Final Report

30 November 2024

Table 1: Group 2

Name of the Student	UH ID
Yuzhen Hu	2299391
Duggimpudi Bala Meghana	2275900
Murtaza Mustafa	2415417
Muhammad Asad Rehan	1951934

Abstract

This report presents the development of a predictive model using the **Communities and Crime Unnormalized Dataset** from the UCI Machine Learning Repository, which includes 125 attributes related to socio-economic factors, racial distribution, and law enforcement resources. The objective was to predict the rate of **burglaries** (target variable: *bulgaries*) across U.S. communities. Due to the dataset's high dimensionality and unnormalized values, **preprocessing** and **feature selection** were crucial for improving model performance.

Several regression models were evaluated, including **Linear Regression**, **Decision Tree Regressor**, **Random Forest Regressor**, **Support Vector Regressor (SVR)**, and **K-Nearest Neighbors (KNN)**. Performance was further optimized through **feature selection** and **hyperparameter tuning**, with the **wrapper method** applied for feature refinement to enhance predictive accuracy.

In the subsequent phase, the refined model's performance was assessed using advanced techniques, such as **XGBoost**, **Extreme Learning Machine (ELM)**, **simple deep learning models**, and **ensemble methods**. These models were compared to identify the key socio-economic and law enforcement factors influencing burglary rates.

The findings highlight how **data-driven insights**, especially from advanced ensemble models, can inform **policy decisions** and guide **resource allocation** for effective crime prevention.

1 Introduction

Predicting crime rates is crucial for law enforcement and policymakers. The **Communities and Crime Unnormalized Dataset** from the UCI Machine Learning Repository

provides socio-economic, demographic, and law enforcement data from U.S. communities, with 125 attributes such as poverty rates, racial distribution, and police funding. This analysis aims to predict burglary rates (target variable: *bulgaries*) and uncover patterns that can inform crime prevention strategies.

Due to the dataset's high dimensionality and unnormalized values, careful preprocessing is necessary, including handling missing data, transforming skewed variables, and performing **feature selection** to enhance model accuracy. Various models, including **Linear Regression**, **Decision Tree Regressor**, **Random Forest Regressor**, **Support Vector Regressor (SVR)**, and **K-Nearest Neighbors (KNN)**, were initially evaluated to predict burglary rates.

However, basic models struggled to fully capture the complex relationships within the data. To improve performance, **feature selection** and **hyperparameter tuning** were applied, followed by **wrapper method** refinement to select the most relevant features.

Finally, the performance of refined models was compared with advanced techniques, including **XGBoost**, **Extreme Learning Machine (ELM)**, and **ensemble learning**, providing deeper insights into the socio-economic and law enforcement factors affecting burglary rates. This report demonstrates how machine learning can inform crime prevention strategies and improve predictive accuracy through model refinement and advanced techniques.

2 Data Cleaning

2.1 Handling Missing Data

Missing data is a common issue in many real-world datasets. In this project, we identified columns with missing values using a straightforward method to calculate the percentage of missing values for each column. Columns with missing data exceeding 80% were either dropped or imputed based on their relevance to the analysis.

2.2 Imputation of Missing Values

For columns with less than 80% missing values, the following imputation methods were applied:

- **Numerical Columns**: The **median** was used to fill in missing values, as it is less sensitive to outliers and provides a robust estimate of the central tendency.
- **ID-related Columns**: For columns representing categories or identifiers, **KNN imputation** was used to estimate missing values based on the similarity of records, preserving inherent relationships within the data.
- **Categorical Columns**: **Label Encoding** was applied to convert categorical variables into numerical format, ensuring compatibility with machine learning models.

3 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was conducted to gain insights into the data and understand its structure. During EDA, special attention was given to the **target variable**,

bulgaries, and its distribution. Key correlations between features and the target were visualized, and potential outliers were identified.

3.1 Feature Distribution

In this subsection, we examine the distribution of each selected feature to understand their characteristics and potential challenges for modeling. Key observations from the feature distributions include:

- **Skewed Distributions**: Many features exhibited skewed distributions, particularly those with long right tails as shown in **Figure 1.** For example:
  - **persPoverty**, **kidsBornNevrMarr**, and **numForeignBorn** displayed long right tails, indicating that most values are concentrated at the lower end, with a few extreme values.
  - **persHomeless** had a large concentration of zeros, suggesting that many records report no homelessness.
- **Feature Concentration**: Some features, such as **persEmergShelt**, showed a concentration of values around zero, indicating that the majority of communities have relatively low emergency shelter presence compared to others.
- **Outliers**: Several features, such as **numForeignBorn** and **houseVacant**, exhibited clear outliers, suggesting that a small number of communities have extreme values compared to the rest.

These observations suggest that certain features may need transformation or special handling to address skewness, outliers, and imbalanced distributions, which could impact the predictive power of the model.

.

3.2 Target Variable Exploration: *bulgaries*

The target variable, *bulgaries*, exhibited a right-skewed distribution as shown in **Figure 2**, with most values concentrated at the lower end and a few high-value outliers. This is typical in crime-related datasets, where most regions experience few burglaries, but certain areas have significantly higher frequencies.

To address the skewness, several transformations were applied:

- **Log-transformation**: To handle zero or near-zero values and reduce the impact of extreme values.
- **Square root-transformation**: To moderate the effect of large values while preserving the distribution.
- **Box-Cox transformation**: To find an optimal transformation that stabilizes variance and reduces skewness.

Histograms and KDE plots were generated to compare the original and transformed distributions, assessing the effectiveness of each transformation in preparing the data for modeling.

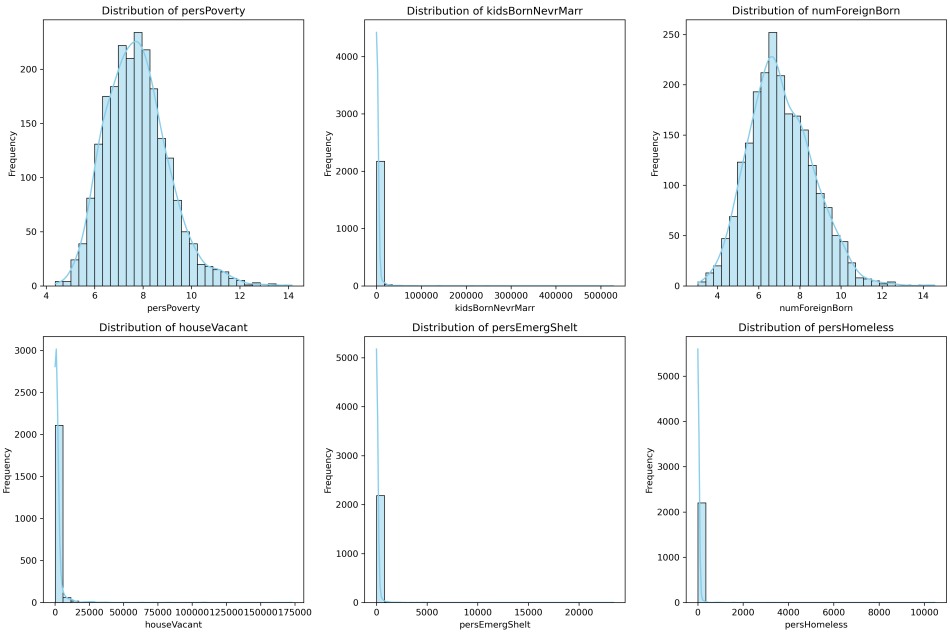


Figure 1: observe the feature dsitribution ,as we can there's a lot skewness in features

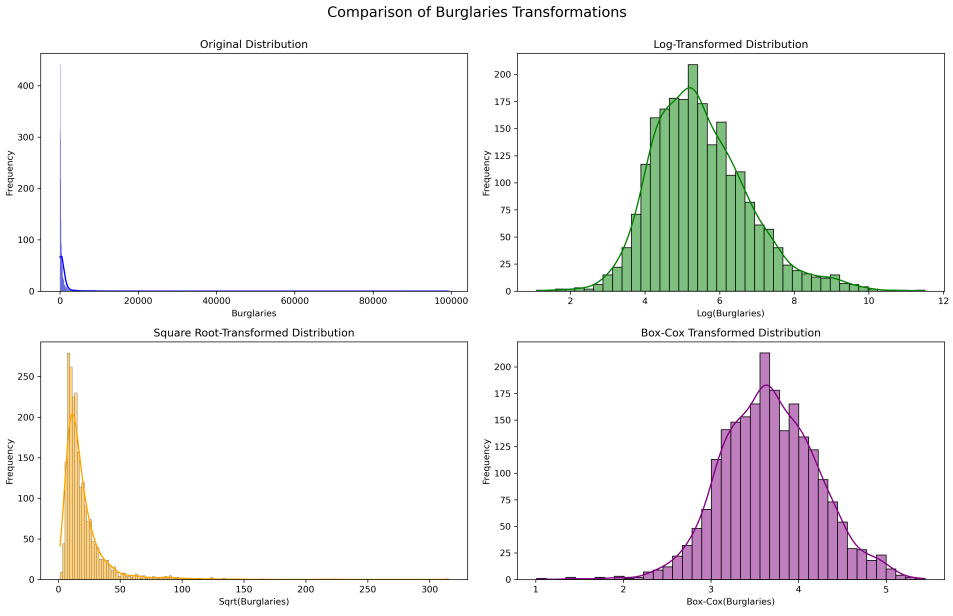


Figure 2: A close-up view of the log-transformation and other transformation showing the reduced skewness in the distribution of target variable 'Bulgaries'

4 Train Test split and evaluation Metric

4.1 Train-Test Split

The data is divided into training and testing sets, with an 80-20 split as shown in Table 2. The training set is used to fit the model, while the test set is reserved for evaluating the model's generalization performance. During training, cross-validation is employed to select optimal hyperparameters, ensuring that the model performs well across different subsets of the training data and avoids overfitting. This approach helps identify the best configuration for the model before evaluating it on the unseen test data.

Dataset	**Shape (Rows, Columns)**
Train Dataset	(1772, 120)
Test Dataset	(443, 120)

Table 2: Train and Test Datasets split.

4.2 Evaluation Metrics

Since this is a regression problem, we use metrics specifically designed to evaluate regression models, including the Root Mean Squared Error (RMSE) and the R^2 score, to assess the model's performance on both the training and test sets for each target variable.

The RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i are the actual values, $\hat{y}_i$ are the predicted values, and n is the number of observations. RMSE measures the average magnitude of prediction errors, with lower values indicating better model performance.

The R^2 score, also known as the coefficient of determination, is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the actual values. The R^2 score represents the proportion of variance in the target variable that is explained by the model.

- An R^2 value of **1** indicates a perfect fit, meaning the model explains all the variability in the target variable, with no prediction error.
- An R^2 value of **0** suggests that the model performs no better than simply predicting the mean of the target variable ($\bar{y}$).
- A negative R^2 value implies that the model performs worse than the mean predictor, indicating poor predictive performance.

By comparing the training and test RMSE values, as well as the R^2 scores, we can evaluate the model's ability to generalize. Significant differences between training and test RMSE values may indicate overfitting, while low R^2 scores suggest that the model fails to adequately capture the relationship between features and the target variable.

5 Model Evaluation and Optimization

The objective of this section is to build and evaluate regression models on the provided dataset. The goal is to assess the performance of various algorithms using key metrics such as **Mean Squared Error (MSE)** and R^2. **Feature selection** and **hyperparameter tuning** will be key steps in refining the models, ensuring that only the most relevant features are used and that each model is optimized for its best performance.

We begin by training the models without any feature selection as a baseline, then apply **feature selection** methods (such as using the top 40 important features for **Random Forest** and **Lasso regression** with regularization) to refine the inputs. Finally, **hyperparameter tuning** will be performed using **cross-validation** to optimize each model's performance.

The models implemented include:

- **Linear Regression**
- **K-Nearest Neighbors (KNN)**
- **Random Forest**
- **Support Vector Machine (SVM)** with both linear and non-linear kernels
- **Decision Tree**

The results of each model, both before and after **feature selection** and **hyperparameter tuning**, will be presented and analyzed to emphasize the impact of these steps on model performance.

5.1 Overview of Regression Model Performance

Model	**Train MSE**	**Train R^2**	**Test MSE**	**Test R^2**
Linear Regression (After FS)	381,464.09	0.97	218,079.41	0.92
Decision Tree (Tuned)	1,124,797.91	0.90	692,712.39	0.74
Random Forest (After FS)	468,989.46	0.96	**194,441.16**	**0.93**
Support Vector Regressor (Tuned)	2,324,247.54	0.80	312,782.60	0.88
K-Nearest Neighbors (After FS)	2,444,938.88	0.79	479,995.23	0.82

Table 3: Comparison of models based on training and testing performance. **Random Forest** achieved the best test performance with the lowest test MSE and highest R^2 among all models.

Table 3 summarizes the performance of various regression models based on Mean Squared Error (MSE) and R^2 for both training and testing datasets. The **best performance** for each model was achieved either through feature selection or hyperparameter tuning. Among all the models, **Random Forest** showed the best test performance, with an R^2 of 0.93 and a test MSE of 0.06. **K-Nearest Neighbors** demonstrated the weakest generalization, with an R^2 of 0.69 and a test MSE of 0.27. **Linear Regression** and **Decision Tree** performed similarly well, with Linear Regression achieving an R^2 of 0.92 and a test MSE of 0.09, while the Decision Tree model, after hyperparameter tuning, had an R^2 of 0.74 and a test MSE of 0.69. These models were optimized using feature selection or hyperparameter tuning for better performance.

5.2 Linear Regression

Linear Regression was used as the baseline model in this study. By modeling the relationship between features and the target variable as a linear equation, it offered a straightforward yet effective approach to understanding the dataset. Despite its simplicity, Linear Regression exhibited robust performance, serving as a reliable benchmark for evaluating the effectiveness of more complex models.

Stage	**Train MSE**	**Train R^2**	**Test MSE**	**TestR^2**
Before Feature Selection	330,376.16	0.97	260,139.59	0.90
After Feature Selection	381,464.09	0.97	218,079.41	0.92
After Hyperparameter Tuning	381,601.03	0.97	217,511.00	0.92

Table 4: Performance summary of **Linear Regression** at different stages. Best hyperparameter for tuning: `alpha=1000`.

5.3 Decision Tree Regressor

The Decision Tree Regressor uses tree-based splits to minimize variance in the target variable. It is capable of capturing non-linear relationships and interactions among features. However, it is prone to overfitting if not properly tuned, making hyperparameter optimization crucial for achieving generalization.

Stage	**MSE(Train)**	**R^2(Train)**	**MSE(Test)**	**R^2(Test)**
Before Feature Selection	0.00	1.00	1,092,504.58	0.58
After Feature Selection(top 40)	0.00	1.00	799,246.49	0.70
After Hyperparameter Tuning	1,124,797.91	0.90	692,712.39	0.74

Table 5: Performance summary of **DecisionTreeRegressor** with Top 40 Features and Hyperparameter Tuning. Best hyperparameters: `min_samples_split=10`, `min_samples_leaf=3`, `max_features=None`, `max_depth=19`, `criterion='friedman_mse'`.

5.4 Random Forest Regressor

The Random Forest Regressor is an ensemble learning method that combines multiple decision trees to enhance predictive accuracy and mitigate overfitting. It achieves this by constructing a multitude of decision trees during training and averaging their predictions, which reduces variance and improves robustness. This approach is particularly effective for datasets with complex relationships, non-linear interactions, and features of varying importance.

Experiments Conducted: To evaluate the performance of the Random Forest Regressor, the following experiments were performed:

- **Before Feature Selection:** The model was trained on the full feature set to establish baseline performance metrics for training and testing data.

- **Feature Selection:** The first Random Forest model was used to rank feature importance, and the top 40 features were selected using Recursive Feature Elimination (RFE). This refined feature set was used to retrain the model and analyze the impact of feature reduction.
- **After Hyperparameter Tuning:** Hyperparameter tuning was performed on the Random Forest model with the top 40 features using a grid search. Key hyperparameters such as the number of trees (`n_estimators`), maximum tree depth (`max_depth`), and minimum samples per leaf (`min_samples_leaf`) were optimized to achieve the best performance.

The results of these experiments, including training and testing performance metrics (MSE and R^2), are summarized in Table 6. These experiments highlight the effectiveness of feature selection and hyperparameter tuning in improving model performance while maintaining a balance between accuracy and generalization.

Stage	**MSE (Train)**	**MSE (Test)**	R^2 **(Train)**	R^2 **(Test)**
Before Feature Selection	515,588.70	232,487.27	0.95	0.91
After Feature Selection (Top 40)	468,989.46	194,441.16	0.96	**0.93**
After Hyperparameter Tuning	1,177,860.33	224,159.18	0.90	0.91

Table 6: Performance of the **Random Forest model** before feature selection, after feature selection, and after hyperparameter tuning. The best hyperparameters after tuning were: 'max_depth'=10, 'min_samples_leaf'=2, 'min_samples_split'=5, 'n_estimators'=150.

5.5 Support Vector Regressor (SVR)

The **Support Vector Regressor** (SVR) is a robust machine learning algorithm designed for regression tasks. It models data within a specified margin of tolerance (ϵ) and utilizes kernel functions, such as the Radial Basis Function (RBF), to capture non-linear relationships effectively. SVR is particularly suitable for datasets with complex patterns and interactions. However, its computational complexity increases significantly with larger datasets, necessitating careful preprocessing and hyperparameter optimization for efficient implementation.

Experiments Conducted:

- **Before Feature Selection:** The SVR model was trained using the full feature set to establish baseline performance metrics.
- **After Feature Selection:** Lasso regression was applied to select the most important features, and the SVR model was retrained on the reduced feature set to evaluate the impact of feature refinement.
- **After Hyperparameter Tuning:** Grid search was employed to optimize SVR hyperparameters (`C`, `epsilon`, and `kernel`). The tuned model was evaluated on the Lasso-selected feature set.

Stage	MSE (Train)	MSE (Test)	R^2 (Train)	R^2 (Test)
Before Feature Selection	7,671,441.42	281,643.38	0.33	0.89
After Feature Selection (Lasso)	7,671,441.42	281,643.38	0.33	0.89
After Hyperparameter Tuning (Lasso)	2,324,247.54	312,782.60	0.80	0.88
Best Hyperparameters: 'C'=10, 'epsilon'=0.1, 'kernel'='rbf'				

Table 7: Performance of the SVR model with Lasso-selected features before and after feature selection, and after hyperparameter tuning.

5.6 K-Nearest Neighbors (KNN) Regressor

The K-Nearest Neighbors (KNN) Regressor is a non-parametric machine learning algorithm that predicts the target value by averaging the values of the k nearest neighbors in the feature space. Known for its simplicity and interpretability, KNN is effective for small and moderately sized datasets with well-behaved features. However, its performance can degrade in high-dimensional spaces due to the curse of dimensionality and sensitivity to hyperparameter choices, such as the number of neighbors (k) and the distance metric.

Stage	MSE (Train)	R^2 (Train)	MSE (Test)	R^2 (Test)
Without Feature Selection	2,437,784.33	0.79	486,828.85	0.81
After Feature Selection (Lasso)	2,444,938.88	0.79	479,995.23	0.82
After Hyperparameter Tuning	0.00	1.00	454,341.28	0.83

Table 8: Performance of **KNN Regressor** without feature selection, after Lasso feature selection, and after hyperparameter tuning (RandomizedSearchCV). Best hyperparameters: metric = euclidean, 'neighbors = 5, weights = distance.

6 Bi-Directional Elimination for Feature Refinement

6.1 Objective

Two feature refinement techniques—Recursive Feature Elimination (RFE) and Sequential Feature Selection (SFS)—were applied to optimize the feature set for improved regression performance. Both methods aim to reduce the number of features while maintaining a balance between simplicity and predictive accuracy.

6.2 Methodology

1. **Initial Feature Selection:** The top 30 features were selected based on feature importance scores obtained from Random Forest.

2. **Recursive Feature Elimination (RFE):** RFE ranks features by recursively removing the least important features and evaluating the model's performance at each step.

3. **Sequential Feature Selection (SFS):** SFS refines the feature set iteratively by adding or removing features in a bi-directional manner. The refinement is guided by R^2 scores using 5-fold cross-validation.

4. **Stopping Criterion:** For both RFE and SFS, the refinement process terminated when further feature changes failed to improve cross-validated R^2, resulting in an optimal feature subset.

6.3 Results

Table 9 compares the test performance metrics and the number of features after applying RFE and SFS. Both methods resulted in compact feature sets with slightly different performance outcomes.

Metric	Before Refinement	After RFE	After SFS
Number of Features	40	12	15
Mean Squared Error	190,513.61	265,314.22	273,425.36
R^2	0.92	0.91	0.90

Table 9: Comparison of test performance metrics and the number of features before and after refinement using Recursive Feature Elimination (RFE) and Sequential Feature Selection (SFS).

Key Observations:

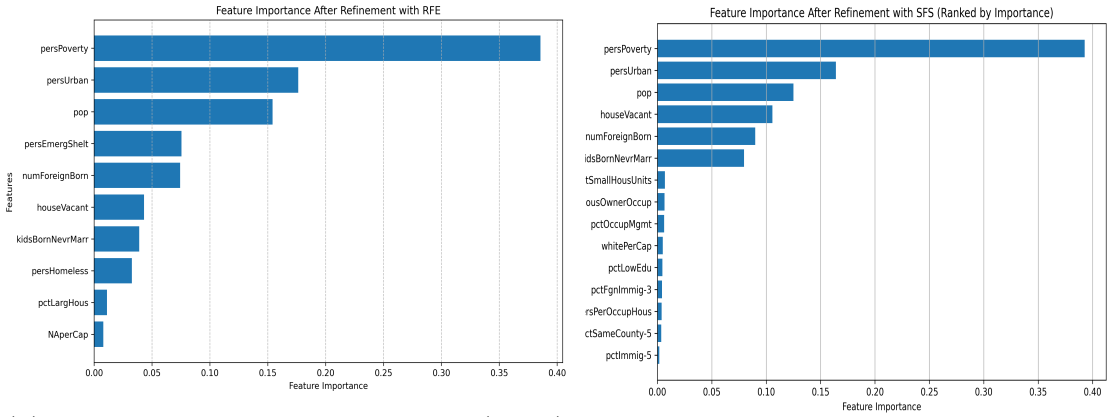
- **Feature Reduction:** Both methods significantly reduced the feature set, with RFE selecting 10 features and SFS refining the set to 15 features.
- **Test Performance:** RFE achieved slightly better test performance ($R^2 = 0.91$) compared to SFS ($R^2 = 0.90$), despite using fewer features.
- **Method Comparison:** RFE resulted in a smaller feature set and superior test performance, making it a better choice for balancing simplicity and predictive accuracy. SFS, while retaining more features, still achieved strong generalization.

6.4 Visualization

The feature importance of the refined feature set is visualized in Figure 3. This visualization ranks the selected features based on their contribution to the model's performance. By focusing on the most important predictors, the refined feature set not only enhances model interpretability but also reduces complexity. The reduced feature set provides a clear understanding of the key factors driving the predictions.

7 Additional Models After Feature Selection

This section explores the implementation and evaluation of various regression models using the 10 refined features identified through Recursive Feature Elimination (RFE). These



(a) Recursive Feature Elimination (RFE) Method

(b) Sequential Forward Selection (SFS) Method

Figure 3: Comparison of feature refinement using Recursive Feature Elimination (RFE) and Sequential Forward Selection (SFS). The left plot shows the RFE method, which iteratively removes features to refine the model, while the right plot shows the SFS method, which adds features sequentially to optimize performance.

refined features serve as the foundation for experiments to analyze performance improvements across different models. Each model is tailored for regression tasks, leveraging the reduced feature set to enhance interpretability and efficiency.

The following models are implemented and evaluated:

- **XGBoost:** A gradient boosting algorithm known for its efficiency and scalability in regression tasks.
- **Extreme Learning Machine (ELM):** A single-layer feedforward neural network designed for rapid training and experimentation.
- **Simple Deep Learning Model:** A feedforward neural network with two hidden layers, incorporating L2 regularization and a `ReLU` activation function.
- **Ensemble Model:** Combines predictions from the top three performing models (XGBoost, ELM, and Simple Deep Learning) using a linear regressor for aggregation.

Below, we describe and evaluate the performance of these models using the refined feature set.

7.1 XGBoost

XGBoost (eXtreme Gradient Boosting) is a scalable and efficient gradient-boosting algorithm that constructs an ensemble of weak learners (typically decision trees) in an iterative manner to minimize prediction error. Known for its speed and performance, XGBoost is highly effective on large datasets and offers robust results through advanced features like regularization, handling missing values, and parallel processing.

Model Performance:

Metric	Training	Testing
Mean Squared Error	16,574.74	201,333.03
R^2	1.00	0.92

7.2 Extreme Learning Machine (ELM)

Extreme Learning Machine (ELM) is a single-layer feedforward neural network where the hidden layer weights are randomly initialized and remain fixed during training. ELM is computationally efficient, making it suitable for rapid experimentation and scenarios where training time is critical. In this study, we used 100 neurons in the hidden layer and the `tanh` activation function.

Model Performance:

Metric	Training	Testing
Mean Squared Error	1,264,430.73	795,244.40
R^2	0.89	0.70

Table 10: Performance metrics of the ELM model with regression using feature and target scaling. The model uses 100 hidden neurons and the `tanh` activation function.

7.3 Simple Deep Learning Model

A simple feedforward neural network with two hidden layers (128 and 64 neurons) was developed to serve as a baseline for deep learning performance in regression tasks. The model uses `ReLU` activation, L2 regularization (0.005), a learning rate of 0.0001, and feature scaling for preprocessing.

Model Performance:

Metric	Training	Testing
Mean Squared Error	291,203.40	194,467.46
R^2	0.97	0.93

Table 11: Performance metrics of the simple deep learning model with two hidden layers (128 and 64 neurons), `ReLU` activation, L2 regularization (0.005), feature scaling, and a learning rate of 0.0001 over 50 epochs.

7.4 Ensemble Model

An ensemble model was constructed by aggregating the predictions of the top three performing models (XGBoost, ELM, and Deep Learning) using a linear regressor. This approach allows the model to learn the optimal combination of predictions from each individual model, leveraging their strengths for improved predictive performance.

Model Performance:

Metric	Training	Testing
Mean Squared Error	342,782.71	187,317.39
R^2	0.97	0.93

Table 12: Performance metrics of the ensemble model created by combining predictions from XGBoost, ELM, and Deep Learning using a linear regressor.

8 Conclusion

The evaluation of the four models highlights the following key insights:

- **XGBoost** demonstrated robust performance without the need for feature scaling, achieving strong results with minimal tuning.
- **ELM** struggled with generalization, producing the lowest test R^2 (0.70) and the highest test MSE, making it less effective for this task.
- **Simple Deep Learning Model** required feature scaling and L2 regularization to achieve good results, demonstrating strong generalization with a test R^2 of 0.93.
- **Ensemble Model**, combining predictions from the top three models, achieved improved generalization and predictive performance, making it the most effective approach overall.

In summary, XGBoost proved robust without additional preprocessing, while the Ensemble Model leveraged the strengths of multiple models to deliver superior generalization and regression performance.

9 Visualizing Model Performance and Insights

9.1 Overview

The goal of this section is to analyze and compare the performance of all implemented models using appropriate metrics and visualizations. Metrics such as Mean Squared Error (MSE) and R^2 are used to evaluate model performance, with visual tools like bar plots providing an intuitive way to identify the best-performing models.

9.2 Key Results

Table 13 summarizes the performance metrics (MSE and R^2) for all models. Figure 4 visually compares the performance of the models, highlighting differences in training and testing metrics.

9.3 Discussion and Conclusion

The performance of various models was analyzed in terms of both their training and testing results. Key insights drawn from the comparison are as follows:

Model	Tr. MSE	Tr. R^2	Test MSE	Test R^2
Random Forest	497077.62	0.96	226357.28	0.91
XGBoost	16574.74	1.00	201333.03	0.92
Extreme Learning Machine (ELM)	1264430.73	0.89	795244.40	0.70
Simple Deep Learning Model	291203.40	0.97	194467.46	**0.93**
Stacked Ensemble Model	342782.71	0.97	187317.39	**0.93**

Table 13: Comparison of Performance Metrics for all the Best Models.

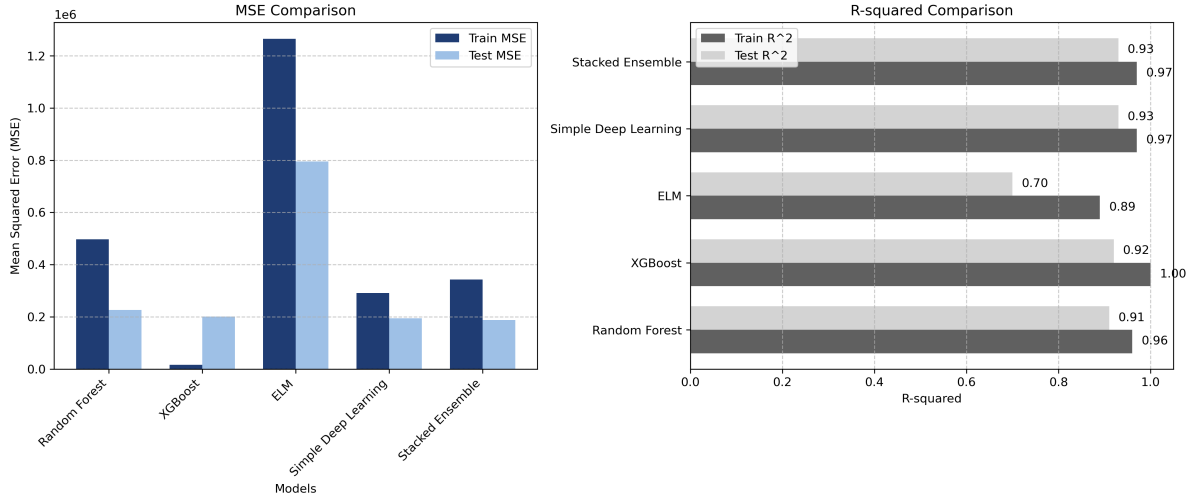


Figure 4: Visualization of Model Performance Metrics: MSE and R^2 values for different models

- **Best Generalization:** The **Stacked Ensemble** and **Simple Deep Learning Model** showed the best balance between training and test performance, with high test R^2 values (0.93), indicating strong generalization.
- **Overfitting: XGBoost** achieved perfect training performance but slightly underperformed on test data, suggesting potential overfitting.
- **ELM Challenges: ELM** struggled with generalization, with a low test R^2 of 0.70 and high test MSE, indicating it's less effective than other models.
- **Deep Learning Advantage:** The **Simple Deep Learning Model** performed well with a high R^2 (0.93), outperforming traditional models like **ELM** and competing well with the ensemble model.
- **Ensemble Learning:** The **Stacked Ensemble** combined the strengths of multiple models, yielding the best overall performance in terms of accuracy and generalization.
- **Random Forest: Random Forest** performed solidly with a test R^2 of 0.91, but it lagged behind deep learning and ensemble models in this task.

In conclusion, **deep learning** and **ensemble models** provide the best generalization and predictive power. While XGBoost excels in training performance, it may suffer from overfitting, and **ELM** struggles with generalization. **Random Forest** remains a reliable and solid choice, though it is outperformed by the deep learning and ensemble approaches.

10 Key Features and Policy Implications

The following features were found to influence burglary rates:

- **Poverty Rate (*persPoverty*)**: High poverty correlates with increased burglaries, suggesting a need for economic intervention.
- **Unmarried Parents (*kidsBornNevrMarr*)**: Higher rates of children born to unmarried parents are linked to more burglaries.
- **Foreign-Born Population (*numForeignBorn*)**: Increased immigration correlates with higher crime rates.
- **Police Funding (*persEmergShelt*)**: More funding correlates with fewer burglaries, highlighting the role of law enforcement.
- **Vacant Housing (*houseVacant*)**: Higher vacancy rates are associated with more burglaries, suggesting vacant homes as targets.
- **Homelessness (*persHomeless*)**: More homelessness is linked to increased burglaries, pointing to social instability.

10.1 Policy Implications

Effective policies include:

- **Economic Support**: Addressing poverty and family structure.
- **Law Enforcement**: Increasing police funding.
- **Housing Interventions**: Reducing vacant properties and supporting the homeless.

10.1.1 Conclusion

Focusing on these areas can help reduce burglaries and improve public safety.