-02 3.551e-02
 5.059e-02 5.735e-02 5.188e-02 7.151e-02 5.660e-02 5.302e-02 4.684e-02
 3.932e-02 4.203e-02 2.875e-02 4.294e-02 1.220e-01 1.150e-01 1.208e-01
 8.187e-02 6.860e-02 1.487e-01 1.143e-01 2.288e-01 2.116e-01 1.396e-01
 1.326e-01 1.712e-01 1.312e-01 1.280e-01 2.636e-01 1.079e-01 1.008e-01
 1.233e-01 2.221e-01 1.423e-01 1.713e-01 1.316e-01 1.510e-01 1.306e-01
 1.448e-01 6.899e-02 7.165e-02 9.299e-02 1.504e-01 9.849e-02 1.690e-01
 3.874e-01 2.591e-01 3.254e-01 8.812e-01 3.401e-01 1.193e+00 5.900e-01
 3.298e-01 9.762e-01 5.578e-01 3.226e-01 3.523e-01 2.498e-01 5.445e-01
 2.909e-01 1.629e+00 3.321e+00 4.097e+00 2.780e+00 2.379e+00 2.155e+00
 2.369e+00 2.331e+00 2.734e+00 1.657e+00 1.496e+00 1.127e+00 2.149e+00
 1.414e+00 3.535e+00 2.447e+00 1.224e+00 1.343e+00 1.425e+00 1.273e+00
 1.463e+00 1.834e+00 1.519e+00 2.242e+00 2.924e+00 2.010e+00 1.800e+00
 2.300e+00 2.450e+00 1.207e+00 2.314e+00 1.391e-01 9.178e-02 8.447e-02
 6.664e-02 7.022e-02 5.425e-02 6.642e-02 5.780e-02 6.588e-02 6.888e-02
 9.103e-02 1.001e-01 8.308e-02 6.047e-02 5.602e-02 7.875e-02 1.258e-01
 8.370e-02 9.068e-02 6.911e-02 8.664e-02 2.187e-02 1.439e-02 1.381e-02
 4.011e-02 4.666e-02 3.768e-02 3.150e-02 1.778e-02 3.445e-02 2.177e-02
 3.510e-02 2.009e-02 1.364e-01 2.297e-01 2.520e-01 1.359e-01 4.357e-01
 1.745e-01 3.758e-01 2.172e-01 1.405e-01 2.895e-01 1.980e-01 4.560e-02
 7.013e-02 1.107e-01 1.143e-01 3.581e-01 4.077e-01 6.236e-01 6.147e-01
```

```
 3.153e-01 5.269e-01 3.821e-01 4.124e-01 2.982e-01 4.418e-01 5.370e-01
 4.630e-01 5.753e-01 3.315e-01 4.479e-01 3.305e-01 5.206e-01 5.118e-01
 8.244e-02 9.252e-02 1.133e-01 1.061e-01 1.029e-01 1.276e-01 2.061e-01
 1.913e-01 3.398e-01 1.966e-01 1.644e-01 1.907e-01 1.403e-01 2.141e-01
 8.221e-02 3.689e-01 4.819e-02 3.548e-02 1.538e-02 6.115e-01 6.635e-01
 6.566e-01 5.401e-01 5.341e-01 5.201e-01 8.253e-01 5.501e-01 7.616e-01
 7.857e-01 5.783e-01 5.405e-01 9.065e-02 2.992e-01 1.621e-01 1.146e-01
 2.219e-01 5.644e-02 9.604e-02 1.047e-01 6.127e-02 7.978e-02 2.104e-01
 3.578e-02 3.705e-02 6.129e-02 1.501e-02 9.060e-03 1.096e-02 1.965e-02
 3.871e-02 4.590e-02 4.297e-02 3.502e-02 7.886e-02 3.615e-02 8.265e-02
 8.199e-02 1.293e-01 5.372e-02 1.410e-01 6.466e-02 5.561e-02 4.417e-02
 3.537e-02 9.266e-02 1.000e-01 5.515e-02 5.479e-02 7.503e-02 4.932e-02
 4.930e-01 3.494e-01 2.635e+00 7.904e-01 2.617e-01 2.694e-01 3.692e-01
 2.536e-01 3.183e-01 2.452e-01 4.020e-01 4.755e-01 1.676e-01 1.816e-01
 3.511e-01 2.839e-01 3.411e-01 1.919e-01 3.035e-01 2.410e-01 6.617e-02
 6.724e-02 4.544e-02 5.023e-02 3.466e-02 5.083e-02 3.738e-02 3.961e-02
 3.427e-02 3.041e-02 3.306e-02 5.497e-02 6.151e-02 1.301e-02 2.498e-02
 2.543e-02 3.049e-02 3.113e-02 6.162e-02 1.870e-02 2.899e-02 6.211e-02
 7.950e-02 7.244e-02 1.709e-02 4.301e-02 1.066e-01 8.983e+00 3.850e+00
 5.202e+00 4.261e+00 4.542e+00 3.837e+00 3.678e+00 4.222e+00 3.474e+00
 4.556e+00 3.697e+00 1.352e+01 4.898e+00 5.670e+00 6.539e+00 9.232e+00
 8.267e+00 1.111e+01 1.850e+01 1.961e+01 1.529e+01 9.823e+00 2.365e+01
 1.787e+01 8.898e+01 1.587e+01 9.187e+00 7.992e+00 2.008e+01 1.681e+01
 2.439e+01 2.260e+01 1.433e+01 8.152e+00 6.962e+00 5.293e+00 1.158e+01
 8.645e+00 1.336e+01 8.717e+00 5.872e+00 7.672e+00 3.835e+01 9.917e+00
 2.505e+01 1.424e+01 9.596e+00 2.480e+01 4.153e+01 6.792e+01 2.072e+01
 1.195e+01 7.404e+00 1.444e+01 5.114e+01 1.405e+01 1.881e+01 2.866e+01
 4.575e+01 1.808e+01 1.083e+01 2.594e+01 7.353e+01 1.181e+01 1.109e+01
 7.023e+00 1.205e+01 7.050e+00 8.792e+00 1.586e+01 1.225e+01 3.766e+01
 7.367e+00 9.339e+00 8.492e+00 1.006e+01 6.444e+00 5.581e+00 1.391e+01
 1.116e+01 1.442e+01 1.518e+01 1.368e+01 9.391e+00 2.205e+01 9.724e+00
 5.666e+00 9.967e+00 1.280e+01 1.067e+01 6.288e+00 9.925e+00 9.329e+00
 7.526e+00 6.718e+00 5.441e+00 5.090e+00 8.248e+00 9.514e+00 4.752e+00
 4.669e+00 8.201e+00 7.752e+00 6.801e+00 4.812e+00 3.693e+00 6.655e+00
 5.821e+00 7.839e+00 3.164e+00 3.775e+00 4.422e+00 1.558e+01 1.308e+01
 4.349e+00 4.038e+00 3.569e+00 4.647e+00 8.056e+00 6.393e+00 4.871e+00
 1.502e+01 1.023e+01 5.824e+00 5.708e+00 5.731e+00 2.818e+00 2.379e+00
 3.674e+00 5.692e+00 4.836e+00 1.509e-01 1.834e-01 2.075e-01 1.057e-01
 1.113e-01 1.733e-01 2.796e-01 1.790e-01 2.896e-01 2.684e-01 2.391e-01
 1.778e-01 2.244e-01 6.263e-02 4.527e-02 6.076e-02 1.096e-01 4.741e-02]
Unique values in 'zn': [ 18.    0.   12.5  75.   21.   90.   85.  100.   25.
17.5  80.   28.
  45.   60.   95.   82.5  30.   22.   20.   40.   55.   52.5  70.   34.
  33.   35. ]
Unique values in 'indus': [ 2.31  7.07  2.18  7.87  8.14  5.96  2.95  6.91  5.64
4.    1.22  0.74
  1.32  5.13  1.38  3.37  6.07 10.81 12.83  4.86  4.49  3.41 15.04  2.89
  8.56 10.01 25.65 21.89 19.58  4.05  2.46  3.44  2.93  0.46  1.52  1.47
```

```
  2.03  2.68 10.59 13.89  6.2   4.93  5.86  3.64  3.75  3.97  6.96  6.41
  3.33  1.21  2.97  2.25  1.76  5.32  4.95 13.92  2.24  6.09  9.9   7.38
  3.24  6.06  5.19  1.89  3.78  4.39  4.15  2.01  1.25  1.69  2.02  1.91
 18.1  27.74  9.69 11.93]
Unique values in 'chas': [0 1]
Unique values in 'nox': [0.538 0.469 0.458 0.524 0.499 0.428 0.448 0.439 0.41
0.403 0.411 0.453
 0.416 0.398 0.409 0.413 0.437 0.426 0.449 0.489 0.464 0.445 0.52  0.547
 0.581 0.624 0.871 0.605 0.51  0.488 0.401 0.422 0.404 0.415 0.55  0.507
 0.504 0.431 0.392 0.394 0.647 0.575 0.447 0.443 0.4   0.389 0.385 0.405
 0.433 0.472 0.544 0.493 0.46  0.438 0.515 0.442 0.518 0.484 0.429 0.435
 0.77  0.718 0.631 0.668 0.671 0.7   0.693 0.659 0.597 0.679 0.614 0.584
 0.713 0.74  0.655 0.58  0.532 0.583 0.609 0.585 0.573]
Unique values in 'rm': [6.575 6.421 7.185 6.998 7.147 6.43  6.012 6.172 5.631
6.004 6.377 6.009
 5.889 5.949 6.096 5.834 5.935 5.99  5.456 5.727 5.57  5.965 6.142 5.813
 5.924 5.599 6.047 6.495 6.674 5.713 6.072 5.95  5.701 5.933 5.841 5.85
 5.966 6.595 7.024 6.77  6.169 6.211 6.069 5.682 5.786 6.03  5.399 5.602
 5.963 6.115 6.511 5.998 5.888 7.249 6.383 6.816 6.145 5.927 5.741 6.456
 6.762 7.104 6.29  5.787 5.878 5.594 5.885 6.417 5.961 6.065 6.245 6.273
 6.286 6.279 6.14  6.232 5.874 6.727 6.619 6.302 6.167 6.389 6.63  6.015
 6.121 7.007 7.079 6.405 6.442 6.249 6.625 6.163 8.069 7.82  7.416 6.781
 6.137 5.851 5.836 6.127 6.474 6.229 6.195 6.715 5.913 6.092 6.254 5.928
 6.176 6.021 5.872 5.731 5.87  5.856 5.879 5.986 5.613 5.693 6.431 5.637
 6.458 6.326 6.372 5.822 5.757 6.335 5.942 6.454 5.857 6.151 6.174 5.019
 5.403 5.468 4.903 6.13  5.628 4.926 5.186 5.597 6.122 5.404 5.012 5.709
 6.129 6.152 5.272 6.943 6.066 6.51  6.25  7.489 7.802 8.375 5.854 6.101
 7.929 5.877 6.319 6.402 5.875 5.88  5.572 6.416 5.859 6.546 6.02  6.315
 6.86  6.98  7.765 6.144 7.155 6.563 5.604 6.153 7.831 6.782 6.556 6.951
 6.739 7.178 6.8   6.604 7.875 7.287 7.107 7.274 6.975 7.135 6.162 7.61
 7.853 8.034 5.891 5.783 6.064 5.344 5.96  5.807 6.375 5.412 6.182 6.642
 5.951 6.373 6.164 6.879 6.618 8.266 8.725 8.04  7.163 7.686 6.552 5.981
 7.412 8.337 8.247 6.726 6.086 6.631 7.358 6.481 6.606 6.897 6.095 6.358
 6.393 5.593 5.605 6.108 6.226 6.433 6.718 6.487 6.438 6.957 8.259 5.876
 7.454 8.704 7.333 6.842 7.203 7.52  8.398 7.327 7.206 5.56  7.014 8.297
 7.47  5.92  6.24  6.538 7.691 6.758 6.854 7.267 6.826 6.482 6.812 6.968
 7.645 7.923 7.088 6.453 6.23  6.209 6.565 6.861 7.148 6.678 6.549 5.79
 6.345 7.041 6.871 6.59  6.982 7.236 6.616 7.42  6.849 6.635 5.972 4.973
 6.023 6.266 6.567 5.705 5.914 5.782 6.382 6.113 6.426 6.376 6.041 5.708
 6.415 6.312 6.083 5.868 6.333 5.706 6.031 6.316 6.31  6.037 5.869 5.895
 6.059 5.985 5.968 7.241 6.54  6.696 6.874 6.014 5.898 6.516 6.939 6.49
 6.579 5.884 6.728 5.663 5.936 6.212 6.395 6.112 6.398 6.251 5.362 5.803
 8.78  3.561 4.963 3.863 4.97  6.683 7.016 6.216 4.906 4.138 7.313 6.649
 6.794 6.38  6.223 6.545 5.536 5.52  4.368 5.277 4.652 5.    4.88  5.39
 6.051 5.036 6.193 5.887 6.471 5.747 5.453 5.852 5.987 6.343 6.404 5.349
 5.531 5.683 5.608 5.617 6.852 6.657 4.628 5.155 4.519 6.434 5.304 5.957
 6.824 6.411 6.006 5.648 6.103 5.565 5.896 5.837 6.202 6.348 6.833 6.425
 6.436 6.208 6.629 6.461 5.627 5.818 6.406 6.219 6.485 6.459 6.341 6.185
```

```
 6.749 6.655 6.297 7.393 6.525 5.976 6.301 6.081 6.701 6.317 6.513 5.759
 5.952 6.003 5.926 6.437 5.427 6.484 6.242 6.75  7.061 5.762 5.871 6.114
 5.905 5.454 5.414 5.093 5.983 5.707 5.67  5.794 6.019 5.569 6.027 6.593
 6.12  6.976]
Unique values in 'age': [ 65.2  78.9  61.1  45.8  54.2  58.7  66.6  96.1 100.
85.9  94.3  82.9
  39.   61.8  84.5  56.5  29.3  81.7  36.6  69.5  98.1  89.2  91.7  94.1
  85.7  90.3  88.8  94.4  87.3  82.   95.   96.9  68.2  61.4  41.5  30.2
  21.8  15.8   2.9   6.6   6.5  40.   33.8  33.3  85.5  95.3  62.   45.7
  63.   21.1  21.4  47.6  21.9  35.7  40.5  29.2  47.2  66.2  93.4  67.8
  43.4  59.5  17.8  31.1  36.8  33.   17.5   7.8   6.2   6.   45.   74.5
  53.7  33.5  70.4  32.2  46.7  48.   56.1  45.1  56.8  86.3  63.1  66.1
  73.9  53.6  28.9  77.3  57.8  69.6  76.   36.9  62.5  79.9  71.3  85.4
  87.4  90.   96.7  91.9  85.2  97.1  91.2  54.4  81.6  92.9  95.4  84.2
  88.2  72.5  82.6  73.1  69.7  84.1  97.   95.8  88.4  95.6  96.   98.8
  94.7  98.9  97.7  97.9  98.4  98.2  93.5  93.6  97.8  95.7  93.8  94.9
  97.3  88.   98.5  94.   97.4  92.6  90.8  93.9  91.8  93.   96.2  79.2
  95.2  94.6  88.5  68.7  33.1  73.4  74.4  58.4  83.3  62.2  92.2  89.8
  68.8  41.1  29.1  38.9  21.5  30.8  26.3   9.9  18.8  32.   34.1  38.3
  15.3  13.9  38.4  15.7  33.2  31.9  22.3  52.5  72.7  59.1  92.1  88.6
  53.8  32.3   9.8  42.4  56.   85.1  92.4  91.3  77.7  80.8  78.3  83.
  86.5  17.   68.1  76.9  73.3  66.5  61.5  76.5  71.6  18.5  42.2  54.3
  65.1  52.9  70.2  34.9  49.1  13.    8.9   6.8   8.4  19.1  34.2  86.9
  81.8  89.4  91.5  94.5  91.6  62.8  84.6  67.   52.6  42.1  16.3  51.8
  32.9  42.8  49.   27.6  32.1  64.5  37.2  49.7  24.8  20.8  31.5  31.3
  45.6  22.9  27.9  27.7  23.4  18.4  42.3  51.   58.   20.1  10.   47.4
  40.4  17.7  58.1  71.9  70.3  82.5  76.7  37.8  52.8  90.4  82.8  83.2
  71.7  67.2  58.8  52.3  49.9  74.3  40.1  14.7  43.7  25.8  17.2  28.4
  23.3  38.1  38.5  34.5  46.3  59.6  37.3  45.4  58.5  49.3  59.7  56.4
  28.1  48.5  29.7  44.4  35.9  36.1  19.5  91.   83.4  81.3  91.1  89.
  87.9  91.4  96.8  97.5  89.6  93.3  99.1  89.5  77.8  89.1  87.6  70.6
  78.7  78.1  86.1  74.8  97.2  96.6  94.8  96.4  98.7  98.3  99.3  80.3
  83.7  84.4  89.9  65.4  48.2  84.7  71.   56.7  84.   90.7  75.   67.6
  64.7  74.9  77.   40.3  41.9  51.9  79.8  53.2  92.7  98.   83.5  54.
  42.6  28.8  72.9  65.3  73.5  79.7  69.1  89.3]
Unique values in 'dis': [ 4.09   4.967  6.062  5.561  5.95   6.082  6.592  6.347
6.227  5.451
  4.707  4.462  4.499  4.258  3.796  3.798  4.012  3.977  4.095  4.4
  4.455  4.682  4.453  4.455  4.239  4.233  4.175  3.99   3.787  3.76
  3.36   3.378  3.934  3.847  5.401  5.721  5.1    5.689  5.87   6.088
  6.815  7.32   8.697  9.188  8.325  7.815  6.932  7.225  6.819  7.226
  7.981  9.223  6.612  6.498  5.287  4.252  4.503  4.052  4.09   5.014
  5.401  4.779  4.438  4.427  3.748  3.422  3.414  3.092  3.092  3.666
  3.615  3.495  2.778  2.856  2.715  2.421  2.107  2.211  2.122  2.433
  2.545  2.678  2.353  2.548  2.256  2.463  2.73   2.747  2.478  2.759
  2.258  2.197  2.087  1.944  2.006  1.993  1.757  1.788  1.812  1.98
  2.119  2.271  2.327  2.47   2.346  2.111  1.967  1.85   1.669  1.669
  1.612  1.439  1.322  1.412  1.346  1.419  1.517  1.461  1.53   1.526
```

```
  1.618  1.592  1.61   1.623  1.749  1.746  1.736  1.877  1.757  1.766
  1.798  1.971  2.041  2.162  2.422  2.283  2.046  2.426  2.1    2.263
  2.389  2.596  2.646  2.702  3.132  3.555  3.317  2.915  2.829  2.741
  2.598  2.701  2.847  2.988  3.28   3.199  3.789  4.567  6.48   6.22
  5.648  7.309  7.653  6.27   5.118  3.945  4.355  4.239  3.875  3.877
  3.665  3.653  3.587  3.112  3.421  2.889  3.363  2.862  3.048  3.272
  2.894  3.216  3.375  3.671  3.838  3.652  4.148  6.19   6.336  7.035
  7.955  8.056  7.827  7.397  8.907  9.22   1.801  1.895  2.011  2.112
  2.14   2.288  2.079  1.93   1.986  2.133  2.422  2.872  3.917  4.429
  4.367  4.078  4.267  4.787  4.863  4.14   4.101  4.695  5.245  5.212
  5.885  7.307  9.089  7.317  5.117  5.503  5.96   6.32   7.828  5.492
  4.022  3.37   3.099  3.183  3.103  2.519  2.64   2.834  3.263  3.602
  3.945  3.999  4.032  3.533  4.002  4.54   4.721  5.416  5.215  5.874
  6.641  6.458  5.985  5.231  5.615  4.812  7.038  6.267  5.732  6.465
  8.014  8.535  8.344  8.792 10.71  12.127 10.586  2.122  2.505  2.723
  2.509  2.518  2.296  2.104  1.905  1.613  1.752  1.511  1.333  1.357
  1.202  1.169  1.13   1.174  1.137  1.316  1.345  1.358  1.386  1.417
  1.519  1.58   1.533  1.44   1.426  1.467  1.518  1.589  1.728  1.927
  2.168  1.77   1.791  1.782  1.726  1.677  1.633  1.49   1.5    1.589
  1.574  1.639  1.703  1.607  1.425  1.178  1.285  1.455  1.466  1.413
  1.528  1.554  1.589  1.658  1.835  1.819  1.647  1.803  1.794  1.859
  1.875  1.951  2.022  2.063  1.91   1.998  1.863  1.936  1.968  2.053
  2.088  2.2    2.316  2.222  2.125  2.003  1.914  1.821  1.817  1.866
  2.065  2.005  1.978  1.896  1.988  2.072  2.198  2.262  2.185  2.324
  2.355  2.368  2.453  2.496  2.436  2.581  2.779  2.783  2.717  2.598
  2.567  2.734  2.802  2.963  3.067  2.872  2.54   2.908  2.824  3.033
  3.099  2.897  2.533  2.43   2.206  2.305  2.101  2.171  3.424  3.332
  3.411  4.098  3.724  3.992  3.546  3.152  1.821  1.755  1.823  1.868
  2.11   2.382  2.799  2.893  2.409  2.4    2.498  2.479  2.288  2.167
  2.389  2.505]
Unique values in 'rad': [ 1  2  3  5  4  8  6  7 24]
Unique values in 'tax': [296 242 222 311 307 279 252 233 243 469 226 313 256 284
216 337 345 305
 398 281 247 270 276 384 432 188 437 403 193 265 255 329 402 348 224 277
 300 330 315 244 264 223 254 198 285 241 293 245 289 358 304 287 430 422
 370 352 351 280 335 411 187 334 666 711 391 273]
Unique values in 'ptratio': [15.3 17.8 18.7 15.2 21.  19.2 18.3 17.9 16.8 21.1
17.3 15.1 19.7 18.6
 16.1 18.9 19.  18.5 18.2 18.  20.9 19.1 21.2 14.7 16.6 15.6 14.4 12.6
 17.  16.4 17.4 15.9 13.  17.6 14.9 13.6 16.  14.8 18.4 19.6 16.9 20.2
 15.5 18.8 22.  20.1]
Unique values in 'black': [3.969e+02 3.928e+02 3.946e+02 3.941e+02 3.956e+02
3.866e+02 3.867e+02
 3.925e+02 3.905e+02 3.800e+02 3.956e+02 3.869e+02 3.868e+02 2.890e+02
 3.909e+02 3.766e+02 3.925e+02 3.945e+02 3.943e+02 3.034e+02 3.769e+02
 3.064e+02 3.879e+02 3.802e+02 3.602e+02 3.767e+02 2.326e+02 3.588e+02
 2.483e+02 3.776e+02 3.934e+02 3.956e+02 3.854e+02 3.834e+02 3.945e+02
 3.894e+02 3.927e+02 3.956e+02 3.940e+02 3.959e+02 3.929e+02 3.907e+02
```

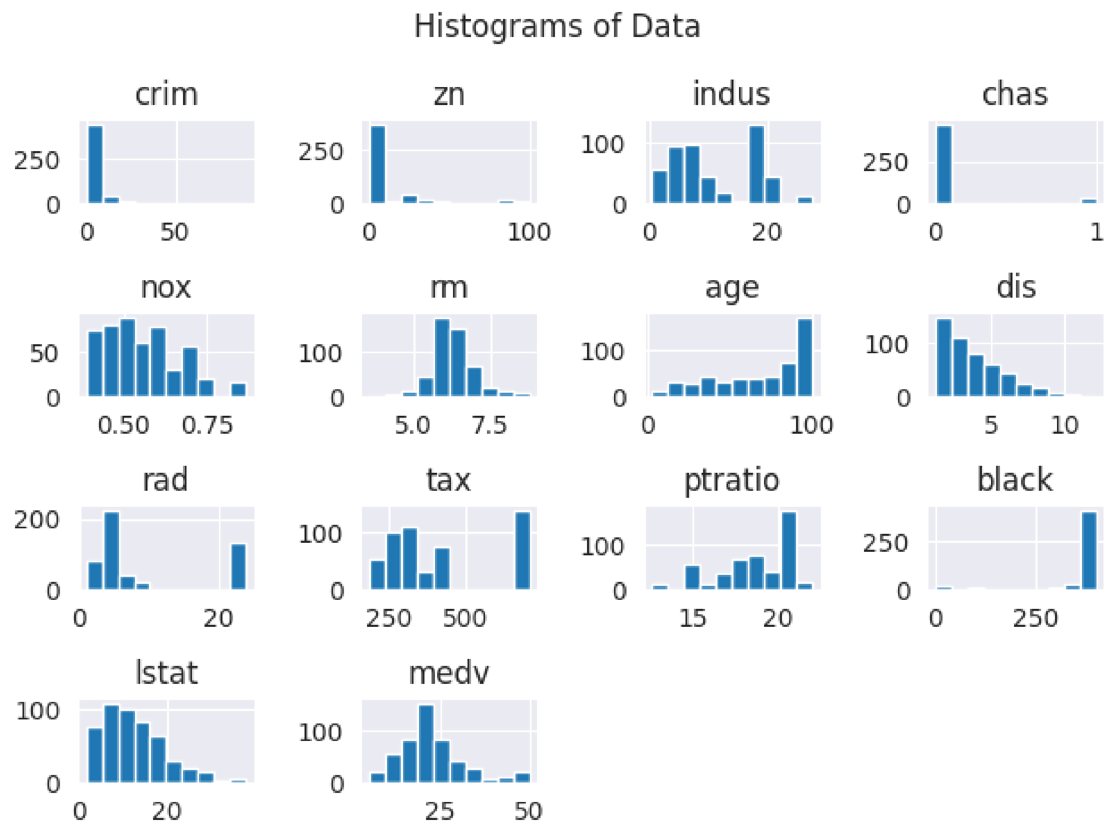
```
 3.951e+02 3.781e+02 3.956e+02 3.932e+02 3.962e+02 3.837e+02 3.769e+02
 3.909e+02 3.772e+02 3.949e+02 3.832e+02 3.737e+02 3.870e+02 3.864e+02
 3.961e+02 3.906e+02 3.923e+02 3.960e+02 3.951e+02 3.922e+02 3.936e+02
 3.950e+02 3.963e+02 3.580e+02 3.918e+02 3.935e+02 3.948e+02 7.080e+01
 3.945e+02 3.927e+02 3.941e+02 3.957e+02 3.877e+02 3.952e+02 3.912e+02
 3.935e+02 3.956e+02 3.949e+02 3.887e+02 3.449e+02 3.933e+02 3.945e+02
 3.386e+02 3.915e+02 3.891e+02 3.777e+02 3.781e+02 3.703e+02 3.794e+02
 3.850e+02 3.593e+02 3.921e+02 3.950e+02 3.858e+02 3.887e+02 2.628e+02
 3.947e+02 3.782e+02 3.941e+02 3.920e+02 3.881e+02 1.729e+02 1.693e+02
 3.917e+02 3.570e+02 3.519e+02 3.728e+02 3.416e+02 3.433e+02 2.619e+02
 3.210e+02 8.801e+01 8.863e+01 3.634e+02 3.539e+02 3.643e+02 3.389e+02
 3.744e+02 3.896e+02 3.884e+02 2.402e+02 3.693e+02 2.276e+02 2.971e+02
 3.300e+02 2.923e+02 3.481e+02 3.955e+02 3.932e+02 3.910e+02 3.913e+02
 3.910e+02 3.871e+02 3.926e+02 3.939e+02 3.828e+02 3.777e+02 3.897e+02
 3.905e+02 3.934e+02 3.767e+02 3.942e+02 3.543e+02 3.922e+02 3.843e+02
 3.938e+02 3.954e+02 3.928e+02 3.906e+02 3.949e+02 3.894e+02 3.813e+02
 3.932e+02 3.909e+02 3.858e+02 3.489e+02 3.936e+02 3.928e+02 3.937e+02
 3.917e+02 3.904e+02 3.851e+02 3.820e+02 3.874e+02 3.721e+02 3.775e+02
 3.803e+02 3.784e+02 3.761e+02 3.859e+02 3.789e+02 3.602e+02 3.768e+02
 3.901e+02 3.794e+02 3.838e+02 3.912e+02 3.946e+02 3.728e+02 3.747e+02
 3.725e+02 3.891e+02 3.902e+02 3.963e+02 3.771e+02 3.861e+02 3.929e+02
 3.952e+02 3.863e+02 3.897e+02 3.833e+02 3.919e+02 3.884e+02 3.869e+02
 3.934e+02 3.879e+02 3.924e+02 3.841e+02 3.845e+02 3.903e+02 3.913e+02
 3.886e+02 3.950e+02 3.908e+02 3.892e+02 3.934e+02 3.873e+02 3.922e+02
 3.955e+02 3.947e+02 3.717e+02 3.929e+02 3.682e+02 3.716e+02 3.909e+02
 3.958e+02 3.836e+02 3.904e+02 3.937e+02 3.934e+02 3.962e+02 3.504e+02
 3.963e+02 3.934e+02 3.957e+02 3.964e+02 3.907e+02 3.952e+02 3.962e+02
 3.911e+02 3.824e+02 3.752e+02 3.686e+02 3.940e+02 3.622e+02 3.894e+02
 3.948e+02 3.961e+02 3.947e+02 3.900e+02 3.880e+02 3.856e+02 3.646e+02
 3.924e+02 3.899e+02 3.708e+02 3.923e+02 3.845e+02 3.828e+02 3.760e+02
 3.777e+02 3.954e+02 3.907e+02 3.746e+02 3.506e+02 3.808e+02 3.530e+02
 3.546e+02 3.547e+02 3.160e+02 1.314e+02 3.755e+02 3.753e+02 3.921e+02
 3.661e+02 3.479e+02 3.630e+02 2.858e+02 3.729e+02 3.944e+02 3.784e+02
 3.920e+02 3.931e+02 3.382e+02 3.761e+02 3.295e+02 3.850e+02 3.702e+02
 3.321e+02 3.146e+02 1.794e+02 2.600e+00 3.505e+01 2.879e+01 2.110e+02
 8.827e+01 2.725e+01 2.157e+01 1.274e+02 1.645e+01 4.845e+01 3.188e+02
 3.200e+02 2.916e+02 2.520e+00 3.650e+00 7.680e+00 2.465e+01 1.882e+01
 9.673e+01 6.072e+01 8.345e+01 8.133e+01 9.795e+01 1.002e+02 1.006e+02
 1.098e+02 2.749e+01 9.320e+00 6.895e+01 3.914e+02 3.860e+02 3.867e+02
 2.405e+02 4.306e+01 3.180e+02 3.885e+02 3.042e+02 3.200e-01 3.553e+02
 3.851e+02 3.759e+02 6.680e+00 5.092e+01 1.048e+01 3.500e+00 2.722e+02
 2.552e+02 3.914e+02 3.938e+02 3.344e+02 2.201e+01 3.313e+02 3.687e+02
 3.953e+02 3.747e+02 3.526e+02 3.028e+02 3.495e+02 3.797e+02 3.833e+02
 3.931e+02 3.953e+02 3.929e+02 3.707e+02 3.886e+02 3.927e+02 3.882e+02
 3.951e+02 3.441e+02 3.184e+02 3.901e+02 3.933e+02 3.958e+02 3.920e+02]
Unique values in 'lstat': [ 4.98  9.14  4.03  2.94  5.33  5.21 12.43 19.15 29.93
17.1  20.45 13.27
 15.71  8.26 10.26  8.47  6.58 14.67 11.69 11.28 21.02 13.83 18.72 19.88
```

```
 16.3  16.51 14.81 17.28 12.8  11.98 22.6  13.04 27.71 18.35 20.34  9.68
 11.41  8.77 10.13  4.32  1.98  4.84  5.81  7.44  9.55 10.21 14.15 18.8
 30.81 16.2  13.45  9.43  5.28  8.43 14.8   4.81  5.77  3.95  6.86  9.22
 13.15 14.44  6.73  9.5   8.05  4.67 10.24  8.1  13.09  8.79  6.72  9.88
  5.52  7.54  6.78  8.94 11.97 10.27 12.34  9.1   5.29  7.22  7.51  9.62
  6.53 12.86  8.44  5.5   5.7   8.81  8.2   8.16  6.21 10.59  6.65 11.34
  4.21  3.57  6.19  9.42  7.67 10.63 13.44 12.33 16.47 18.66 14.09 12.27
 15.55 13.   10.16 16.21 17.09 10.45 15.76 12.04 10.3  15.37 13.61 14.37
 14.27 17.93 25.41 17.58 27.26 17.19 15.39 18.34 12.6  12.26 11.12 15.03
 17.31 16.96 16.9  14.59 21.32 18.46 24.16 34.41 26.82 26.42 29.29 27.8
 16.65 29.53 28.32 21.45 14.1  13.28 12.12 15.79 15.12 15.02 16.14  4.59
  6.43  7.39  1.73  1.92  3.32 11.64  9.81  3.7  12.14 11.1  11.32 14.43
 12.03 14.69  9.04  9.64 10.11  6.29  6.92  5.04  7.56  9.45  4.82  5.68
 13.98  4.45  6.68  4.56  5.39  5.1   4.69  2.87  5.03  4.38  2.97  4.08
  8.61  6.62  7.43  3.11  3.81  2.88 10.87 10.97 18.06 14.66 23.09 17.27
 23.98 16.03  9.38 29.55  9.47 13.51  9.69 17.92 10.5   9.71 21.46  9.93
  7.6   4.14  4.63  3.13  6.36  3.92  3.76 11.65  5.25  2.47 10.88  9.54
  4.73  7.37 11.38 12.4  11.22  5.19 12.5   9.16 10.15  9.52  6.56  5.9
  3.59  3.53  3.54  6.57  9.25  5.12  7.79  6.9   9.59  7.26  5.91 11.25
 14.79  3.16 13.65  6.59  7.73  2.98  6.05  4.16  7.19  4.85  3.01  7.85
  8.23 12.93  7.14  9.51  3.33  3.56  4.7   8.58 10.4   6.27 15.84  4.97
  4.74  6.07  8.67  4.86  6.93  8.93  6.47  7.53  4.54  9.97 12.64  5.98
 11.72  7.9   9.28 11.5  18.33 15.94 10.36 12.73  7.2   6.87  7.7  11.74
  6.12  5.08  6.15 12.79  7.34  9.09  7.83  6.75  8.01  9.8  10.56  8.51
  9.74  9.29  5.49  8.65  7.18  4.61 10.53 12.67  5.99  5.89  4.5   5.57
 17.6  11.48 14.19 10.19 14.64  7.12 14.   13.33  3.26  3.73  2.96  9.53
  8.88 34.77 37.97 23.24 21.24 23.69 21.78 17.21 21.08 23.6  24.56 30.63
 28.28 31.99 30.62 20.85 17.11 18.76 25.68 15.17 16.35 17.12 19.37 19.92
 30.59 29.97 26.77 20.32 20.31 19.77 27.38 22.98 23.34 12.13 26.4  19.78
 21.22 34.37 20.08 36.98 29.05 25.79 26.64 20.62 22.74 15.7  23.29 17.16
 24.39 15.69 14.52 21.52 24.08 17.64 19.69 16.22 23.27 18.05 26.45 34.02
 22.88 22.11 19.52 16.59 18.85 23.79 17.79 16.44 18.13 19.31 17.44 17.73
 16.74 18.71 19.01 16.94 16.23 14.7  16.42 14.65 13.99 10.29 13.22 14.13
 17.15 14.76 16.29 12.87 14.36 11.66 18.14 24.1  18.68 24.91 18.03 13.11
 10.74  7.74  7.01 10.42 13.34 10.58 14.98 11.45 23.97 29.68 18.07 13.35
 12.01 13.59 21.14 12.92 15.1  14.33  9.67  9.08  5.64  6.48  7.88]
Unique values in 'medv': [24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.
21.7 20.4 18.2
 19.9 23.1 17.5 20.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8 18.4 21.
 12.7 13.2 13.1 13.5 20.  24.7 30.8 34.9 26.6 25.3 21.2 19.3 14.4 19.4
 19.7 20.5 25.  23.4 35.4 31.6 23.3 18.7 16.  22.2 33.  23.5 22.  17.4
 20.9 24.2 22.8 24.1 21.4 20.8 20.3 28.  23.9 24.8 22.5 23.6 22.6 20.6
 28.4 38.7 43.8 33.2 27.5 26.5 18.6 20.1 19.5 19.8 18.8 18.5 18.3 19.2
 17.3 15.7 16.2 18.  14.3 23.  18.1 17.1 13.3 17.8 14.  13.4 11.8 13.8
 14.6 15.4 21.5 15.3 17.  41.3 24.3 27.  50.  22.7 23.8 22.3 19.1 29.4
 23.2 24.6 29.9 37.2 39.8 37.9 32.5 26.4 29.6 32.  29.8 37.  30.5 36.4
 31.1 29.1 33.3 30.3 34.6 32.9 42.3 48.5 24.4 22.4 28.1 23.7 26.7 30.1
 44.8 37.6 46.7 31.5 31.7 41.7 48.3 29.  25.1 17.6 24.5 26.2 42.8 21.9
```

```
44.   36.   33.8 43.1 48.8 31.   36.5 30.7 43.5 20.7 21.1 25.2 35.2 32.4
33.1 35.1 45.4 46.   32.2 28.5 37.3 27.9 28.6 36.1 28.2 16.1 22.1 19.
32.7 31.2 17.2 16.8 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 11.5
15.1  9.7 12.5  8.5  5.   6.3  5.6 12.1  8.3 11.9 17.9 16.3  7.   7.5
 8.4 16.7 14.2 11.7 11.   9.5 14.1  9.6  8.7 12.8 10.8 14.9 12.6 13.
16.4 17.7 12.  21.8  8.1]
```

[3]:

```
data.hist()
plt.suptitle(f"Histograms of Data")
plt.tight_layout()
plt.show()
```

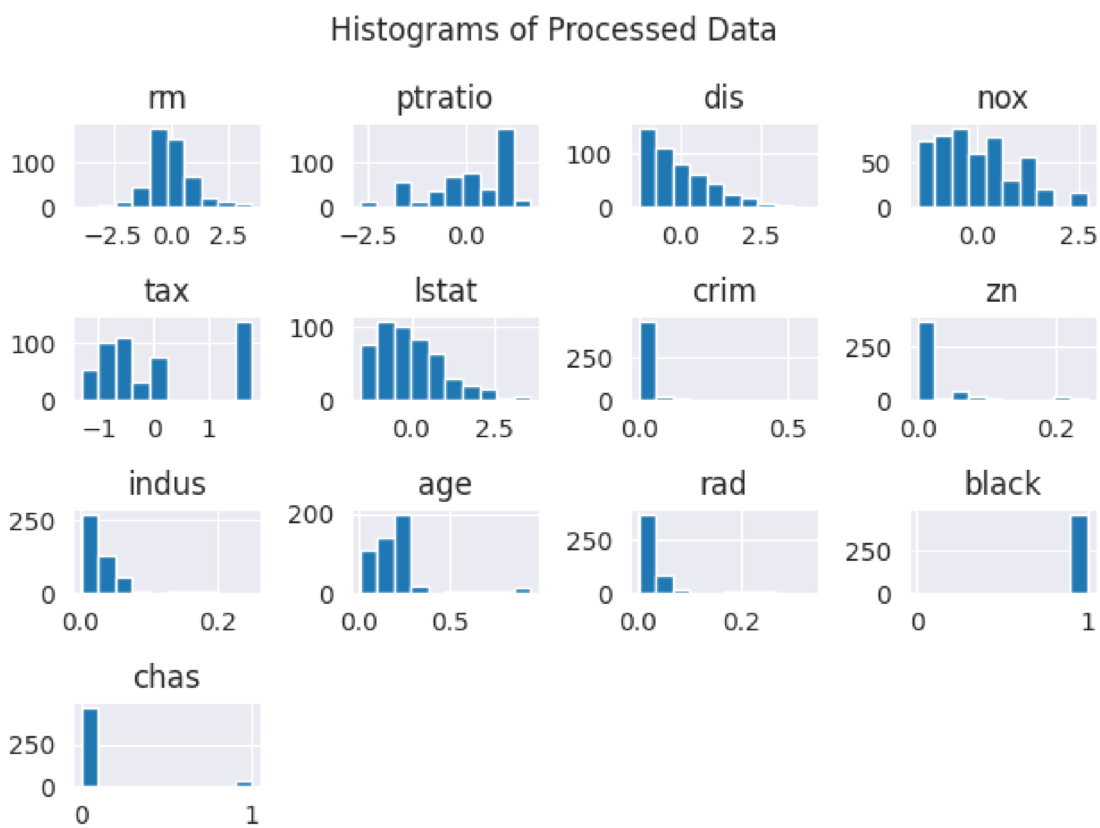


[4]:

```
features_to_standardize = ['rm', 'ptratio', 'dis', 'nox', 'tax', 'lstat']
features_to_normalize = ['crim', 'zn', 'indus', 'age', 'rad', 'black']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), features_to_standardize),
        ('norm', Normalizer(), features_to_normalize)
    ],
    remainder='passthrough'
```

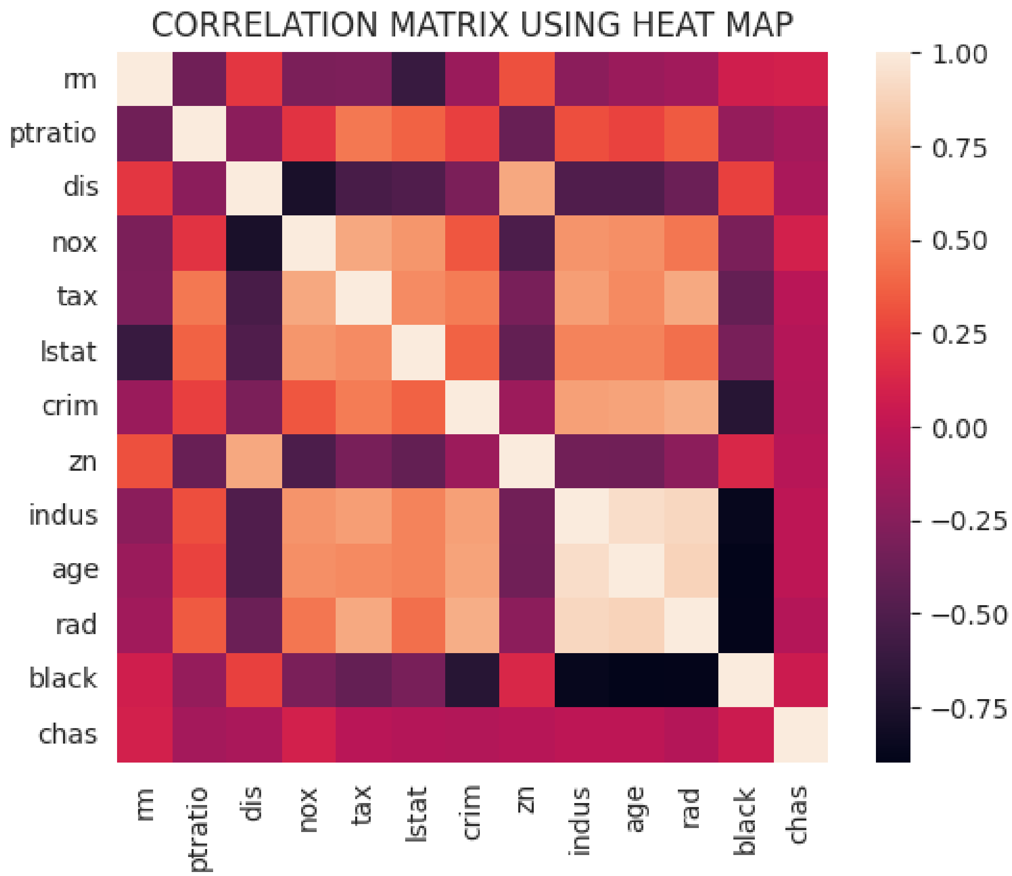
```
)

# Create the preprocessing pipeline
preprocessing_pipeline = Pipeline([
    ('preprocessor', preprocessor)
])
X = data.drop('medv', axis=1)
y = data['medv']
X_processed = preprocessing_pipeline.fit_transform(X)
```

[5]:
```
all_features = features_to_standardize + features_to_normalize + ['chas']

# Convert X_processed back to a DataFrame
X_processed = DataFrame(X_processed, columns=all_features)
X_processed.hist()
plt.suptitle("Histograms of Processed Data")
plt.tight_layout()
plt.show()
```

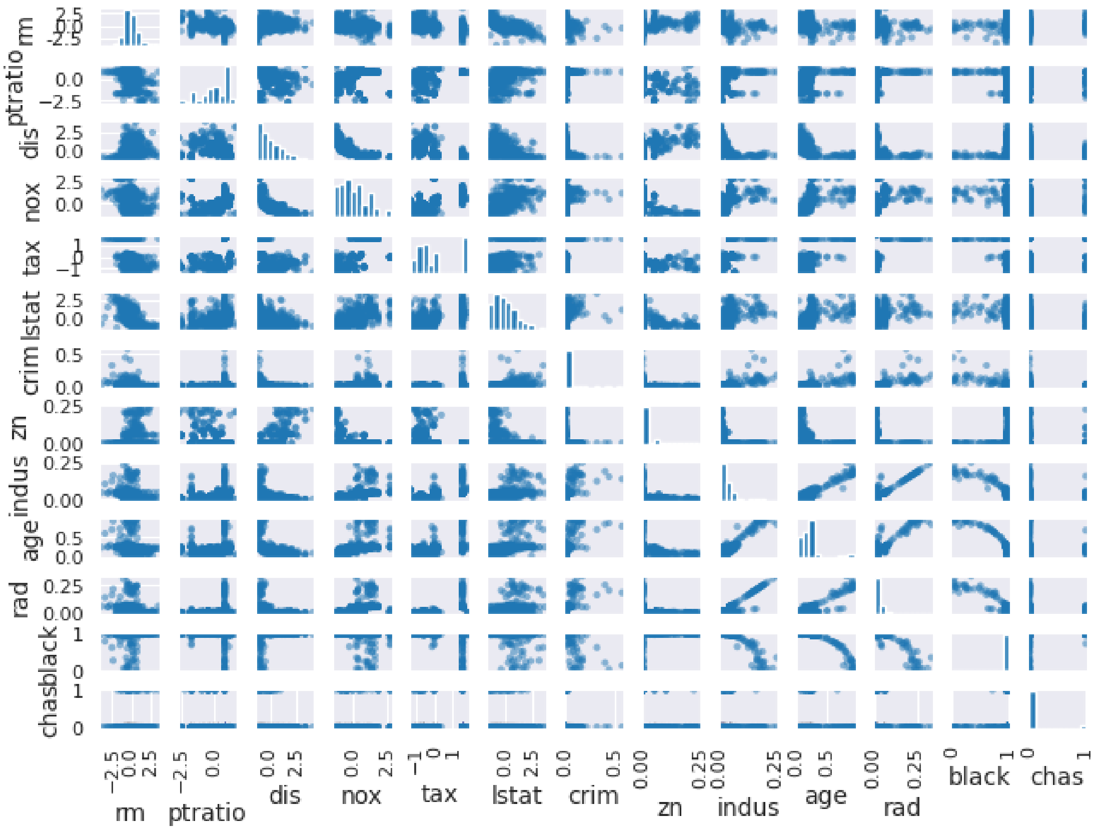


[6]:
```
plt.figure() # new plot
#plt.tight_layout()
corMat = X_processed.corr(method='pearson')
print(corMat)
## plot correlation matrix as a heat map
sns.heatmap(corMat, square=True)
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.title(f"CORRELATION MATRIX USING HEAT MAP")
plt.show()

## scatter plot of all data
plt.figure()
# # The output overlaps itself, resize it to display better (w padding)
scatter_matrix(X_processed)
plt.tight_layout(pad=0.1)
plt.show()
```

```
               rm   ptratio       dis       nox       tax     lstat      crim  \
rm       1.000000 -0.355501  0.205246 -0.302188 -0.292048 -0.613808 -0.167682
ptratio -0.355501  1.000000 -0.232471  0.188933  0.460853  0.374044  0.239932
dis      0.205246 -0.232471  1.000000 -0.769230 -0.534432 -0.496996 -0.301191
nox     -0.302188  0.188933 -0.769230  1.000000  0.668023  0.590879  0.329051
tax     -0.292048  0.460853 -0.534432  0.668023  1.000000  0.543993  0.476945
lstat   -0.613808  0.374044 -0.496996  0.590879  0.543993  1.000000  0.372586
crim    -0.167682  0.239932 -0.301191  0.329051  0.476945  0.372586  1.000000
zn       0.310664 -0.392270  0.669000 -0.518360 -0.314757 -0.412822 -0.160441
indus   -0.236454  0.297331 -0.498208  0.578670  0.623161  0.504321  0.636268
age     -0.167547  0.252975 -0.502044  0.556711  0.535171  0.508832  0.648392
rad     -0.138524  0.346488 -0.373426  0.455850  0.678546  0.428482  0.697182
black    0.077473 -0.190044  0.248901 -0.308355 -0.401559 -0.317661 -0.702350
chas     0.091251 -0.121515 -0.099176  0.091203 -0.035587 -0.053929 -0.060321

               zn     indus       age       rad     black      chas
rm       0.310664 -0.236454 -0.167547 -0.138524  0.077473  0.091251
ptratio -0.392270  0.297331  0.252975  0.346488 -0.190044 -0.121515
dis      0.669000 -0.498208 -0.502044 -0.373426  0.248901 -0.099176
nox     -0.518360  0.578670  0.556711  0.455850 -0.308355  0.091203
tax     -0.314757  0.623161  0.535171  0.678546 -0.401559 -0.035587
lstat   -0.412822  0.504321  0.508832  0.428482 -0.317661 -0.053929
crim    -0.160441  0.636268  0.648392  0.697182 -0.702350 -0.060321
zn       1.000000 -0.344436 -0.348234 -0.229033  0.132311 -0.043254
indus   -0.344436  1.000000  0.926254  0.902276 -0.864464 -0.010808
age     -0.348234  0.926254  1.000000  0.877650 -0.896251 -0.011293
rad     -0.229033  0.902276  0.877650  1.000000 -0.881809 -0.053457
black    0.132311 -0.864464 -0.896251 -0.881809  1.000000  0.052738
chas    -0.043254 -0.010808 -0.011293 -0.053457  0.052738  1.000000
```




```
<Figure size 640x480 with 0 Axes>
```



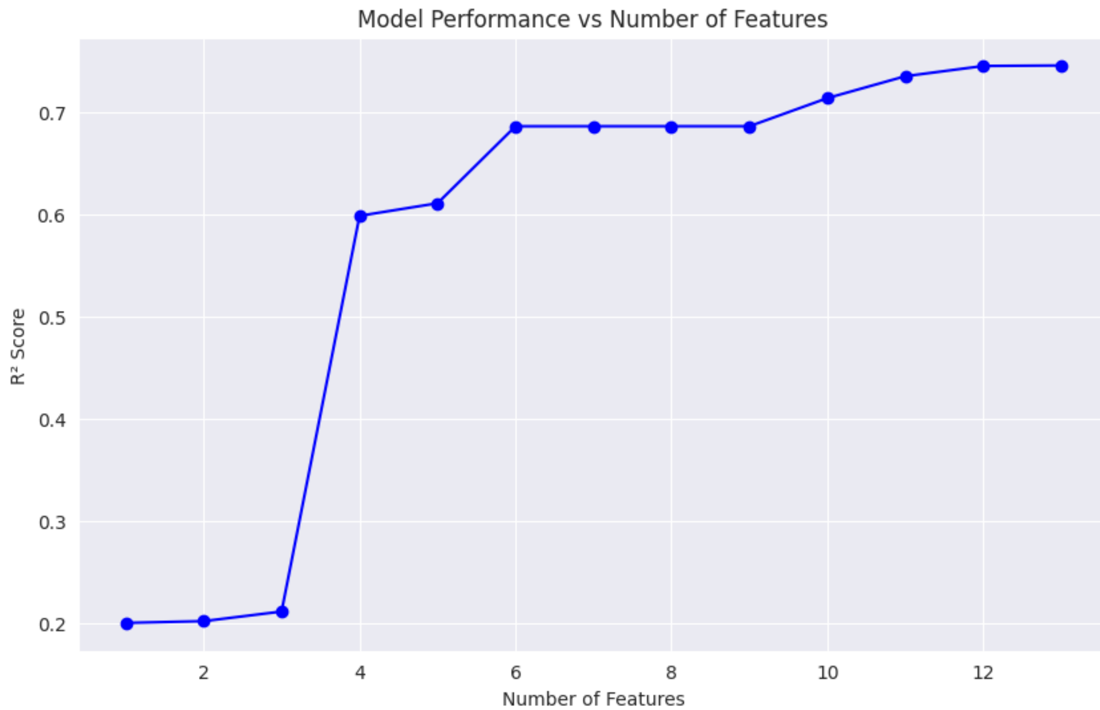

```
[7]: X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.
     ↪2, random_state=42)
```

```
[8]: def determine_optimal_number_of_features(X, y):
         feature_counts = range(1, X.shape[1] + 1)
         scores = []
         print(feature_counts)
         for num_features in feature_counts:
             model = LinearRegression()
             rfe = RFE(model, n_features_to_select = num_features)
             fit = rfe.fit(X, y)
             print("Num Features:", fit.n_features_)
             print("Selected Features:", fit.support_)
             print("Feature Ranking:", fit.ranking_)
             scores.append(rfe.score(X,y))
         # Plot results
         plt.figure(figsize=(10, 6))
         plt.plot(feature_counts, scores, 'b-', marker='o')
         plt.xlabel('Number of Features')
         plt.ylabel('R² Score')
```

```
    plt.title('Model Performance vs Number of Features')
    plt.grid(True)
    plt.show()
    best_num_features = feature_counts[argmax(scores)]
    print(f"Optimal number of features: {best_num_features}")
    print(f"Best score: {max(scores):.4f}")
    print(f"Features selected: {X.columns[fit.support_]}")
    return feature_counts, scores

feature_counts, scores = determine_optimal_number_of_features(X_train, y_train)
```

```
range(1, 14)
Num Features: 1
Selected Features: [False False False False False False False False  True False
False False
 False]
Feature Ranking: [ 4 11 10 12 13  6  3  7  1  8  2  9  5]
Num Features: 2
Selected Features: [False False False False False False False False  True False
True False
 False]
Feature Ranking: [ 3 10  9 11 12  5  2  6  1  7  1  8  4]
Num Features: 3
Selected Features: [False False False False False False  True False  True False
True False
 False]
Feature Ranking: [ 2  9  8 10 11  4  1  5  1  6  1  7  3]
Num Features: 4
Selected Features: [ True False False False False False  True False  True False
True False
 False]
Feature Ranking: [ 1  8  7  9 10  3  1  4  1  5  1  6  2]
Num Features: 5
Selected Features: [ True False False False False False  True False  True False
True False
  True]
Feature Ranking: [1 7 6 8 9 2 1 3 1 4 1 5 1]
Num Features: 6
Selected Features: [ True False False False False  True  True False  True False
True False
  True]
Feature Ranking: [1 6 5 7 8 1 1 2 1 3 1 4 1]
Num Features: 7
Selected Features: [ True False False False False  True  True  True  True False
True False
  True]
Feature Ranking: [1 5 4 6 7 1 1 1 1 2 1 3 1]
Num Features: 8
```

```
Selected Features: [ True False False False False  True  True  True  True  True
True False
  True]
Feature Ranking: [1 4 3 5 6 1 1 1 1 1 1 2 1]
Num Features: 9
Selected Features: [ True False False False False  True  True  True  True  True
True  True
  True]
Feature Ranking: [1 3 2 4 5 1 1 1 1 1 1 1 1]
Num Features: 10
Selected Features: [ True False  True False False  True  True  True  True  True
True  True
  True]
Feature Ranking: [1 2 1 3 4 1 1 1 1 1 1 1 1]
Num Features: 11
Selected Features: [ True  True  True False False  True  True  True  True  True
True  True
  True]
Feature Ranking: [1 1 1 2 3 1 1 1 1 1 1 1 1]
Num Features: 12
Selected Features: [ True  True  True  True False  True  True  True  True  True
True  True
  True]
Feature Ranking: [1 1 1 1 2 1 1 1 1 1 1 1 1]
Num Features: 13
Selected Features: [ True  True  True  True  True  True  True  True  True  True
True  True
  True]
Feature Ranking: [1 1 1 1 1 1 1 1 1 1 1 1 1]
```




```
Optimal number of features: 13
Best score: 0.7460
Features selected: Index(['rm', 'ptratio', 'dis', 'nox', 'tax', 'lstat', 'crim',
'zn', 'indus',
       'age', 'rad', 'black', 'chas'],
      dtype='object')
```

[9]:

```
def determine_features_within_threshold(_feature_counts, _scores):
    best_score = max(_scores)
    best_score_index = argmax(_scores)
    threshold = best_score * 0.98
    best_num_features = _feature_counts[best_score_index]
    i = best_score_index
    for i in range(best_score_index, 0, -1):
        if _scores[i] < threshold:
            print(f"Lowest number of features within 1% of best score:␣
↪{_feature_counts[i]}")
            break
print("Raw Data")
determine_features_within_threshold(feature_counts, scores)
```

```
Raw Data
Lowest number of features within 1% of best score: 10
```

[10]:
```
def stepwise_selection(X, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
        """ Perform a forward-backward feature selection
        based on p-value from statsmodels.api.OLS
        Arguments:
            X - pandas.DataFrame with candidate features
            y - list-like with the target
            initial_list - list of features to start with (column names of X)
            threshold_in - include a feature if its p-value < threshold_in
            threshold_out - exclude a feature if its p-value > threshold_out
            verbose - whether to print the sequence of inclusions and exclusions
        Returns: list of selected features
        Always set threshold_in < threshold_out to avoid infinite looping.
        See https://en.wikipedia.org/wiki/Stepwise_regression for the details
        """
        included = list(initial_list)
        while True:
            changed=False
            # forward step
            excluded = list(set(X.columns)-set(included))
            new_pval = Series(index=excluded)
            for new_column in excluded:
                model = sm.OLS(y, sm.
↪add_constant(DataFrame(X[included+[new_column]]))).fit()
                new_pval[new_column] = model.pvalues[new_column]
            best_pval = new_pval.min()
            if best_pval < threshold_in:
                best_feature = new_pval.idxmin()
                included.append(best_feature)
                changed=True
                if verbose:
                    print('Add  {:30} with p-value {:.6}'.format(best_feature,␣
↪best_pval))

            # backward step
            model = sm.OLS(y, sm.add_constant(DataFrame(X[included]))).fit()
            # use all coefs except intercept
            pvalues = model.pvalues.iloc[1:]
            worst_pval = pvalues.max() # null if pvalues is empty
            if worst_pval > threshold_out:
                changed=True
                worst_feature = pvalues.idxmax()
                included.remove(worst_feature)
                if verbose:
```

```
                    print('Drop {:30} with p-value {:.6}'.format(worst_feature,␣
 ↪worst_pval))
            if not changed:
                break
        return included
```

[11]:
```
result = stepwise_selection(X_train, y_train)
```

```
Add  lstat                          with p-value 3.19774e-70
Add  rm                             with p-value 3.26237e-25
Add  ptratio                        with p-value 3.70851e-11
Add  black                          with p-value 1.28801e-05
Add  dis                            with p-value 2.99151e-06
Add  nox                            with p-value 1.0268e-05
Add  chas                           with p-value 0.000880151
```

1.1 Build a multiple linear regression model using the RFE and the stepwise methods.

[12]:
```
# Build a multiple linear regression model using the RFE and the stepwise␣
 ↪methods.
# RFE
rfe_model = LinearRegression()
rfe = RFE(rfe_model, n_features_to_select=10)
fit = rfe.fit(X_train, y_train)
selected_columns = X_train.columns[fit.support_]
X_train_rfe = X_train[selected_columns]
rfe_model.fit(X_train_rfe, y_train)
print("Num Features:", fit.n_features_)
print("Selected Features:", fit.support_)
print("Feature Ranking:", fit.ranking_)

# Stepwise
step_model = LinearRegression()
# Build a model using the selected features in result
X_train_step = X_train[result]
step_model.fit(X_train_step, y_train)
print("Num Features:", len(result))
print("Selected Features:", result)
print("Feature Ranking:", [X_train.columns.get_loc(x) for x in result])

# Compare the two models across the training and test sets
def compare_models(model1, model2, X_test_rfe, X_test_step, y_test):
    # Calculate R² score for both models
    r2_score1 = model1.score(X_test_rfe, y_test)
    r2_score2 = model2.score(X_test_step, y_test)
```

```
    print(f"RFE Model R² Score: {r2_score1:.4f}")
    print(f"Stepwise Model R² Score: {r2_score2:.4f}")

# Prepare test data with the correct features for each model
X_test_rfe = X_test[selected_columns]  # Only use columns selected by RFE
X_test_step = X_test[result]           # Only use columns selected by stepwise

# Compare models with appropriate test data
compare_models(rfe_model, step_model, X_test_rfe, X_test_step, y_test)
```

```
Num Features: 10
Selected Features: [ True False  True False False  True  True  True  True  True
True  True
  True]
Feature Ranking: [1 2 1 3 4 1 1 1 1 1 1 1 1]
Num Features: 7
Selected Features: ['lstat', 'rm', 'ptratio', 'black', 'dis', 'nox', 'chas']
Feature Ranking: [5, 0, 1, 11, 2, 3, 12]
RFE Model R² Score: 0.5904
Stepwise Model R² Score: 0.6359
```

1. Use the Boston dataset and design a regression model using MLP regressor.

[13]:
```
# Set random seeds for reproducibility
tf.keras.backend.clear_session()
random.seed(42)
tf.random.set_seed(42)
data = read_csv(filename)
data = data.drop('index', axis=1)
X = data.drop('medv', axis=1)
y = data['medv']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train_full, X_test, y_train_full, y_test = train_test_split(X_scaled, y,␣
↪test_size=0.2, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,␣
↪y_train_full, test_size=0.2, random_state=42)
```

[14]:
```
# input_layer = tf.keras.layers.Input(shape=(X_train.shape[1],))
# # #hidden1 = tf.keras.layers.Dense(64, activation="relu")(input_layer)
# hidden2 = tf.keras.layers.Dense(32, activation="relu",␣
↪kernel_initializer='he_normal')(input_layer)#hidden1)
# # #hidden2 = tf.keras.layers.Dense(32, activation='selu',␣
↪kernel_initializer='lecun_normal')(input_layer)
# output = tf.keras.layers.Dense(1)(hidden2)  # No activation for regression
# model = tf.keras.models.Model(inputs=[input_layer], outputs=[output])
```

```
# # # import SGD optimizer to use momentum
# # #sgd = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# sgd = 'sgd'
# model.compile(
#     loss="mean_squared_error",
#     optimizer=sgd,
#     metrics=["mae"]
# )
# model.summary()
```

[15]:
```
# history = model.fit(
#     X_train, y_train,
#     #epochs=100,  # Adjusted based on the later results and the output of 100␣
 ↪epochs showing diminishing returns
#     epochs=50,
#     validation_data=(X_valid, y_valid),
#     verbose=1
# )
```

[16]:
```
configs = [
    {"layers": [32], "activation": "relu"},
    {"layers": [64], "activation": "relu"},
    {"layers": [32], "activation": "tanh"},
    {"layers": [64], "activation": "tanh"},
    {"layers": [32], "activation": "selu"},
    {"layers": [32], "activation": "sigmoid"},
]

results = []

for config in configs:
    print(f"Trying config: {config}")
    tf.keras.backend.clear_session()
    input_layer = tf.keras.layers.Input(shape=(X_train.shape[1],))
    x = input_layer
    for units in config["layers"]:
        x = tf.keras.layers.Dense(units, activation=config["activation"])(x)
    output = tf.keras.layers.Dense(1)(x)
    model = tf.keras.models.Model(inputs=input_layer, outputs=output)
    # sgd = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
    sgd = 'sgd'
    model.compile(loss="mean_squared_error", optimizer=sgd, metrics=["mae"])
    model.summary()
    history = model.fit(
        X_train, y_train,
        epochs=50,
```

```
        validation_data=(X_valid, y_valid),
        verbose=0
    )
    val_mae_hist = history.history["val_mae"]
    min_val_mae = np.min(val_mae_hist)
    best_epoch = np.argmin(val_mae_hist)
    best_loss = history.history["loss"][best_epoch]
    best_mae = history.history["mae"][best_epoch]
    best_val_loss = history.history["val_loss"][best_epoch]
    print(f"Best epoch: {best_epoch+1} | loss: {best_loss:.4f} - mae: {best_mae:
↪.4f} - val_loss: {best_val_loss:.4f} - val_mae: {min_val_mae:.4f}")
    results.append({
        "config": config,
        "epoch": best_epoch+1,
        "loss": best_loss,
        "mae": best_mae,
        "val_loss": best_val_loss,
        "val_mae": min_val_mae,
        "history": history.history
    })

df_results = pd.DataFrame(results)
display(df_results)
```

```
Trying config: {'layers': [32], 'activation': 'relu'}

I0000 00:00:1745783157.064373  324049 gpu_device.cc:2019] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 2266 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 3050 Ti Laptop GPU, pci bus id: 0000:01:00.0, compute
capability: 8.6
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 13)	0
dense (Dense)	(None, 32)	448
dense_1 (Dense)	(None, 1)	33

Total params: 481 (1.88 KB)

Trainable params: 481 (1.88 KB)

Non-trainable params: 0 (0.00 B)

```
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1745783157.868269  324289 service.cc:152] XLA service 0x7a8db0005ef0
initialized for platform CUDA (this does not guarantee that XLA will be used).
Devices:
I0000 00:00:1745783157.868283  324289 service.cc:160]   StreamExecutor device
(0): NVIDIA GeForce RTX 3050 Ti Laptop GPU, Compute Capability 8.6
2025-04-27 12:45:57.882169: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
I0000 00:00:1745783157.916625  324289 cuda_dnn.cc:529] Loaded cuDNN version
90300
I0000 00:00:1745783158.311882  324289 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.
```

```
Best epoch: 23 | loss: 11.0756 - mae: 2.3654 - val_loss: 16.2515 - val_mae:
2.9409
Trying config: {'layers': [64], 'activation': 'relu'}
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 13)	0
dense (Dense)	(None, 64)	896
dense_1 (Dense)	(None, 1)	65

Total params: 961 (3.75 KB)

Trainable params: 961 (3.75 KB)

Non-trainable params: 0 (0.00 B)

```
Best epoch: 47 | loss: 7.3931 - mae: 1.9724 - val_loss: 13.6585 - val_mae:
2.6965
Trying config: {'layers': [32], 'activation': 'tanh'}
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 13)	0
dense (Dense)	(None, 32)	448
dense_1 (Dense)	(None, 1)	33

Total params: 481 (1.88 KB)

Trainable params: 481 (1.88 KB)

Non-trainable params: 0 (0.00 B)

```
Best epoch: 50 | loss: 5.9727 - mae: 1.8345 - val_loss: 24.5085 - val_mae:
3.0831
Trying config: {'layers': [64], 'activation': 'tanh'}
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 13)	0
dense (Dense)	(None, 64)	896
dense_1 (Dense)	(None, 1)	65

Total params: 961 (3.75 KB)

Trainable params: 961 (3.75 KB)

Non-trainable params: 0 (0.00 B)

```
Best epoch: 50 | loss: 6.9823 - mae: 1.9809 - val_loss: 25.5173 - val_mae:
3.1006
Trying config: {'layers': [32], 'activation': 'selu'}
```

Model: "functional"

```
 Layer (type)                    Output Shape                 Param #

 input_layer (InputLayer)        (None, 13)                         0

 dense (Dense)                   (None, 32)                       448

 dense_1 (Dense)                 (None, 1)                         33


 Total params: 481 (1.88 KB)

 Trainable params: 481 (1.88 KB)

 Non-trainable params: 0 (0.00 B)
```

```
Best epoch: 49 | loss: 11.7249 - mae: 2.4786 - val_loss: 24.2591 - val_mae:
3.3577
Trying config: {'layers': [32], 'activation': 'sigmoid'}
```

Model: "functional"

```
 Layer (type)                    Output Shape                 Param #

 input_layer (InputLayer)        (None, 13)                         0

 dense (Dense)                   (None, 32)                       448

 dense_1 (Dense)                 (None, 1)                         33


 Total params: 481 (1.88 KB)

 Trainable params: 481 (1.88 KB)

 Non-trainable params: 0 (0.00 B)
```

```
Best epoch: 50 | loss: 15.1463 - mae: 2.7888 - val_loss: 21.9319 - val_mae:
2.9929
                                      config  epoch       loss       mae  \
0     {'layers': [32], 'activation': 'relu'}     23  11.075621  2.365373
1     {'layers': [64], 'activation': 'relu'}     47   7.393131  1.972393
```

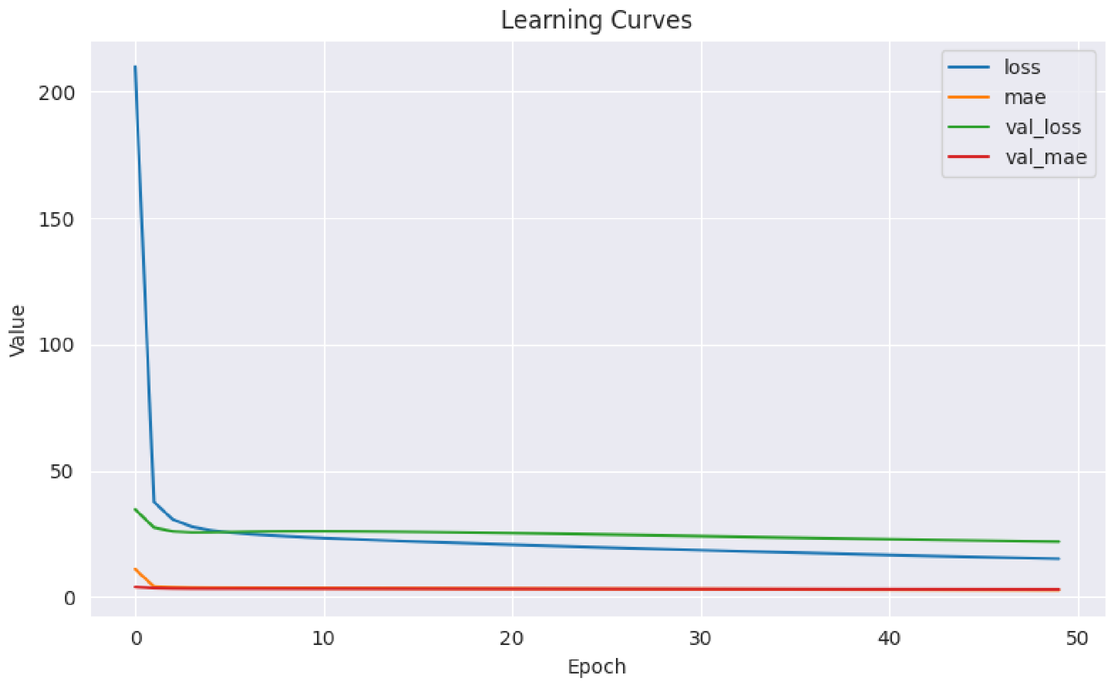
```
2     {'layers': [32], 'activation': 'tanh'}     50    5.972709  1.834482
3     {'layers': [64], 'activation': 'tanh'}     50    6.982259  1.980907
4     {'layers': [32], 'activation': 'selu'}     49   11.724866  2.478612
5  {'layers': [32], 'activation': 'sigmoid'}     50   15.146283  2.788818

    val_loss   val_mae                                            history
0  16.251486  2.940853  {'loss': [176.3515625, 29.739450454711914, 23…
1  13.658543  2.696483  {'loss': [175.93368530273438, 24.4227046966552…
2  24.508467  3.083100  {'loss': [328.7857360839844, 40.27022171020508…
3  25.517303  3.100601  {'loss': [293.4860534667969, 35.54962158203125…
4  24.259071  3.357651  {'loss': [235.65719604492188, 34.0321159362793…
5  21.931890  2.992933  {'loss': [210.00242614746094, 37.6197090148925…
```

[17]:
```
# Select the best configuration
best_idx = df_results["val_mae"].idxmin()
best_result = results[best_idx]
best_config = best_result["config"]
best_history = best_result["history"]

print(f"Best config: {best_config}")
```

```
Best config: {'layers': [64], 'activation': 'relu'}
```

[18]:
```
# Plot learning curves
# DataFrame(history.history).plot(figsize=(8, 5))
DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.title("Learning Curves")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.tight_layout()
plt.show()

# Evaluate the model on test data
test_loss, test_mae = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test).flatten()
mlp_mse = mean_squared_error(y_test, y_pred)
mlp_r2 = r2_score(y_test, y_pred)

print(f"MLP Test MSE: {mlp_mse:.4f}")
print(f"MLP Test MAE: {test_mae:.4f}")
print(f"MLP Test R²: {mlp_r2:.4f}")

# Compare with Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
```

```
lr_mse = mean_squared_error(y_test, y_pred_lr)
lr_r2 = r2_score(y_test, y_pred_lr)

print(f"Linear Regression Test MSE: {lr_mse:.4f}")
print(f"Linear Regression Test R²: {lr_r2:.4f}")
```




```
4/4              0s 42ms/step - loss:
12.4506 - mae: 2.3833
4/4              0s 32ms/step
MLP Test MSE: 18.2761
MLP Test MAE: 2.6796
MLP Test R²: 0.7508
Linear Regression Test MSE: 25.1021
Linear Regression Test R²: 0.6577
```

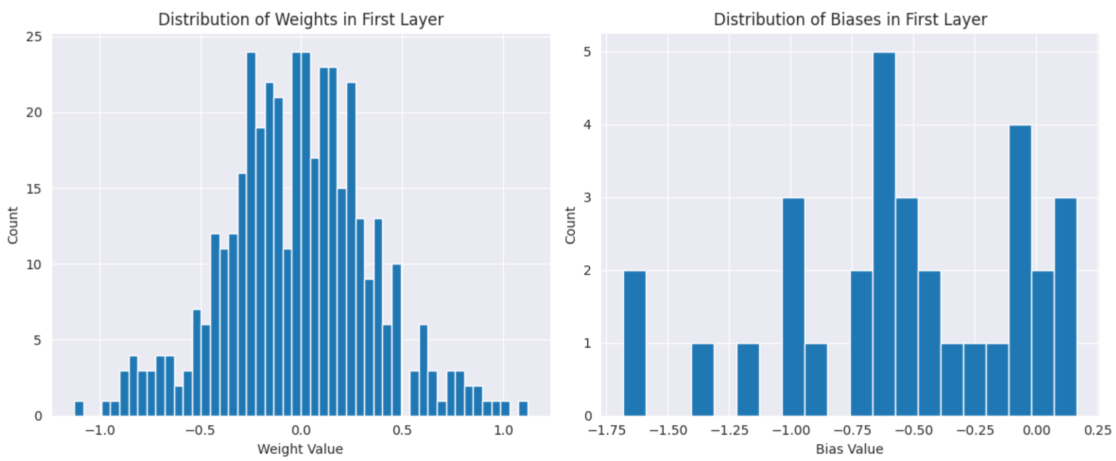
2. Compare the results with MLR model using cross validation. (rerun with MLR)

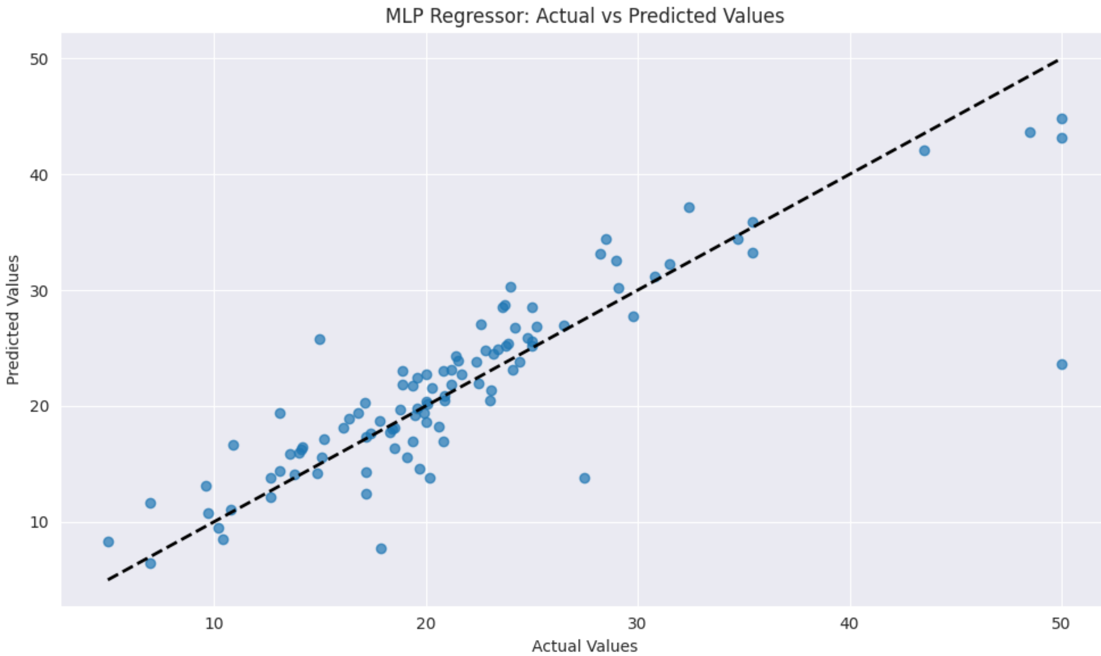
The MLP model outperforms the MLR model with cross validation in both lower test MSE and higher R^2 scores. This indicates that the MLP model with 64 units and ReLU activation has better generalization performance and prediction accuracy. The MLP's capacity to model complex, nonlinear relationships led to superior performance over the linear regression approach.

[19]:
```
weights, biases = model.layers[1].get_weights()
print("First layer weights shape:", weights.shape)
print("First layer biases shape:", biases.shape)
```

```python
# Plot the first few weights to visualize what the model learned
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.hist(weights.flatten(), bins=50)
plt.title("Distribution of Weights in First Layer")
plt.xlabel("Weight Value")
plt.ylabel("Count")

plt.subplot(1, 2, 2)
plt.hist(biases, bins=20)
plt.title("Distribution of Biases in First Layer")
plt.xlabel("Bias Value")
plt.ylabel("Count")

plt.tight_layout()
plt.show()

# Plot predictions vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("MLP Regressor: Actual vs Predicted Values")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
First layer weights shape: (13, 32)
First layer biases shape: (32,)
```





3. Comment on the MLP regressor architecture and its relationship with overfitting.

The MLP regressor used here has a relatively simple architecture, consisting of an input layer, a single hidden dense layer (either 32 or 64 units), and an output layer. This simplicity is a key factor in preventing overfitting, as it limits the model's capacity to learn noise and patterns that do not generalize well to new data. The architecture also includes dropout regularization (0.1) to further prevent overfitting by randomly setting a fraction of input units to 0 during training.