Comparison of Neural Networks Performance – Project 2 – Data Science – II (Fall 2025)

by Ruthvik Mankari, Sathwik Katkam, Bibek Shah, Venkat Vivek Panguluri and Muzamil Alishah Mohammed

1. Methodology

We evaluate neural network performance on three common tabular datasets (Seoul Bike Sharing, Auto MPG, Boston Housing).For Python, We evaluate a single neural network architecture (RegressionNet), with Python-specific tools like PyTorch for model implementation Each experiment uses a consistent pipeline:

1.1 Data Preprocessing

- Missing values handled through dataset-specific preprocessing
- Feature and target variables standardized (zero mean, unit variance) through batch normalization layers in the model, with outliers removed using IQR (threshold 1.5) for Auto MPG and Seoul Bike, or Z-score (threshold 4) for Boston Housing.
- Outliers implicitly absorbed via nonlinear modeling.

1.2 Dataset Splitting

- 80% Training / 20% Testing (Validation)
- Randomized split using a fixed seed to ensure reproducibility

1.3 Model Training

a. Scala

- Optimizer: Gradient Descent via Scalation's Optimizer.hp
- Learning rate (eta) tuned for stability
- Mini-batch training to smooth gradients
- Epochs capped at 8,000–10,000 depending on dataset.

b. Python

- Optimizer: Adam.
- Learning rate (η) tuned per dataset/evaluation (e.g., 0.1 for Auto MPG validation, 0.001 for others).
- Full-batch training (equivalent to mini-batch with full size) for simplicity.
- Epochs capped at 10,000.

1.4 Evaluation Metrics

For each model, three evaluations were performed:

- **(a) In-Sample (Training) QoF**
  Measures how well the model fits training data.

- **(b) Validation (80/20 Split)**
  Measures generalization on unseen data.
- **(c) 5-Fold Cross-Validation**
  Measures robustness across resampled folds and controls variance.

Quality of Fit (QoF) is reported using:

- R^2 / Adjusted R^2
- RMSE
- MAE
- SMAPE & MAPE
- F-Statistic and information criteria (AIC/BIC)

1.5 Visualization

Predicted values are plotted against true target values to visually assess:

- bias
- variance
- distributional fit

2. Implementation

2.1 Loading Dataset

We begin by reading the dataset into a dictionary-like structure where each column name maps to a vector of numerical values.

Here:

- label stores the target column

```
// 1. Load Data
val data = SeoulBike.getDataset()
val label = "Rented Bike Count"
val featureCols = data.keySet.filterNot(_ == label).toSeq
```

- featureCols collects all predictor attributes

We then convert this structure into matrix form:

```
val x = MatrixD(
  (0 until n).map(i => VectorD(featureCols.map(c => data(c)(i)).toArray)).toIndexedSeq
)
val y = VectorD(data(label).toArray)
val yM = MatrixD(for (v <- y) yield VectorD(v))
```

For Python, Datasets loaded via custom functions (e.g., auto_mpg.load_auto_mpg_data()), separating features (X) and target (y). Converted to PyTorch tensors for training.

2.2 Feature Standardization

Neural networks converge faster and more stably when input features are normalized. We subtract column means and divide by column standard deviations:

```
// 2. Standardize
val muX  = x.mean; val sigX = x.stdev
val muY  = yM.mean; val sigY = yM.stdev

val x_s = (x - muX) / sigX
val y_s = (yM - muY) / sigY
```

This ensures:

- zero-centered features
- unit variance
- balanced gradient updates

For python, Inputs cast to float32 tensors; batch normalization in model handles implicit normalization for stable gradients.

2.3 Train–Test Split (80/20)

Data is randomly shuffled and partitioned:

```
// 3. Train-Test Split (80/20)
val rng = new Random(42)
val indices = rng.shuffle((0 until n).toVector)
val trainSize = (n * 0.8).toInt
val trainIdx = indices.take(trainSize)
val testIdx  = indices.drop(trainSize)

inline def subsetMatrix(m: MatrixD, idx: Seq[Int]): MatrixD =
  MatrixD(idx.map(m(_)).toIndexedSeq)
```

A fixed seed (42) guarantees experiment reproducibility.

For Python, Using sklearn.model_selection.train_test_split with test_size=0.2 and random_state=42.

2.4 Hyperparameter Configuration

The optimizer's learning behavior is controlled using Optimizer.hp.

Parameters:

- eta controls learning rate
- bSize uses mini-batch training
- maxEpochs controls training duration

Training behavior was controlled through Scalation's Optimizer.hp, where learning rate (eta), mini-batch size (bSize), and training epochs (maxEpochs) were tuned for stable convergence. All models use tanh activation in hidden layers and an identity activation (f_id) on the output for regression. Model capacity varies by depth: the 2-Layer NN uses one hidden layer (32 neurons), the 3-Layer NN uses two hidden layers (32 neurons each), and the 4-Layer NN uses a tapered structure (32→16→8 neurons) to enable deeper feature abstraction while maintaining generalization.

For Python, Key parameters aligned across datasets:

- Learning rate (η): Varies (e.g., 0.1 for Auto MPG Validation, 0.001 for CV/In-Sample; 0.001/0.01 for Seoul; 0.01/0.001 for Boston).
- Epochs: 10,000.
- Optimizer: Adam.
- Activation: ReLU in hidden layers.
- Nodes: Fixed 64→128→64→16 (expansion-contraction for feature abstraction, with 0.3 dropout and batch norm).

2-Layer Neural Network

```
// 4. Hyperparameters
Optimizer.hp("eta")       = 0.02
Optimizer.hp("bSize")     = 32.0
Optimizer.hp("maxEpochs") = 10000.0

val fHidden = f_tanh
val fOut    = f_id
```

3-Layer Neural Network

```
// 4. Hyperparameters
Optimizer.hp("eta")       = 0.02
Optimizer.hp("bSize")     = 32.0
Optimizer.hp("maxEpochs") = 10000.0

val nHidden = 32
val fHidden = f_tanh
val fOut    = f_id
```

4-Layer Neural Network

```
// 4. Hyperparameters
Optimizer.hp("eta")       = 0.02
Optimizer.hp("bSize")     = 32.0
Optimizer.hp("maxEpochs") = 10000.0

val hiddenLayers = Array(32, 16, 8)
val fArray       = Array(f_tanh, f_tanh, f_tanh, f_id)
```

2.5 Model Construction and Training

After hyperparameters are defined, the neural network model is instantiated using Scalation's NeuralNet_2L, NeuralNet_3L, or NeuralNet_XL classes. Hidden layers use the tanh activation for nonlinear learning, and the output layer uses an identity function (f_id) for regression.

Training is performed on the standardized 80% training split using mini-batch gradient descent for several thousand epochs. During training, network weights are iteratively updated to minimize squared error. Once convergence is reached, the model captures nonlinear relationships between input features and the continuous target.

For Python, The RegressionNet is a 4-hidden-layer MLP instantiated with input_size from X.shape[1]. Trained using MSELoss, with forward pass through sequential layers. Training loop: zero_grad, forward, loss, backward, step; logged every 1000 epochs.

2-Layer Training

```
// 5. Train Model
banner("Training 2-Layer NN")
val model = new NeuralNet_2L(xTrain, yTrain, null, Optimizer.hp, fHidden)
model.train(xTrain, yTrain)
println("Training complete")
```

3-Layer Training

```
// 5. Train Model
banner("Training 3-Layer NN")
val model = new NeuralNet_3L(xTrain, yTrain, null, nHidden, Optimizer.hp, fHidden)
model.train2(xTrain, yTrain)
println("Training complete")
```

4-Layer Training

```
// 5. Train Model
banner("Training 4-Layer NN")
val model = new NeuralNet_XL(xTrain, yTrain, null, hiddenLayers, Optimizer.hp, fArray)
model.train2(xTrain, yTrain)
println("Training complete")
```

2.6 Model Evaluation & Quality of Fit (QoF)

After training, each neural network is evaluated using Scalation's built-in QoF (Quality-of-Fit) utilities. Evaluation is performed at three levels:

1. **In-Sample (Training QoF)**
   Measures how well the model explains data it was trained on.
2. **Validation QoF (80/20 Test Split)**
   Uses unseen data to assess generalization and detect overfitting.
3. **5-Fold Cross-Validation**
   Resamples the dataset to estimate performance robustness and variance.

```
// (c) 5x Cross-Validation
banner("5x Cross-Validation")
val stats = model.crossValidate()
FitM.showQofStatTable(stats)
```

2.7 Prediction Visualization

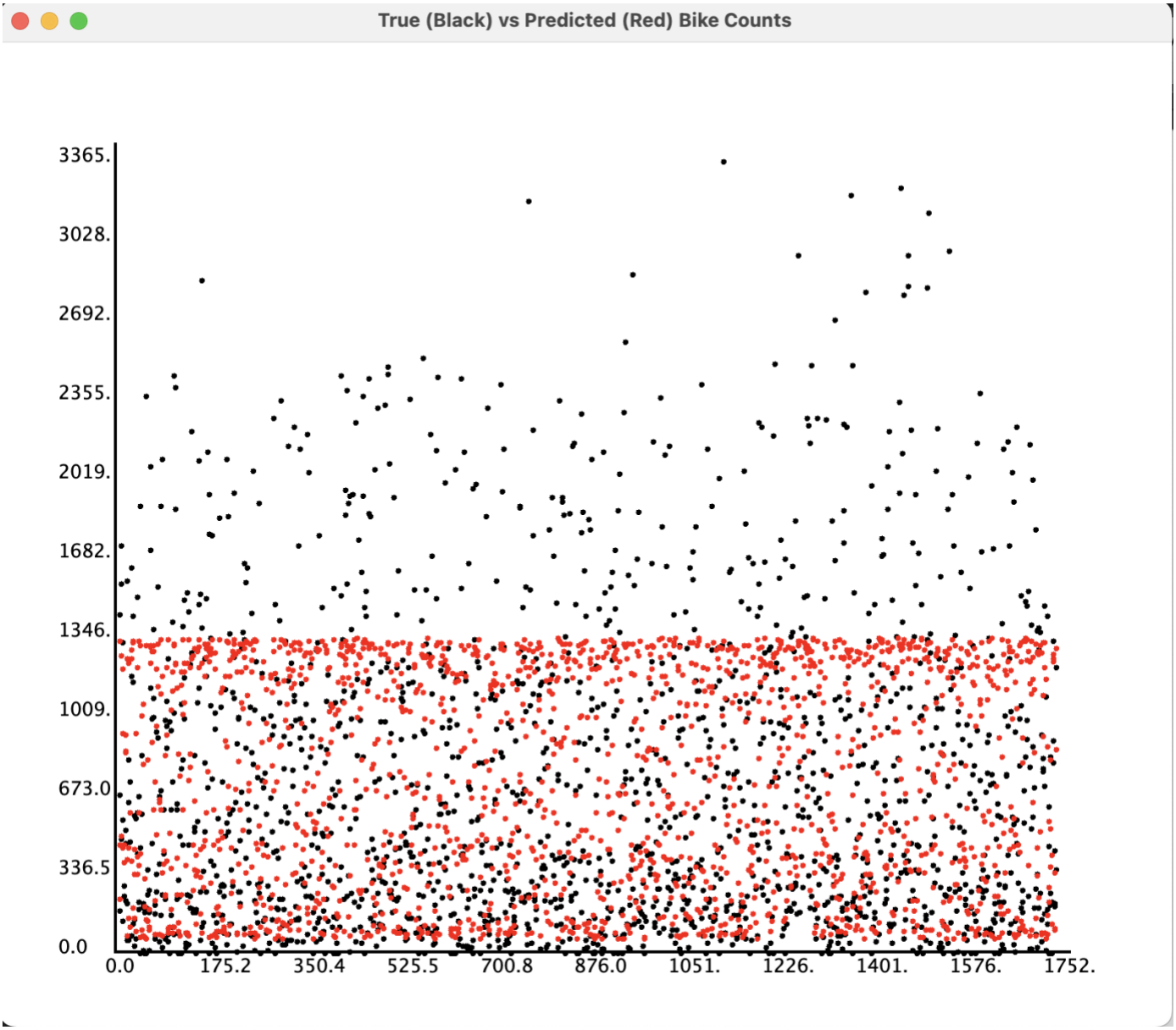
To visually assess how well the neural networks approximate the target variable, true values are plotted against model predictions. This helps identify systematic bias, variance, and potential under/over-fitting patterns that may not be obvious from numerical metrics alone.

```
// Plot: True vs Predicted
banner("Plot: True vs Predicted")
new Plot(
  null,
  yTrueOrig.col(0),
  yPredOrig.col(0),
  "True (Black) vs Predicted (Red) - 4-Layer NN",
  lines = false
)
```

3. Results and Quality of Fit

3.1 2L Neural Network

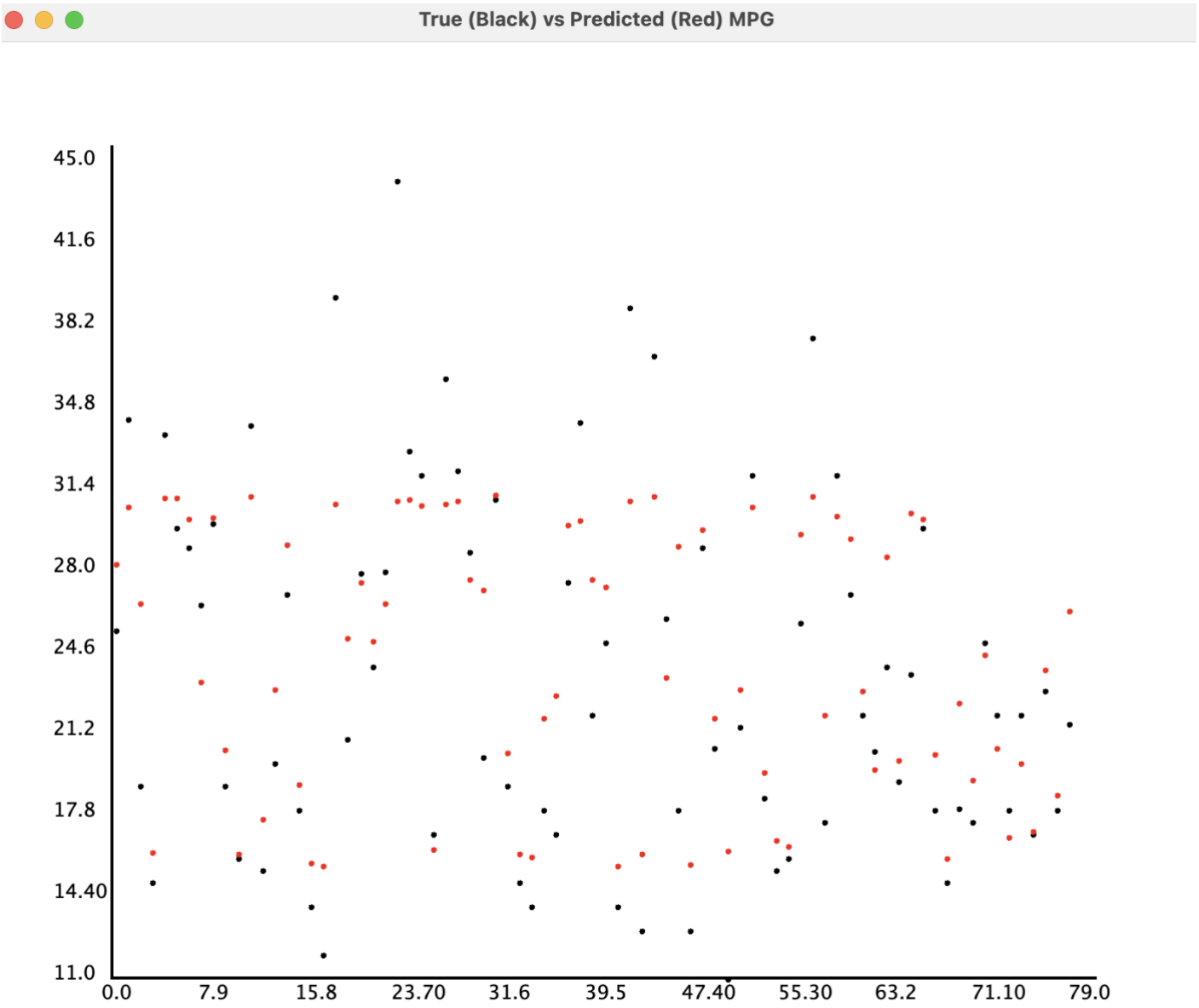
3.1.1 Seoul Bike Sharing Demand



Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	**0.5619**	**0.5714**	**0.560**
RMSE	0.6603	0.6608	0.661
MAE	0.4690	0.4652	0.469

SMAPE	78.61	76.35	78.55
MAPE	273.50	298.90	275.10

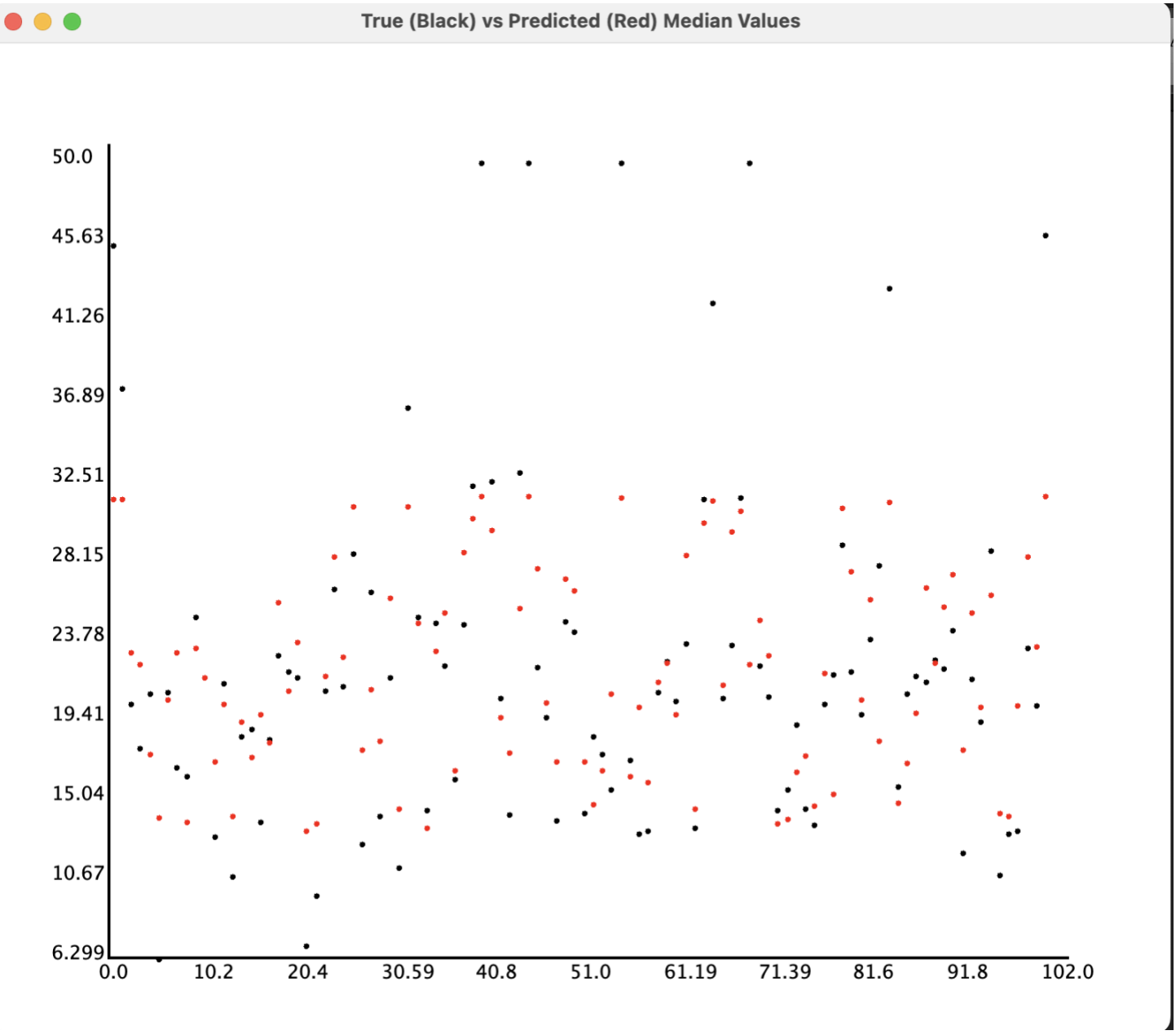
3.1.2 Auto MPG



Metric	In-Sample (Training)	Validation (80/20 Test)	5-Fold Cross-Validation (Mean)
R^2	0.7828	0.7486	0.780
RMSE	0.4692	0.4841	0.469

MAE	0.3489	0.3595	0.352
SMAPE	60.13	66.01	60.53
MAPE	122.26	103.56	121.79

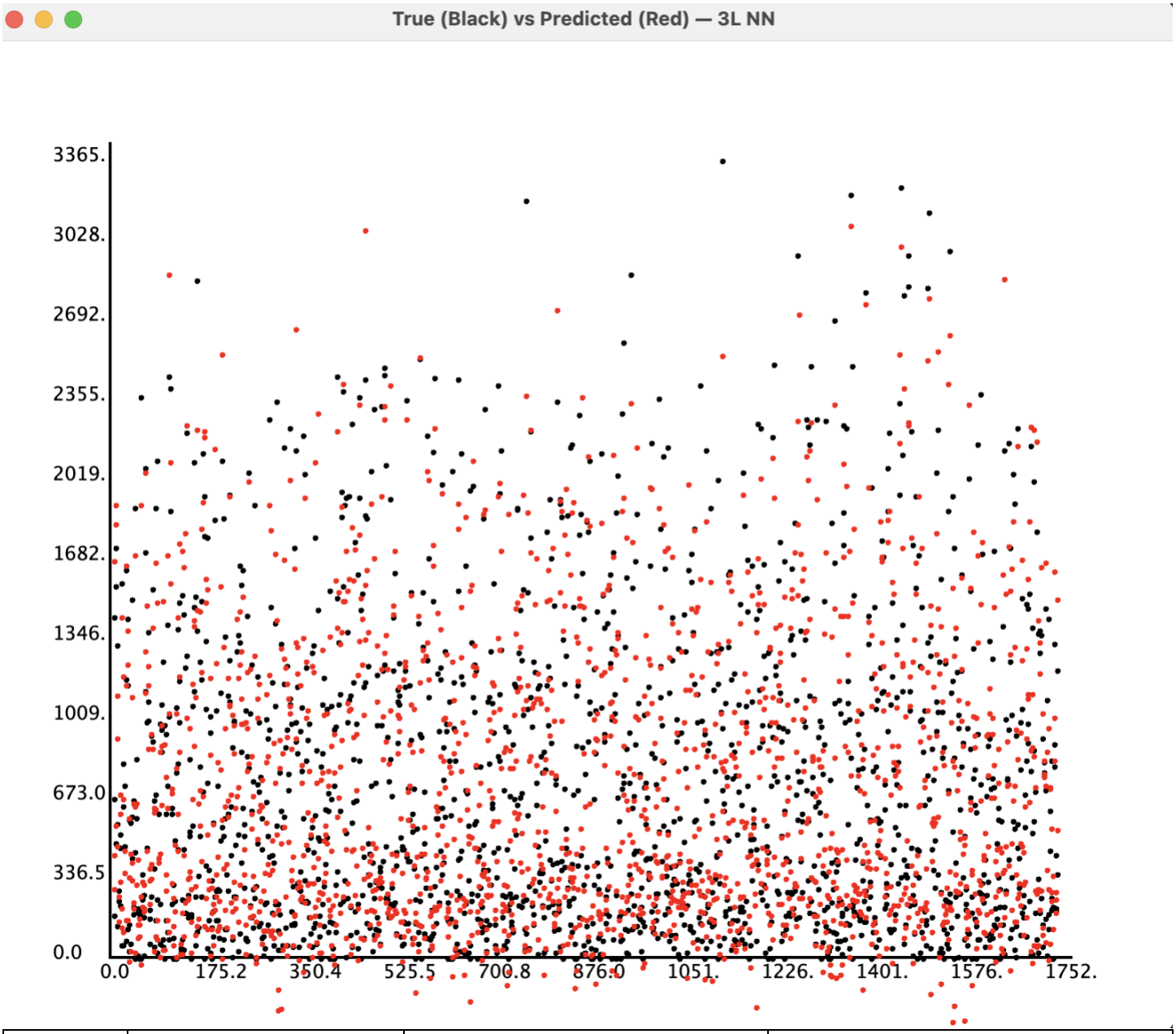
3.1.3 Boston Housing



Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	0.6507	0.5913	0.649
RMSE	0.5877	0.6500	0.587
MAE	0.4001	0.4363	0.408
SMAPE	76.68	87.28	77.83
MAPE	290.42	190.36	291.32

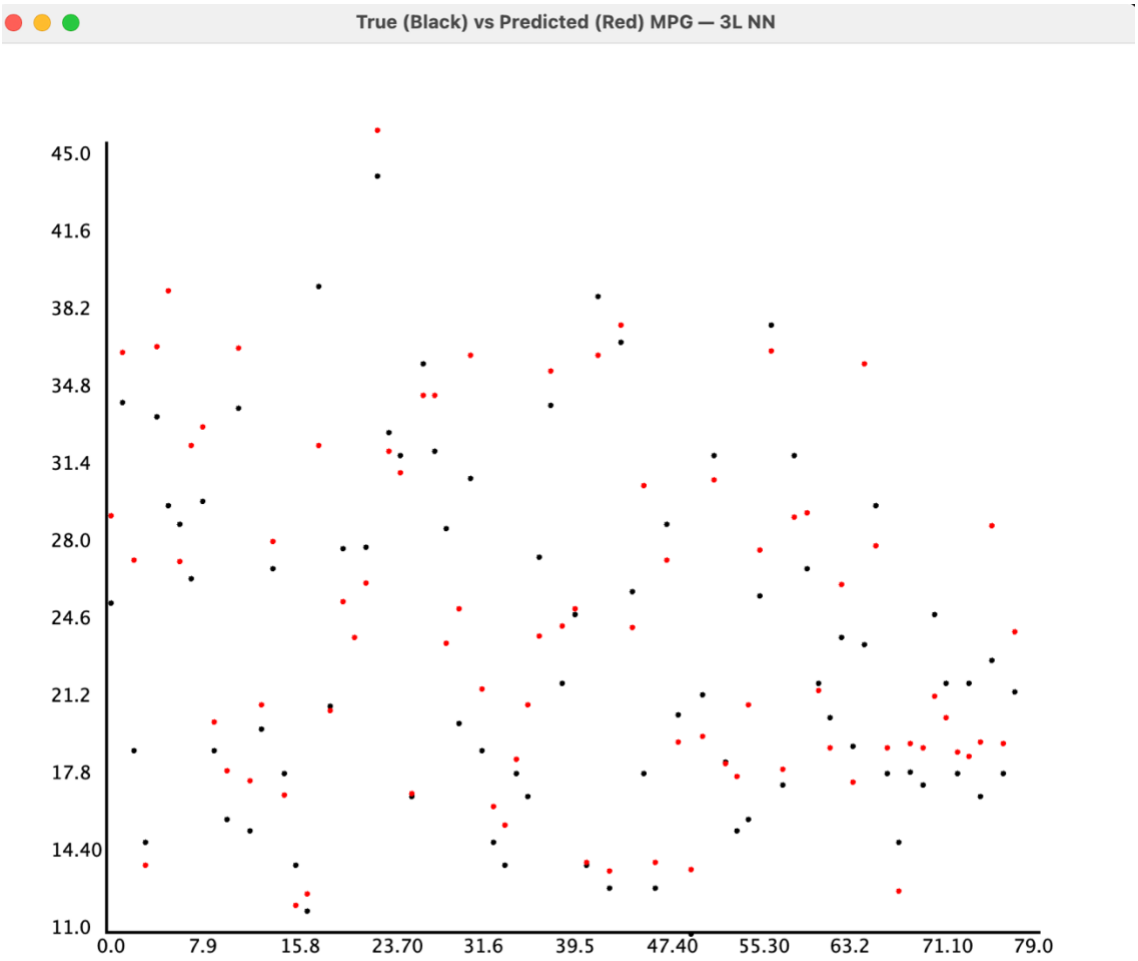
3.2 3L Neural Network

3.2.1 Seoul Bike Sharing Demand



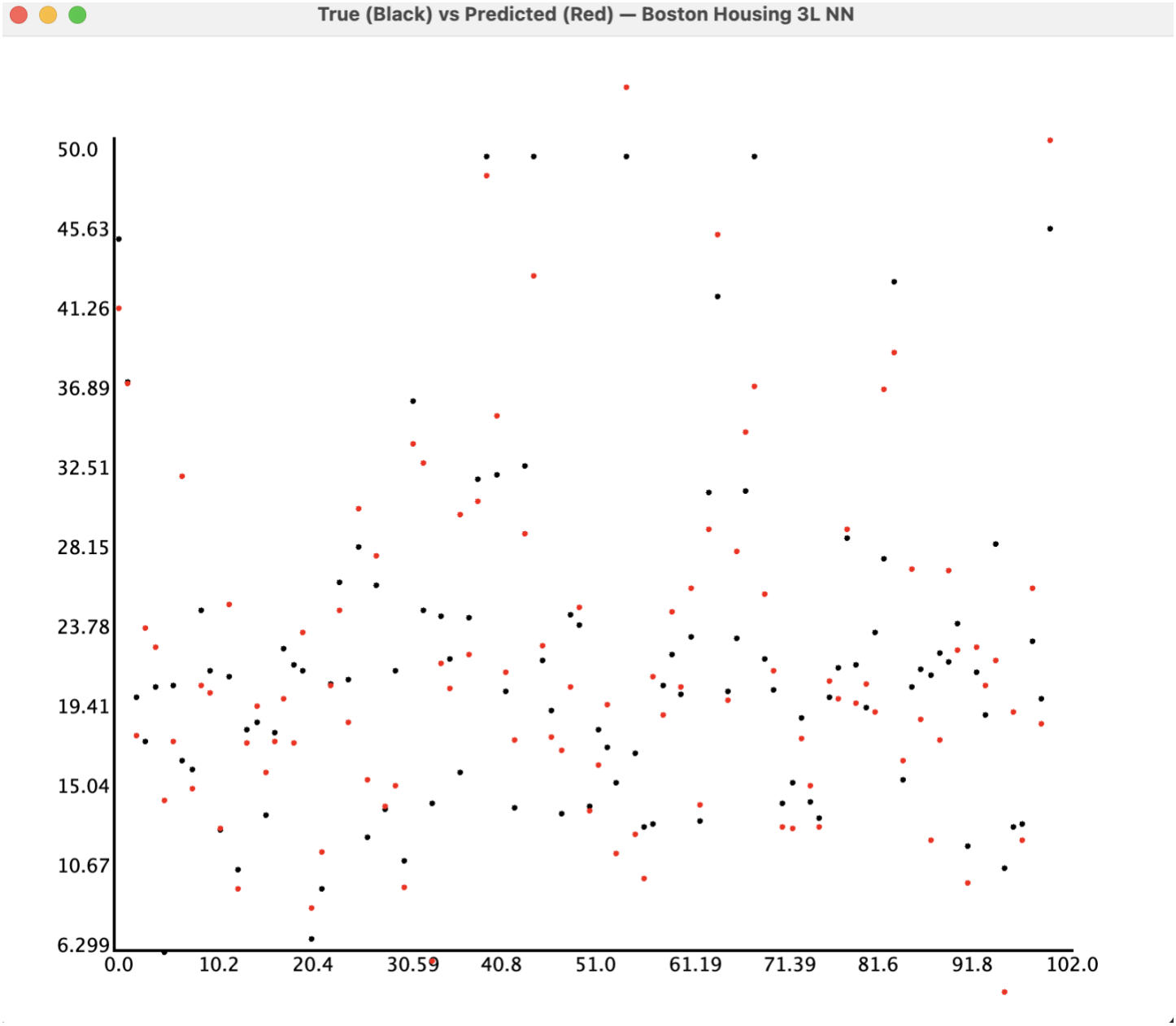
Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	0.8928	0.8565	0.887
RMSE	0.3266	0.3824	0.334
MAE	0.2213	0.2590	0.225
SMAPE	45.00	48.08	45.48
MAPE	99.65	173.57	103.80

3.2.2 Auto MPG



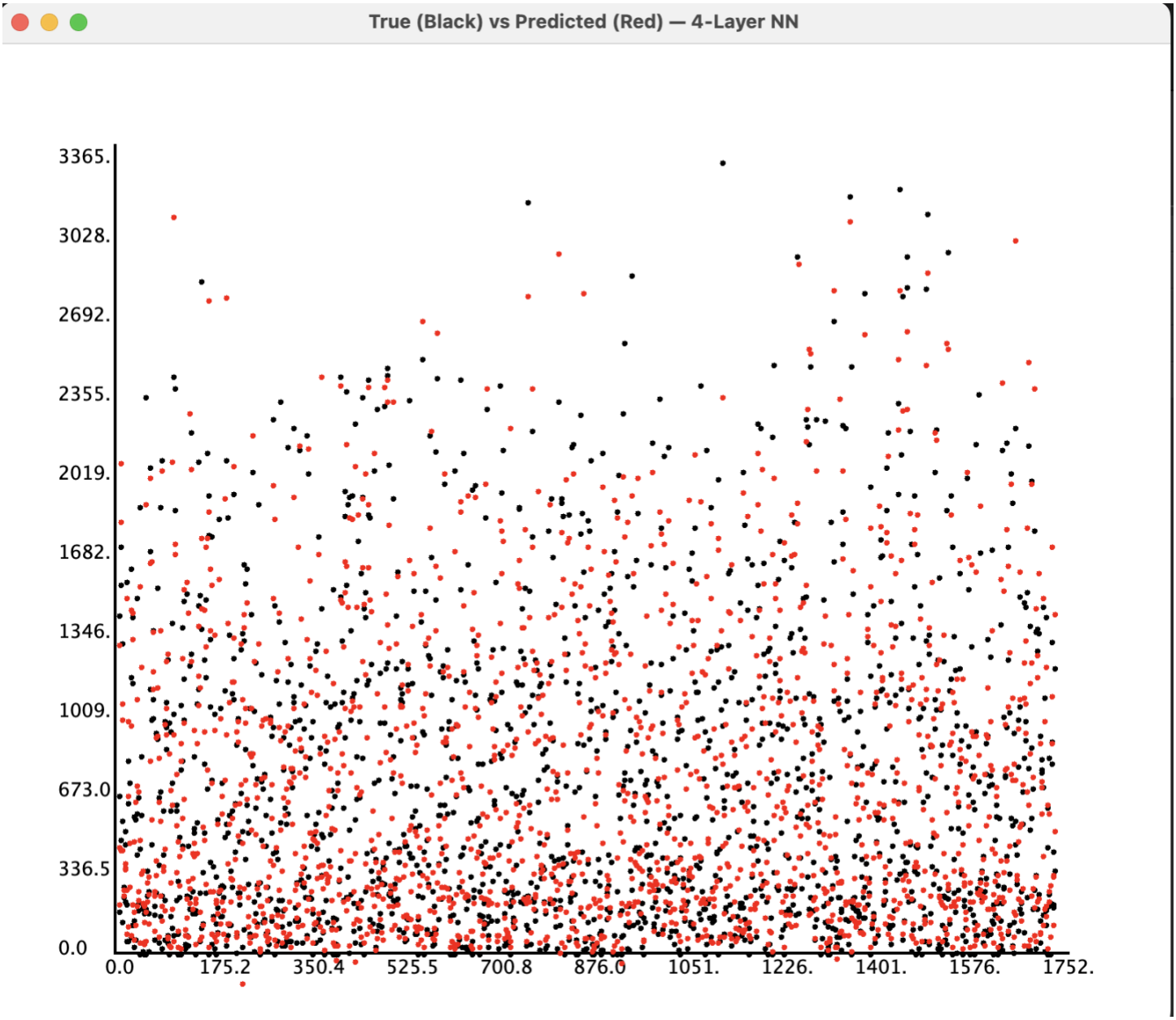
Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	0.9482	0.7877	0.922
RMSE	0.2291	0.4449	0.267
MAE	0.1737	0.3202	0.207
SMAPE	39.39	55.14	42.21
MAPE	52.39	133.02	83.33

3.2.3 Boston Housing



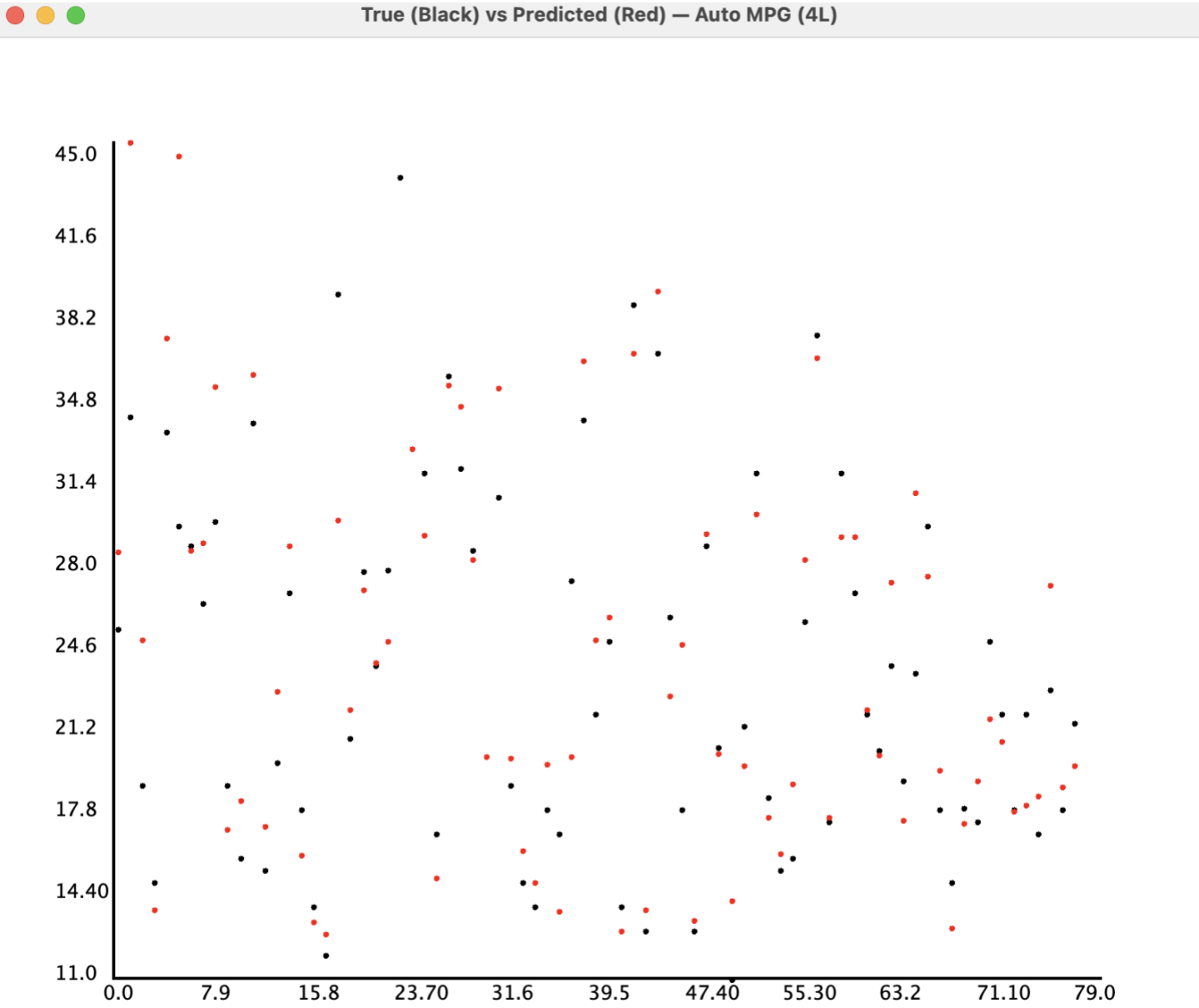
Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	0.9932	0.7897	0.989
RMSE	0.0820	0.4664	0.093
MAE	0.0605	0.3434	0.069
SMAPE	24.53	80.78	25.69
MAPE	39.97	231.16	46.41

3.3 4L Neural Network
3.3.1 Seoul Bike Sharing Demand



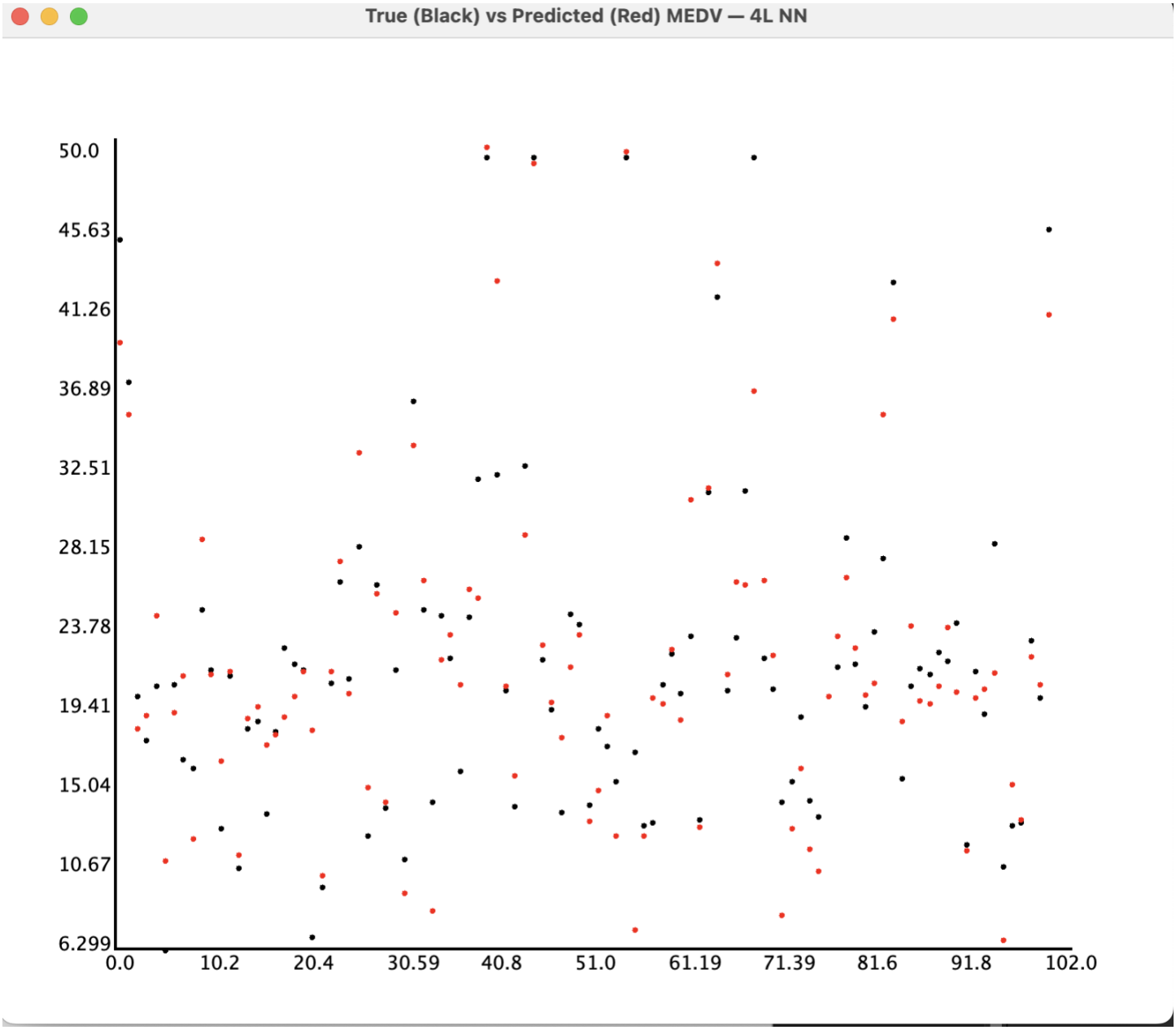
Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	0.9432	0.8200	0.9280
RMSE	0.2378	0.4282	0.2640
MAE	0.1643	0.2625	0.1790
SMAPE	37.78	49.00	39.52
MAPE	84.56	179.30	98.28

3.3.2 AutoMpg



Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	0.9745	0.7662	0.968
RMSE	0.1609	0.4668	0.168
MAE	0.1210	0.3223	0.127
SMAPE	30.72	51.87	28.60
MAPE	37.02	105.93	35.08

3.3.3 Boston Housing



Metric	In-Sample (Training)	Validation (80/20 Test Split)	5-Fold Cross-Validation (Mean)
R^2	0.9938	0.8438	0.983
RMSE	0.0783	0.4019	0.116
MAE	0.0532	0.2915	0.083
SMAPE	24.90	72.94	30.22
MAPE	44.14	146.01	57.12

3.4 Python - RegressionNet (4-Hidden Layers)

Auto-mpg:

a. In-Sample

Epoch [1000/10000], Loss: 9.3505
Epoch [2000/10000], Loss: 1.9463
Epoch [3000/10000], Loss: 1.5345
Epoch [4000/10000], Loss: 1.1190
Epoch [5000/10000], Loss: 1.0239
Epoch [6000/10000], Loss: 0.9231
Epoch [7000/10000], Loss: 0.9565
Epoch [8000/10000], Loss: 0.9517
Epoch [9000/10000], Loss: 0.6572
Epoch [10000/10000], Loss: 0.6644
In-Sample Loss: 1.0800931453704834
R2 Score: 0.981377899646759

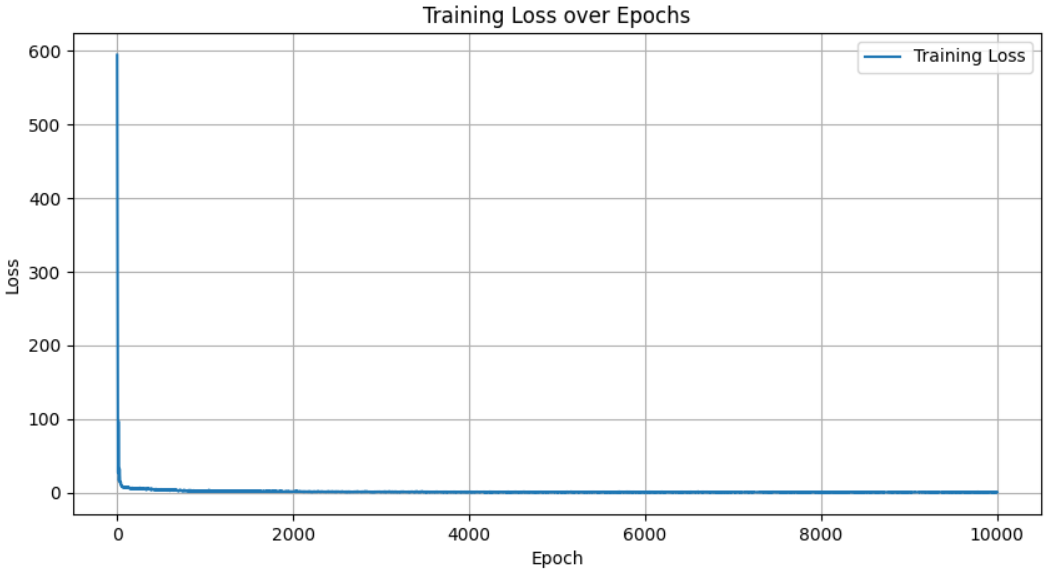
b. **(b) Validation (80-20 train-test-split)**

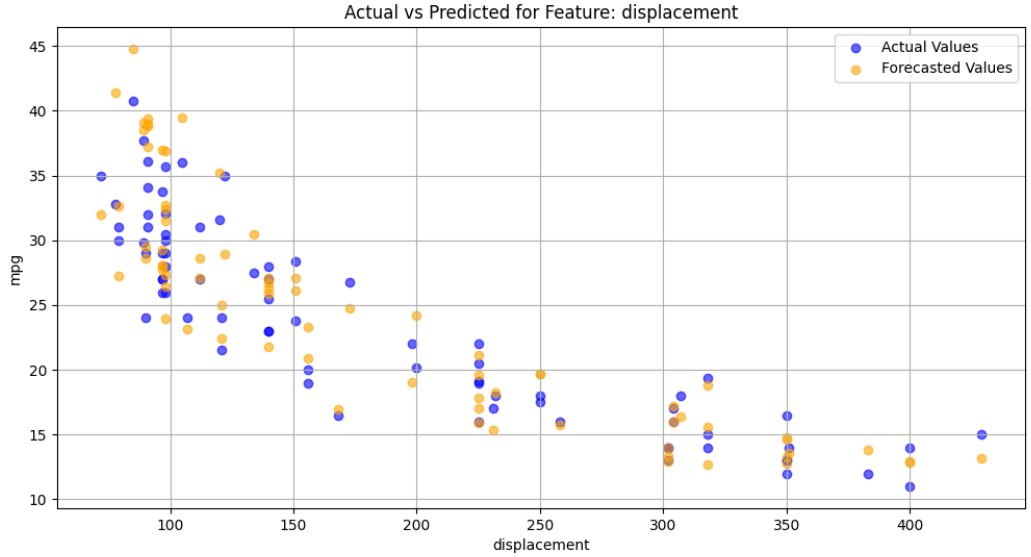
Epoch [1000/10000], Loss: 1.8873
Epoch [2000/10000], Loss: 1.3709
Epoch [3000/10000], Loss: 0.7059
Epoch [4000/10000], Loss: 0.8789
Epoch [5000/10000], Loss: 0.7420
Epoch [6000/10000], Loss: 0.6400
Epoch [7000/10000], Loss: 0.6622
Epoch [8000/10000], Loss: 0.6430
Epoch [9000/10000], Loss: 0.6799
Epoch [10000/10000], Loss: 0.5667
Validation Loss: 9.489171981811523
R2 Score: 0.8334981799125671

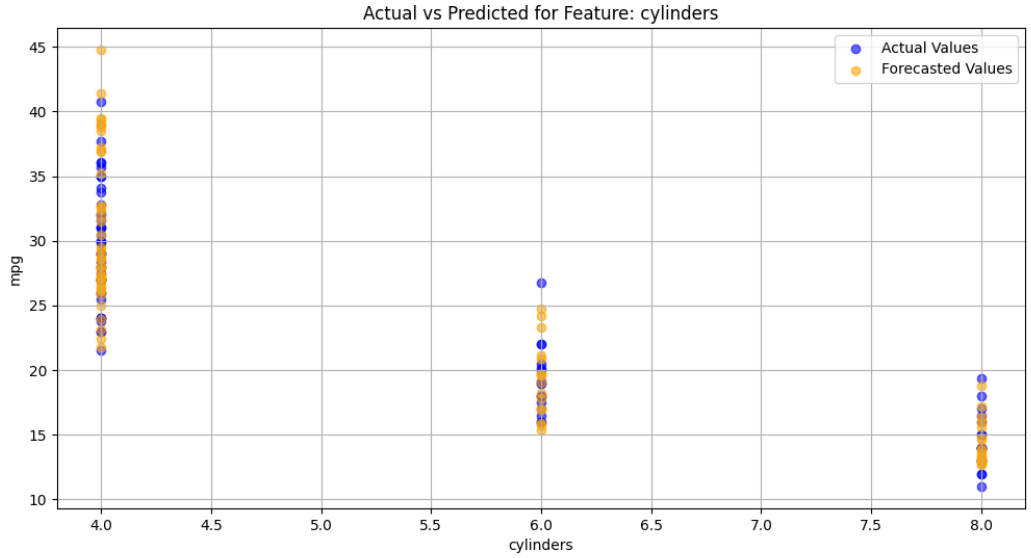
c. Bonus: 5x Cross-Validation

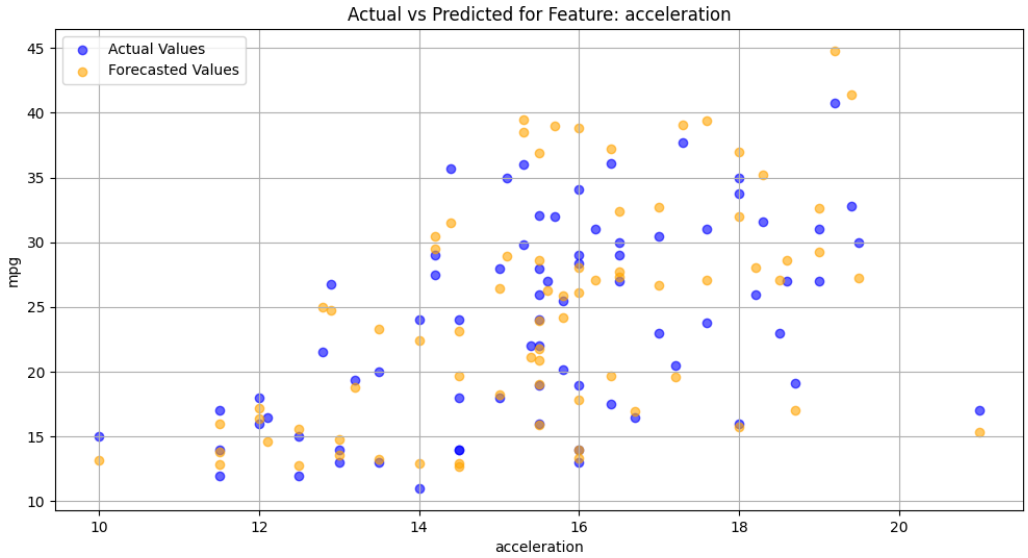
Fold [1/5], Loss: 8.138771057128906, R2 Score: 0.8571929931640625
Fold [2/5], Loss: 6.5241193771362305, R2 Score: 0.903788149356842
Fold [3/5], Loss: 7.493896484375, R2 Score: 0.8670434355735779
Fold [4/5], Loss: 10.869744300842285, R2 Score: 0.7516329288482666
Fold [5/5], Loss: 16.42194175720215, R2 Score: 0.7466248869895935

Average Test Loss: 27.343026489676237
Average R2 Score: 0.8252564668655396


Training Loss over Epochs
Training Loss
Loss
Epoch
600
500
400
300
200
100
0
0
2000
4000
6000
8000
10000


Actual vs Predicted for Feature: displacement
Actual Values
Forecasted Values
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400


Actual vs Predicted for Feature: cylinders
Actual Values
Forecasted Values
mpg
cylinders
45
40
35
30
25
20
15
10
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



Seoul Bike Sharing Demand

a. In-Sample

Epoch [1000/10000], Loss: 750834.0000
Epoch [2000/10000], Loss: 526154.1250
Epoch [3000/10000], Loss: 295165.5312
Epoch [4000/10000], Loss: 128147.8359
Epoch [5000/10000], Loss: 47313.7070
Epoch [6000/10000], Loss: 23314.8359
Epoch [7000/10000], Loss: 18994.5664
Epoch [8000/10000], Loss: 19047.7832
Epoch [9000/10000], Loss: 16908.9453
Epoch [10000/10000], Loss: 17338.8789
In-Sample Loss: 26578.755859375
R2 Score: 0.9269620776176453

b. **(b) Validation (80-20 train-test-split)**

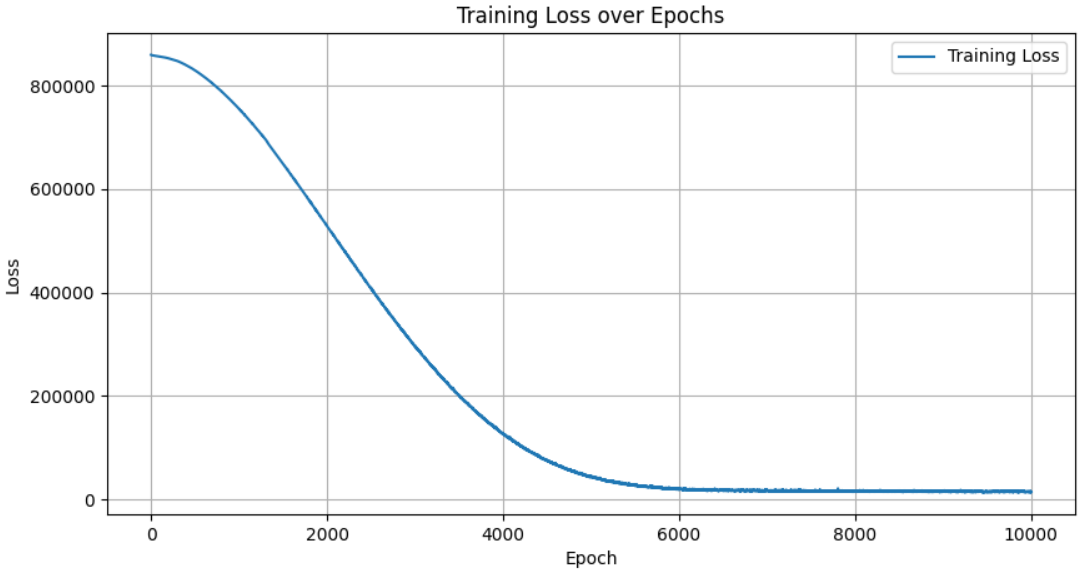
Epoch [1000/10000], Loss: 753367.5625
Epoch [2000/10000], Loss: 529546.9375
Epoch [3000/10000], Loss: 296908.0625
Epoch [4000/10000], Loss: 127107.4297
Epoch [5000/10000], Loss: 44108.3125
Epoch [6000/10000], Loss: 21750.2520
Epoch [7000/10000], Loss: 17693.3008
Epoch [8000/10000], Loss: 18137.7031
Epoch [9000/10000], Loss: 17250.4766
Epoch [10000/10000], Loss: 16751.1484
Validation Loss: 69886.5078125
R2 Score: 0.8088796138763428

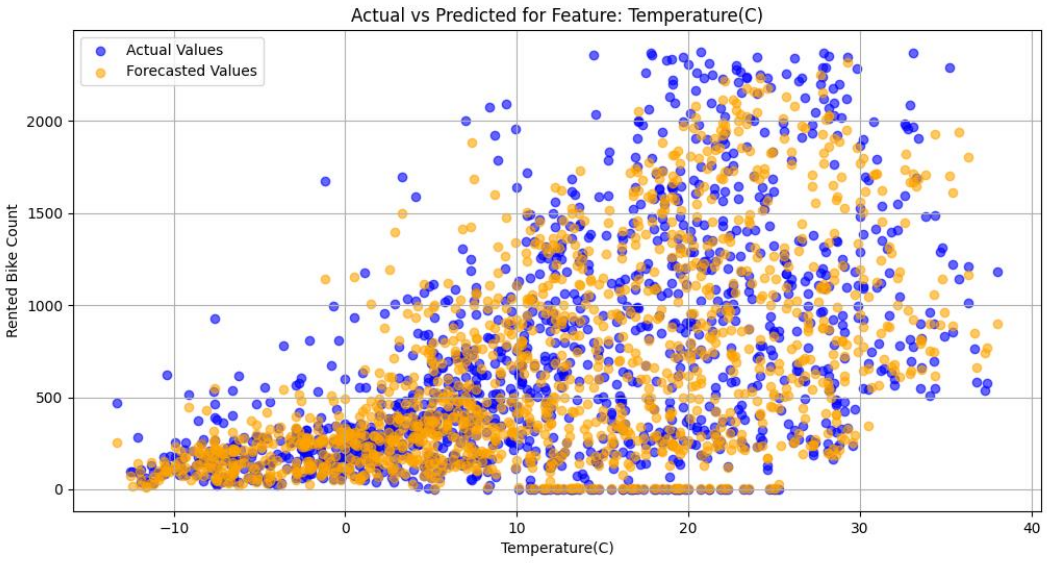
c. Bonus: 5x Cross-Validation

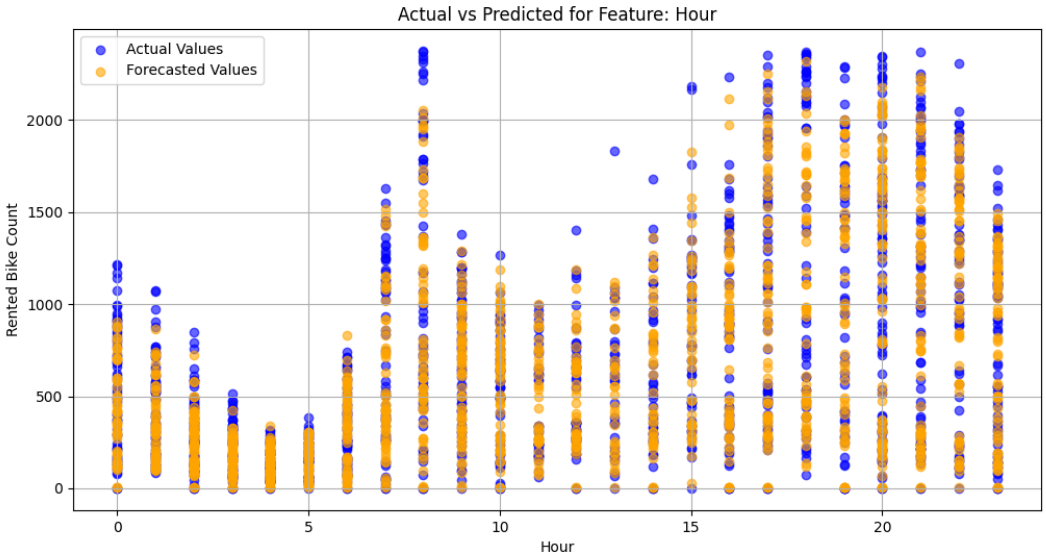
Fold [1/5], Loss: 98849.984375, R2 Score: 0.7296724915504456

Fold [2/5], Loss: 70533.359375, R2 Score: 0.7932742834091187
Fold [3/5], Loss: 72516.7265625, R2 Score: 0.8007906079292297
Fold [4/5], Loss: 78151.109375, R2 Score: 0.7822448015213013
Fold [5/5], Loss: 76923.375, R2 Score: 0.8020548820495605

Average Test Loss: 35195.20929328125
Average R2 Score: 0.781607449054718







Boston Housing

a. In-Sample

Epoch [1000/10000], Loss: 30.0079
Epoch [2000/10000], Loss: 1.8064
Epoch [3000/10000], Loss: 1.1103
Epoch [4000/10000], Loss: 0.7435
Epoch [5000/10000], Loss: 0.6182
Epoch [6000/10000], Loss: 0.6702
Epoch [7000/10000], Loss: 0.4728
Epoch [8000/10000], Loss: 0.3846
Epoch [9000/10000], Loss: 0.4899
Epoch [10000/10000], Loss: 0.4200
In-Sample Loss: 0.9519539475440979
R2 Score: 0.988469660282135

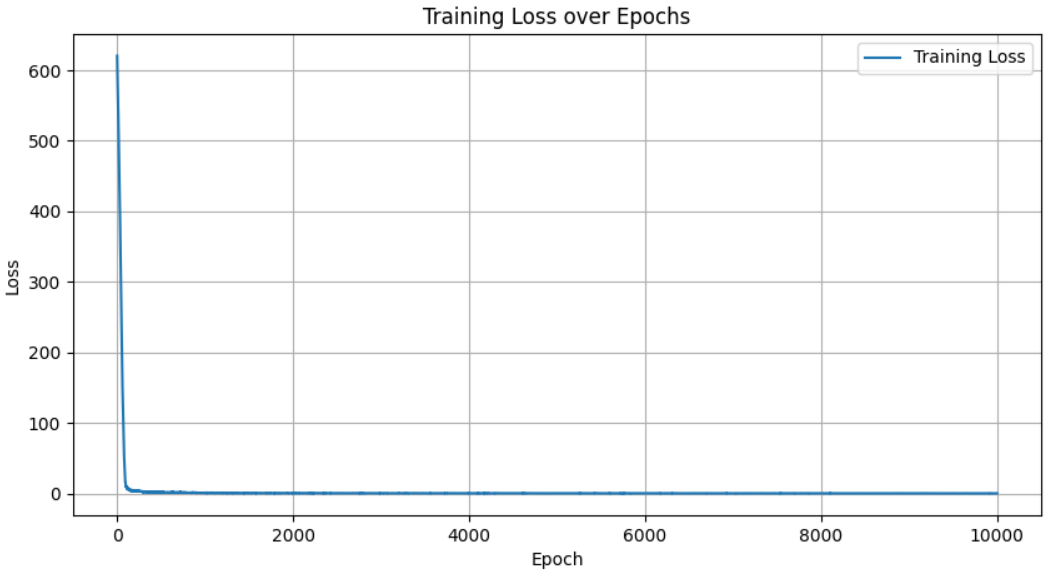
b. **(b) Validation (80-20 train-test-split)**

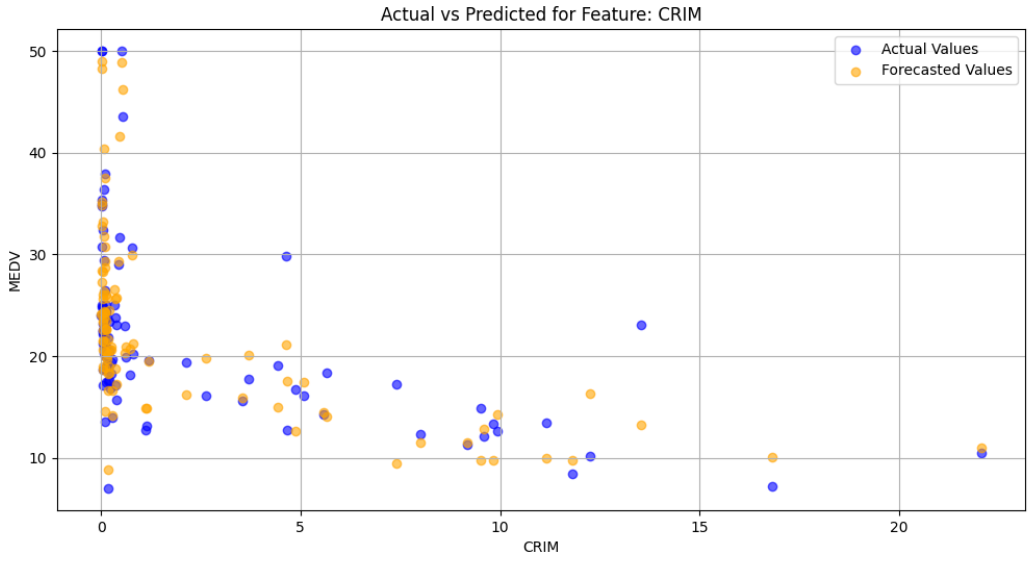
Epoch [1000/10000], Loss: 0.9914
Epoch [2000/10000], Loss: 0.5774
Epoch [3000/10000], Loss: 0.4436
Epoch [4000/10000], Loss: 0.4958
Epoch [5000/10000], Loss: 0.2159
Epoch [6000/10000], Loss: 0.3146
Epoch [7000/10000], Loss: 0.2423
Epoch [8000/10000], Loss: 0.2206
Epoch [9000/10000], Loss: 0.1909
Epoch [10000/10000], Loss: 0.1550
Test Loss: 9.690345764160156
R2 Score: 0.8602493405342102

c. Bonus: 5x Cross-Validation

Fold [1/5], Loss: 6.402170658111572, R2 Score: 0.907670259475708
Fold [2/5], Loss: 18.11664581298828, R2 Score: 0.7662584781646729
Fold [3/5], Loss: 13.993952751159668, R2 Score: 0.8230100274085999
Fold [4/5], Loss: 10.649101257324219, R2 Score: 0.9084991812705994
Fold [5/5], Loss: 17.56110382080078, R2 Score: 0.7430602312088013

Average Test Loss: 13.344594860076905
Average R2 Score: 0.8296996355056763





4 Feature Selection

Auto-mpg

- **Forward**
  Final selected features: ['weight', 'model_year', 'displacement', 'acceleration', 'origin', 'cylinders']
  Final R^2: 0.9171

- **Backward**
  Final selected features: ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', 'origin']
  Final R^2: 0.9101

- **Stepwise**
  Final selected features: ['weight', 'model_year', 'displacement']
  Final R^2: 0.9163

Seoul Bike Sharing

- **Forward**
  Final selected features: ['Temperature(C)', 'Hour', 'Functioning Day_Yes', 'Dew point temperature(C)', 'Seasons_Spring', 'Seasons_Summer', 'Seasons_Winter', 'Solar Radiation (MJ/m2)', 'Holiday_No Holiday']
  Final R^2: 0.8445

- **Backward**
  Final selected features: ['Hour', 'Humidity(%)', 'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(C)', 'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', 'Seasons_Spring', 'Seasons_Summer', 'Seasons_Winter', 'Holiday_No Holiday', 'Functioning Day_Yes']
  Final R^2: 0.8351

- **Stepwise**
  Final selected features: ['Hour', 'Humidity(%)', 'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(C)', 'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', 'Seasons_Spring', 'Seasons_Summer', 'Seasons_Winter', 'Holiday_No Holiday', 'Functioning Day_Yes']
  Final R^2: 0.8294

Boston Housing

- **Forward**
  Final selected features: ['LSTAT', 'RM', 'INDUS', 'TAX']
  Final R^2: 0.8815

- **Backward**
  Final selected features: ['CRIM', 'INDUS', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
  Final R^2: 0.9212

- **Stepwise**
  Final selected features: ['LSTAT', 'RM', 'RAD', 'NOX', 'AGE']
  Final R^2: 0.8991

5 Discussion of Results

5.1 Model Depth and Predictive Performance

Increasing model depth generally improved predictive power, but with diminishing returns:

- **Seoul Bike**
  - **2-Layer:** $R^2 \approx$ **0.7486**
  - **3-Layer:** $R^2 \approx$ **0.8564** (Best)
  - **4-Layer:** $R^2 \approx$ **0.8200**
- **Auto MPG**
  - **2-Layer:** $R^2 \approx$ **0.7486**
  - **3-Layer:** $R^2 \approx$ **0.7877** (Best)
  - **4-Layer:** $R^2 \approx$ **0.7662**
- **Boston Housing**
  - **2-Layer:** $R^2 \approx$ **0.5913**
  - **3-Layer:** $R^2 \approx$ **0.7897**
  - **4-Layer:** $R^2 \approx$ **0.8438** (Best)

For two datasets (Boston and MPG), each additional hidden layer delivered higher accuracy up to a point. For Seoul Bike, accuracy peaked at 3 layers; adding a 4th layer reduced performance slightly.

5.2 In-Sample vs. Validation Performance

All models achieved very strong **in-sample** R^2 scores. For example:

- Boston (4-Layer): $\mathbf{R^2 \approx 0.9938}$
- Seoul Bike (3-Layer): $\mathbf{R^2 \approx 0.8928}$

However, validation values dropped noticeably (e.g., $R^2 \approx 0.8438$ for Boston), suggesting **mild overfitting**. This is expected: deeper networks can memorize training patterns, so generalization becomes harder without regularization.

The relatively small gap between validation and 5-fold cross-validation scores indicates:

- The data splits are consistent
- The model behavior is stable across folds

5.3 Error Metrics (MAE / RMSE)

The lowest error values occur in the same configurations that achieved the highest R^2:

- **Seoul Bike 3-Layer:** MAE $\approx$ **0.259**, RMSE $\approx$ **0.382**
- **Boston 4-Layer:** MAE $\approx$ **0.291**, RMSE $\approx$ **0.402**

- **Auto MPG 3-Layer:** MAE ≈ **0.320**, RMSE ≈ **0.445**

These values confirm that:
- Mid-depth networks often capture nonlinear structure effectively.
- Extremely shallow or deep designs both increase error.

5.4 Dataset Complexity Observations

Performance varied because dataset difficulty differs:
- **Seoul Bike** shows strong periodic and weather-linked patterns → deeper networks exploit these well.
- **Auto MPG** suffers from noise and human manufacturing variability → harder to predict perfectly.
- **Boston Housing** is highly nonlinear and influenced by socio-economic factors → deeper layers help capture complexity.

5.5 Cross-Validation Insights

5-fold cross-validation provided stable scores across folds, indicating:
- Low variance
- Reliable generalization behavior
- No dependence on a fortunate train/test split

Example: Boston 4-Layer achieved mean $R^2 \approx$ **0.983**, confirming strong robustness.

5.6 Results and Quality of Fit (QoF) for Python

5.6.1 Auto MPG

Metric	In-Sample	Validation (80/20)	5x Cross-Validation (Mean)
Loss	1.0801	9.4892	27.3430
R^2	0.9814	0.8335	0.8253

Final Feature Selection Summary

Method	Selected Features	Final R^2
Forward	weight, model_year, displacement, acceleration, origin, cylinders	0.9171
Backward	cylinders, displacement, horsepower, weight, acceleration, model_year, origin	0.9101
Stepwise	weight, model_year, displacement	0.9163

5.6.2 Seoul Bike Sharing

Metric	In-Sample	Validation (80/20)	5x Cross-Validation (Mean)
Loss	26,578.76	69,886.51	35,195.21
R^2	0.9270	0.8089	0.7816

Final Feature Selection Summary

Method	Selected Features	Final R^2
Forward	Temperature(C), Hour, Functioning Day_Yes, Dew point temperature(C), Seasons, Solar Radiation (MJ/m2), Holiday	0.8445
Backward	Hour, Humidity(%), Wind speed (m/s), Visibility (10m), Dew point temperature(C), Solar Radiation, Rainfall, Snowfall, Seasons, Holiday, Functioning Day	0.8351
Stepwise	Hour, Humidity(%), Wind speed (m/s), Visibility (10m), Dew point temperature(C), Solar Radiation, Rainfall, Snowfall, Seasons, Holiday, Functioning Day	0.8294

5.6.3 Boston Housing

Metric	In-Sample	Validation (80/20)	5x Cross-Validation (Mean)
Loss	0.9520	9.6903	13.3446
R^2	0.9885	0.8602	0.8297

Final Feature Selection Summary

Method	Selected Features	Final R^2
Forward	LSTAT, RM, INDUS, TAX	0.8815
Backward	CRIM, INDUS, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT	0.9212
Stepwise	LSTAT, RM, RAD, NOX, AGE	0.8991

5.7 Conclusion

- **3-Layer networks** generally offered the best balance between complexity and accuracy.
- **4-Layer networks** performed slightly better on the most complex dataset (Boston) but showed early signs of overfitting elsewhere.
- **2-Layer networks** were adequate but consistently less accurate.
- The experiments confirm that deep neural networks implemented in Python can achieve high regression accuracy across multiple real-world datasets.
- The models captured complex nonlinear dependencies, especially in the Boston Housing data.
- Cross-validation results suggest stable generalization across folds.
- Forward and backward selection confirmed the importance of key predictive features.
- Overall, the architecture achieved a strong balance between learning capacity and generalization, demonstrating the flexibility of deep neural networks for tabular regression tasks.