Bulgarian Diploma Thesis

User Car Predictor with Machine Learning

Aleksandar Ivanov, ID#200231727

Student: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,

Date: \_\_\_\_\_\_\_\_\_\_\_

Supervisor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,

Date: \_\_\_\_\_\_\_\_\_\_\_

Department of Computer Science AUBG

Blagoevgrad 2025

Table of Contents

[1. Introduction 3](#_Toc211705323)

[1.1 Motivation 3](#_Toc211705324)

[1.2 Project Overview 4](#_Toc211705325)

[2. Specification of the Software Requirements and their Analysis 5](#_Toc211705326)

[2.1 Functional requirements 5](#_Toc211705327)

[2.2. Non-functional Requirements 7](#_Toc211705328)

[2.3. Requirements Analysis 7](#_Toc211705329)

[2.3.1. Analysis of Functional Requriements 7](#_Toc211705330)

[2.3.2 Analysis of Non-functional Requirements 12](#_Toc211705331)

[3. Design of Software Solution 12](#_Toc211705332)

[3.1. Main algorithms 12](#_Toc211705333)

[3.1.1. Linear regression 12](#_Toc211705334)

[3.1.2. Ridge Regression 13](#_Toc211705335)

[3.1.3. Random Forest Regression 14](#_Toc211705336)

[3.1.4. Gradient Boosting Regression 15](#_Toc211705337)

[3.2. Algorithm Complexity 16](#_Toc211705338)

[3.2.1. Linear Regression Complexity 16](#_Toc211705339)

[3.2.2. Ridge Regression Complexity 16](#_Toc211705340)

[3.2.3. Random Forest Complexity 17](#_Toc211705341)

[3.2.4. Gradient Boosting Complexity 17](#_Toc211705342)

[3.3. Software Architecture 18](#_Toc211705343)

[References 18](#_Toc211705344)

# 1. Introduction

## 1.1 Motivation

One of the fastest-moving parts of the global automotive industry is the used car market. Each year, millions of cars are sold and bought in the second-hand market and their prices are influenced by a lot of factors like brand, model, mileage, year of production, engine, transmission and fuel type (gasoline, diesel, hybrid, electric). For buyers who are not aware of the details in the pricing of vehicles it can lead to overpaying. On the other side, for sellers, inaccurate research on how they should price their vehicle may lead to losses or overpricing it which will cause delays in selling.

Usually, the valuation of vehicles is handled manually or by online platforms that rely on historical average (Amik et al.). However, these methods are not the most optimal as of today. Expert-based evaluations can be subjective, time-consuming and difficult to scale across large markets (Alnajim et al.). Using online platforms, while they are automated way to predict prices, they often fail to make the right prediction (Pal et al.; Amik et al.). These inaccuracies can cause a huge financial damage to the sellers because the market is very competitive.

Machine learning (ML) gives us a new and improved way to predict prices for vehicles. By using large datasets from previously sold vehicles, ML can find patterns. ML can find relations in data that is non-linear like such as brand, model, mileage, year of production which will lead to calculations which are more precise. Using supervised algorithms such as linear regression, gradient boosting, random forest accuracy is significantly improved compared to manual and rule-based techniques (Amik et al.; Alnajim et al.; Pal et al.).

The motivation behind this project is to make a working ML-powered web application that can give correct predictions to prices on used vehicles given their key attributes. Both buyers and sellers can benefit from such system because it will introduce a lot more transparency and reducing the risk of overvaluation or undervaluation. It will make dealerships more competitive as they are now going to be able to adapt to the price changes much quicker. This project can also be of use of academic purpose as it shows how ML techniques can be applied to help a real-world business problem. In this way, the project how data-centric approaches can be used to modernize an industry which relies on subjective judgement.

## 1.2 Project Overview

The project is a data-driven web application for predicting used-car prices. The backend is implemented in ASP.NET Core (C#) and exposes RESTful controllers; the frontend is a React + TypeScript single-page application that calls these endpoints. The system trains and serves four algorithms: Linear Regression, Ridge Regression, Random Forest, and Gradient Boosting.

Preprocessing is handled by dedicated components: a FeatureScaler for numeric inputs and a LabelScaler for target prices. The LabelScaler inverts predictions back to the original currency scale. These components are reusable across training and inference and are serialized with the trained models so the web app uses the exact same pipeline used during training.

In its core the application integrates several machine learning algorithms, including Linear Regression, Gradient Boosting and Random Forest. These algorithms will be trained on publicly available used-cars datasets, which contain the following information: brand, model, year of production, transmission, engine type and fuel type. The system also integrates preprocessing components like a FeatureScaler and LabelScaler, which were implemented specifically for this project. The *FeatureScaler* is responsible for normalizing the numeric input features (e.g., mileage, production year) into a standardized range before training. The *LabelScaler* performs the inverse operation by rescaling the model’s predicted values back to the original monetary scale. These components form part of the backend preprocessing pipeline and are integrated into the application’s overall architecture as independent, reusable modules.

The project has two main goals. On the practical side, it is a tool which can be used by dealerships/sellers and buyers to make the process of buying a used car more transparent and aligned with the current market. And on the academical side it can be used as a case-study for what machine learning is most suitable by enabling direct comparison between the models in terms of their accuracy and computational cost and time cost.

In addition, the project aims to provide the end user the full pipeline not only a command line interface app by implementing user-friendly web experience. Which will demonstrate how each of the algorithms perform under the same conditions and the results will be shown to the end-user in a way that a non-expert can use them. By combining everything mentioned the project will provide a robust solution for used car predictions.

# 2. Specification of the Software Requirements and their Analysis

## 2.1 Functional requirements

* Algorithms: Train and serve predictions from Linear Regression, Ridge, Random Forest, and Gradient Boosting.
* Single-car query: User selects odometer, year, manufacturer, model, fuel, transmission, and planned purchase year; the app returns a predicted price from each algorithm.
* Multi-year projection: For a configured car, the app produces a year-by-year projection for a user-selected horizon and renders a line chart.
* Batch predictions: Users can upload a CSV to obtain per-row predictions from all algorithms.
* Data preprocessing pipeline:
  + handles missing values (impute/drop with a summary report),
  + **Outlier handling.** Outliers are detected **per numeric feature** using **Tukey’s IQR rule** on the **training split** only.
  + encodes categorical features,
  + scales numeric features where needed (z-score for linear models; trees receive the same columns),
  + enforces a rare-category policy to remove categories with insufficient data representation in the training set, ensuring model consistency across all dataset splits.
* Dataset ingestion: Load historical listings from CSV with columns: price, odometer, year, manufacturer, model, fuel, transmission.
* Training & validation: Support k-fold cross-validation and hold-out evaluation (default 80/20). Fit all preprocessing on the training split only and apply on the validation/test to avoid leakage.
* Evaluation: Report MAE, RMSE, and R² for each algorithm; show a compact metrics table and predicted-vs-actual visuals in the UI.
* Web UI: React SPA that calls ASP.NET Core controllers to obtain predictions and render charts/tables.
* Model comaprison UI:
  + User view: for a configured vehicle, show each algorithm’s predicted price and (if the user specifies a horizon *n* years) render a multi-year projection chart of predicted prices for the next *n* years. Allow overlaying projections from multiple algorithms, toggling prediction intervals, and exporting the chart.
  + Evaluation view (internal/reporting): show diagnostic plots (predicted vs. held-out targets and residuals) and a metrics table with MAE, RMSE, R² reported as mean ± stdev across cross-validation folds.
* **Two-vehicle comparison UI (same algorithm):** Enable a **comparison mode** where the user selects **one algorithm** and **two vehicle configurations** (price, odometer, year, manufacturer, model, fuel, transmission, purchase year/horizon n). The app renders an **overlay chart** of the two **multi-year price projections** (same horizon, same y-axis), with a legend and tooltips, and provides a **diff table** (per-year Δ and average Δ over the horizon).

## 2.2. Non-functional Requirements

1. The application should be configurable. It should allow selection, enabling users to select attributes such as fuel type, transmission, and purchase year.
2. The system should achieve a level of accuracy, with at least one algorithm reaching an 𝑅² ≥ 0.7 on validation data.
3. The web interface should be intuitive and easy to use for non-expert users, presenting predictions and comparison charts in a clear and accessible way.
4. The web application should be deployable on common operating systems (Windows, macOS, Linux) and run in modern browsers without additional plugins.

## 2.3. Requirements Analysis

### 2.3.1. Analysis of Functional Requriements

The core of the application lies in its machine learning algorithms. These were selected to provide a balance between interpretability, accuracy, and computational efficiency.

Linear Regression was included because of its simplicity and transparency. It provides an interpretable baseline against which more advanced models can be compared. However, linear regression is often insufficient for capturing the nonlinear relationships that exist in car price data, such as the interaction between mileage and production year (J. I. Kang et al.).

Ridge Regression extends linear regression by introducing regularization, which penalizes overly complex models and reduces the risk of overfitting. This makes them more stable for datasets that include correlated variables, a common situation in real-world car listings (Ting Tin Tin et al.).

Random Forest Regression was chosen due to its ability to handle nonlinear interactions and its robustness to noisy features. Research consistently reports that Random Forest achieves strong performance in car price prediction tasks (J. I. Kang et al.; Gegic et al.).

Gradient Boosting was selected because it incrementally corrects errors of weaker learners, often producing the most accurate results. While it requires more computation and careful tuning, boosting methods are reported to achieve state-of-the-art performance in car price prediction tasks (J. I. Kang et al.).

To demonstrate the implementation requirements, the algorithms can be summarized in pseudocode:

1. procedure TrainUsedCarModels(dataset, anchorYear, maxConfigs, specificModel)

2. vehicles ← LoadCSV(dataset)

3. groupedModels ← Group vehicles by normalized Model

4.

5. if specificModel is given then

6. modelsToTrain ← {specificModel}

7. else

8. modelsToTrain ← all models with ≥ 50 samples

9. end if

10.

11. for each modelName in modelsToTrain do

12. rows ← Filter vehicles where Model = modelName

13. (X\_raw, y\_raw, fuelTypes, transmissions) ← Preprocess(rows, anchorYear)

14.

15. # Split into training and test subsets

16. (X\_trainRaw, y\_trainRaw, X\_testRaw, y\_testRaw) ← Split(X\_raw, y\_raw, ratio = 0.8)

17.

18. # Standardize features and labels

19. featureScaler ← Fit FeatureScaler on X\_trainRaw

20. labelScaler ← Fit LabelScaler (log-space) on y\_trainRaw

21.

22. X\_train ← Transform(featureScaler, X\_trainRaw)

23. X\_test ← Transform(featureScaler, X\_testRaw)

24. y\_train ← Transform(labelScaler, y\_trainRaw)

25. y\_test ← Transform(labelScaler, y\_testRaw)

26.

27. # 1. Linear Regression baseline

28. linear ← Fit LinearRegression on (X\_train, y\_train)

29. metrics\_linear ← Evaluate(linear, X\_test, y\_test, labelScaler)

30.

31. # 2. Ridge Regression

32. ridge ← TrainWithBestParamsKFold(X\_train, y\_train, labelScaler, folds = 5)

33. metrics\_ridge ← Evaluate(ridge, X\_test, y\_test, labelScaler)

34.

35. (X\_train1, y\_train1, X\_val1, y\_val1) ← Split(X\_train, y\_train, ratio = 0.75)

36. ridge\_train\_pred ← Predict(ridge, X\_train1)

37. ridge\_val\_pred ← Predict(ridge, X\_val1)

38. trainResiduals ← y\_train1 - ridge\_train\_pred

39. valResiduals ← y\_val1 - ridge\_val\_pred

40.

41. # 3. Random Forest

42. rf ← TrainResidualsWithBestParams(X\_train1, trainResiduals,

43. X\_val1, valResiduals,

44. maxConfigs)

45. metrics\_ridge\_rf ← EvaluateCombined(ridge, rf, X\_test, y\_test, labelScaler)

46.

47. # 4. Gradient Boosting

48. gb ← TrainResidualsWithBestParams(X\_train1, trainResiduals,

49. X\_val1, valResiduals,

50. maxConfigs)

51. metrics\_ridge\_gb ← EvaluateCombined(ridge, gb, X\_test, y\_test, labelScaler)

52.

53. SaveModelBundle({

54. "Linear": linear,

55. "Ridge": ridge,

56. "RandomForest": rf,

57. "GradientBoosting": gb,

58. "FeatureScaler": featureScaler,

59. "LabelScaler": labelScaler,

60. "Metrics": {

61. linear, ridge, ridge\_rf, ridge\_gb

62. },

63. "TrainingTimes": recorded durations,

64. "Meta": { anchorYear, rows count, manufacturer, modelName }

65. })

66. end for

67. end procedure

68.

69.

70. procedure Evaluate(model, X\_test, y\_test, scaler)

71. predictions ← InverseTransform(scaler, Predict(model, X\_test))

72. truth ← InverseTransform(scaler, y\_test)

73. return { MAE, RMSE, R2 } computed between predictions and truth

74. end procedure

75.

76. procedure EvaluateCombined(baseModel, residualModel, X\_test, y\_test, scaler)

77. basePred ← Predict(baseModel, X\_test)

78. resPred ← Predict(residualModel, X\_test)

79. combined ← basePred + resPred

80. predictions ← InverseTransform(scaler, combined)

81. truth ← InverseTransform(scaler, y\_test)

82. return { MAE, RMSE, R2 } computed between predictions and truth

83. end procedure

84.

Another essential requirement is the data preprocessing pipeline. Raw vehicle listings often contain missing values, outliers, and inconsistent formats. Without cleaning, models may learn spurious relationships or perform poorly on unseen data. To address this, the pipeline will include:

* Handling missing values (forward fill or row removal when thresholds are exceeded).
* Detecting and removing outliers using statistical measures such as the z-score.
* Encoding categorical variables (e.g., transmission, fuel type).
* Scaling numerical features for models sensitive to feature magnitude.

These preprocessing steps are consistently identified as key to improving generalization and stability in predictive modeling (Gegic et al.; Haque et al.).The process is sumarized in Figure 1:

A diagram of a data processing process

AI-generated content may be incorrect.

*Figure 1. Preprocessing and processing activity diagrams.*

A well-designed user interface is what distinguishes a good software package from an average one. A simple, intuitive, and comprehensive dashboard drastically improves user experience and accessibility. The dashboard in this application is designed to hide the complexities of the underlying machine learning algorithms, allowing the user to obtain predictions and comparisons without needing to understand the technical implementation details.

The interface will provide:

* Main navigation menu providing access to four prediction-related views, allowing users to switch easily between modes.
* Input configuration for vehicle details such as manufacturer, model, year of production, mileage, fuel type, transmission, and target year.
* Simultaneous algorithm evaluation, where all implemented models (Linear, Ridge, Random Forest, Gradient Boosting) are executed automatically for a given input.
* Real-time results display, showing the predicted price for each algorithm without requiring additional user tuning.
* Simplified workflow, focusing on single-car predictions rather than batch or manual hyperparameter selection.
* Visual feedback, including charts and tables summarizing predicted values, trends across years, and evaluation metrics (MAE, RMSE, R²).
* User-friendly design, emphasizing clarity, minimal configuration, and fast comparison between algorithms’ outputs.

These elements transform the project into a valuable tool for both practitioners and students, as it enables experimentation and fosters understanding of supervised learning algorithms (J. I. Kang et al.; Ting Tin Tin et al.).

A diagram of a process

AI-generated content may be incorrect.

*Figure 2. Use case diagram for the user dashboard.*

The workflow of the system consists of – preprocessing, training, evaluation(prediction) and visualization. The application has preprocesedthe data by handling missing values and outliers. Each algorithms predictions are visualized through charts and error metrics. Finally, the system provides facilities to compare models under identical conditions.

This sequence ensures reproducibility and transparency while aligning with established practices in machine learning pipelines (Haque et al.).

A diagram of a process

AI-generated content may be incorrect.

*Figure 3. Activity diagram for the system workflow.*

Finally, the performance of the regression algorithms isevaluated using metrics. The most common in regression tasks are the coefficient of determination (R²), the mean absolute error (MAE), and the root mean squared error (RMSE). The R² value reflects how predictions fit real data, with values near 1 indicating a strong fit. RMSE provides an error measure but is sensitive to outliers. And MAE is more robust to extreme values. Together, these three metrics provide a balanced assessment of the performance of each algorithm.

Based on prior research in car price prediction, ensemble models such as Random Forest and Gradient Boosting are expected to achieve the highest R² values, typically above 0.8 on large datasets (J. I. Kang et al.; Gegic et al.). Linear Regression and Ridge regression generally produce lower scores, often around 0.6–0.7 (Ting Tin Tin et al.). These score results are not guaranteed because the actual performance depends on dataset quality, preprocessing, and parameter tuning, but they provide reasonable targets for evaluation.

Example Use Case Narratives

i. Name: Single Prediction

* Actor: User
* Entry Condition: The application is running and pre-trained models are loaded.
* Main Flow:
  1. From the Home screen, the user selects Single Prediction.
  2. The user specifies the car’s make, model, year, mileage, fuel type, and transmission.
  3. The system applies all available algorithms (Linear, Ridge, Random Forest, Gradient Boosting) to compute the predicted price.
  4. The result screen displays numeric predictions along with performance metrics.
* Result Screen: Shows predicted prices from each algorithm, along with MAE, MSE, R², and training/inference timings.
* Exit Condition: The user clicks Back Home to return to the main menu.
* Exceptions: If required attributes are missing, the system prompts the user to complete the input form.

ii. Name: Single Ranged Prediction

* Actor: User
* Entry Condition: The application is running and pre-trained models are loaded.
* Main Flow:
  1. From Home, the user selects Single Ranged Prediction.
  2. The user provides a fixed configuration for the car and defines a range of production years.
  3. The system applies all algorithms to generate predicted prices for each year in the range.
  4. The result screen visualizes how price changes over time.
* Result Screen: A multi-line chart (year vs. predicted price), with one line per algorithm, and a table containing MAE, MSE, R², and timing data.
* Exit Condition: The user clicks Back Home after viewing results.
* Exceptions: Invalid year ranges (e.g., min ≥ max) prompt the user to correct inputs.

iii. Name: Compare Predictions (Two Cars)

* Actor: User
* Entry Condition: The application is running and pre-trained models are loaded.
* Main Flow:
  1. From Home, the user selects Compare Predictions.
  2. The user selects two different car configurations (make, model, and attributes).
  3. The system applies all algorithms to both cars and computes their predicted prices.
  4. The result screen presents the comparison.
* Result Screen: A bar chart showing predicted prices for each algorithm per car, with MAE, MSE, R², and timing metrics.
* Exit Condition: The user clicks Back Home to return to the main screen.
* Exceptions: If fewer than two valid car configurations are provided, the system displays a prompt to complete both selections.

iv. Name: Compare Ranged Predictions

* Actor: User
* Entry Condition: The application is running and pre-trained models are loaded.
* Main Flow:
  1. From Home, the user selects Compare Ranged Predictions.
  2. The user selects two cars and then chooses one algorithm for comparison.
  3. A range of years is defined.
  4. The system generates predictions for both cars over the year range using the selected algorithm.
  5. The result screen displays the trends.
* Result Screen: A two-line chart (year vs. predicted price) comparing both cars using the chosen algorithm, with corresponding MAE, MSE, R², and timing information.
* Exit Condition: The user clicks Back Home to end the session.
* Exceptions: If no algorithm or invalid year range is selected, the system displays a warning.

Global Notes

* Every result screen includes:
  + Predicted price(s)
  + MAE, MSE, R², and timing metrics
  + Appropriate chart (line or bar) for visual comparison
* Each use case starts from Home and ends with a Back Home button.
* The system ensures consistency and reproducibility across all prediction modes.

### 2.3.2. Analysis of Non-functional Requirements

The scalability and configurability requirements are defined to ensure that the application is efficient and adaptable. The modular structure of the backend is developed using ASP.NET Core. This allows for independent training, and prediction components. This also guarantees scalability, such as adding new algorithms or modifying the data pipeline.

Configurability is achieved by command-line arguments that control parameters like anchor year, dataset size, and search configurations. This design gives the users ability to tune training duration to match their specific need.

**Usability** is made with the React-based user interface, which provides four c prediction modes—Single, Single Ranged, Compare, and Compare Ranged. Each mode offers user friendly UI with fast response times, visualizations with bar and line charts and performance metrics.

Finally, the system’s responsiveness and maintainability are ensured by the use of pre-trained models and a compliance to the MVC architecture, which ensures clean code and scalability.

# 3. Design of Software Solution

## 3.1. Main algorithms

### 3.1.1. Linear regression

Linear Regression was chosen because of its simplicity and transparency. It provides an interpretable baseline against which more advanced models can be compared. However, linear regression is often insufficient for capturing the nonlinear relationships that exist in car price data, such as the interaction between mileage and production year (J. I. Kang et al.).

However, the strict assumption of linear relationships is often unrealistic in the used car domain. Real-world depreciation curves tend to be nonlinear: for instance, mileage might reduce value steeply initially, but the marginal effect slows as the car ages. Some researchers have proposed S-curve or logistic models precisely because linear fits lead to increasing errors at extreme values (Salim and Abu).

Moreover, linear regression is vulnerable to outliers. In used car datasets, extreme prices or rare models can disproportionately skew the regression line, reducing robustness (Amik et al.). Also, when features like age and mileage are highly correlated, coefficient estimates can become unstable or misleading (Ting Tin Tin et al.).

Despite these limitations, linear regression is still very valuable as a baseline model. In comparative studies, it often achieves very good error rates around 80–85% before more advanced algorithms are applied. Its speed, ease of training, and minimal hyperparameter tuning make it useful especially when rapid prototyping.

### 3.1.2. Ridge Regression

Ridge Regression is a regularized linear model that balances interpretability and predictive performance by introducing a penalty term to reduce overfitting. Specifically, it applies L2 regularization, which adds a squared penalty to the magnitude of coefficients. This discourages large weights and stabilizes estimates when predictors are highly correlated, effectively reducing variance at the cost of a small bias.

Recent studies have shown that Ridge Regression remains a competitive method for price prediction tasks, offering a reliable balance between accuracy and stability (Zhang and Politis). It performs particularly well when most predictors contribute moderately to the outcome, making it well suited for datasets where relationships between features and price are distributed across multiple variables.

In used car price prediction, Ridge Regression has demonstrated greater consistency than standard linear regression, particularly in the presence of multicollinearity and noisy features (Bergmann and Feuerriegel). Despite its limitation of retaining all predictors, Ridge offers computational efficiency and improved generalization, positioning it as a strong intermediate model between basic linear methods and more complex machine learning approaches.

### 3.1.3. Random Forest Regression

Random Forest Regression builds a large number of decision trees, each trained on a different sample of the data, and then combines their outputs to generate consistent and accurate predictions (Breiman). This approach reduces overfitting, which is a problem in single decision trees. By averaging the results of many trees, Random Forest captures complex patterns and interactions between features that models like linear regression, often cannot see.

In used car price prediction, Random Forest has proven to be one of the most effective and practical algorithms. It performs well because it can naturally model the nonlinear relationships between features such as mileage, age, brand, and engine size without requiring heavy preprocessing or feature transformation (Pal et al.; Bergmann and Feuerriegel). Research has shown that Random Forest models consistently outperform basic regression techniques in accuracy and robustness across various vehicle datasets (Pal et al.).

A key advantage of Random Forest is its ability to remain stable and reliable even when data contains noise or outliers. Since predictions are averaged across many independent trees, individual anomalies—such as mispriced listings—have a limited effect on the final prediction (Mentch and Zhou). Another important feature is the algorithm’s ability to estimate feature importance, which helps determine which variables most strongly influence the predicted price. Studies commonly find that mileage and model year have the highest importance scores, while other attributes like fuel type or number of doors play smaller roles (Molnar).

However, Random Forest also has limitations. It requires more computational resources than simpler algorithms, and its internal logic is less interpretable because predictions are produced by averaging hundreds of trees. When the dataset contains many weak or redundant features, its performance can also decrease slightly unless preprocessing and feature selection are applied carefully (Scornet). Despite these drawbacks, Random Forest remains a reliable and versatile model for this project. It offers an excellent balance between accuracy, robustness, and practicality, making it one of the core algorithms.

### 3.1.4. Gradient Boosting Regression

Gradient Boosting Regression is a machine learning algorithm that builds models sequentially. Each new model focuses on correcting the errors made by the previous ones. It can be epxlained with an golf example. Where a person shoots, makes a mistake analyses their mistake and shoots again in an attempt to make a better shot.

It optimizes a loss function by fitting to the residuals of prior predictions, which allows it to improve performance over time (Friedman). This process enables Gradient Boosting to learn nonlinear relationships and feature interactions which is an advantage that is especially useful in car price prediction, where variables such as mileage, age, etc.

In practice, one of the most widely used implementations of Gradient Boosting is Extreme Gradient Boosting (XGBoost), known for its computational efficiency, regularization, and scalability. Research applying XGBoost to vehicle pricing has demonstrated high predictive accuracy when paired with proper preprocessing and parameter tuning (Qian). Similarly, Gradient Boosting is one of the best performing algorithms for estimating used car resale values across diverse datasets (Bergmann and Feuerriegel).

A major strength of Gradient Boosting is the control over bias and variance through parameters such as the learning rate, tree depth, and number of estimators. Each tree in the sequence attempts to reduce the residual errors of the previous ones, enabling the model to capture data patterns that simpler algorithms often overlook. However, this flexibility also makes the algorithm more prone to overfitting if parameters are not carefully tuned. Multiple studies highlight the importance of hyperparameter optimization and early stopping to prevent overfitting and ensure stable results (Zheng).

Gradient Boosting is also useful for interpretability tools such as feature importance. This allow users to understand how individual features influence price predictions. When compared to Random Forest, Gradient Boosting requires more resources and a more complex training process. In cases where datasets are small or contain high levels of noise, simpler models can sometimes achieve comparable results with less tuning.

## 3.2. Algorithm Complexity

The algorithms chosen for this senior project were chosen for their accuracy when it comes to used car prediction systems and also their efficiency which is proven by the researchers.

### 3.2.1. Linear Regression Complexity

Linear Regression serves as the base model due to its analytical simplicity and low cost for computation. Training a linear model on a dataset with samples and features requires solving the normal equation . Which has a time complexity of . But we need to compute the matrix inverse which can be both computationally expensive and unstable for large feature matrices.

To overcome this, QR decomposition is used to solve the problem more efficiently. What QR decomposition does is, it factorizes the feature matrix into an orthogonal matrix and an upper triangular matrix such that . Substituting this into the normal equation gives . This approach avoids matrix inversion and improves both stability and efficiency (Amik et al.; Salim and Abu). The decomposition itself has a computational complexity of

### 3.2.2. Ridge Regression Complexity

Ridge Regression extends the linear regression model by introducing an regularization. Which is used to penalize large coefficients, reducing overfitting and improving stability in the presence of multicollinearity (Zhang and Politis; Ting Tin Tin et al.). Ridge Regression uses the same mathematical foundation as the linear regression algorithm, but it adds : , with having the role of regularization parameter. The inclusion of this penalty term adds minimal computational overhead, and its complexity remains .

### 3.2.3. Random Forest Complexity

**Random Forest Regression** has strong predictive performance which is achieved by averaging the outputs of multiple decision trees. The training complexity of a single decision tree is . For a group with trees, the total training cost is (Scornet; Mentch and Zhou). But the advantage of Random Forests is that they can be trained in parallel, which has a big benefit when it comes to reducing runtime and efficiency with complexity of .

### 3.2.4. Gradient Boosting Complexity

**Gradient Boosting Regression** enhances predictive accuracy by building trees sequentially, with each new tree errors are corrected in an attempt to make a better prediction (Friedman). The training complexity of Gradient Boosting is , where represents the number of boosting iterations. While this makes boosting models more computationally demanding comapred to Random Forests, they generally achieve higher accuracy when tuned appropriately. Research has demonstrated that boosting techniques, particularly XGBoost(which is a more complex expansion on gradient boosting), outperform traditional models in capturing nonlinear relationships between vehicle attributes and price(Qian; Zheng; Bergmann and Feuerriegel).

## 3.3. Software Architecture

The application is developed using the **Model-View-Controller (MVC)** pattern. This structure ensures separation of the responsibilities of data management, user interface, and central logic. The backend, developed with **ASP.NET Core**, which implements the **Model** and **Controller** components from the MVC pattern. And the **View part of the pattern** is handled by a **React and TypeScript in a** single-page application (SPA).

In this architecture, the **Model** is responsble for the machine learning and data processing logic which responsible for training, evaluating, and serving predictions for used car prices. And it implements the following algorithms - Linear Regression, Ridge Regression, Random Forest, and Gradient Boosting,  including the residual-learning combination (Ridge + RandomForest/GradientBoosting) - along with the data preprocessing pipeline that handles tasks such as feature encoding, scaling, and dataset validation.

The **Controller** acts as the middleman between the UI and the backend. Its main functionallity is to expose RESTful endpoints which establish the communication between the frontend and the backend. Each controller receives input data from the user, invokes the corresponding code and returns structured JSON responses containing what was requested.

The **View** component is implemented as a React-based web interface. It provides a user-friendly UI where users can and visualize prediction results, compare the performance of different algorithms and comapre different car prices throughout the years. The React application communicates asynchronously with the backend through HTTP requests.

Supporting modules are also added in addition to the main MVC components. They provide additional services such as feature scaling, label scaling, data validation, and model serialization. These utility classes maintain the consistency and reusability of the system across different operations. The interactions between all layers of the project are showcased in Figure 4:

A diagram of a software development process

AI-generated content may be incorrect.

*Figure 4. System Achitecture Diagram.*

The MVC design pattern was selected because it enables a clear separation between the frontend and the backend, which is essential for a web-based architecture that combines a React frontend and a .NET backend. Another way this can be developed is with monolithic designs but in this situation all UI elements and business logic coexists in the same codebase which makes everything much more couples. On the other hand, MVC ensures that the data processing and machine learning components remain fully independent from the UI. This separation facilitates parallel development, easier debugging, and improved scalability when deploying new algorithms or updating the frontend. The modular organization between the controller and model layers make MVC practical for applications that expose RESTful APIs and require consistent communication with an external UI.

### 3.3.1. Model(s)

There are four main algorithms - Linear Regression, Ridge Regression, Random Forest, and Gradient Boosting. They provide the core predictive functionality of the application. They form the backend and have been implemented according to object-oriented programming (OOP) principles.

At the highest level, an abstract class named Regressor. This class defines the common interface and functionality which is shared between all the models models. It includes methods for model training and prediction (Fit() and Predict()). Also there is a set of evaluation utilities for computing key performance metrics: coefficient of determination (), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

The LinearRegression class implements the core regression logic and serves as the base implementation for linear modeling. The RidgeRegression class inherits from *LinearRegression* and overrides the training logic by adding regularization to the weight updates. The DecisionTreeRegressor class serves as a reusable component for RandomForestRegressor and GradientBoostingRegressor.

The DecisionTreeRegressor class serves as a reusable component rather than a standalone predictive model. It defines the logic for recursive tree construction, which is reused by more complex ensemble algorithms. Both the RandomForestRegressor and GradientBoostingRegressor classes inherit from this component internally. Each of them inherits from *IRegressor* and uses multiple instances of *DecisionTreeRegressor* to accommodate their needs.

The implemented hierarchy ensures high modularity and reusability and scalability. All regression models share a common structure defined by *IRegressor*, while tree-based algorithms extend their functionality by combining multiple decision trees. With this approach there is code duplication and simplifies future expansion, as new regression types can be added by implementing or extending the same base interface.

The high-level class hierarchy, with the relevant inheritance relationships and shared interfaces, is illustrated in Figure 5.

A diagram of a computer program

AI-generated content may be incorrect.

*Figure 5. Regression Model Class Hierarchy.*

### 3.3.2. View(s)

The view layer represents the UI of the application and is implemented as a SPA using TypeScript with React. The UI provides an intuitive and minimal way for users to interact with the prediction system without requiring prior knowledge of machine learning concepts.

The entry point to the SPA is the Main Menu, which serves as the navigation hub for the application. From there, users have the choice to make a prediction in 4 different ways:

* **Single Prediction** – predicting the price of a single car for a specific year using all available algorithms.
* **Single Ranged Prediction** – visualizes the predicted price trend of a single car model across a selected year range using all algorithms including a line chart to visualise the trend.
* **Compare Models** – displays the performance of all algorithms using a bar chart and a table which contains the evaluation metrics (MAE, RMSE, and R²).
* **Compare Ranged Prediction** – compares two different car models across a range of years using a selected algorithm, providing line chart visualizations of price evolution.

The UI makes a asynchronous communication with the backend through HTTP requests which include the payload of the car that we want to make a prediction for. ThePrediction Controller receives the requests executes the prediction and it returns a structured JSON response to the UI. The results show dynamicly in the web browser with the use of React States and Charts. Also each view has a consistant structure with a “Go Back”, “Go Home” and “Next buttons”. This ensures that the UI is smooth troughtout all pages.

#### 3.3.2.1. Main Dashboard Overview

The **Main Dashboard** has 4 options as mentioned in the point above. Its design is kept minimal to ensure simplicity when using the platform. Each prediction view begins with a form where the user provides vehicle-specific attributes.

#### 3.3.2.2. Single Prediction View

The **Single Prediction** view allows users to obtain a price prediction for a specific car and get predictions based on all implemented algorithms — Linear Regression, Ridge Regression, Random Forest, and Gradient Boosting.

The user has to enter the following inputs to get a prediction:

* Manufacturer
* Model
* Year of Production
* Target Year
* Mileage (Odometer Reading)
* Transmission Type
* Fuel Type

After filling all inputs, the user makes a prediction request by clicking on the “Predict” button. The entered data is sent as a JSON payload to the backend through the Prediction Controller, which returns the results for each algorithm.

The output includes both price predictions and evaluation metrics (MAE, MSE, and R²), alongside timing information for each model. The results are presented in a structured list that updates dynamically using React states. The view includes navigation buttons for “Go Back”, “Go Home”, and “Next”, ensuring a smooth transition between different sections of the UI.

#### 3.3.2.3. Single Ranged Prediction View

The Single Ranged Prediction view adds more on top of the single prediction by giving users the option to observe how a car’s price is going to chnage across a selected year range. This view uses all algorithms to produce multiple predictions for each year within the time range.

The input attributes are the same as in the single prediction view with the only difference being the inclusion of a starting year and an ending year. Once submitted, the backend processes the range by predicting prices for every year within the specified year interval.

The results are visualized using a line chart, where one line shows each algorithms’ predictions for the specified interval. This way users can easily identify what prediction each algorithm has made for the inteval of years and also visualise the long-term depreciation of the vehicle. Below the chart, the corresponding evaluation metrics and timing data are displayed as in the single prediction view.

#### 3.3.2.4. Compare Models View

The Compare Models compares the overall performance of all four algorithms for two cars. The inputs are the same as in the single prediction view but the user is required to enter them twice bacause this view requires two vechicles.

On the results screen the user is presented with a bar chart that contrasts algorithms based on their price predictions. Under the chart, a metrics table displays the numerical results of all metrics and timings of each algorithm.

#### 3.3.2.5. Compare Ranged Prediction View

The Compare Ranged Prediction view enables a direct comparison between two car models over a interval of years. The inputs are the same as in the Ranged Prediction with the small change that the user has to select two different cars and also select a prefered algorithm.

The backend processes both inputs across the same year range and returns the predictions. The results are visualized using a dual-line chart, where each line represents the predictions of the algorithm for one car in the desired interval.

### 3.3.3. Controller(s)

The Controller(s) connects the user interface and the machine learning logic. The goal of the Controller(s) is to process user requests, execute the models which will produce the prediction. And at the end structure the predictions into a JSON and return it to the front end.

The main controller in this application is the PredictionController. It is responsible for handling all prediction. It is has exposed 4 endpoints under “.../api/v1/prediction”. Which are used accordingly to the view which is selected in the UI.

Each request sent to the PredictionController contains car-specific payload with the following inputs: manufacturer, model, year of production, mileage, transmission, fuel type, target year, start year(ranged requests only), end year(ranged requests only), algorithm type(ranged requests only). When the controller receives the data it makes a input validation and delegates the computation to the ActiveModel and ModelHotLoader services. They are responsible for loading the correct trained model bundle from the processed dataset folder. After that the appropriate algorithms are called — Linear Regression, Ridge Regression, Random Forest, and Gradient Boosting — to generate predictions and compute the performance metrics.

In the end when the results are computed, the controller structures them into a JSON response that is sent back to the frontend. And they are displayed through charts and tables. The PredictionController does not contain any computational logic by itself. It is more like a orchestrator for the communication between the backend components.

## 3.4. Primary Vehicle Dataset and Feature Selection

The system is designed to support different datasets but the initial implementation focuses on a single dataset containing used car listings collected from publicly available dataset. The dataset includes vehicles of various brands, models and it is sufficient for model training and evaluation. The data contains the following attributes manufacturer, model, year of production, mileage, fuel type, transmission, and price.

During preprocessing, categorical variables such as fuel type and transmission are encoded. Year and mileage which are numerica attributes are scaled for performance purposes. Missing and inconsistent records are removed, and outliers are filtered out to preserve data quality.

The target variable for prediction for all models is the price of the vehicle. The remaining features features serve as predictors which help the regressors learn about the relationship betweem the features and the price of the vehicle.

## 3.5. Custom Dataset Requirements

The initial implementation doesn’t have U for training but the backend fully supports retraining with custom datasets through the command-line interface (CLI). This makes the application very flexible as the user can provide the application their custom CSV and they can use it to train the models and get predictions based on it.

Custom datasets should be provided in CSV (comma-separated values) format. The file should include at least the following columns:

price, year, manufacturer, model, odometer, fuel, transmission

11900, 2017, hyundai, elantra, 45000, gas, automatic

The price, year and odometer columns must be numerical. Fuel and transmission columns requrie plain text (e.g., “manual”, “diesel”). Missing, null, or non-numeric entries in the numeric columns are automatically filtered during preprocessing to prevent training errors.

# 4. Implementation

# References

Alnajim, Thuraya, et al. “An Intelligent Vehicle Price Estimation Approach Using a Deep Neural Network Model.” *World Electric Vehicle Journal*, vol. 15, no. 8, 2024, https://doi.org/10.3390/wevj15080345.

Amik, Fahad R., et al. “Application of Machine Learning Techniques to Predict the Price of Pre-Owned Cars in Bangladesh.” *Information*, vol. 12, no. 12, 2021, https://doi.org/10.3390/info12120514.

Bergmann, Svenja, and Stefan Feuerriegel. “Machine Learning for Predicting Used Car Resale Prices Using Granular Vehicle Equipment Information.” *Expert Systems with Applications*, vol. 263, Mar. 2025, p. 125640, https://doi.org/10.1016/j.eswa.2024.125640.

Friedman, Jerome H. “Greedy Function Approximation: A Gradient Boosting Machine.” *Annals of Statistics*, vol. 29, no. 5, 2001, pp. 1189–232, https://doi.org/10.1214/aos/1013203451.

Gegic, Enis, et al. “Car Price Prediction Using Machine Learning Techniques.” *TEM Journal*, vol. 8, no. 1, Feb. 2019, pp. 113–18. Academic Search Ultimate, 135048125, *EBSCOhost*, https://doi.org/10.18421/TEM81-16.

Haque, Alimul, et al. “Implication of Different Data Split Ratio on the Performance of Model in Price Prediction of Used Vehicles Using Regression Analysis.” *Data & Metadata*, vol. 3, Jan. 2024, pp. 1–14. Academic Search Ultimate, 181529805, *EBSCOhost*, https://doi.org/10.56294/dm2024425.

J. I. Kang, et al. “Comparing Regression Models Predicting the Price of Used Cars in Big Data.” *2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, 2022, pp. 01–04, https://doi.org/10.1109/ICCE-Asia57006.2022.9954633.

Mentch, Lucas, and Siyu Zhou. “Randomization as Regularization: A Degrees of Freedom Explanation for Random Forest Success.” *Journal of Machine Learning Research*, vol. 21, no. 171, 2020, pp. 1–36.

Molnar, Christoph. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Edited by 2, Christoph Molnar, 2023, https://christophm.github.io/interpretable-ml-book/feature-importance.html. Feature Importance.

Pal, Nabarun, et al. *How Much Is My Car Worth? A Methodology for Predicting Used Cars Prices Using Random Forest*. arXiv preprint. 2017, https://arxiv.org/abs/1711.06970.

Qian, Tingyu. “Used Car Price Prediction by Using XGBoost.” *BCP Business & Management*, vol. 44, Apr. 2023, pp. 62–68, https://doi.org/10.54691/bcpbm.v44i.4794.

Salim, Fadzilah, and Nur Abu. “Used Car Price Estimation: Moving from Linear Regression towards a New S-Curve Model.” *International Journal of Business and Society*, vol. 22, Dec. 2021, pp. 1174–87, https://doi.org/10.33736/ijbs.4293.2021.

Scornet, Erwan. *Trees, Forests, and Impurity-Based Variable Importance*. 2021, https://arxiv.org/abs/2001.04295.

Ting Tin Tin, et al. “Car Depreciation Price Prediction Using Multiple Machine Learning Algorithms.” *Pakistan Journal of Life & Social Sciences*, vol. 22, no. 2, July 2024, pp. 19239–51. Academic Search Ultimate, 185141745, *EBSCOhost*, https://doi.org/10.57239/PJLSS-2024-22.2.001411.

Zhang, Yunyi, and Dimitris N. Politis. *Ridge Regression Revisited: Debiasing, Thresholding and Bootstrap*. 2021, https://arxiv.org/abs/2009.08071.

Zheng, Yufan. “Machine Learning Optimization and Challenges in Used Car Price Prediction.” *ITM Web of Conferences*, vol. 70, Jan. 2025, https://doi.org/10.1051/itmconf/20257004032.