1)Total number of parameters in your initial model (6629 parameters)
2) Number of layers used in your initial architecture. (3 layers)

```
Initial Model Summary:

Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	1,664
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17

```
Total params: 6,629 (25.90 KB)

Trainable params: 2,209 (8.63 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 4,420 (17.27 KB)

Number of layers in model: 3
```

```
3843/3843 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1647/1647 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
                                   Train      Test
Root Mean Squared Error         88344.21  88140.54
Mean Absolute Error             66692.28  66502.38
Mean Absolute Percentage Error     11.17     11.16
R2 score                            0.81      0.81
```

Model Improvement by adding layers

```python
def create_improved_regression_model(input_shape, params={}):
    model = tf.keras.Sequential([
        tf.keras.layers.InputLayer(shape=input_shape),
        tf.keras.layers.Dense(64, activation='relu'),  # more neurons
        tf.keras.layers.Dense(32, activation='relu'),  # added layer
        tf.keras.layers.Dense(16, activation='relu'),
        tf.keras.layers.Dense(1)
    ])

    model.compile(optimizer='adam', loss='mean_squared_error')

    return model
```

[9]

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 64)	3,328
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 16)	528
dense_6 (Dense)	(None, 1)	17

```
Total params: 17,861 (69.77 KB)

Trainable params: 5,953 (23.25 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 11,908 (46.52 KB)

Number of layers in model: 4
```

```python
    df_results.loc['Mean Absolute Percentage Error', 'Train'] = mean_absolute_per
    df_results.loc['R2 score', 'Train'] = r2_score(y_train, y_pred)

    y_pred = model_improved.predict(X_test)
    df_results.loc['Root Mean Squared Error', 'Test'] = np.sqrt(mean_squared_erro
    df_results.loc['Mean Absolute Error', 'Test'] = mean_absolute_error(y_test, y
    df_results.loc['Mean Absolute Percentage Error', 'Test'] = mean_absolute_perc
    df_results.loc['R2 score', 'Test'] = r2_score(y_test, y_pred)

    df_results = df_results.astype('Float64').round(2)
    df_results.to_csv('./data/model_evaluation.csv')

    print(df_results)
```

[14]

```
3843/3843 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1647/1647 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
                                   Train     Test
Root Mean Squared Error         63733.52  63566.5
Mean Absolute Error             46195.35 46109.85
Mean Absolute Percentage Error       7.5      7.5
R2 score                             0.9      0.9
```

```python
# Khin Hpone seeing th total parameters and layer in model
# Show model summary to see total params
print("Initial Model Summary:")
model_initial.summary()

# Print number of layers
print(f"Number of layers in model: {len(model_initial.layers)}")

# Khin Hpone seeing th total parameters and layer in model
# Show model summary to see total params
print("Improved Model Summary:")
model_improved.summary()

# Print number of layers
print(f"Number of layers in model: {len(model_improved.layers)}")
```

[12]

```
Initial Model Summary:

Model: "sequential"
```

To improve performance, I have modified the architecture by adding an extra hidden layer and increasing the number of neurons in each layer. The improved model used **4 layers**, totaling **17,861 parameters**.

```
chapter2 > Project_01 > SGHDB_improved.ipynb > M↓ Step 5: > # Function to evaluate model

# Evaluate both models
results_initial = evaluate_model(model_initial, X_train, y_train, X_test, y_test)
results_improved = evaluate_model(model_improved, X_train, y_train, X_test, y_test)

# Create dataframe to compare
df_comparison = pd.DataFrame({
    'Initial Model': results_initial,
    'Improved Model': results_improved
}).T

df_comparison['Training Time (s)'] = [initial_train_time, improved_train_time]
df_comparison = df_comparison.round(2)

df_comparison.to_csv('./data/model_comparison.csv')

#Khin Hpone
[15]
3843/3843 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
1647/1647 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
3843/3843 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
1647/1647 ━━━━━━━━━━━━━━━━━━━━ 2s 918us/step
```

	RMSE Train	RMSE Test	R2 Train	R2 Test	Training Time (s)
Initial Model	88344.21	88140.54	0.81	0.81	113.46
Improved Model	63733.52	63566.50	0.90	0.90	122.51

```
PS C:\Users\ASUS\Desktop\mmdt\Intro-to-Deep-Learning>
```

3) Model improvement analysis:

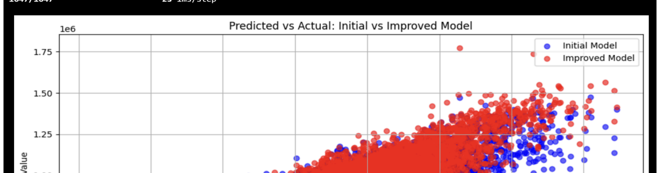
The improvement involved adding one more hidden layer and increasing the model's capacity to capture complex patterns in the data. The revised architecture used Dense layers with 64, 32, 16, and 1 neurons, respectively, with ReLU activation in hidden layers. **Training Time and Performance improved as can be seen** through the ◀️**screenshot** Performance Comparison ◀️

```
# Predictions
import data_config as cfg
import importlib
import data_config as cfg
importlib.reload(cfg)

y_pred_initial_test = model_initial.predict(X_test)
y_pred_improved_test = model_improved.predict(X_test)

cfg.plot_predictions_comparison(y_test, y_pred_initial_test, y_pred_improved_test, fname='model_prediction_comparison.png')

1647/1647 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
1647/1647 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
```

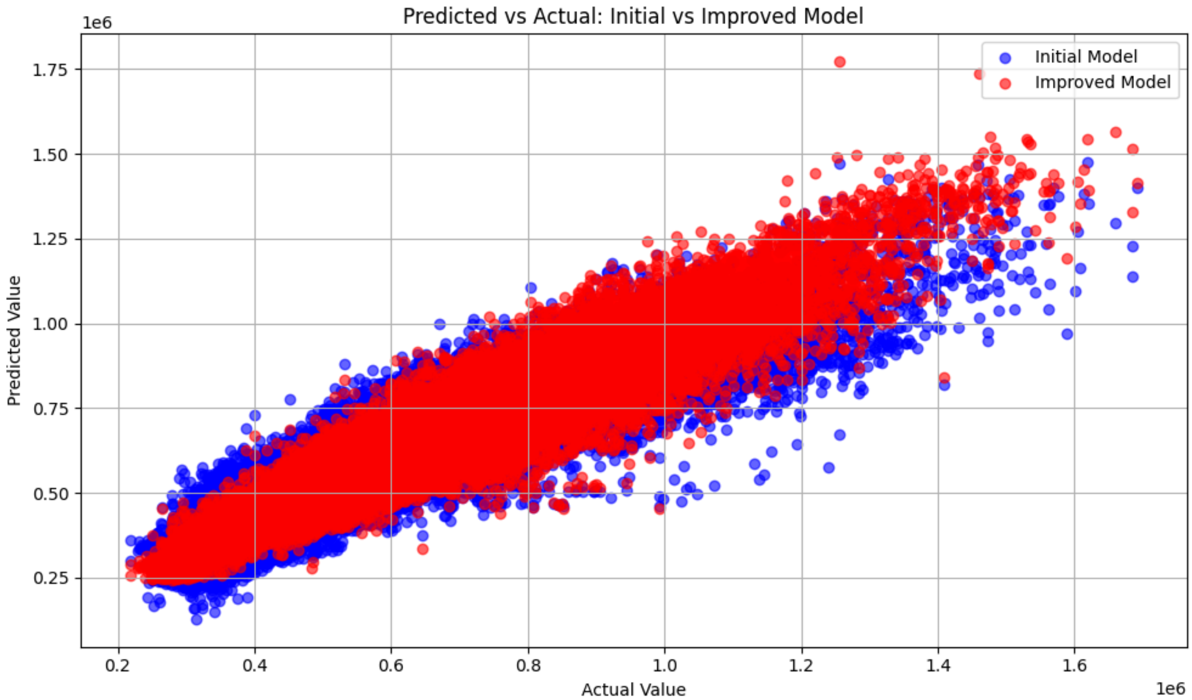


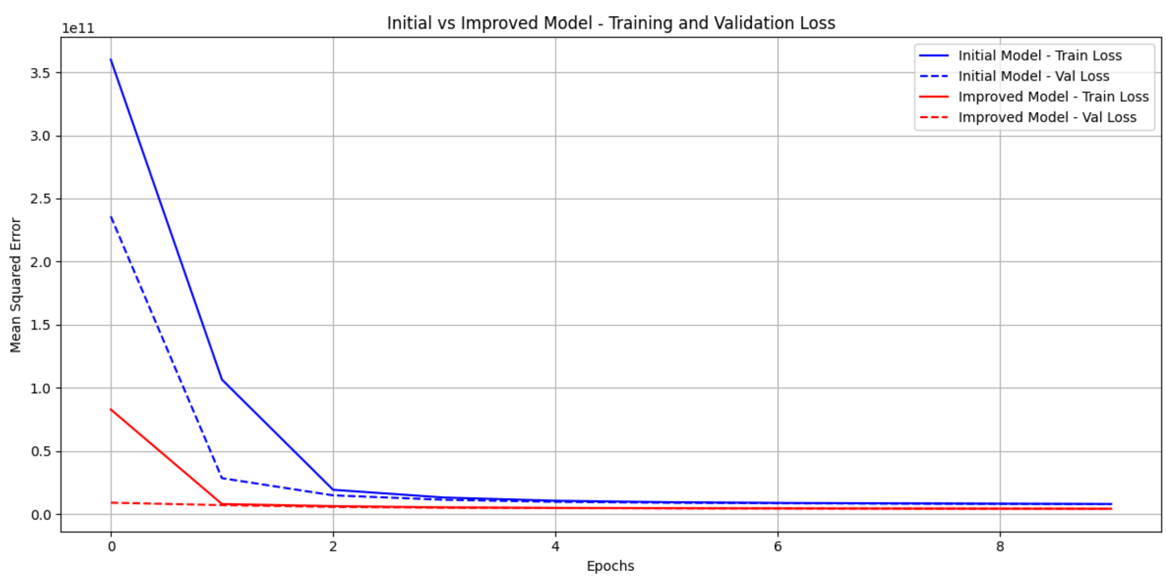

```
C:\Users\ASUS\Desktop\mmdt\Intro-to-Deep-Learning>
```

Visualisation:
The loss curves show faster and smoother convergence for the improved model. The predicted vs. actual scatter plot also indicates tighter clustering around the diagonal, meaning improved predictive accuracy. (Please see detailed visualisation on the next page▶️)

```
import data_config as cfg
import importlib     #importing needed to reconncect to the data_config module again and again
import data_config as cfg
importlib.reload(cfg)
# Plot training & validation loss for both models
cfg.plot_loss_comparison(history_initial, history_improved, fname='model_loss_comparison.png')
```

VISUALISATION





Therefore, adding more layers and neurons enhanced the model's ability to learn from the data, resulting in significantly lower error and improved R^2 scores on both training and test sets. Although training time increased slightly (~9 seconds), the performance gain justifies this trade-off. The deeper model captured more complex relationships, improving generalization and predictive power.