Experimenting with other advanced model

Read clean data

In [34]:
```
import pandas as pd

# Specify the path to your CSV file
file_path = 'Group_2_clean_Data..csv'

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

df.head()
```

Out[34]:

	communityname	State	countyCode	communityCode	fold	pop	perHoush	pctBlack	pctWhite	pctAsian	...	burglaries
0	149.0	28.0	55.0	509.0	1.0	11980.0	3.10	1.37	91.78	6.50	...	14.1
1	1034.0	35.0	58.0	424.0	1.0	23123.0	2.82	0.80	95.57	3.44	...	57.0
2	1780.0	34.0	114.0	959.0	1.0	29344.0	2.43	0.74	94.33	3.43	...	274.0
3	664.0	31.0	53.0	213.0	1.0	16656.0	2.40	1.70	97.35	0.50	...	225.0
4	140.0	22.0	82.0	471.0	1.0	11245.0	2.76	0.53	89.16	1.17	...	91.0

5 rows × 125 columns

In [ ]:
```
## we are going to use the features selected using wraping elimninative method
```

In [35]:
```
# using refined model
select_feature= ['persPoverty', 'persUrban', 'numForeignBorn', 'kidsBornNevrMarr', 'pop', 'houseVacant', 'persEmergSh
df_feature = df[select_feature]
df_target = df['burglaries']  # Select the 'violentPerPop' column as the target variable
```

```
df_feature.head(5)
```

Out[35]:

	persPoverty	persUrban	numForeignBorn	kidsBornNevrMarr	pop	houseVacant	persEmergShelt	persHomeless	pctLargH
0	227.0	11980.1	1277.0	31.0	11980.0	64.0	11.0	0.1	
1	885.0	23123.0	1920.0	43.0	23123.0	240.0	0.0	0.0	
2	1389.0	29344.0	1468.0	164.0	29344.0	544.0	16.0	0.0	
3	2831.0	0.0	339.0	561.0	16656.0	669.0	0.0	0.0	
4	2855.0	0.0	196.0	402.0	11245.0	333.0	2.0	0.0	

i XGBOOST model

In [5]:
```
import xgboost
print(xgboost.__file__)
```

```
None
```

In [6]:
```
df_target.shape
```

Out[6]:
```
(2215,)
```

In [9]:
```
import pandas as pd
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Refined feature set and target
select_features = ['persPoverty', 'persUrban', 'numForeignBorn', 'kidsBornNevrMarr', 'pop',
                   'houseVacant', 'persEmergShelt', 'persHomeless', 'pctLargHous', 'NAperCap']
df_feature = df[select_features]  # Select refined features
df_target = df['burglaries']  # Select the target variable

# Train-test split
X_train_selected, X_test_selected, y_train, y_test = train_test_split(
```

```
    df_feature, df_target, test_size=0.2, random_state=42
)

# Initialize and train the XGBoost model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model.fit(X_train_selected, y_train)

# Predictions
y_pred_train = xgb_model.predict(X_train_selected)  # Predictions on the training set
y_pred_test = xgb_model.predict(X_test_selected)  # Predictions on the test set

# Evaluate on training set
train_mse = mean_squared_error(y_train, y_pred_train)
train_r2 = r2_score(y_train, y_pred_train)

# Evaluate on test set
test_mse = mean_squared_error(y_test, y_pred_test)
test_r2 = r2_score(y_test, y_pred_test)

# Print evaluation metrics
print(f"XGBoost Model:")
print(f"Training Set Metrics:")
print(f"  - Mean Squared Error: {train_mse:.2f}")
print(f"  - R-squared: {train_r2:.2f}")
print(f"Testing Set Metrics:")
print(f"  - Mean Squared Error: {test_mse:.2f}")
print(f"  - R-squared: {test_r2:.2f}")
```

```
XGBoost Model:
Training Set Metrics:
  - Mean Squared Error: 16574.74
  - R-squared: 1.00
Testing Set Metrics:
  - Mean Squared Error: 201333.03
  - R-squared: 0.92
```

ii Extreme Learning Machine (ELM)

In [63]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

```
from sklearn.preprocessing import StandardScaler
from hpelm import ELM

# Split into training and testing sets first
X_train, X_test, y_train, y_test = train_test_split(
    df_feature.values, df_target.values, test_size=0.2, random_state=42
)

# Scale the features (X)
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)  # Fit on training data and transform it
X_test_scaled = scaler_X.transform(X_test)  # Transform test data using the same scaler

# Scale the target variable (y)
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1)).flatten()  # Fit and transform y_train
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).flatten()  # Transform y_test

# Initialize the ELM model for regression
elm_model = ELM(inputs=X_train_scaled.shape[1], outputs=1, classification="", accelerator="CPU")

# Add hidden neurons
elm_model.add_neurons(64, "tanh")  # Adjust the number of neurons as needed

# Train the ELM model
elm_model.train(X_train_scaled, y_train_scaled, "R")  # Train with scaled y_train

# Predict on the training and testing sets
y_train_scaled_pred = elm_model.predict(X_train_scaled).flatten()
y_test_scaled_pred = elm_model.predict(X_test_scaled).flatten()

# Inverse-transform predictions back to the original scale
y_train_pred = scaler_y.inverse_transform(y_train_scaled_pred.reshape(-1, 1)).flatten()
y_test_pred = scaler_y.inverse_transform(y_test_scaled_pred.reshape(-1, 1)).flatten()

# Evaluate the model using original-scale metrics
train_mse = mean_squared_error(y_train, y_train_pred)  # Use original-scale y
train_r2 = r2_score(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)  # Use original-scale y
test_r2 = r2_score(y_test, y_test_pred)

# Print the results
```

```
print("ELM Model Performance (Regression with Feature and Target Scaling):")
print("Training Set:")
print(f"  - Mean Squared Error: {train_mse:.2f}")
print(f"  - R-squared: {train_r2:.2f}")
print("Testing Set:")
print(f"  - Mean Squared Error: {test_mse:.2f}")
print(f"  - R-squared: {test_r2:.2f}")
```

```
ELM Model Performance (Regression with Feature and Target Scaling):
Training Set:
  - Mean Squared Error: 1264430.73
  - R-squared: 0.89
Testing Set:
  - Mean Squared Error: 795244.40
  - R-squared: 0.70
```

iii Simple Deep Learning Model with Two Layers

In [56]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    df_feature.values, df_target.values, test_size=0.2, random_state=42
)

# Scale the features (X)
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)  # Fit and transform the training set
X_test_scaled = scaler_X.transform(X_test)  # Transform the test set

# Scale the target variable (y)
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).flatten()
```

```
# Build a Multilayer Perceptron (MLP) with regularization
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],), kernel_regularizer=l2(0.005)),
    Dense(64, activation='relu', kernel_regularizer=l2(0.005)),
    Dense(1)  # Output layer for regression
])

# Compile the model with a smaller learning rate
model.compile(optimizer=Adam(learning_rate=0.0001), loss='mse', metrics=['mse'])

# Train the model
history = model.fit(X_train_scaled, y_train_scaled, epochs=50, batch_size=16, validation_split=0.1, verbose=1)

# Predict on the training and testing sets
y_train_scaled_pred = model.predict(X_train_scaled).flatten()
y_test_scaled_pred = model.predict(X_test_scaled).flatten()

# Inverse-transform predictions back to the original scale
y_train_pred = scaler_y.inverse_transform(y_train_scaled_pred.reshape(-1, 1)).flatten()
y_test_pred = scaler_y.inverse_transform(y_test_scaled_pred.reshape(-1, 1)).flatten()

# Evaluate the model using original-scale metrics
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

# Print the results
print("MLP Model Performance :")
print("Training Set:")
print(f"  - Mean Squared Error: {train_mse:.2f}")
print(f"  - R-squared: {train_r2:.2f}")
print("Testing Set:")
print(f"  - Mean Squared Error: {test_mse:.2f}")
print(f"  - R-squared: {test_r2:.2f}")

# Plot the training and validation loss
import matplotlib.pyplot as plt

train_loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
plt.figure(figsize=(10, 6))
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Over Epochs')
plt.legend()
plt.grid(True)
plt.show()
```

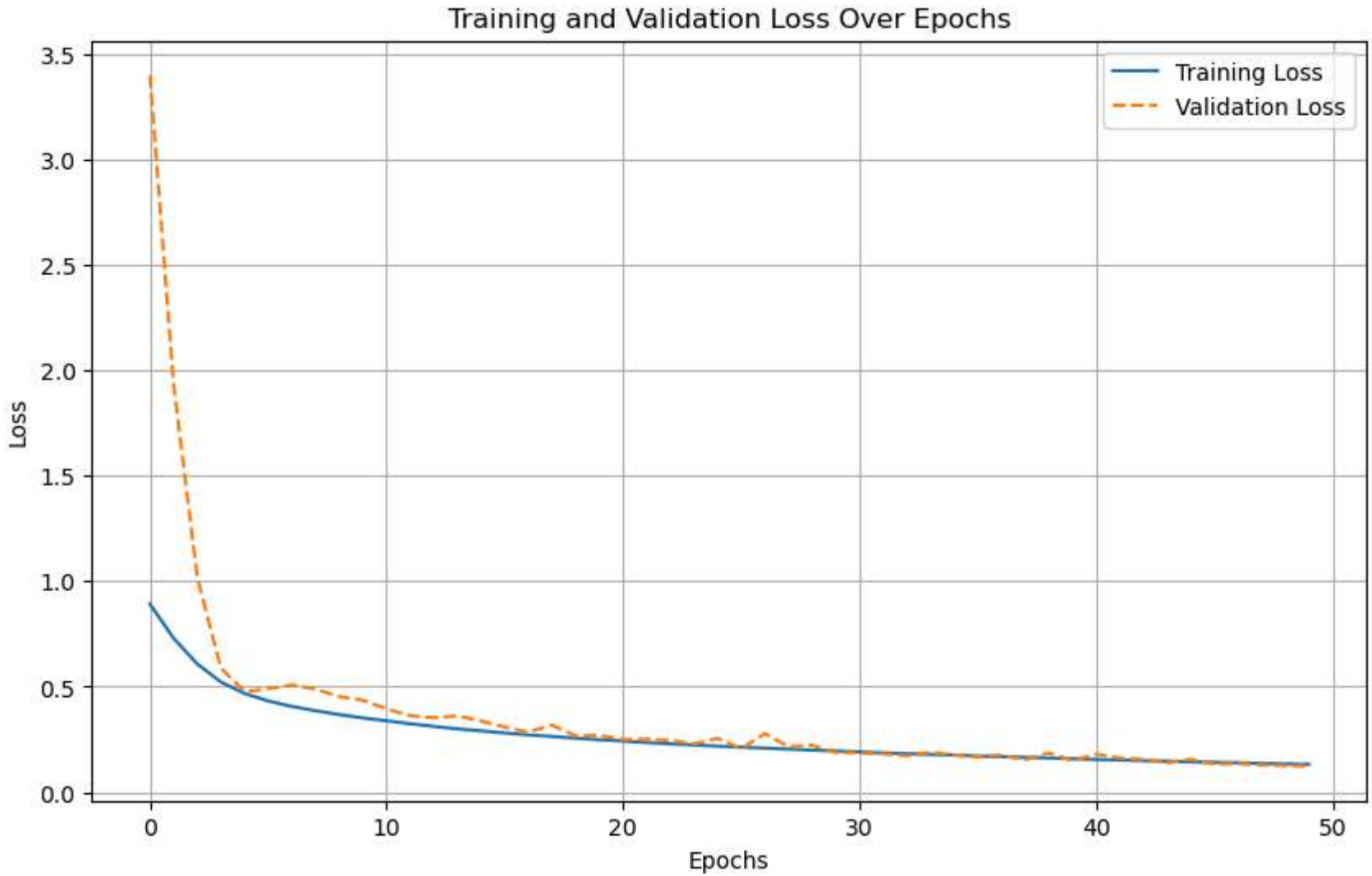
```
Epoch 1/50
```

```
C:\Users\hyz20\miniconda3\envs\gpu-env\lib\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)
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 1.0025 - mse: 0.4918 - val_loss: 3.3982 - val_mse: 2.9098
Epoch 2/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 604us/step - loss: 0.6984 - mse: 0.2172 - val_loss: 1.9356 - val_mse: 1.4751
Epoch 3/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 502us/step - loss: 0.6297 - mse: 0.1756 - val_loss: 1.0196 - val_mse: 0.5842
Epoch 4/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 564us/step - loss: 0.5223 - mse: 0.0929 - val_loss: 0.5881 - val_mse: 0.1760
Epoch 5/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 524us/step - loss: 0.4598 - mse: 0.0532 - val_loss: 0.4766 - val_mse: 0.0859
Epoch 6/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 516us/step - loss: 0.4268 - mse: 0.0412 - val_loss: 0.4904 - val_mse: 0.1195
Epoch 7/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 582us/step - loss: 0.4129 - mse: 0.0467 - val_loss: 0.5088 - val_mse: 0.1562
Epoch 8/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 513us/step - loss: 0.3889 - mse: 0.0407 - val_loss: 0.4881 - val_mse: 0.1524
Epoch 9/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 539us/step - loss: 0.3667 - mse: 0.0350 - val_loss: 0.4532 - val_mse: 0.1329
Epoch 10/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 500us/step - loss: 0.3494 - mse: 0.0328 - val_loss: 0.4357 - val_mse: 0.1296
Epoch 11/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 492us/step - loss: 0.3413 - mse: 0.0384 - val_loss: 0.3953 - val_mse: 0.1020
Epoch 12/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 509us/step - loss: 0.3229 - mse: 0.0328 - val_loss: 0.3627 - val_mse: 0.0815
Epoch 13/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 533us/step - loss: 0.3071 - mse: 0.0287 - val_loss: 0.3533 - val_mse: 0.0830
Epoch 14/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 590us/step - loss: 0.3050 - mse: 0.0373 - val_loss: 0.3619 - val_mse: 0.1017
Epoch 15/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 550us/step - loss: 0.2835 - mse: 0.0257 - val_loss: 0.3375 - val_mse: 0.0868
Epoch 16/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 512us/step - loss: 0.2809 - mse: 0.0324 - val_loss: 0.3086 - val_mse: 0.0668
Epoch 17/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 506us/step - loss: 0.2668 - mse: 0.0270 - val_loss: 0.2849 - val_mse: 0.0513
Epoch 18/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 507us/step - loss: 0.2640 - mse: 0.0323 - val_loss: 0.3178 - val_mse: 0.0917
Epoch 19/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 579us/step - loss: 0.2540 - mse: 0.0296 - val_loss: 0.2668 - val_mse: 0.0478
Epoch 20/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 555us/step - loss: 0.2426 - mse: 0.0252 - val_loss: 0.2695 - val_mse: 0.0571
Epoch 21/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 515us/step - loss: 0.2407 - mse: 0.0299 - val_loss: 0.2503 - val_mse: 0.0440
Epoch 22/50
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 517us/step - loss: 0.2331 - mse: 0.0283 - val_loss: 0.2517 - val_mse: 0.0513
Epoch 23/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 494us/step - loss: 0.2382 - mse: 0.0390 - val_loss: 0.2461 - val_mse: 0.0510
Epoch 24/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 523us/step - loss: 0.2241 - mse: 0.0304 - val_loss: 0.2290 - val_mse: 0.0393
Epoch 25/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 503us/step - loss: 0.2198 - mse: 0.0314 - val_loss: 0.2549 - val_mse: 0.0700
Epoch 26/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 583us/step - loss: 0.2203 - mse: 0.0366 - val_loss: 0.2068 - val_mse: 0.0269
Epoch 27/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 511us/step - loss: 0.2078 - mse: 0.0290 - val_loss: 0.2777 - val_mse: 0.1022
Epoch 28/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 515us/step - loss: 0.2052 - mse: 0.0308 - val_loss: 0.2159 - val_mse: 0.0447
Epoch 29/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 504us/step - loss: 0.2028 - mse: 0.0327 - val_loss: 0.2229 - val_mse: 0.0559
Epoch 30/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 509us/step - loss: 0.1896 - mse: 0.0237 - val_loss: 0.1876 - val_mse: 0.0250
Epoch 31/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 525us/step - loss: 0.1945 - mse: 0.0328 - val_loss: 0.1863 - val_mse: 0.0274
Epoch 32/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 514us/step - loss: 0.1828 - mse: 0.0249 - val_loss: 0.1812 - val_mse: 0.0262
Epoch 33/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 510us/step - loss: 0.1851 - mse: 0.0310 - val_loss: 0.1725 - val_mse: 0.0213
Epoch 34/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 526us/step - loss: 0.1776 - mse: 0.0273 - val_loss: 0.1889 - val_mse: 0.0412
Epoch 35/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 489us/step - loss: 0.1714 - mse: 0.0244 - val_loss: 0.1757 - val_mse: 0.0314
Epoch 36/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 626us/step - loss: 0.1728 - mse: 0.0293 - val_loss: 0.1670 - val_mse: 0.0262
Epoch 37/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 561us/step - loss: 0.1829 - mse: 0.0430 - val_loss: 0.1761 - val_mse: 0.0385
Epoch 38/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 565us/step - loss: 0.1625 - mse: 0.0256 - val_loss: 0.1540 - val_mse: 0.0197
Epoch 39/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 516us/step - loss: 0.1637 - mse: 0.0302 - val_loss: 0.1858 - val_mse: 0.0544
Epoch 40/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 506us/step - loss: 0.1591 - mse: 0.0284 - val_loss: 0.1505 - val_mse: 0.0221
Epoch 41/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 445us/step - loss: 0.1610 - mse: 0.0334 - val_loss: 0.1803 - val_mse: 0.0547
Epoch 42/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 508us/step - loss: 0.1513 - mse: 0.0264 - val_loss: 0.1621 - val_mse: 0.0392
Epoch 43/50
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 457us/step - loss: 0.1533 - mse: 0.0312 - val_loss: 0.1558 - val_mse: 0.0357
Epoch 44/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 517us/step - loss: 0.1395 - mse: 0.0201 - val_loss: 0.1386 - val_mse: 0.0212
Epoch 45/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 513us/step - loss: 0.1427 - mse: 0.0260 - val_loss: 0.1564 - val_mse: 0.0415
Epoch 46/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 537us/step - loss: 0.1434 - mse: 0.0291 - val_loss: 0.1337 - val_mse: 0.0214
Epoch 47/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 514us/step - loss: 0.1430 - mse: 0.0314 - val_loss: 0.1350 - val_mse: 0.0250
Epoch 48/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 517us/step - loss: 0.1299 - mse: 0.0204 - val_loss: 0.1289 - val_mse: 0.0213
Epoch 49/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 555us/step - loss: 0.1362 - mse: 0.0292 - val_loss: 0.1255 - val_mse: 0.0202
Epoch 50/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 520us/step - loss: 0.1292 - mse: 0.0243 - val_loss: 0.1236 - val_mse: 0.0204
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 538us/step
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 385us/step
MLP Model Performance (With Fixes):
Training Set:
  - Mean Squared Error: 291203.40
  - R-squared: 0.97
Testing Set:
  - Mean Squared Error: 194467.46
  - R-squared: 0.93
```



iv Ensemble Model Combining the Top 3 Models

In [ ]:
```
import numpy as np

# Combine predictions using a weighted average (adjust weights based on performance)
weights = [0.5, 0.3, 0.2]  # Example weights
y_pred_ensemble = (weights[0] * y_pred_xgb +
```

```
                  weights[1] * y_pred_elm +
                  weights[2] * y_pred_dl)

# Evaluate
ensemble_mse = mean_squared_error(y_test, y_pred_ensemble)
ensemble_r2 = r2_score(y_test, y_pred_ensemble)

print(f"Ensemble Model:")
print(f"  - Mean Squared Error: {ensemble_mse:.2f}")
print(f"  - R-squared: {ensemble_r2:.2f}")
```