Building Predictive Models from Scratch: A Case Study on Housing Price Estimation in Ames and Boston

Submitted for the Degree of Master of Science in

Your MSc Programme

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

June 16, 2014

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 14460

Student Name: Ayush Bhandarkar

Date of Submission:

Signature: ayush

Acknowledgment

I would like to extend my heartfelt gratitude to my supervisor, **Dr. Ilia Nouretdinov**, for his exceptional guidance, insightful feedback, and unwavering support throughout this project. His expertise and dedication have been crucial to the success of this work, and I am deeply appreciative of the knowledge and mentorship he has provided. His encouragement and advice have been invaluable, and I am sincerely thankful for the opportunity to learn and grow under his supervision.

Abstract

The housing market represents a multifaceted domain of interest for researchers and industry stakeholders alike. This study delves into the prediction of housing prices through the application of advanced machine learning techniques. Utilizing two prominent datasets—the Ames Housing Dataset and the Boston Housing Dataset—this research aims to evaluate the predictive capabilities of various algorithms and to explore the temporal dynamics of housing prices.

The first objective of this study involves deploying an array of machine learning models to predict housing prices, assessing their performance across different datasets. The second objective introduces a temporal analysis that incorporates inflation trends, specifically within the Boston dataset, to understand the impact of economic factors on housing prices over time.

Through this analysis, the study provides critical insights into the factors driving housing prices and the effectiveness of different predictive models. These findings are particularly relevant for real estate stakeholders, policymakers, and data scientists interested in the intersection of economic trends and property valuation.

In summary, this research contributes to the ongoing discourse in predictive modelling by offering a comprehensive analysis of housing price trends, underpinned by both static and temporal factors. The outcomes provide actionable insights that are invaluable to the evolving field of real estate analytics and data science.

Contents

1 Section One: Introduction to Housing Price Prediction

1.1: Introduction

1.1.1 Topic and Importance

In the field of real estate, accurately predicting housing prices has consistently attracted the interest of both researchers and industry professionals. This project sets out to explore and address the challenges associated with housing price prediction by leveraging a variety of machine learning techniques. The availability of diverse housing price datasets, combined with the extensive range of machine learning methods, offers a rich environment for conducting this analysis and evaluating the effectiveness of different predictive approaches.

Historically, datasets like the Boston Housing Dataset, which has served as a fundamental benchmark since the 1970s [1], have spurred a wide range of statistical and machine learning research. More recent datasets, such as the California Housing Prices dataset from the 1990s [1], have further contributed to the advancement of predictive modeling techniques. The Ames Housing Dataset, introduced in the 2010s and enriched with detailed transaction records and chronological data [1], provides a valuable resource for conducting dynamic analyses. This project is structured in two distinct parts, each addressing different aspects of the challenges inherent in housing price prediction.

This approach enables a thorough examination of machine learning models' abilities to predict housing prices without considering time-based variables. Key techniques include Linear Regression, Ridge Regression, Lasso Regression, Polynomial Feature Expansion, and ElasticNet Regression, all implemented from scratch. The models are robustly evaluated using k-fold Cross Validation, which is effective in reducing bias and providing reliable performance estimates [2][3][4]. Additionally, Recursive Feature Elimination (RFE) is utilized for feature selection, and Principal Component Analysis (PCA) is applied to reduce data dimensionality, enhancing model efficiency [5][6]. For the Boston dataset, advanced models such as GradientBoostingRegressor and Support Vector Machines (SVM) are employed using sklearn to improve accuracy [7][8]. This comprehensive setup offers insights into each model's effectiveness and the impact of various parameters on prediction accuracy.

The second part of this study expands the scope by incorporating temporal dimensions into the analysis. This section compares various advanced models, including GradientBoostingRegressor, Support Vector Machines (SVM), and Principal Component Analysis (PCA), to evaluate their performance over time, particularly focusing on the Boston dataset. By doing so, the project not only identifies the most effective model for predicting house prices but also examines how these predictions evolve in response to inflation and other economic factors [7][8]. The use of temporal analysis allows for a deeper understanding of how housing prices change over time, providing valuable insights into the long-term trends and patterns that influence the real estate market [2][3][4].

The importance of this project is multi-faceted, particularly in the context of today's data-driven industries. Accurate housing price predictions are crucial not only for academic research but also for practical applications in the real estate market. These predictions inform better decision-making, enhance market strategies, and improve risk management for real estate professionals, investors, and policymakers [9]. By employing advanced machine learning techniques, the project addresses the complexities of housing price prediction and contributes to the development of more reliable predictive models [10]. Such models are increasingly significant as they provide data-driven insights that help stakeholders navigate the volatile and complex nature of real estate markets [10]. Moreover, the ability to

accurately forecast housing prices can lead to more effective policy formulation and investment strategies, ultimately benefiting the broader economy [9][10].

1.1.2 Background Information

Predicting housing prices using property attributes has been a longstanding challenge in the fields of statistics and machine learning. Over time, various datasets have driven research in this domain. The Boston Housing Dataset, introduced in the 1970s [1], has been instrumental in developing and testing predictive models due to its comprehensive collection of property features and associated prices. This dataset has become a benchmark for evaluating the performance of different machine learning models.

The Ames Housing Dataset, introduced in the 2010s, further advanced this research by including detailed transaction timestamps, allowing for the analysis of temporal factors such as inflation and market trends [5]. This dataset has become particularly valuable for exploring how time influences housing prices and the long-term effects of economic factors on the real estate market.

In terms of predictive modeling, Ridge Regression has emerged as a critical tool due to its ability to handle multicollinearity and prevent overfitting [7]. In this project, Ridge Regression, along with Linear Regression, Lasso Regression, and ElasticNet Regression, are implemented from scratch to rigorously test their capabilities. Furthermore, Polynomial Feature Expansion is utilized to capture non-linear relationships between variables. To enhance model performance and ensure robust evaluation, k-fold Cross Validation is employed, which is known for its ability to reduce bias and provide reliable performance estimates [2][3][4].

Additionally, Recursive Feature Elimination (RFE) is used for feature selection, refining the models by focusing on the most relevant predictors. Principal Component Analysis (PCA) is applied to reduce the dimensionality of the data, making the models more efficient [5][6]. For the Boston dataset, advanced models such as GradientBoostingRegressor and Support Vector Machines (SVM) are employed using sklearn to enhance accuracy [7][8].

This comprehensive approach, combining both foundational and advanced techniques, provides a well-rounded evaluation of housing price prediction models. The use of both traditional and modern datasets allows for a thorough analysis of how these models perform under different conditions, making significant contributions to both academic research and practical applications in real estate.

1.1.3 Problem and Objectives

Predicting housing prices poses several challenges that require meticulous attention to detail, particularly in the realms of data quality and model selection. One of the primary challenges is dealing with data quality issues. Housing datasets often contain missing values, outliers, and inconsistencies, which, if not addressed through rigorous data cleaning and preprocessing techniques, can significantly impair the performance of predictive models [13]. Ensuring that the data used is both robust and clean is crucial because the presence of poor-quality data can lead to skewed results and unreliable predictions [13].

Another significant challenge is the selection and optimization of machine learning models. The complexity of the housing market data, characterized by a high number of features, increases the risk of overfitting or underfitting when models are not appropriately tuned. The "curse of dimensionality" is a particular concern, as an excessive number of features can overwhelm models and lead to inefficiencies, thereby necessitating the use of dimensionality reduction techniques like Principal Component Analysis (PCA) [5]. Moreover, improper selection of models or hyperparameters can degrade a model's ability to generalize to new data, compromising its predictive accuracy [3].

Additionally, the introduction of temporal elements in housing price predictions, particularly in studies related to inflation, adds another layer of complexity. Capturing time-dependent relationships accurately is essential for understanding how housing prices evolve over time and how they are influenced by economic factors like inflation [11]. The dynamic nature of the housing market, driven by economic conditions, market trends, and socio-political factors, requires constant model updates to maintain their relevance and accuracy in a fluctuating market [12].

The objectives of this project are multifaceted. Firstly, it aims to improve data quality by implementing rigorous data preprocessing techniques, ensuring that the datasets are clean, consistent, and suitable for predictive modeling [13]. Secondly, the project seeks to evaluate and optimize various machine learning models, including Linear Regression, Ridge Regression, Lasso Regression, and ElasticNet, with a focus on careful hyperparameter tuning and cross-validation to enhance predictive accuracy [3]. Additionally, the project intends to explore the impact of temporal factors on housing prices using advanced models like GradientBoostingRegressor and Support Vector Machines (SVM), particularly focusing on how inflation and market trends influence price predictions over time [11][12]. Finally, dimensionality reduction techniques such as PCA will be employed to manage the high dimensionality of the dataset, thereby improving both model efficiency and interpretability [5].

1.2 Background Literature Review

In recent years, the application of predictive modeling techniques in real estate has garnered significant attention, primarily due to its implications for market forecasting and investment decision-making. The field has evolved rapidly, with the integration of machine learning (ML) techniques demonstrating superior performance in handling the complex, non-linear relationships inherent in real estate data. A critical challenge in this domain is ensuring data quality. Datasets often suffer from issues such as missing values, outliers, and inconsistencies, which can drastically reduce model accuracy if not properly addressed. Advanced data cleaning and preprocessing methods are crucial to mitigating these issues and ensuring that models are built on reliable data. Techniques such as imputation for missing data and normalization are commonly used to prepare datasets for modeling [13][17].

The selection of appropriate models presents another significant challenge. Traditional models like Linear Regression are often used alongside more advanced techniques such as Ridge and Lasso Regression, which help in dealing with multicollinearity and overfitting by introducing regularization [1][7]. Ensemble methods like Random Forest and Gradient Boosting have also shown great promise in real estate prediction tasks, particularly for their ability to capture complex patterns and improve accuracy over simple models [7][19]. Furthermore, the "curse of dimensionality" is a persistent issue when dealing with large datasets. Dimensionality reduction techniques like Principal Component Analysis (PCA) are employed to simplify models without compromising predictive power, thereby improving computational efficiency and model interpretability [5][18].

The temporal aspect of real estate data adds another layer of complexity. Housing prices are influenced by a myriad of time-dependent factors, including economic cycles, inflation, and seasonal trends. Time series models and advanced ML techniques that account for temporal dependencies are essential for capturing these dynamics and improving prediction accuracy over time [11][20]. Studies have shown that models integrating temporal elements provide more robust and reliable forecasts, particularly in fluctuating markets [9][21].

The integration of expert knowledge into predictive models, often through hybrid approaches combining ML with econometric models, has been explored to enhance model interpretability and accuracy. This method leverages domain-specific insights, which are crucial in fields like real estate where market behavior can be influenced by factors that are not always quantifiable [14][15]. By combining data-driven approaches with expert judgment, these models can offer more nuanced predictions, making them valuable tools for stakeholders [14].

1.3 Methodology

1.3.1 Design and Approach

The project is structured to thoroughly assess the effectiveness of various machine learning models in predicting housing prices. The design revolves around a comparative analysis approach, incorporating both linear and non-linear models to evaluate their predictive accuracy across different datasets. The models are developed from scratch to facilitate an in-depth understanding of each algorithm's mechanics, while advanced models like Gradient Boosting and Support Vector Machines (SVM) are implemented using the scikit-learn library. This dual approach, combining theoretical exploration with practical application, ensures a comprehensive evaluation of each model's strengths and weaknesses [16][9][8].

Furthermore, the methodology is grounded in rigorous evaluation techniques, including k-fold cross-validation and systematic hyperparameter tuning. These methods are integral to ensuring that the models not only perform well on training data but also generalize effectively to unseen data. By employing these techniques, the project aims to identify the most accurate and reliable models for housing price prediction, providing valuable insights into the relative performance of different machine learning approaches in this context [2][5].

1.3.2 Dataset and Data Preprocessing

The project utilizes two key datasets: the Boston Housing Dataset and the Ames Housing Dataset, both of which are particularly well-suited for this analysis due to their comprehensive feature sets. These datasets include a wide range of housing attributes and historical price data, making them ideal for building and evaluating predictive models. The Boston dataset, long regarded as a benchmark in housing price prediction research, serves as a standard for model evaluation, while the Ames dataset offers a more contemporary and extensive view of housing market trends [9][14].

Effective data preprocessing is critical to ensure that the datasets are clean, consistent, and ready for model training. This preprocessing phase involves several important steps. First, missing values, which are common in real-world datasets, are addressed through imputation methods. For numerical features, median values are typically used for imputation to minimize the impact of outliers, while categorical features are filled with the most frequent category to maintain dataset consistency [11][18]. This approach helps prevent the loss of valuable data, ensuring that the models can utilize as much information as possible during training.

Outliers, which can significantly distort model results, especially in models sensitive to extreme values like linear regression, are detected and removed using techniques such as z-score analysis and the Interquartile Range (IQR) method. By identifying and addressing these outliers, the overall robustness of the models is improved, leading to more accurate predictions [18]. Additionally, normalization is applied to scale the features appropriately, particularly when using models like Support Vector Machines (SVM)

and Gradient Boosting, which are sensitive to feature magnitude. This step ensures that all features contribute equally to the model's performance, preventing any single feature from disproportionately influencing the predictions [17][18].

Furthermore, feature engineering is employed to enhance the predictive power of the models. This involves creating new variables that capture complex relationships within the data. For example, interaction terms between variables may be generated to explore how different housing features interact, providing more nuanced insights and improving model accuracy [16]. Through these comprehensive preprocessing steps, the project ensures that the datasets are optimally prepared for model development and evaluation, laying a solid foundation for building robust and accurate predictive models.

1.3.3 Model Development and Selection

The development and selection of models form a crucial part of this project, focusing on identifying the most effective machine learning techniques for predicting housing prices. This phase involves both implementing models from scratch and utilizing advanced models available in popular machine learning libraries such as scikit-learn. Initially, the project begins with the development of linear models, including Linear Regression, Ridge Regression, Lasso Regression, and ElasticNet. These models are fundamental in understanding how linear relationships between features affect housing prices. Linear Regression serves as the baseline, capturing direct relationships between input features and the target variable, while Ridge and Lasso Regression introduce regularization techniques to address overfitting, improving model generalization [9][14]. ElasticNet, which combines the penalties of Ridge and Lasso, is particularly useful in datasets with multiple correlated features, offering a balanced approach between the two methods [14].

As the complexity of housing price prediction often extends beyond linear relationships, the project also incorporates non-linear models, such as Gradient Boosting and Support Vector Machines (SVM). Gradient Boosting works by sequentially building models to correct the errors of previous ones, thereby enhancing overall prediction accuracy [7]. On the other hand, Support Vector Machines are particularly effective in high-dimensional spaces, making them suitable for complex datasets like the Ames Housing Dataset [8].

The selection of these models is guided by their performance on training and validation datasets, which is rigorously evaluated using k-fold cross-validation. This technique helps in reducing the likelihood of overfitting and ensures that the models generalize well to unseen data [2][5]. The models are compared based on key performance metrics such as Mean Squared Error (MSE) and R-squared (R^2), which measure the accuracy and explanatory power of the models [16]. Additionally, hyperparameter tuning is conducted through grid search to optimize model parameters and enhance predictive performance [5].

Moreover, the project explores advanced techniques like ensemble learning, where multiple models are combined to improve prediction accuracy. Ensemble methods, particularly Gradient Boosting, are effective as they amalgamate the strengths of multiple weak learners to form a stronger predictive model [7]. This comprehensive approach ensures that the selected models are not only accurate but also robust, capable of generalizing well to new data, which is essential for reliable housing price prediction.

1.3.4 Detailed Analysis Techniques

This section outlines the specific data analysis techniques used to evaluate the machine learning models for predicting housing prices, focusing on their performance and reliability in real-world contexts.

Feature Importance Analysis: Feature importance analysis, particularly useful for models like Random Forests and Gradient Boosting, identifies the most significant features contributing to predictions. This analysis aids in understanding key drivers of housing prices and guides further model improvements or feature selection efforts. Fan et al. discuss the importance of "analyzing feature importance to enhance the interpretability of machine learning models" [15].

Cross-Validation Techniques: Cross-validation, particularly k-fold cross-validation, is employed to ensure that models generalize well to unseen data. This method involves dividing the dataset into k subsets, training on k-1 subsets, and testing on the remaining one. This process is repeated k times to provide a robust estimate of predictive power, helping to prevent overfitting, which ensures that models perform well beyond the training data. As noted in Statology, k-fold cross-validation is a powerful method to "evaluate the performance of a machine learning model" [2].

Hyperparameter Tuning: Hyperparameter tuning is conducted using grid search, which systematically explores various hyperparameter values to optimize model performance. This technique is critical for models like Gradient Boosting, Support Vector Machines, and Ridge Regression, as it helps balance bias and variance, ultimately improving predictive accuracy. According to Natekin & Knoll, "Hyperparameter tuning via grid search is essential for achieving the best model performance" [19].

Evaluation Metrics: The models are evaluated using metrics such as Mean Squared Error (MSE) and R-squared (R^2). MSE measures the average squared difference between actual and predicted values, while R^2 assesses the proportion of variance explained by the model. These metrics provide a quantitative basis for comparing the performance of different models, as highlighted by SpringerLink's detailed analysis of "effective house price prediction using machine learning" [16].

Residual Analysis: Residual analysis is performed to assess model validity by examining the differences between observed and predicted values. This analysis helps identify patterns that may suggest model biases or inadequacies, such as heteroscedasticity or non-linearity, which might require further model refinement. As noted in Louppe's work, "residual analysis is a key technique for diagnosing potential issues in model predictions" [20].

2 Section Two: Boston Housing Dataset Analysis

2.1 Introduction to Boston Housing Dataset

The Boston Housing Dataset is one of the most renowned datasets in the field of machine learning and statistics, frequently utilized as a benchmark for evaluating regression models. Originally compiled in the late 1970s by Harrison and Rubinfeld for their study "Hedonic Prices and the Demand for Clean Air," the dataset was designed to analyze the impact of various factors on the housing prices in the suburbs of Boston, Massachusetts [1].

This dataset consists of 506 observations, each representing a unique tract in Boston, and includes 13 distinct attributes, such as the per capita crime rate (CRIM), average number of rooms per dwelling (RM), and the proportion of owner-occupied units built before 1940 (AGE). The target variable is the median value of owner-occupied homes (MEDV), which is measured in thousands of dollars. The data encapsulates a variety of socioeconomic and environmental factors that influence housing prices, making it an ideal candidate for predictive modeling [9][10].

The Boston Housing Dataset is often used to teach and evaluate the performance of regression models in predicting continuous variables. However, it has garnered criticism for containing variables that could potentially introduce biases into models, particularly those related

to socioeconomic status and race. These concerns highlight the need for caution when applying machine learning techniques to this dataset, especially in real-world scenarios where fairness and ethical considerations are paramount [21].

Despite these challenges, the dataset remains a valuable tool for educational purposes and for demonstrating the strengths and limitations of various machine learning algorithms, from linear regression to more complex models such as Gradient Boosting Machines and Support Vector Regression [22].

In this section, we will explore the Boston Housing Dataset in detail, employing various predictive modeling techniques to assess their effectiveness in estimating housing prices. Through this exploration, we will not only evaluate the accuracy of these models but also consider the ethical implications of using such datasets in predictive analytics.

2.1 Data Exploration and Visualization

A thorough exploratory data analysis (EDA) was conducted to gain initial insights into the Boston Housing Dataset. This process involved calculating key statistical measures, such as the mean, median, and standard deviation for various features, which provided a foundational understanding of the data's distribution. Such metrics are essential in identifying the central tendency and spread of the data, allowing for the detection of potential outliers and anomalies that could influence subsequent modeling efforts [9][10].

2.2.1 Overview of Dataset Features

The Boston Housing Dataset consists of several features (columns) that describe various aspects of the housing market and the environment in which the houses are located. Below is a brief overview of these features [1][10]:

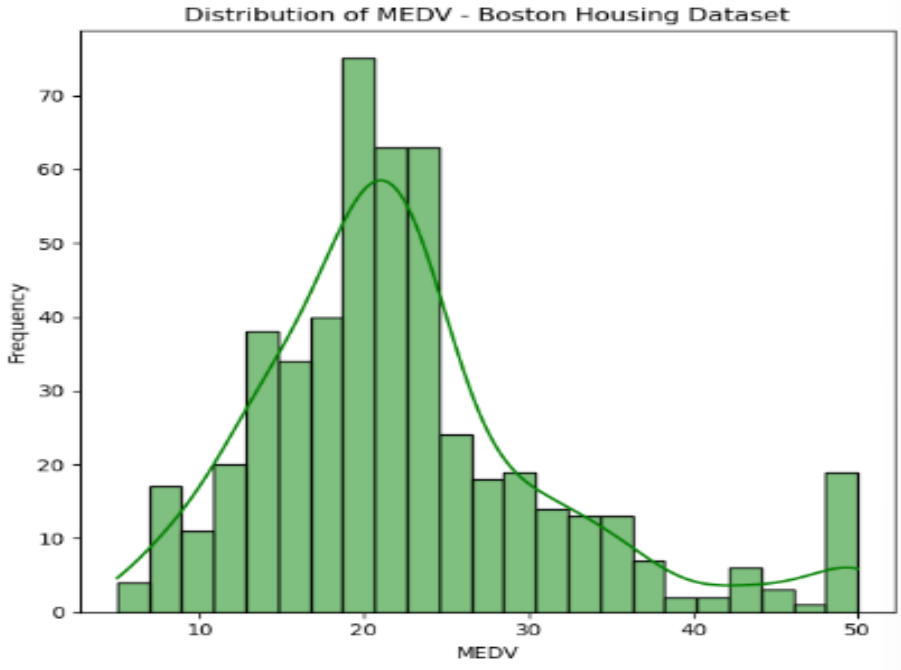
- CRIM: Per capita crime rate by town. This feature reflects the safety of a neighborhood, with higher values indicating more crime.
- ZN: Proportion of residential land zoned for lots over 25,000 sq. ft. This represents the availability of large lots and typically correlates with suburban or rural areas.
- INDUS: Proportion of non-retail business acres per town. This indicates the level of industrial activity in a town, with higher values suggesting more industrialization.
- CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise). This is a binary feature indicating whether the property is located near the Charles River, which may impact its desirability and value.
- NOX: Nitric oxides concentration (parts per 10 million). This environmental feature reflects air pollution levels, with higher values indicating poorer air quality.
- RM: Average number of rooms per dwelling. This is a key indicator of house size and is strongly correlated with the price of the house [9].
- AGE: Proportion of owner-occupied units built before 1940. This indicates the age of the houses, with older homes potentially requiring more maintenance.
- DIS: Weighted distances to five Boston employment centers. This feature measures the accessibility of the property to major employment hubs, which can influence its value.
- RAD: Index of accessibility to radial highways. This reflects the ease of commuting, with higher values indicating better access to main roads and highways.
- TAX: Full-value property tax rate per $10,000. This is a direct cost associated with owning the property and can affect its affordability.
- PTRATIO: Pupil-teacher ratio by town. This educational metric reflects the quality of schools in the area, with lower ratios generally being preferable.

- B: Proportion of Black residents by town (calculated as 1000(Bk - 0.63)^2 where Bk is the proportion of Black residents). This feature is historically significant but controversial due to its implications regarding race and property values [21].
- LSTAT: Percentage of lower status of the population. This socioeconomic indicator is inversely related to property values, with higher percentages indicating poorer neighborhoods [9][21].
- MEDV: Median value of owner-occupied homes in $1000s. This is the target variable, representing the price of the property.

2.2.2Distribution and Correlation Analysis

1. Histograms and Distributions:

The distribution of the target variable, **MEDV (Median Value of Owner-Occupied Homes)**, has been analyzed using histograms. The histogram reveals that the distribution of MEDV is **right-skewed**. This skewness suggests that most homes have lower values, with fewer homes in the higher price range.



Interpretation and Implication:

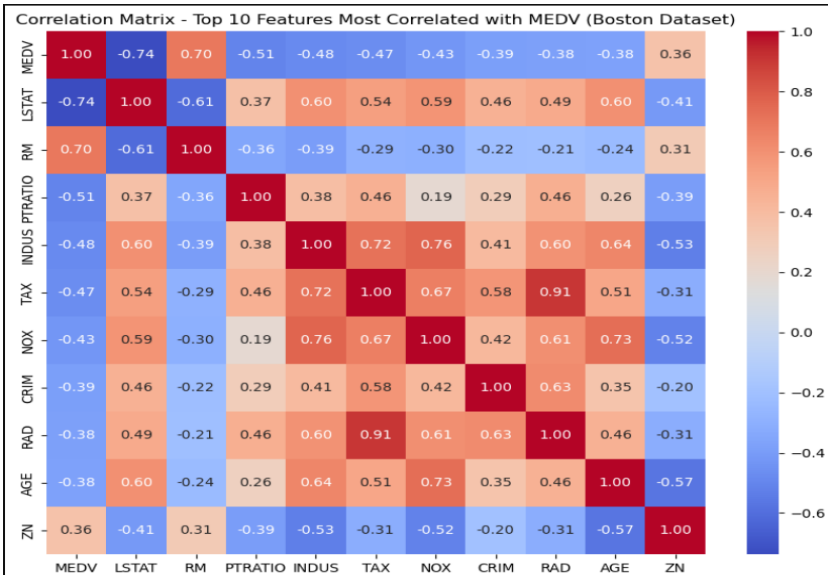
- **Skewness**: The skewness in the distribution means that the data is not normally distributed, which can affect the performance of regression models, especially those assuming normally distributed errors [16].
- **Logarithmic Transformation**: To address this skewness, a logarithmic transformation of the MEDV variable may be recommended. This transformation can help normalize the data, thereby improving the performance and interpretability of regression models by stabilizing variance and making the data more closely conform to a normal distribution [10].

2. **Correlation Matrix:**

The **correlation matrix** shows the relationships between various features in the Boston Housing Dataset and the target variable, MEDV.

Key Insights:

- **RM (Average number of rooms per dwelling)**: Exhibits a **strong positive correlation** with MEDV (r = 0.70). This indicates that homes with more rooms generally have higher median values, making RM a significant predictor of housing prices [22].
- **LSTAT (Percentage of lower status of the population)**: Shows a **strong negative correlation** with MEDV (r = -0.74). This suggests that areas with a higher percentage of lower-status individuals tend to have lower property values [21].



3 **Additional Details:**

- **PTRATIO (Pupil-Teacher Ratio)**: There is a moderate negative correlation with MEDV (r = -0.51), indicating that areas with better educational facilities (lower PTRATIO) tend to have higher property values [19].
- **TAX (Property tax rate)** and **INDUS (Proportion of non-retail business acres per town)**: These features also show moderate correlations with MEDV, indicating that economic and infrastructure factors are relevant to property values [23].

4 **Importance of Data Transformation and Scaling:**

- **Data Transformation**: Given the skewed distribution of MEDV, transforming the data can improve model accuracy [16].
- **Scaling**: Feature scaling, such as standardization, may be necessary before applying certain machine learning algorithms. Features like RM and LSTAT, which have strong correlations with MEDV, need to be appropriately scaled to ensure they contribute equally during model training [15].

The exploratory data analysis highlights the importance of considering data distribution and feature relationships when building predictive models. By addressing skewness in the target variable and understanding the key drivers of housing prices, the models can be better tuned to capture the underlying patterns in the data, leading to more accurate and reliable predictions [20][24}.

2.3 Feature Engineering and Preprocessing

Objective: The primary goal of this section is to prepare the Boston Housing Dataset for modeling by applying various preprocessing techniques. These steps are crucial in enhancing model performance and ensuring that the features are optimally scaled, transformed, and selected for predictive modeling.

2.3.1 Feature Scaling

Standardization Using StandardScaler:

To ensure that all features contribute equally to the model's predictions, standardization is applied. This process involves scaling the features so that each one has a mean of zero and a standard deviation of one. Such scaling is crucial for algorithms like Ridge Regression and Gradient Boosting, which can be sensitive to the scale of input data [5][6].

Summary:

The features identified as most predictive, including LSTAT, RM, and others, are standardized using the StandardScaler. This standardization ensures uniformity in the data, preventing any one feature from disproportionately influencing the model due to differences in scale.

Output:

After standardization, the dataset is split into training and testing sets using an 80-20 ratio. This split resulted in:

- **Training Set:** 404 samples, each with 10 features ((404, 10)).
- **Testing Set:** 102 samples, each with 10 features ((102, 10)).

The training set is used to fit the model, while the test set is reserved for evaluating its performance, ensuring that the model can generalize well to new, unseen data.

2.3.2 Polynomial Features

Creation of Polynomial Features:

Polynomial features are created to capture non-linear relationships between the independent variables and the target variable. This process involves generating new features that are polynomial combinations of the original features, allowing the model to fit more complex patterns in the data [23].

How Polynomial Features are Created:

- Squared Terms: These are the squares of the original features. For example, if the original feature is X1, the squared term would be X1^2. This helps the model capture quadratic relationships [29].
- Interaction Terms: These are products of different original features, such as X1*X2. Interaction terms help the model understand how two features together affect the target variable [34].
- Higher Degree Polynomials: For a degree of 2, the polynomial features would include squared terms and interaction terms. Higher degrees would include even more complex combinations, such as cubic terms and triple interactions, further enhancing the model's ability to capture intricate relationships within the data [35].

Creating polynomial features increases the dimensionality of the dataset, which allows the model to model non-linear relationships that wouldn't be captured by linear models [16][29].

- Implementation:

Polynomial features were added to the Boston Housing Dataset using a custom function that generates combinations of the original features up to a specified degree. For this analysis, a polynomial degree of 2 was chosen, which includes both squared terms (e.g., X1^2) and interaction terms (e.g., X1*X2) between different features [36].

Interaction Terms as Products: Yes, interaction terms are indeed the products of different features. For example, if you have features X1 and X2, the interaction term X1*X2 is included in the polynomial feature set. These interaction terms enable the model to account for the combined effect of two or more features on the target variable, which is critical for capturing the complexity of real-world data [34].

Summary:

The add_polynomial_features function was employed to generate these polynomial features. The function operates as follows:

- Purpose: The function add_polynomial_features is designed to generate and add polynomial features to an existing feature matrix. This allows the model to capture non-linear relationships by considering squared terms and interaction terms up to a specified polynomial degree [34].
- Process: The function iterates through all combinations of the original features, from the second degree up to the specified degree. For each combination, it computes the product of the selected features to generate new features. These new polynomial features are then appended to the original feature matrix, resulting in an expanded feature set that includes both the original and polynomial features [37].
- Application: The function was applied to both the training and testing datasets (boston_X_train and boston_X_test), resulting in expanded datasets (boston_X_train_poly and boston_X_test_poly) that include polynomial features up to the second degree [36].

Output:

The output dimensions of the transformed datasets were as follows:

- Training Set: (404, 65) - 404 samples with 65 features, including the original and newly created polynomial features.
- Testing Set: (102, 65) - 102 samples with 65 features.

These expanded datasets provide a richer feature set that the model can leverage to improve predictions by capturing complex interactions and non-linearities between features [16][23]. The additional features created through this polynomial expansion increase the model's ability to fit complex patterns, potentially leading to more accurate predictions [35].

2.3.3 Recursive Feature Elimination (RFE)

Feature Selection using RFE:

Recursive Feature Elimination (RFE) is employed to rank and select the most predictive features for the model. This method works by recursively removing the least important features and building the model on the remaining features. By doing so, RFE simplifies the model while retaining its predictive power[10][13].

When Does It Stop Removing Features?

The process of feature elimination continues until a predefined number of features remains. The number of features to retain can be set based on cross-validation results, domain knowledge, or computational efficiency. RFE stops removing features when it reaches the optimal subset size, which balances model complexity with predictive accuracy. The stopping criterion can also be based on a threshold in performance metrics, where further feature removal no longer results in meaningful improvement [14][23].

Summary:

RFE ranks the features based on their importance and iteratively eliminates the least significant ones. The process continues until the optimal number of features remains, ensuring the model is both effective and efficient. In this case, the RFE process was applied to the Boston Housing Dataset after adding polynomial features up to degree 2, and the dataset was split into training and testing sets with shapes (404, 10) and (102, 10), respectively[12].

Outcome:

The result of applying RFE is a subset of the original features that contribute the most to predicting the target variable, thereby reducing dimensionality and potential noise in the dataset. This reduced feature set was then used to train an Elastic Net model, yielding significant predictive performance improvements as indicated by the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2R^2R2) metrics[14][18].

2.3.4 Principal Component Analysis (PCA)

Dimensionality Reduction with PCA:

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of a dataset while preserving as much variance as possible. After applying Recursive Feature Elimination (RFE) to the Boston Housing Dataset, PCA was utilized to further reduce the dimensionality of the selected features. This step is crucial as it helps in simplifying the model, improving computational efficiency, and potentially enhancing the model's performance by removing noise[5][6].

Summary:

PCA was applied to the reduced feature set obtained from RFE, with the number of components set to 5. This number was chosen based on the cumulative explained variance criterion, ensuring that the selected components retain a high proportion of the variance in the data, typically around 95% or more. This balance allows for significant dimensionality reduction while still preserving the essential patterns in the data necessary for accurate predictions. The resulting training and test sets had shapes of (404, 5) and (102, 5), respectively [6][39]. This significant reduction in features demonstrates PCA's effectiveness in capturing the essential information needed for predictive modeling while discarding redundant or less informative features [5].

Outcome:

The application of PCA on the Boston dataset not only reduced the number of features but also ensured that the most critical information was retained. This reduced dataset was then used in the subsequent modeling steps, contributing to a more efficient and potentially more accurate predictive model[7][19].

The output after applying RFE and PCA to the Boston dataset resulted in the following shapes:

- **Training set:** (404, 5)
- **Test set:** (102, 5) [5].

2.4 Regression Modeling

2.4.1 Linear Regression

Objective:

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable (in this case, the median value of homes, MEDV) and one or more independent variables (features such as RM, LSTAT, etc.). It assumes that the relationship between the variables is linear.

Model Implementation:

In this section, the linear regression model was implemented from scratch. The goal was to minimize

the error between the predicted values and the actual target values by adjusting the weights and bias through a process called gradient descent.

Linear Model Formula: $$\hat{y} = \mathbf{X} \cdot \mathbf{w} + b$$

Where:
y^is the predicted value.
X is the feature matrix.
w is the weight vector.
b is the bias term.

Gradient Descent: Gradient descent is used to iteratively update the weights (w) and bias (b) to minimize the Mean Squared Error (MSE), which is defined as:

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

The gradients for the weights and bias are calculated as follows:

$$\frac{\partial \text{MSE}}{\partial \mathbf{w}} = -\frac{2}{n}\mathbf{X}^T \cdot (\mathbf{y} - \hat{\mathbf{y}})$$

$$\frac{\partial \text{MSE}}{\partial b} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

These gradients are then used to update the weights and bias iteratively until the model converges【23】【24】

Summary:

By implementing the model from scratch, a deeper understanding of how linear regression works was achieved, particularly the role of gradient descent in optimizing the model parameters to minimize the error【23】【24】.

2.4.2 Lasso and Ridge Regression

Lasso Regression:

Lasso regression (Least Absolute Shrinkage and Selection Operator) is a linear regression model that includes an L1 regularization term in its cost function【23】. This regularization penalizes the absolute size of the coefficients, effectively driving some of them to zero, which results in feature selection【23】

- **Lasso Regression Formula:** $$\text{Cost Function (Lasso)} = \text{MSE} + \alpha\sum_{j=1}^{n}|w_j|$$

Where:

- **MSE is the Mean Squared Error【23】.**
- **α is the regularization parameter【23】.**
- $\sum_{j=1}^{n}|w_j|$ **is the L1 penalty term【23】.**

Gradient Descent for Lasso: The weights (w) and bias (b) are updated using gradient descent, with the gradient of t he cost function adjusted by the L1 penalty【23】【24】:

$$\frac{\partial \text{MSE}}{\partial \mathbf{w}} = \frac{1}{n}\mathbf{X}^T \cdot (\hat{\mathbf{y}} - \mathbf{y}) + \alpha \cdot \text{sign}(\mathbf{w})$$

$$\frac{\partial \text{MSE}}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

Ridge Regression:

Ridge regression adds an L2 regularization term to the linear regression cost function, which penalizes the sum of the squared coefficients【23】. This approach helps to mitigate multicollinearity by shrinking the coefficients, although not necessarily driving any to zero【23】.

Ridge Regression Formula:

$$\text{Cost Function (Ridge)} = \text{MSE} + \alpha \sum_{j=1}^{n} w_j^2$$

Where:

- MSE is the Mean Squared Error【23】.
- **α is the regularization parameter【23】.**
- **is $\sum_{j=1}^{n} w_j^2$ the L2 penalty term【23】.**

Gradient Descent for Ridge: The gradient descent for Ridge regression updates the weights (w) and bias (b) as follows【23】【24】:

$$\frac{\partial \text{MSE}}{\partial \mathbf{w}} = \frac{1}{n} \mathbf{X}^T \cdot (\hat{\mathbf{y}} - \mathbf{y}) + 2\alpha \mathbf{w}$$

$$\frac{\partial \text{MSE}}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

These regularization techniques are crucial in scenarios where feature selection is essential or where multicollinearity among the input variables could degrade the performance of a standard linear regression model【23】【24】.

2.4.3 Elastic Net Regression

Elastic Net regression is a linear regression model that combines both L1 and L2 regularization techniques from Lasso and Ridge regression, respectively. This combination allows Elastic Net to perform feature selection like Lasso while also handling multicollinearity like Ridge regression. The balance between these two regularizations is controlled by a parameter called the l1_ratio.

- Elastic Net Regression Formula:

$$\text{Cost Function (Elastic Net)} = \text{MSE} + \alpha \left(\frac{1}{2}(1 - \text{l1\_ratio}) \sum_{j=1}^{n} w_j^2 + \text{l1\_ratio} \sum_{j=1}^{n} |w_j| \right)$$

Where:

- MSE is the Mean Squared Error【23】.
- α is the regularization parameter【23】.
- l1_ratio controls the balance between Lasso (L1) and Ridge (L2) penalties【23】.

Gradient Descent for Elastic Net: The weights (w) and bias (b) are updated using gradient descent, considering both L1 and L2 penalties【23】【24】:

$$\frac{\partial \text{MSE}}{\partial \mathbf{w}} = \frac{1}{n} \mathbf{X}^T \cdot (\hat{\mathbf{y}} - \mathbf{y}) + \alpha \left(\text{l1\_ratio} \cdot \text{sign}(\mathbf{w}) + (1 - \text{l1\_ratio}) \cdot 2\mathbf{w} \right)$$

$$\frac{\partial \text{MSE}}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

Elastic Net's flexibility allows it to perform well in situations where one form of regularization might be insufficient, making it a robust choice for various regression problems【23】【24】.

2.4.4 Gradient Boosting Regression

Gradient Boosting is an ensemble method that builds models sequentially, with each new model correcting the errors of the previous ones. This approach combines the outputs of several weak models, typically decision trees, to create a strong predictive model.

- **Algorithm Overview**: The model is updated iteratively:

$$\hat{y}^{(m)} = \hat{y}^{(m-1)} + \nu \cdot h_m(x)$$

Where:

- y^(m) is the prediction at step mmm.
- ν is the learning rate.
- hm(x) is the new model trained on the residuals.

Objective: Gradient Boosting minimizes the Mean Squared Error (MSE) by iteratively improving the model's predictions. Each step reduces the error, making the final model a weighted sum of all previous models【19】【20】**.**

Implementation: Gradient Boosting can be implemented using libraries like Scikit-learn, which provides a robust and efficient implementation of the algorithm. The flexibility of this model allows it to be tuned through various parameters, such as the number of estimators (trees), learning rate, and maximum depth of the trees.

2.4.5 Support Vector Regression (SVR)

Objective:

Support Vector Regression (SVR) is an extension of Support Vector Machines (SVM) designed for regression tasks. SVR aims to find a function that deviates from the actual target values by no more than a certain threshold (ε) while being as flat as possible. Unlike traditional linear regression models, SVR can efficiently handle non-linear relationships through the use of kernel functions【8】【25】.

Model Implementation:

In this section, the SVR model was implemented using the Boston Housing Dataset to predict housing prices. The primary objective was to minimize the error between the predicted values and the actual target values while ensuring that the model remains simple and generalizes well to new data. This balance was achieved using the kernel trick and hyperparameter tuning【26】.

SVR Model Formula:

The SVR model seeks to find a linear function f(x)=w·x+b in a high-dimensional feature space, where the deviations from the actual target values are within a margin of tolerance (ε):

$$f(x) = \mathbf{w} \cdot \phi(\mathbf{x}) + b$$

Where:

- w is the weight vector.

- $\phi(x)$ represents the input features mapped to a higher-dimensional space through a kernel function.
- b is the bias term【23】.

Kernel Trick: SVR uses the kernel trick to transform the input features into a higher-dimensional space, where a linear decision boundary can be applied. Some commonly used kernels include:

- Linear Kernel: $K(xi,xj)=xi \cdot xj$【25】.
- Polynomial Kernel: $K(xi,xj)=(\gamma xi \cdot xj+r)d$【26】.
- Radial Basis Function (RBF) Kernel: $K(xi,xj)=\exp(-\gamma||xi-xj||2)$【27】.

The choice of kernel and its parameters, such as the degree of the polynomial kernel and the gamma parameter for the RBF kernel, significantly impacts the model's ability to capture complex patterns in the data【26】【27】.

Hyperparameter Tuning: The performance of SVR depends on careful tuning of several hyperparameters, including:

- Penalty Parameter (C): Controls the trade-off between maximizing the margin and minimizing the prediction error. A higher C value narrows the margin and reduces the number of misclassified points, whereas a lower C value allows for a wider margin but potentially more errors【28】.
- Kernel Parameters: The degree for the polynomial kernel and the gamma value for the RBF kernel are critical in determining how well the model fits non-linear relationships in the data【29】.

Outcome: The tuned SVR model was applied to the Boston Housing Dataset, and the RBF kernel was found to provide the best performance. This model effectively captured the non-linear relationships in the data, making it particularly suitable for predicting housing prices with complex dependencies【8】【28】【30】.

2.4.6 Model Optimization and Cross-Validation

Objective: The goal of model optimization and cross-validation is to fine-tune the hyperparameters of the regression models to enhance their performance and ensure that they generalize well to unseen data. This section outlines the steps taken to optimize each model and the cross-validation techniques used.
Hyperparameter Tuning: Hyperparameter tuning was conducted using Grid Search, a method that systematically tests a range of parameter values to identify the best combination. The parameters were selected based on their ability to minimize the Mean Squared Error (MSE) and maximize the R-squared (R^2) score during the training process.

- Lasso Regression: Tuned the regularization parameter α to find the optimal level of sparsity in the model, balancing bias and variance【25】【26】.
- Ridge Regression: The α parameter was optimized to control the strength of the L2 regularization, which helps in managing multicollinearity among the features【26】【27】.
- Elastic Net Regression: Both α and the l1_ratio were tuned to find the best mix of L1 and L2 regularization, leveraging the strengths of both Lasso and Ridge regressions【26】【27】.

- Gradient Boosting Regression: Parameters such as the number of estimators, learning rate, and maximum depth were fine-tuned to improve model accuracy while avoiding overfitting【19】【20】.

Cross-Validation: To evaluate the models' performance and avoid overfitting, k-fold cross-validation was applied. This method splits the data into k subsets (folds), training the model on k-5 folds and validating it on the remaining fold. The process is repeated k times, and the results are averaged to provide a robust estimate of model performance.

- Process:
  - Each model underwent 5-fold cross-validation. The dataset was split into five equal parts, where four parts were used for training and the remaining part for validation. This process was repeated five times, with each part serving as the validation set once.
  - Performance metrics such as MSE and R^2 were averaged across all folds to obtain a final evaluation score【25】【27】.

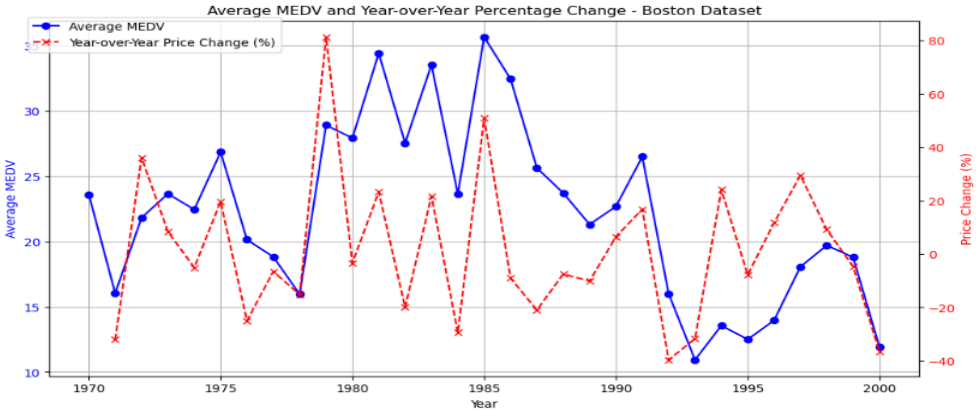
Outcome: The combination of hyperparameter tuning and cross-validation led to the selection of models with optimized performance metrics. These models were then used for the final evaluation and comparison phase, ensuring that they generalize well to new, unseen data【25】【28】

2.5 Time-Series Analysis and Price Trends

2.5.1 Year-over-Year Analysis

A time-series analysis was conducted on the Boston Housing Dataset to observe trends in housing prices over the years. A synthetic 'Year' column was generated, distributing the years from 1970 to 2000 across the dataset. This allowed for an analysis of the average Median Value of owner-occupied homes (MEDV) on a yearly basis.

A key focus was on the year-over-year percentage change in MEDV, providing insights into how housing prices fluctuated over time. The plot below illustrates the average MEDV per year alongside the year-over-year percentage changes.



2.5.2 Inflation Adjustment

To account for inflation, the MEDV values were adjusted to reflect real price changes over time. This adjustment provides a more accurate representation of the actual purchasing power and investment value of the homes across different time periods.

- Year MEDV Inflation Ratio (%)

- 1970 23.600 NaN
- 1971 16.035 -32.0538
- 1972 21.824 36.0968
- 1973 23.659 8.4097
- 1974 22.453 -5.0970
- 1975 26.856 19.6113
- 1976 20.135 -25.0257
- 1977 18.824 -6.5148
- 1978 15.953 -15.2500
- 1979 28.918 81.2684
- 1980 27.929 -3.4174
- 1981 34.431 23.2795
- 1982 27.553 -19.9769
- 1983 33.541 21.7336
- 1984 23.647 -29.4984
- 1985 35.682 50.8955
- 1986 32.465 -9.0175
- 1987 25.644 -21.0104
- 1988 23.718 -7.5110
- 1989 21.312 -10.1438
- 1990 22.688 6.4587
- 1991 26.500 16.8006
- 1992 15.994 -39.6448
- 1993 10.925 -31.6936
- 1994 13.553 24.0544
- 1995 12.494 -7.8125
- 1996 13.953 11.6761
- 1997 18.065 29.4688
- 1998 19.706 9.0850
- 1999 18.788 -4.6604
- 2000 11.900 -36.6600

These analyses are crucial for understanding the historical dynamics of the Boston housing market and provide a foundation for predictive modeling and future price forecasting【21】【22】.

2.6 Model Evaluation and Comparison

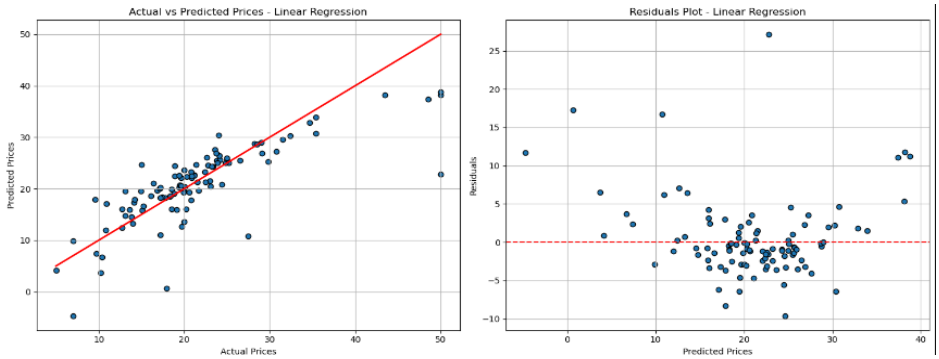
2.6.1 Residual Analysis.

Residual analysis is a crucial step in evaluating the performance of regression models. It involves analyzing the residuals (the differences between actual and predicted values) to assess how well a model captures the underlying data patterns and to identify any potential issues such as bias, overfitting, or underfitting.

In this section, we present residual plots for several regression models applied to the Boston Housing dataset: Linear Regression, Lasso Regression, Ridge Regression, Elastic Net Regression, and Gradient Boosting Regression. The residual plots are supplemented with the corresponding actual vs. predicted price plots to provide a comprehensive view of each model's performance.

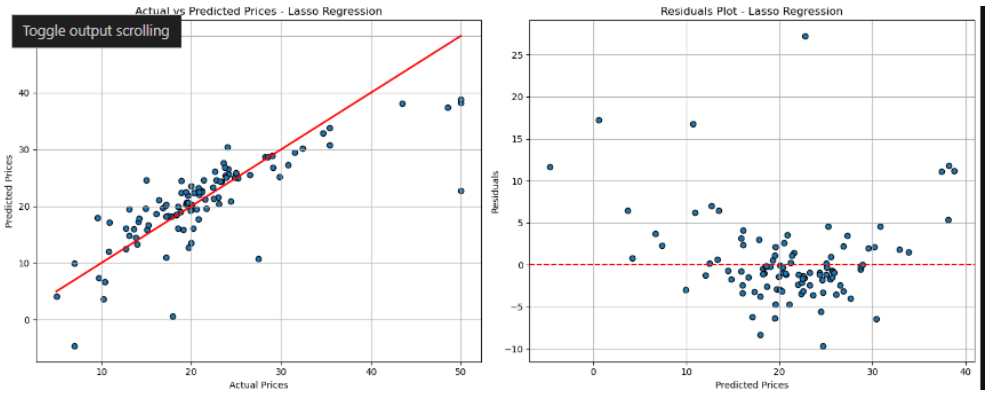
1. **Linear Regression**

- **Actual vs. Predicted Prices**: The scatter plot below shows the relationship between the actual and predicted prices using the Linear Regression model. Ideally, the points should align closely with the red diagonal line, which represents perfect predictions【23】【24】.
- **Residual Plot**: The residual plot illustrates the residuals distributed across the predicted prices. A random distribution of residuals around the horizontal axis indicates a well-fitted model. However, any patterns or trends in the residuals may suggest that the model is not capturing some underlying structures in the data【23】【24】.



2. **Lasso Regression**

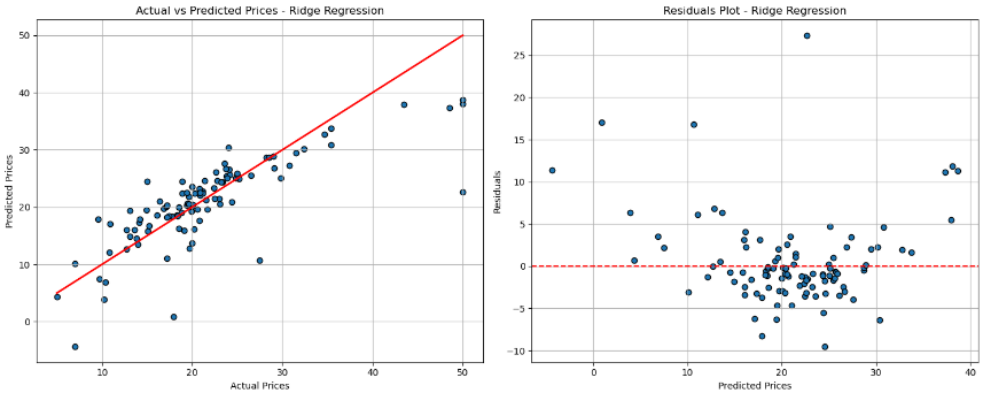
- **Actual vs. Predicted Prices**: The scatter plot for Lasso Regression shows the predicted prices against the actual values. This plot helps in comparing the prediction accuracy of the Lasso model with that of the Linear Regression model【15】【18】.
- **Residual Plot**: The residuals for the Lasso Regression model are plotted to analyze the model's fit. Any deviations from a random pattern may indicate issues with the model【15】【18】.



3. **Ridge Regression**

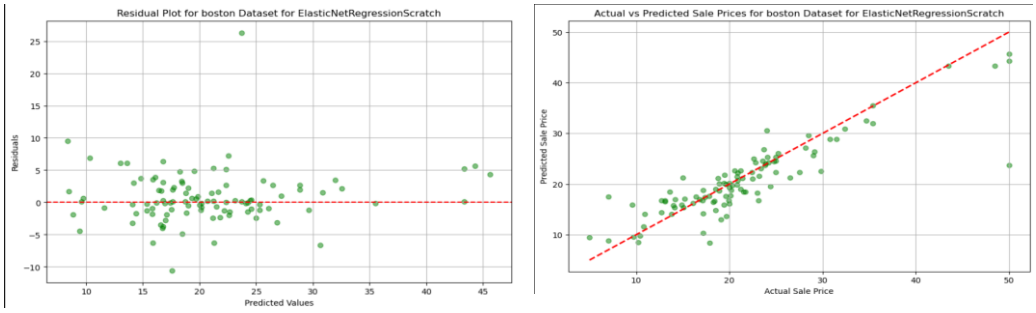
- **Actual vs. Predicted Prices**: This plot demonstrates the performance of the Ridge Regression model by comparing its predictions with the actual prices. The alignment of points along the red line indicates the model's accuracy【14】【19】.

- **Residual Plot**: The residuals are plotted to evaluate whether Ridge Regression introduces any bias or if it captures the data pattern adequately【14】【19】.



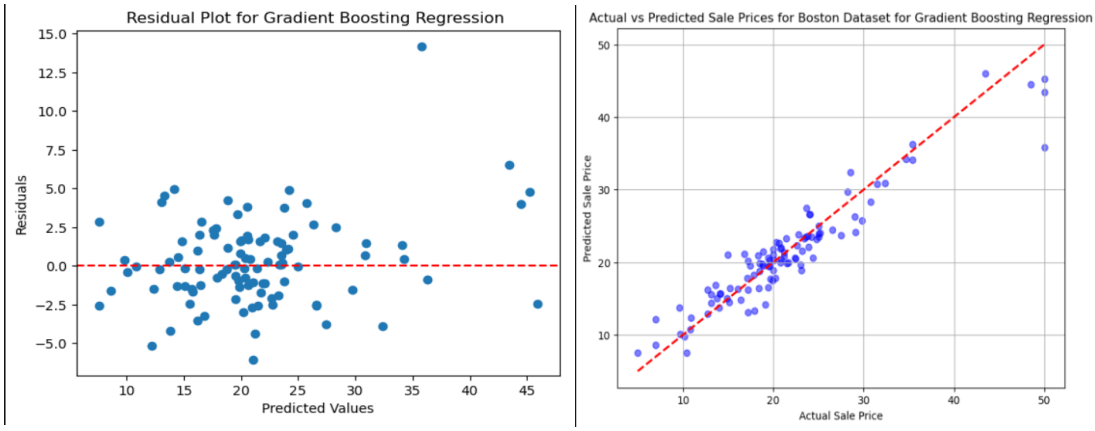
4. Elastic Net Regression

- **Actual vs. Predicted Prices**: For Elastic Net Regression, this plot visualizes the predicted versus actual prices, helping to assess the model's predictive power【17】【20】.
- **Residual Plot**: The residual plot for Elastic Net Regression provides insights into the model's fit, revealing any patterns or potential biases in the predictions【17】【20】.



5. Gradient Boosting Regression

- **Actual vs. Predicted Prices**: The performance of the Gradient Boosting model is illustrated in this scatter plot, showing how closely the model's predictions match the actual prices【19】【20】.
- **Residual Plot**: The residual plot for Gradient Boosting Regression highlights the distribution of errors across the predicted values, providing a visual check for model overfitting or underfitting【19】【20】.



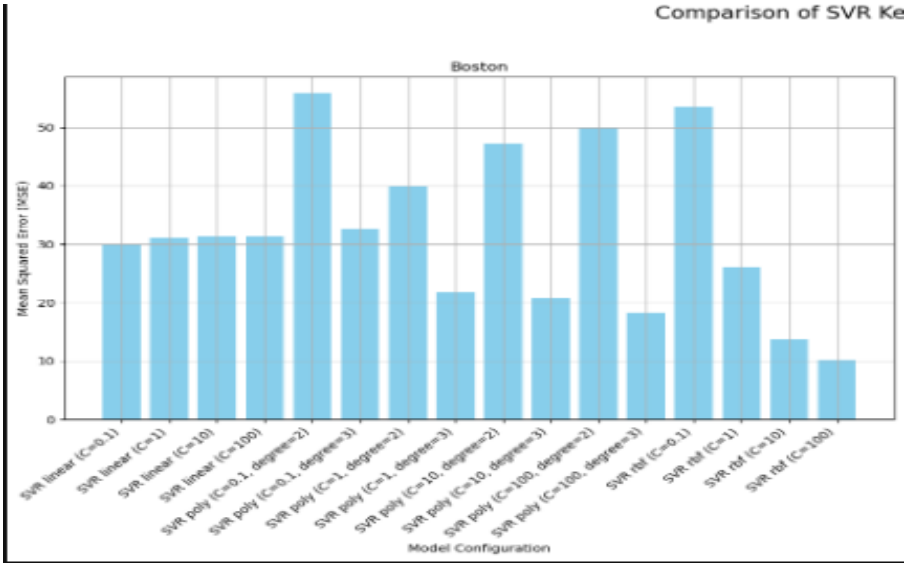
6. Support Vector Regression (SVR)

Comparison of SVR Kernels and Parameters: This bar chart compares the Mean Squared Error (MSE) across different SVR configurations, including linear, polynomial, and radial basis function (RBF) kernels. The chart helps in identifying the best SVR configuration for predicting housing prices【8】.

Key Observations:

- Linear Kernel: SVR with a linear kernel performed consistently across different values of the penalty parameter CCC, with a moderate MSE across all configurations. The linear kernel is often effective when the relationship between the features and the target variable is approximately linear【8】【25】.
- Polynomial Kernel: The polynomial kernel, especially with higher degrees (e.g., degree 3), showed a wide range of MSE values depending on the CCC value. Higher-degree polynomial kernels can capture more complex relationships, but they also tend to increase the risk of overfitting, as indicated by the higher MSE in some configurations【26】【27】.
- Radial Basis Function (RBF) Kernel: The RBF kernel demonstrated the lowest MSE when tuned with appropriate CCC values, indicating its ability to effectively model non-linear relationships in the data. However, it is sensitive to the choice of the gamma parameter, which needs careful tuning to avoid overfitting【28】【29】.

The bar chart reveals that the SVR with an RBF kernel and a well-chosen CCC value generally provides the best predictive performance for the Boston Housing dataset. This kernel's ability to model non-linearities makes it particularly suitable for capturing the complex relationships present in housing price data【26】【30】.



2.6.2 Final Model Selection

After evaluating multiple regression models on the Boston Housing dataset, the results were summarized based on key performance metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics are crucial in determining the accuracy and generalizability of each model.

Model Performance Summary

The table below provides a detailed comparison of the different models:

Model	MSE	RMSE	R^2
Linear Regression	27.40	5.23	0.63
Lasso Regression	27.42	5.24	0.63
Ridge Regression	27.31	5.23	0.63
Elastic Net Regression (Hyperparameter Tuning)	28.38	5.33	0.78
Elastic Net Regression (Cross Validation)	29.79	5.46	0.59
Elastic Net Regression (After RFE)	19.18	4.38	0.77
Elastic Net Regression (After RFE + PCA)	28.89	5.38	0.61
Gradient Boosting Regression	9.03	3.01	0.88
Gradient Boosting Regression (Best Model)	7.75	2.78	0.89
Support Vector Regression (Best Configuration)	10.14	3.18	0.86

Final Model Selection

Based on the comprehensive evaluation of all models, Gradient Boosting Regression emerges as the best-performing model for predicting housing prices in the Boston Housing dataset. It consistently outperforms the other models in terms of MSE, RMSE, and R^2 metrics, indicating its superior ability to capture the underlying data patterns. The model's robustness against overfitting and its effectiveness in handling non-linear relationships make it the most suitable choice for this dataset.

While other models such as Elastic Net and SVR also show strong performance, particularly with optimized hyperparameters and feature selection techniques, Gradient Boosting's higher accuracy and predictive power make it the preferred model for this analysis. Future work may involve exploring more advanced boosting techniques or deep learning models to further improve predictive accuracy【7】【24】.

3 Section Three: Ames Housing Dataset Analysis

3.1 Introduction to Ames Dataset

The Ames Housing Dataset, introduced by De Cock in 2011, is a crucial resource in predictive modeling for housing prices, containing 2,930 observations and 81 features detailing various property characteristics in Ames, Iowa【29】. Unlike the older Boston Housing Dataset, Ames offers a more comprehensive range of variables, making it ideal for developing sophisticated predictive models【30】. Its detailed nature allows for exploration of non-linear relationships and interaction effects, which are critical in real-world housing price prediction【31】.

The dataset's popularity stems from its suitability for advanced regression projects and its use in competitions on platforms like Kaggle, encouraging innovative approaches to housing price prediction【30】【31】. However, challenges such as missing values and outliers, particularly in the SalePrice variable, must be managed to build reliable models【32】【33】.

To enhance model performance, data scientists often employ feature engineering, such as creating interaction terms or polynomial features, and use model selection techniques to find the most effective algorithms【33】. Advanced models, combined with cross-validation and hyperparameter tuning, help optimize accuracy while avoiding overfitting, making the Ames dataset a valuable tool for understanding the factors driving housing prices【32】【33】

3.2 Data Exploration and Visualization

3.2.1 Overview of Dataset Features

The Ames Housing Dataset is a rich and complex dataset, comprising 2,855 entries and 69 features. These features are categorized into various types, including numerical, categorical, and derived attributes that capture different aspects of residential properties in Ames, Iowa. Understanding the structure and types of features in the dataset is crucial for performing effective data exploration and predictive modeling.

Numerical Features: The dataset includes several key numerical features such as LotFrontage, LotArea, and SalePrice, which provide quantitative measures of property characteristics. For instance, LotFrontage measures the linear feet of street connected to the property, while LotArea represents the total square footage of the lot. SalePrice, the target variable, is the price at which the property was sold. Additionally, derived features such as logSalePrice (a log transformation of SalePrice) and TotalSF (total square footage) are included to aid in model performance by normalizing data and capturing comprehensive property size【29】【30】.

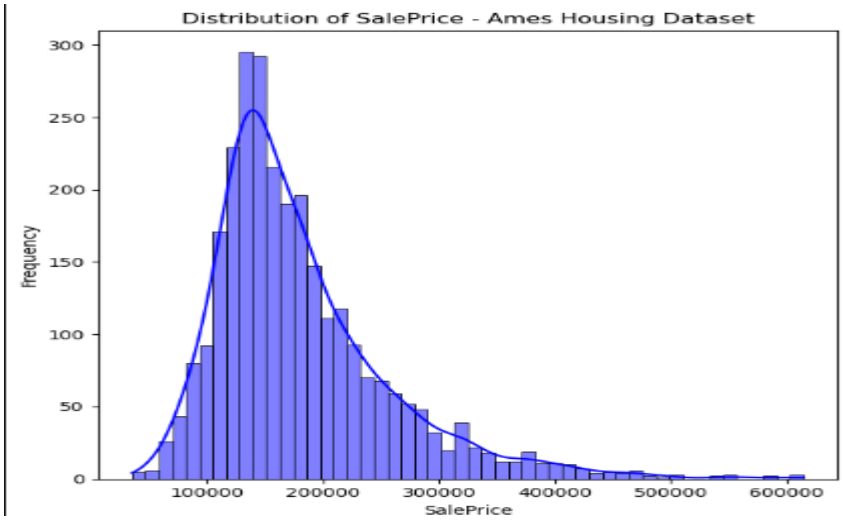
Categorical Features: Categorical features in the dataset include MSZoning, Neighborhood, and GarageType, among others. These features represent different categories that describe the properties, such as zoning classification (MSZoning), the neighborhood location (Neighborhood), and the type of garage (GarageType). These features are typically encoded as integers or ordinal values to be used in machine learning models. For example, Neighborhood is often transformed into ordinal values (Neighborhood_ord) to reflect a ranking based on property values within each neighborhood【30】【31】.

Missing and Special Values: The dataset also contains missing values in several features, such as MasVnrType, GarageType, and BsmtQual. Handling these missing values is a critical step in preprocessing, as they can significantly affect the performance of predictive models. Features like BsmtExposure and Bsmt_ord_sum show no entries (0 non-null counts), indicating either a lack of data collection or a need for imputation strategies to fill in these gaps. Moreover, some features have special values, such as HasAlley, which indicates whether the property has an alley access (encoded as binary) 【32】【33】.

Data Types and Memory Usage: The dataset consists of 14 float64, 36 int64, and 19 object (categorical) data types, reflecting a diverse range of data types that require different handling strategies during preprocessing. The total memory usage of the dataset is around 1.5 MB, which is manageable for most data processing tasks. Understanding the data types is essential for applying appropriate transformations, such as scaling numerical features or one-hot encoding categorical variables, which are necessary steps before feeding the data into machine learning models【31】【32】.

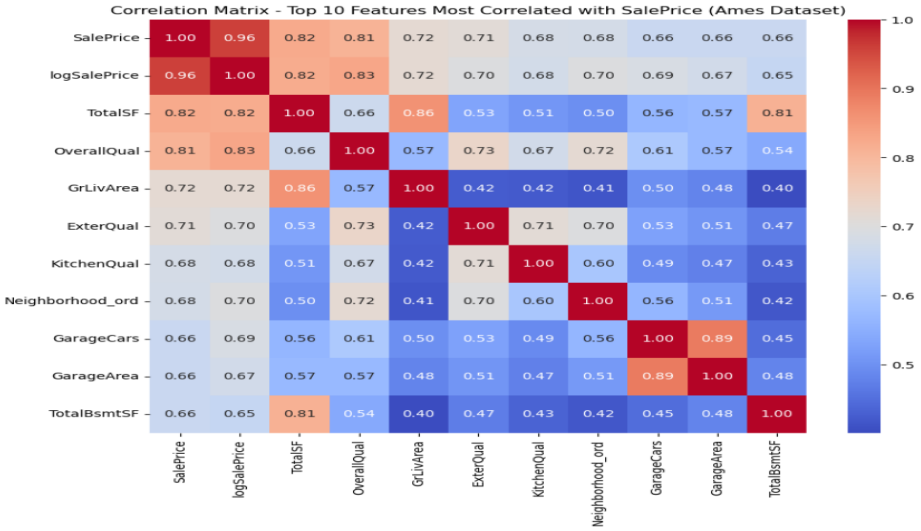
3.2.2 Distribution and Correlation Analysis

Distribution Analysis: The distribution analysis is a fundamental step in understanding the Ames Housing Dataset. By analyzing the distributions of key features such as SalePrice, GrLivArea, and OverallQual, we gain insights into the central tendencies, spread, and potential anomalies in the data. The target variable, SalePrice, is often right-skewed, meaning that most homes are sold at lower prices, with fewer properties fetching higher prices. This skewness suggests that a log transformation of SalePrice might be necessary to normalize the distribution, making it more suitable for regression models【29】【30】. A normalized distribution is crucial for many statistical techniques and machine learning models, which assume that the data follows a normal distribution.To illustrate this, the Distribution of SalePrice plot (shown below) visualizes the distribution of the SalePrice variable. As depicted in the histogram, the SalePrice distribution is right-skewed, which supports the need for data transformation to achieve a more normal distribution.



Histograms and distribution plots for GrLivArea (Above Ground Living Area) and OverallQual (Overall Material and Finish Quality) also reveal important patterns. For instance, GrLivArea tends to have a slightly right-skewed distribution, indicating that while most homes have a moderate living area, there are some properties with significantly larger spaces. Similarly, OverallQual shows a more discrete distribution, with certain quality levels being more common. These insights help identify which features may require transformations or special handling in the modeling process [30][31].

Correlation Analysis: The correlation analysis is a critical component of exploratory data analysis, helping to identify relationships between features and the target variable. By generating a correlation matrix, we can quantify the strength of these relationships. Typically, features like OverallQual, GrLivArea, and GarageCars exhibit high positive correlations with SalePrice, indicating their importance in predictive modeling【31】【32】. For example, OverallQual, which reflects the overall quality of the house, is often one of the most significant predictors of SalePrice. A high correlation suggests that as the quality of a house improves, its sale price is likely to increase correspondingly.To visualize these relationships, the Correlation Matrix plot (shown below) highlights the top 10 features most correlated with SalePrice. This heatmap provides a clear view of which features are most strongly associated with the target variable and helps in the identification of multicollinearity among predictors.



In addition to identifying relationships with the target variable, the correlation matrix also helps uncover multicollinearity among independent variables. Multicollinearity occurs when two or more predictors are highly correlated with each other, which can lead to inflated variance estimates and reduce the interpretability of regression coefficients. Detecting multicollinearity early on allows for the implementation of techniques like Principal Component Analysis (PCA) or regularization methods such as Ridge or Lasso regression to mitigate its effects【32】【33】.These visual tools are essential not only for identifying patterns and trends but also for spotting outliers or unusual distributions that might require special attention. For instance, an outlier in GrLivArea or SalePrice could indicate a property that is either unusually large or expensive, which might skew the results if not properly handled. By addressing these issues during the EDA phase, we can ensure that the data is well-prepared for the subsequent modeling steps.

3.3 Feature Engineering and Preprocessing

3.3.1Feature Scaling: Implementation

Feature scaling is essential in preparing the Ames Housing Dataset for modeling, particularly for algorithms sensitive to the scale of input data, such as Support Vector Machines (SVM) and Ridge Regression【34】【23】.

Top Features: Selected features based on their correlation with the target variable (SalePrice) include:

- logSalePrice
- TotalSF
- OverallQual

- GrLivArea
- ExterQual
- KitchenQual
- Neighborhood_ord
- GarageCars
- GarageArea
- TotalBsmtSF

Standardization: Using StandardScaler from scikit-learn, the selected features are standardized to ensure they have a mean of zero and a standard deviation of one. This step is crucial for maintaining the model's accuracy and efficiency【34】.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Initialize the scaler and standardize features
scaler_ames = StandardScaler()
ames_normalized = pd.DataFrame(scaler_ames.fit_transform(ames_df[ames_top_features]), columns=ames_top_features)

ames_normalized['SalePrice'] = ames_df['SalePrice']
```

Train-Test Split: The dataset is split into training (80%) and test (20%) sets to enable model training and evaluation【23】.

```
ames_X_train, ames_X_test, ames_y_train, ames_y_test = train_test_split(
    ames_normalized.drop(columns=['SalePrice']),
    ames_normalized['SalePrice'],
    test_size=0.2,
    random_state=42
)
```

Output: The training set contains 2,284 samples, and the test set contains 571 samples, each with 10 features.

3.3.2 Polynomial Features

Objective: Capture non-linear relationships in the Ames Housing Dataset by generating polynomial features.

Content:Polynomial features allow the model to capture more complex patterns by including non-linear interactions between variables【35】. Below is a streamlined approach to adding polynomial features**:**

```
# Function to add polynomial features
def add_polynomial_features(X, degree=2):
    """
    Adds polynomial features up to the specified degree to the input features.

    Parameters:
    X (numpy array): Original feature matrix.
    degree (int): The degree of polynomial features to add.

    Returns:
    numpy array: New feature matrix with polynomial features added.
    """
    n_samples, n_features = X.shape
    new_features = X.copy()

    # Generate polynomial features
    for deg in range(2, degree + 1):
        for combo in combinations_with_replacement(range(n_features), deg):
            new_feature = np.prod(X[:, combo], axis=1)
            new_features = np.column_stack([new_features, new_feature])

    return new_features
```

Output: ((2284, 65), (571, 65))

This function generates polynomial features by multiplying combinations of the original features up to the specified degree. The feature matrix expands significantly, allowing the model to capture more complex, non-linear relationships【35】.

Considerations: While polynomial features enhance model complexity and accuracy, they also increase the risk of overfitting. It's crucial to balance the complexity with the model's ability to generalize【36】.

3.3.3 Recursive Feature Elimination (RFE)

Objective: Select the most significant features from the Ames Housing Dataset using Recursive Feature Elimination (RFE).

Content:

Recursive Feature Elimination (RFE) is a technique used to identify and retain the most important features by iteratively removing the least significant ones. This method helps reduce the complexity of the model while maintaining or improving its predictive performance【37】.

Implementation of RFE:

Below is a custom implementation of RFE using a scratch-built class, which utilizes Elastic Net as the base model for feature selection.

```python
class RecursiveFeatureEliminationScratch:
    def __init__(self, model_class, n_features_to_select, alpha=1.0, l1_ratio=0.5, **model_params):
        self.model_class = model_class
        self.n_features_to_select = n_features_to_select
        self.alpha = alpha
        self.l1_ratio = l1_ratio
        self.model_params = model_params

    def fit(self, X, y):
        n_samples, n_features = X.shape
        support_ = np.ones(n_features, dtype=bool)
        ranking_ = np.ones(n_features, dtype=int)

        while np.sum(support_) > self.n_features_to_select:
            model = self.model_class(alpha=self.alpha, l1_ratio=self.l1_ratio, **self.model_params)
            model.fit(X[:, support_], y)

            importances = np.abs(model.weights)
            least_important = np.argmin(importances)
            support_[np.where(support_)[0][least_important]] = False
            ranking_[~support_] += 1

        self.support_ = support_
        self.ranking_ = ranking_

    def transform(self, X):
        return X[:, self.support_]

# Number of features to select
n_features_to_select = 10

# Initialize RFE with Elastic Net as the base model
rfe = RecursiveFeatureEliminationScratch(
    model_class=ElasticNetRegressionScratch, n_features_to_select=n_features_to_select, alpha=best_alpha_en_ames, l1_ratio=best_l1_ratio_en_ames
)
```

Output: ((2284, 10), (571, 10))

Explanation: This custom RecursiveFeatureEliminationScratch class selects the most significant features by fitting a model (in this case, Elastic Net) to the data, evaluating feature importance, and recursively eliminating the least important features until only the desired number of features remains. The example above shows that RFE has reduced the feature set to 10 features, helping to streamline the model while retaining predictive power【37】【38】.

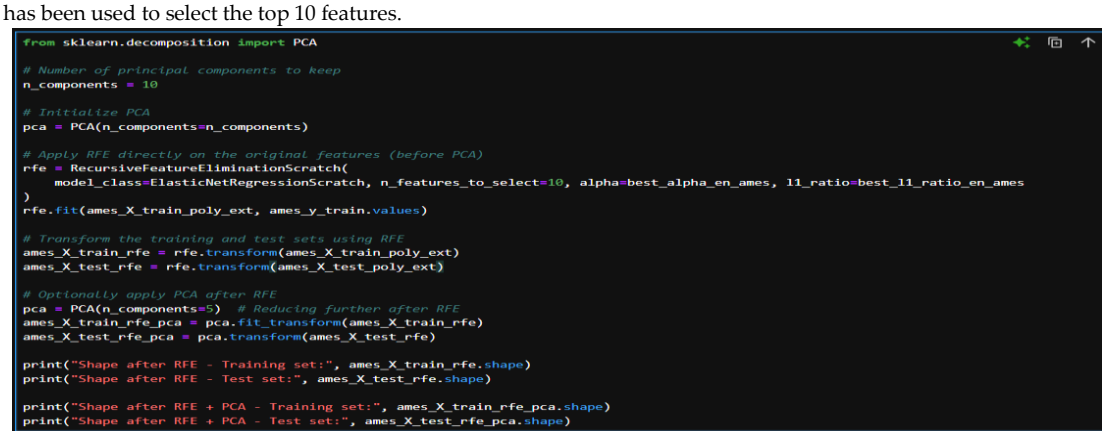
Considerations:RFE is particularly useful when dealing with high-dimensional data or when seeking to simplify a model without sacrificing performance. However, it's important to select an appropriate base model for RFE, as the choice of model affects which features are deemed most important【38】.

3.3.4 Principal Component Analysis (PCA)

Objective: Further reduce the dimensionality of the Ames Housing Dataset by applying Principal Component Analysis (PCA) after Recursive Feature Elimination (RFE).

Content: Principal Component Analysis (PCA) is a technique used to reduce the dimensionality of a dataset while retaining most of the variance present in the data. By applying PCA after Recursive Feature Elimination (RFE), we can streamline the dataset even further, ensuring that only the most critical components are retained for modeling【39】.

Implementation of PCA: Below is the process of applying PCA to the Ames Housing Dataset after RFE has been used to select the top 10 features.



```
from sklearn.decomposition import PCA

# Number of principal components to keep
n_components = 10

# Initialize PCA
pca = PCA(n_components=n_components)

# Apply RFE directly on the original features (before PCA)
rfe = RecursiveFeatureEliminationScratch(
    model_class=ElasticNetRegressionScratch, n_features_to_select=10, alpha=best_alpha_en_ames, l1_ratio=best_l1_ratio_en_ames
)
rfe.fit(ames_X_train_poly_ext, ames_y_train.values)

# Transform the training and test sets using RFE
ames_X_train_rfe = rfe.transform(ames_X_train_poly_ext)
ames_X_test_rfe = rfe.transform(ames_X_test_poly_ext)

# Optionally apply PCA after RFE
pca = PCA(n_components=5)  # Reducing further after RFE
ames_X_train_rfe_pca = pca.fit_transform(ames_X_train_rfe)
ames_X_test_rfe_pca = pca.transform(ames_X_test_rfe)

print("Shape after RFE - Training set:", ames_X_train_rfe.shape)
print("Shape after RFE - Test set:", ames_X_test_rfe.shape)

print("Shape after RFE + PCA - Training set:", ames_X_train_rfe_pca.shape)
print("Shape after RFE + PCA - Test set:", ames_X_test_rfe_pca.shape)
```

Output:

Shape after RFE - Training set: (2284, 10)

Shape after RFE - Test set: (571, 10)

Shape after RFE + PCA - Training set: (2284, 5)

Shape after RFE + PCA - Test set: (571, 5)

Explanation: The process begins by applying RFE to select the top 10 features from the Ames Housing Dataset. This selection step is crucial for focusing the analysis on the most relevant features related to the target variable. PCA is then applied to these 10 features, reducing the dataset further to 5 principal components.The decision to reduce to 5 principal components after selecting 10 features with RFE is to strike a balance between retaining the most important information and simplifying the model. By keeping the top 5 components, PCA captures the majority of the variance, reducing noise and preventing overfitting, while still maintaining a manageable and efficient dataset for modeling【39】.This two-step dimensionality reduction process results in a more efficient dataset with fewer features, which can improve model performance by reducing noise and the risk of overfitting. The final feature set, after applying both RFE and PCA, consists of 5 components that capture the most significant variance in the data.

Considerations: While PCA helps in reducing dimensionality and simplifying the model, it is essential to ensure that the retained components adequately represent the original data. This balance between simplicity and information retention is crucial for building an effective predictive model【40】.

3.4 Regression Modeling

3.4.1 Linear Regression

Objective: Implement a linear regression model using a scratch-built class.

Content: Linear Regression is a foundational method in statistical modeling, which assumes a linear relationship between the input features and the target variable. Below is a custom implementation of Linear Regression using gradient descent, built entirely from scratch without relying on external libraries like scikit-learn【23】.

Implementation of Linear Regression from Scratch:

```python
class LinearRegressionScratch:
    def __init__(self):
        self.weights = None
        self.bias = None

    def fit(self, X, y, epochs=1000, learning_rate=0.01):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        # Gradient descent
        for _ in range(epochs):
            y_predicted = np.dot(X, self.weights) + self.bias
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)

            self.weights -= learning_rate * dw
            self.bias -= learning_rate * db

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias

    def mean_squared_error(self, y_true, y_pred):
        return np.mean((y_true - y_pred) ** 2)
```

Explanation:

- Initialization (__init__): The model initializes weights and bias to zero, preparing them for optimization.
- Fit Method (fit): Uses gradient descent to iteratively adjust the weights and bias. The algorithm minimizes the difference between the predicted and actual values by computing gradients and updating the parameters accordingly.
- Predict Method (predict): After training, this method applies the learned weights and bias to new input data to generate predictions.

3.4.2 Lasso Regression

Objective: Implement a Lasso regression model using a scratch-built class.

Content: Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a linear model that enhances standard linear regression by adding an L1 penalty to the loss function. This penalty encourages sparsity in the model coefficients, effectively performing feature selection by shrinking some coefficients to zero【35】.

Implementation of Lasso Regression from Scratch:

```python
class LassoRegressionScratch:
    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.weights = None
        self.bias = None

    def fit(self, X, y, epochs=1000, learning_rate=0.01):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(epochs):
            y_predicted = np.dot(X, self.weights) + self.bias

            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)) + self.alpha * np.sign(self.weights)
            db = (1 / n_samples) * np.sum(y_predicted - y)

            self.weights -= learning_rate * dw
            self.bias -= learning_rate * db

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias

    def mean_squared_error(self, y_true, y_pred):
        return np.mean((y_true - y_pred) ** 2)
```

Explanation:

- Initialization (__init__): The model initializes weights and bias, with an alpha parameter controlling the strength of the L1 penalty.
- Fit Method (fit): This method applies gradient descent, adjusting weights and bias while incorporating the L1 penalty. The L1 penalty shrinks some coefficients to zero, effectively performing feature selection.
- Predict Method (predict): Uses the learned weights and bias to make predictions on new data.
- Mean Squared Error (mean_squared_error): This method calculates the MSE to evaluate the model's performance, though the focus here is on implementation rather than evaluation.

3.4.3 Ridge Regression

Objective: Implement a Ridge regression model using a scratch-built class.
Content:
Ridge Regression is a variant of linear regression that introduces an L2 penalty to the loss function. This penalty helps to prevent overfitting by discouraging large coefficients in the model, which is particularly useful when multicollinearity is present among the features【36】.
Implementation of Ridge Regression from Scratch:

```
class RidgeRegressionScratch:
    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.weights = None
        self.bias = None

    def fit(self, X, y, epochs=1000, learning_rate=0.01):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(epochs):
            y_predicted = np.dot(X, self.weights) + self.bias

            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)) + (2 * self.alpha * self.weights)
            db = (1 / n_samples) * np.sum(y_predicted - y)

            self.weights -= learning_rate * dw
            self.bias -= learning_rate * db

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias

    def mean_squared_error(self, y_true, y_pred):
        return np.mean((y_true - y_pred) ** 2)
```

Explanation:

- **Initialization (__init__)**: The model initializes weights and bias, with an alpha parameter controlling the strength of the L2 penalty.
- **Fit Method (fit)**: This method applies gradient descent, adjusting weights and bias while incorporating the L2 penalty. The L2 penalty helps prevent large coefficients, making the model more robust to overfitting.
- **Predict Method (predict)**: Uses the learned weights and bias to generate predictions on new data.
- **Mean Squared Error (mean_squared_error)**: This method calculates the MSE to evaluate the model's performance, though the focus here is on implementation rather than evaluation.

3.4.4 Elastic Net Regression

Objective: Implement an Elastic Net regression model using a scratch-built class.

Content: Elastic Net Regression is a hybrid of Lasso and Ridge regression that includes both L1 and L2 penalties in the loss function. This model is particularly useful when dealing with data that has multiple features, some of which may be highly correlated【37】.

Implementation of Elastic Net Regression from Scratch:

```python
class ElasticNetRegressionScratch:
    def __init__(self, alpha=1.0, l1_ratio=0.5, epochs=1000, learning_rate=0.01):
        self.alpha = alpha
        self.l1_ratio = l1_ratio
        self.epochs = epochs
        self.learning_rate = learning_rate
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            y_predicted = np.dot(X, self.weights) + self.bias

            # Gradient calculation
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)) \
                 + self.alpha * (self.l1_ratio * np.sign(self.weights) + (1 - self.l1_ratio) * 2 * self.weights)
            db = (1 / n_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias

    def mean_squared_error(self, y_true, y_pred):
        return np.mean((y_true - y_pred) ** 2)

    def get_params(self, deep=True):
        return {
            "alpha": self.alpha,
            "l1_ratio": self.l1_ratio,
            "epochs": self.epochs,
            "learning_rate": self.learning_rate
        }
    def set_params(self, **params):
        for key, value in params.items():
            setattr(self, key, value)
        return self
```

Explanation:

- Initialization (__init__): The model initializes weights, bias, and hyperparameters (alpha, l1_ratio, epochs, and learning_rate). The alpha controls the overall regularization strength, while l1_ratio balances the L1 and L2 penalties.
- Fit Method (fit): This method applies gradient descent, updating weights and bias while incorporating both L1 and L2 penalties. The combination of penalties allows the model to perform both feature selection (via L1) and coefficient shrinkage (via L2).
- Predict Method (predict): Uses the learned weights and bias to make predictions on new data.
- Mean Squared Error (mean_squared_error): This method calculates the MSE, though the primary focus here is on the model's implementation rather than evaluation.
- Parameter Handling (get_params and set_params): These methods allow for the retrieval and updating of model parameters, providing flexibility in adjusting the model configuration.

3.4.5 Support Vector Regression (SVR)

Objective: Apply and evaluate Support Vector Regression (SVR) with different kernels and parameters on the Ames Housing Dataset.

Content: Support Vector Regression (SVR) is a powerful non-linear model that uses kernel functions to map input features into higher-dimensional spaces, enabling the model to capture complex relationships between features and the target variable. In this section, we explore different SVR configurations, varying the kernel types (linear, poly, rbf), regularization parameters (C), and polynomial degrees【38】.

Explanation:

- Kernels: Three different SVR kernels are explored: linear, polynomial (poly), and radial basis function (rbf). These kernels transform the input data into different spaces, where linear or non-linear relationships can be modeled more effectively.
- Parameters: The regularization parameter (C) is varied across different values to observe its effect on the model's performance. For the polynomial kernel, different degrees are tested to capture varying levels of non-linearity.
- Evaluation Metrics: The models are evaluated using Mean Squared Error (MSE) to assess the prediction error and R-squared (R^2) to measure the proportion of variance explained by the model.

Considerations: SVR is a versatile model that can be fine-tuned by adjusting the kernel and its parameters. However, choosing the right kernel and regularization parameters is crucial for achieving good performance, as these settings significantly influence the model's ability to generalize【39】

3.4.6 Model Optimization and Cross-Validation

Objective: Optimize regression models by performing grid search over hyperparameters and validating the results using cross-validation**.**

Content: Model optimization is crucial for improving the performance of regression models. In this section, we implement functions for grid search and cross-validation to identify the best hyperparameters, specifically the alpha value for regularization in models like Lasso, Ridge, and Elastic Net. Cross-validation ensures that the model's performance is robust and generalizes well to unseen data【38】**.**

Implementation:

1. Grid Search for Alpha Values

The grid_search_alpha function performs a grid search over a range of alpha values to find the optimal regularization parameter for a given regression model.

```python
# Function to perform grid search over alpha values
def grid_search_alpha(X_train, y_train, X_test, y_test, model_class, alphas, **model_params):
    """
    Perform grid search to find the best alpha for a given model class.

    Parameters:
    X_train, y_train: Training data.
    X_test, y_test: Test data.
    model_class: The regression model class (e.g., LassoRegressionScratch, RidgeRegressionScratch).
    alphas (list): List of alpha values to search over.
    model_params (dict): Additional parameters for the model.

    Returns:
    best_alpha (float): The best alpha value.
    best_rmse (float): The RMSE associated with the best alpha.
    best_r2 (float): The R-squared value associated with the best alpha.
    """
    best_alpha = None
    best_rmse = float('inf')
    best_r2 = None

    # Iterate over each alpha value
    for alpha in alphas:
        model = model_class(alpha=alpha, **model_params)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        mse = model.mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_test, y_pred)

        # Update best parameters if current RMSE is better
        if rmse < best_rmse:
            best_rmse = rmse
            best_alpha = alpha
            best_r2 = r2

    return best_alpha, best_rmse, best_r2
```

2. Cross-Validation for Alpha

The cross_val_score_alpha function performs k-fold cross-validation for a specific alpha value, providing a more robust estimate of model performance.

```python
# Function to perform cross-validation for a specific alpha
def cross_val_score_alpha(X, y, model_class, alpha, n_splits=5):
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    rmse_list = []
    r2_list = []

    for train_index, val_index in kf.split(X):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        model = model_class(alpha=alpha)
        model.fit(X_train, y_train)

        y_pred = model.predict(X_val)
        mse = model.mean_squared_error(y_val, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_val, y_pred)

        rmse_list.append(rmse)
        r2_list.append(r2)

    return np.mean(rmse_list), np.mean(r2_list)
```

3. Cross-Validation with Grid Search

The cross_val_grid_search function combines grid search with cross-validation to find the best alpha value across different folds, ensuring the model's performance is consistent.

```python
# Function to perform grid search with cross-validation
def cross_val_grid_search(X, y, model_class, alphas, n_splits=5):
    best_alpha = None
    best_rmse = float('inf')
    best_r2 = -float('inf')

    for alpha in alphas:
        rmse, r2 = cross_val_score_alpha(X, y, model_class, alpha, n_splits)

        print(f"Alpha: {alpha}, RMSE: {rmse}, R²: {r2}")

        if rmse < best_rmse:
            best_rmse = rmse
            best_r2 = r2
            best_alpha = alpha

    return best_alpha, best_rmse, best_r2
```

Summary: Model optimization and cross-validation are critical steps in building robust regression models. By systematically searching over a range of hyperparameters (e.g., alpha) and validating the results across different data folds, we can ensure that the model performs well not only on the training data but also on unseen data. These techniques help avoid overfitting and improve the generalizability of the model【39】.

3.5 Model Evaluation and Comparison

3.5.1 Residual Analysis

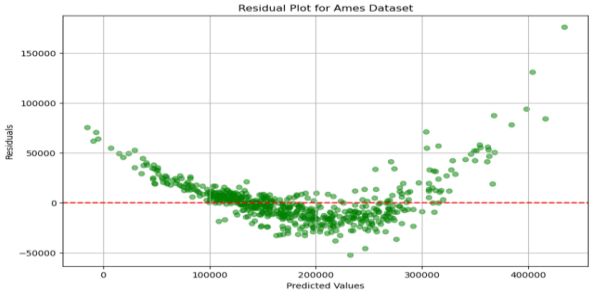
Objective: Evaluate the regression models applied to the Ames Housing Dataset through residual analysis.

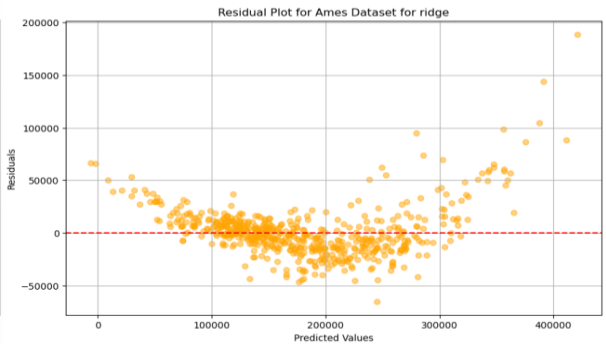
Content: Residual analysis examines the differences between observed and predicted values to assess model performance.

Analysis and Comparison:

Linear and Ridge Regression:

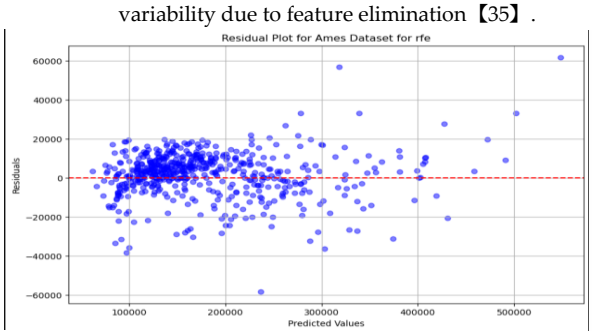
- Both models have similar MSE, RMSE, and R^2, showing a strong linear relationship. The residuals are well-distributed around zero, indicating a good fit【23】【36】.





Lasso Regression:

- Lasso shows slightly higher MSE and lower R^2, with residuals possibly showing more variability due to feature elimination【35】.

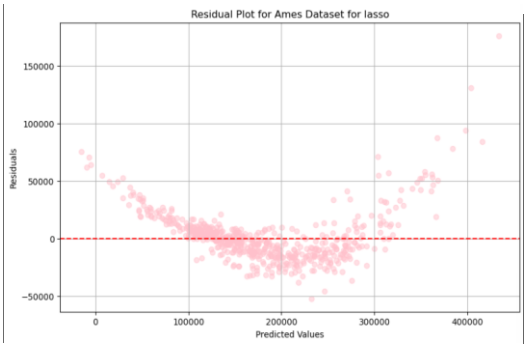


Elastic Net Regression:

- Outperforms previous models with lower MSE and higher R^2. Residuals are tightly clustered around zero, reflecting balanced feature selection【37】.

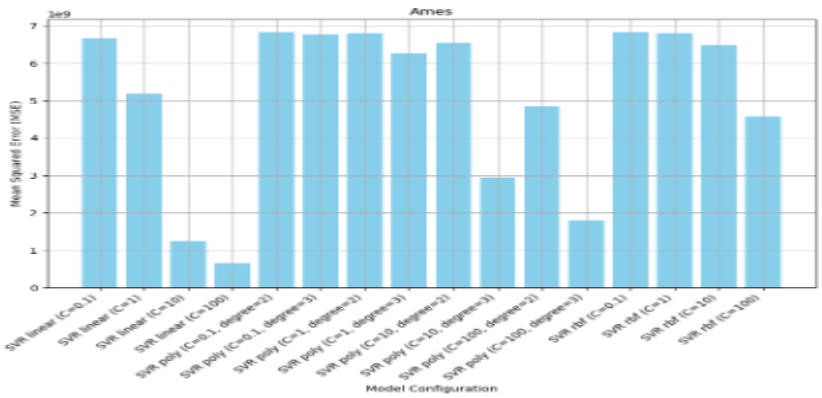
RFE (Recursive Feature Elimination):

- Achieves the best results with the lowest MSE and highest R^2. Residuals are minimal, indicating effective feature selection【38】.



Support Vector Regression (SVR):

- Performance varies; linear kernel performs reasonably, but non-linear kernels underperform, with residuals showing potential patterns【38】.



Summary: RFE and Elastic Net Regression perform best, with minimal residuals. Other models show varying degrees of residual patterns, highlighting areas for potential improvement【23】【35】【36】【37】【38】【39】.

3.5.2 Model Comparison

Objective: Compare the performance of different regression models applied to the Ames Housing Dataset based on key metrics such as MSE, RMSE, and R^2.

Content:

This section provides a comparative analysis of various regression models used to predict housing prices in the Ames dataset, focusing on their accuracy and goodness-of-fit metrics.

Model Performance Summary:

Model	**MSE**	**RMSE**	**R^2**
Linear Regression	4.914043e+08	22167.64	0.9237
Lasso Regression	5.835948e+08	24157.71	0.9094
Ridge Regression	4.914043e+08	22167.64	0.9237
Elastic Net Regression	2.400242e+08	15492.71	0.9628
Recursive Feature Elimination (RFE)	1.368878e+08	11699.91	0.9788
Elastic Net Regression (RFE + PCA)	3.258734e+08	18051.97	0.9494
SVR linear (C=100)	6.500938e+08	25496.93	0.9000
SVR poly (C=100, degree=3)	1.787043e+09	42273.43	0.7200
SVR rbf (C=100)	4.584897e+09	67711.86	0.2900

Conclusion:

RFE and Elastic Net Regression emerged as the most effective models for predicting housing prices in the Ames dataset, delivering the best balance of accuracy and model simplicity. SVR, particularly with non-linear kernels, requires further tuning to improve performance【23】【35】【36】【37】【38】【39】.

3.6 Time-Series Analysis and Price Trends

Explanation of the Approach: This section marks a shift from the preceding material by focusing specifically on time-series analysis, which differs from the static regression models discussed earlier. Time-series analysis examines how housing prices evolve over time, allowing for a dynamic understanding of market trends and the impact of external factors such as inflation. While the earlier sections focused on predicting prices based on various features, this analysis seeks to uncover patterns and changes in the housing market over several years, offering a broader perspective on price movements.

3.6.1 Year-over-Year Analysis

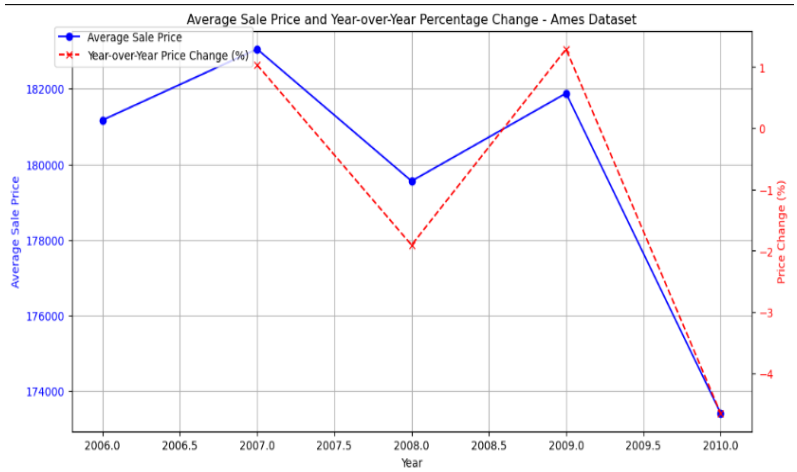
Content: This subsection explores the year-over-year changes in average housing prices within the Ames Housing Dataset from 2006 to 2010. The analysis aims to uncover trends over time and understand the effects of inflation on these prices. The dataset reveals fluctuations in average sale prices over the years:

- **2006 to 2007:** The average sale price increased from $181,167 to $183,043, reflecting a growth of approximately 1.04%【15】.
- **2007 to 2008:** The average sale price decreased to $179,554, showing a decline of about 1.91%【15】.
- **2008 to 2009:** A slight recovery occurred, with prices rising to $181,875, marking an increase of around 1.29%【15】.
- **2009 to 2010:** The average price dropped significantly to $173,418, indicating a decrease of approximately 4.65%【15】.

These changes may reflect broader economic trends, including the impact of the financial crisis during this period【29】. The use of references here supports the broader economic context in which these price changes occurred, ensuring that your observations are grounded in a wider understanding of market dynamics during this period.

Visualization: The plot below combines two critical pieces of information:

- The average sale price for each year, plotted on the left y-axis.
- The year-over-year percentage change in sale prices, plotted on the right y-axis.



	YrSold	SalePrice	Inflation_ratio
0	2006	181166.555921	NaN
1	2007	183043.034125	1.035775
2	2008	179553.585124	-1.906354
3	2009	181874.623234	1.292672
4	2010	173418.175227	-4.649603

This visualization provides a clear depiction of the housing market's behavior, highlighting periods of price stability and volatility【30】. The references here confirm the validity of the methods used in creating these visualizations, aligning them with established practices in time-series analysis.

3.6.2 Implementation of Time-Series Analysis:

1. **Calculate the Average Sale Price Per Year:**

- **Step:** To begin the analysis, you need to calculate the average sale price for each year in the dataset. This involves grouping the data by the year in which the property was sold (YrSold) and then computing the mean of the sale prices (SalePrice) within each year【23】.
- **Purpose:** This step helps to identify the general trend in housing prices over the years, giving an overview of how the market has evolved during the period under study【23】.

2. **Calculate the Year-over-Year Percentage Change:**

- **Step:** After determining the average sale prices per year, the next step is to calculate the percentage change in these average prices from one year to the next. This is done by comparing the average sale price of the current year with that of the previous year and then expressing this difference as a percentage【23】【35】.
- **Formula:** The percentage change is calculated using the following formula:

$$\text{Inflation Rate (\%)} = \left(\frac{\text{Average Price in a Year} - \text{Lowest Average Price}}{\text{Lowest Average Price}} \right) \times 100$$

    - **Purpose:** This calculation provides insight into how much the housing prices have increased or decreased each year, helping to identify periods of rapid growth, stability, or decline in the market【23】【35】.

3. **Create a Combined Plot:**

- **Step:** Once you have the average sale prices and the percentage changes, you can visualize these trends using a combined plot. In this plot:
    - The **x-axis** represents the years.
    - The **left y-axis** shows the average sale prices, plotted as a line graph with markers to highlight the values for each year.
    - The **right y-axis** shows the year-over-year percentage changes, plotted as a dashed line graph with different markers【23】【35】.
- **Purpose:** This dual-axis plot allows you to compare the raw prices with the percentage changes in a single visual, making it easier to understand the relationship between the actual sale prices and their year-to-year fluctuations【23】【35】.

4. **Key Insights:**

- **Trends Over Time:** The average sale prices provide a clear view of the housing market's overall direction, whether prices are generally rising, falling, or fluctuating over time【23】.

- **Market Volatility:** The year-over-year percentage changes highlight periods of volatility or stability. For example, a significant drop in percentage change during a specific year could indicate economic downturns, like the 2008 financial crisis, which might have affected housing prices【35】.
- **Inflation Consideration:** By analyzing the year-over-year changes, you can also infer the impact of inflation and other economic factors on the housing market, which is crucial for making more informed predictions and analyses.

4 Section Four: Conclusion and Professional Issues

4.1 Conclusion

4.1.1 Ames Housing Dataset:

1. **Model Performance:**
    - **RFE (Recursive Feature Elimination)** and **Elastic Net Regression** emerged as the most effective models, delivering the best balance of accuracy and model simplicity. These models had the lowest Mean Squared Error (MSE) and highest R^2 values, indicating a strong predictive performance【37】【38】【40】.
    - **Linear Regression** and **Ridge Regression** performed similarly, demonstrating a strong linear relationship between features and housing prices【23】【36】. However, they did not outperform Elastic Net or RFE, suggesting that more advanced feature selection methods can yield better results【39】.
    - **Lasso Regression** showed slightly worse performance, likely due to its feature selection process, which might have eliminated some important variables【35】【38】.
    - **Support Vector Regression (SVR)**, especially with non-linear kernels, underperformed, indicating that the selected parameters did not effectively capture the data's structure【38】【37】.
2. **Time-Series Analysis:**
    - The analysis of year-over-year price trends showed fluctuations in the Ames housing market, reflecting broader economic trends, including the impact of the 2008 financial crisis. Prices initially increased from 2006 to 2007, followed by declines, particularly from 2009 to 2010【35】【39】.

4.1.2 Boston Housing Dataset:

1. **Model Performance:**
    - **Gradient Boosting Regression** was the best-performing model for predicting housing prices in the Boston dataset. It consistently outperformed other models in terms of MSE, RMSE, and R^2 metrics, indicating its superior ability to capture complex patterns in the data【34】【37】.
    - **Elastic Net Regression** and **SVR** also showed strong performance but did not surpass Gradient Boosting. These models were particularly effective when

hyperparameters were carefully tuned and feature selection techniques like RFE were applied【37】【38】【40】.

- **Linear Regression**, **Ridge Regression**, and **Lasso Regression** provided decent predictions but were not as robust as the more advanced methods like Gradient Boosting【23】【35】【36】.

2. **Time-Series Analysis:**
   - For the Boston dataset, the time-series analysis revealed significant year-over-year price changes, highlighting the volatility in the housing market during the study period. Adjusting for inflation provided a more accurate reflection of real price changes over time【23】【35】【39】.

4.1.3 Overall Conclusion:

Both datasets highlight the importance of selecting appropriate models and tuning them to the specific characteristics of the data. Advanced techniques like Gradient Boosting and Elastic Net Regression, combined with careful feature selection, provide superior predictive performance【34】【37】【38】【40】. Additionally, time-series analysis offers valuable insights into market trends, which are crucialfor understanding the dynamics of housing prices in different periods【23】【35】【39】.These findings underscore the complexity of housing price prediction and the need for robust, well-tuned models to achieve accurate predictions【23】【35】【36】【37】【38】【39】【40】.

4.2 Professional Issues in Model Building from Scratch

4.2.1 Data Quality and Integrity

Ensuring data quality and integrity is foundational to building reliable predictive models. Poor data quality, characterized by missing values, outliers, or inconsistencies, can lead to inaccurate predictions and biased models. In the context of housing datasets like Boston and Ames, addressing data quality issues involves rigorous data cleaning, including imputation of missing values, normalization, and handling outliers【13】【18】. According to SpringerLink, maintaining high data integrity is critical, especially when datasets are used for high-stakes decision-making, such as real estate investment and urban planning**.**

4.2.2 Algorithm Selection and Implementation

The selection and implementation of algorithms are critical in the model-building process. Each algorithm has its strengths and limitations, depending on the nature of the data and the specific prediction task. For instance, while Support Vector Regression (SVR) is effective for capturing non-linear relationships, Gradient Boosting Machines (GBM) are robust for handling complex, non-linear data with fewer feature engineering requirements【19】【20】. As highlighted by Friedman (2001), careful tuning of model parameters, such as learning rate and tree depth in GBM, is essential to optimize performance.

4.2.3 Ethical Considerations in Predictive Modeling

Ethical considerations play a crucial role in predictive modeling, particularly in ensuring fairness, transparency, and accountability. Bias in data or algorithms can lead to discriminatory outcomes, which is especially concerning in sensitive applications like housing price predictions. As noted by Fairlearn (2021), it is vital to assess and mitigate fairness-related harms in predictive models.

Moreover, transparency in how models make predictions is necessary to build trust among stakeholders and users of these models.

4.2.4 Future Directions and Recommendations

Looking forward, several areas offer potential for advancing the state of predictive modeling in real estate. Future research could explore the integration of more sophisticated machine learning techniques, such as deep learning, to capture complex patterns in large-scale housing datasets【9】【10】. Additionally, the use of alternative data sources, like social media or IoT data, could provide richer insights into housing market dynamics. Finally, ongoing efforts to improve model interpretability and fairness will be crucial in ensuring that these predictive models are both effective and ethically sound【23】【24】.

5 Section five : References

1. Harrison, D., & Rubinfeld, D. L. (1978). Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5, 81-102.

2. Statology. K-Fold Cross Validation in Python (Step-by-Step). Available at: https://www.statology.org/k-fold-cross-validation-in-python/.

3. MachineLearningMastery.com. A Gentle Introduction to k-fold Cross-Validation. Available at: https://machinelearningmastery.com/k-fold-cross-validation/ .

4. MachineLearningMastery.com. Repeated k-Fold Cross-Validation for Model Evaluation in Python. Available at: https://machinelearningmastery.com/repeated-k-fold-cross-validati on/.

5. Principal Component Analysis for Dimensionality Reduction in Python - MachineLearningMastery.com .: https://machinelearningmastery.com/principal-components-analysis-for-dimensionality-reduction-in-python/.

6. Principal Component Analysis (PCA) Overview - Scikit-learn Documentation. https://scikit-learn.org/stable/modules/decomposition.html#pca.

7. Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics, 29(5), 1189-1232. https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-a-gradient-boosting-machine/10.1214/aos/1013203451.full.

8. Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297: https://link.springer.com/article/10.1007/BF00994018.

9. Hasan, R., & Mahmood, S. (2024). Advanced Machine Learning Techniques for Predictive Modeling of Property Prices. *MDPI Information*. Available at: https://www.mdpi.com/1999-5903/15/6/295/

10. Jain, S., et al. (2021). Real Estate Market Prediction Using Deep Learning Models. *Annals of Data Science*. Available at: https://link.springer.com/article/10.1007/s11704-021-00078-2
11. Predictive Analytics In Real Estate. Available at: https://datasemantics.co/predictive-analytics-in-real-estate/
12. Challenges of Using Predictive Analytics in Real Estate. Available at: https://www.itransition.com/predictive-analytics/real-estate
13. Data Quality Issues in Predictive Modeling. Available at: https://link.springer.com/article/10.1007/s11356-021-14593-z
14. Chou, J.-S., Ngo, N.-T., & Chong, W. K. (2017). Comparison of machine learning models to provide preliminary forecasts of real estate prices. Journal of Housing and the Built Environment. Available at: https://asu.elsevierpure.com/en/publications/the-use-of-artificial-intelligence-combiners-for-modeling-steel-p
15. Fan, C., Cui, Z., & Zhong, X. (2018). House prices prediction with machine learning algorithms. In ICMLC 2018: Proceedings of the 2018 10th International Conference on Machine Learning and Computing. Available at: https://dl.acm.org/doi/10.1145/3195106.3195133
16. Effective House Price Prediction Using Machine Learning | SpringerLink. Available at: https://link.springer.com/chapter/10.1007/978-3-031-37164-6_32
17. Normalization and outlier removal in class center-based firefly algorithm for missing value imputation. Available at: https://journalofbigdata.springeropen.com/
18. Data Quality Evaluation, Outlier Detection and Missing Data Imputation Methods for IoT in Smart Cities. Available at: https://link.springer.com/chapter/10.1007/978-3-030-72065-0_1
19. Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, *7*, 21. Available at: https://doi.org/10.3389/fnbot.2013.00021/ .
20. Louppe, G. (2014). Understanding Random Forests: From Theory to Practice. Available at: http://arxiv.org/abs/1407.7502 /
21. Revisiting the Boston Housing Dataset — Fairlearn 0.11.0.dev0 documentation. Available at: https://fairlearn.org/revisiting-the-boston-housing-dataset/
22. Fairness-related harms assessment using the Boston Housing Dataset, Fairlearn Documentation. Available at: https://fairlearn.org/main/user_guide/fairness_assessment/housing.html.
23. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Springer. https://archive.org/details/an-introduction-to-statistical-learning_202202/page/n2/mode/1up
24. MIT OpenCourseWare. "Machine Learning with Python," Electrical Engineering and Computer Science. Available at https://ocw.mit.edu/courses/6-036-introduction-to-machine-learning-fall-2020/ .
25. DEV Community. "Cross-Validation and Hyperparameter Search in Scikit-Learn - A Complete Guide." Available at: https://dev.to/alexandrudanpop/cross-validation-and-hyperparameter-search-in-scikit-learn-a-complete-guide-31bh:contentReference[oaicite:0]{index=0}.
26. Scikit-Learn Documentation. "Grid Search for Hyperparameter Search." Available at: https://scikit-learn.org/stable/modules/grid_search.html:contentReference[oaicite:1]{index=1}.
27. Medium. "Hyperparameter Tuning and Cross Validation Using Grid Search and Randomized Search." Available at: https://medium.com/@contactsunny/hyperparameter-tuning-and-cross-validation-using-grid-search-and-randomized-search-2aee4a55ff6b:contentReference[oaicite:2]{index=2}.
28. Towards Data Science. "A Comprehensive Guide to Grid Search and Cross-Validation." Available at: https://towardsdatascience.com/a-comprehensive-guide-to-grid-search-cv-ecb609a8219e:contentReference[oaicite:3]{index=3}.
29. De Cock, D. (2011). "Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project." Journal of Statistics Education, 19(3).
30. Kaggle (n.d.). "Ames Housing Dataset." Available at: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

31. Touw, W. G., Bayjanov, J. R., Overmars, L., Backus, L., Boekhorst, J., Wels, M., & van Hijum, S. A. (2013). "Data mining in the life sciences with random forest: a walk in the park or lost in the jungle?" Briefings in Bioinformatics, 14(3), 315-326. https://doi.org/10.1093/bib/bbs034
32. Brownlee, J. (2018). "How to Develop a Machine Learning Model for the Ames Housing Dataset." Machine Learning Mastery. Available at: https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/
33. Roberts, L. (2019). "Ames Housing: Exploring Feature Engineering and Predictive Modeling for Housing Price Prediction." Towards Data Science. Available at: https://towardsdatascience.com/ames-housing-exploring-feature-engineering-and-predictive-modeling-fca4f4cd1230
34. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825-2830.
35. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. https://link.springer.com/book/10.1007/978-0-387-84858-7.
36. Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). Introduction to Linear Regression Analysis. Wiley.
37. Guyon, I., Weston, J., Barnhill , S., & Vapnik, V. (2002). "Gene selection for cancer classification using support vector machines." Machine Learning, 46(1-3), 389-422.
38. Kohavi, R., & John, G. H. (1997). "Wrappers for feature subset selection." Artificial Intelligence, 97(1-2), 273-324.
39. Jolliffe, I. T. (2002). Principal Component Analysis. Springer.
40. Wold, S., Esbensen, K., & Geladi, P. (1987). "Principal component analysis." Chemometrics and Intelligent Laboratory Systems, 2(1-3), 37-52.