CAR PRICE ESTIMATOR

Daniel Klinger

Peter Symonds College
Candidate no. 9400

Contents

Analysis

Background to project

With the automobile industry ever growing, cars have become an essential part of our lives. In the UK, the average person goes through 8 cars in their lifetime. Therefore, it is inevitable that everyone will have to go through the process of buying and selling a car multiple times in their lifetime.

The used car market is extremely unpredictable which stems from the dynamic nature of supply and demand. For example, the global electric chip shortage that began in 2020 lead to a substantial plummet in the car supply as semiconductor chips are essential for modern vehicles. As a result, second-hand car prices soared up as much as 32% between 2021 and 2022 as more buyers turned to used cars. Since then, the ever-changing levels of supply and demand see the used car market in a state of variable volatility.

Due to the evident unpredictability of the market, the process of pricing a car can be very complex. Often, the process relies on the expertise of specialists to provide a quote for the car which can be both time-consuming and subjective.

To address this challenge, the focus of my project is to produce a car price estimator. The car price estimator aims to provide a car price valuation with reasonable accuracy using machine learning regression. This valuation will be computed given a select number of predictor variables regarding the characteristics and condition of the car. These predictor variables include model, mileage, year of production, engine size, leather interior and many more.

Overall, I aim to develop a car price estimator that produces valuations to some level of reliability to aid buyers and sellers to come to informed decisions, regarding the price of the car in question.

Identification of end-users

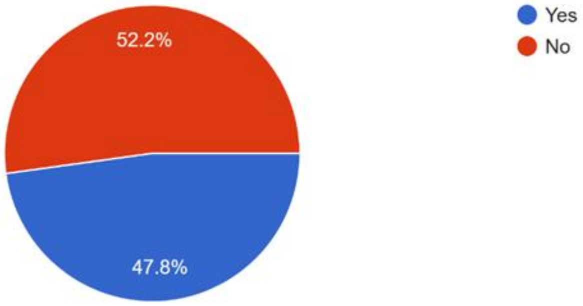
The prospective users will consist of individuals who are interested in buying and selling cars. These users include people looking to sell their car to gauge an estimate of a sensible asking price as well as potential car buyers who want to determine whether the seller is advertising a fair price.

My primary client for this project will be Bernhard Klinger, who is interested in selling his 2014 Honda Civic.

Survey for prospective users

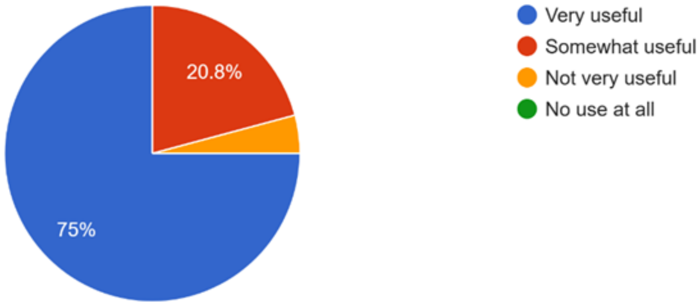
Have you used online valuation tools for estimating the price of used cars before?

23 responses



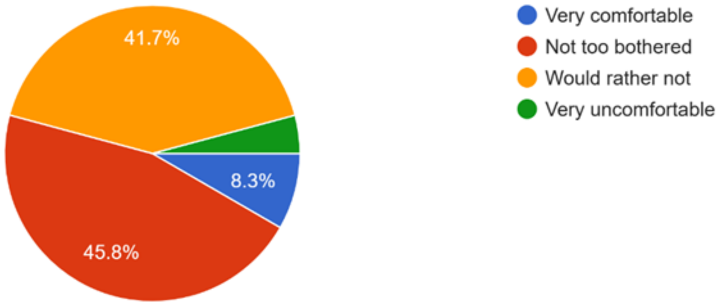
When buying/selling a car, how useful would it be to have a quick tool to estimate the price of your car?

24 responses



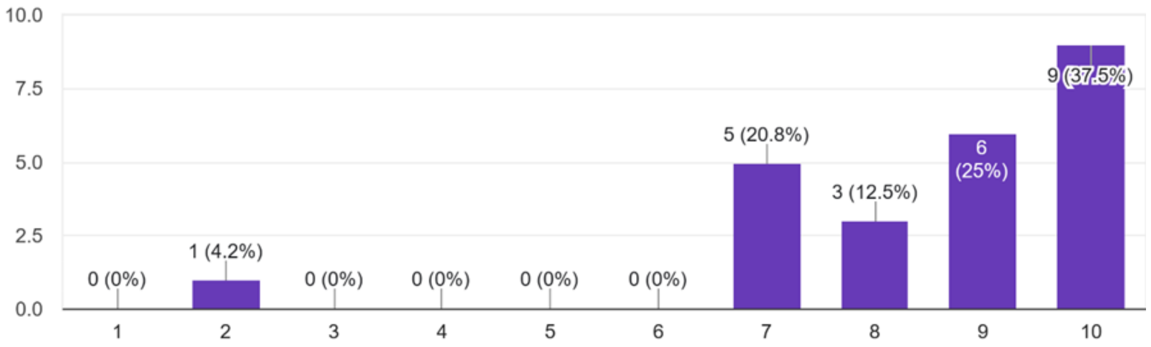
How comfortable are you with providing personal data (e.g., contact details, postcode) to a car price valuation tool?

24 responses



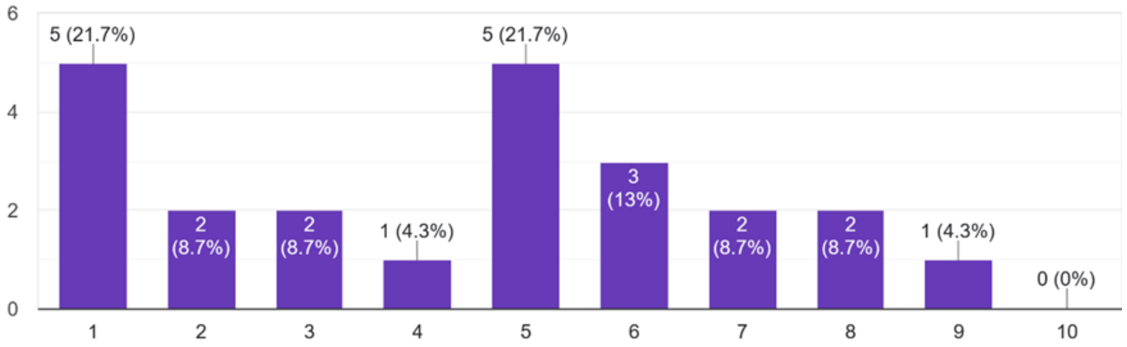
How useful would it be to be able to customise the car details that you enter? (e.g., model, mileage, fuel type)

24 responses



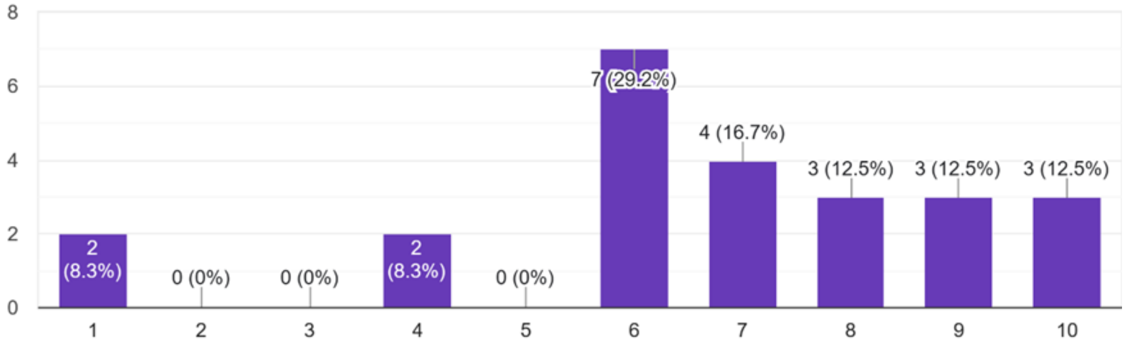
How beneficial is having an option to print a copy of the price estimation?

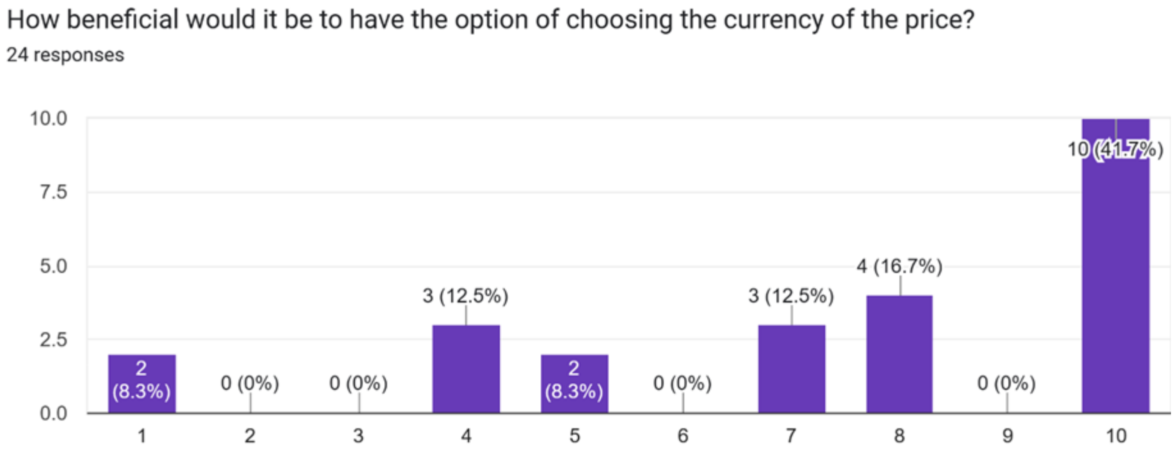
23 responses



How useful would it be to have an account for accessing your past price estimations?

24 responses




How beneficial would it be to have the option of choosing the currency of the price?

24 responses

Analysis of the Survey

Car Price Estimator potential features	Mean score /10
Customised vehicle input details	8.54
Option to print the price estimation	4.39
Account to access past price estimations	6.71
Option to select the currency	7.38

Key points gained from prospective users:

- The general concept of a car valuation tool is largely considered to be very useful for helping users with the buying or selling of a car.
- The general consensus regarding entering contact details is that users would prefer to not enter contact details.
- It is very important to provide the ability for users to customise car details entered.
- Most users feel it is unnecessary to have an option to print the price estimation.
- Most users feel a log-in system with accounts has some use, but it is not an essential user requirement.
- Users had conflicting opinions about the option of choosing the currency.
    - Could be useful if you want to buy/sell a car when outside of the UK.
    - But the tool will be aimed at the UK market so currency would always be GBP.

Interview with primary client

How often do you purchase used cars?

Bernhard: Every 10 years.

What are the challenges you endure when faced with the prospect of selling your car?

Bernhard: The fact that I don’t know how much it is worth. Also, I’m not sure how much bumps or scratches on the car affect the price, and whether I can even sell it past a certain condition.

Have you used any tools for estimating the price of a used car?

Bernhard: I have yeah. Back when I was in America, there used to be some magazines that you could buy at the shop. They had all the different cars, their prices and some tables that tell you how to adjust the base price based on the condition of the car. Then, those were obviously created into websites later.

How was your experience with this tool?

Bernhard: It was quite detailed, and it did help me to sell my old car. I used it to find a new car too, so it was helpful. But, of course, it would have been much better with an automated app to do it much quicker.

What are the key functionalities that you would like to see in the car price estimator?

Bernhard: It should have an easy-to-use interface with some buttons, and minimal inputs. Also, it should, of course, come up with a good price. Ideally, it will show a confidence level with how good the estimate is. This could be with a colour like green, amber, red.

How important is it that the tool is user-friendly?

Bernhard: Very important because I’m not very good at using tools. I want it to be very simple, not over-the-top and don’t ask me 100 questions about my car, just ask what’s necessary.

Would it be necessary to include a log-in system in order to gain access to your past price estimations?

Bernhard: I don’t think I will need that as I only use it once every ten years, so I don’t need to see what I have done ten years ago.

Would it be useful to include an option of displaying a report about how the algorithm estimates the car price?

Bernhard: Not really. I would like an indication of the confidence level of the estimate, so I know whether I need to try other tools to get multiple opinions of the price or whether your tool is so good that I don’t need to.

Is there any additional information that should be displayed about the car that would be useful, apart from the price?

Bernhard: Maybe just a few details to confirm that the tool is estimating the price of the right car.

Are there any other thoughts you would like to share about the car price estimator project?

Bernhard: I want my privacy protected; I don’t want you to share my data with others. And I don’t want my name to be mentioned so I don’t need a message saying, “Hello Bernhard, here is your price estimation”. It shouldn’t crash and it should be properly tested before, with quality assurance on what you’ve done.

Summary of the interview

In the interview with the primary client, several key insights were raised which will allow me to tailor the user requirements accordingly. Initially, Bernhard expressed his frustrations regarding the challenges when selling his car and highlighted that he is unsure about how the condition of the car affects its price. This would be difficult for me to account for within my program as there is no data in my dataset regarding a condition rating of each car, however, I assured Bernhard that I can provide an accurate base price, which can then be self-adjusted if there are any major problems with the condition.

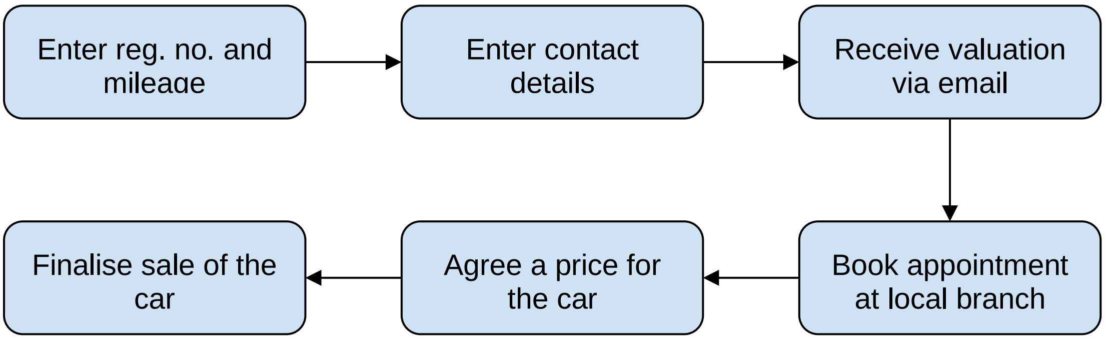
As well as this, he expressed a strong desire for a very simple user-interface with minimal inputs and only those which are necessary. He also emphasised the importance of fast and accurate results as well as a fully working program to limit the stress of selling his car. Additionally, he made it clear that there would be zero benefits to implementing a log-in system because he only needs to change his car every ten years. Therefore, it would be somewhat pointless having an account for an application that he would so rarely use.

He also raised an idea to enhance the tool’s reliability, which is to include a colour-coded confidence level that will indicate to the user how accurate the estimation is likely to be. Lastly, he emphasised that he wants his privacy protected and requested that no contact details should be taken.

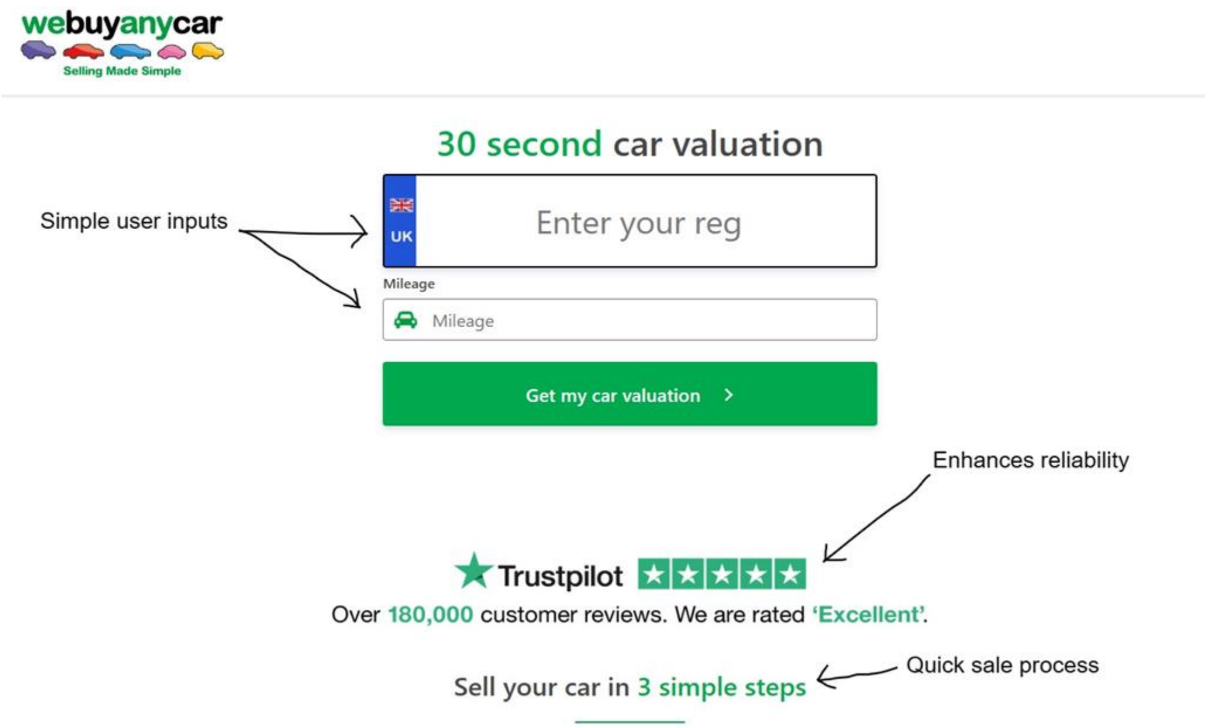
Description of current system

There are multiple existing Free Car Valuation websites, one of the most prominent being WeBuyAnyCar.com. WeBuyAnyCar.com is an online platform that offers a convenient and hassle-free process for users to sell used cars. The process consists of three simple steps: receiving an instant free valuation, booking an appointment at your local branch and the actual sale of your car. This convenient approach allows the user to complete the seemingly complex task of selling your car, all in one place without having to search for a buyer. For my project, I will focus on the instant car valuation stage of the sale process and uncover potential drawbacks that could be addressed through my car price estimator.

Workflow diagram illustrating the WeBuyAnyCar process for customers:


Enter reg. no. and mileage
Enter contact details
Receive valuation via email
Book appointment at local branch
Agree a price for the car
Finalise sale of the car

Page 1 of the 30-second car valuation feature

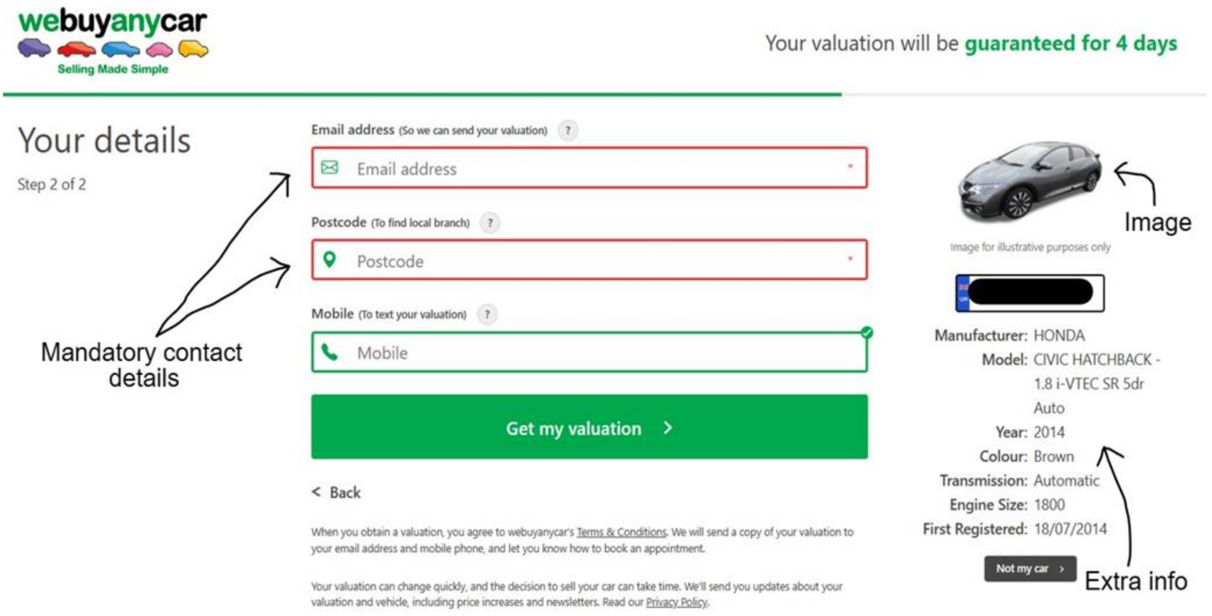


30 second car valuation

Over 180,000 customer reviews. We are rated 'Excellent'.

Sell your car in 3 simple steps

Page 2 after submitting registration number and mileage



Your valuation will be **guaranteed for 4 days**

Your details

Step 2 of 2

Email address (So we can send your valuation) ?

Email address *

Postcode (To find local branch) ?

Postcode *

Mobile (To text your valuation) ?

Mobile

When you obtain a valuation, you agree to webuyanycar's Terms & Conditions. We will send a copy of your valuation to your email address and mobile phone, and let you know how to book an appointment.

Your valuation can change quickly, and the decision to sell your car can take time. We'll send you updates about your valuation and vehicle, including price increases and newsletters. Read our Privacy Policy.

Image for illustrative purposes only

[redacted]

Manufacturer:	HONDA
Model:	CIVIC HATCHBACK - 1.8 i-VTEC SR 5dr Auto
Year:	2014
Colour:	Brown
Transmission:	Automatic
Engine Size:	1800
First Registered:	18/07/2014

Advantages of the system:

- Initial 30-second car valuation.
- Simple but user-friendly interface.
- Requires users to enter minimal details about the car: only their registration number and mileage.
- ‘3 million users’ and ‘rated excellent’ – enhances the reliability and reputation of the site.
- ‘Sell your car in 3 simple steps’ – provides a convenient selling experience all in one place.
- Displays useful vehicle information including an image.
- Price estimations use data provided by CAP, which is frequently updated and adheres to market fluctuations.

Disadvantages of the system:

- Requires the user to enter correct details in order to get an instant valuation.
    - Some users feel uncomfortable providing personal data (evidenced by prospective users’ survey).
    - Site has additional responsibilities regarding personal data protection to adhere to the GDPR.
- Relies on user’s knowledge of the car’s registration number.
    - Buyers looking to gauge whether an asking price is reasonable often won’t have access to the registration number of a car.

Overview of the current system

The 30-second car valuation feature on WeBuyAnyCar.com is simple and effective for its purpose of serving as the initial stage of the sale process that WeBuyAnyCar offers. As for my project, I plan to incorporate a similar user interface that is very simple to use, with dropdowns, buttons and text inputs. This should allow my application to be as accessible as possible for everyone and provide users a quick and easy experience to receive their car price estimation. The system’s registration number input is especially convenient for users, as this is used as a parameter for an API request, which allows the system to gain extended information about the car used to estimate its price, whilst only asking for a single user input. The display of vehicle information is helpful for confirming that it is the right car being assigned a price valuation, so I plan to replicate this idea in my solution.

The main disadvantage of the system is that users are required to enter contact details. Whilst this is vital in order for WeBuyAnyCar to contact their customers to arrange an appointment and complete the sale, it is unnecessary if the user just wants a third-party car valuation. Additionally, some people are uncomfortable giving away personal data online. Therefore, my project will not require any contact details, and will only ask for the necessary information that can help lead to a more accurate car price estimation.

Acceptable limitations

- There is no free API that contains the registration number along with respective car details, hence the ability to have few user inputs and provide select user input customisation is limited.
- The dataset I will use does not contain any predictive variables regarding the condition so it will not be possible to produce an estimate that considers the specific condition of the car; however, the general wear and tear of a car will be considered through the predictor ‘year of production’.
- The dataset consists of only 20,000 records and its reliability is unknown, therefore, the car price estimation can only reach a limited level of accuracy.
- The dataset contains cars that were produced in 2020 at the latest, so the tool will not be able to estimate the price of cars that are newer than 2020.
- The tool cannot estimate the car price if the user has a rare car model that is not in the dataset.
- I am unable to access live data so the dataset could be somewhat outdated.
- There are no free APIs or downloadable datasets containing images of a range of car models so I cannot include an image of the car as part of the output.

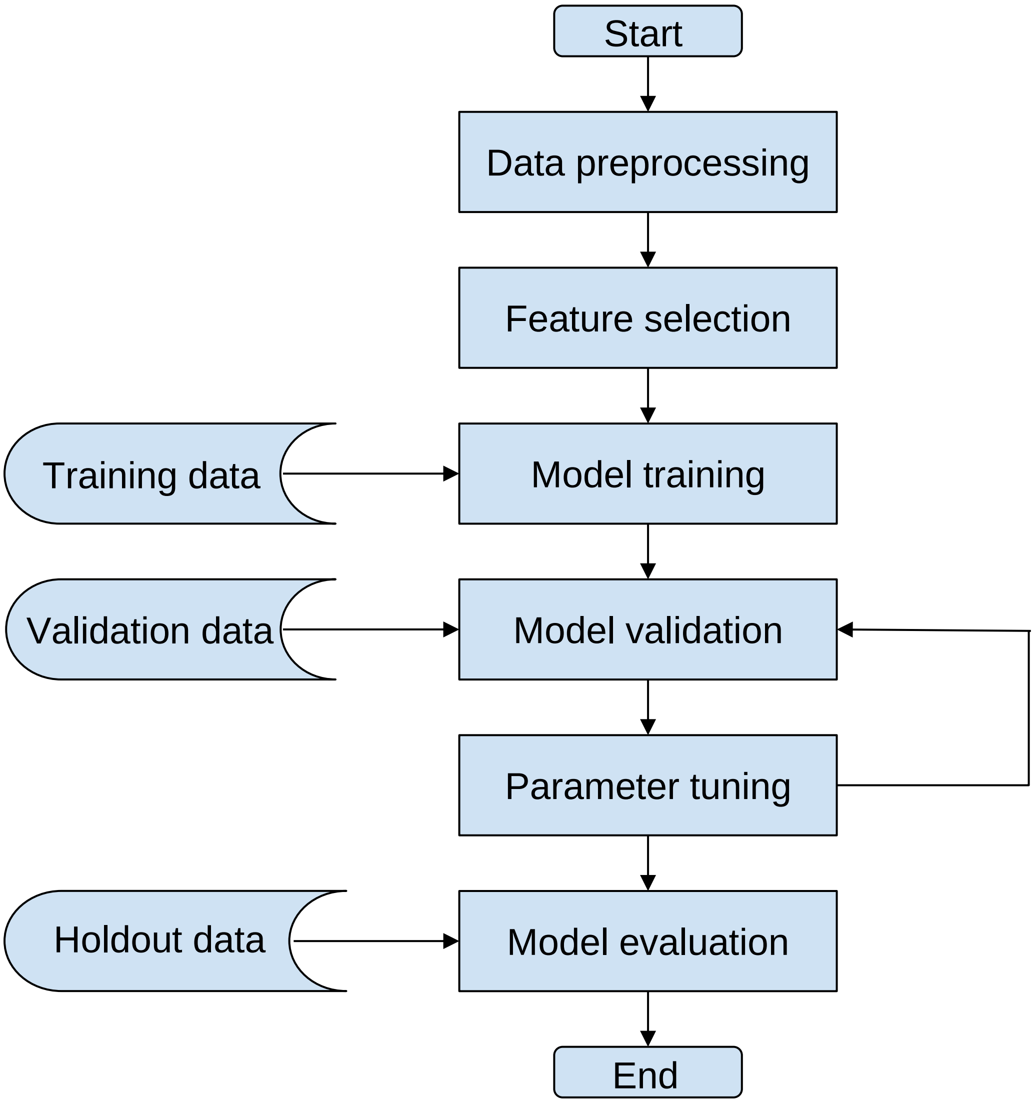
Modelling

What is regression?

Regression is a statistical technique used to analyse the relationship between a dependent variable (variable being predicted) and one or more independent variables (predictor variables). This relationship can be used to predict the dependent variable, given the corresponding independent variable(s). In other words, regression is a method of finding a mathematical model that can predict the value of a variable based on other variables. There are many different regression techniques, ranging from relatively simple to highly advanced methods with few or many variables.

Regression model workflow diagram

Below shows a typical workflow for developing a regression model.



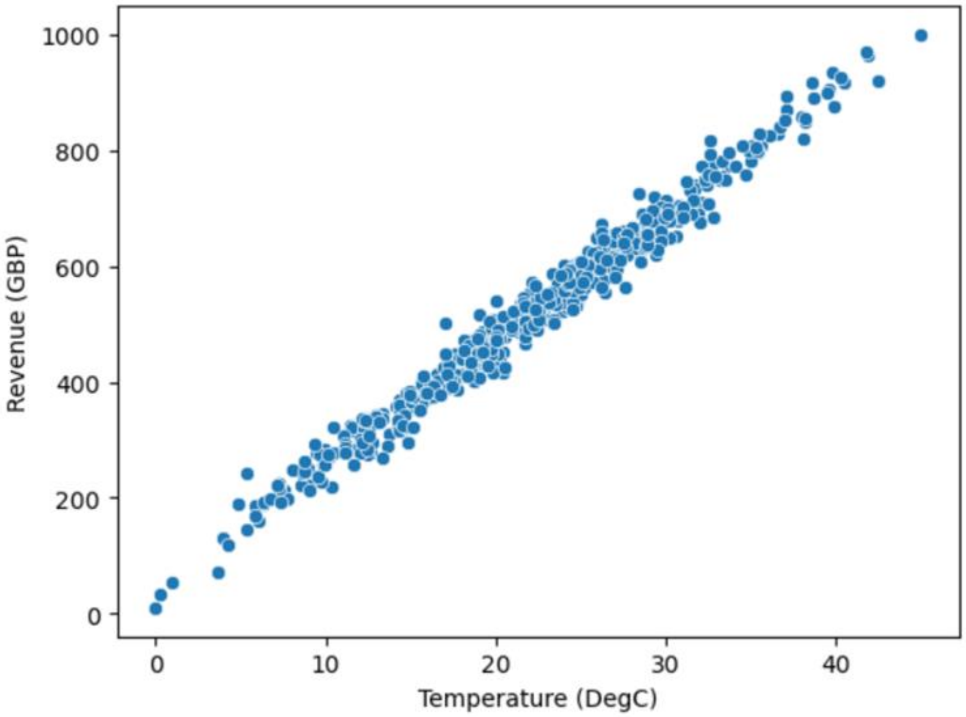
Linear regression explained

Linear regression is a parametric form of regression that is identified through its linearity within the regression coefficients. However, it can model both linear and non-linear relationships.

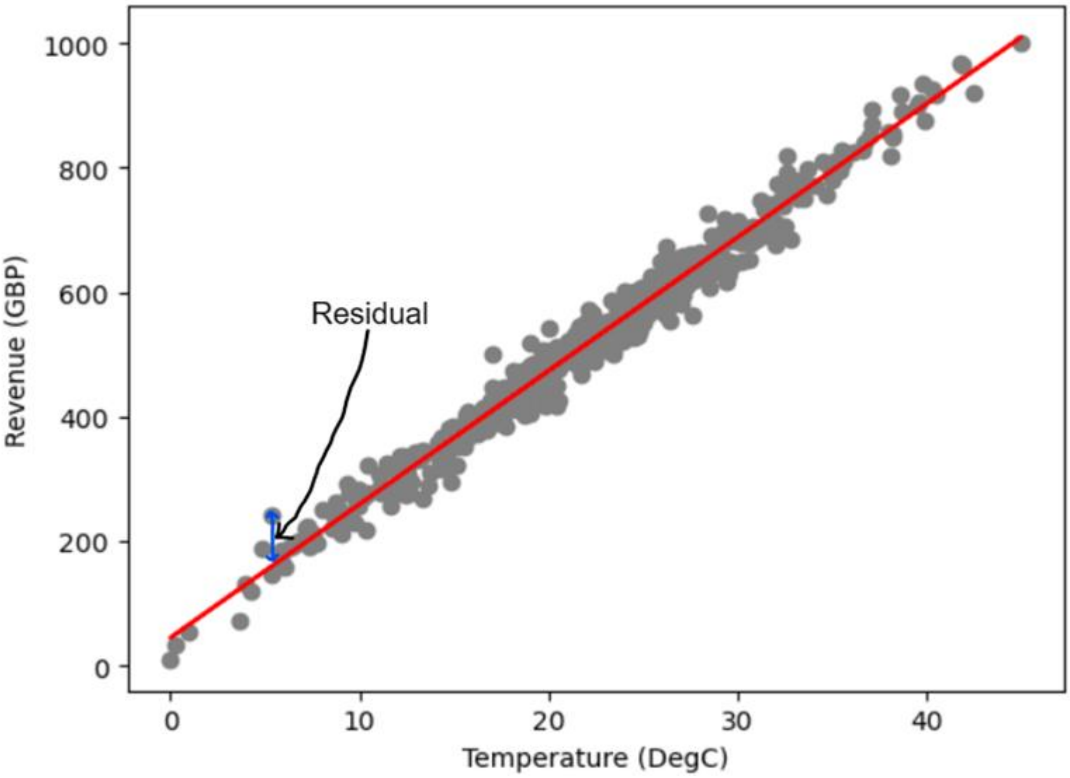
I have carried out some initial prototyping for a very simple regression task in Python using the SciKit Learn library in Python, which I have demonstrated below.

The simplest linear regression technique is creatively named Simple Linear regression. This models a linear relationship and consists of two variables only: one dependent variable and one independent variable.

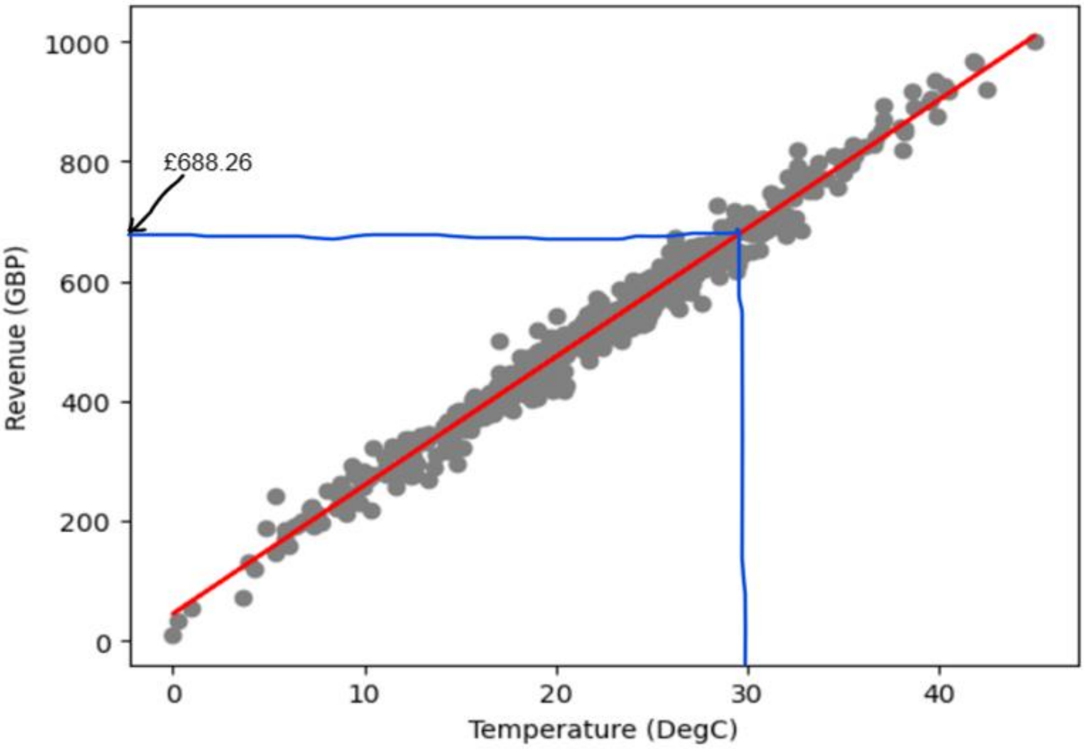
To make things simple, I will use the example of an ice cream business with the goal to predict today's revenue based on the outside air temperature. We are given some data about the revenue and temperature for the last 500 days. The data looks like this:



Using simple linear regression, a line can be created in the form y = mx + b to fit the data, which will allow us to predict revenues. To do this, an algorithm is executed which calculates the values of m and b, such that the sum of the residuals squared is minimised. A residual is the difference between the actual value of the dependent variable (revenue in this case) and the predicted value. Once the m and b values have been calculated (in this case, m= 21.45 and b= 44.76), the training of the regression model is complete. The regression line can be seen in red below.



Now, if I wanted to predict the revenue the business will generate today at a temperature of 30 degrees, I can use the trained model and simply substitute 30 as the x value in the equation y = mx + b. The resulting y value would be the revenue prediction. In simpler terms, I could use the regression line above, by drawing a vertical line up from 30 degrees to the regression line and then a horizontal line to the y axis, as shown below.



Now the regression model has been created, we can evaluate its accuracy. To do this, the model can be tested with a smaller set of unseen data points, known as test data. By comparing the predicted values and the respective actual values, various error metrics can be calculated. For example, Mean Squared Error is calculated by the sum of (actual value – predicted value)^2 ÷ number of test values.

This same concept is replicated in more complex linear regression techniques such as polynomial, multivariate, ridge and lasso regression as an extremely powerful way to make predictions in much more complicated scenarios. Each technique works better in different scenarios depending on the quality, volume and distribution of data.

Linear regression relies on some assumptions:

- Depending on the technique being linear or polynomial, the dependent variable has a linear or curvilinear relationship with the predictor variables.
- Predictor variables are normally distributed (in the shape of a bell curve)
- No or little multicollinearity – independent variables are not highly correlated as it can be difficult to isolate the individual effect on the dependent variable.
- In general, the residuals should be independent of each other and have no correlation.
- Homoscedasticity – the variance of the residuals is constant across all values of the predictor variables.

Decision tree regression explained

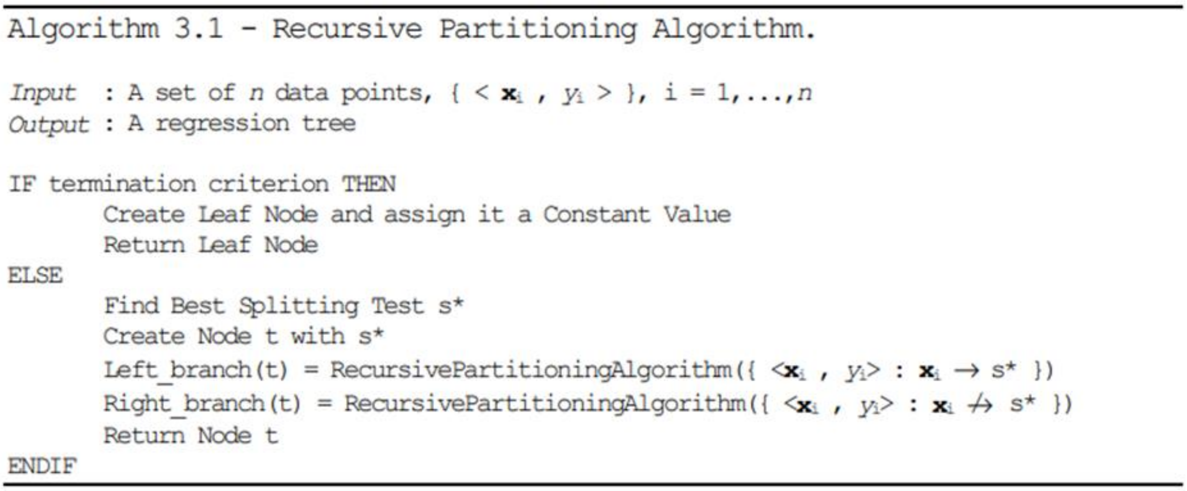
In addition to linear regression, there are numerous other categories of regression techniques, one of them being tree-based regression. Tree-based regression encompasses various powerful techniques like gradient-boosting and random forests that all stem from decision trees. Its main benefits are that it can represent non-linear relationships and it can process both numerical and categorical data. It also relies less on strict assumptions about distribution and correlation than linear regression, making it a flexible technique to model complex relationships.

Decision tree regression works by recursively splitting all the data points, using a binary tree, until the termination criteria are met for each leaf node. Termination criteria are the conditions that determine when a leaf node is created; common examples include the maximum depth of the tree or having a minimum number of data points per leaf. Data is split by binary conditions that partition the data optimally such that similar data points are grouped together.

The process of computing the optimal splitting condition begins by considering all possible threshold values (points where data is split into two partitions) for each feature. Next, the quality of the split is measured using variance reduction which quantifies the similarity of data points within each partition. Variance reduction is calculated by Variance(parent) – Σ Variance(child). This variance reduction calculation is then repeated for all features and thresholds, and the feature-threshold combination that results in the greatest variance reduction is selected as the split condition. For example, this may be: ‘Mileage >= 100,000?’.

To predict the value of the dependent variable of a given sample, the algorithm traverses down the tree by evaluating the feature conditions at each node until a leaf node is reached. Once a leaf node has been reached, the predicted value associated with the leaf node becomes the prediction. This predicted value is usually the mean or median of the training data at the given leaf node.

For example, below is a sample of pseudocode for producing a regression tree:

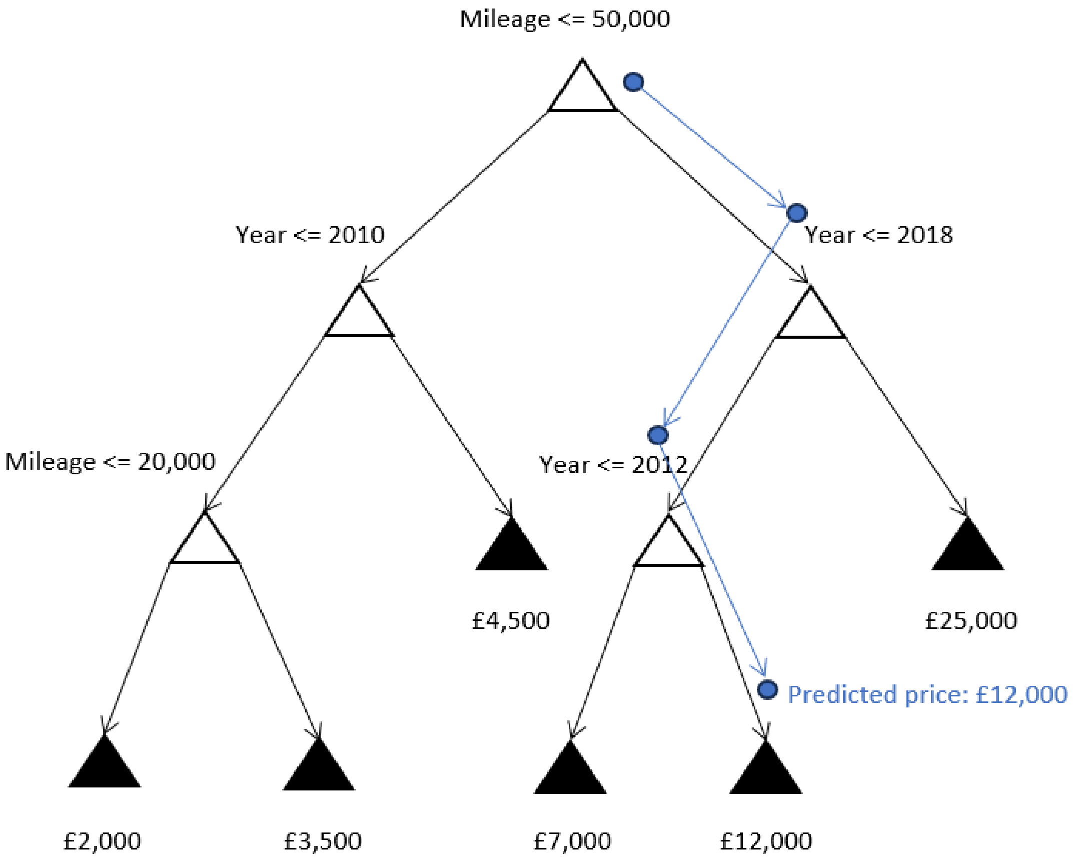


```
Algorithm 3.1 - Recursive Partitioning Algorithm.

Input  : A set of $n$ data points, { < $\mathbf{x}_i$ , $y_i$ > }, i = 1,...,$n$
Output : A regression tree

IF termination criterion THEN
       Create Leaf Node and assign it a Constant Value
       Return Leaf Node
ELSE
       Find Best Splitting Test s*
       Create Node t with s*
       Left_branch(t) = RecursivePartitioningAlgorithm({ <$\mathbf{x}_i$ , $y_i$> : $\mathbf{x}_i \rightarrow s^*$ })
       Right_branch(t) = RecursivePartitioningAlgorithm({ <$\mathbf{x}_i$ , $y_i$> : $\mathbf{x}_i \not\rightarrow s^*$ })
       Return Node t
ENDIF
```

Taken from: Luis Fernando Rainho Alves Torgo Microsoft Word - PhD Thesis.doc (up.pt)

Below is an example of a very simple partitioning tree.

The path in blue, shows how a car's price with Mileage 65,000 and Year 2018 is predicted by traversing a regression tree.



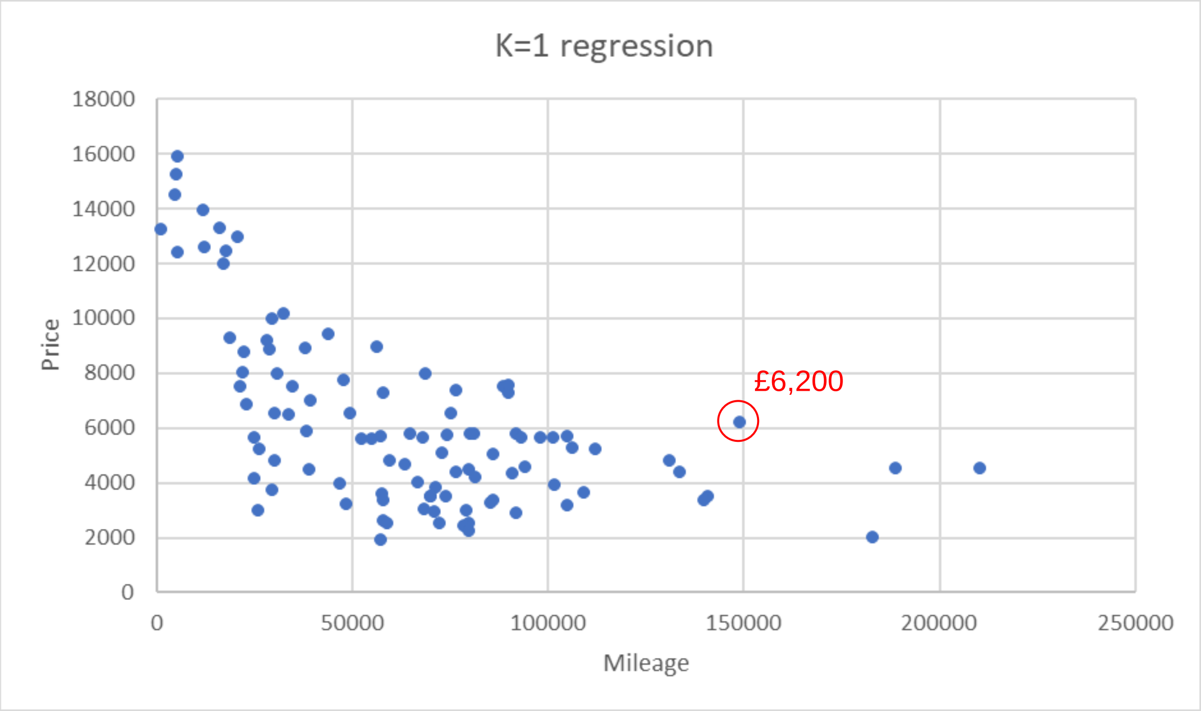
K-nearest neighbours regression explained

K-nearest neighbours (KNN) regression is another technique that predicts a value by averaging several neighbouring data points, where K is the number of neighbours.

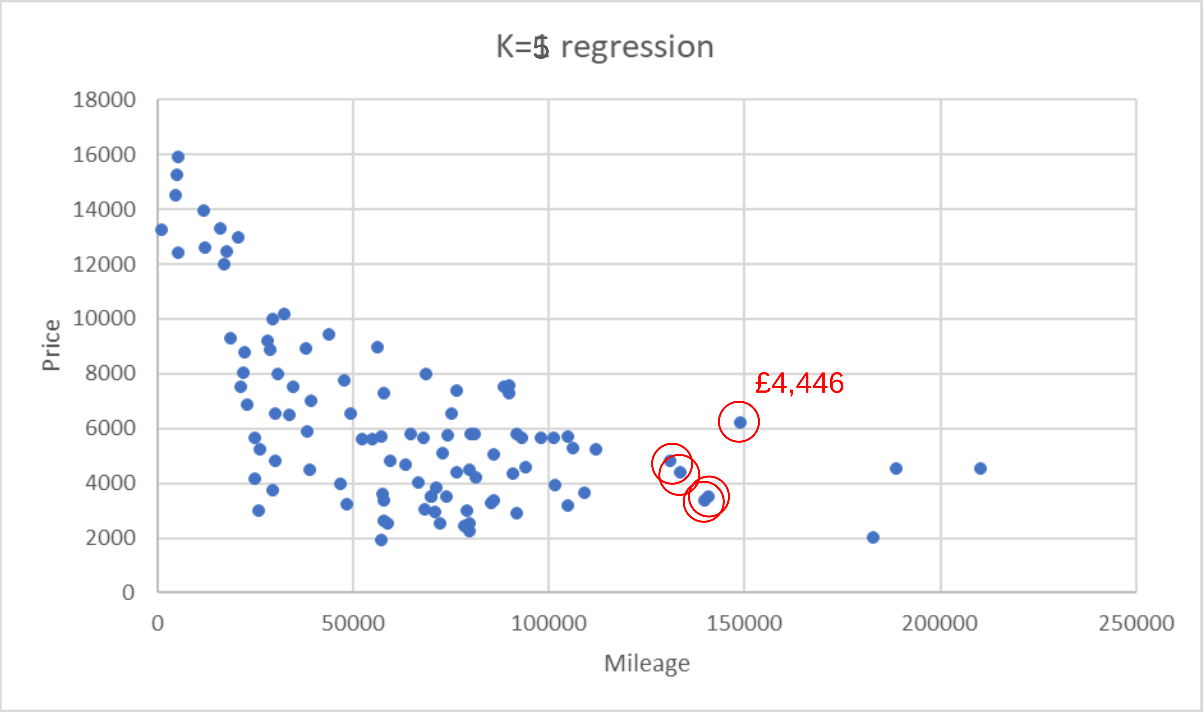
KNN regression is a more flexible method as it does not rely on assumptions about linear relationships and normal distribution. This is because it is a non-parametric technique that does not predict values by establishing an overall relationship, it only concerns neighbouring data points. However, the main disadvantage of KNN regression is that it can be difficult to locate neighbours with high dimensionality (lots of predictor variables).

For example, we can use KNN regression to predict the price of used Honda cars based on their mileage.

Below, shows how a Honda's price with 150,000 mileage can be predicted with KNN regression with different K values. The neighbours are shown in red.



For the nearest K neighbours, an average of the mileage across these neighbours will be calculated as the predicted price. In this case, I have calculated the mean for the nearest 5 neighbours, which are circled in red.



Clearly, the values of K have a large impact on the accuracy of the estimation. If K is too small, the model will overfit the training data and likewise if K is too large, the model will underfit the data and it won't capture all the patterns in the data. Therefore, the sweet spot for the K value depends entirely on the data. A common method for determining the optimal K value is to use cross-validation.

KNN regression assumptions:

- Data points which are close to each other are highly similar.
- All the predictive features equally contribute to the dependent variable.
- Data points are independent of each other – data points are unrelated and do not influence each other.
- KNN does not have strict assumptions on the distribution of variables, however it is sensitive to the scale of variables so predictor variables should be scaled so that the Euclidean distance (square root of the sum of the squares of the distances) is more representative.

Statistical Formulae

Notation definitions:

- n is the number of data points in a dataset.
- x_i is a given value of the independent variable.
- $\tilde{x}$ is the median of the independent variable.
- $\bar{x}$ is the mean of the independent variable.
- y_i is a given value of the dependent variable.
- $\bar{y}$ is the mean of the dependent variable.
- $\hat{y}_i$ is the predicted value of the dependent variable.
- σ is the standard deviation of a dataset.

Distribution and variance:

Mean- the average of a dataset.

$$\bar{x} = (\Sigma x_i) / n$$

Median- the middle value of an ordered dataset.

- When n is odd, the median is the ((n + 1) / 2)th value.
- When n is even, the median is the average of (n / 2)th value and ((n / 2) + 1)th value.

Range- the difference between the highest and lowest values in a dataset.

$$\text{Range}(X) = \text{Max}(X) - \text{Min}(X)$$

Interquartile range- the range of values in the middle 50% of the data.

$$IQR = Q3 - Q1$$

Standard deviation- measure of the spread of values around the mean.

$$\sigma = \sqrt{} ((\Sigma(x_i - \bar{x})^2) / n)$$

Variance- measure of the spread of values around the mean.

$$\sigma^2 = \Sigma(x_i - \bar{x})^2 / n$$

Covariance- measure that indicates how changes in one variable are associated with changes in another variable.

$$\text{cov}(X, Y) = \Sigma((x_i - \bar{x}) * (y_i - \bar{y})) / (n - 1)$$

Outlier identification:

Z-score- indicates how many standard deviations away from the mean a data point is.

$z = (x_i - \bar{x}) / \sigma$

Modified Z-score- a more robust method than using Z-scores which uses the median.

Modified $z = 0.6745 * (x_i - \tilde{x}) / MAD$

Median Absolute Deviation- a robust measure of variability.

$MAD = med (|x_i - \tilde{x}|)$

IQR range outlier detection- method for detecting outliers by observing beyond the lower and upper quartiles.

Outliers beyond: $(Q1 – k * IQR)$ or $(Q3 + k * IQR)$ where k is usually 1.5 or 3

Correlation:

Pearson Correlation coefficient- measure of the strength and direction of the linear relationship between two variables.

$r = \Sigma((x_i - \bar{x}) * (y_i - \bar{y})) / \sqrt{} (\Sigma(x_i - \bar{x})^2 * \Sigma(y_i - \bar{y})^2)$

Point Biserial Correlation coefficient- measure of the relationship between a continuous variable and a binary variable.

$r_{pb} = (\bar{y}_1 - \bar{y}_2) * (\sqrt{} (p * q)) / \sigma_y$

- $\bar{y}_1$ and $\bar{y}_2$ are the means of the continuous variable when the binary value is 1 and 0 respectively.
- p and q are the proportions of the binary value 1 and 0 respectively.)

Regression error metrics:

Mean absolute error- the average of the magnitude of residuals.

$MAE = \Sigma(|y_i - \hat{y}_i|) / n$

Mean square error- the residuals are squared which gives higher weight to larger errors.

$MSE = \Sigma((y_i - \hat{y}_i)^2) / n$

Root mean square error- the average magnitude of errors scaled to the dependent variable.

$RMSE = \surd\ (\Sigma((y_i - \hat{y}_i)^2) / n)$

R-squared- represents the proportion of the variance in the dependent variable that can be explained by the independent variables, with the value ranging from 0 to 1.

$R^2 = 1 - \Sigma((y_i - \hat{y}_i)^2) / \Sigma(y_i - \bar{y})^2)$

Adjusted R-squared- adjusts R-squared to account for the number of predictive variables.

$Adj\ R^2 = 1 - ((1 - R^2) * (n - 1) / (n - k - 1))$

Objectives

Main objective: To produce a car price estimator that will calculate a car price based on a set of carefully selected variables (such as mileage, model, fuel type, leather interior etc.) using machine learning techniques. Also, to develop and evaluate a variety of machine learning regression models (such as simple linear, multi linear, decision trees, KNN) and integrate the best performing model into my final program.

Inputs:

1. All inputs should be entered through a GUI that complies with the following:
    a. A form that contains suitable controls for user inputs depending on their types such as:
        i. Enumeration.
        ii. Boolean.
        iii. Integer.
        iv. Single (Float).
    b. The form should be simple and easy to use:
        i. Content is spaced and not too dense.
        ii. Controls are sensibly organised.
        iii. All inputs are necessary and impactful.
    c. Live dropdown updates:
        i. Model dropdown displays only applicable models for the user's selected make.
    d. Includes a reset button:
        i. Resets all user inputs to their original value/state.
2. All inputs must be validated before submitting the form.
    a. Errors should be raised for the following:
        i. Incorrect data type for text-based inputs.
        ii. Not in range for numeric inputs.
        iii. If no option has been selected for dropdowns.
        iv. If no option has been checked for radio buttons.
        v. Attempt at entering a value that is not an option in the dropdown.
        vi. Attempt at selecting a model before the make.
    b. Final validation check when form is submitted:
        i. Display a message if any of the inputs have not been completed or incorrectly entered.

Processing:

3. Regression analysis before final program.
    a. Initial dataset validation.
        i. Dataset must be checked against a data dictionary to ensure data is in a valid form.
    b. Dataset must be pre-processed via the following techniques:
        i. Check for missing values.
        ii. Selectively delete or impute missing values.
        iii. Check for duplicate records and remove them if found.

- iv. Locate outliers for each numerical feature.
- v. Selectively delete or impute outliers.
- vi. Engineer new features if suitable.
- vii. Transform categorical variables into a numerical format.
- viii. Scale numerical features.

- c. Training data must undergo feature selection via the following techniques:
  - i. Calculate correlation coefficients for continuous and dichotomous variables.
  - ii. Visualise data correlation through scatter plots.
  - iii. Visualise data distribution through box plots.
  - iv. Obtain a ranking of feature importance.
- d. Each model should be trained differently:
  - i. Simple linear regression training coded from scratch (without a library).
  - ii. Multi linear regression training coded from scratch.
  - iii. K-nearest neighbours regression coded from scratch.
  - iv. Decision tree training coded using a C# supported library.
- e. Each model should be validated as follows:
  - i. The model’s performance is assessed using cross validation.
  - ii. The error metrics used are selected such that there is minimised bias toward any one model.
- f. The development of each regression model should follow this process:
  - i. Preprocess data.
  - ii. Split data into training and test.
  - iii. Perform feature selection.
  - iv. Train model with training data.
  - v. Fit model with test data.
  - vi. Evaluate the model with a range of error metrics.
  - vii. Fine-tune the model by altering features and/or hyperparameters.
- g. Final model evaluation:
  - i. Each fine-tuned model is tested once more.
  - ii. Best-performing model is determined.
  - iii. Best model integrated with the UI application.

4. Integrate the best model with the UI.
   - a. Regression model should be used to estimate the car price as follows:
     - i. Model initially saved to file.
     - ii. Model instantiated at run-time.
     - iii. User’s input data fed into model.
     - iv. Model predicts the car price.
     - v. Price checked for validity.
     - vi. Confidence level calculated.
5. Object-oriented programming paradigm used to create the program in C#.
   - a. Program should consist of two projects:
     - i. Regression analysis console application.
     - ii. GUI form application.
   - b. Regression analysis project:
     - i. Project should be split into sensible classes.
     - ii. Each class must have a clear responsibility.

iii. Utilise inheritance and composition where necessary.

Outputs:

6. After submitting the form, an output page should be displayed.
    a. The output page should comply with the following:
        i. Price formatted well and in large lettering.
        ii. User’s vehicle information is displayed.
        iii. Content organised nicely.
        iv. There is a colour-coded confidence level regarding the accuracy of the estimation shown.
        v. An error message is displayed if there was a problem with generating a price estimation.
        vi. The date of valuation is displayed.
        vii. There is an option to ‘Go Back’.
        viii. There is an option to ‘print’.
    b. The ‘Go Back’ button should comply with the following:
        i. Takes the user back to the form page.
        ii. Form page still contains the user's entered details.
    c. The print button should comply with the following:
        i. Print preview appears initially.
        ii. Print details can be customised according to the user’s preferences.
        iii. Document should print if the user confirms so.

Research log

Date (w/b)	Research
12/06/2023	Used car market: https://exactlyhowlong.com/how-many-cars-does-an-average-person-own-in-a-lifetime-and-why/ https://www.theguardian.com/business/2023/mar/06/uk-car-sales-rise-february-industry-recovers-chip-shortage https://www.banburyguardian.co.uk/lifestyle/cars/used-car-prices-4108733 Dataset: https://www.kaggle.com/datasets/sidharth178/car-prices-dataset Udemy Machine Learning Regression course by Dr. Ryan Ahmed: https://www.udemy.com/course/machine-learning-regression-masterclass-in-python/
19/06/2023	Analysis of current system: https://www.webuyanycar.com/
26/06/2023	Statistical formulae: https://study.com/academy/lesson/the-correlation-coefficient-definition-formula-example.html#:~:text=Course%2048K%20views-,Correlation%20Coefficient%20Formula,(%20%E2%88%91%20Y%20)%202%20)%20. https://www.scribbr.co.uk/stats/statistical-outliers/ https://www.statology.org/modified-z-score/ https://study.com/learn/lesson/median-absolute-deviation-formula-examples.html https://medium.com/analytics-vidhya/point-biserial-correlation-coefficient-b572fcad80aa

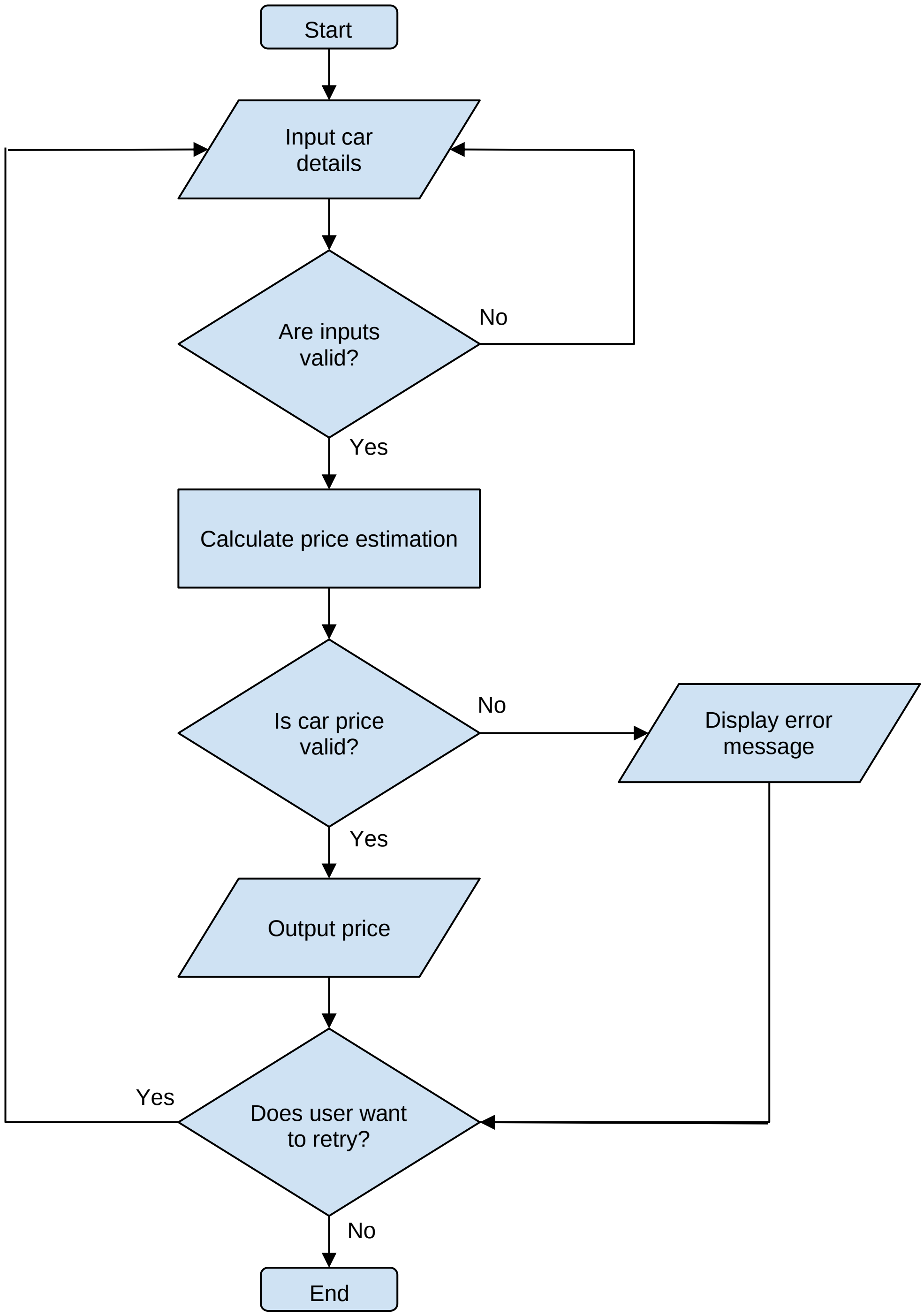
03/07/2023	Decision tree regression: https://c3.ai/glossary/data-science/tree-based-models/#:~:text=What%20are%20Tree%2DBased%20Models,or%20value% 20of%20a %20 home. https://www.youtube.com/watch?v=UhY5vPfQIrA KNN regression: https://www.dcc.fc.up.pt/~ltorgo/PhD/th3.pdf https://bookdown.org/tpinto_home/Regression-and-Classification/k-nearest-neighbours-regression.html
10/07/2023	Linear regression assumptions: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/assumptions-of-linear-regression/ KNN assumptions: https://neptune.ai/blog/knn-algorithm-explanation-opportunities-limitations#:~:text=A%20core%20assumption%20of%20KNN,related%20and%20%20similar%20they%20are.&text=Typically%20used%20with%20data%20trans%20mitted,also%20used%20with%20categorical%20variables.
17/07/2023	Imputing data: https://www.mastersindatascience.org/learning/how-to-deal-with-missing-data/#:~:text=The%20imputation%20method%20develops%20reasonable,option%20is%20to%20remove%20data. DataTable class: https://learn.microsoft.com/en-us/dotnet/api/system.data.datatable?view=net-7.0#methods Feature scaling: https://en.wikipedia.org/wiki/Feature_scaling
24/07/2023	OxyPlot: https://oxyplot.readthedocs.io/en/latest/introduction/features.html Cross-validation: https://www.cs.cmu.edu/~schneide/tut5/node42.html#:~:text=Cross%20validation%20is%20a%20model,it%20has%20not%20already%20seen. Regular expressions: https://www.w3schools.com/python/python_regex.asp
31/08/2023	Gradient descent: https://www.youtube.com/watch?v=sDv4f4s2SB Linear regression from scratch: https://www.youtube.com/watch?v=VmbA0pi2c

07/08/2023	Microsoft.ML Fast tree regression trainer: https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fasttreeregressiontrainer?view=ml-dotnet
14/08/2023	Merge sort: https://www.mygreatlearning.com/blog/merge-sort/
04/09/2023	ML.NET regression: https://learn.microsoft.com/en-us/dotnet/machine-learning/tutorials/predict-prices
18/09/2023	Gradient descent: https://medium.com/analytics-vidhya/implementing-gradient-descent-for-multi-linear-regression-from-scratch-3e31c114ae12
09/10/2023	Windows forms: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-7.0
16/10/2023	Print page with Windows Forms: https://learn.microsoft.com/en-nz/dotnet/desktop/winforms/advanced/how-to-print-a-windows-form?view=netframeworkdesktop-4.8

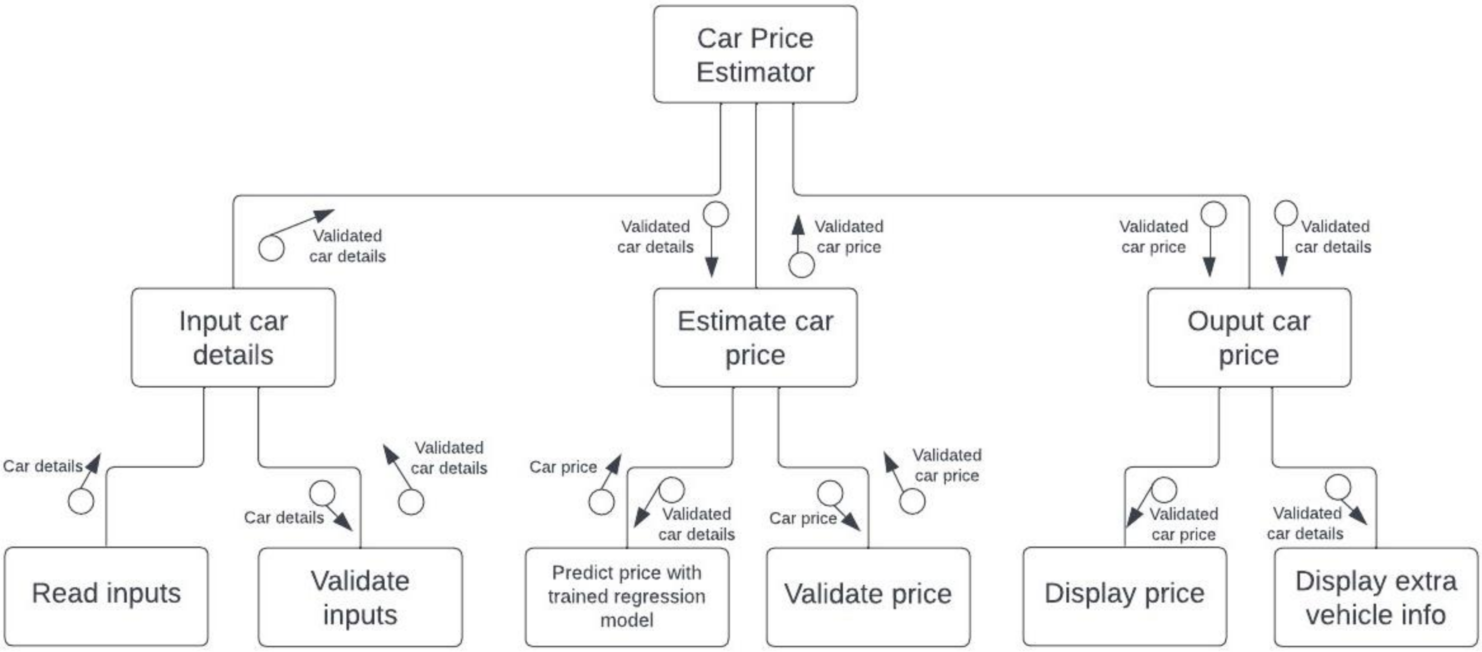
Design

General System Flowchart

The following flowchart provides a simple outline about how the final program should operate and deliver a car price estimation, after having conducted regression analysis to determine the best algorithm for the estimation. The individual processes will be broken down further later in my design.



Modular Structure diagram

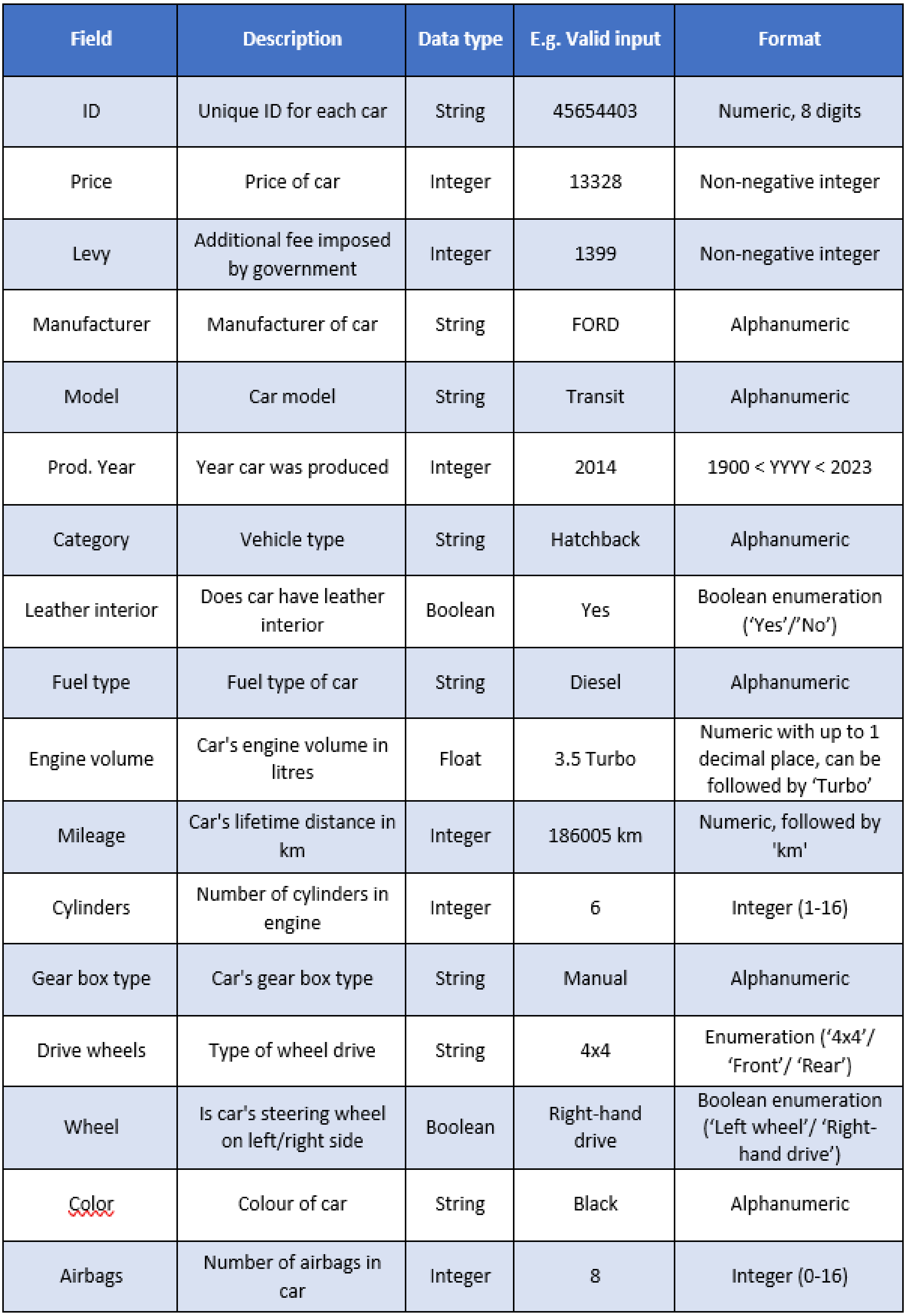


IPO chart

Input	Processing	Output
Car details such as: - Manufacturer - Model - Mileage - Year	Validate user input. Apply machine learning regression model on user's car details to generate a predicted price.	Estimated price. Additional vehicle information.

The above diagrams illustrate how the final program should operate, once a sufficiently accurate regression model has been obtained. The next few pages focus on the design for the processes required in order to obtain an accurate regression model.

Data Dictionary



Field	Description	Data type	E.g. Valid input	Format
ID	Unique ID for each car	String	45654403	Numeric, 8 digits
Price	Price of car	Integer	13328	Non-negative integer
Levy	Additional fee imposed by government	Integer	1399	Non-negative integer
Manufacturer	Manufacturer of car	String	FORD	Alphanumeric
Model	Car model	String	Transit	Alphanumeric
Prod. Year	Year car was produced	Integer	2014	1900 < YYYY < 2023
Category	Vehicle type	String	Hatchback	Alphanumeric
Leather interior	Does car have leather interior	Boolean	Yes	Boolean enumeration (‘Yes’/’No’)
Fuel type	Fuel type of car	String	Diesel	Alphanumeric
Engine volume	Car's engine volume in litres	Float	3.5 Turbo	Numeric with up to 1 decimal place, can be followed by ‘Turbo’
Mileage	Car's lifetime distance in km	Integer	186005 km	Numeric, followed by 'km'
Cylinders	Number of cylinders in engine	Integer	6	Integer (1-16)
Gear box type	Car's gear box type	String	Manual	Alphanumeric
Drive wheels	Type of wheel drive	String	4x4	Enumeration (‘4x4’/ ‘Front’/ ‘Rear’)
Wheel	Is car's steering wheel on left/right side	Boolean	Right-hand drive	Boolean enumeration (‘Left wheel’/ ‘Right-hand drive’)
Color	Colour of car	String	Black	Alphanumeric
Airbags	Number of airbags in car	Integer	8	Integer (0-16)

Initial dataset validation using a data dictionary

Before processing the dataset to be used for the regression analysis process, I need to validate the dataset and check that there are no erroneous/unexpected values. I will do this by checking that each data entry follows the rules for its field defined in a data dictionary.

The data dictionary for this will be simplified to the one above and will contain a column containing regular expression patterns to check that the format of each data entry meets with the predefined rules. The data type column will also be important for converting each field to the correct data type, ready for processing.

To implement the regular expressions format checks, I will use the ‘RegularExpressions’ namespace. It provides the method ‘Regex.IsMatch()’ which is very useful for determining whether a specified input string matches a regular expression pattern. It takes two parameters, the input string (any given data entry in my case) and a regular expression pattern; it returns true if the input string matches the pattern, and false if otherwise.

Regular expression pattern symbols:

- ^ → starts with
- $ → ends with
- + → number of occurrences >= 1
- [ ] → set of characters
- ( ) → group
- ? → optional
- | → or
- { } → specified number of occurrences

The data dictionary to be used for the dataset validation is shown below.

<table>
<tr><th>Field</th><th>Data type</th><th>Regular expression</th></tr>
<tr><td>ID</td><td>String</td><td>^\d{8}$</td></tr>
<tr><td>Price</td><td>Integer</td><td>^\d+$</td></tr>
<tr><td>Levy</td><td>Integer</td><td>^\d+|-$</td></tr>
<tr><td>Manufacturer</td><td>String</td><td>^[a-zA-Z\s\-]+$</td></tr>
<tr><td>Model</td><td>String</td><td>^[a-zA-Z0-9\s\-./+()<>!]+$</td></tr>
<tr><td>Prod. Year</td><td>Integer</td><td>^19[0-9]{2}|200[0-9]|201[0-9]|202[0-3]$</td></tr>
<tr><td>Category</td><td>String</td><td>^[a-zA-Z\s]+$</td></tr>
<tr><td>Leather interior</td><td>String</td><td>^Yes|No$</td></tr>
<tr><td>Fuel type</td><td>String</td><td>^[a-zA-Z\s\-]+$</td></tr>
<tr><td>Engine volume</td><td>String</td><td>^[0-2]?\d(\.\d)?(Turbo)?$</td></tr>
<tr><td>Mileage</td><td>String</td><td>^\d+ km$</td></tr>
<tr><td>Cylinders</td><td>Integer</td><td>^\d|1[0-6]$</td></tr>
<tr><td>Gear box type</td><td>String</td><td>^[a-zA-Z]+$</td></tr>
<tr><td>Drive wheels</td><td>String</td><td>^4x4|Rear|Front$</td></tr>
<tr><td>Wheel</td><td>String</td><td>^Left wheel|Right-hand drive$</td></tr>
<tr><td>Colour</td><td>String</td><td>^[a-zA-Z\s]+$</td></tr>
<tr><td>Airbags</td><td>Integer</td><td>^\d|1[0-6]$</td></tr>
</table>

Description of data structure to store dataset

Currently, the car price data is saved in a csv file- having downloaded it from Kaggle. Regarding my program, I will read this data and store it in a ‘DataTable’, which is a class part of the .NET Framework and the ‘System.Data’ namespace. ‘DataTable’ is a data structure that holds a collection of ‘DataRow’ and ‘DataColumn’ objects.

I have opted for using the ‘DataTable’ structure as opposed to a two-dimensional array or array of tuples. This is because, throughout the various stages of regression analysis, records and fields will be removed and added, hence a 2D array would be unsuitable due to the requirement of a variable number of rows and columns. In addition, a ‘DataTable’ can handle tabular data with different data types which is vital for my project as my dataset consists of a range of data types. As well as this, it is designed to handle large amounts of data efficiently and thus would be able to store the (approximately) 20,000 records in my dataset. However, I will not need an external database as the dataset is still relatively small. The ‘DataTable’ class also allows columns to be assigned meaningful names, of which I can use the field names listed in the data dictionary, and columns can be accessed by name (as opposed to indexes). This makes it hugely beneficial for regression analysis as columns will be constantly processed in a variety of ways, and access via meaningful column names makes this much easier.

The ‘DataTable’ class consists of methods and properties that can be useful for regression analysis:

- ‘Rows’: Gets the collection of ‘DataRow’ objects.
- ‘Columns’: Gets the collection of ‘DataColumn’ objects.
- ‘NewRow()’: Creates a new row.
- ‘Select()’: Retrieves an array of ‘DataRow’ objects based on filter criteria.
- ‘ImportRow()’: Copies a ‘DataRow’ object into the table.
- ‘Clone()’: Copies the tabular structure of a table.

Also, each ‘DataRow’ has a property called ‘ItemArray’ which stores the values in a given row in an array and provides additional properties and methods to the row itself.

The table below previews how the car price dataset would be stored in a ‘DataTable’.

	ID	Price	Levy	Manufacturer	Model	…	Airbags
Indexes	0	1	2	3	4	…	16
0	45654403	13328	1399	Lexus	RX 450	…	12
1	44731507	16621	1018	Chevrolet	Equinox	…	8
2	45774419	8467	0	Honda	FIT	…	2
…	…	…	…	…	…	…	…
19237	45813273	470	753	Hyundai	Sonata	…	12

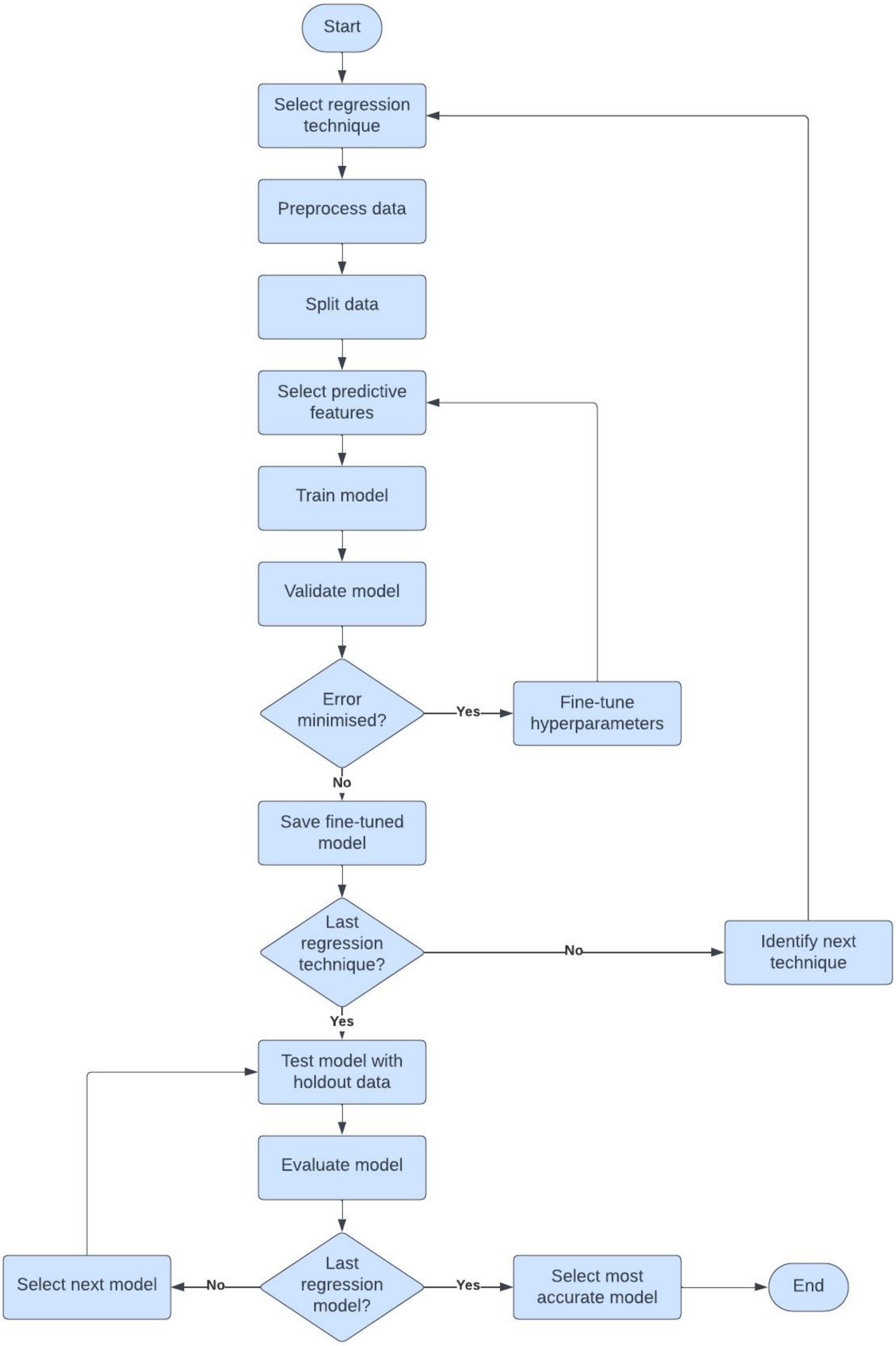
Additional data structures

In addition to the datatable, I will use other data structures for individual methods, where another data structure is suitable for completing a certain task.

Data structure	Use of data structure
Dictionary (key: string, value: list of strings)	The keys will be the car manufacturers and their values will be a list of models of the given manufacturer. The purpose of this dictionary will be to allow the model dropdown options to be updated according to the selected manufacturer. The use of this dictionary means that the model dropdown can be populated by simply referencing the dictionary with the selected manufacturer.
Dictionary (key: string, value: integer)	Another use of a dictionary will be for counting the number of each manufacturer. Hence, the keys will be the manufacturers and the values will be the count. Using the dictionary means that manufacturers with few occurrences could be changed to ‘Other’ with a selection statement by checking the value of each key-value pair.
List	I will use many lists of various data types (double, integer, string) in situations where the data structure needs to change in size. For example, I will use a list to store the names of invalid user inputs in order to display a meaningful error message. A list is necessary for this use case as the number of invalid user inputs is variable as it is dependent on what the user enters, thus a list is optimal.
Array	I will also use many arrays of various data types in situations where the data structure will be fixed in size. For example, I will use it in statistical functions like calculating the mean of an array of numbers, as well as the merge sort method.
Indexed distance	I will create a new simple data structure called ‘IndexedDistance’, containing two attributes: an index and a distance. This will be used as part of the KNN regression model implementation where distances must be sorted, whilst keeping track of the data point the distance is associated with.

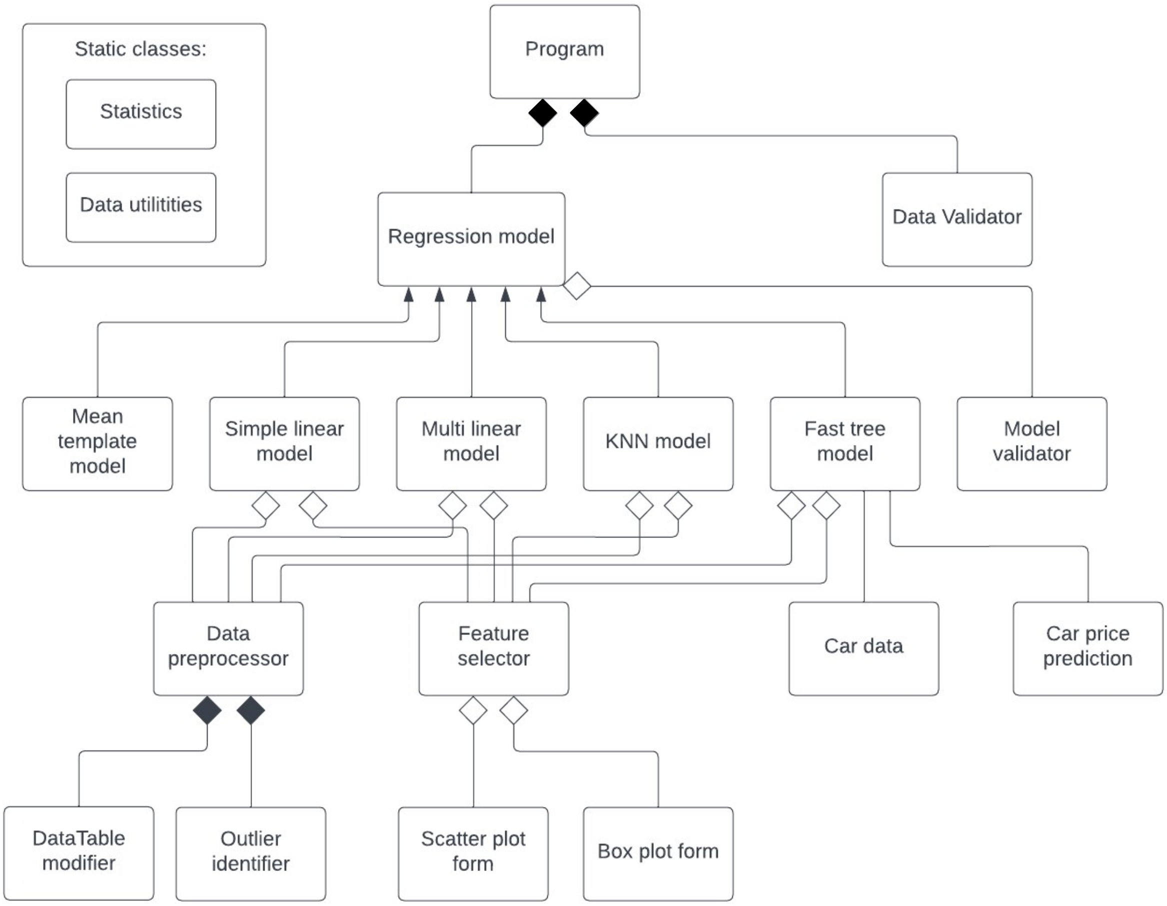
Regression analysis flowchart

The following flowchart outlines the process of regression analysis that will be conducted to identify the best regression technique for predicting the car price to integrate into the final system.



UML Class diagram

The following UML Class diagram illustrates the relationships between the various classes to be used for regression analysis.



The classes will have relationships to other classes to varying degrees such as aggregation, composition and inheritance as shown in the diagram. The Statistics and Data Utilities classes are static and will be used by lots of different classes. I will incorporate polymorphism by making the regression model class an abstract class, acting as an interface, that contains definitions of common methods, which will be implemented in the specific regression model classes. The data validator class encapsulates the initial data validation and loading of the dataset as explained in some of the pages above.

Regression analysis process

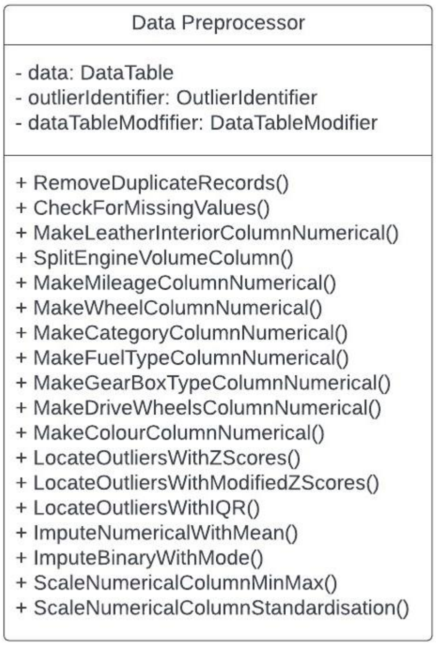
Data Preprocessing

Data preprocessing is the process of preparing, cleaning and transforming raw data such that it is suitable for model training.

This process will differ for various regression techniques however some procedures are common across all techniques like handling missing values. Therefore, I will create a data preprocessor class that contains a range of methods for the many preprocessing processes. Subsequently, different methods will be called depending on the regression technique in order for the data to be pre-processed appropriately. For example, decision tree regression does not require feature scaling whereas KNN regression requires all features to have a similar scale.

The general process I will conduct for data preprocessing is:

1. Check for and remove duplicate records.
2. Check for missing values.
3. Missing data is deleted or imputed depending on:
   i. Whether data is missing at random or in patterns.
   ii. How significant the variable is.
   iii. The amount of missing data.
4. Remove trailing strings like ‘km’ in the mileage column.
5. Engineer new features (where necessary).
   i. The engine volume field can be split into two features: a numerical variable regarding engine displacement and a binary variable for whether or not it is a turbo engine.
6. Locate and decide whether to remove or impute outliers.
7. Encode categorical variables into numerical form (where necessary).
8. Scale data to a common range to prevent variables from dominating each other.



Data Preprocessor

- data: DataTable
- outlierIdentifier: OutlierIdentifier
- dataTableModfifier: DataTableModifier

+ RemoveDuplicateRecords()
+ CheckForMissingValues()
+ MakeLeatherInteriorColumnNumerical()
+ SplitEngineVolumeColumn()
+ MakeMileageColumnNumerical()
+ MakeWheelColumnNumerical()
+ MakeCategoryColumnNumerical()
+ MakeFuelTypeColumnNumerical()
+ MakeGearBoxTypeColumnNumerical()
+ MakeDriveWheelsColumnNumerical()
+ MakeColourColumnNumerical()
+ LocateOutliersWithZScores()
+ LocateOutliersWithModifiedZScores()
+ LocateOutliersWithIQR()
+ ImputeNumericalWithMean()
+ ImputeBinaryWithMode()
+ ScaleNumericalColumnMinMax()
+ ScaleNumericalColumnStandardisation()

Techniques for imputing data

- To impute continuous variables, the mean of the observed data for the given variable will be calculated.
- To impute dichotomous variables, the mode (most frequent) of the given variable will be used.

Techniques for identifying outliers

To locate outliers for each variable (including dependent variable), there are a few methods that I may use in my project:

- Calculate the absolute Z-score for each data point (with mean) and identify data points where this is above a certain threshold (commonly 2-3).
    - This assumes a normal distribution
- Calculate the modified Z-score for each data point (with median) and similarly remove outliers.
- Calculate the interquartile range, and then the lower and upper thresholds and remove data points below the threshold or above the threshold

Techniques for encoding categorical variables into numerical form

There are many techniques that can be used to convert categorical data to numerical form for enabling a regression model to process it. As for my project, the dataset's categorical variables are entirely nominal data and none of them are ordinal data (data in ordered categories). As a result, I will use an encoding technique called One-Hot Encoding. One-Hot Encoding involves creating new binary columns for each category of a categorical variable. For example, the fuel type model column will be split into many columns called 'Is Diesel', 'Is Petrol', 'Is Hydrogen' etc.

Techniques for scaling data

The purpose of scaling is to balance the impact of all predictor variables on the target variable and is especially important for KNN (due to its distance calculation) as well as linear regression (due to the potential of biased coefficients).

There are two common scaling techniques that I may use:

- Min-Max Scaling
    - Scales data to a range between 0 and 1 but is sensitive to outliers
    - $X_{scaled} = (X - X_{min}) / (X_{max} - X_{min})$
- Standardisation
    - Scales data to have a mean of 0 and standard deviation of 1 and more robust to outliers.
    - $X_{scaled} = (X - X_{mean}) / X_{std}$

Example pseudocode methods for the Data Preprocessor class

Duplicate record check:

```
SUB RemoveDuplicateRecords(data)

     FOR EACH row in data:

          IF row not in uniqueRecords THEN:

               uniqueRecords.Add(row)

          ELSE:

               data.Remove(row)

          ENDIF

     ENDFOR

     RETURN data
```

Missing values check:

```
SUB CheckForMissingValues(data)

     FOR EACH row in data:

          FOR EACH value in row:

               IF value = NULL THEN:

                    hasMissingValue ← true

                    Print(value + ‘missing at row’ + row_index)

               ENDIF

          ENDFOR

     ENDFOR

     IF hasMissingValues = false THEN:

               Print(‘No missing values found’)

     ENDIF
```

Remove ‘km’ suffix from mileage column:

```
SUB RemoveKmFromMileage()
     FOR EACH value in mileageColumn:
          IF value.EndsWith(“km”) THEN:
               value.TrimEnd(“km”)
          ENDIF
     ENFOR
```

Calculate Z-score to identify outliers:

```
SUB LocateOutliersWithZScores(dataColumn)
     SET threshold ← 3
     FOR i in range(0 TO Length(dataColumn)-1):
          zScore ← (value – mean) / std_dev
          IF Abs(zScore) > threshold THEN:
               outliers.Add(i)
          ENDIF
     ENDFOR
     RETURN outliers
# returns list of row indices corresponding to rows with outliers
```

Min-Max Scaling:

```
SUB MinMaxScale(dataColumn)
     FOR EACH value in dataColumn:
          scaledValue ← (value – minValue) / (maxValue – minValue)
          scaledColumn.Add(scaledValue)
     ENDFOR
     data.Add(scaledColumn)
```

Merge sort

Two of the methods for outlier detection mentioned above require items to be sorted: modified z-scores require the median to be calculated (which requires items to be sorted) and the interquartile range method also requires items to be sorted to determine the quartiles. Hence, I will implement the merge sort algorithm to order an array of values in ascending order.

The merge sort algorithm is a divide and conquer algorithm and works by recursively breaking down the array into two sub-arrays, sorting each sub-array and then merging the two sub-arrays.

The following pseudocode subroutines demonstrate how the merge sort will be implemented.

```
SUB MergeSort(array)

      IF Length(array) <= 1 THEN:

            RETURN

      ENDIF

      mid ← Length(array) / 2

      leftArr ← Copy(array, 0 TO mid - 1)

      rightArr ← Copy(array, mid TO Length(Array) - 1)

      MergeSort(leftArr)

      MergeSort(rightArr)

      Merge(array, leftArr, rightArr)
```

```
# Merges elements from the two arrays in ascending order
SUB Merge(array, leftArr, rightArr)
    i,j,k ← 0
    WHILE i < Length(leftArr) AND j < Length(rightArr):
        IF leftArr[i] <= rightArr[j] THEN:
            array[k] ← leftArr[i]
            i ← i + 1
        ELSE:
            array[k] ← rightArr[j]
            j ← j + 1
        ENDIF
        k ← k + 1
    ENDWHILE
    # Copy remaining values into array (if any)
    WHILE i < Length(leftArr):
        array[k] ← leftArr[i]
        i ← i + 1
        k ← k + 1
    ENDWHILE
    WHILE j < Length(rightArr):
        array[k] ← rightArr[j]
        j ← j + 1
        k ← k + 1
    ENDWHILE
```

Splitting data

The process of splitting data into training data and test data is fundamental for most machine learning algorithms and is an essential step for evaluating the performance of a model and its ability to generalise.

As for my project, I will split the dataset into three sets: a larger training dataset, and two smaller sets: validation data and holdout data.

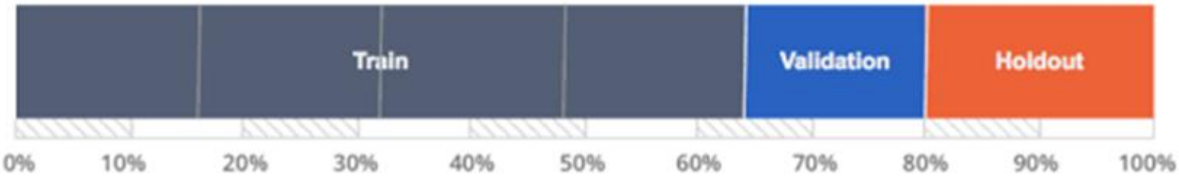
The purpose of training data is to train the regression model to develop a relationship between the predictor variables and the dependent variable. This dataset is typically the largest because regression algorithms require a lot of data to model accurate relationships.

Validation data is a form of test data, and its purpose is to test how well the model can generalise to unseen data. The predictor variables from the validation data are given to the trained model in order to predict the dependent variable; these predictions are compared with the actual values provided by the validation dataset. Error metrics can be calculated from this, and the model can be altered to reduce this error.

Holdout data acts as a final test that the model has not seen during the training and validation process. Its purpose is to provide a completely independent and unbiased evaluation of the model's performance and best indicates how the model will perform in real-life scenarios. Holdout data also addresses the potential issue of overfitting, where the model has adapted to perform so well on the training and validation data but fails to generalise to unseen data.

I will use training data for feature selection processes and will use it in unison alongside validation data in order to fine-tune each regression model to optimise its performance. Holdout data will then be used once all the regression models have been tuned and will be used for the final step to determine which model is the best and will be integrated into the final system.

I plan to split the data into ratios similar to the diagram below.



Taken from https://www.datarobot.com/wiki/training-validation-holdout/

Feature Selection

Feature selection is the process of choosing relevant and significant features (predictor variables) from the original set of features in the dataset. The goal is to improve the model's performance by reducing the dimensionality of the data and removing features that have little impact on the target variable.

There is a wide range of methods for aiding the process of choosing the most optimal features, but in my project, I will limit to the following:

- Calculating correlation coefficients (for continuous and binary variables)
    - Pearson correlation coefficients for ranking the significance of continuous variables.
    - Point biserial correlation coefficients for ranking the significance of dichotomous variables.
- Visualising data
    - Scatter plots for visualising the correlation between continuous predictors and the dependent variable (car price).
    - Box plots for visualising the distribution of the continuous predictor variables.

Pseudocode for calculating Pearson correlation coefficients:

```
SUB PearsonCorrelation(X,Y)

    n ← Length(X)

    FOR i in range(0 TO n-1):

        differenceX ← X[i] - meanX

        differenceY ← Y[i] - meanY

        productOfDifferences ← differenceX * differenceY

        sumOfProducts ← sumOfProducts + productOfDifferences

        sumOfSqrDifferencesX ← sumOfSqrDifferencesX + differenceX^2

        sumOfSqrDifferencesY ← sumOfSqrDifferencesY + differenceY^2

    ENDFOR

    pearsonCoeff ← sumOfProducts / Sqrt(sumOfSqrDifferencesX * sumOfSqrDifferencesY)

    RETURN pearsonCoeff
```

Data Visualisation using the OxyPlot library

Throughout the data preprocessing and feature selection processes, the ability to visualise the data through charts and graphs is very beneficial. To do this, I will use the OxyPlot library.

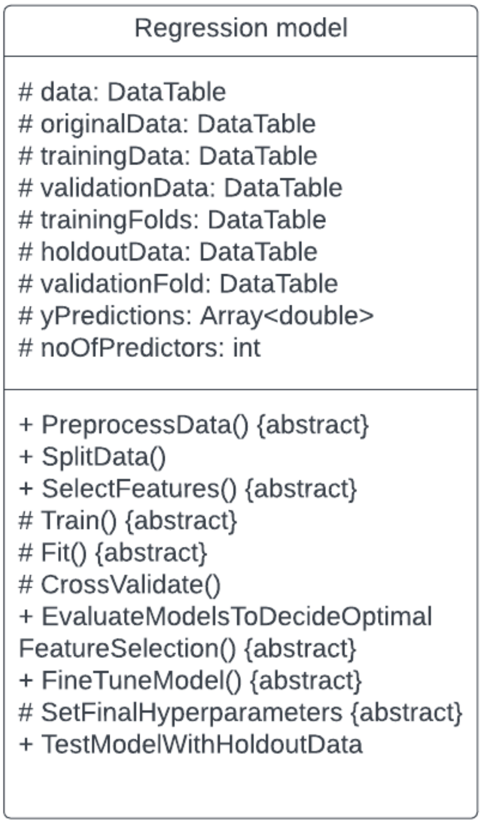
OxyPlot is a powerful and flexible open-source plot generation library based on .NET that can handle various types of charts and customisations.

OxyPlot has the following features that will be useful for my application:

- XY (horizontal and vertical axes) and cartesian (same scale on X and Y axis) plot types
- Custom annotations
- Box plots – for visualising skew and distribution of data
- Scatter plots – for visualising the correlation of data
- The ability to customise colours, axes and plot mark sizes

Regression models

All regression models will follow a similar structure and below shows the regression model abstract class that defines the common regression model methods. The implementation for the abstract methods will be provided in the derived sub classes corresponding to each regression model.



Regression model

data: DataTable
originalData: DataTable
trainingData: DataTable
validationData: DataTable
trainingFolds: DataTable
holdoutData: DataTable
validationFold: DataTable
yPredictions: Array<double>
noOfPredictors: int

+ PreprocessData() {abstract}
+ SplitData()
+ SelectFeatures() {abstract}
# Train() {abstract}
# Fit() {abstract}
# CrossValidate()
+ EvaluateModelsToDecideOptimal FeatureSelection() {abstract}
+ FineTuneModel() {abstract}
SetFinalHyperparameters {abstract}
+ TestModelWithHoldoutData

Simple Linear regression model

I will program the simple linear regression model training from scratch using the closed-form solution where the regression coefficients can be computed using mathematical formulae, without the need for more complex optimisation algorithms like gradient descent.

The formulae are derived from minimising the mean square error, which can be written as a function of m and b (the gradient and intercept coefficients) by writing the predicted y value in the form y = mx + b. The minimum point of the parabola is located by taking the partial derivatives with respect to m and b and setting them to zero. From there, the equations can be algebraically manipulated to get the following formulae:

- m = cov(X, Y) / var(X)
- b = mean(Y) – m * mean(X)

Pseudocode for training a simple linear regression model and obtaining the predicted values:

```
SUB Train()
    xTrain ← GetXTrain()
    yTrain ← GetYTrain()
    m ← Covariance(xTrain, yTrain) / Variance(xTrain)
    b ← Mean(yTrain) - m * Mean(xTrain)
    RETURN (m, b)
# covariance, variance and mean will all be methods belonging to the
statistics class


SUB Fit()
    FOR EACH value in xTest:
        yPredict ⟵ m * xTest + b
        yPredictions.Add(yPredict)
    ENDFOR
# yPredictions will be an attribute of the simple linear regression
model class and would then be used to calculate error metrics for
model validation
```

Multi linear regression model

Like simple linear regression, multi linear regression involves modelling linear relationships between multiple independent variables and the target variable. I will program the training of a multivariate regression model from scratch using a gradient descent optimisation algorithm.

Gradient descent works by minimising the cost function through updating the weights and the intercept in the direction of the steepest descent of the function. It takes larger steps where the gradient is steep and smaller steps where the gradient is flatter, in order to efficiently locate the minimum point. The cost function, in this case, will be the Mean Square Error. The weights are the linear coefficients and there is one weight for each feature.

Gradient descent is not the only method of generating a multi-linear regression model. The absolute minimum of the cost function can be found by setting each partial derivative to zero. This leaves a series of equations which can be solved using an n by n (where n is the number of linear coefficients) inverse matrix, providing the optimal weights and intercept. However, when n is large, this inverse matrix calculation can be very computationally intensive. Therefore, I have opted for the use of gradient descent as it is a more common approach in real-life scenarios when there are many predictors.

The gradient descent process is outlined below.

1. Initialise the learning rate and convergence threshold hyperparameters.
2. Initialise each weight and the intercept with a starting value (usually 0).
3. Calculate the partial derivatives of the mean square error with respect to the weights.
4. Calculate the partial derivative of the mean square error with respect to the intercept.
5. Update the weights and the intercept by subtracting the product of the learning rate and the corresponding partial derivative.
6. Repeat steps 3 to 5 until either the maximum number of iterations have been complete or the convergence criteria have been met.

The convergence criteria I will implement will be that the change in the mean square error will have to be sufficiently small (below the threshold).

The following hyperparameters can be changed to fine-tune the model are:

- Learning rate
    - A larger learning rate can lead to faster convergence but can also risk skipping the optimal solution, which, in extreme cases, can lead to the algorithm to progressively take steps away from the minimum and diverge.
    - A smaller learning rate may need too many iterations to converge to the optimal solution.
- Convergence threshold
    - A larger threshold will mean the algorithm will require less iterations but could not lead to the optimal solution with sufficient accuracy.

Gradient descent pseudocode

This ‘TrainGradientDescent()’ subroutine will belong to the Multi Linear regression model class. This class will have a few attributes including weights, intercept, learningRate, maxNoOfEpochs and features. The function iterates up to a maximum number of epochs depending on when (if ever) the convergence criteria is met. In each epoch, it predicts target values using the current weights and intercept, calculates the current cost function and calculates partial derivatives of the cost function with respect to the weights and the intercept. Next, the weights and intercept are updated by subtracting the product of the learning rate and the respective partial derivative. This process is repeated at each iteration until the difference in the cost function is below the convergence threshold.

```
SUB TrainGradientDescent()

    # initialise weights and intercept

    # define learning rate, max number of epochs and convergence threshold

    FOR epoch in range(0 TO maxNoOfEpochs):

        yPredicts ← Predict()

        currentCost ← CalculateCost(yPredicts)

        IF Abs(previousCost - currentCost) < threshold THEN:

            BREAK

        ENDIF

        weightDerivatives ← CalculateWeightDerivatives(yPredicts)

        interceptDerivative ← CalculateInterceptDerivative(yPredicts)

        FOR i in range(0 TO Length(Weights)):

            weights[i] ← weights[i] - learningRate * weightDerivatives[i]

        ENDFOR

        intercept ← intercept - learningRate * interceptDerivative

        previousCost ← currentCost

    ENDFOR
```

The following ‘Predict()’ method is responsible for predicting target values. It iterates through each data point and initialises the predicted target value variable with the intercept and then iterates through each predictor variable’s coefficient and adds the product of the predictor value for the given data point and the corresponding weight to the predicted target value. An array of predicted values is then returned.

```
SUB Predict()

    FOR EACH row in trainingData:

        yPred ← intercept

        FOR i in range(0 TO Length(weights)-1):

            yPred ← yPred + weights[i] * row[i]

        ENDFOR

        yPredicted.Add(yPred)

    ENDFOR

    RETURN yPredicted
```

The following ‘CalculateCost()’ method calculates the mean square error when the current model is used to predict the dependent variable. It iterates through each value in the yPredicts array and calculates the residual to sum the squares of the residuals, and then divided this sum by the number of data points.

```
SUB CalculateCost(yPredicts)

    FOR i in range(0 TO Length(yPredicts)-1):

        residual ← y[i] - yPredicts[i]

        sumOfSqrResiduals ← sumOfSqrResiduals + residual^2

    ENDFOR

    RETURN (sumOfSqrResiduals / Length(yPredicts))
```

The ‘CalculateWeightDerivatives()’ method calculates the partial derivatives of the mean square error with respect to each weight. It iterates through each feature and then through each data point to calculate the sum of products of the residual and the value of the feature, before calculating the derivative for each feature/weight. An array containing the partial derivative with respect to each weight is then returned.

```
SUB CalculateWeightDerivatives(yPredicts)

    noOfDataPoints ← Length(trainingData.Rows)

    FOR EACH feature in features:

        FOR i in range(0 TO noOfDataPoints):

            residual ← y[i] – yPredicted[i]

            sumOfProducts ← trainingData.Rows[i][feature] * residual

        ENDFOR

        partialDerivatives.Add(-(2/noOfDataPoints) * sumOfProducts)

    ENDFOR

    RETURN partialDerivatives
```

The following ‘CalculateInterceptDerivative()’ method similarly calculates the partial derivative but with respect to the intercept. It iterates through each data point in order to calculate the sum of residuals. The partial derivative is then returned.

```
SUB CalculateInterceptDerivative(yPredicts)

    noOfDataPoints ← Length(trainingData.Rows)

    FOR i in range(0 TO noOfDataPoints):

        residual ← y[i] – yPredicted[i]

        sumOfResiduals ← sumOfResiduals + residual

    ENDFOR

    RETURN (-(2/noOfDataPoints) * sumOfResiduals)
```

KNN Regression model

I will program the basic K-nearest neighbours regression model implementation from scratch, where target values are predicted by calculating the mean of the k-nearest training data points. The value of K will be a hyperparameter that will need tuning to find the optimal value. KNN regression does not have a training phase as it does not establish any complex mathematical relationships using the training data. Instead, the training data will simply be used in the process of locating the neighbours and calculating the mean to predict the target values for the validation data which will then be compared with the actual target variable values.

KNN pseudocode

The following subroutine iterates through each data point in the validation data and calculates the distance to each of the training data points. The smallest k distances are sorted in descending order then the k nearest neighbours are chosen. Next, the mean of the corresponding target values is calculated as the predicted value. This process is repeated for each validation data point and a list of predicted values is returned. The pseudocode assumes the existence of a nested class called IndexedDistance (mentioned in the additional data structures section) which is used to sort the distances in descending order whilst keeping track of the row index.

```
SUB Fit()

    # initialise k value

    FOR EACH queryPoint in validationData.Rows:

        FOR i in range(0 TO Length(trainingData.Rows)):

            distance ← CalculateDistance(queryPoint, dataPoint)

            indexedDistance ← (i, distance)

            distances.Add(indexedDistance)

        ENDFOR

        BubbleSortDescending(distances)

        kIndex ← 0

        FOR j in range(Length(distances)-1 TO Length(distances)-k):

            kNeighbours[kIndex] ← distances[j].Index

            kIndex ← kIndex + 1

        ENDFOR
        FOR EACH neighbour in kNeighbours:
            ySum ← ySum + trainingData.Rows[neighbour]["Price"]
        ENDFOR
        yPredictions.Add(ySum / k)
    ENDFOR
    RETURN yPredicted
```

The following subroutine calculates the Euclidean distance between two data points by iterating through each dimension to sum the squares of differences. This is then rooted to result in the Euclidean distance. This pseudocode assumes the existence of the attribute 'features' belonging to the KNN regression model class.

```
SUB CalculateDistance(dataPoint1, dataPoint2)
    FOR EACH feature in features:
        difference ← dataPoint1[feature] - dataPoint2[feature]
        squaredSum ← squaredSum + difference^2
    ENDFOR
    RETURN sqrt(squaredSum)
```

The subroutine below uses a modified version of the bubble sort algorithm to perform k passes in order to sort the k smallest values at the end of the array. Whilst the bubble sort is an inefficient algorithm with time complexity O(n^2), I am not using it to sort the whole list, just the smallest k items. Because it does not require n passes (only k passes, where k is small), bubble sort is most suitable as opposed to merge sort or quicksort. This is because the entire list does not require sorting and using a method like merge sort would mean n-k items are redundantly sorted.

```
SUB BubbleSortDescending(distances)
    FOR i in range(0 TO k-1):
        FOR j in range(0 TO Length(distances)-i-2):
            IF distances[j].Distance < distances[j+1].Distance
THEN:
                temp ← distances[j]
```

```
                    distances[j] ← distances[j+1]
                    distances[j+1] ← temp
               ENDIF
          ENDFOR
     ENDFOR
     RETURN distances
```

Fast tree regression model

To implement a decision tree regression model, I will use the Fast Tree regression trainer provided by the ML.NET library and the Microsoft.ML.Trainers.FastTree namespace.

FastTree details:

- Input label column data must be of the Single data type.
- Input data does not have to be normalised.
- Trainer outputs a score column which contains the predicted scores (car prices, in my case)
- An efficient implementation of a gradient boosting algorithm.
- Builds each regression tree in a stepwise fashion, using a predefined loss function to measure the error for each error and corrects it for the next.
- Series of such trees are created and the optimal one is selected.

In order to implement the fast tree regression trainer, I will create two classes: car price data - which holds the input feature data, and car price prediction which contains an attribute for the score column - where the model will output its predicted car prices.

Model validation

Model validation is the process of evaluating how well the model performs on unseen validation data. The technique I will use for model validation is K-fold cross-validation due to its reduced bias, better utilisation of data and robustness. It ensures every data point is used in the test set exactly once and is less sensitive to the specific partitioning of data.

K-fold cross-validation involves dividing the dataset into k number of folds (subsets) and the model is trained and evaluated k times. Each time, one of the k subsets is used as the validation fold and the other k-1 folds are combined to form the training set. For each iteration, the model’s performance is measured using error metrics and then the average error is computed.

The CrossValidate() method will belong to the Regression Model abstract class, which will also define the trainingFolds and validationFold attributes, vital for the cross-validation process.

Example pseudocode for cross-validation

```
SUB CrossValidate()

    k ← 5

    dataSubsets ← DivideDataInto5(data)

    FOR i in range(0 TO k-1):

        FOR j in range(0 TO k-1):

            IF j = i THEN:

                validationFold.Merge(dataSubsets[j])

            ELSE:

                trainingFolds.Merge(dataSubsets[j])

        ENDFOR

        Train()

        Fit()

        maeSum ← maeSum + CalculateMAE() # example error metric

    ENDFOR

    mae ← maeSum / k

    RETURN mae
```

Below are the error metrics I will use and create methods for in the Model Validator class:

- Mean absolute error
- Mean square error
- Root mean square error
- R-squared
    - A coefficient ranging from 0 to 1, 1 being perfect, 0 being as good as using a horizontal line (the mean) to predict.
- Adjusted R-squared
    - Similar to R-squared but penalises the addition of useless predictors.

The Model Validator class will also define two 1-dimensional double array attributes: yPred and yActual.

Here is some example pseudocode for calculating the mean absolute error:

```
SUB CalculateMAE()

    FOR i in range(0 TO Length(yActual)-1):

        residual ← yActual - yPredicted

        sumOfResiduals ← sumOfResiduals + Abs(residual)

    ENDFOR

    mae ← sumOfResiduals / Length(yActual)

    RETURN mae
```

Hyperparameter fine-tuning

Hyperparameter fine-tuning is the process of searching for the best combination of hyperparameters to optimise model performance. There are a few different methods for hyperparameter fine-tuning; the one I will use is grid search. I will use grid search due to its computational efficiency and simplicity, however it may not provide the absolute most optimal hyperparameter combination.

Each regression model will have different hyperparameters, but the general grid search process will be the same.

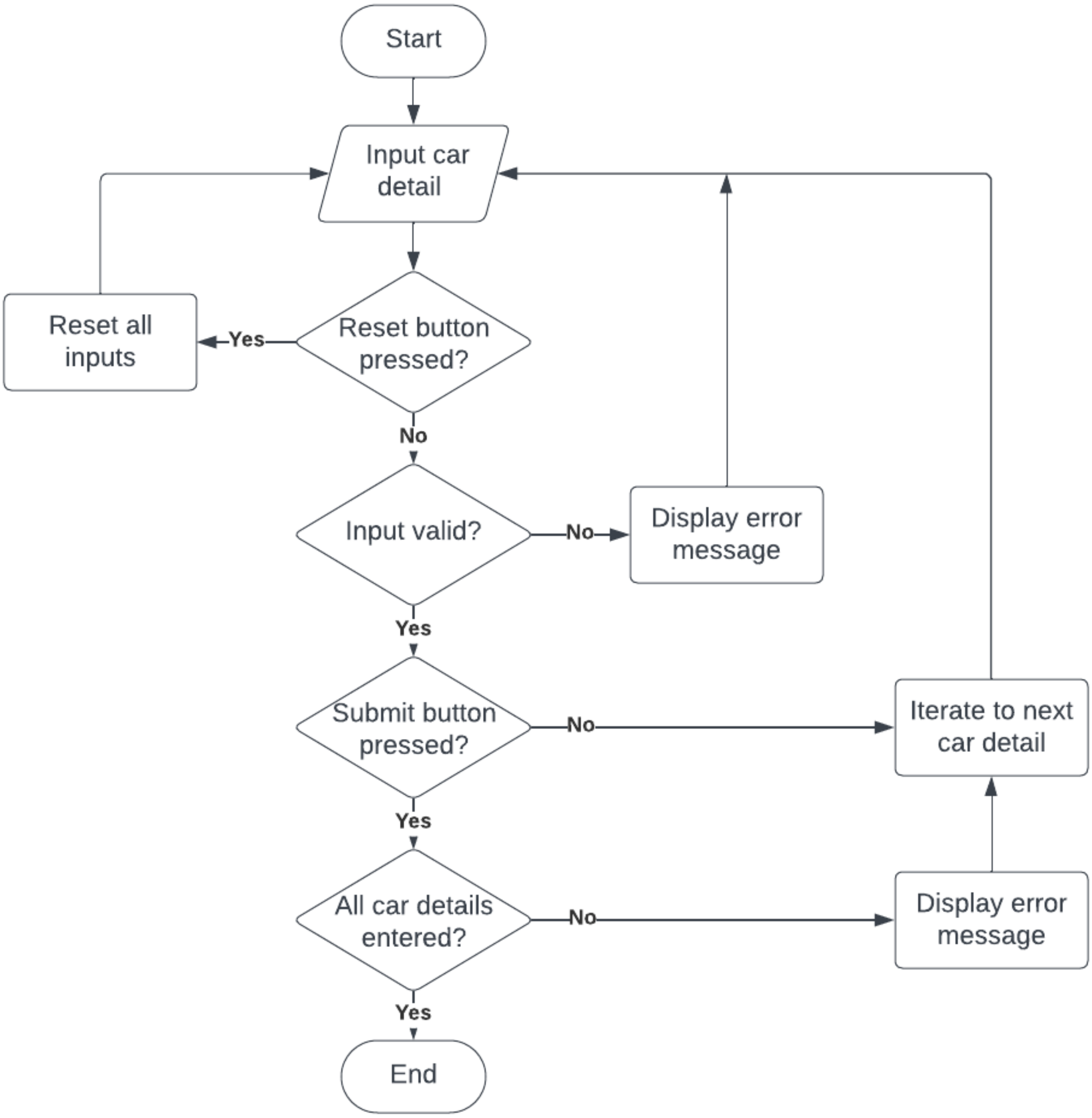
Grid search process:

1. Define a set of possible values to try for each hyperparameter
2. Run the model with each hyperparameter value.
3. Obtain the error metrics after each iteration of training and fitting the model.
4. Select the hyperparameter value with the best error metrics.

User interface and final system design

After obtaining the best regression model, this model will be integrated into a final system that will allow the user to have the price of their car estimated.

The flowchart below outlines the process of inputting the user's car details, in order to estimate the price of their car.

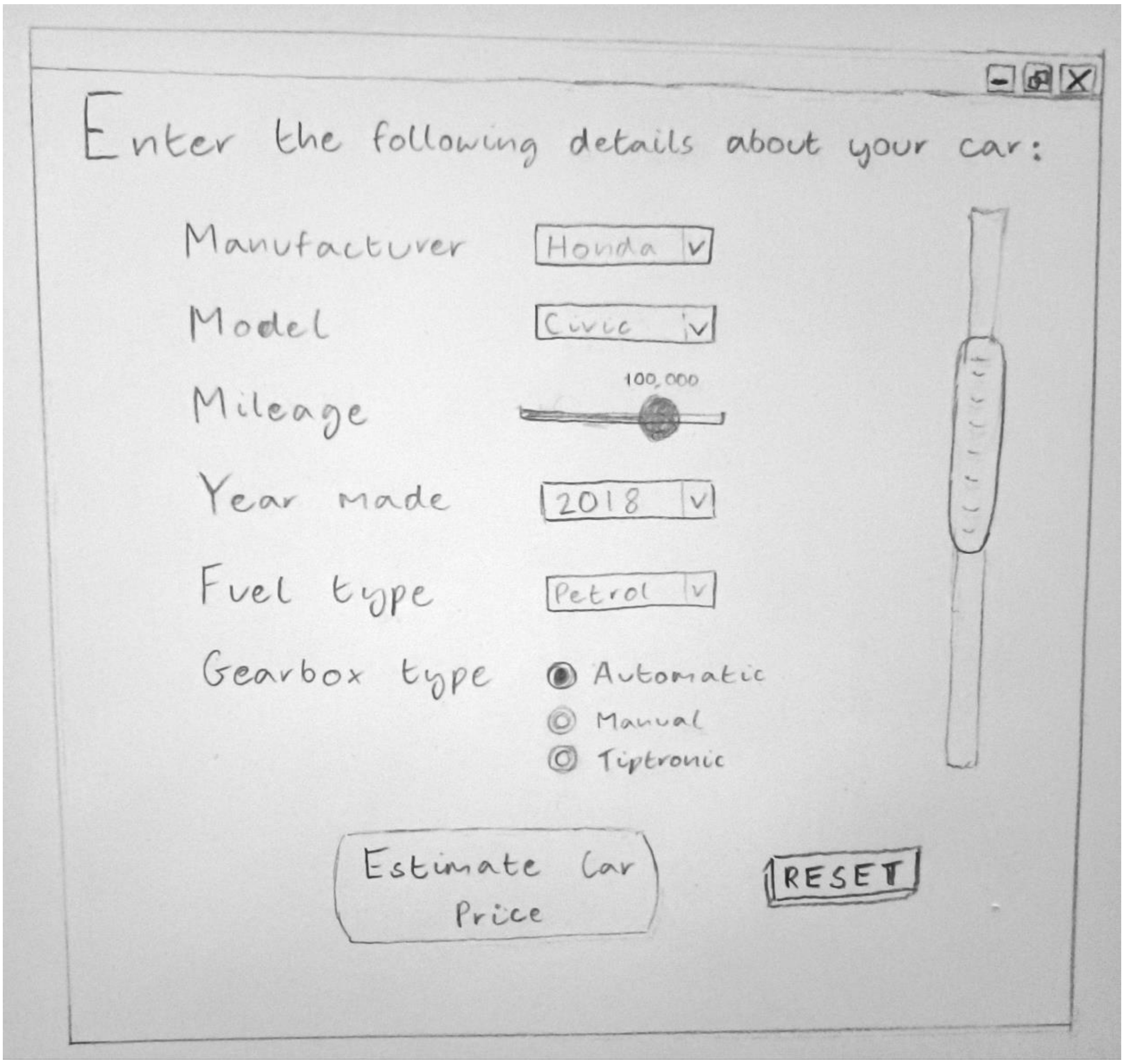


Initial UI Design

Initial drawings:

These drawings are initial prototypes outlining how the interface of the system should look like, providing a starting point in order to implement a final design of the user interface.

The drawing below shows the page containing a form for the user to input their car details. It outlines how dropdowns can be used for the manufacturer and model, and any other inputs that have discrete options. Also, sliders can be used for continuous numerical inputs and radio buttons for car details that have very few discrete options (e.g. gearbox type). The submit button is shown alongside the reset button.



Enter the following details about your car:

Manufacturer: Honda v

Model: Civic v

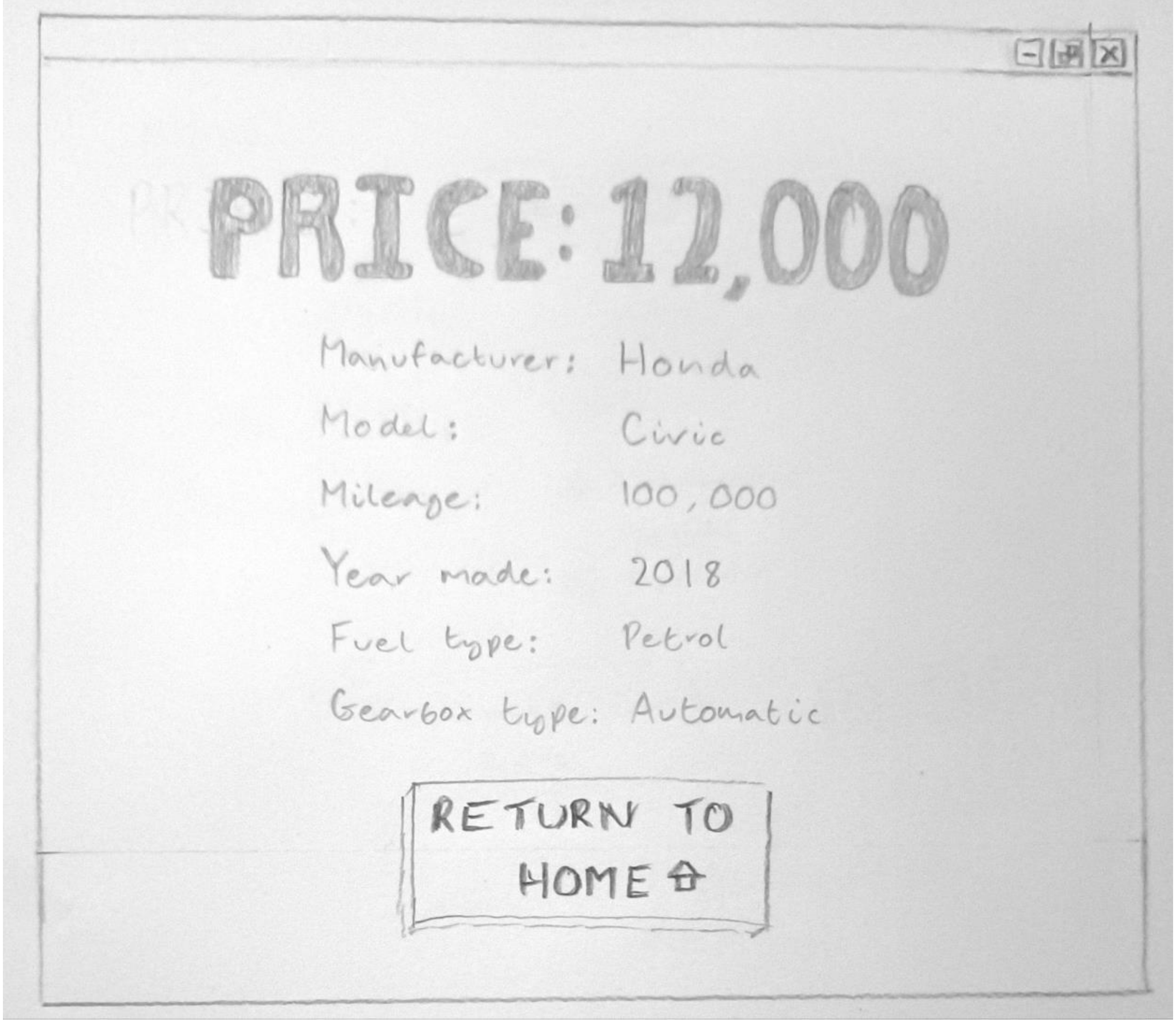
Mileage: 100,000

Year made: 2018 v

Fuel type: Petrol v

Gearbox type: ● Automatic ○ Manual ○ Tiptronic

This second drawing is a design for the page where the user’s car price estimation is displayed. It outlines how the price should be shown in bold lettering. Also, the user’s car details are displayed in smaller lettering, providing confirmation that the user has correctly entered the details and that it is their car being valued. The return button is shown below as well, which allows the user to return to the form page and get another price estimation.



PRICE: 12,000

Manufacturer: Honda

Model: Civic

Mileage: 100,000

Year made: 2018

Fuel type: Petrol

Gearbox type: Automatic

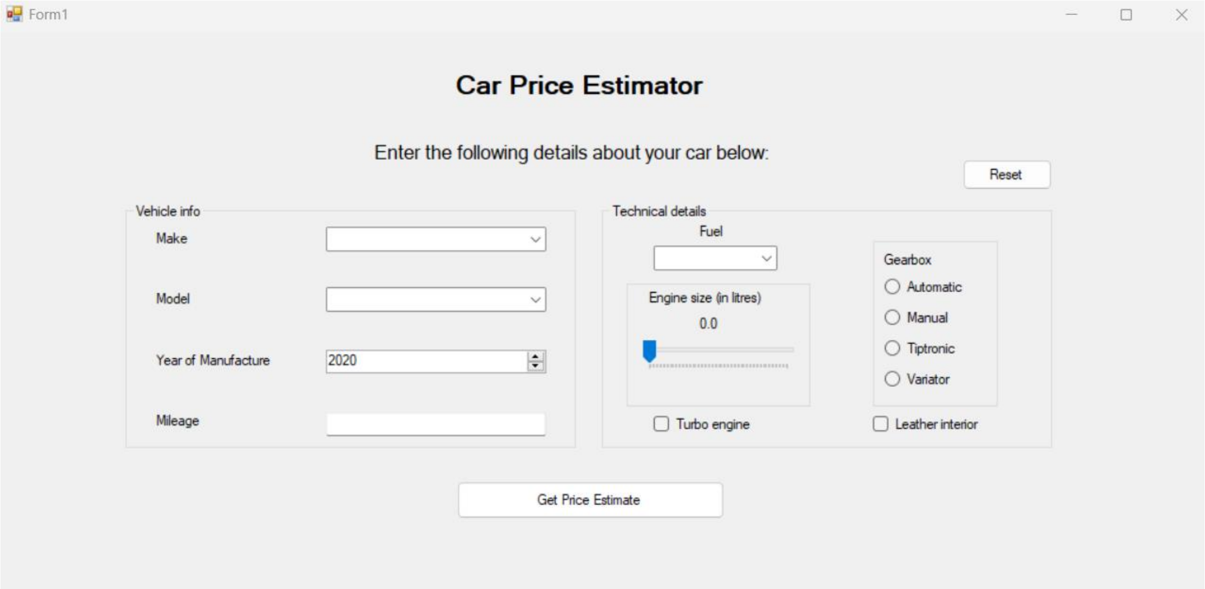
Using Windows Forms to develop the GUI

The GUI will be developed using Windows Forms, which is a graphical user interface framework provided by Microsoft that can be used to aid the creation of interactive desktop applications. When using a development environment like Visual Studio, Windows Forms allows you to create smart-client apps that display information, request input from users, and communicate with remote computers over a network.

Windows Forms contains a variety of controls (elements that display data or accept data input) that can be added to forms: text boxes, buttons, drop-down boxes, and radio buttons. The drag-and-drop Windows Forms Designer in Visual Studio allows you to select controls you want and place them wherever on the form.

Visual Studio Form design:

Home page



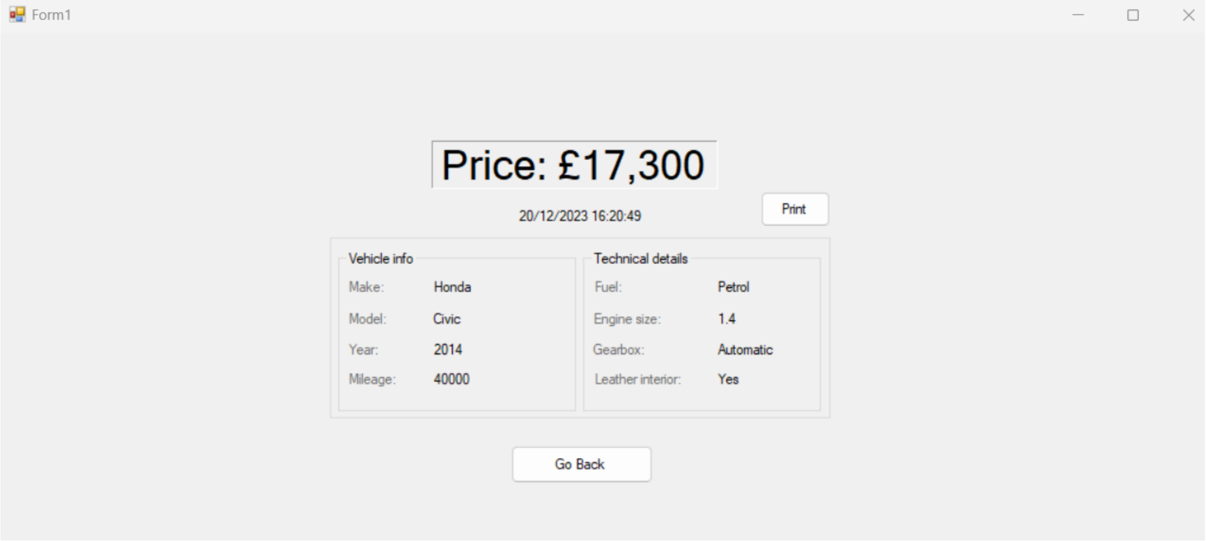
The home page design consists of a range of user inputs organised in two sections: vehicle information and technical details.

The vehicle information container comprises dropdown selection menus for the make and model. When a make is selected, the model dropdown will update so that only the applicable models for the given make are displayed. The input type for the year of manufacture is a number incrementor, as this limits the need for user validation as well as being convenient for the user. For mileage, the input can take a wide range of values, hence a text input is most suitable.

The technical details container comprises a dropdown, a slider, a set of radio buttons and two checkboxes for binary inputs. The choice to use a dropdown or radio buttons depends on the number of possible options, and the slider is useful when the input has a small range.

The reset button is located in the top right of the window, providing the user with an option to reset the inputs.

Output page



Price: £17,300

20/12/2023 16:20:49

Vehicle info

Make: Honda

Model: Civic

Year: 2014

Mileage: 40000

Technical details

Fuel: Petrol

Engine size: 1.4

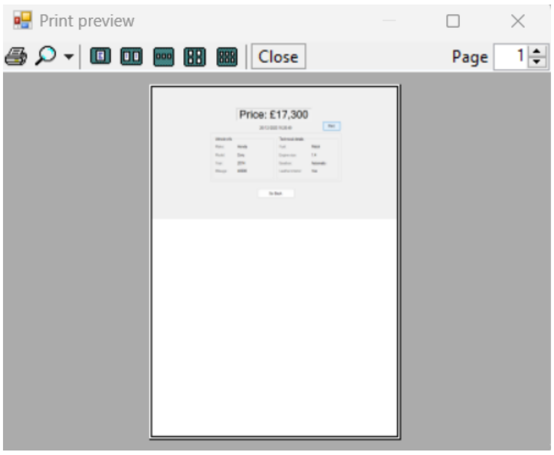
Gearbox: Automatic

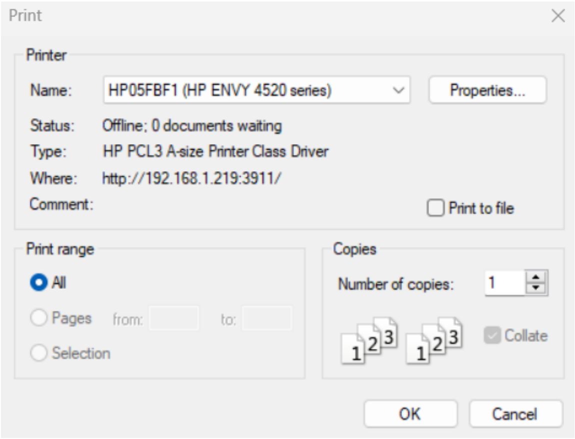
Leather interior: Yes

The output page design comprises a large display of the price estimation to the nearest hundred pounds and simple layout showing the user’s car details. The car details are similarly organised to the layout and order of the equivalent inputs in order to provide user clarity.

The timestamp is a useful addition for the user as the value of a car changes over time, therefore, the timestamp confirms when the car valuation was made. The print button allows the user to print off the output page to have a paper copy of the price estimation.

After pressing the print button, below are the windows form designs for the print preview and print dialog.





Printer

Name: HP05FBF1 (HP ENVY 4520 series)

Status: Offline; 0 documents waiting

Type: HP PCL3 A-size Printer Class Driver

Where: http://192.168.1.219:3911/

Comment:

□ Print to file

Print range

● All

○ Pages from: to:

○ Selection

Copies

Number of copies: 1

☑ Collate

Structure of the final system file

The windows form .cs file will encapsulate many methods that each correspond to a given event. Each method is triggered in response to the specific event. For example, I will incorporate a method called submitButton_Click() which will comprise some logic for validating that all inputs have been entered correctly. Alternatively, a method will be needed for when the user attempts to select a car model from the dropdown to ensure that the user has already selected a car make - below shows a pseudocode example for this.

```
SUB modelDropdown_dropdown(Events)

     IF makeDropdown.SelectedItem() = null THEN:

          ShowError("Make must be selected before model.")

     ENDIF
```

Uses of files

In my project, I will use a variety of files for a range of uses:

- The car price data and the corresponding data dictionary used for the regression model is saved locally (in the Debug folder) as a csv file, such that the data can be easily accessed and loaded into a data table at run-time.
- After the regression models have been fine-tuned, the models will be tested with the holdout data, and each model's performance will be written to a text file at run-time such that the performance of each model can be easily compared.
- The best performing model will be trained and then saved as a zip file, in order for the price estimation process in the final system to be quick and efficient, as the trained model can then be loaded at run-time and then it can predict the car price.
- In order for the make and model dropdowns to be populated, data is required from the original car price dataset used to generate the model. Hence, the makes and models are stored as a JSON file, which is then read at system run-time to populate the make and model dropdowns.

Technical Solution

My technical solution consists of two projects, the Regression Analysis Console application project and the Final System Windows Forms application. I have copied the regression analysis project code first and the windows forms UI project code immediately after.

The regression analysis project is split into around 20 different files, and I have given a sub-heading for each of these files, to indicate what file the code below belongs to. All the code in this regression analysis project was coded by me (none is package-generated).

The Windows Forms UI project consists of the Form1.cs file, which contains code written by me, and Form1.Designer.cs file, which comprises entirely package-generated code. The package-generated code is positioned at the end of the Technical Solution, clearly indicated by a sub-heading with capital letters.

RegressionAnalysisProj

Program.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;

namespace RegressionAnalysisProj
{
    internal class Program
    {

        static void Main(string[] args)
        {
            DataValidator dv = new DataValidator("train.csv", "data-
dictionary.csv");
            dv.ValidateData();

            DataTable originalData = dv.GetData();
            DataTable data1 = originalData.Copy();
            DataTable data2 = originalData.Copy();
            DataTable data3 = originalData.Copy();
            DataTable data4 = originalData.Copy();
            DataTable data5 = originalData.Copy();

            RegressionModel[] models = new RegressionModel[5];
            models[0] = new MeanTemplateModel(data1);
            models[1] = new SimpleLinearRegressionModel(data2);
            models[2] = new MultiLinearRegressionModel(data3);
            models[3] = new KNNRegressionModel(data4);
            models[4] = new FastTreeRegressionModel(data5);


            StreamWriter sw = new StreamWriter("regression-analysis-
conclusion.txt");

            foreach (RegressionModel model in models)
            {
                model.PreProcessData();
                model.SplitData();
                model.SelectFeatures();
                model.EvaluateModelsToDecideOptimalFeatureSelection();
                string modelConclusion = model.FineTuneModel();
                sw.WriteLine(modelConclusion);
                sw.WriteLine();
            }
            sw.Close();

            /** Testing with holdout data **/

            List<string> holdoutDataConclusion = new List<string>();

            foreach (RegressionModel model in models)
            {
                model.PreProcessData();
                model.SplitData();
                string line = model.TestModelWithHoldoutData();
                holdoutDataConclusion.Add(line);
            }

            Console.WriteLine("Final model testing with holdout data:");
            Console.WriteLine($"{"Mean teplate model",-22}
{holdoutDataConclusion[0]}");
            Console.WriteLine($"{"Simple linear model",-22}
{holdoutDataConclusion[1]}");
            Console.WriteLine($"{"Multi linear model",-22}
{holdoutDataConclusion[2]}");
            Console.WriteLine($"{"KNN model",-22} {holdoutDataConclusion[3]}");
            Console.WriteLine($"{"Fast tree model",-22}
{holdoutDataConclusion[4]}");
            Console.ReadLine();

            /** Best performing model **/

            FastTreeRegressionModel treeModel = new
FastTreeRegressionModel(originalData);
            treeModel.PreProcessData();
            treeModel.SplitData();
            treeModel.DetermineHyperparameterWithHoldoutData();
            treeModel.SaveTrainedModelToFile();
            treeModel.SaveMakeAndModelToFile();

            Console.ReadLine();
        }

    }
}
```

DataValidator.cs

```
using System;
using System.Data;
using System.IO;
using System.Text.RegularExpressions;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the initial validation of the dataset
    public class DataValidator
    {
        private DataTable data;
        private DataTable dataDict;
        private bool fileReadFailed;

        public DataValidator(string dataFilename, string dataDictFilename)
        {
            ReadDataDictionaryFile(dataDictFilename);
            ReadDataFile(dataFilename);
        }

        public DataTable GetData()
        {
            return data;
        }

        // Validates data using the data dictionary
        public void ValidateData()
        {
            if (!fileReadFailed)
            {
                Console.WriteLine();
                Console.WriteLine("Checking data validity...");
                for (int i = 0; i < data.Columns.Count; i++)
                {
                    string regExPattern = dataDict.Rows[i].Field<string>("Regular
expression pattern");
                    foreach (DataRow row in data.Rows)
                    {
                        bool valid = ValidateDataEntry(row[i].ToString(),
regExPattern);
                        if (!valid)
                        {
                            // Some special characters in the data have a
character encoding issue so are displayed as '?'
                            // ?s are replaced with _ so it will be known in the
later stages of regression analysis that this data has some lost characters
                            Console.WriteLine(String.Format("Invalid: {0,-30}
{1,-5} {2,-5}", row[i], data.Rows.IndexOf(row).ToString(), i.ToString()));
                            Char[] temp = row[i].ToString().ToCharArray();
                            for (int j = 0; j < temp.Length; j++)
                            {
                                if (char.ConvertToUtf32(row[i].ToString(), j) >
255)

                                {
                                    temp[j] = '_';
                                }
                            }
                            string updatedValue = new string(temp);
                            row[i] = updatedValue;
                            Console.WriteLine("Updated: " + row[i]);
                            Console.WriteLine();
                        }
                    }
                }

                // Final check after updated data entries
                Console.WriteLine("Confirming data validity...");
                int invalidCount = 0;
                for (int i = 0; i < data.Columns.Count; i++)
                {
                    string regExPattern = dataDict.Rows[i].Field<string>("Regular
expression pattern");
```

```
                    foreach (DataRow row in data.Rows)
                    {
                        bool regExValid = ValidateDataEntry(row[i].ToString(),
regExPattern);
                        if (!regExValid)
                        {
                            string newPattern = regExPattern.Replace("]", "_]");
                            if (!ValidateDataEntry(row[i].ToString(),
newPattern))
                            {
                                invalidCount++;
                                Console.WriteLine(String.Format("Invalid: {0,-30}
{1,-5} {2,-5}", row[i], data.Rows.IndexOf(row).ToString(), i.ToString()));
                            }
                        }
                    }
                }
                if (invalidCount == 0)
                {
                    Console.WriteLine("Data has passed validity checks. Ready for
preprocessing.");
                }

            }
        }

        // Reads data dictionary csv file and populates dataDict attribute
        // params: data dictionary file name
        private void ReadDataDictionaryFile(string dataDictFilename)
        {
            dataDict = new DataTable();
            fileReadFailed = false;
            try
            {
                using (StreamReader sr = new StreamReader(dataDictFilename))
                {
                    string[] columnNames = sr.ReadLine().Split(',');
                    foreach (string columnName in columnNames)
                    {
                        dataDict.Columns.Add(columnName);
                    }

                    while (!sr.EndOfStream)
                    {
                        string[] record = sr.ReadLine().Split(',');
                        DataRow row = dataDict.NewRow();
                        for (int i = 0; i < columnNames.Length; i++)
                        {
                            row[i] = record[i];
                        }
                        dataDict.Rows.Add(row);
                    }
                }
            }
            catch (Exception)
            {
                fileReadFailed = true;
                Console.WriteLine("Error when trying to read data dictionary
file.");
            }
        }

        // Reads data csv file and populates data attribute
        // params: data filename
        private void ReadDataFile(string dataFilename)
        {
            data = new DataTable();
            try
            {
                using (StreamReader sr = new StreamReader(dataFilename))
                {
                    // store fields
                    int colIndex = 0;
                    string[] fieldNames = sr.ReadLine().Split(',');
                    foreach (string fieldName in fieldNames)
                    {
                        string dataType = GetIntendedColumnDataType(dataDict,
colIndex);
                        if (dataType == "Integer")
                        {
                            data.Columns.Add(fieldName, typeof(int));
                        }
                        else if (dataType == "String")
                        {
                            data.Columns.Add(fieldName, typeof(string));
                        }
                        colIndex++;
                        //Console.WriteLine(String.Format("{0,-20} {1,-5}",
fieldName, data.Columns[fieldName].DataType.ToString()));
                    }
                    // store records
                    while (!sr.EndOfStream)
                    {
                        string[] record = sr.ReadLine().Split(',');
                        DataRow row = data.NewRow();
                        for (int i = 0; i < fieldNames.Length; i++)
                        {
                            // Levy Field contains '-' which should be read as 0
                            if (i == 2 && record[i] == "-")
                            {
                                row[i] = 0;
                            }
                            else
                            {
                                row[i] = record[i];
                            }
                        }
                        data.Rows.Add(row);
                    }
                }
            }
            catch (Exception)
            {
                fileReadFailed = true;
                Console.WriteLine("Error when trying to read dataset file.");
                Console.ReadLine();
                Environment.Exit(1);
            }
        }

        // Retrieves data type in row i of the data dictionary
        // params: data dictionary, row index of dictionary
        // returns: intended data type for the column
        private string GetIntendedColumnDataType(DataTable dataDict, int i)
        {
            string dataType = dataDict.Rows[i]["Data type"].ToString();
            return dataType;
        }

        // Validates data entry by checking it matches its corresponding regular 
expression pattern
        // params: input string, regular expression pattern
        // returns: boolean for whether the input string is valid or not
        private bool ValidateDataEntry(string value, string pattern)
        {
            bool isValid = Regex.IsMatch(value, pattern);
            if (isValid)
            {
                return true;
            }
            return false;
        }
    }
}
```

DataPreprocessor.cs

```
using System;
using System.Collections.Generic;
using System.Data;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the methods for the data preprocessing stage
    internal class DataPreprocessor
    {
        public DataTable data;
        private OutlierIdentfier outlierIdentfier;
        private DataTableModifier dataTableModifier;

        // Constrcutor to instantiate the outlieridentfiier and datatablemodifier 
objects, and to populate the data attribute
        public DataPreprocessor(DataTable argData)
        {
            data = argData;
            outlierIdentfier = new OutlierIdentfier(data);
            dataTableModifier = new DataTableModifier(data);
        }

        // Delegates task of removing duplicate records to dataTableModifier 
instance
        public void RemoveDuplicateRecords()
        {
            dataTableModifier.RemoveDuplicateRecords();
        }

        // Checks if there are missing values and, if found, writes their indices 
to the console
        public void CheckForMissingValues()
        {
            bool hasMissingValues = false;
            foreach (DataRow row in data.Rows)
            {
                for (int i = 0; i < row.ItemArray.Length; i++)
                {
                    if (string.IsNullOrEmpty(row.ItemArray[i].ToString()))
                    {
                        hasMissingValues = true;
                        Console.WriteLine("Missing value at row {0} column {1}",
data.Rows.IndexOf(row), i);
                    }
                }
            }
            if (!hasMissingValues)
            {
                Console.WriteLine("No null/missing values.");
            }
        }

        // Delegates task of making the leather interior column numerical to
dataTableModifier instance
        public void MakeLeatherInteriorColumnNumerical()
        {
            dataTableModifier.MakeLeatherInteriorColumnNumerical();
        }

        // Delegates task of splitting the engine volume column to
dataTableModifier instance
        public void SplitEngineVolumeColumn()
        {
            dataTableModifier.SplitEngineVolumeColumn();
        }

        // Delegates task of making the mileage column numerical to
dataTableModifier instance
        public void MakeMileageColumnNumerical()
        {
            dataTableModifier.MakeMileageColumnNumerical();
        }

        // Delegates task of making the wheel column numerical to
dataTableModifier instance
        public void MakeWheelColumnNumerical()
        {
            dataTableModifier.MakeWheelColumnNumerical();
        }

        // Delegates task of making the category column numerical to
dataTableModifier instance
        public void MakeCategoryColumnNumerical()
        {
            dataTableModifier.MakeCategoryColumnNumerical();
        }

        // Delegates task of making the fuel type column numerical to
dataTableModifier instance
        public void MakeFuelTypeColumnNumerical()
        {
            dataTableModifier.MakeFuelTypeColumnNumerical();
        }

        // Delegates task of making the gearbox column numerical to
dataTableModifier instance
        public void MakeGearBoxTypeColumnNumerical()
        {
            dataTableModifier.MakeGearBoxTypeColumnNumerical();
        }

        // Delegates task of making drive wheels column numerical to
dataTableModifier instance
```

```
        public void MakeDriveWheelsColumnNumerical()
        {
            dataTableModifier.MakeDriveWheelsColumnNumerical();
        }

        // Delegates task of making colour column numerical to dataTableModifier
instance
        public void MakeColourColumnNumerical()
        {
            dataTableModifier.MakeColourColumnNumerical();
        }

        // Delegates task of locating outliers using z-scores to
outlierIdentifier instance
        // params: columnName, z-Score threshold
        // returns: list of row numbers containing outliers
        public List<int> LocateOutliersWithZScores(string columnName, double
threshold)
        {
            return outlierIdentfier.LocateOutliersWithZScores(columnName,
threshold);
        }

        // Delegates task of locating outliers using modified z-scores to
outlierIdentifier instance
        // params: column name, mod. z-Score threshold
        // returns: list of row numbers containing outliers
        public List<int> LocateOutliersWithModifiedZScores(string columnName,
double threshold)
        {
            return outlierIdentfier.LocateOutliersWithModifiedZScores(columnName,
threshold);
        }

        // Delegates task of locating outliers using IQR to outlierIdentifier
instance
        // params: column name
        public void LocateOutliersWithIQR(string columnName)
        {
            outlierIdentfier.LocateOutliersWithIQR(columnName);
        }

        // Imputes the mean value for a given data entry in a numerical (non-
binary) column
        // params: column name, row number
        public void ImputeNumericalWithMean(string columnName, int rowNo)
        {
            double[] columnArray =
DataUtilities.GetColumnValuesAsDoubleArray(data, columnName);
            data.Rows[rowNo][columnName] = Statistics.CalculateMean(columnArray);
        }

        // Imputes the mode binary value (1 or 0) for a given data entry in a
binary column
        // params: column name, row number
        public void ImputeBinaryWithMode(string columnName, int rowNo)
        {
            double[] columnArray =
DataUtilities.GetColumnValuesAsDoubleArray(data, columnName);
            data.Rows[rowNo][columnName] =
int.Parse(Statistics.CalculateMode(columnArray).ToString());
        }
        // Scales a given column's values using min-max scaling
        // params: column name
        public void ScaleNumericalColumnMinMax(string columnName)
        {
            DataColumn scaledColumn = new DataColumn(columnName + " scaled",
typeof(double));
            data.Columns.Add(scaledColumn);
            double minValue = double.Parse(data.Rows[0][columnName].ToString());
            double maxValue = double.Parse(data.Rows[0][columnName].ToString());
            foreach(DataRow row in data.Rows)
            {
                double value = double.Parse(row[columnName].ToString());
                if (value < minValue)
                {
                    minValue = value;
                }
                if (value > maxValue)
                {
                    maxValue = value;
                }
            }
            foreach(DataRow row in data.Rows)
            {
                double currentValue = double.Parse(row[columnName].ToString());
                double scaledValue = (currentValue - minValue) / (maxValue -
minValue);
                row[columnName + " scaled"] = scaledValue;
            }
        }

        // Scales a given column's values using standardisation
        // params: column name
        public void ScaleNumericalColumnStandardisation(string columnName)
        {
            DataColumn scaledColumn = new DataColumn(columnName + " scaled",
typeof(double));
            data.Columns.Add(scaledColumn);
            double[] columnArray =
DataUtilities.GetColumnValuesAsDoubleArray(data, columnName);
            double mean = Statistics.CalculateMean(columnArray);
            double stdDev = Statistics.CalculateStdDev(columnArray);
            foreach (DataRow row in data.Rows)
            {
                double currentValue = double.Parse(row[columnName].ToString());
                double scaledValue = (currentValue - mean) / stdDev;
                row[columnName + " scaled"] = scaledValue;
            }
        }
    }
}
```

DataTableModifier.cs

```
using System;
using System.Data;
using System.Linq;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the methods used to alter the structure of the
data
```

```
    internal class DataTableModifier
    {
        public DataTable data;

        public DataTableModifier(DataTable argData)
        {
            data = argData;
        }

        // Checks if there are duplicate rows and, if found, removes them
        public void RemoveDuplicateRecords()
        {
            int initialNoOfRows = data.Rows.Count;

            // removes ID column
            if (data.Columns.Contains("ID"))
            {
                data.Columns.Remove("ID");
            }

            var duplicates = data.AsEnumerable()
                .GroupBy(row => string.Join(",", row.ItemArray.Select(item =>
item.ToString())))
                .Where(group => group.Count() > 1); // assigns groups of
duplicate rows to duplicates
            if (duplicates.Any())
            {
                foreach(var group in duplicates)
                {
                    foreach (var row in group.Skip(1)) // skip first row because
need to keep one copy of duplicate
                    {
                        data.Rows.Remove(row);
                    }
                }
                int noOfDeletedRows = initialNoOfRows - data.Rows.Count;
                Console.WriteLine($"{noOfDeletedRows} Duplicate records
removed.");
            }
            else
            {
                Console.WriteLine("No duplicate records found.");
            }
        }

        // Converts Leather interior column from string to integer, with boolean
values
        // Stores 1 if the car has a leather interior, and 0 if not
        public void MakeLeatherInteriorColumnNumerical()
        {
            DataColumn newColumn = new DataColumn("Is leather interior",
typeof(int));
            data.Columns.Add(newColumn);
            foreach (DataRow row in data.Rows)
            {
                int intValue = 0;
                string value = row["Leather interior"].ToString();
                if (value == "Yes")
                {
                    intValue = 1;
                }
                row["Is leather interior"] = intValue;
            }
            data.Columns.Remove("Leather interior");
            Console.WriteLine("Leathor interior column has been modified.");
        }

        // Splits Engine Volume column into a column containing the numerical
volume
        // And a boolean column (of integer data type) which stores a 1 if the
car has a turbo engine, and 0 if not
        public void SplitEngineVolumeColumn()
        {
            DataColumn newColumn1 = new DataColumn("Engine displacement",
typeof(float));
            DataColumn newColumn2 = new DataColumn("Is engine turbo",
typeof(int));
            data.Columns.Add(newColumn1);
            data.Columns.Add(newColumn2);
            foreach (DataRow row in data.Rows)
            {
                int isTurbo = 0;
                string value = row["Engine volume"].ToString();
                string[] substrings = value.Split(' ');
                float engineDisplacement = float.Parse(substrings[0]);
                if (substrings.Length == 2)
                {
                    if (substrings[1] == "Turbo")
                    {
                        isTurbo = 1;
                    }
                }
                row["Engine displacement"] = engineDisplacement;
                row["Is engine turbo"] = isTurbo;
            }
            data.Columns.Remove("Engine volume");
            Console.WriteLine("Engine volume column has been split.");
        }

        // Converts Mileage column from string to integer by removing taling 'km'
        public void MakeMileageColumnNumerical()
        {
            DataColumn newColumn = new DataColumn("Mileage in km", typeof(int));
            data.Columns.Add(newColumn);
            foreach (DataRow row in data.Rows)
            {
                row["Mileage in km"] =
int.Parse(row["Mileage"].ToString().TrimEnd(' ', 'k', 'm'));
            }
            data.Columns.Remove("Mileage");
            Console.WriteLine("Mileage column has been modified.");
        }

        // Converts Wheel column from string to integer, with boolean values
        // Stores 1 if the car's steering wheel is on the left, and 0 if it's on
the right
        public void MakeWheelColumnNumerical()
        {
            DataColumn newColumn = new DataColumn("Is wheel left", typeof(int));
            data.Columns.Add(newColumn);
            foreach (DataRow row in data.Rows)
            {
                int intValue = 0;
                string value = row["Wheel"].ToString();
                if (value == "Left wheel")
                {
                    intValue = 1;
                }
                row["Is wheel left"] = intValue;
            }
            data.Columns.Remove("Wheel");
            Console.WriteLine("Wheel column has been modified.");
        }

        // Splits Category column into multiple boolean columns (of integer data
type)
        // Each category value has its own column with a 1 stored where the car's
category is this value
        public void MakeCategoryColumnNumerical()
        {
            DataColumn newColumn1 = new DataColumn("Is cabriolet", typeof(int));
            DataColumn newColumn2 = new DataColumn("Is coupe", typeof(int));
            DataColumn newColumn3 = new DataColumn("Is goods wagon",
typeof(int));
            DataColumn newColumn4 = new DataColumn("Is hatchback", typeof(int));
            DataColumn newColumn5 = new DataColumn("Is jeep", typeof(int));
            DataColumn newColumn6 = new DataColumn("Is limousine", typeof(int));
            DataColumn newColumn7 = new DataColumn("Is microbus", typeof(int));
            DataColumn newColumn8 = new DataColumn("Is minivan", typeof(int));
            DataColumn newColumn9 = new DataColumn("Is pickup", typeof(int));
            DataColumn newColumn10 = new DataColumn("Is sedan", typeof(int));
            DataColumn newColumn11 = new DataColumn("Is universal", typeof(int));
            data.Columns.Add(newColumn1);
            data.Columns.Add(newColumn2);
            data.Columns.Add(newColumn3);
            data.Columns.Add(newColumn4);
            data.Columns.Add(newColumn5);
            data.Columns.Add(newColumn6);
            data.Columns.Add(newColumn7);
            data.Columns.Add(newColumn8);
            data.Columns.Add(newColumn9);
            data.Columns.Add(newColumn10);
            data.Columns.Add(newColumn11);
            foreach (DataRow row in data.Rows)
            {
                row["Is cabriolet"] = row["Is coupe"] = row["Is goods wagon"] =
row["Is hatchback"] = row["Is jeep"] = row["Is limousine"] = row["Is microbus"] =
row["Is minivan"] = row["Is pickup"] = row["Is sedan"] = row["Is universal"] = 0;
                string categoryValue = row["Category"].ToString();
                switch (categoryValue)
                {
                    case "Cabriolet":
                        row["Is cabriolet"] = 1;
                        break;
                    case "Coupe":
                        row["Is coupe"] = 1;
                        break;
                    case "Goods wagon":
                        row["Is goods wagon"] = 1;
                        break;
                    case "Hatchback":
                        row["Is hatchback"] = 1;
                        break;
                    case "Jeep":
                        row["Is jeep"] = 1;
                        break;
                    case "Limousine":
                        row["Is limousine"] = 1;
                        break;
                    case "Microbus":
                        row["Is microbus"] = 1;
                        break;
                    case "Minivan":
                        row["Is minivan"] = 1;
                        break;
                    case "Pickup":
                        row["Is pickup"] = 1;
                        break;
                    case "Sedan":
                        row["Is sedan"] = 1;
                        break;
                    case "Universal":
                        row["Is universal"] = 1;
                        break;
                    default:
                        Console.WriteLine("There is no matching car category.");
                        Console.ReadLine();
                        break;
                }
            }
            data.Columns.Remove("Category");
            Console.WriteLine("Category column has been modified.");
        }

        // Splits Fuel type column into multiple boolean columns (of integer data
type)
        // Each fuel type value has its own corresponding boolean column
        public void MakeFuelTypeColumnNumerical()
        {
            DataColumn newColumn1 = new DataColumn("Is CNG", typeof(int));
            DataColumn newColumn2 = new DataColumn("Is diesel", typeof(int));
            DataColumn newColumn3 = new DataColumn("Is hybrid", typeof(int));
            DataColumn newColumn4 = new DataColumn("Is hydrogen", typeof(int));
            DataColumn newColumn5 = new DataColumn("Is LPG", typeof(int));
            DataColumn newColumn6 = new DataColumn("Is petrol", typeof(int));
            DataColumn newColumn7 = new DataColumn("Is plug-in hybrid",
typeof(int));
            data.Columns.Add(newColumn1);
            data.Columns.Add(newColumn2);
            data.Columns.Add(newColumn3);
            data.Columns.Add(newColumn4);
            data.Columns.Add(newColumn5);
            data.Columns.Add(newColumn6);
            data.Columns.Add(newColumn7);
            foreach (DataRow row in data.Rows)
            {
                row["Is CNG"] = row["Is diesel"] = row["Is hybrid"] = row["Is
hydrogen"] = row["Is LPG"] = row["Is petrol"] = row["Is plug-in hybrid"] = 0;
                string fuelTypeValue = row["Fuel Type"].ToString();
                switch (fuelTypeValue)
                {
                    case "CNG":
                        row["Is CNG"] = 1;
                        break;
                    case "Diesel":
                        row["Is diesel"] = 1;
                        break;
                    case "Hybrid":
                        row["Is hybrid"] = 1;
                        break;
                    case "Hydrogen":
                        row["Is hydrogen"] = 1;
                        break;
                    case "LPG":
                        row["Is LPG"] = 1;
                        break;
                    case "Petrol":
                        row["Is petrol"] = 1;
                        break;
                    case "Plug-in Hybrid":
                        row["Is plug-in hybrid"] = 1;
                        break;
                }
            }
            data.Columns.Remove("Fuel type");
            Console.WriteLine("Fuel type column has been modified.");
        }

        // Splits Gear box type column into multiple boolean columns (of integer
data type)
        // Each gear box type value has its own corresponding boolean column
        public void MakeGearBoxTypeColumnNumerical()
        {
            DataColumn newColumn1 = new DataColumn("Is automatic", typeof(int));
            DataColumn newColumn2 = new DataColumn("Is manual", typeof(int));
            DataColumn newColumn3 = new DataColumn("Is tiptronic", typeof(int));
            DataColumn newColumn4 = new DataColumn("Is variator", typeof(int));
            data.Columns.Add(newColumn1);
            data.Columns.Add(newColumn2);
            data.Columns.Add(newColumn3);
            data.Columns.Add(newColumn4);
            foreach (DataRow row in data.Rows)
            {
                row["Is automatic"] = row["Is manual"] = row["Is tiptronic"] =
row["Is variator"] = 0;
                string gearBoxValue = row["Gear box type"].ToString();
                switch (gearBoxValue)
                {
                    case "Automatic":
                        row["Is automatic"] = 1;
                        break;
                    case "Manual":
                        row["Is manual"] = 1;
                        break;
                    case "Tiptronic":
                        row["Is tiptronic"] = 1;
                        break;
                    case "Variator":
                        row["Is variator"] = 1;
                        break;
                    default:
                        Console.WriteLine("There is no matching gear box type.");
                        Console.ReadLine();
                        break;

                }
            }
            data.Columns.Remove("Gear box type");
            Console.WriteLine("Gear box type column has been modified.");
        }

        // Splits Drive wheels column into multiple boolean columns (of integer
data type)
        // Each drive wheels value has its own corresponding boolean column
        public void MakeDriveWheelsColumnNumerical()
        {
            DataColumn newColumn1 = new DataColumn("Is 4x4", typeof(int));
            DataColumn newColumn2 = new DataColumn("Is front", typeof(int));
            DataColumn newColumn3 = new DataColumn("Is rear", typeof(int));
            data.Columns.Add(newColumn1);
            data.Columns.Add(newColumn2);
            data.Columns.Add(newColumn3);
            foreach (DataRow row in data.Rows)
            {
                row["Is 4x4"] = row["Is front"] = row["Is rear"] = 0;
                string driveWheelsValue = row["Drive wheels"].ToString();
                switch (driveWheelsValue)
                {
                    case "4x4":
                        row["Is 4x4"] = 1;
                        break;
                    case "Front":
                        row["Is front"] = 1;
                        break;
                    case "Rear":
                        row["Is rear"] = 1;
                        break;
                    default:
                        Console.WriteLine("There is no matching drive wheels
type.");
                        Console.ReadLine();
                        break;
                }
            }
            data.Columns.Remove("Drive wheels");
            Console.WriteLine("Drive wheels column has been modified.");
        }

        // Splits Colour column into multiple boolean columns (of integer data
type)
        // Each colour value has its own corresponding boolean column
        public void MakeColourColumnNumerical()
        {
            DataColumn newColumn1 = new DataColumn("Is beige", typeof(int));
            DataColumn newColumn2 = new DataColumn("Is black", typeof(int));
            DataColumn newColumn3 = new DataColumn("Is blue", typeof(int));
            DataColumn newColumn4 = new DataColumn("Is brown", typeof(int));
            DataColumn newColumn5 = new DataColumn("Is carnelian red",
typeof(int));
            DataColumn newColumn6 = new DataColumn("Is golden", typeof(int));
            DataColumn newColumn7 = new DataColumn("Is green", typeof(int));
            DataColumn newColumn8 = new DataColumn("Is grey", typeof(int));
            DataColumn newColumn9 = new DataColumn("Is orange", typeof(int));
            DataColumn newColumn10 = new DataColumn("Is pink", typeof(int));
            DataColumn newColumn11 = new DataColumn("Is purple", typeof(int));
            DataColumn newColumn12 = new DataColumn("Is red", typeof(int));
            DataColumn newColumn13 = new DataColumn("Is silver", typeof(int));
            DataColumn newColumn14 = new DataColumn("Is sky blue", typeof(int));
            DataColumn newColumn15 = new DataColumn("Is white", typeof(int));
            DataColumn newColumn16 = new DataColumn("Is yellow", typeof(int));
            data.Columns.Add(newColumn1);
            data.Columns.Add(newColumn2);
            data.Columns.Add(newColumn3);
            data.Columns.Add(newColumn4);
            data.Columns.Add(newColumn5);
            data.Columns.Add(newColumn6);
            data.Columns.Add(newColumn7);
            data.Columns.Add(newColumn8);
            data.Columns.Add(newColumn9);
            data.Columns.Add(newColumn10);
            data.Columns.Add(newColumn11);
            data.Columns.Add(newColumn12);
            data.Columns.Add(newColumn13);
            data.Columns.Add(newColumn14);
            data.Columns.Add(newColumn15);
            data.Columns.Add(newColumn16);
            foreach (DataRow row in data.Rows)
            {
                row["Is beige"] = row["Is blue"] = row["Is black"] = row["Is
brown"] = row["Is carnelian red"] = row["Is golden"] =
                    row["Is green"] = row["Is grey"] = row["Is orange"] = row["Is
pink"] = row["Is purple"] = row["Is red"] =
                    row["Is silver"] = row["Is sky blue"] = row["Is white"] =
row["Is yellow"] = 0;
                string colourValue = row["Color"].ToString();
                switch (colourValue)
                {
                    case "Beige":
                        row["Is beige"] = 1;
                        break;
                    case "Blue":
                        row["Is blue"] = 1;
                        break;
                    case "Black":
                        row["Is black"] = 1;
                        break;
                    case "Brown":
                        row["Is brown"] = 1;
                        break;
                    case "Carnelian red":
                        row["Is carnelian red"] = 1;
                        break;
                    case "Golden":
                        row["Is golden"] = 1;
                        break;
                    case "Green":
                        row["Is green"] = 1;
                        break;
                    case "Grey":
                        row["Is grey"] = 1;
                        break;
                    case "Orange":
                        row["Is orange"] = 1;
                        break;
                    case "Pink":
                        row["Is pink"] = 1;
                        break;
                    case "Purple":
                        row["Is purple"] = 1;
                        break;
                    case "Red":
                        row["Is red"] = 1;
                        break;
                    case "Silver":
                        row["Is silver"] = 1;
                        break;
                    case "Sky blue":
                        row["Is sky blue"] = 1;
                        break;
                    case "White":
                        row["Is white"] = 1;
                        break;
                    case "Yellow":
                        row["Is yellow"] = 1;
                        break;
                    default:
                        Console.WriteLine("There is no matching colour.");
                        Console.ReadLine();
                        break;
                }
            }
            data.Columns.Remove("Color");
            Console.WriteLine("Color column has been modified.");
        }
    }
}
```

OutlierIdentifier.cs

```
using System;
using System.Collections.Generic;
using System.Data;

namespace RegressionAnalysisProj
{
    // Class encapsulating the methods for locating outliers
    internal class OutlierIdentfier
    {
        public DataTable data;
        public OutlierIdentfier(DataTable argData)
        {
            data = argData;
        }

        // Locates outliers using z-scores in a given column
        // params: column name, z-score threshold
        // returns: list of row numbers containing outliers
        public List<int> LocateOutliersWithZScores(string columnName, double
threshold)
        {
            List<int> outlierRows = new List<int>();
            double[] columnValues =
DataUtilities.GetColumnValuesAsDoubleArray(data, columnName);
            double mean = Statistics.CalculateMean(columnValues);
            double stdDev = Statistics.CalculateStdDev(columnValues);
            Console.WriteLine("Checking for outliers using Z-scores for column
{0}...", columnName);
            for (int i = 0; i < columnValues.Length; i++)
            {
                double value = columnValues[i];
                double zScore = Statistics.CalculateZScore(value, mean, stdDev);
                if (Math.Abs(zScore) > threshold)
                {
                    outlierRows.Add(i);
                    //Console.WriteLine($"Outlier. Value: {value,-10} Z-Score:
{zScore,-10} Row: {i,-5}");
                }
            }
            return outlierRows;
        }

        // Locates outliers using modified z-scores in a given column
        // params: column name, z-score threshold
        // returns: list of row numbers containing outliers
        public List<int> LocateOutliersWithModifiedZScores(string columnName, 
double threshold)
        {
            List<int> outlierRows = new List<int>();
            double[] columnValues = 
DataUtilities.GetColumnValuesAsDoubleArray(data, columnName);
            double median = Statistics.CalculateMedian(columnValues);
            double mad = Statistics.CalculateMedianAbsDev(columnValues);
            Console.WriteLine("Checking for outliers using modified Z-scores for 
column {0}...", columnName);
            for (int i = 0; i < columnValues.Length; i++)
            {
                double value = columnValues[i];
                double modZScore = Statistics.CalculateModifiedZScore(value, 
median, mad);
                if (Math.Abs(modZScore) > threshold)
                {
                    outlierRows.Add(i);
                    Console.WriteLine($"Outlier. Value: {value,-10} Modified Z-
Score: {modZScore,-10} Row: {i,-5}");
                }
            }
            return outlierRows;
        }

        // Locates outliers using the IQR method in a given column
        // params: column name
        public void LocateOutliersWithIQR(string columnName)
        {
            double[] values = DataUtilities.GetColumnValuesAsDoubleArray(data, 
columnName);
            double[] sortedValues = new double[values.Length];
            Array.Copy(values, sortedValues, values.Length);
            Statistics.MergeSort(sortedValues);
            double q1 = Statistics.CalculatePercentile(sortedValues, 0.25);
            double q3 = Statistics.CalculatePercentile(sortedValues, 0.75);
            double iqr = q3 - q1;
            double lowerThreshold = q1 - 1.5 * iqr;
            double upperThreshold = q3 + 1.5 * iqr;
            Console.WriteLine("Checking for outliers using the IQR method for 
column {0}...", columnName);
            for (int i=0; i<data.Rows.Count; i++)
            {
                if (values[i] < lowerThreshold || values[i] > upperThreshold)
                {
                    Console.WriteLine($"Outlier. Value: {values[i],-10} Row: {i,-
5}");
                }
            }
            Console.ReadLine();
        }
    }
}
```

FeatureSelector.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the methods for aiding feature selection
    internal class FeatureSelector
    {
        private DataTable data;

        public FeatureSelector(DataTable argData)
        {
            data = argData;
        }

        // Instantiates a scatter plot form to visualise the correlation of a
continous predictor and the car price
        // params: column name
        public void DisplayScatterPlot(string columnName)
        {
            ScatterPlotForm form = new ScatterPlotForm(data, columnName,
"Price");
        }

        // Instantiates a box plot form to visualise the distribution/skewness of
non-binary predictors
        // params: array of column names
        public void DisplayBoxPlots(string[] columnNames)
        {
            BoxPlotForm form = new BoxPlotForm(data, columnNames);
        }

        // Outputs a ranking of how correlated the non-binary predictors are
based on their pearson coefficient
        // params: array of column names
        public void RankNumericalColumnsByPearsonCoefficients(string[]
columnNames)
        {
            Dictionary<string, double> columnCoeffPairs = new Dictionary<string,
double>();
            double[] yValues = DataUtilities.GetColumnValuesAsDoubleArray(data,
"Price");
            for (int i = 0; i < columnNames.Length; i++)
            {
                double[] xValues =
DataUtilities.GetColumnValuesAsDoubleArray(data, columnNames[i]);
                double coeff =
Statistics.CalculatePearsonCorrelationCoefficient(xValues, yValues);
                columnCoeffPairs.Add(columnNames[i], coeff);
            }
            var sortedList = columnCoeffPairs.OrderByDescending(kvp =>
Math.Abs(kvp.Value)).ToList(); // orders key-value pairs by value in descending
order
            Console.WriteLine("Predictors ranked by their pearson correlation
coefficient:");
            for (int i = 0; i < sortedList.Count; i++)
            {
                Console.WriteLine($"{i+1}. {sortedList[i].Key,-20} Coeff:
{sortedList[i].Value}");
            }
        }

        // Outputs a ranking of how correlated the binary predictors are based on
their point biserial correlation coefficient
```

```
        // params: array of column names
        public void RankBinaryColumnsByPointBiserialCoefficients(string[]
columnNames)
        {
            Dictionary<string, double> columnCoeffPairs = new Dictionary<string,
double>();
            double[] yValues = DataUtilities.GetColumnValuesAsDoubleArray(data,
"Price");
            for (int i = 0; i < columnNames.Length; i++)
            {
                double[] xValues =
DataUtilities.GetColumnValuesAsDoubleArray(data, columnNames[i]);
                double coeff =
Statistics.CalculatePointBiserialCorrelationCoefficient(xValues, yValues);
                columnCoeffPairs.Add(columnNames[i], coeff);
            }
            var sortedList = columnCoeffPairs.OrderByDescending(kvp =>
Math.Abs(kvp.Value)).ToList(); // orders key-value pairs by value in descending
order
            Console.WriteLine("Predictors ranked by their point biserial
correlation coefficient:");
            for (int i = 0; i < sortedList.Count; i++)
            {
                Console.WriteLine($"{i+1}. {sortedList[i].Key,-20} Coeff:
{sortedList[i].Value}");
            }
        }

        // Displays the percentage of 1s for each binary column passed
        // params: array of column names
        public void DisplayBinaryColumnRatios(string[] columnNames)
        {
            string s = "Binary Column Name:";
            Console.WriteLine($"{s,-20} Percentage of 1s:");
            for (int i = 0; i < columnNames.Length; i++)
            {
                double[] values =
DataUtilities.GetColumnValuesAsDoubleArray(data, columnNames[i]);
                int n = values.Length;
                int noOfOnes = 0;
                foreach(var value in values)
                {
                    if (value == 1)
                    {
                        noOfOnes++;
                    }
                }
                double percentage = noOfOnes / (double)n * 100;
                percentage = Math.Round(percentage, 1);
                Console.WriteLine($"{columnNames[i],-20} {percentage}%");
            }
        }
    }
}
```

ScatterPlotForm.cs

```
using System;
using System.Data;
using System.Windows.Forms;
using OxyPlot;
using OxyPlot.Axes;
using OxyPlot.Series;
using OxyPlot.WindowsForms;

namespace RegressionAnalysisProj
{
    // Class that contains the logic for creating a form with a scatter plot
    internal class ScatterPlotForm
    {
        private DataTable data;
        private string xColumnName;
        private string yColumnName;
        public ScatterPlotForm(DataTable data, string xColumnName, string
yColumnName)
        {
            Application.EnableVisualStyles();

            this.data = data;
            this.xColumnName = xColumnName;
            this.yColumnName = yColumnName;

            PlotModel plotModel = CreateScatterPlot();

            // Displays the plot in a Windows Form
            var plotForm = new Form();
            var plotView = new PlotView();
            plotView.Dock = DockStyle.Fill;
            plotView.Model = plotModel;
            plotForm.Controls.Add(plotView);

            // Shows the form
            Application.Run(plotForm);
        }

        // Creates the scatter plot
        // returns: plot model
        private PlotModel CreateScatterPlot()
        {
            var plotModel = new PlotModel { Title = "Scatter Plot" };
            var scatterSeries = new ScatterSeries()
            {
                MarkerType = MarkerType.Circle,
                MarkerSize = 3
            };

            foreach (DataRow row in data.Rows)
            {
                double x = Convert.ToDouble(row[xColumnName]);
                double y = Convert.ToDouble(row[yColumnName]);
                scatterSeries.Points.Add(new ScatterPoint(x, y));
            }

            var xAxis = new LinearAxis { Position = AxisPosition.Bottom, Title =
xColumnName };
            var yAxis = new LinearAxis { Position = AxisPosition.Left, Title =
yColumnName };
            plotModel.Axes.Add(xAxis);
            plotModel.Axes.Add(yAxis);

            plotModel.Series.Add(scatterSeries);
            return plotModel;
        }
    }
}
```

BoxPlotForm.cs

```
using OxyPlot.Axes;
using OxyPlot.Series;
using OxyPlot.WindowsForms;
using OxyPlot;
using System;
using System.Data;
using System.Windows.Forms;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the implementation to display a form with boxplots
    internal class BoxPlotForm
    {
        private DataTable data;
        private string[] variables;
        public BoxPlotForm(DataTable argData, string[] variables)
        {
            Application.EnableVisualStyles();

            data = argData;
            this.variables = variables;

            PlotModel plotModel = CreateBoxPlot();

            // Displays the box plot in a Windows Form
            var plotForm = new Form();
            var plotView = new PlotView();
            plotView.Dock = DockStyle.Fill;
            plotView.Model = plotModel;
            plotForm.Controls.Add(plotView);


            // Shows the form
            Application.Run(plotForm);
        }

        // Creates a plotmodel object with a boxplot series, consisting of 1 or
more individual box plots
        private PlotModel CreateBoxPlot()
        {
            var plotModel = new PlotModel { Title = "Box Plot" };

            var boxPlotSeries = new BoxPlotSeries()
            {
                Fill = OxyColors.LightGray
            };
            var xAxis = new CategoryAxis
            {
                Position = AxisPosition.Bottom,
            };

            int i;
            for (i = 0; i < variables.Length; i++)
            {
                double[] array = DataUtilities.GetColumnValuesAsDoubleArray(data,
variables[i]);
                double[] sortedArray = new double[array.Length];
                Array.Copy(array, sortedArray, array.Length);
                Statistics.MergeSort(sortedArray);
                double min = Statistics.CalculatePercentile(sortedArray, 0);
                double q1 = Statistics.CalculatePercentile(sortedArray, 0.25);
                double q2 = Statistics.CalculatePercentile(sortedArray, 0.5);
                double q3 = Statistics.CalculatePercentile(sortedArray, 0.75);
                double max = Statistics.CalculatePercentile(sortedArray, 1);
                BoxPlotItem bp = new BoxPlotItem(i, min, q1, q2, q3, max);
                boxPlotSeries.Items.Add(bp);
                xAxis.Labels.Add(variables[i]);
            }
            BoxPlotItem emptyBp = new BoxPlotItem(i, 0, 0, 0, 0, 0);
            boxPlotSeries.Items.Add(emptyBp);

            plotModel.Series.Add(boxPlotSeries);

            plotModel.Axes.Add(xAxis);
            return plotModel;
        }
    }
}
```

RegressionModel.cs

```
using System;
using System.Data;

namespace RegressionAnalysisProj
{
    // Abstract class that encapsulates the core components of developing a
regression model
    internal abstract class RegressionModel
    {
        protected DataTable data; // attribute is passed by reference throughout
        protected DataTable trainingData; // to be used for feature selection,
train and fitting
        protected DataTable validationData; // to be used for model validation
        protected DataTable holdoutData; // to be used for final model evaluation
        protected DataTable trainingFolds;
        protected DataTable validationFold;
        protected double[] yPredictions;
        protected DataTable originalData;
        protected int noOfPredictors;
        public RegressionModel(DataTable data)
        {
            this.data = data;
            originalData = data.Copy();
        }
        // For preprocessing data, ready for feature selection and training
        abstract public void PreProcessData();
        public void SplitData()
        {
            DataTable[] dataArray = DataUtilities.SplitData(data, 0.72, 0.18); //
ratios chosen according to k=5
            trainingData = dataArray[0];
            validationData = dataArray[1];
            holdoutData = dataArray[2];
        }

        // For performing tasks like visualising data and ranking features to
enable feature selection
```

```
        abstract public void SelectFeatures();

        // For training the model using the given regression model's algorithm
        abstract protected void Train();

        // For predicting the validation car prices using the trained regression
model
        abstract protected void Fit();

        // Cross-validates using k-fold validation and averages the four
regression error metrics
        // returns: array of error values
        protected double[] CrossValidate()
        {
            // error metrics
            double maeSum = 0;
            double rmseSum = 0;
            double r2Sum = 0;
            double adjR2Sum = 0;
            // set k here
            int k = 5;
            DataTable[] dataArray = DataUtilities.DivideDataInto4(trainingData);
            DataTable[] dataSubSets = new DataTable[5];
            dataSubSets[0] = validationData;
            dataArray.CopyTo(dataSubSets, 1);
            trainingFolds = dataSubSets[0].Clone();
            validationFold = dataSubSets[0].Clone();
            for (int i = 0; i < k; i++)
            {
                trainingFolds.Clear();
                validationFold.Clear();
                for (int j = 0; j < k; j++)
                {
                    if (j == i)
                    {
                        validationFold.Merge(dataSubSets[j]);
                    }
                    else
                    {
                        trainingFolds.Merge(dataSubSets[j]);
                    }
                }
                Train();
                Fit();
                double[] yActual =
DataUtilities.GetColumnValuesAsDoubleArray(validationFold, "Price");
                ModelValidator mv = new ModelValidator(yPredictions, yActual);
                maeSum += mv.CalculateMAE();
                rmseSum += mv.CalculateRMSE();
                r2Sum += mv.CalculateRSquared();
                adjR2Sum += mv.CalculateAdjustedRSquared(noOfPredictors);
            }
            double mae = Math.Round(maeSum / k, 6);
            double rmse = Math.Round(rmseSum / k, 6);
            double r2 = Math.Round(r2Sum / k, 6);
            double adjR2 = Math.Round(adjR2Sum / k, 6);
            double[] errorArray = { mae, rmse, r2, adjR2 };
            return errorArray;
        }

        // For training and validating the regression model with different
features to determine optimal feature combination
        abstract public void EvaluateModelsToDecideOptimalFeatureSelection();
        // For fine-tuning any hyperparameters
        abstract public string FineTuneModel();

        // For setting the values of the final hyperparameter values
        abstract protected void SetFinalHyperparameters();

        // Tests the model's performance with holdout data
        // returns: string containing the model's errors
        public string TestModelWithHoldoutData()
        {
            trainingFolds = trainingData.Clone();
            trainingFolds.Merge(trainingData);
            trainingFolds.Merge(validationData);
            validationFold = holdoutData.Copy();
            SetFinalHyperparameters();
            Train();
            Fit();
            double[] yActual =
DataUtilities.GetColumnValuesAsDoubleArray(validationFold, "Price");
            ModelValidator mv = new ModelValidator(yPredictions, yActual);
            return $"MAE: {Math.Round(mv.CalculateMAE(),6),-15} RMSE:
{Math.Round(mv.CalculateRMSE(),6),-15} R-Squared:
{Math.Round(mv.CalculateRSquared(),6),-15} Adj R-Squared:
{Math.Round(mv.CalculateAdjustedRSquared(noOfPredictors),6),-15}";
        }
    }
}
```

MeanTemplateModel.cs

```
using System;
using System.Data;

namespace RegressionAnalysisProj
{
    // Class encapsulating the implementation of a mean template model, purely
for comparisons with other models
    internal class MeanTemplateModel : RegressionModel
    {
        public MeanTemplateModel(DataTable data) : base(data) // calls
RegressionModel constructor
        {
            noOfPredictors = 0;
        }
        public override void PreProcessData()
        {
            // empty
        }

        public override void SelectFeatures()
        {
            // empty
        }

        protected override void Train()
        {
            // empty
        }

        protected override void Fit()
        {
            double[] yActualValues =
DataUtilities.GetColumnValuesAsDoubleArray(trainingFolds, "Price");
            double yMean = Statistics.CalculateMean(yActualValues);
            yPredictions = new double[validationFold.Rows.Count];
            for (int i = 0; i < yPredictions.Length; i++)
            {
                yPredictions[i] = yMean;
            }
        }

        public override void EvaluateModelsToDecideOptimalFeatureSelection()
        {
            // empty
        }
        public override string FineTuneModel()
        {
            double[] errorArray = CrossValidate();
            Console.WriteLine("Mean template model conclusion written to file");
            return $"Mean template model: \n" +
                   $"Errors: MAE = {errorArray[0]}, RMSE = {errorArray[1]}, R-
Squared = {errorArray[2]} \n" +
                   $"Hyperparameters: None \n" +
                   $"Features: None";
        }

        protected override void SetFinalHyperparameters()
        {
            // empty
        }
    }
}
```

SimpleLinearRegressionModel.cs

```
using System;
using System.Collections.Generic;
using System.Data;

namespace RegressionAnalysisProj
{
    // Class encapsulating the implementation of the simple linear regression
model
    internal class SimpleLinearRegressionModel : RegressionModel
    {
        private string xFeature;
        private double m;
        private double b;

        public SimpleLinearRegressionModel(DataTable data) : base(data) // calls
RegressionModel constructor
        {
            noOfPredictors = 1;
        }

        public override void PreProcessData()
        {
            DataPreprocessor dp = new DataPreprocessor(data);
            // initial data checks
            dp.CheckForMissingValues();
            dp.RemoveDuplicateRecords();
            // convert data to numerical form
            dp.SplitEngineVolumeColumn();
            dp.MakeMileageColumnNumerical();
            // locate outliers
            DataUtilities.DisplayColumnNames(data);

            List<int> priceOutliers = dp.LocateOutliersWithZScores("Price", 3);
            data.Rows[priceOutliers[1]].Delete();

            List<int> levyOutliers = dp.LocateOutliersWithZScores("Levy", 5);

            List<int> prodYearOutliers = dp.LocateOutliersWithZScores("Prod.
year", 5);

            List<int> cylindersOutliers =
dp.LocateOutliersWithZScores("Cylinders", 5);

            List<int> airbagsOutliers = dp.LocateOutliersWithZScores("Airbags",
3);

            dp.LocateOutliersWithIQR("Airbags");

            List<int> engineDisplacementOutliers =
dp.LocateOutliersWithZScores("Engine displacement", 6);
            DataUtilities.DisplayRows(data,
engineDisplacementOutliers.ToArray());
            Console.ReadLine();
            dp.ImputeNumericalWithMean("Engine displacement",
engineDisplacementOutliers[0]);

            List<int> mileageInKmOutliers =
dp.LocateOutliersWithModifiedZScores("Mileage in km", 3);
            int j = 0;
            foreach (int i in mileageInKmOutliers)
            {
                data.Rows[i-j].Delete();
                j++;
            }

            // scale data
            dp.ScaleNumericalColumnStandardisation("Levy");
            dp.ScaleNumericalColumnStandardisation("Prod. year");
            dp.ScaleNumericalColumnStandardisation("Cylinders");
            dp.ScaleNumericalColumnStandardisation("Airbags");
            dp.ScaleNumericalColumnStandardisation("Engine displacement");
            dp.ScaleNumericalColumnStandardisation("Mileage in km");
        }

        public override void SelectFeatures()
        {
            // perform feature selection on training data
            FeatureSelector fs = new FeatureSelector(trainingData);

            // display plots
            string[] columnNames = {"Levy scaled", "Prod. year scaled",
"Cylinders scaled", "Airbags scaled", "Engine displacement scaled", "Mileage in
km scaled" };
            fs.DisplayBoxPlots(columnNames);
            fs.DisplayScatterPlot("Levy");
            fs.DisplayScatterPlot("Prod. year");
            fs.DisplayScatterPlot("Cylinders");
            fs.DisplayScatterPlot("Airbags");
            fs.DisplayScatterPlot("Engine displacement");
            fs.DisplayScatterPlot("Mileage in km");

            // correlation coefficients
            fs.RankNumericalColumnsByPearsonCoefficients(columnNames);
        }

        protected override void Train()
        {
            double[] xTrain =
DataUtilities.GetColumnValuesAsDoubleArray(trainingFolds, xFeature);
            double[] yTrain =
DataUtilities.GetColumnValuesAsDoubleArray(trainingFolds, "Price");
            double m = Statistics.CalculateCoVariance(xTrain, yTrain) /
Statistics.CalculateVariance(xTrain);
            double b = Statistics.CalculateMean(yTrain) - m *
Statistics.CalculateMean(xTrain);
            this.m = m;
            this.b = b;
        }

        protected override void Fit()
        {
            double[] xValidation =
DataUtilities.GetColumnValuesAsDoubleArray(validationFold, xFeature);
            yPredictions = new double[xValidation.Length];
            for (int i=0; i < yPredictions.Length; i++)
            {
                yPredictions[i] = m * xValidation[i] + b;
            }
        }

        public override void EvaluateModelsToDecideOptimalFeatureSelection()
        {
            string[] xFeatures = { "Prod. year scaled", "Mileage in km scaled",
"Engine displacement scaled", "Cylinders scaled", "Levy scaled", "Airbags scaled"
};
            Console.WriteLine("Table showing simple linear model performance for
each feature:");
            Console.WriteLine($"{"Feature",-30} {"MAE",-20} {"RMSE",-20} {"R-
Squared",-20}");
            // tests the model with a different feature each time
            foreach (string feature in xFeatures)
            {
                xFeature = feature;
                double[] errorArray = CrossValidate();
                Console.WriteLine($"{feature,-30} {errorArray[0],-20}
{errorArray[1],-20} {errorArray[2],-20}");
            }
            Console.ReadLine();
        }

        public override string FineTuneModel()
        {
            xFeature = "Prod. year scaled";
            double optimalThreshold = 0;
            double bestRMSE = double.PositiveInfinity;
            double[] bestErrors = new double[4];
            double[] zScoreThresholdValues = { 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6,
6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10 };
            DataTable preppedData = originalData.Copy();

            DataPreprocessor dp = new DataPreprocessor(preppedData);
            dp.RemoveDuplicateRecords();
            List<int> priceOutliers = dp.LocateOutliersWithZScores("Price", 3);
            preppedData.Rows[priceOutliers[1]].Delete();

            // tests the model with different z-score threshold values by 
instantiating a new model each iteration
            foreach (double value in zScoreThresholdValues)
            {
                DataTable tempData = preppedData.Copy();
                DataPreprocessor tempDp = new DataPreprocessor(tempData);
                SimpleLinearRegressionModel tempLinearModel = new 
SimpleLinearRegressionModel(tempData);
                tempLinearModel.xFeature = "Prod. year";

                List<int> prodYearOutliers = 
tempDp.LocateOutliersWithZScores("Prod. year", value);
                int j = 0;
                foreach (int i in prodYearOutliers)
                {
                    tempData.Rows[i - j].Delete();
                    j++;
                }

                tempLinearModel.SplitData();
                double[] errorArray = tempLinearModel.CrossValidate();
                double rmse = errorArray[1];

                if (rmse < bestRMSE)
                {
                    bestRMSE = rmse;
                    errorArray.CopyTo(bestErrors,0);
                    optimalThreshold = value;
                }
                Console.WriteLine($"RMSE: {rmse,-20} Threshold value: {value}");
            }
            Console.WriteLine("Simple linear model conclusion written to file");
            return $"Simple linear regression model: \n" +
                   $"Errors: MAE = {bestErrors[0]}, RMSE = {bestErrors[1]}, R-
Squared = {bestErrors[2]} \n" +
                   $"Hyperparameters: Z-score threshold for Prod. Year outliers = 
{optimalThreshold} \n" +
                   $"Features: {xFeature}";
        }

        protected override void SetFinalHyperparameters()
        {
            DataPreprocessor dp = new DataPreprocessor(data);
            dp.LocateOutliersWithZScores("Prod. year", 4);
            xFeature = "Prod. year scaled";
            noOfPredictors = 1;
        }
    }
}
```

MultiLinearRegressionModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;

namespace RegressionAnalysisProj
```

```
{
    // Class encapsulating the implementation of the multi linear regression
model
    internal class MultiLinearRegressionModel : RegressionModel
    {
        private int maxNoOfEpochs;
        private double learningRate;
        private double convergenceThreshold;
        private double[] weights;
        private double intercept;
        private List<string> xFeatures;
        private int epochCount;
        private bool hasConverged;
        public MultiLinearRegressionModel(DataTable data) : base (data)
        {

        }

        public override void PreProcessData()
        {
            DataPreprocessor dp = new DataPreprocessor(data);
            // initial data checks
            dp.CheckForMissingValues();
            dp.RemoveDuplicateRecords();
            // convert data to numerical form
            dp.SplitEngineVolumeColumn();
            dp.MakeCategoryColumnNumerical();
            dp.MakeColourColumnNumerical();
            dp.MakeDriveWheelsColumnNumerical();
            dp.MakeFuelTypeColumnNumerical();
            dp.MakeGearBoxTypeColumnNumerical();
            dp.MakeLeatherInteriorColumnNumerical();
            dp.MakeMileageColumnNumerical();
            dp.MakeWheelColumnNumerical();
            // locate outliers
            DataUtilities.DisplayColumnNames(data);

            List<int> priceOutliers = dp.LocateOutliersWithZScores("Price", 3);
            data.Rows[priceOutliers[1]].Delete();

            List<int> levyOutliers = dp.LocateOutliersWithZScores("Levy", 5);

            List<int> prodYearOutliers = dp.LocateOutliersWithZScores("Prod.
year", 5);

            List<int> cylindersOutliers =
dp.LocateOutliersWithZScores("Cylinders", 5);

            List<int> airbagsOutliers = dp.LocateOutliersWithZScores("Airbags",
3);

            List<int> engineDisplacementOutliers =
dp.LocateOutliersWithZScores("Engine displacement", 6);
            dp.ImputeNumericalWithMean("Engine displacement",
engineDisplacementOutliers[0]);

            List<int> mileageInKmOutliers =
dp.LocateOutliersWithModifiedZScores("Mileage in km", 3.5);
            int j = 0;
            foreach (int i in mileageInKmOutliers)
            {
                data.Rows[i - j].Delete();
                j++;
            }

            // scale data
            dp.ScaleNumericalColumnStandardisation("Levy");
            dp.ScaleNumericalColumnStandardisation("Prod. year");
            dp.ScaleNumericalColumnStandardisation("Cylinders");
            dp.ScaleNumericalColumnStandardisation("Airbags");
            dp.ScaleNumericalColumnStandardisation("Engine displacement");
            dp.ScaleNumericalColumnStandardisation("Mileage in km");
        }

        public override void SelectFeatures()
        {
            FeatureSelector fs = new FeatureSelector(trainingData);

            // display plots
            string[] continuousColumns = { "Levy scaled", "Prod. year scaled",
"Cylinders scaled", "Airbags scaled", "Engine displacement scaled", "Mileage in
km scaled" };
            List<string> binaryColumns = new List<string>();
            foreach (var column in data.Columns)
            {
                if (column.ToString().StartsWith("Is"))
                {
                    binaryColumns.Add(column.ToString());
                }
            }
            /*
            fs.DisplayBoxPlots(continuousColumns);
            fs.DisplayScatterPlot("Levy");
            fs.DisplayScatterPlot("Prod. year");
            fs.DisplayScatterPlot("Cylinders");
            fs.DisplayScatterPlot("Airbags");
            fs.DisplayScatterPlot("Engine displacement");
            fs.DisplayScatterPlot("Mileage in km");*/

            // correlation coefficients
            fs.RankNumericalColumnsByPearsonCoefficients(continuousColumns);

fs.RankBinaryColumnsByPointBiserialCoefficients(binaryColumns.ToArray());
            Console.ReadLine();
        }

        protected override void Train()
        {
            maxNoOfEpochs = 1000;
            hasConverged = false;
            weights = new double[xFeatures.Count];
            intercept = 0;
            convergenceThreshold = 0.1;
            double previousCost = double.MaxValue;
            double currentCost = 0;

            // Gradient descent iterations
            for (int epoch = 0; epoch < maxNoOfEpochs; epoch++)
            {
                double[] yPredicts = GDPredict();
                currentCost = GDCalculateCost(yPredicts);

                if (Math.Abs(previousCost - currentCost) < convergenceThreshold)
                {
                    //Console.WriteLine($"Loop broken as change in MSE is
sufficiently small after {epoch + 1} loops");
```

```
                    epochCount = epoch + 1;
                    hasConverged = true;
                    break;
                }

                double[] weightDerivatives =
GDCalculateWeightDerivatives(yPredicts);
                double interceptDerivative =
GDCalculateInterceptDerivative(yPredicts);
                for (int i = 0; i < weights.Length; i++)
                {
                    weights[i] -= learningRate * weightDerivatives[i];
                }
                intercept -= learningRate * interceptDerivative;

                previousCost = currentCost;
            }
        }

        // Gets the temporary price predictions using the current weights
        // returns: y predictions
        private double[] GDPredict()
        {
            List<double> yPredicts = new List<double>();
            foreach (DataRow row in trainingFolds.Rows)
            {
                double yPred = intercept;
                for (int i = 0; i < xFeatures.Count; i++)
                {
                    yPred += weights[i] * Convert.ToDouble(row[xFeatures[i]]);
                }
                yPredicts.Add(yPred);
            }
            return yPredicts.ToArray();
        }

        // Calculates the partial derivatives with respect to each weight
        // parmas: current y predictions
        // returns: partial derivatives w.r.t. weight
        private double[] GDCalculateWeightDerivatives(double[] yPredicts)
        {
            List<double> partialDerivatives = new List<double>();
            int noOfDataPoints = trainingFolds.Rows.Count;
            double sumOfProducts = 0;
            foreach (string feature in xFeatures)
            {
                for (int i=0; i<noOfDataPoints; i++)
                {
                    double residual =
(Convert.ToDouble(trainingFolds.Rows[i]["Price"]) - yPredicts[i]);
                    sumOfProducts +=
Convert.ToDouble(trainingFolds.Rows[i][feature]) * residual;
                }
                partialDerivatives.Add(-(2 / (double)noOfDataPoints) *
sumOfProducts);
            }
            return partialDerivatives.ToArray();
        }

        // Calculates the partial derivative with respect to the intercept
        // params: current y predictions
        // returns: partial derivative w.r.t intercept
        private double GDCalculateInterceptDerivative(double[] yPredicts)
        {
            int noOfDataPoints = trainingFolds.Rows.Count;
            double sumOfResiduals = 0;
            for (int i = 0; i < noOfDataPoints; i++)
            {
                sumOfResiduals +=
(Convert.ToDouble(trainingFolds.Rows[i]["Price"]) - yPredicts[i]);
            }
            return (-(2 / (double)noOfDataPoints) * sumOfResiduals);
        }
        // Calculates the current cost function (Mean Square Error)
        // params: y predictions
        // returns: current MSE
        private double GDCalculateCost(double[] yPredicts)
        {
            double sumOfSquaredResiduals = 0;
            for (int i=0; i<yPredicts.Length; i++)
            {
                double yActual =
Convert.ToDouble(trainingFolds.Rows[i]["Price"]);
                double residual = yActual - yPredicts[i];
                sumOfSquaredResiduals += Math.Pow(residual, 2);
            }
            return sumOfSquaredResiduals / yPredicts.Length;
        }

        protected override void Fit()
        {
            yPredictions = new double[validationFold.Rows.Count];
            for (int i = 0; i < yPredictions.Length; i++)
            {
                yPredictions[i] = intercept;
                for (int j = 0; j < xFeatures.Count; j++)
                {
                   yPredictions[i] += weights[j] *
Convert.ToDouble(validationFold.Rows[i][xFeatures[j]]);
                }
            }
        }

        public override void EvaluateModelsToDecideOptimalFeatureSelection()
        {
            List<string> possibleFeatures = new List<string>() { "Prod. year
scaled", "Mileage in km scaled", "Engine displacement scaled",
                "Is jeep", "Is diesel", "Is tiptronic", "Is leather interior",
"Is engine turbo", "Cylinders scaled" }; // ordered by feature importance

            Console.WriteLine("Table showing Multi Linear Gradient Descent
regression model performance for a number of features:");
            Console.WriteLine($"{"Feature no.",-12} {"Added feature",-28}
{"MAE",-14} {"RMSE",-14} {"R-Squared",-14} {"Adj R-Squared",-14} {"Epochs",-6}");
            // Adds a new predictor each time
            for (int i = 0; i < possibleFeatures.Count; i++)
            {
                xFeatures = possibleFeatures.Take(i + 1).ToList();
                noOfPredictors = xFeatures.Count;
                learningRate = 0.1;
                double[] errorArray = CrossValidate();
                Console.WriteLine($"{xFeatures.Count,-12} {possibleFeatures[i],-
28} {errorArray[0],-14} {errorArray[1],-14} {errorArray[2],-14} {errorArray[3],-
14} {epochCount,-6}");
            }
        }
        public override string FineTuneModel()
        {
            xFeatures = new List<string>() { "Prod. year scaled", "Mileage in km
scaled", "Engine displacement scaled",
                "Is jeep", "Is diesel", "Is tiptronic" };
            noOfPredictors = xFeatures.Count;
            double[] possibleLearningRates = { 1, 0.75, 0.5, 0.25, 0.1 };
            double optimalLearningRate = 0;
            double bestEpochCount = double.PositiveInfinity;

            Console.WriteLine("Fine-tuning the learning rate hyperparameter:");
            Console.WriteLine($"{"Learning rate",-18} {"Number of epochs",-18}");
            foreach (double lr in possibleLearningRates)
            {
                learningRate = lr;
                trainingFolds = trainingData;
                Train();
                if (!hasConverged)
                {
                    Console.WriteLine($"{learningRate,-18} {"Not converged",-
18}");
                }
                else
                {
                    Console.WriteLine($"{learningRate,-18} {epochCount,-18}");
                    if (epochCount < bestEpochCount)
                    {
                        optimalLearningRate = learningRate;
                        bestEpochCount = epochCount;
                    }
                }
            }
            learningRate = optimalLearningRate;
            double[] errorArray = CrossValidate();

            string xFeaturesStr = String.Join(", ", xFeatures.ToArray());
            xFeaturesStr.TrimEnd(',', ' ');
            Console.WriteLine("Multi linear model conclusion written to file");
            return $"Multi linear regression model using gradient descent: \n" +
                   $"Errors: MAE = {errorArray[0]}, RMSE = {errorArray[1]}, R-
Squared = {errorArray[2]}, Adjusted R-Squared = {errorArray[3]} \n" +
                   $"Hyperparameters: Learning rate = {optimalLearningRate}, Max
no. of epochs = {maxNoOfEpochs}, Convergence threshold = {convergenceThreshold}
\n" +
                   $"Features: {xFeaturesStr}";
        }

        protected override void SetFinalHyperparameters()
        {
            learningRate = 0.5;
            xFeatures = new List<string>() { "Prod. year scaled", "Mileage in km
scaled", "Engine displacement scaled",
                "Is jeep", "Is diesel", "Is tiptronic" };
            noOfPredictors = 6;
        }
    }
}
```

KNNRegressionModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the implemenatation of the KNN regression model
    internal class KNNRegressionModel : RegressionModel
    {
        List<string> xFeatures;
        private int k;
        public KNNRegressionModel(DataTable data) : base(data)
        {

        }

        public override void PreProcessData()
        {
            DataPreprocessor dp = new DataPreprocessor(data);
            // initial data checks
            dp.CheckForMissingValues();
            dp.RemoveDuplicateRecords();
            // convert data to numerical form
            dp.SplitEngineVolumeColumn();
            dp.MakeMileageColumnNumerical();
            dp.MakeCategoryColumnNumerical();
            dp.MakeColourColumnNumerical();
            dp.MakeDriveWheelsColumnNumerical();
            dp.MakeFuelTypeColumnNumerical();
            dp.MakeGearBoxTypeColumnNumerical();
            dp.MakeLeatherInteriorColumnNumerical();
            dp.MakeWheelColumnNumerical();
            // locate outliers
            DataUtilities.DisplayColumnNames(data);

            List<int> priceOutliers = dp.LocateOutliersWithZScores("Price", 3);
            data.Rows[priceOutliers[1]].Delete();

            List<int> levyOutliers = dp.LocateOutliersWithZScores("Levy", 5);

            List<int> prodYearOutliers = dp.LocateOutliersWithZScores("Prod.
year", 5);

            List<int> cylindersOutliers =
dp.LocateOutliersWithZScores("Cylinders", 5);

            List<int> airbagsOutliers = dp.LocateOutliersWithZScores("Airbags",
3);

            List<int> engineDisplacementOutliers =
dp.LocateOutliersWithZScores("Engine displacement", 6);
            DataUtilities.DisplayRows(data,
engineDisplacementOutliers.ToArray());
            dp.ImputeNumericalWithMean("Engine displacement",
engineDisplacementOutliers[0]);

            List<int> mileageInKmOutliers =
dp.LocateOutliersWithModifiedZScores("Mileage in km", 3);
            int j = 0;
            foreach (int i in mileageInKmOutliers)
            {
                data.Rows[i - j].Delete();
                j++;
            }

            // scale data
            dp.ScaleNumericalColumnMinMax("Levy");
            dp.ScaleNumericalColumnStandardisation("Prod. year");
            dp.ScaleNumericalColumnStandardisation("Cylinders");
            dp.ScaleNumericalColumnStandardisation("Airbags");
            dp.ScaleNumericalColumnStandardisation("Engine displacement");
            dp.ScaleNumericalColumnStandardisation("Mileage in km");

        }

        public override void SelectFeatures()
        {
            FeatureSelector fs = new FeatureSelector(trainingData);

            // display plots
            string[] continuousColumns = { "Levy scaled", "Prod. year scaled",
"Cylinders scaled", "Airbags scaled", "Engine displacement scaled", "Mileage in
km scaled" };
            List<string> binaryColumns = new List<string>();
            foreach (var column in data.Columns)
            {
                if (column.ToString().StartsWith("Is"))
                {
                    binaryColumns.Add(column.ToString());
                }
            }
            /*
            fs.DisplayBoxPlots(continuousColumns);
            fs.DisplayScatterPlot("Levy");
            fs.DisplayScatterPlot("Prod. year");
            fs.DisplayScatterPlot("Cylinders");
            fs.DisplayScatterPlot("Airbags");
            fs.DisplayScatterPlot("Engine displacement");
            fs.DisplayScatterPlot("Mileage in km");*/

            // correlation coefficients
            fs.RankNumericalColumnsByPearsonCoefficients(continuousColumns);

fs.RankBinaryColumnsByPointBiserialCoefficients(binaryColumns.ToArray());
            Console.ReadLine();
        }

        protected override void Train()
        {
            // empty - not really any training process for KNN
        }

        protected override void Fit()
        {
            yPredictions = new double[validationFold.Rows.Count];
            // iterates through each validation query point
            for (int i = 0; i < validationFold.Rows.Count; i++)
            {
                IndexedDistance[] distances = new
IndexedDistance[trainingFolds.Rows.Count];
                // iterates through each data point and calculates a distance
from the query point
```

```
                for (int trainingRowNo = 0; trainingRowNo < distances.Length;
trainingRowNo++)
                {
                    double distance = CalculateDistance(validationFold.Rows[i],
trainingFolds.Rows[trainingRowNo]);
                    IndexedDistance indexedDistance = new
IndexedDistance(trainingRowNo, distance);
                    distances[trainingRowNo] = indexedDistance;
                }

                BubbleSortDescending(distances);
                int[] kNeighbours = new int[k];
                int kIndex = 0;
                for (int j = distances.Length - 1; j >= distances.Length - k; j--
)
                {
                    kNeighbours[kIndex] = distances[j].Index;
                    kIndex++;
                }

                double ySum = 0;
                foreach(int neighbour in kNeighbours)
                {
                    ySum +=
Convert.ToDouble(trainingFolds.Rows[neighbour]["Price"]);
                }
                yPredictions[i] = ySum / k; // prediction is the average of the k
nearest neighbours
            }
        }

        // Nested class for sorting the distances whilst keeping track of the
data point that the distance is associated with
        private class IndexedDistance
        {
            public int Index { get; set; }
            public double Distance { get; set; }

            public IndexedDistance(int index, double distance)
            {
                Index = index;
                Distance = distance;
            }
        }

        // Calculates the distance between two data points (rows)
        // params: data row of query point, data row of data point
        // returns: euclidean distance
        private double CalculateDistance(DataRow row1, DataRow row2)
        {
            double squaredSum = 0;
            foreach (string feature in xFeatures)
            {
                double difference = Convert.ToDouble(row1[feature]) -
Convert.ToDouble(row2[feature]);
                squaredSum += Math.Pow(difference, 2);
            }
            return Math.Sqrt(squaredSum);
        }
        // Sorts the k smallest distances by performing k bubble sort passes
        // params: array of distances
        // returns: sorted array of distances
        private IndexedDistance[] BubbleSortDescending(IndexedDistance[]
distances)
        {
            for (int i=0; i < k; i++)
            {
                for (int j=0; j < distances.Length-i-1; j++)
                {
                    if (distances[j].Distance < distances[j + 1].Distance)
                    {
                        IndexedDistance temp = distances[j];
                        distances[j] = distances[j + 1];
                        distances[j + 1] = temp;
                    }
                }
            }
            return distances;
        }

        // Used for testing the modified bubble sort
        public void BubbleSortTest()
        {
            IndexedDistance[] testArray = new IndexedDistance[10];
            Random random = new Random();
            for (int i= 0; i < 10; i++)
            {
                int num = random.Next(1, 10);
                IndexedDistance a = new IndexedDistance(i, num);
                testArray[i] = a;
            }
            BubbleSortDescending(testArray);
        }

        public override void EvaluateModelsToDecideOptimalFeatureSelection()
        {
            k = 10;
            List<string> possibleFeatures = new List<string>() { "Prod. year
scaled", "Mileage in km scaled", "Engine displacement scaled",
                "Is jeep", "Is diesel", "Is tiptronic", "Is leather interior",
"Is engine turbo", "Cylinders scaled" }; // ordered by feature importance

            Console.WriteLine("Table showing KNN regression model performance for
a number of features:");
            Console.WriteLine($"{"No. of features",-15} {"Added feature",-32}
{"MAE",-18} {"RMSE",-18} {"R-Squared",-18} {"Adj R-Squared",-18}");
            // adds a new feature each time
            for (int i = 0; i < possibleFeatures.Count; i++)
            {
                xFeatures = possibleFeatures.Take(i + 1).ToList();
                noOfPredictors = xFeatures.Count;
                double[] errorArray = CrossValidate();
                Console.WriteLine($"{xFeatures.Count,-15} {possibleFeatures[i],-
32} {errorArray[0],-18} {errorArray[1],-18} {errorArray[2],-18} {errorArray[3],-
18}");
            }
        }

        public override string FineTuneModel()
        {
            xFeatures = new List<string>(){ "Prod. year scaled", "Mileage in km
scaled", "Engine displacement scaled",
                "Is jeep", "Is diesel", "Is tiptronic" };
            double bestRMSE = double.PositiveInfinity;
            double[] bestErrors = new double[4];
            double optimalK = 0;
            int[] kValues = new int[] { 5, 10, 20, 30, 40, 50 };

            Console.WriteLine("Fine-tuning the k value hyperparameter:");
            Console.WriteLine($"{"K value",-10} {"MAE",-18} {"RMSE",-18} {"R-
Squared",-18} {"Adj R-Squared",-18}");
            // test different k values
            for (int i = 0; i < kValues.Length; i++)
            {
                k = kValues[i];
                double[] errorArray = CrossValidate();
                Console.WriteLine($"{k,-10} {errorArray[0],-18} {errorArray[1],-
18} {errorArray[2],-18} {errorArray[3],-18}");
                if (errorArray[1] < bestRMSE)
                {
                    bestRMSE = errorArray[1];
                    errorArray.CopyTo(bestErrors, 0);
                    optimalK = k;
                }
            }

            string xFeaturesStr = String.Join(", ",xFeatures.ToArray());
            xFeaturesStr.TrimEnd(',', ' ');
            Console.WriteLine("KNN model conclusion written to file");
            return $"K-nearest neighbours regression model: \n" +
                   $"Errors: MAE = {bestErrors[0]}, RMSE = {bestErrors[1]}, R-
Squared = {bestErrors[2]}, Adjusted R-Squared = {bestErrors[3]} \n" +
                   $"Hyperparameters: K = {optimalK} \n" +
                   $"Features: {xFeaturesStr}";
        }

        protected override void SetFinalHyperparameters()
        {
            k = 10;
            xFeatures = new List<string>() { "Prod. year scaled", "Mileage in km
scaled", "Engine displacement scaled",
                "Is jeep", "Is diesel", "Is tiptronic" };
            noOfPredictors = 6;
        }
    }
}
```

FastTreeRegressionModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using Microsoft.ML;
using System.IO;
using Newtonsoft.Json;
using System.Text.RegularExpressions;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the implemenation of the Fast tree regression
trainer provided by Microsoft.ML
    internal class FastTreeRegressionModel : RegressionModel
    {
        ITransformer model;
        MLContext mlContext;
        string trainDataPath;
        string testDataPath;
        List<string> xFeatures;
        int minSampleCountPerLeaf;
        public FastTreeRegressionModel(DataTable data) : base(data)
        {
            minSampleCountPerLeaf = 10;
        }

        public override void PreProcessData()
        {
            DataPreprocessor dp = new DataPreprocessor(data);
            // initial data checks
            dp.CheckForMissingValues();
            dp.RemoveDuplicateRecords();
            // convert data to numerical form
            dp.SplitEngineVolumeColumn();
            dp.MakeCategoryColumnNumerical();
            dp.MakeColourColumnNumerical();
            dp.MakeDriveWheelsColumnNumerical();
            dp.MakeFuelTypeColumnNumerical();
            dp.MakeGearBoxTypeColumnNumerical();
            dp.MakeLeatherInteriorColumnNumerical();
            dp.MakeMileageColumnNumerical();
            dp.MakeWheelColumnNumerical();

            // locate outliers
            DataUtilities.DisplayColumnNames(data);

            List<int> priceOutliers = dp.LocateOutliersWithZScores("Price", 3);
            DataUtilities.DisplayRows(data, priceOutliers.ToArray());
            data.Rows[priceOutliers[1]].Delete();

            List<int> levyOutliers = dp.LocateOutliersWithZScores("Levy", 5);

            List<int> prodYearOutliers = dp.LocateOutliersWithZScores("Prod. 
year", 3);
            DataUtilities.DisplayRows(data, prodYearOutliers.ToArray());
            int j = 0;
            foreach (int i in prodYearOutliers)
            {
                data.Rows[i - j].Delete();
                j++;
            }

            List<int> cylindersOutliers = 
dp.LocateOutliersWithZScores("Cylinders", 5);

            List<int> airbagsOutliers = dp.LocateOutliersWithZScores("Airbags", 
3);

            List<int> engineDisplacementOutliers = 
dp.LocateOutliersWithZScores("Engine displacement", 6);
            dp.ImputeNumericalWithMean("Engine displacement", 
engineDisplacementOutliers[0]);

            List<int> mileageInKmOutliers = 
dp.LocateOutliersWithModifiedZScores("Mileage in km", 3.5);
            j = 0;
            foreach (int i in mileageInKmOutliers)
            {
                data.Rows[i - j].Delete();
                j++;
            }
            Console.ReadLine();

            // Manufacturers which appear less than 10 times given 'Other' value
as there are not enough data points
            Dictionary<string,int> uniqueManufacturers = new
Dictionary<string,int>();
            foreach (DataRow row in data.Rows)
            {
                if
(uniqueManufacturers.ContainsKey(row["Manufacturer"].ToString()))
                {
                    uniqueManufacturers[row["Manufacturer"].ToString()]++;
                }
                else
                {
                    uniqueManufacturers.Add(row["Manufacturer"].ToString(), 1);
                }
            }
            int rowNo = 0;
            foreach (var kvp in uniqueManufacturers)
            {
                if (kvp.Value < 10)
                {
                    data.Rows[rowNo]["Manufacturer"] = "OTHER";
                }
                rowNo++;
            }


            // scale data
            dp.ScaleNumericalColumnStandardisation("Levy");
            dp.ScaleNumericalColumnStandardisation("Prod. year");
            dp.ScaleNumericalColumnStandardisation("Cylinders");
            dp.ScaleNumericalColumnStandardisation("Airbags");
            dp.ScaleNumericalColumnStandardisation("Engine displacement");
            dp.ScaleNumericalColumnStandardisation("Mileage in km");
        }

        public override void SelectFeatures()
        {
            // feature selection
            FeatureSelector fs = new FeatureSelector(trainingData);

            // display plots
            string[] continuousColumns = { "Levy scaled", "Prod. year scaled",
"Cylinders scaled", "Airbags scaled", "Engine displacement scaled", "Mileage in
km scaled" };
            List<string> binaryColumns = new List<string>();
            foreach (var column in data.Columns)
            {
                if (column.ToString().StartsWith("Is"))
                {
                    binaryColumns.Add(column.ToString());
                }
            }
            /*
            fs.DisplayBoxPlots(continuousColumns);
            fs.DisplayScatterPlot("Levy");
            fs.DisplayScatterPlot("Prod. year");
            fs.DisplayScatterPlot("Cylinders");
            fs.DisplayScatterPlot("Airbags");
            fs.DisplayScatterPlot("Engine displacement");
            fs.DisplayScatterPlot("Mileage in km");*/

            // correlation coefficients
            fs.RankNumericalColumnsByPearsonCoefficients(continuousColumns);

fs.RankBinaryColumnsByPointBiserialCoefficients(binaryColumns.ToArray());
            Console.ReadLine();
        }
        protected override void Train()
        {
            trainDataPath = "prepped-training-data.csv";
            WriteDataToCsv(trainingFolds, trainDataPath);
            mlContext = new MLContext(seed: 0);
            IDataView dataView =
mlContext.Data.LoadFromTextFile<CarData>(trainDataPath, hasHeader: true,
separatorChar: ',');

            var pipeline = mlContext.Transforms.CopyColumns(outputColumnName:
"Label", inputColumnName: "Price")

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"ManufacturerEncoded", inputColumnName: "Manufacturer"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"ModelEncoded", inputColumnName: "Model"))
                .Append(mlContext.Transforms.Concatenate("Features",
xFeatures.ToArray())) // change predictors by passing the corresponding
attributes

.Append(mlContext.Regression.Trainers.FastTree(minimumExampleCountPerLeaf:
minSampleCountPerLeaf));

            model = pipeline.Fit(dataView);
        }

        // Writes data to a csv file
        // params: data, file path
        private void WriteDataToCsv(DataTable data, string filePath)
        {
            using (StreamWriter sw = new StreamWriter(filePath))
            {
                // write the column names as the header row
                sw.WriteLine(string.Join(",",
data.Columns.Cast<DataColumn>().Select(col => col.ColumnName)));

                // write each row's values of data
                foreach (DataRow row in data.Rows)
                {
                    sw.WriteLine(String.Join(",", row.ItemArray));
                }
            }
        }
        protected override void Fit()
        {
            testDataPath = "validation-data.csv";
            WriteDataToCsv(validationFold, testDataPath);

            IDataView dataView =
mlContext.Data.LoadFromTextFile<CarData>(testDataPath, hasHeader: true,
separatorChar: ',');

            var modelPredictions = model.Transform(dataView);
            var predictedValues =
mlContext.Data.CreateEnumerable<CarPricePrediction>(modelPredictions,
reuseRowObject: false);

            yPredictions = new double[validationFold.Rows.Count];
            int i = 0;
            foreach (var prediction in predictedValues)
            {
                yPredictions[i] = prediction.Price;
                i++;
            }

        }

        // Fits the training data (instead of validation) for the purpose of
viewing if the decision tree overfits the training data
        public void FitTrainingData()
        {

            IDataView dataView =
mlContext.Data.LoadFromTextFile<CarData>(trainDataPath, hasHeader: true,
separatorChar: ',');

            var modelPredictions = model.Transform(dataView);

            var records = mlContext.Data.CreateEnumerable<CarData>(dataView,
reuseRowObject: false);

            var predictedValues =
mlContext.Data.CreateEnumerable<CarPricePrediction>(modelPredictions,
reuseRowObject: false);

            int i = 0;
            List<double> preds = new List<double>();
            List<double> actual = new List<double>();
            foreach (var prediction in predictedValues)
            {
                preds.Add(prediction.Price);
                i++;
            }
            foreach (var record in records)
            {
                actual.Add(record.Price);
            }

            ModelValidator mv = new ModelValidator(preds.ToArray(),
actual.ToArray());
            double mae = Math.Round(mv.CalculateMAE(),6);
            double rmse = Math.Round(mv.CalculateRMSE(),6);
            double r2 = Math.Round(mv.CalculateRSquared(),6);
            double adjR2 =
Math.Round(mv.CalculateAdjustedRSquared(noOfPredictors), 6);
            Console.WriteLine($"{"Training",-12} {"",-16} {"",-20} {mae,-14}
{rmse,-14} {r2,-14} {adjR2,-14}");
        }

        public override void EvaluateModelsToDecideOptimalFeatureSelection()
        {
            // decision trees prone to overffitting so I have implemented a
comparison of the model's performance on the training data vs unseen validation
data
            List<string> possibleFeatures = new List<string>() { "ProdYear",
"MileageInKm", "EngineDisplacement", "ManufacturerEncoded", "ModelEncoded",
                "IsJeep", "IsDiesel", "IsTiptronic", "IsLeatherInterior",
"IsEngineTurbo", "IsWheelLeft", "IsHatchback", "IsSedan", "IsManual", "Is4x4",
                "IsHybrid", "IsFront", "IsBlack", "IsSilver", "IsAutomatic",
"IsGreen", "IsGoodsWagon", "IsLPG", "IsUniversal", "IsRed",
                "IsVariator", "IsPetrol" }; // ordered by feature importance

            Console.WriteLine("Table showing Fast tree regression trainer
performance for a number of features:");
            Console.WriteLine($"{"Data",-12} {"No. of features",-16} {"Added
feature", -20} {"MAE",-14} {"RMSE",-14} {"R-Sqaured",-14} {"Adj R-squared",-
14}");
            // adds new feature each time
            for (int i=0; i<possibleFeatures.Count; i++)
            {
                xFeatures = possibleFeatures.Take(i + 1).ToList();
                noOfPredictors = xFeatures.Count;
                double[] errorArray = CrossValidate();
                Console.WriteLine($"{"Validation",-12} {xFeatures.Count,-16}
{possibleFeatures[i],-20} {errorArray[0],-14} {errorArray[1],-14}
{errorArray[2],-14} {errorArray[3],-14}");
                FitTrainingData();
            }

        }

        public override string FineTuneModel()
        {
            xFeatures = new List<string>() { "ProdYear", "MileageInKm",
"EngineDisplacement", "ManufacturerEncoded", "ModelEncoded",
                "IsJeep", "IsDiesel", "IsTiptronic", "IsLeatherInterior",
"IsEngineTurbo" };
            noOfPredictors = xFeatures.Count;
            int[] samplesPerLeaf = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20 };
            int optimalValue = 0;
            double bestRMSE = double.PositiveInfinity;
            double[] bestErrors = new double[4];

            Console.WriteLine("Fine-tuning the minimum sample count per leaf
hyperparameter:");
            Console.WriteLine($"{"Min. samples per leaf",-24} {"MAE",-18}
{"RMSE",-18} {"R-Squared",-18} {"Adj R-Squared",-18}");
            // tests different values for the samples per leaf
            foreach (int value in samplesPerLeaf)
            {
                minSampleCountPerLeaf = value;
                double[] errorArray = CrossValidate();
                Console.WriteLine($"{value,-24} {errorArray[0],-18}
{errorArray[1],-18} {errorArray[2],-18} {errorArray[3],-18}");
                if (errorArray[1] < bestRMSE)
                {
                    bestRMSE = errorArray[1];
                    errorArray.CopyTo(bestErrors, 0);
                    optimalValue = value;
                }
            }

            string xFeaturesStr = String.Join(", ", xFeatures.ToArray());
            xFeaturesStr.TrimEnd(',', ' ');
            Console.WriteLine("Fast tree model conclusion written to file");
            return $"Microsft.ML Fast tree regression trainer: \n" +
                   $"Errors: MAE = {bestErrors[0]}, RMSE = {bestErrors[1]}, R-
Squared = {bestErrors[2]}, Adjusted R-Squared = {bestErrors[3]} \n" +
                   $"Hyperparameters: Minimum sample count per leaf = 
{optimalValue} ***Inconclusive value - check with holdout data*** \n" +
                   $"Features: {xFeaturesStr}";
        }

        protected override void SetFinalHyperparameters()
        {
            minSampleCountPerLeaf = 8;
            xFeatures = new List<string>() { "ProdYear", "MileageInKm", 
"EngineDisplacement", "ManufacturerEncoded", "ModelEncoded",
                "IsJeep", "IsDiesel", "IsTiptronic", "IsLeatherInterior", 
"IsEngineTurbo" };
            noOfPredictors = 10;
        }

        // Uses holdout data to verify the optimal value for the number of 
samples per leaf hyperparameter
        public void DetermineHyperparameterWithHoldoutData()
        {
            trainingFolds = trainingData.Clone();
            trainingFolds.Merge(trainingData);
            trainingFolds.Merge(validationData);
            validationFold = holdoutData.Copy();
            xFeatures = new List<string>() { "ProdYear", "MileageInKm", 
"EngineDisplacement", "ManufacturerEncoded", "ModelEncoded",
                "IsJeep", "IsDiesel", "IsTiptronic", "IsLeatherInterior", 
"IsEngineTurbo" };
            noOfPredictors = xFeatures.Count;
            int[] samplesPerLeaf = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 
12, 13, 14, 15, 16, 17, 18, 19, 20 };
            int optimalValue = 0;
            double bestRMSE = double.PositiveInfinity;
            double[] bestErrors = new double[4];

            Console.WriteLine("Fine-tuning the minimum sample count per leaf 
hyperparameter:");
            Console.WriteLine($"{"Min. samples per leaf",-24} {"MAE",-18} 
{"RMSE",-18} {"R-Squared",-18} {"Adj R-Squared",-18}");
            // tests different values for the samples per leaf
            foreach (int value in samplesPerLeaf)
            {
                minSampleCountPerLeaf = value;
                double[] errorArray = CrossValidate();
                Console.WriteLine($"{value,-24} {errorArray[0],-18} 
{errorArray[1],-18} {errorArray[2],-18} {errorArray[3],-18}");
                if (errorArray[1] < bestRMSE)
                {
                    bestRMSE = errorArray[1];
                    errorArray.CopyTo(bestErrors, 0);
                    optimalValue = value;
                }
            }
        }

        // Saves trained model to zip file for UI integration
        public void SaveTrainedModelToFile()
        {
            minSampleCountPerLeaf = 8;
            trainingFolds = data;
            xFeatures = new List<string>() {"ProdYear", "MileageInKm", 
"EngineDisplacement", "ManufacturerEncoded", "ModelEncoded",
                "IsJeep", "IsDiesel", "IsTiptronic", "IsLeatherInterior", 
"IsEngineTurbo"};
```

```
            Train();
            mlContext.Model.Save(model, null, "trained-model.zip");
            Console.WriteLine("Trained fast tree model has been written to file:
trained-model.zip");
        }
        // Saves the makes and models to a json file for UI integration
        public void SaveMakeAndModelToFile()
        {
            DataColumn makeColumn = data.Columns["Manufacturer"];
            DataColumn modelColumn = data.Columns["Model"];
            Dictionary<string, List<string>> makesAndModels = new
Dictionary<string, List<string>>();
            foreach(DataRow row in data.Rows)
            {
                string makeModel = row["Manufacturer"].ToString() + " " +
row["Model"].ToString();
                bool containsOnlyAscii = ContainsOnlyAsciiCharacters(makeModel);
                if (containsOnlyAscii && !Regex.IsMatch(makeModel, "_+"))
                {
                    string key = row["Manufacturer"].ToString();
                    if
(!makesAndModels.ContainsKey(row["Manufacturer"].ToString()))
                    {
                        List<string> value = new List<string> {
row["Model"].ToString() };
                        makesAndModels.Add(key, value);
                    }
                    else if
(!makesAndModels[key].Contains(row["Model"].ToString()))
                    {
                        makesAndModels[key].Add(row["Model"].ToString());
                    }
                }
                else
                {
                    Console.WriteLine($"Did not write '{makeModel}' to file.");
                }
            }

            string json = JsonConvert.SerializeObject(makesAndModels,
Formatting.Indented);

            string jsonFilePath = "make-model-data.json";
            File.WriteAllText(jsonFilePath, json);
            Console.WriteLine($"Make and model data has been written to file:
{jsonFilePath}");
        }

        // Checks that a string contains only ascii characters
        // params: input string
        // returns: boolean for whether it contains ascii or not
        private bool ContainsOnlyAsciiCharacters(string value)
        {
            bool containsOnlyAscii = true;
            foreach (char c in value)
            {
                if (c > 127)
                {
                    containsOnlyAscii = false;
                }
            }
            return containsOnlyAscii;
        }
```

```
    }
}
```

MicrosoftMLCars.cs

```
using Microsoft.ML.Data;

namespace RegressionAnalysisProj
{
    // Class containing all the car data attributes for Microsoft.ML
implementation
    public class CarData
    {
        [LoadColumn(0)]
        public float Price;

        [LoadColumn(1)]
        public float Levy;

        [LoadColumn(2)]
        public string Manufacturer;

        [LoadColumn(3)]
        public string Model;

        [LoadColumn(4)]
        public float ProdYear;

        [LoadColumn(5)]
        public float Cylinders;

        [LoadColumn(6)]
        public float Airbags;

        [LoadColumn(7)]
        public float EngineDisplacement;

        [LoadColumn(8)]
        public float IsEngineTurbo;

        [LoadColumn(9)]
        public float IsCabriolet;

        [LoadColumn(10)]
        public float IsCoupe;

        [LoadColumn(11)]
        public float IsGoodsWagon;

        [LoadColumn(12)]
        public float IsHatchback;

        [LoadColumn(13)]
        public float IsJeep;

        [LoadColumn(14)]
        public float IsLimousine;

        [LoadColumn(15)]
        public float IsMicrobus;
        [LoadColumn(16)]
        public float IsMinivan;

        [LoadColumn(17)]
        public float IsPickup;

        [LoadColumn(18)]
        public float IsSedan;

        [LoadColumn(19)]
        public float IsUniversal;

        [LoadColumn(20)]
        public float IsBeige;

        [LoadColumn(21)]
        public float IsBlack;

        [LoadColumn(22)]
        public float IsBlue;

        [LoadColumn(23)]
        public float IsBrown;

        [LoadColumn(24)]
        public float IsCarnelianRed;

        [LoadColumn(25)]
        public float IsGolden;

        [LoadColumn(26)]
        public float IsGreen;

        [LoadColumn(27)]
        public float IsGrey;

        [LoadColumn(28)]
        public float IsOrange;

        [LoadColumn(29)]
        public float IsPink;

        [LoadColumn(30)]
        public float IsPurple;

        [LoadColumn(31)]
        public float IsRed;

        [LoadColumn(32)]
        public float IsSilver;

        [LoadColumn(33)]
        public float IsSkyBlue;

        [LoadColumn(34)]
        public float IsWhite;

        [LoadColumn(35)]
        public float IsYellow;

        [LoadColumn(36)]
        public float Is4x4;
        [LoadColumn(37)]
        public float IsFront;

        [LoadColumn(38)]
        public float IsRear;

        [LoadColumn(39)]
        public float IsCNG;

        [LoadColumn(40)]
        public float IsDiesel;

        [LoadColumn(41)]
        public float IsHybrid;

        [LoadColumn(42)]
        public float IsHydrogen;

        [LoadColumn(43)]
        public float IsLPG;

        [LoadColumn(44)]
        public float IsPetrol;

        [LoadColumn(45)]
        public float IsPlugInHybrid;

        [LoadColumn(46)]
        public float IsAutomatic;

        [LoadColumn(47)]
        public float IsManual;

        [LoadColumn(48)]
        public float IsTiptronic;

        [LoadColumn(49)]
        public float IsVariator;

        [LoadColumn(50)]
        public float IsLeatherInterior;

        [LoadColumn(51)]
        public float MileageInKm;

        [LoadColumn(52)]
        public float IsWheelLeft;

        [LoadColumn(53)]
        public float LevyScaled;

        [LoadColumn(54)]
        public float ProdYearScaled;

        [LoadColumn(55)]
        public float CylindersScaled;

        [LoadColumn(56)]
        public float AirbagsScaled;

        [LoadColumn(57)]
        public float EngineDisplacementScaled;
        [LoadColumn(58)]
        public float MileageInKmScaled;

    }

    // Class containing the price prediction atribute for Microsoft.ML 
implementation
    public class CarPricePrediction
    {
        [ColumnName("Score")]
        public float Price;
    }
}
```

ModelValidator.cs

```
using System;

namespace RegressionAnalysisProj
{
    // Class that encapsulates the methods for calculating regression error 
metrics
    internal class ModelValidator
    {
        private double[] yPred;
        private double[] yActual;
        public ModelValidator(double[] yPredictions, double[] yActual)
        {
            yPred = yPredictions;
            this.yActual = yActual;
        }

        // Calculates the Mean Absolute Error (MAE)
        // returns: MAE
        public double CalculateMAE()
        {
            if (yActual.Length != yPred.Length)
            {
                throw new Exception("Error. Different number of actual and 
predicted values");
            }
            double sumOfAbsResiduals = 0;
            for (int i = 0; i < yActual.Length; i++)
            {
                sumOfAbsResiduals += Math.Abs(yActual[i] - yPred[i]);
            }
            double mae = sumOfAbsResiduals / yActual.Length;
            return mae;
        }

        // Calculates the Root Mean Square Error (RMSE)
        // returns: RMSE
        public double CalculateRMSE()
        {
            double rmse = Math.Sqrt(CalculateMSE());
            return rmse;
        }

        // Calculates the Mean Square Error (MSE)
        // returns: MSE
        private double CalculateMSE()
        {
            if (yActual.Length != yPred.Length)
            {
                throw new Exception("Error. Different number of actual and 
predicted values");
            }
            double sumOfSqrResiduals = 0;
            for (int i = 0; i < yActual.Length; i++)
            {
                sumOfSqrResiduals += Math.Pow(yActual[i] - yPred[i], 2);
            }
            double mse = sumOfSqrResiduals / yActual.Length;
            return mse;
        }

        // Calculates the coeffecition of determination (r2)
        // returns: r2
        public double CalculateRSquared()
        {
            if (yActual.Length != yPred.Length)
            {
                throw new Exception("Error. Different number of actual and 
predicted values");
            }
            double actualMean = Statistics.CalculateMean(yActual);
            double sumOfSqrResiduals = 0;
            double totalVariance = 0;
            for (int i = 0; i < yActual.Length; i++)
            {
                sumOfSqrResiduals += Math.Pow(yActual[i] - yPred[i], 2);
                totalVariance += Math.Pow(yActual[i] - actualMean, 2);
            }
            double r2 = 1 - sumOfSqrResiduals / totalVariance;
            return r2;
        }

        // Calculates the Adjusted R-squared value
        // params: number of predictors
        // returns: adjusted r2
        public double CalculateAdjustedRSquared(int noOfPredictors)
        {
            if (yActual.Length != yPred.Length)
            {
                throw new Exception("Error. Different number of actual and 
predicted values");
            }
            double r2 = CalculateRSquared();
            int n = yActual.Length;
            double adjR2 = 1 - (1 - r2) * ((double)n - 1) / ((double)n - 
(double)noOfPredictors - 1);
            return adjR2;
        }

    }
}
```

Statistics.cs

```
using System;
using System.Collections.Generic;
```

```
namespace RegressionAnalysisProj
{
    // Static class containing methods for a range of statistical calculations
    static internal class Statistics
    {
        // Calculates the mean of an array of values
        // params: array of values
        // returns: mean
        public static double CalculateMean(double[] values)
        {
            double sumOfValues = 0;
            foreach(double value in values)
            {
                sumOfValues += value;
            }
            double mean = sumOfValues / values.Length;
            return mean;
        }

        // Calculates the mode of an array of values
        // params: array of values
        // returns: mode
        public static double CalculateMode(double[] values)
        {
            Dictionary <double, int> valueCounter = new Dictionary<double,
int>();
            foreach(double number in values)
            {
                if (valueCounter.ContainsKey(number))
                {
                    valueCounter[number]++;
                }

                else
                {
                    // adds a new key to dictionary and sets its value to 1
                    valueCounter[number] = 1;
                }
            }
            double mode = 0;
            int maxCount = 0;
            foreach (var pair in valueCounter)
            {
                if (pair.Value > maxCount)
                {
                    maxCount = pair.Value;
                    mode = pair.Key;
                }
            }
            return mode;
        }

        // Calculates the median of an array of values
        // If no. of values is odd, middle value is taken; if it is even, the
mean from the two middle values is taken
        // params: array of values
        // returns: median
        public static double CalculateMedian(double[] values)
        {
            double[] sortedValues = new double[values.Length];
            Array.Copy(values, sortedValues, values.Length);
            MergeSort(sortedValues);
            int i = 0;
            int n = sortedValues.Length;
            double median = 0;
            if (n % 2 == 1)
            {
                i = ((n + 1) / 2) - 1;
                median = sortedValues[i];
            }
            else
            {
                i = n / 2;
                double[] array = { sortedValues[i-1], sortedValues[i] };
                median = CalculateMean(array);
            }
            return median;
        }

        // Calculates the population variance of an array of values
        // params: values
        // returns: variance
        public static double CalculateVariance(double[] values)
        {
            double[] squaredDifferences = new double[values.Length];
            double mean = CalculateMean(values);
            for (int i = 0; i < squaredDifferences.Length; i++)
            {
                double squaredDifference = Math.Pow((values[i] - mean), 2);
                squaredDifferences[i] = squaredDifference;
            }
            double variance = CalculateMean(squaredDifferences);
            return variance;
        }

        // Calcualtes the standard deviation of an array of values
        // params: array of values
        // returns: standard deviation
        public static double CalculateStdDev(double[] values)
        {
            double stdDev = Math.Pow(CalculateVariance(values), 0.5);
            return stdDev;
        }

        // Calculates the covariance for two corresponding arrays of values (of a
different variable)
        // params: array of values of variable x, array of values of variable y
        // returns: covariance
        public static double CalculateCoVariance(double[] xValues, double[]
yValues)
        {
            double xMean = CalculateMean(xValues);
            double yMean = CalculateMean(yValues);
            double sumOfProducts = 0;
            for (int i = 0; i < xValues.Length; i++)
            {
                double xDifference = xValues[i] - xMean;
                double yDifference = yValues[i] - yMean;
                sumOfProducts += (xDifference * yDifference);
            }
            double covariance = sumOfProducts / (xValues.Length - 1);
            return covariance;
        }

        // Calculates the z-score for a value
        // params: value, mean, standard deviation
        // returns: z-score
        public static double CalculateZScore(double value, double mean, double
stdDev)
        {
            double zScore = (value - mean) / stdDev;
            return zScore;
        }

        // Calculates the median absolute deviation of an array of values
        // params: array of values
        // returns: median absolute deviation
        public static double CalculateMedianAbsDev(double[] values)
        {
            double median = CalculateMedian(values);
            double[] absoluteDeviations = new double[values.Length];
            for (int i=0; i<values.Length; i++)
            {
                absoluteDeviations[i] = Math.Abs(values[i]-median);
            }
            double mad = CalculateMedian(absoluteDeviations);
            return mad;
        }

        // Calculates the modified z-score for a value
        // params: value, median, median absolute deviation
        // returns: modified z-score
        public static double CalculateModifiedZScore(double value, double median,
double mad)
        {
            double modZScore = (0.6745 * (value - median)) / mad;
            return modZScore;
        }

        // Calculates a given percentile for an array of values
        // params: array of sorted values, percentile
        // returns: the value at that percentile
        public static double CalculatePercentile(double[] sortedValues, double
percentile)
        {
            int n = sortedValues.Length;
            double index = (n - 1) * percentile;
            int lowerIndex = (int)Math.Floor(index); // rounds down to nearest
integer
            int upperIndex = (int)Math.Ceiling(index); // rounds up to nearest
integer

            if (lowerIndex == upperIndex)
            {
                return sortedValues[lowerIndex];
            }
            else
            {
                double lowerValue = sortedValues[lowerIndex];
                double upperValue = sortedValues[upperIndex];
                return lowerValue + (index - lowerIndex) * (upperValue -
lowerValue);
            }
        }
        // Calculates the pearson correlation coefficient for an array of x
values according to an array of y values
        // params: array of values of x variable, array of values of y variable
        // returns: pearson correlation coefficient
        public static double CalculatePearsonCorrelationCoefficient(double[]
xValues, double[] yValues)
        {
            double xMean = CalculateMean(xValues);
            double yMean = CalculateMean(yValues);
            double xDifference, yDifference, productOfDifferences, sumOfProducts
= 0, xSumOfSquaredDifferences = 0, ySumOfSquaredDifferences = 0;
            double pearsonCoeff;
            int n = xValues.Length;
            for (int i = 0; i < n; i++)
            {
                xDifference = xValues[i] - xMean;
                yDifference = yValues[i] - yMean;
                productOfDifferences = xDifference * yDifference;
                sumOfProducts += productOfDifferences;
                xSumOfSquaredDifferences += Math.Pow(xDifference, 2);
                ySumOfSquaredDifferences += Math.Pow(yDifference, 2);
            }
            pearsonCoeff = sumOfProducts / Math.Pow(xSumOfSquaredDifferences *
ySumOfSquaredDifferences, 0.5);
            return pearsonCoeff;
        }

        // Calculates the point biserial correlation coefficient for an array of
dichotomous x values according to an array of continous y values
        // params: array of values of dichotomous x variable, array of values of
continous y variable
        // returns: point biserial correlation coefficient
        public static double
CalculatePointBiserialCorrelationCoefficient(double[] xBinaryValues, double[]
yValues)
        {
            List<double> yValuesWith0 = new List<double>();
            List<double> yValuesWith1 = new List<double>();
            for (int i=0; i<yValues.Length; i++)
            {
                if (xBinaryValues[i] == 0)
                {
                    yValuesWith0.Add(yValues[i]);
                }
                else if (xBinaryValues[i] == 1)
                {
                    yValuesWith1.Add(yValues[i]);
                }
            }
            double meanY0 = CalculateMean(yValuesWith0.ToArray());
            double meanY1 = CalculateMean(yValuesWith1.ToArray());
            double stdDev = CalculateStdDev(yValues);
            int n0 = yValuesWith0.Count;
            int n1 = yValuesWith1.Count;
            int n = yValues.Length;
            double correctionFactor = Math.Sqrt(n0 * n1 / (n * (n - 1.0)));
            double coeff = (meanY1 - meanY0) * correctionFactor / stdDev;
            return coeff;
        }

        // Sorts an array into ascending order using the merge sort algorithm
        // params: array of values of the double datatype
        public static void MergeSort(double[] array)
        {
            if (array.Length <= 1)
            {
                return;
            }
            int mid = array.Length / 2;
            double[] leftArr = new double[mid];
            double[] rightArr = new double[array.Length - mid];

            Array.Copy(array, leftArr, mid);
            Array.Copy(array, mid, rightArr, 0, array.Length - mid);

            // Recursive calls to sort the two sub-arrays
            MergeSort(leftArr);
            MergeSort(rightArr);

            // Merge the two sorted sub-arrays
            Merge(array, leftArr, rightArr);
        }

        // Merges two sub-arrays into the original array
        // params: original array, left sub-array, right sub-array
        private static void Merge(double[] array, double[] leftArr, double[]
rightArr)
        {
            int i = 0, j = 0, k = 0;

            // Compares elements from two arrays and stops merging once one of
the entire arrays have been merged
            while (i < leftArr.Length && j < rightArr.Length)
            {
                if (leftArr[i] <= rightArr[j])
                {
                    array[k] = leftArr[i];
                    i++;
                }
                else
                {
                    array[k] = rightArr[j];
                    j++;
                }
                k++;
            }

            // Remaining elements copied (if any)
            while (i < leftArr.Length)
            {
                array[k] = leftArr[i];
                i++;
                k++;
            }
            while (j < rightArr.Length)
            {
                array[k] = rightArr[j];
                j++;
                k++;
            }
        }
    }
}
```

DataUtilities.cs

```
using System;
using System.Data;
using System.Linq;

namespace RegressionAnalysisProj
{
    // Static class containing useful fuctionality regarding data
    static internal class DataUtilities
    {
        // Returns an array of values (of double datatype) in a given column
        // params: data, column name
        // returns: array of columns values
        public static double[] GetColumnValuesAsDoubleArray(DataTable data,
string columnName)
        {
            double[] array = new double[data.Rows.Count];
            for (int i = 0; i < array.Length; i++)
            {
                array[i] = Convert.ToDouble(data.Rows[i][columnName]);
            }
            return array;
        }

        // Displays a given number of rows of data
        // params: data, number of rows
        public static void DisplayData(DataTable data, int noOfRows)
        {
            int rowNo = 0;
            foreach (DataRow row in data.Rows)
            {
                Console.WriteLine();
                foreach (DataColumn column in data.Columns)
                {
                    Console.Write($"{row[column],-20}");
                    if (row[column] == null)
                    {
                        Console.WriteLine("***null value***");
                    }
                }
                Console.WriteLine();
                rowNo++;
                if (rowNo == noOfRows)
                {
                    break;
                }
            }
        }

        // Displays given rows
        // params: data, row numbers
        public static void DisplayRows(DataTable data, int[] rowNumbers)
        {
            Console.WriteLine();
            foreach (int rowNo in rowNumbers)
            {
                foreach (DataColumn column in data.Columns)
                {
                    Console.Write($"{data.Rows[rowNo][column],-20}");
                    if (data.Rows[rowNo][column] == null)
                    {
                        Console.WriteLine("***null value***");
                    }
                }
                Console.WriteLine();
            }
        }

        // Displays all column names of the data
        // params: data
        public static void DisplayColumnNames(DataTable data)
        {
            foreach (DataColumn column in data.Columns)
            {
                Console.Write($"{column.ColumnName,-20}");
            }
            Console.WriteLine();
        }

        // Splits data into training, validation and holdout data.
        // params: data, trainingRatio, validationRatio
        // returns: array containing training, validationa and holdout datatables
        public static DataTable[] SplitData(DataTable data, double trainingRatio,
double validationRatio)
        {
            int rowCount = data.Rows.Count;
            int trainingRowCount = (int)Math.Round(rowCount * trainingRatio);
            int validationRowCount = (int)Math.Round(rowCount * validationRatio);
            Random rnd = new Random(1); // 1 is for fixed random seed so the data
division does not affect regression model performance
            var shuffledRows = data.AsEnumerable().OrderBy(row =>
rnd.Next()).ToArray(); // shuffled the rows randomly into an array
            DataTable trainingData = data.Clone();
            DataTable validationData = data.Clone();
            DataTable holdoutData = data.Clone();
            for (int i = 0; i < rowCount; i++)
            {
                if (i < trainingRowCount)
                {
                    trainingData.ImportRow(shuffledRows[i]);
                }
                else if (i < trainingRowCount + validationRowCount)
                {
                    validationData.ImportRow(shuffledRows[i]);
                }
                else
                {
                    holdoutData.ImportRow(shuffledRows[i]);
                }
            }
            DataTable[] dataArray = { trainingData, validationData, holdoutData
};
            return dataArray;
        }


        // Divides data into 4 equal subsets
        // params: data
        // returns: array of divided datatables
        public static DataTable[] DivideDataInto4(DataTable data)
        {
            int rowCount = data.Rows.Count;
            Random rnd = new Random(1); // 1 is for fixed random seed so the data
division does not affect regression model performance
            var shuffledRows = data.AsEnumerable().OrderBy(row =>
rnd.Next()).ToArray(); // shuffled the rows randomly into an array
            int splitSize = rowCount / 4;
            int remainder = rowCount % 4;
            int startIndex = 0;
            DataTable[] dataArray = new DataTable[4];
            for (int i = 0; i < 4; i++)
            {
                int endIndex = startIndex + splitSize;
                if (i < remainder)
                {
                    endIndex++;
                }
                DataTable dataSubset = data.Clone();
                for (int j = startIndex; j < endIndex; j++)
                {
                    dataSubset.ImportRow(shuffledRows[j]);
                }
                dataArray[i] = dataSubset;
                startIndex = endIndex;
            }
            return dataArray;
        }
    }
}
```

WindowsFormsUI

Form1.cs (NOT Package-Generated)

```
using RegressionAnalysisProj;
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.ML;
using Newtonsoft.Json;
using System.IO;
using System.Drawing.Printing;

namespace WindowsFormsUI
{
    public partial class form1 : Form
    {
        private Dictionary<string, List<string>> makeModelData;
        private PrintDocument printDocument;
        List<string> makes;
        private PredictionEngine<CarData, CarPricePrediction> predictionFunction;
        private const Single MilesToKmMultiplier = 1.60934f;
        private const int DefaultProdYear = 2020;
        private const Single DefaultEngineSize = 0f;
        private const string DefaultMileage = "";
        public form1()
        {
            InitializeComponent();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.None;
        }

        // Event handler for when the form loads
        // Loads the regression model and make-model data from file, and prepares
the form panel
        private void form1_Load(object sender, EventArgs e)
        {
            string jsonFilePath = "C:\\Users\\danny\\OneDrive\\Documents\\A
Levels\\Computing\\NEA\\NEA_Solution\\bin\\Debug\\make-model-data.json";
            makeModelData = JsonConvert.DeserializeObject<Dictionary<string,
List<string>>>(File.ReadAllText(jsonFilePath));
            formPanel.BringToFront();
            PopulateMakeDropdown();
            string modelPath = "C:\\Users\\danny\\OneDrive\\Documents\\A
Levels\\Computing\\NEA\\NEA_Solution\\bin\\Debug\\trained-model.zip";
            MLContext mlContext = new MLContext();
            var model = mlContext.Model.Load(modelPath, out var modelSchema);
            predictionFunction = mlContext.Model.CreatePredictionEngine<CarData,
CarPricePrediction>(model);
        }

        // Event handler for when the selected index of the make dropdown is
changed (a new make is selected)
        // Populates the model dropdown with corresponding models
        private void makeDropdown_SelectedIndexChanged(object sender, EventArgs
e)
        {
            modelDropdown.Text = "";
            modelDropdown.Items.Clear();
            errorProvider.SetError(makeDropdown, null);
            errorProvider.SetError(modelDropdown, null);
            if (makeDropdown.SelectedIndex != -1)
            {
                string make = makeDropdown.SelectedItem.ToString();
                List<string> models = makeModelData[make.ToUpper()];
                models.Sort();
                foreach (string model in models)
                {
                    modelDropdown.Items.Add(model);
                }
            }
        }

        // Event handler for when the user leaves the make dropdown
        // Checks that a make has been selected and whether the text entered in
the dropdown is a make
        private void makeDropdown_Leave(object sender, EventArgs e)
        {
            if (makeDropdown.SelectedIndex < 0)
            {
                errorProvider.SetError(makeDropdown, "Please select a make from
the dropdown.");
            }


            if (!makes.Contains(makeDropdown.Text))
            {
                makeDropdown.SelectedIndex = -1;
                modelDropdown.Items.Clear();
                modelDropdown.Text = "";
            }
        }

        // Event handler for when the selected index of the model dropdown
changes
        // Removes the error indication as long as a make has been selected
        private void modelDropdown_SelectedIndexChanged(object sender, EventArgs
e)
        {
            if (makeDropdown.SelectedIndex > -1)
            {
                errorProvider.SetError(modelDropdown, null);
            }
        }

        // Event handler for when the user leaves the model dropdown
        // Checks that a model has been selected
        private void modelDropdown_Leave(object sender, EventArgs e)
        {
            if (modelDropdown.SelectedIndex < 0)
            {
                errorProvider.SetError(modelDropdown, "Please select a model from
the dropdown.");
            }
        }

        // Event handler for when the model dropdown drops down
        // Displays an error message if make has not yet been selected
        private void modelDropdown_DropDown(object sender, EventArgs e)
        {
            if (makeDropdown.SelectedItem == null)
            {
                errorProvider.SetError(modelDropdown, "Make must be selected
before the model");
                MessageBox.Show("The make must be selected before the model.");
            }
        }

        // Event handler for when the text in the mileage textbox changes
        // Validates that the mileage entered is an integer between 0 and 250,000
        private void mileageTextBox_TextChanged(object sender, EventArgs e)
        {
            try
            {
                if (mileageTextBox.Text == null)
                {
                    errorProvider.SetError(mileageTextBox, "Mileage cannot be
null.");
                }
                if (Convert.ToInt32(mileageTextBox.Text) < 0 ||
Convert.ToInt32(mileageTextBox.Text) > 250000)
                    // 250,000 because this is the approximately the largest
value mileage used in the training process, after outlier removal.
                {
                    errorProvider.SetError(mileageTextBox, "Mileage must be a
number less than or equal to 250000");
                }
                else
                {
                    errorProvider.SetError(mileageTextBox, null);
                }
            }
            catch (Exception)
            {
                errorProvider.SetError(mileageTextBox, "Mileage not in valid
format.");
            }
        }

        // Event handler for when the fuel dropdown selected index changes
        // Removes error indication
        private void fuelDropdown_SelectedIndexChanged(object sender, EventArgs
e)
        {
            errorProvider.SetError(fuelDropdown, null);
        }

        // Event handler for when the user leaves the fuel dropdown
        // Checks that a fuel type has been selected
        private void fuelDropdown_Leave(object sender, EventArgs e)
        {
            if (fuelDropdown.SelectedIndex < 0)
            {
                errorProvider.SetError(fuelDropdown, "Please select a fuel from
the dropdown.");
            }
        }

        // Event handler for when the engine size track bar scrolls
        // Adjusts the value of the label to match the position of the pointer on
the spectrum
        private void trackBar_Scroll(object sender, EventArgs e)
        {
            double currentValue = trackBar.Value / 10.0;
            trackBarLabel.Text = currentValue.ToString("0.0");
        }

        // Event handler for when the submit button is clicked
        // Displays a message stating incorrect inputs if there are any present
        private void submitButton_Click(object sender, EventArgs e)
        {
            List<string> invalidInputs = CheckIfAllInputsHaveBeenCompleted();
            if (!(invalidInputs.Count == 0))
            {
                string invalidInputStr = "";
                for (int i=0; i<invalidInputs.Count; i++)
                {
                    invalidInputStr += invalidInputs[i];
                    if (i != invalidInputs.Count - 1)
                    {
                        invalidInputStr += ", ";
                    }
                }
                MessageBox.Show($"Make sure you have correctly entered the
following: {invalidInputStr}");
            }
            else
            {
                PredictCarPrice();
                PrepareOutputPanel();
                outputPanel.BringToFront();
            }
        }
        // Event handler for when the reset button is clicked
        // Resets form
        private void resetButton_Click(object sender, EventArgs e)
        {
            ResetForm();
        }

        // Event handler for when the 'Go Back' button is clicked
        // Displays the form panel
        private void retryButton_Click(object sender, EventArgs e)
        {
            formPanel.BringToFront();
        }

        // Event handler for when the print button is clicked
        // Displays a print preview dialog and an event handler is set for when
the print preview is closed
        private void printButton_Click(object sender, EventArgs e)
        {
            printDocument = new PrintDocument();
            printDocument.PrintPage += new PrintPageEventHandler(PrintNewPage);

            PrintPreviewDialog printPreviewDialog = new PrintPreviewDialog();
            printPreviewDialog.Document = printDocument;
            printPreviewDialog.Show();

            printPreviewDialog.FormClosed += printPreviewDialog_FormClosed;
        }

        // Event handler for when the print preview dialog is closed
        // Prints the document if the user presses 'OK'
        private void printPreviewDialog_FormClosed(object sender,
FormClosedEventArgs e)
        {
            PrintDialog printDialog = new PrintDialog();
            printDialog.Document = printDocument;

            if (printDialog.ShowDialog() == DialogResult.OK)
            {
                printDocument.Print();
            }
        }

        // Captures the visual content of the output screen and draws it on a
print page
        private void PrintNewPage(object sender, PrintPageEventArgs e)
        {
            Bitmap bitmap = new Bitmap(outputPanel.Width, outputPanel.Height);
            outputPanel.DrawToBitmap(bitmap, new Rectangle(0, 0,
outputPanel.Width, outputPanel.Height));
            e.Graphics.DrawImage(bitmap, new PointF(0, 0));
            bitmap.Dispose();
        }

        // Formats makes in correct case, orders them alphabetically and adds
them to the make dropdown
        private void PopulateMakeDropdown()
        {
            makes = new List<string>();
            foreach (var kvp in makeModelData)
            {
                string make = kvp.Key.ToLower();
                char initial = char.ToUpper(make[0]);
                string substring = make.Substring(1);
                if (make != "bmw")
                {
                    for (int i = 0; i < substring.Length; i++)
                    {
                        if (substring[i] == ' ' || substring[i] == '-') // '-' is
for mercedes-benz
                        {
                            substring = substring.Substring(0, i + 1) +
char.ToUpper(substring[i + 1]) + substring.Substring(i + 2);
                        }
                    }
                    make = initial + substring;
                }
                else
                {
                    make = "BMW";
                }
                makes.Add(make);
            }
            makes.Sort();
            foreach (string make in makes)
            {
                makeDropdown.Items.Add(make);
            }
        }

        // Checks for inputs that have errors or are not entered
        // returns: list of invalid inputs
        private List<string> CheckIfAllInputsHaveBeenCompleted()
        {
            List<string> invalidInputs = new List<string>();
            if (makeDropdown.SelectedItem == null ||
!string.IsNullOrEmpty(errorProvider.GetError(makeDropdown)))
            {
                invalidInputs.Add("Make");
            }
            if (modelDropdown.SelectedItem == null ||
!string.IsNullOrEmpty(errorProvider.GetError(modelDropdown)))
            {
                invalidInputs.Add("Model");
            }
            if (!string.IsNullOrEmpty(errorProvider.GetError(yearNumericUpDown)))
            {
                invalidInputs.Add("Year of manufacture");
            }
            if (string.IsNullOrEmpty(mileageTextBox.Text) ||
!string.IsNullOrEmpty(errorProvider.GetError(mileageTextBox)))
            {
                invalidInputs.Add("Mileage");
            }
            if (fuelDropdown.SelectedItem == null ||
!string.IsNullOrEmpty(errorProvider.GetError(fuelDropdown)))
            {
                invalidInputs.Add("Fuel");
            }
            if (!automaticRadioBtn.Checked && !manualRadioBtn.Checked &&
!tiptronicRadioBtn.Checked && !variatorRadioBtn.Checked)
            {
                invalidInputs.Add("Gearbox");
            }
            return invalidInputs;
        }

        // Prepares the ouput panel using the input data from the form panel
        private void PrepareOutputPanel()
        {
            makePlaceholder.Text = makeDropdown.SelectedItem.ToString();
            modelPlaceholder.Text = modelDropdown.SelectedItem.ToString();
            yearPlaceholder.Text = yearNumericUpDown.Value.ToString();
            mileagePlaceholder.Text = mileageTextBox.Text;
            fuelPlaceholder.Text = fuelDropdown.SelectedItem.ToString();
            double engineSize = trackBar.Value / 10.0;
            enginePlaceholder.Text = engineSize.ToString("0.0");
            if (turboEngineCheckbox.Checked)
            {
                enginePlaceholder.Text += " Turbo";
            }
            if (automaticRadioBtn.Checked)
            {
                gearboxPlaceholder.Text = "Automatic";
            }
            else if (manualRadioBtn.Checked)
            {
                gearboxPlaceholder.Text = "Manual";
            }
            else if (tiptronicRadioBtn.Checked)
            {
                gearboxPlaceholder.Text = "Tiptronic";
            }
            else if (variatorRadioBtn.Checked)
            {
                gearboxPlaceholder.Text = "Variator";
            }
            if (leatherInteriorCheckbox.Checked)
            {
                leatherInteriorPlaceholder.Text = "Yes";
            }
            else
            {
                leatherInteriorPlaceholder.Text = "No";
            }
            dateLabel.Text = DateTime.Now.ToString();
        }

        // Estimates and displays the car price using the prediction engine,
derived from the integrated regression model
        private void PredictCarPrice()
        {
            var carSample = new CarData()
            {
                Manufacturer = makeDropdown.SelectedItem.ToString(),
                Model = modelDropdown.SelectedItem.ToString(),
                ProdYear = Convert.ToSingle(yearNumericUpDown.Value),
                MileageInKm = Convert.ToSingle(mileageTextBox.Text) *
MilesToKmMultiplier, // convert from miles to km
                EngineDisplacement = (Single)trackBar.Value / 10,
                IsJeep = CalculateIsJeep(),
                IsDiesel = Convert.ToSingle(fuelDropdown.ToString() == "Diesel"),
                IsTiptronic = Convert.ToSingle(tiptronicRadioBtn.Checked),
                IsLeatherInterior =
Convert.ToSingle(leatherInteriorCheckbox.Checked),
                IsEngineTurbo = Convert.ToSingle(turboEngineCheckbox.Checked)
            };
            try
            {
                var prediction = predictionFunction.Predict(carSample);
                int price = Convert.ToInt32(prediction.Price);
                int roundedPrice = ((price + 50) / 100) * 100;
                string formattedPrice = roundedPrice.ToString("#,0");
                priceLabel.Text = "Price: £" + formattedPrice;
            }
            catch (Exception)
            {
                MessageBox.Show("Error when computing the price.");
                formPanel.BringToFront();
            }
        }
        // Uses the input data from the form to calculate the value for IsJeep, a
predictor required for the regression model
        // By counting the number of equivalent models that have 'Jeep' as their
value in the dataset used to train the model
        private Single CalculateIsJeep()
        {
            string dataPath = "C:\\Users\\danny\\OneDrive\\Documents\\A
Levels\\Computing\\NEA\\NEA_Solution\\bin\\Debug\\train.csv";
            DataTable data = new DataTable();
            using (StreamReader sr = new StreamReader(dataPath))
            {
                string[] fieldNames = sr.ReadLine().Split(',');
                foreach (string fieldName in fieldNames)
                {
                    data.Columns.Add(fieldName);
                }
                while (!sr.EndOfStream)
                {
                    string[] record = sr.ReadLine().Split(',');
                    DataRow row = data.NewRow();
                    for (int i = 0; i < fieldNames.Length; i++)
                    {
                        row[i] = record[i];
                    }
                    data.Rows.Add(row);
                }
            }
            string make = makeDropdown.SelectedItem.ToString().ToUpper();
            string model = modelDropdown.SelectedItem.ToString().ToUpper();
            int jeepCount = 0;
            int nonJeepCount = 0;
            foreach (DataRow row in data.Rows)
            {
                if (row["Manufacturer"].ToString().ToUpper() == make &&
row["Model"].ToString().ToUpper() == model)
                {
                    if (row["Category"].ToString() == "Jeep")
                    {
                        jeepCount++;
                    }
                    else
                    {
                        nonJeepCount++;
                    }
                }
            }
            if (jeepCount > nonJeepCount)
            {
                return 1;
            }
            else
            {
                return 0;
            }
        }

        // Resets the form by returning all inputs to their initial value/state
        private void ResetForm()
        {
            makeDropdown.SelectedIndex = -1;
            modelDropdown.SelectedIndex = -1;
            yearNumericUpDown.Value = DefaultProdYear;
            mileageTextBox.Text = DefaultMileage;
            fuelDropdown.SelectedIndex = -1;
            automaticRadioBtn.Checked = false;
            manualRadioBtn.Checked = false;
            tiptronicRadioBtn.Checked = false;
            variatorRadioBtn.Checked = false;
            trackBar.Value = (int)(DefaultEngineSize * 10);
            trackBarLabel.Text = DefaultEngineSize.ToString("0.0");
            turboEngineCheckbox.Checked = false;
            leatherInteriorCheckbox.Checked = false;
            errorProvider.SetError(mileageTextBox, null);
            errorProvider.SetError(makeDropdown, null);
            errorProvider.SetError(modelDropdown, null);
            errorProvider.SetError(fuelDropdown, null);
        }
    }
}
```

Form1.Designer.cs (PACKAGE-GENERATED)

```
namespace WindowsFormsUI
{
    partial class form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.mileageLabel = new System.Windows.Forms.Label();
            this.mileageTextBox = new System.Windows.Forms.TextBox();
            this.errorProvider = new
System.Windows.Forms.ErrorProvider(this.components);
            this.engineSizeLabel = new System.Windows.Forms.Label();
            this.trackBar = new System.Windows.Forms.TrackBar();
            this.trackBarLabel = new System.Windows.Forms.Label();
            this.makeLabel = new System.Windows.Forms.Label();
            this.makeDropdown = new System.Windows.Forms.ComboBox();
            this.modelLabel = new System.Windows.Forms.Label();
            this.modelDropdown = new System.Windows.Forms.ComboBox();
            this.yearLabel = new System.Windows.Forms.Label();
            this.vehicleInfoBox = new System.Windows.Forms.GroupBox();
            this.yearNumericUpDown = new System.Windows.Forms.NumericUpDown();
            this.gearboxLabel = new System.Windows.Forms.Label();
            this.automaticRadioBtn = new System.Windows.Forms.RadioButton();
            this.manualRadioBtn = new System.Windows.Forms.RadioButton();
            this.tiptronicRadioBtn = new System.Windows.Forms.RadioButton();
            this.variatorRadioBtn = new System.Windows.Forms.RadioButton();
            this.gearboxBox = new System.Windows.Forms.GroupBox();
            this.leatherInteriorCheckbox = new System.Windows.Forms.CheckBox();
            this.fuelDropdown = new System.Windows.Forms.ComboBox();
            this.fuelLabel = new System.Windows.Forms.Label();
            this.turboEngineCheckbox = new System.Windows.Forms.CheckBox();
            this.engineSizeBox = new System.Windows.Forms.GroupBox();
            this.technicalDetailsBox = new System.Windows.Forms.GroupBox();
            this.subtitleLabel = new System.Windows.Forms.Label();
            this.titleLabel = new System.Windows.Forms.Label();
            this.submitButton = new System.Windows.Forms.Button();
            this.formPanel = new System.Windows.Forms.Panel();
            this.resetButton = new System.Windows.Forms.Button();
            this.outputPanel = new System.Windows.Forms.Panel();
            this.dateLabel = new System.Windows.Forms.Label();
            this.printButton = new System.Windows.Forms.Button();
            this.retryButton = new System.Windows.Forms.Button();
            this.priceLabel = new System.Windows.Forms.Label();
            this.outerBox2 = new System.Windows.Forms.GroupBox();
            this.technicalDetailsBox2 = new System.Windows.Forms.GroupBox();
            this.engineSizeLabel2 = new System.Windows.Forms.Label();
            this.leatherInteriorPlaceholder = new System.Windows.Forms.Label();
            this.enginePlaceholder = new System.Windows.Forms.Label();
            this.gearboxPlaceholder = new System.Windows.Forms.Label();
            this.gearboxLabel2 = new System.Windows.Forms.Label();
            this.leatherInteriorLabel2 = new System.Windows.Forms.Label();
            this.fuelLabel2 = new System.Windows.Forms.Label();
            this.fuelPlaceholder = new System.Windows.Forms.Label();
            this.vehicleInfoBox2 = new System.Windows.Forms.GroupBox();
            this.makeLabel2 = new System.Windows.Forms.Label();
            this.mileagePlaceholder = new System.Windows.Forms.Label();
            this.modelLabel2 = new System.Windows.Forms.Label();
            this.yearPlaceholder = new System.Windows.Forms.Label();
            this.yearLabel2 = new System.Windows.Forms.Label();
            this.modelPlaceholder = new System.Windows.Forms.Label();
            this.mileageLabel2 = new System.Windows.Forms.Label();
            this.makePlaceholder = new System.Windows.Forms.Label();

((System.ComponentModel.ISupportInitialize)(this.errorProvider)).BeginInit();

((System.ComponentModel.ISupportInitialize)(this.trackBar)).BeginInit();
            this.vehicleInfoBox.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.yearNumericUpDown)).BeginInit();
            this.gearboxBox.SuspendLayout();
            this.engineSizeBox.SuspendLayout();
            this.technicalDetailsBox.SuspendLayout();
            this.formPanel.SuspendLayout();
            this.outputPanel.SuspendLayout();
            this.outerBox2.SuspendLayout();
            this.technicalDetailsBox2.SuspendLayout();
            this.vehicleInfoBox2.SuspendLayout();
            this.SuspendLayout();
            //
            // mileageLabel
            //
            this.mileageLabel.AutoSize = true;
            this.mileageLabel.Location = new System.Drawing.Point(23, 177);
            this.mileageLabel.Name = "mileageLabel";
            this.mileageLabel.Size = new System.Drawing.Size(56, 16);
            this.mileageLabel.TabIndex = 6;
            this.mileageLabel.Text = "Mileage";
            //
            // mileageTextBox
            //
            this.mileageTextBox.Location = new System.Drawing.Point(170, 177);
            this.mileageTextBox.MaxLength = 6;
            this.mileageTextBox.Name = "mileageTextBox";
            this.mileageTextBox.Size = new System.Drawing.Size(187, 22);
            this.mileageTextBox.TabIndex = 8;
            this.mileageTextBox.TextChanged += new
System.EventHandler(this.mileageTextBox_TextChanged);
            //
            // errorProvider
            //
            this.errorProvider.BlinkStyle =
System.Windows.Forms.ErrorBlinkStyle.NeverBlink;
            this.errorProvider.ContainerControl = this;
            //
            // engineSizeLabel
            //
            this.engineSizeLabel.AutoSize = true;
            this.engineSizeLabel.Location = new System.Drawing.Point(16, 11);
            this.engineSizeLabel.Name = "engineSizeLabel";
            this.engineSizeLabel.Size = new System.Drawing.Size(128, 16);
            this.engineSizeLabel.TabIndex = 9;
            this.engineSizeLabel.Text = "Engine size (in litres)";
            //
            // trackBar
            //
            this.trackBar.LargeChange = 10;
            this.trackBar.Location = new System.Drawing.Point(27, 114);
            this.trackBar.Maximum = 60;
            this.trackBar.Name = "trackBar";
            this.trackBar.Size = new System.Drawing.Size(144, 56);
            this.trackBar.TabIndex = 10;
            this.trackBar.Scroll += new
System.EventHandler(this.trackBar_Scroll);
            //
            // trackBarLabel
            //
            this.trackBarLabel.AutoSize = true;
            this.trackBarLabel.Location = new System.Drawing.Point(59, 33);
            this.trackBarLabel.Name = "trackBarLabel";
            this.trackBarLabel.Size = new System.Drawing.Size(24, 16);
            this.trackBarLabel.TabIndex = 11;
            this.trackBarLabel.Text = "0.0";
            //
            // makeLabel
            //
            this.makeLabel.AutoSize = true;
            this.makeLabel.Location = new System.Drawing.Point(23, 23);
            this.makeLabel.Name = "makeLabel";
            this.makeLabel.Size = new System.Drawing.Size(41, 16);
            this.makeLabel.TabIndex = 0;
            this.makeLabel.Text = "Make";
            //
            // makeDropdown
            //
            this.makeDropdown.FormattingEnabled = true;
            this.makeDropdown.Location = new System.Drawing.Point(170, 20);
            this.makeDropdown.Name = "makeDropdown";
            this.makeDropdown.Size = new System.Drawing.Size(187, 24);
            this.makeDropdown.TabIndex = 1;
            this.makeDropdown.SelectedIndexChanged += new
System.EventHandler(this.makeDropdown_SelectedIndexChanged);
            this.makeDropdown.Leave += new
System.EventHandler(this.makeDropdown_Leave);
            //
            // modelLabel
            //
            this.modelLabel.AutoSize = true;
            this.modelLabel.Location = new System.Drawing.Point(23, 74);
            this.modelLabel.Name = "modelLabel";
            this.modelLabel.Size = new System.Drawing.Size(45, 16);
            this.modelLabel.TabIndex = 2;
            this.modelLabel.Text = "Model";
            //
            // modelDropdown
            //
            this.modelDropdown.FormattingEnabled = true;
            this.modelDropdown.Location = new System.Drawing.Point(170, 71);
            this.modelDropdown.Name = "modelDropdown";
            this.modelDropdown.Size = new System.Drawing.Size(187, 24);
            this.modelDropdown.TabIndex = 3;
            this.modelDropdown.DropDown += new
System.EventHandler(this.modelDropdown_DropDown);
            this.modelDropdown.SelectedIndexChanged += new
System.EventHandler(this.modelDropdown_SelectedIndexChanged);
            this.modelDropdown.Leave += new
System.EventHandler(this.modelDropdown_Leave);
            //
            // yearLabel
            //
            this.yearLabel.AutoSize = true;
            this.yearLabel.Location = new System.Drawing.Point(23, 126);
            this.yearLabel.Name = "yearLabel";
            this.yearLabel.Size = new System.Drawing.Size(126, 16);
            this.yearLabel.TabIndex = 4;
            this.yearLabel.Text = "Year of Manufacture";
            //
            // vehicleInfoBox
            //
            this.vehicleInfoBox.Controls.Add(this.yearNumericUpDown);
            this.vehicleInfoBox.Controls.Add(this.mileageTextBox);
            this.vehicleInfoBox.Controls.Add(this.mileageLabel);
            this.vehicleInfoBox.Controls.Add(this.yearLabel);
            this.vehicleInfoBox.Controls.Add(this.modelDropdown);
            this.vehicleInfoBox.Controls.Add(this.modelLabel);
            this.vehicleInfoBox.Controls.Add(this.makeDropdown);
            this.vehicleInfoBox.Controls.Add(this.makeLabel);
            this.vehicleInfoBox.Location = new System.Drawing.Point(23, 124);
            this.vehicleInfoBox.Name = "vehicleInfoBox";
            this.vehicleInfoBox.Size = new System.Drawing.Size(382, 208);
            this.vehicleInfoBox.TabIndex = 17;
            this.vehicleInfoBox.TabStop = false;
            this.vehicleInfoBox.Text = "Vehicle info";
            //
            // yearNumericUpDown
            //
```

```
this.yearNumericUpDown.Location = new System.Drawing.Point(170, 124);
this.yearNumericUpDown.Maximum = new decimal(new int[] {
2020,
0,
0,
0});
this.yearNumericUpDown.Minimum = new decimal(new int[] {
1990,
0,
0,
0});
this.yearNumericUpDown.Name = "yearNumericUpDown";
this.yearNumericUpDown.Size = new System.Drawing.Size(187, 22);
this.yearNumericUpDown.TabIndex = 29;
this.yearNumericUpDown.Value = new decimal(new int[] {
2020,
0,
0,
0});
//
// gearboxLabel
//
this.gearboxLabel.AutoSize = true;
this.gearboxLabel.Location = new System.Drawing.Point(7, 15);
this.gearboxLabel.Name = "gearboxLabel";
this.gearboxLabel.Size = new System.Drawing.Size(59, 16);
this.gearboxLabel.TabIndex = 12;
this.gearboxLabel.Text = "Gearbox";
//
// automaticRadioBtn
//
this.automaticRadioBtn.AutoSize = true;
this.automaticRadioBtn.Location = new System.Drawing.Point(10, 36);
this.automaticRadioBtn.Name = "automaticRadioBtn";
this.automaticRadioBtn.Size = new System.Drawing.Size(87, 20);
this.automaticRadioBtn.TabIndex = 13;
this.automaticRadioBtn.TabStop = true;
this.automaticRadioBtn.Text = "Automatic";
this.automaticRadioBtn.UseVisualStyleBackColor = true;
//
// manualRadioBtn
//
this.manualRadioBtn.AutoSize = true;
this.manualRadioBtn.Location = new System.Drawing.Point(10, 62);
this.manualRadioBtn.Name = "manualRadioBtn";
this.manualRadioBtn.Size = new System.Drawing.Size(72, 20);
this.manualRadioBtn.TabIndex = 14;
this.manualRadioBtn.TabStop = true;
this.manualRadioBtn.Text = "Manual";
this.manualRadioBtn.UseVisualStyleBackColor = true;
//
// tiptronicRadioBtn
//
this.tiptronicRadioBtn.AutoSize = true;
this.tiptronicRadioBtn.Location = new System.Drawing.Point(10, 88);
this.tiptronicRadioBtn.Name = "tiptronicRadioBtn";
this.tiptronicRadioBtn.Size = new System.Drawing.Size(80, 20);
this.tiptronicRadioBtn.TabIndex = 15;
this.tiptronicRadioBtn.TabStop = true;
this.tiptronicRadioBtn.Text = "Tiptronic";
this.tiptronicRadioBtn.UseVisualStyleBackColor = true;
//
// variatorRadioBtn
```

```
            //
            this.variatorRadioBtn.AutoSize = true;
            this.variatorRadioBtn.Location = new System.Drawing.Point(10, 114);
            this.variatorRadioBtn.Name = "variatorRadioBtn";
            this.variatorRadioBtn.Size = new System.Drawing.Size(75, 20);
            this.variatorRadioBtn.TabIndex = 16;
            this.variatorRadioBtn.TabStop = true;
            this.variatorRadioBtn.Text = "Variator";
            this.variatorRadioBtn.UseVisualStyleBackColor = true;
            //
            // gearboxBox
            //
            this.gearboxBox.Controls.Add(this.variatorRadioBtn);
            this.gearboxBox.Controls.Add(this.tiptronicRadioBtn);
            this.gearboxBox.Controls.Add(this.manualRadioBtn);
            this.gearboxBox.Controls.Add(this.automaticRadioBtn);
            this.gearboxBox.Controls.Add(this.gearboxLabel);
            this.gearboxBox.Location = new System.Drawing.Point(230, 26);
            this.gearboxBox.Name = "gearboxBox";
            this.gearboxBox.Size = new System.Drawing.Size(106, 147);
            this.gearboxBox.TabIndex = 18;
            this.gearboxBox.TabStop = false;
            //
            // leatherInteriorCheckbox
            //
            this.leatherInteriorCheckbox.AutoSize = true;
            this.leatherInteriorCheckbox.Location = new System.Drawing.Point(230,
179);
            this.leatherInteriorCheckbox.Name = "leatherInteriorCheckbox";
            this.leatherInteriorCheckbox.Size = new System.Drawing.Size(117, 20);
            this.leatherInteriorCheckbox.TabIndex = 20;
            this.leatherInteriorCheckbox.Text = "Leather interior";
            this.leatherInteriorCheckbox.UseVisualStyleBackColor = true;
            //
            // fuelDropdown
            //
            this.fuelDropdown.FormattingEnabled = true;
            this.fuelDropdown.Items.AddRange(new object[] {
            "CNG",
            "Diesel",
            "Hybrid",
            "Hydrogen",
            "LPG",
            "Petrol",
            "Plug-in Hybrid"});
            this.fuelDropdown.Location = new System.Drawing.Point(44, 36);
            this.fuelDropdown.Name = "fuelDropdown";
            this.fuelDropdown.Size = new System.Drawing.Size(105, 24);
            this.fuelDropdown.TabIndex = 22;
            this.fuelDropdown.SelectedIndexChanged += new
System.EventHandler(this.fuelDropdown_SelectedIndexChanged);
            this.fuelDropdown.Leave += new
System.EventHandler(this.fuelDropdown_Leave);
            //
            // fuelLabel
            //
            this.fuelLabel.AutoSize = true;
            this.fuelLabel.Location = new System.Drawing.Point(80, 17);
            this.fuelLabel.Name = "fuelLabel";
            this.fuelLabel.Size = new System.Drawing.Size(33, 16);
            this.fuelLabel.TabIndex = 21;
            this.fuelLabel.Text = "Fuel";            //
            // turboEngineCheckbox
            //
            this.turboEngineCheckbox.AutoSize = true;
            this.turboEngineCheckbox.Location = new System.Drawing.Point(44,
179);
            this.turboEngineCheckbox.Name = "turboEngineCheckbox";
            this.turboEngineCheckbox.Size = new System.Drawing.Size(109, 20);
            this.turboEngineCheckbox.TabIndex = 23;
            this.turboEngineCheckbox.Text = "Turbo engine";
            this.turboEngineCheckbox.UseVisualStyleBackColor = true;
            //
            // engineSizeBox
            //
            this.engineSizeBox.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.None;
            this.engineSizeBox.Controls.Add(this.trackBarLabel);
            this.engineSizeBox.Controls.Add(this.engineSizeLabel);
            this.engineSizeBox.Location = new System.Drawing.Point(21, 62);
            this.engineSizeBox.Name = "engineSizeBox";
            this.engineSizeBox.Size = new System.Drawing.Size(156, 111);
            this.engineSizeBox.TabIndex = 24;
            this.engineSizeBox.TabStop = false;
            //
            // technicalDetailsBox
            //
            this.technicalDetailsBox.Controls.Add(this.trackBar);
            this.technicalDetailsBox.Controls.Add(this.turboEngineCheckbox);
            this.technicalDetailsBox.Controls.Add(this.fuelDropdown);
            this.technicalDetailsBox.Controls.Add(this.fuelLabel);
            this.technicalDetailsBox.Controls.Add(this.leatherInteriorCheckbox);
            this.technicalDetailsBox.Controls.Add(this.gearboxBox);
            this.technicalDetailsBox.Controls.Add(this.engineSizeBox);
            this.technicalDetailsBox.ForeColor =
System.Drawing.SystemColors.ControlText;
            this.technicalDetailsBox.Location = new System.Drawing.Point(427,
124);
            this.technicalDetailsBox.Name = "technicalDetailsBox";
            this.technicalDetailsBox.Size = new System.Drawing.Size(382, 208);
            this.technicalDetailsBox.TabIndex = 25;
            this.technicalDetailsBox.TabStop = false;
            this.technicalDetailsBox.Text = "Technical details";
            //
            // subtitleLabel
            //
            this.subtitleLabel.AutoSize = true;
            this.subtitleLabel.Font = new System.Drawing.Font("Microsoft Sans
Serif", 12F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.subtitleLabel.Location = new System.Drawing.Point(230, 71);
            this.subtitleLabel.Name = "subtitleLabel";
            this.subtitleLabel.Size = new System.Drawing.Size(422, 25);
            this.subtitleLabel.TabIndex = 26;
            this.subtitleLabel.Text = "Enter the following details about your car
below:";
            //
            // titleLabel
            //
            this.titleLabel.AutoSize = true;
            this.titleLabel.Font = new System.Drawing.Font("Microsoft Sans
Serif", 16.2F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.titleLabel.Location = new System.Drawing.Point(298, 11);
            this.titleLabel.Name = "titleLabel";
            this.titleLabel.Size = new System.Drawing.Size(277, 32);
            this.titleLabel.TabIndex = 27;
            this.titleLabel.Text = "Car Price Estimator";
            //
            // submitButton
            //
            this.submitButton.BackColor =
System.Drawing.SystemColors.ButtonHighlight;
            this.submitButton.Location = new System.Drawing.Point(304, 359);
            this.submitButton.Name = "submitButton";
            this.submitButton.Size = new System.Drawing.Size(227, 32);
            this.submitButton.TabIndex = 28;
            this.submitButton.Text = "Get Price Estimate";
            this.submitButton.UseVisualStyleBackColor = false;
            this.submitButton.Click += new
System.EventHandler(this.submitButton_Click);
            //
            // formPanel
            //
            this.formPanel.Controls.Add(this.resetButton);
            this.formPanel.Controls.Add(this.submitButton);
            this.formPanel.Controls.Add(this.titleLabel);
            this.formPanel.Controls.Add(this.subtitleLabel);
            this.formPanel.Controls.Add(this.technicalDetailsBox);
            this.formPanel.Controls.Add(this.vehicleInfoBox);
            this.formPanel.Location = new System.Drawing.Point(87, 20);
            this.formPanel.Name = "formPanel";
            this.formPanel.Size = new System.Drawing.Size(833, 427);
            this.formPanel.TabIndex = 29;
            //
            // resetButton
            //
            this.resetButton.Location = new System.Drawing.Point(733, 87);
            this.resetButton.Name = "resetButton";
            this.resetButton.Size = new System.Drawing.Size(76, 26);
            this.resetButton.TabIndex = 29;
            this.resetButton.Text = "Reset";
            this.resetButton.UseVisualStyleBackColor = true;
            this.resetButton.Click += new
System.EventHandler(this.resetButton_Click);
            //
            // outputPanel
            //
            this.outputPanel.Controls.Add(this.dateLabel);
            this.outputPanel.Controls.Add(this.printButton);
            this.outputPanel.Controls.Add(this.retryButton);
            this.outputPanel.Controls.Add(this.priceLabel);
            this.outputPanel.Controls.Add(this.outerBox2);
            this.outputPanel.Location = new System.Drawing.Point(87, 20);
            this.outputPanel.Name = "outputPanel";
            this.outputPanel.Size = new System.Drawing.Size(833, 427);
            this.outputPanel.TabIndex = 30;
            //
            // dateLabel
            //
            this.dateLabel.AutoSize = true;
            this.dateLabel.Font = new System.Drawing.Font("Microsoft Sans Serif",
7.8F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(0)));
            this.dateLabel.Location = new System.Drawing.Point(350, 128);
            this.dateLabel.Name = "dateLabel";
            this.dateLabel.Size = new System.Drawing.Size(92, 16);
            this.dateLabel.TabIndex = 28;
            this.dateLabel.Text = "dateTimeNow";
            //
            // printButton
            //
            this.printButton.Location = new System.Drawing.Point(557, 114);
            this.printButton.Name = "printButton";
            this.printButton.Size = new System.Drawing.Size(59, 30);
            this.printButton.TabIndex = 27;
            this.printButton.Text = "Print";
            this.printButton.UseVisualStyleBackColor = true;
            this.printButton.Click += new
System.EventHandler(this.printButton_Click);
            //
            // retryButton
            //
            this.retryButton.Location = new System.Drawing.Point(346, 329);
            this.retryButton.Name = "retryButton";
            this.retryButton.Size = new System.Drawing.Size(120, 32);
            this.retryButton.TabIndex = 25;
            this.retryButton.Text = "Go Back";
            this.retryButton.UseVisualStyleBackColor = true;
            this.retryButton.Click += new
System.EventHandler(this.retryButton_Click);
            //
            // priceLabel
            //
            this.priceLabel.AutoSize = true;
            this.priceLabel.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D;
            this.priceLabel.Font = new System.Drawing.Font("Microsoft Sans
Serif", 25.8F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.priceLabel.Location = new System.Drawing.Point(279, 71);
            this.priceLabel.Name = "priceLabel";
            this.priceLabel.Size = new System.Drawing.Size(170, 53);
            this.priceLabel.TabIndex = 1;
            this.priceLabel.Text = "Price: £";
            //
            // outerBox2
            //
            this.outerBox2.Controls.Add(this.technicalDetailsBox2);
            this.outerBox2.Controls.Add(this.vehicleInfoBox2);
            this.outerBox2.Location = new System.Drawing.Point(193, 147);
            this.outerBox2.Name = "outerBox2";
            this.outerBox2.Size = new System.Drawing.Size(423, 160);
            this.outerBox2.TabIndex = 26;
            this.outerBox2.TabStop = false;            //
            // technicalDetailsBox2
            //
            this.technicalDetailsBox2.BackColor =
System.Drawing.SystemColors.Control;
            this.technicalDetailsBox2.Controls.Add(this.engineSizeLabel2);

this.technicalDetailsBox2.Controls.Add(this.leatherInteriorPlaceholder);
            this.technicalDetailsBox2.Controls.Add(this.enginePlaceholder);
            this.technicalDetailsBox2.Controls.Add(this.gearboxPlaceholder);
            this.technicalDetailsBox2.Controls.Add(this.gearboxLabel2);
            this.technicalDetailsBox2.Controls.Add(this.leatherInteriorLabel2);
            this.technicalDetailsBox2.Controls.Add(this.fuelLabel2);
            this.technicalDetailsBox2.Controls.Add(this.fuelPlaceholder);
            this.technicalDetailsBox2.Location = new System.Drawing.Point(214,
17);
            this.technicalDetailsBox2.Name = "technicalDetailsBox2";
            this.technicalDetailsBox2.Size = new System.Drawing.Size(201, 137);
            this.technicalDetailsBox2.TabIndex = 27;
            this.technicalDetailsBox2.TabStop = false;
            this.technicalDetailsBox2.Text = "Technical details";
            //
            // engineSizeLabel2
            //
            this.engineSizeLabel2.AutoSize = true;
            this.engineSizeLabel2.ForeColor =
System.Drawing.SystemColors.GrayText;
            this.engineSizeLabel2.Location = new System.Drawing.Point(6, 51);
            this.engineSizeLabel2.Name = "engineSizeLabel2";
            this.engineSizeLabel2.Size = new System.Drawing.Size(79, 16);
            this.engineSizeLabel2.TabIndex = 23;
            this.engineSizeLabel2.Text = "Engine size:";
            //
            // leatherInteriorPlaceholder
            //
            this.leatherInteriorPlaceholder.AutoSize = true;
            this.leatherInteriorPlaceholder.Location = new
System.Drawing.Point(111, 102);
            this.leatherInteriorPlaceholder.Name = "leatherInteriorPlaceholder";
            this.leatherInteriorPlaceholder.Size = new System.Drawing.Size(48,
16);
            this.leatherInteriorPlaceholder.TabIndex = 22;
            this.leatherInteriorPlaceholder.Text = "yes/no";
            //
            // enginePlaceholder
            //
            this.enginePlaceholder.AutoSize = true;
            this.enginePlaceholder.Location = new System.Drawing.Point(111, 51);
            this.enginePlaceholder.Name = "enginePlaceholder";
            this.enginePlaceholder.Size = new System.Drawing.Size(78, 16);
            this.enginePlaceholder.TabIndex = 24;
            this.enginePlaceholder.Text = "engineHere";
            //
            // gearboxPlaceholder
            //
            this.gearboxPlaceholder.AutoSize = true;
            this.gearboxPlaceholder.Location = new System.Drawing.Point(111, 77);
            this.gearboxPlaceholder.Name = "gearboxPlaceholder";
            this.gearboxPlaceholder.Size = new System.Drawing.Size(87, 16);
            this.gearboxPlaceholder.TabIndex = 20;
            this.gearboxPlaceholder.Text = "gearboxHere";
            //
            // gearboxLabel2
            //
            this.gearboxLabel2.AutoSize = true;
            this.gearboxLabel2.ForeColor = System.Drawing.SystemColors.GrayText;
            this.gearboxLabel2.Location = new System.Drawing.Point(6, 77);
            this.gearboxLabel2.Name = "gearboxLabel2";
            this.gearboxLabel2.Size = new System.Drawing.Size(62, 16);
            this.gearboxLabel2.TabIndex = 16;
            this.gearboxLabel2.Text = "Gearbox:";
            //
            // leatherInteriorLabel2
            //
            this.leatherInteriorLabel2.AutoSize = true;
            this.leatherInteriorLabel2.ForeColor =
System.Drawing.SystemColors.GrayText;
            this.leatherInteriorLabel2.Location = new System.Drawing.Point(7,
102);
            this.leatherInteriorLabel2.Name = "leatherInteriorLabel2";
            this.leatherInteriorLabel2.Size = new System.Drawing.Size(98, 16);
            this.leatherInteriorLabel2.TabIndex = 18;
            this.leatherInteriorLabel2.Text = "Leather interior:";
            // 
            // fuelLabel2
            // 
            this.fuelLabel2.AutoSize = true;
            this.fuelLabel2.ForeColor = System.Drawing.SystemColors.GrayText;
            this.fuelLabel2.Location = new System.Drawing.Point(7, 24);
            this.fuelLabel2.Name = "fuelLabel2";
            this.fuelLabel2.Size = new System.Drawing.Size(36, 16);
            this.fuelLabel2.TabIndex = 15;
            this.fuelLabel2.Text = "Fuel:";
            // 
            // fuelPlaceholder
            // 
            this.fuelPlaceholder.AutoSize = true;
            this.fuelPlaceholder.Location = new System.Drawing.Point(111, 24);
            this.fuelPlaceholder.Name = "fuelPlaceholder";
            this.fuelPlaceholder.Size = new System.Drawing.Size(58, 16);
            this.fuelPlaceholder.TabIndex = 19;
            this.fuelPlaceholder.Text = "fuelHere";
            // 
            // vehicleInfoBox2
            // 
            this.vehicleInfoBox2.Controls.Add(this.makeLabel2);
            this.vehicleInfoBox2.Controls.Add(this.mileagePlaceholder);
            this.vehicleInfoBox2.Controls.Add(this.modelLabel2);
            this.vehicleInfoBox2.Controls.Add(this.yearPlaceholder);
            this.vehicleInfoBox2.Controls.Add(this.yearLabel2);
            this.vehicleInfoBox2.Controls.Add(this.modelPlaceholder);
            this.vehicleInfoBox2.Controls.Add(this.mileageLabel2);
            this.vehicleInfoBox2.Controls.Add(this.makePlaceholder);
            this.vehicleInfoBox2.Location = new System.Drawing.Point(7, 17);
            this.vehicleInfoBox2.Name = "vehicleInfoBox2";
            this.vehicleInfoBox2.Size = new System.Drawing.Size(201, 137);
            this.vehicleInfoBox2.TabIndex = 27;
            this.vehicleInfoBox2.TabStop = false;
            this.vehicleInfoBox2.Text = "Vehicle info";
            // 
            // makeLabel2
            // 
            this.makeLabel2.AutoSize = true;
            this.makeLabel2.ForeColor = System.Drawing.SystemColors.GrayText;
            this.makeLabel2.Location = new System.Drawing.Point(6, 24);
            this.makeLabel2.Name = "makeLabel2";
            this.makeLabel2.Size = new System.Drawing.Size(44, 16);
            this.makeLabel2.TabIndex = 7;
            this.makeLabel2.Text = "Make:";
            // 
            // mileagePlaceholder
            // 
            this.mileagePlaceholder.AutoSize = true;
            this.mileagePlaceholder.Location = new System.Drawing.Point(78, 102);
            this.mileagePlaceholder.Name = "mileagePlaceholder";
            this.mileagePlaceholder.Size = new System.Drawing.Size(86, 16);
            this.mileagePlaceholder.TabIndex = 14;
            this.mileagePlaceholder.Text = "mileageHere";
            // 
            // modelLabel2
            // 
            this.modelLabel2.AutoSize = true;
            this.modelLabel2.ForeColor = System.Drawing.SystemColors.GrayText;
            this.modelLabel2.Location = new System.Drawing.Point(6, 51);
            this.modelLabel2.Name = "modelLabel2";
            this.modelLabel2.Size = new System.Drawing.Size(48, 16);
            this.modelLabel2.TabIndex = 8;
            this.modelLabel2.Text = "Model:";
            //
            // yearPlaceholder
            //
            this.yearPlaceholder.AutoSize = true;
            this.yearPlaceholder.Location = new System.Drawing.Point(78, 77);
            this.yearPlaceholder.Name = "yearPlaceholder";
            this.yearPlaceholder.Size = new System.Drawing.Size(64, 16);
            this.yearPlaceholder.TabIndex = 13;
            this.yearPlaceholder.Text = "yearHere";
            //
            // yearLabel2
            //
            this.yearLabel2.AutoSize = true;
            this.yearLabel2.ForeColor = System.Drawing.SystemColors.GrayText;
            this.yearLabel2.Location = new System.Drawing.Point(6, 77);
            this.yearLabel2.Name = "yearLabel2";
            this.yearLabel2.Size = new System.Drawing.Size(39, 16);
            this.yearLabel2.TabIndex = 9;
            this.yearLabel2.Text = "Year:";            //
            // modelPlaceholder
            //
            this.modelPlaceholder.AutoSize = true;
            this.modelPlaceholder.Location = new System.Drawing.Point(78, 51);
            this.modelPlaceholder.Name = "modelPlaceholder";
            this.modelPlaceholder.Size = new System.Drawing.Size(75, 16);
            this.modelPlaceholder.TabIndex = 12;
            this.modelPlaceholder.Text = "modelHere";
            //
            // mileageLabel2
            //
            this.mileageLabel2.AutoSize = true;
            this.mileageLabel2.ForeColor = System.Drawing.SystemColors.GrayText;
            this.mileageLabel2.Location = new System.Drawing.Point(6, 102);
            this.mileageLabel2.Name = "mileageLabel2";
            this.mileageLabel2.Size = new System.Drawing.Size(59, 16);
            this.mileageLabel2.TabIndex = 10;
            this.mileageLabel2.Text = "Mileage:";
            //
            // makePlaceholder
            //
            this.makePlaceholder.AutoSize = true;
            this.makePlaceholder.Location = new System.Drawing.Point(78, 24);
            this.makePlaceholder.Name = "makePlaceholder";
            this.makePlaceholder.Size = new System.Drawing.Size(71, 16);
            this.makePlaceholder.TabIndex = 11;
            this.makePlaceholder.Text = "makeHere";
            //
            // form1
            //
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.None;
            this.ClientSize = new System.Drawing.Size(1030, 487);
            this.Controls.Add(this.formPanel);
            this.Controls.Add(this.outputPanel);
            this.Name = "form1";
            this.Text = "Form1";
            this.Load += new System.EventHandler(this.form1_Load);

((System.ComponentModel.ISupportInitialize)(this.errorProvider)).EndInit();
```

```
((System.ComponentModel.ISupportInitialize)(this.trackBar)).EndInit();
            this.vehicleInfoBox.ResumeLayout(false);
            this.vehicleInfoBox.PerformLayout();

((System.ComponentModel.ISupportInitialize)(this.yearNumericUpDown)).EndInit();
            this.gearboxBox.ResumeLayout(false);
            this.gearboxBox.PerformLayout();
            this.engineSizeBox.ResumeLayout(false);
            this.engineSizeBox.PerformLayout();
            this.technicalDetailsBox.ResumeLayout(false);
            this.technicalDetailsBox.PerformLayout();
            this.formPanel.ResumeLayout(false);
            this.formPanel.PerformLayout();
            this.outputPanel.ResumeLayout(false);
            this.outputPanel.PerformLayout();
            this.outerBox2.ResumeLayout(false);
            this.technicalDetailsBox2.ResumeLayout(false);
            this.technicalDetailsBox2.PerformLayout();
            this.vehicleInfoBox2.ResumeLayout(false);
            this.vehicleInfoBox2.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion
        private System.Windows.Forms.Label mileageLabel;
        private System.Windows.Forms.TextBox mileageTextBox;
        private System.Windows.Forms.ErrorProvider errorProvider;
        private System.Windows.Forms.Label engineSizeLabel;
        private System.Windows.Forms.TrackBar trackBar;
        private System.Windows.Forms.Label trackBarLabel;
        private System.Windows.Forms.Label yearLabel;
        private System.Windows.Forms.ComboBox modelDropdown;
        private System.Windows.Forms.Label modelLabel;
        private System.Windows.Forms.ComboBox makeDropdown;
        private System.Windows.Forms.Label makeLabel;
        private System.Windows.Forms.GroupBox vehicleInfoBox;
        private System.Windows.Forms.RadioButton variatorRadioBtn;
        private System.Windows.Forms.RadioButton tiptronicRadioBtn;
        private System.Windows.Forms.RadioButton manualRadioBtn;
        private System.Windows.Forms.RadioButton automaticRadioBtn;
        private System.Windows.Forms.Label gearboxLabel;
        private System.Windows.Forms.GroupBox gearboxBox;
        private System.Windows.Forms.ComboBox fuelDropdown;
        private System.Windows.Forms.Label fuelLabel;
        private System.Windows.Forms.CheckBox leatherInteriorCheckbox;
        private System.Windows.Forms.CheckBox turboEngineCheckbox;
        private System.Windows.Forms.GroupBox engineSizeBox;
        private System.Windows.Forms.Label subtitleLabel;
        private System.Windows.Forms.GroupBox technicalDetailsBox;
        private System.Windows.Forms.Button submitButton;
        private System.Windows.Forms.Label titleLabel;
        private System.Windows.Forms.Panel outputPanel;
        private System.Windows.Forms.Panel formPanel;
        private System.Windows.Forms.NumericUpDown yearNumericUpDown;
        private System.Windows.Forms.Label priceLabel;
        private System.Windows.Forms.Label leatherInteriorPlaceholder;
        private System.Windows.Forms.Label gearboxPlaceholder;
        private System.Windows.Forms.Label fuelPlaceholder;
        private System.Windows.Forms.Label leatherInteriorLabel2;
        private System.Windows.Forms.Label gearboxLabel2;
        private System.Windows.Forms.Label fuelLabel2;
        private System.Windows.Forms.Label mileagePlaceholder;
        private System.Windows.Forms.Label yearPlaceholder;
        private System.Windows.Forms.Label modelPlaceholder;
        private System.Windows.Forms.Label makePlaceholder;
        private System.Windows.Forms.Label mileageLabel2;
        private System.Windows.Forms.Label yearLabel2;
        private System.Windows.Forms.Label modelLabel2;
        private System.Windows.Forms.Label makeLabel2;
        private System.Windows.Forms.Button retryButton;
        private System.Windows.Forms.Label enginePlaceholder;
        private System.Windows.Forms.Label engineSizeLabel2;
        private System.Windows.Forms.GroupBox outerBox2;
        private System.Windows.Forms.GroupBox vehicleInfoBox2;
        private System.Windows.Forms.GroupBox technicalDetailsBox2;
        private System.Windows.Forms.Button resetButton;
        private System.Windows.Forms.Button printButton;
        private System.Windows.Forms.Label dateLabel;
    }
}
```

Testing

The testing of my project will be split into the following main sections: the component testing of the regression analysis phase of the project, and the final system testing which includes testing the user interface after integrating the regression model into the system.

For the component testing of the regression analysis process, I have shown below some carefully selected and representative samples of the individual tests on the main methods of the program, ordered in a test table below. For the tests regarding statistical calculations, I used online calculators and excel to obtain expected outcomes. Most of the tests were carried out using the Visual Studio debugger. I have chosen to only display evidence for testing where necessary - for data visualisation; all other tests gave relatively simple outputs on the console.

Following these tests, I have demonstrated the flow of the process of training and testing different regression models through labelled screenshots from the console below.

Lastly, I have presented the final system testing through a test table of a similar format to the regression analysis component testing. I have also provided screenshots below the test table which display the test results after each test has been completed.

Component testing of regression analysis

Test table

Test no.	Purpose of testing	Section of program	Test data (pg. 150)	Expected outcome	Actual outcome	Test passed (Y/N)
1	To check that duplicate rows are detected and removed correctly from the datatable.	Method: RemoveDuplicateRecords() Class: DataTableModifier	Dataset 1	The 3 duplicate rows are removed.	As expected.	Y
2	To check that missing values are correctly detected in the datable.	Method: CheckForMissingValues() Class: DataPreprocessor	Dataset 1	The null value is detected and its indexed location is written to the console.	Null value not detected.	N
3	To check that	Method:	Dataset	Leather	As	Y

	the leather interior column is correctly encoded in numerical form.	MakeLeatherInteriorColumnNumerical() Class: DataTableModifier	1	interior column is numerical and consists of 1s and 0s.	expected.	
4	To check that the engine column is correctly split into two numerical columns for engine displacement and turbo engine.	Method: SplitEngineVolumeColumn() Class: DataTableModifier	Dataset 1	Two new numerical columns: engine displacement and a binary column for turbo engine.	As expected.	Y
5	To check that the mileage column is correctly converted to a numerical column.	Method: MakeMileageColumnNumerical() Class: DataTableModifier	Dataset 1	Mileage column is numerical with no tailing ‘km’.	As expected.	Y
6	To check that the wheel column is correctly encoded in numerical form.	Method: MakeWheelColumnNumerical() Class: DataTableModifier	Dataset 1	Wheel column is numerical consisting of 1s and 0s.	As expected.	Y
7	To check that the category column is correctly divided into binary columns.	Method: MakeCategoryColumnNumerical() Class: DataTableModifier	Dataset 1	Category column converted to many binary columns corresponding to each category.	As expected.	Y
8	To check that the fuel type column is correctly divided into binary columns.	Method: MakeFuelTypeColumnNumerical() Class: DataTableMo	Dataset 1	Fuel type column converted to many binary columns corresponding to each	As expected.	Y

		difier		fuel type.		
9	To check that the gearbox column is correctly divided into binary columns.	Method: MakeGearBoxTypeColumnNumerical() Class: DataTableModifier	Dataset 1	Gearbox column converted to many binary columns corresponding to each gearbox type.	As expected.	Y
10	To check that the drive wheels column is correctly divided into binary columns.	Method: MakeDriveWheelsColumnNumerical() Class: DataTableModifier	Dataset 1	Drive wheels column converted to many binary columns corresponding to each drive type.	As expected.	Y
11	To check that the colour column is correctly divided into binary columns.	Method: MakeColourColumnNumerical() Class: DataTableModifier	Dataset 1	Colour column converted to many binary columns corresponding to each colour.	As expected.	Y
12	To check that an array of values in a column is correctly populated and returned.	Method: GetColumnValuesAsDoubleArray() Class: DataUtilities	Dataset 1, Column name: Price	Returned array contains all the values in the price column.	As expected.	Y
13	To check that data is correctly displayed on the console.	Method: DisplayData() Class: DataUtilities	Dataset 1, Row numbers: [1, 3, 6]	Rows 1, 3 and 6 are written to the console in a readable format.	As expected.	Y
14	To check that the column names are correctly displayed on the console.	Method: DisplayColumnNames() Class: DataUtilities	Dataset 1	All column names in dataset 1 are written to the console.	As expected.	Y
15	To check that	Method:	Array 1	Returned	As	Y

	the mean of an array of numbers is correctly calculated.	CalculateMean() Class: Statistics		mean value is 52.75.	expected.	
17	To check that the mode of an array of numbers is correctly calculated.	Method: CalculateMode() Class: Statistics	Array 1	Returned mode value is 10.	As expected.	Y
18	To check that the median of an array of numbers is correctly calculated.	Method: CalculateMedian() Class: Statistics	Array 1	Returned median value is 52.5.	As expected.	Y
19	To check that the population variance of an array of numbers is correctly calculated.	Method: CalculateVariance() Class: Statistics	Array 1	Returned variance value is 808.6875.	As expected.	Y
20	To check that the standard deviation of an array of numbers is correctly calculated.	Method: CalculateStdDev() Class: Statistics	Array 1	Returned standard deviation value is 28.437431.	As expected.	Y
21	To check that the covariance of two arrays of numbers is correctly calculated.	Method: CalculateCoVariance() Class: Statistics	Array 1, Array 2	Returned covariance value is 1714.27632.	As expected.	Y
22	To check that the z-score of a given value is correctly calculated.	Method: CalculateZScore() Class: Statistics	Value: 55, Mean: 52.75, Standard deviation: 28.437431	Returned z-score value is 0.07912.	As expected.	Y
23	To check that the median	Method: CalculateMe	Array 1	Returned median	As expected.	Y

	absolute deviation of an array of numbers is correctly calculated.	dianAbsDev() Class: Statistics		absolute deviation value is 25.		
24	To check that the modified z-score of a given value is correctly calculated.	Method: CalculateModifiedZScore() Class: Statistics	Value: 55, Median: 52.5, Median absolute deviation: 25	Returned modified z-score value is 0.06745.	As expected.	Y
25	To check that a given percentile for an array of numbers in ascending order is correctly calculated.	Method: CalculatePercentile() Class: Statistics	Array 1 (sorted), percentile: 0.25	Returned value at the 25th percentile is 27.5.	Returned value is 28.75 but this is fine as it is close to the expected value but there is just a minor difference in the implementation which is acceptable.	Y
26	To check that the Pearson correlation coefficient for two arrays of continuous values is correctly calculated.	Method: CalculatePearsonCorrelationCoefficient() Class: Statistics	Array 1, Array 2	Returned Pearson correlation coefficient value is 0.9799.	As expected.	Y
27	To check that the Point Biserial correlation coefficient for an array of dichotomous x values and an	Method: CalculatePointBiserialCorrelationCoefficient() Class:	Array 3, Array 2	Returned Point biserial correlation coefficient value is 0.7579.	Returned value is 0.7776 but this is fine as it is close to the	Y

	array of continuous y values is correctly calculated.	Statistics			expected value and there is just a minor difference in the implementation which is acceptable.	
28	To check that the merge sort algorithm works correctly.	Method: MergeSort() Class: Statistics	Array 1	Sorts array 1 in ascending order.	As expected.	Y
29	To check that outliers are correctly located using z-scores.	Method: LocateOutliersWithZScores() Class: OutlierIdentifier	Dataset 1, Column name: Price, Threshold: 2	Returns row number 4 only as the rows containing outliers.	As expected.	Y
30	To check that outliers are correctly located using modified z-scores.	Method: LocateOutliersWithModifiedZScores() Class: OutlierIdentifier	Dataset 1, Column name: Price, Threshold: 2	Returns row number 4 only as the rows containing outliers.	As expected.	Y
31	To check that outliers are correctly located and displayed using the interquartile range method.	Method: LocateOutliersWithIQR() Class: OutlierIdentifier	Dataset 1, Column name: Price	Displays price of row 4 only.	As expected.	Y
32	To check that a column is scaled correctly using min-max scaling.	Method: ScaleNumericalColumnMinMax() Class: DataPreprocessor	Dataset 1, Column name: Airbags	A new column created containing cylinders min-max scaled values, with	As expected.	Y

				first row 0.333.		
33	To check that a column is scaled correctly using standardisation.	Method: ScaleNumericalColumnStandardisation() Class: DataPreprocessor	Dataset 1, Column name: Airbags	A new column created containing cylinders standardised values, with first row -0.3536.	As expected.	Y
34	To check that a scatter plot is correctly displayed to visualise the correlation between the price and continuous predictors.	Method: CreateScatterPlot() Class: ScatterPlotForm	Dataset 1, Column name: Prod. year	Displays a labelled scatter plot with the data points correctly plotted.	As expected. (See pg. 151)	Y
35	To check that box plots are correctly displayed to visualise the distribution of predictors.	Method: CreateBoxPlot() Class: BoxPlotForm	Dataset 1, Column names: Cylinders, Airbags	Displays two labelled and correctly plotted box plots.	As expected. (See pg. 151)	Y
36	To check that continuous columns are correctly displayed in order of their Pearson correlation coefficients.	Method: RankNumericalColumnsByPearsonCoefficients() Class: FeatureSelector	Dataset 1, Column names: Prod. year, Cylinders, Airbags	Displays columns in descending order of the magnitude of their correlation coefficients.	As expected.	Y
37	To check that binary columns are correctly displayed in order of their point biserial correlation coefficients.	Method: RankBinaryColumnsByPointBiserialCoefficients() Class: FeatureSelector	Dataset 1, Column names: Is automatic, Is leather interior, Is engine turbo	Displays columns in descending order of the magnitude of their correlation coefficients.	As expected.	Y

38	To check that the percentage of 1s in binary columns are correctly calculated and displayed.	Method: DisplayBinaryColumnRatios() Class: FeatureSelector	Dataset 1, Column names: Is automatic, Is leather interior, Is engine turbo	Each binary column name is displayed with its corresponding percentage of 1s.	As expected.	Y
39	To check that the mean absolute error is correctly calculated.	Method: CalculateMAE() Class: ModelValidator	Y-Predictions: Array 1, Y-Actual: Array 4	Returned mean absolute error value is 6.5.	As expected.	Y
40	To check that the root mean square error is correctly calculated.	Method: CalculateRMSE() Class: ModelValidator	Y-Predictions: Array 1, Y-Actual: Array 4	Returned root mean square error value is 8.5147.	As expected.	Y
41	To check that the coefficient of determination (R^2) is correctly calculated.	Method: CalculateRSquared() Class: ModelValidator	Y-Predictions: Array 1, Y-Actual: Array 4	Returned R^2 value is 0.9286.	As expected.	Y
42	To check that Adjusted R^2 is correctly calculated.	Method: CalculateAdjustedRSquared() Class: ModelValidator	Y-Predictions: Array 1, Y-Actual: Array 4, Number of predictors: 3	Returned Adjusted R^2 value is 0.9152.	As expected.	Y
43	To check that data is correctly split into training, validation and holdout data.	Method: SplitData() Class: DataUtilities	Dataset 1, Training ratio: 0.6, Validation ratio:	Returned array should contain 3 datatables with random records:	Array contains training data with 5 rows, but validation	N

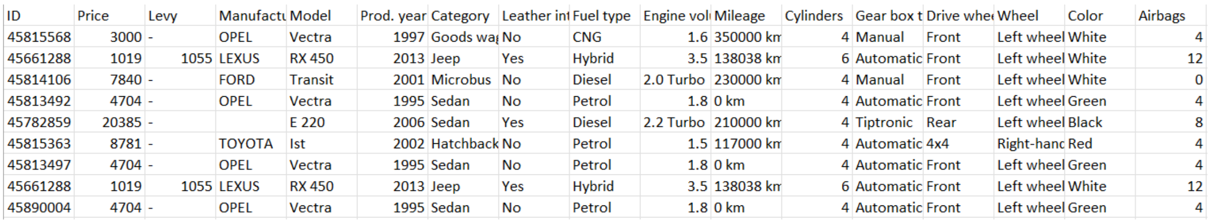
			0.3	training data with 5 rows, validation data with 3 rows, holdout data with 1 row.	and holdout data both have 2 rows.	
44	To check that data is split correctly into four subsets (for cross validation).	Method: DivideDataInto4() Class: DataUtilities	Dataset 1	Returned array should contain four datatables: three should have two rows, and one should have three rows.	As expected.	Y
45	To check that a regression model is correctly validated using k-fold cross validation.	Method: CrossValidate() Class: RegressionModel	Dataset 1	Four folds should be used for training and one fold for validation, and the validation fold is switched each of the five iterations.	As expected.	Y
46	To check that the modified bubble sort algorithm sorts the smallest k values in descending order.	Method: BubbleSortDescending() Class: KNNRegressionModel	Array 1 (but altered so that each value is an indexed distance with the array 1 values as distances), k: 3	Returned array should have 3 smallest distances at the last 3 indexes of the array.	As expected.	Y

Re-testing

Test no.	Corrective actions	Test data	Expected outcome	Actual outcome	Test passed (Y/N)
2	The boolean condition for checking for the existence of missing values changed from ‘row.ItemArray[i] == null’ to ‘string.IsNullOrEmpty(row.ItemArray[i].ToString())’.	Dataset 1	The null value is detected and its indexed location is written to the console.	As expected.	Y
43	An additional piece of logic was added when assigning the row counts of each subset of data. This was to include the method Math.Round() which proved to provide a better split compared to before where a double was simply casted to an integer, meaning the number was truncated and not rounded.	Dataset 1, Training ratio: 0.6, Validation ratio: 0.3	Returned array should contain 3 datatables with random records: training data with 5 rows, validation data with 3 rows, holdout data with 1 row.	As expected.	Y

Test data

- Dataset 1



ID	Price	Levy	Manufactu	Model	Prod. year	Category	Leather int	Fuel type	Engine vol	Mileage	Cylinders	Gear box t	Drive whee	Wheel	Color	Airbags
45815568	3000	-	OPEL	Vectra	1997	Goods wag	No	CNG	1.6	350000 km	4	Manual	Front	Left wheel	White	4
45661288	1019	1055	LEXUS	RX 450	2013	Jeep	Yes	Hybrid	3.5	138038 km	6	Automatic	Front	Left wheel	White	12
45814106	7840	-	FORD	Transit	2001	Microbus	No	Diesel	2.0 Turbo	230000 km	4	Manual	Front	Left wheel	White	0
45813492	4704	-	OPEL	Vectra	1995	Sedan	No	Petrol	1.8	0 km	4	Automatic	Front	Left wheel	Green	4
45782859	20385	-		E 220	2006	Sedan	Yes	Diesel	2.2 Turbo	210000 km	4	Tiptronic	Rear	Left wheel	Black	8
45815363	8781	-	TOYOTA	Ist	2002	Hatchback	No	Petrol	1.5	117000 km	4	Automatic	4x4	Right-hand	Red	4
45813497	4704	-	OPEL	Vectra	1995	Sedan	No	Petrol	1.8	0 km	4	Automatic	Front	Left wheel	Green	4
45661288	1019	1055	LEXUS	RX 450	2013	Jeep	Yes	Hybrid	3.5	138038 km	6	Automatic	Front	Left wheel	White	12
45890004	4704	-	OPEL	Vectra	1995	Sedan	No	Petrol	1.8	0 km	4	Automatic	Front	Left wheel	Green	4

- Array 1

{ 55, 20, 75, 10, 45, 85, 35, 65, 95, 25, 80, 30, 15, 70, 90, 60, 40, 50, 100, 10 }

- Array 2

{ 100, 50, 140, 30, 80, 160, 65, 120, 180, 45, 170, 65, 30, 140, 185, 110, 60, 80, 230, 25 }

- Array 3

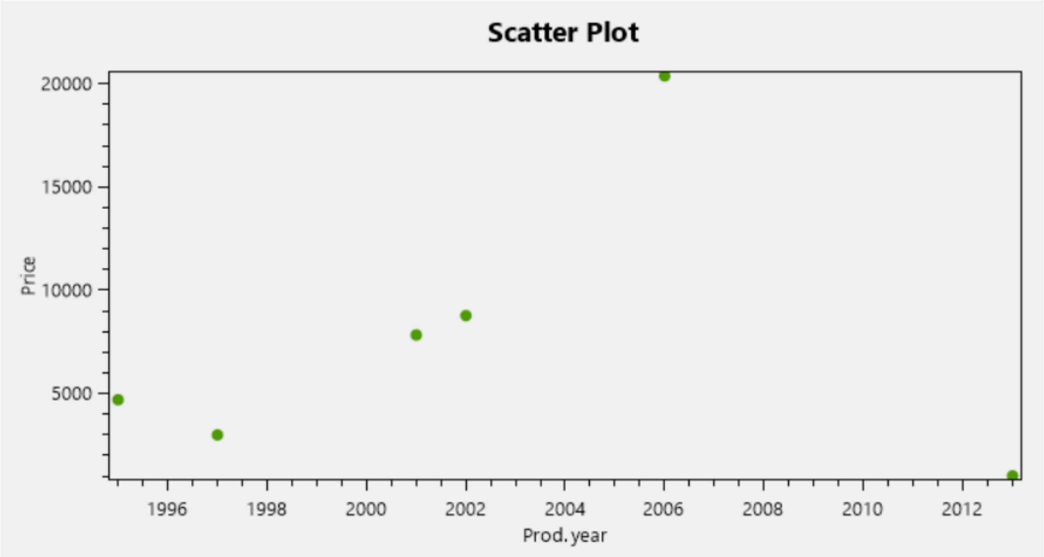
{ 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0 }

- Array 4

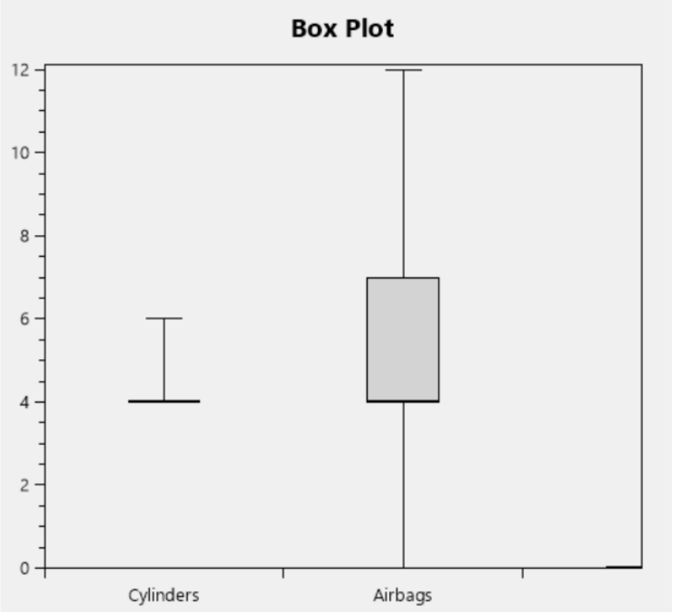
{ 50, 0, 90, 5, 45, 85, 40, 60, 80, 20, 90, 25, 15, 70, 90, 70, 35, 40, 110, 15 }

Test evidence

- Test 33



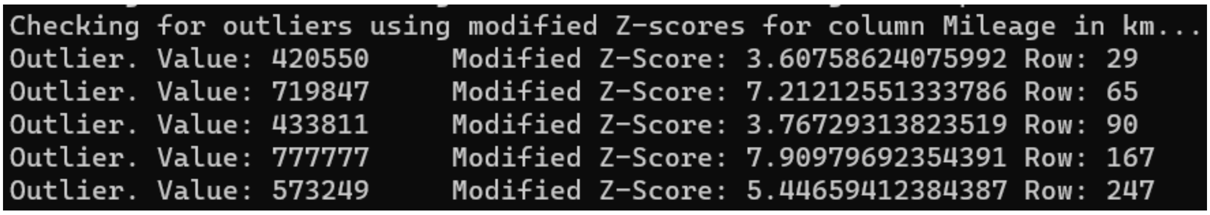
- Test 34



Screenshots illustrating the regression analysis process carried out

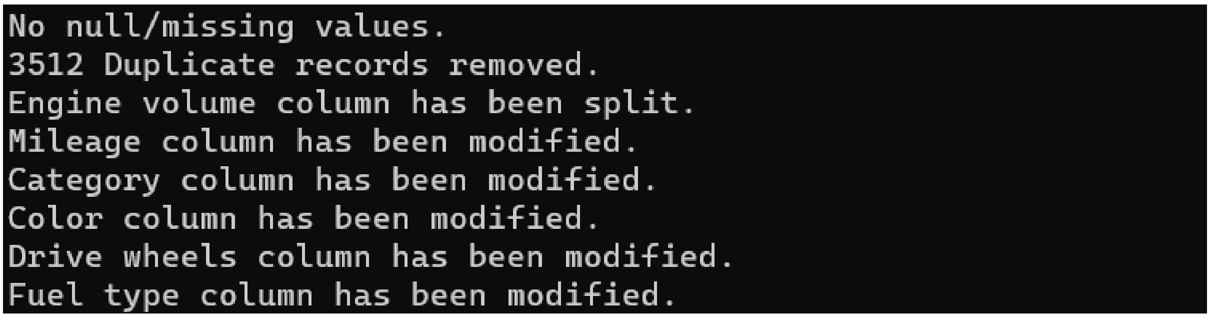
The first stage of the development of each regression model was the data preprocessing, where I checked for duplicate records and missing values, encoded data numerically, located outliers and chose whether to impute or delete appropriately.

Locating outliers:



```
Checking for outliers using modified Z-scores for column Mileage in km...
Outlier. Value: 420550     Modified Z-Score: 3.60758624075992 Row: 29
Outlier. Value: 719847     Modified Z-Score: 7.21212551333786 Row: 65
Outlier. Value: 433811     Modified Z-Score: 3.76729313823519 Row: 90
Outlier. Value: 777777     Modified Z-Score: 7.90979692354391 Row: 167
Outlier. Value: 573249     Modified Z-Score: 5.44659412384387 Row: 247
```

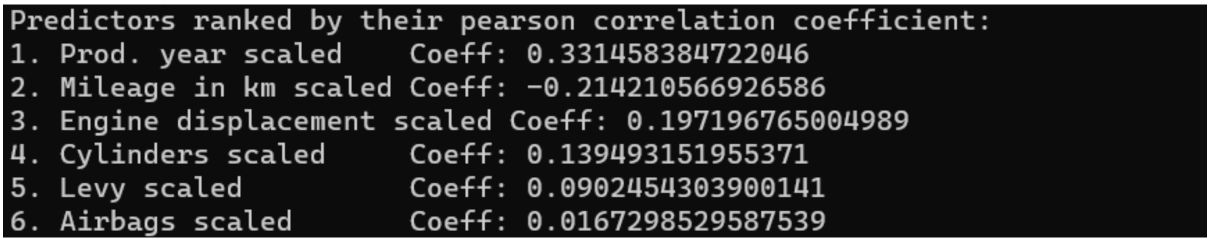
Data checks and modifications:



```
No null/missing values.
3512 Duplicate records removed.
Engine volume column has been split.
Mileage column has been modified.
Category column has been modified.
Color column has been modified.
Drive wheels column has been modified.
Fuel type column has been modified.
```

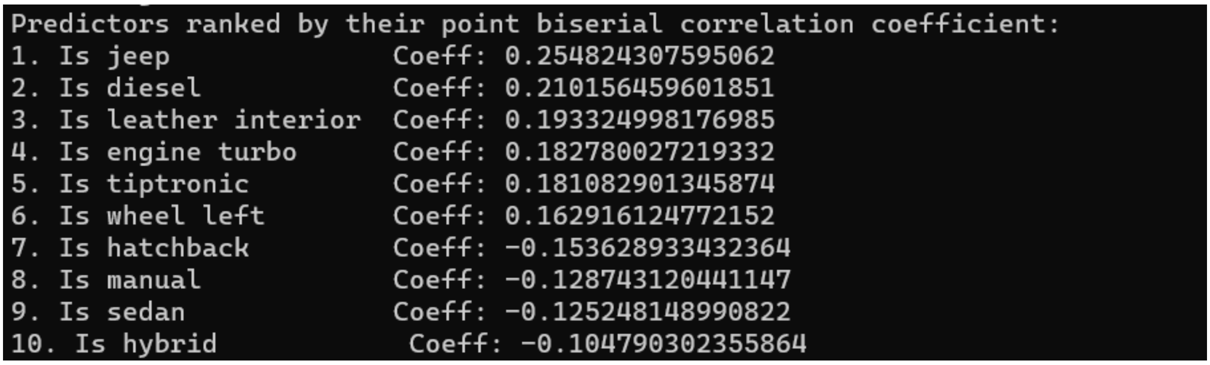
The next stage was feature selection and involved data visualisation and rankings of features based on their correlation coefficients.

Continuous predictor ranking:



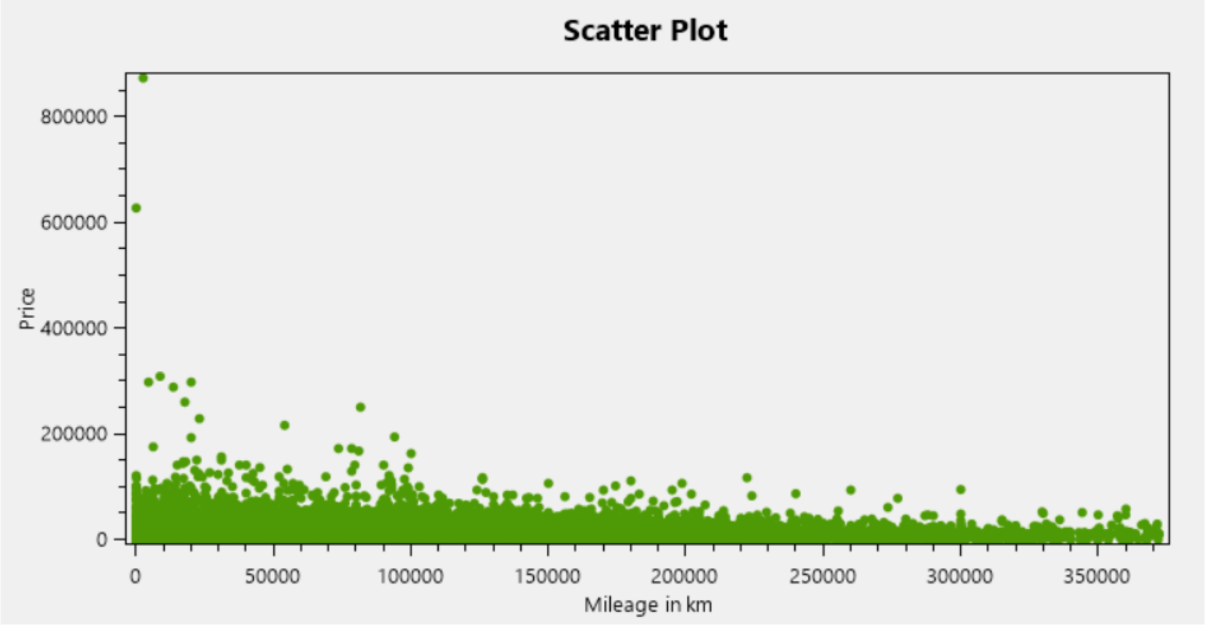
```
Predictors ranked by their pearson correlation coefficient:
1. Prod. year scaled    Coeff: 0.331458384722046
2. Mileage in km scaled Coeff: -0.214210566926586
3. Engine displacement scaled Coeff: 0.197196765004989
4. Cylinders scaled     Coeff: 0.139493151955371
5. Levy scaled          Coeff: 0.0902454303900141
6. Airbags scaled       Coeff: 0.0167298529587539
```

Binary predictor ranking:

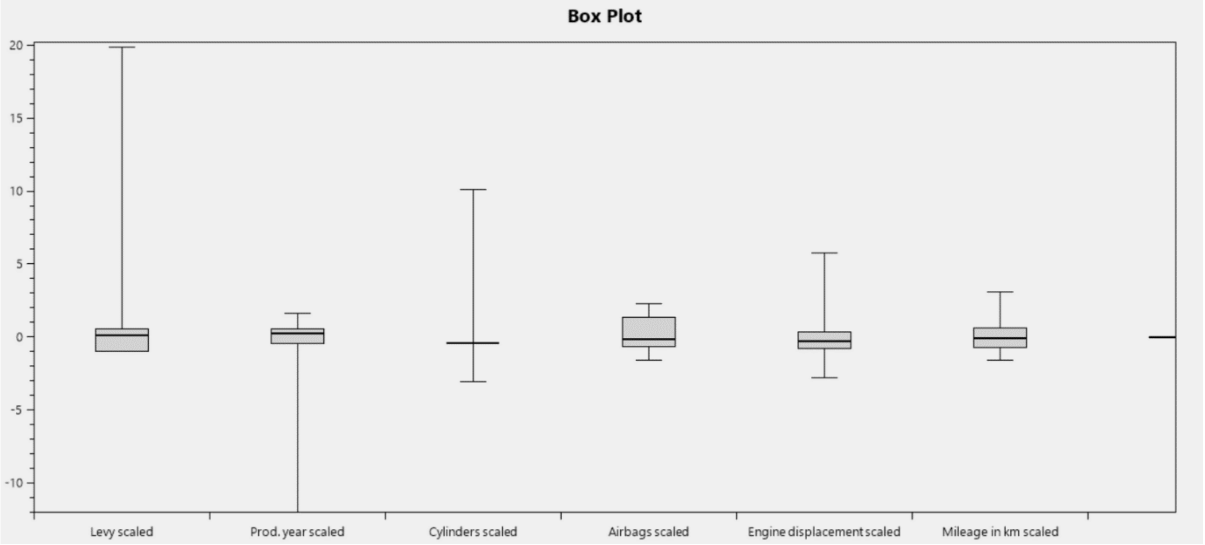


```
Predictors ranked by their point biserial correlation coefficient:
1. Is jeep              Coeff: 0.254824307595062
2. Is diesel            Coeff: 0.210156459601851
3. Is leather interior  Coeff: 0.193324998176985
4. Is engine turbo      Coeff: 0.182780027219332
5. Is tiptronic         Coeff: 0.181082901345874
6. Is wheel left        Coeff: 0.162916124772152
7. Is hatchback         Coeff: -0.153628933432364
8. Is manual            Coeff: -0.128743120441147
9. Is sedan             Coeff: -0.125248148990822
10. Is hybrid            Coeff: -0.104790302355864
```

Scatter plot:

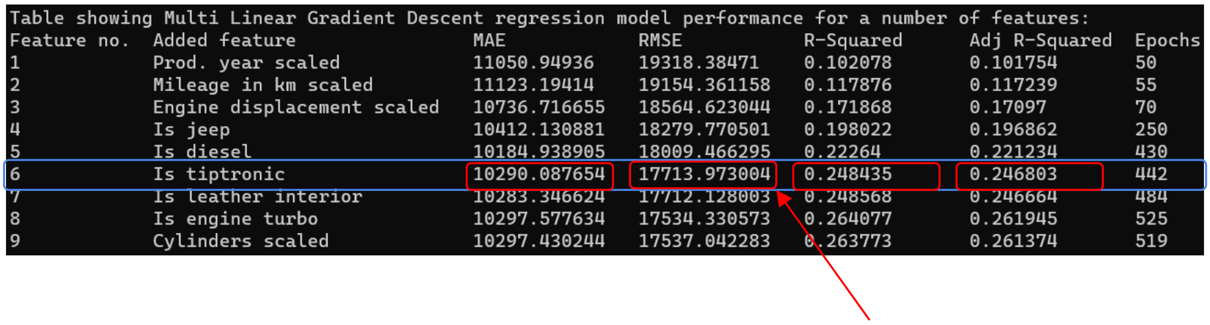


Box plots:



The next part of the regression analysis process was the training of the regression model. After training, each model was validated in order to view how well the model performs; the model is validated many times with different features in order to determine the optimal combination of features. I have shown below the result of this process for some of the regression models.

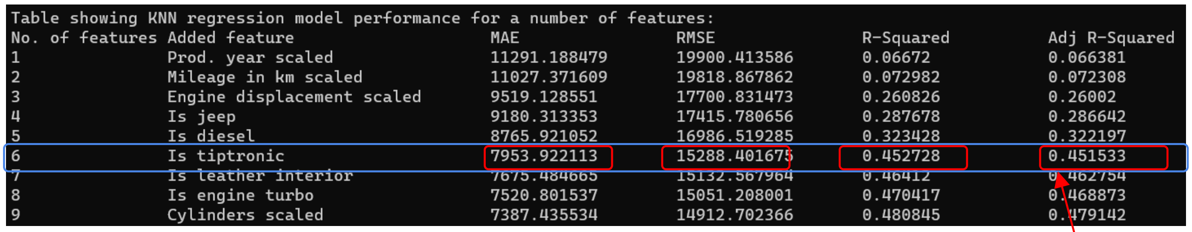
Multi linear regression model using gradient descent:



```
Table showing Multi Linear Gradient Descent regression model performance for a number of features:
Feature no.   Added feature                 MAE             RMSE            R-Squared       Adj R-Squared   Epochs
1             Prod. year scaled             11050.94936     19318.38471     0.102078        0.101754        50
2             Mileage in km scaled          11123.19414     19154.361158    0.117876        0.117239        55
3             Engine displacement scaled    10736.716655    18564.623044    0.171868        0.17097         70
4             Is jeep                       10412.130881    18279.770501    0.198022        0.196862        250
5             Is diesel                     10184.938905    18009.466295    0.22264         0.221234        430
6             Is tiptronic                  10290.087654    17713.973004    0.248435        0.246803        442
7             Is leather interior           10283.346624    17712.128003    0.248568        0.246664        484
8             Is engine turbo               10297.577634    17534.330573    0.264077        0.261945        525
9             Cylinders scaled              10297.430244    17537.042283    0.263773        0.261374        519
```

Reduction in error decreased so ‘Is Tiptronic’ is the last useful feature as features after cause only small reductions in error, meaning they will likely hinder the model and lead to overfitting.

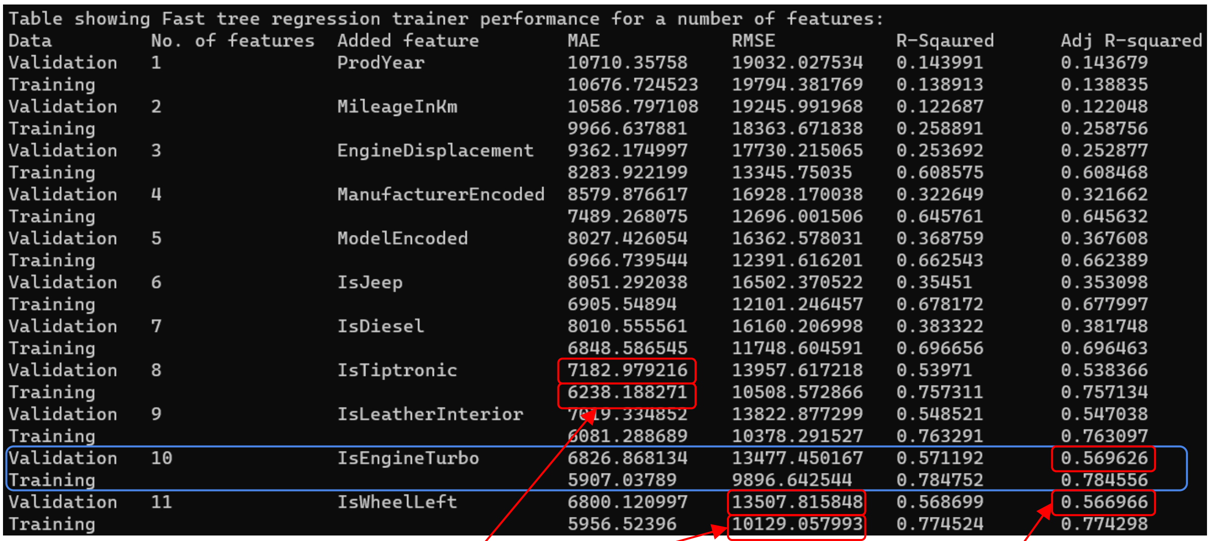
KNN regression model:



```
Table showing KNN regression model performance for a number of features:
No. of features Added feature                 MAE             RMSE            R-Squared       Adj R-Squared
1               Prod. year scaled             11291.188479    19900.413586    0.06672         0.066381
2               Mileage in km scaled          11027.371609    19818.867862    0.072982        0.072308
3               Engine displacement scaled    9519.128551     17700.831473    0.260826        0.26002
4               Is jeep                       9180.313353     17415.780656    0.287678        0.286642
5               Is diesel                     8765.921052     16986.519285    0.323428        0.322197
6               Is tiptronic                  7953.922113     15288.401675    0.452728        0.451533
7               Is leather interior           7675.484665     15132.567964    0.46412         0.462754
8               Is engine turbo               7520.801537     15051.208001    0.470417        0.468873
9               Cylinders scaled              7387.435534     14912.702366    0.480845        0.479142
```

Clearly shown by the Adjusted R-Squared, ‘Is tiptronic’ is the last feature to have a large positive impact on the model.

Fast tree regression model:



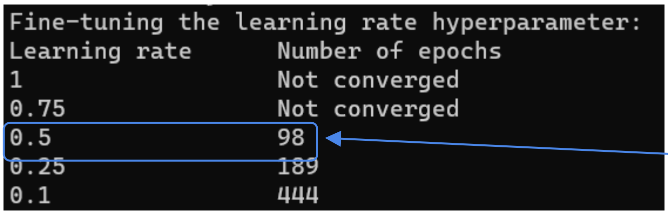
```
Table showing Fast tree regression trainer performance for a number of features:
Data          No. of features   Added feature       MAE             RMSE            R-Sqaured       Adj R-squared
Validation    1                 ProdYear            10710.35758     19032.027534    0.143991        0.143679
Training                                            10676.724523    19794.381769    0.138913        0.138835
Validation    2                 MileageInKm         10586.797108    19245.991968    0.122687        0.122048
Training                                            9966.637881     18363.671838    0.258891        0.258756
Validation    3                 EngineDisplacement  9362.174997     17730.215065    0.253692        0.252877
Training                                            8283.922199     13345.75035     0.608575        0.608468
Validation    4                 ManufacturerEncoded 8579.876617     16928.170038    0.322649        0.321662
Training                                            7489.268075     12696.001506    0.645761        0.645632
Validation    5                 ModelEncoded        8027.426054     16362.578031    0.368759        0.367608
Training                                            6966.739544     12391.616201    0.662543        0.662389
Validation    6                 IsJeep              8051.292038     16502.370522    0.35451         0.353098
Training                                            6905.54894      12101.246457    0.678172        0.677997
Validation    7                 IsDiesel            8010.555561     16160.206998    0.383322        0.381748
Training                                            6848.586545     11748.604591    0.696656        0.696463
Validation    8                 IsTiptronic         7182.979216     13957.617218    0.53971         0.538366
Training                                            6238.188271     10508.572866    0.757311        0.757134
Validation    9                 IsLeatherInterior   7019.334852     13822.877299    0.548521        0.547038
Training                                            6081.288689     10378.291527    0.763291        0.763097
Validation    10                IsEngineTurbo       6826.868134     13477.450167    0.571192        0.569626
Training                                            5907.03789      9896.642544     0.784752        0.784556
Validation    11                IsWheelLeft         6800.120997     13507.815848    0.568699        0.566966
Training                                            5956.52396      10129.057993    0.774524        0.774298
```

The model clearly performs better on the training data than the validation data which indicates there is some overfitting (which is to be

Clearly, adding features after ‘IsEngineTurbo’ is detrimental to the model’s performance.

After determining the optimal feature combination for the model, the model can be fine-tuned by adjusting significant hyperparameters. I have shown two examples of this below.

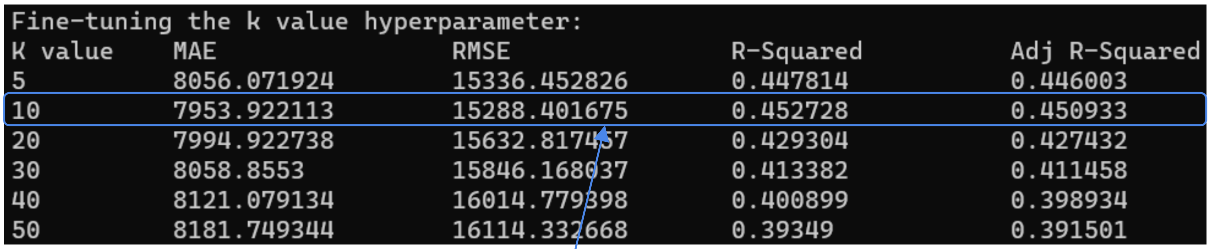
Gradient descent algorithm: (Multi linear regression model)



```
Fine-tuning the learning rate hyperparameter:
Learning rate       Number of epochs
1                   Not converged
0.75                Not converged
0.5                 98
0.25                189
0.1                 444
```

A learning rate of 0.5 is optimal because it leads to convergence in only 98 epochs, meaning the

KNN regression model:



```
Fine-tuning the k value hyperparameter:
K value     MAE                 RMSE                R-Squared           Adj R-Squared
5           8056.071924         15336.452826        0.447814            0.446003
10          7953.922113         15288.401675        0.452728            0.450933
20          7994.922738         15632.817457        0.429304            0.427432
30          8058.8553           15846.168037        0.413382            0.411458
40          8121.079134         16014.779398        0.400899            0.398934
50          8181.749344         16114.332668        0.39349             0.391501
```

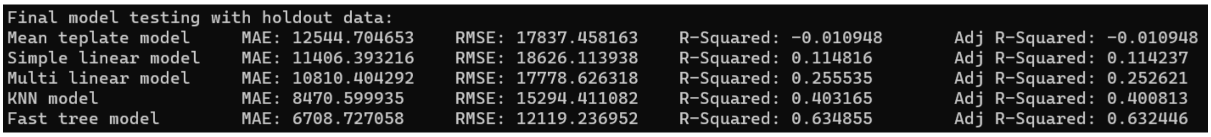
When k=10, the model performs best with least error.

After all the models had been fine-tuned, the summaries of each were written to the regression-analysis-conclusion.txt file at run-time, in order to help conclude which model is the best and the one to be integrated into the final system.

Here is the contents:

```
Mean template model:
Errors: MAE = 14258.16907, RMSE = 105403.00715, R-Squared = -0.007762
Hyperparameters: None
Features: None

Simple linear regression model:
Errors: MAE = 11021.46628, RMSE = 19091.115517, R-Squared = 0.11079
Hyperparameters: Z-score threshold for Prod. Year outliers = 4
Features: Prod. year scaled

Multi linear regression model using gradient descent:
Errors: MAE = 10290.016309, RMSE = 17713.968782, R-Squared = 0.248435, Adjusted R-Squared = 0.246804
Hyperparameters: Learning rate = 0.5, Max no. of epochs = 1000, Convergence threshold = 0.1
Features: Prod. year scaled, Mileage in km scaled, Engine displacement scaled, Is jeep, Is diesel, Is
tiptronic

K-nearest neighbours regression model:
Errors: MAE = 7953.922113, RMSE = 15288.401675, R-Squared = 0.452728, Adjusted R-Squared = 0.450933
Hyperparameters: K = 10
Features: Prod. year scaled, Mileage in km scaled, Engine displacement scaled, Is jeep, Is diesel, Is
tiptronic

Microsft.ML Fast tree regression trainer:
Errors: MAE = 6803.006273, RMSE = 13244.066399, R-Squared = 0.586265, Adjusted R-Squared = 0.584755
Hyperparameters: Minimum sample count per leaf = 8 ***Inconclusive value - check with holdout data***
Features: ProdYear, MileageInKm, EngineDisplacement, ManufacturerEncoded, ModelEncoded, IsJeep, IsDiesel,
IsTiptronic, IsLeatherInterior, IsEngineTurbo
```

Lastly, the models are tested with holdout data to gauge whether the model is able to generalise and predict new, unseen data. Below is a table that shows the model performance with the holdout data.



```
Final model testing with holdout data:
Mean teplate model     MAE: 12544.704653    RMSE: 17837.458163    R-Squared: -0.010948      Adj R-Squared: -0.010948
Simple linear model    MAE: 11406.393216    RMSE: 18626.113938    R-Squared: 0.114816       Adj R-Squared: 0.114237
Multi linear model     MAE: 10810.404292    RMSE: 17778.626318    R-Squared: 0.255535       Adj R-Squared: 0.252621
KNN model              MAE: 8470.599935     RMSE: 15294.411082    R-Squared: 0.403165       Adj R-Squared: 0.400813
Fast tree model        MAE: 6708.727058     RMSE: 12119.236952    R-Squared: 0.634855       Adj R-Squared: 0.632446
```

Clearly, the fast tree regression model performs the best, so this was the model I integrated into the final car price estimator system.

Final system testing

Test table

Test no.	Purpose of testing	Test data (Typical, Boundary, Erroneous)	Expected outcome	Actual outcome	Test passed (Y/N)
1	To check that the validation of the car make input is working.	T: Audi selected.	No errors indicated and model dropdown consists of Audi models.	As expected. (evidence 1.1)	Y
		E: Make dropdown clicked but no make selected.	Suitable error indicated.	As expected. (evidence 1.2)	Y
2	To check that the validation of the car model input is working.	T: Audi selected for make, and A4 selected for model.	No errors indicated.	As expected. (evidence 2.1)	Y
		E$_1$: Attempt to select a model before entering a make.	Error icon and message displayed.	As expected. (evidence 2.2)	Y
		E$_2$: Audi selected for the make but no model	Error icon and message displayed.	As expected. (evidence 2.3)	Y

		selected after clicking the model dropdown.			
3	To check that the validation of the year of manufacture input is working.	E: Year 1950 typed and enter button pressed.	Entered value is replaced with the lowest valid year: 1990.	As expected. (evidence 3.1)	Y
4	To check that the validation of the mileage input is working.	T: 10000 entered.	No errors indicated.	As expected. (evidence 4.1)	Y
		B: 250000 entered.	No errors indicated.	As expected. (evidence 4.2)	Y
		E_1: 400000 entered	Suitable error indicated	As expected. (evidence 4.3)	Y
		E_2: ‘abcdef’ entered	Suitable error indicated.	As expected. (evidence 4.4)	Y
5	To check that the validation of the fuel input is working.	E: Fuel dropdown clicked but no fuel selected.	Suitable error indicated.	As expected. (evidence 5.1)	Y
6	To check that the reset button works as intended.	Some inputs entered and some errors indicated - See evidence 6.1.	All inputs are reset to their initial state, and all error indications are removed.	Inputs are reset correctly but error indications are still present. (evidence 6.1)	N
7	To check that the program works correctly when the make is initially selected then de-selected.	Alfa Romeo selected as the make, then de-selected, and then model dropdown clicked.	Suitable errors indicated and model dropdown is empty.	Errors indicated but alfa romeo models appear in dropdown. (evidence 7.1)	N
8	To check that the validation of the input completion after the ‘Get	T: Valid inputs entered - See evidence 8.1.	No errors indicated and a new page is displayed with the price.	As expected. (evidence 8.1)	Y

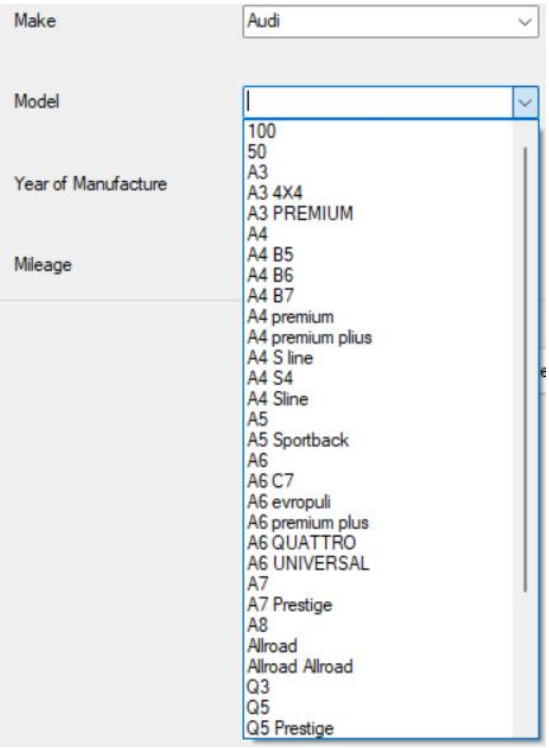
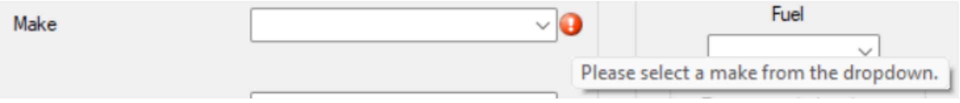
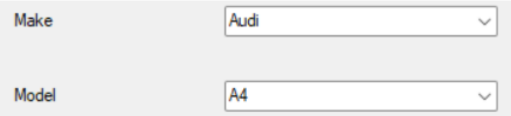
	Price Estimate' button is pressed works.	E: Some invalid inputs entered - See evidence 8.2.	Suitable error message displayed.	As expected. (evidence 8.2)	Y
9	To check that the car details are correctly displayed on the output page.	Inputs entered - see evidence 9.1.	Car details displayed match exactly with what was entered.	As expected. (evidence 9.1)	Y
10 (a)	To check that the print button shows a print preview.	Inputs entered - see evidence 10.1 - and print button pressed.	Print preview shown contains the price and additional info.	As expected. (evidence 10.1)	Y
10 (b)	To check that the print dialog show after closing the print preview	Test that follows 10a. Close button on the print preview pressed.	Print dialog shown.	As expected (evidence 10.2)	Y
10 (c)	To check that the price estimation can be printed successfully.	Test that follows 10b. Local printer selected and 'OK' button on the print dialog pressed.	Page successfully printed and matches the appearance of the print preview.	As expected. (evidence 10.3)	Y

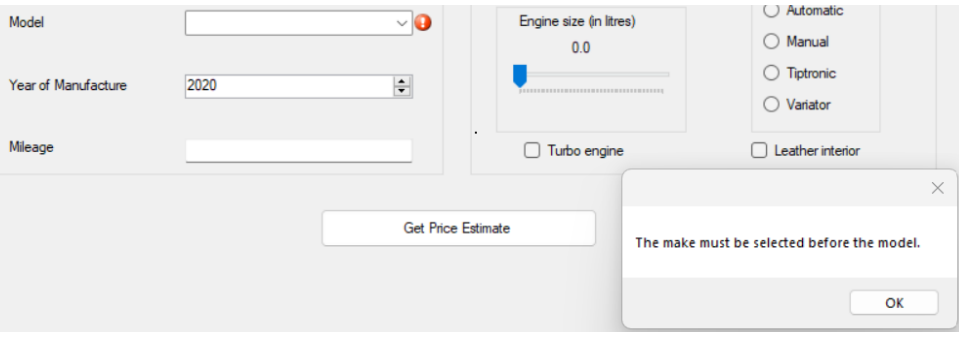
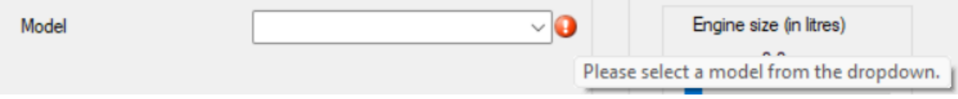
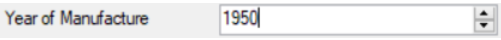
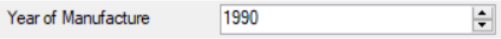
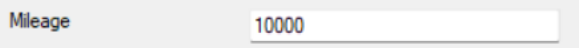
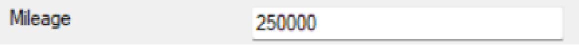
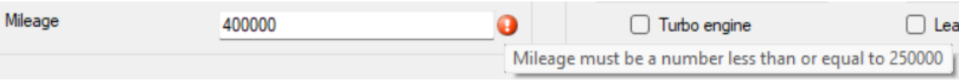
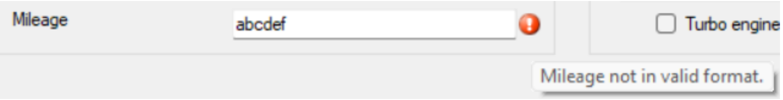
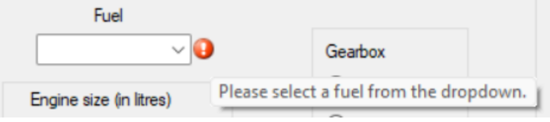
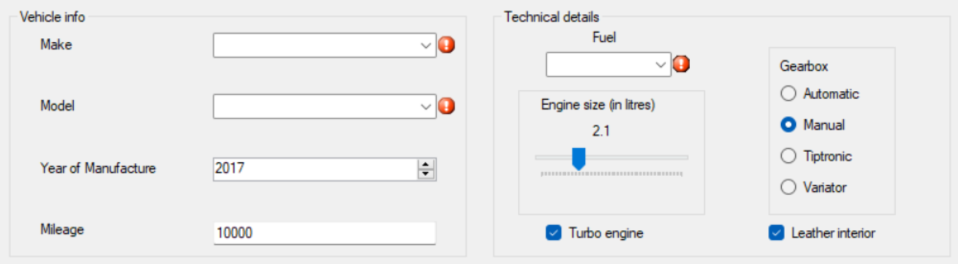
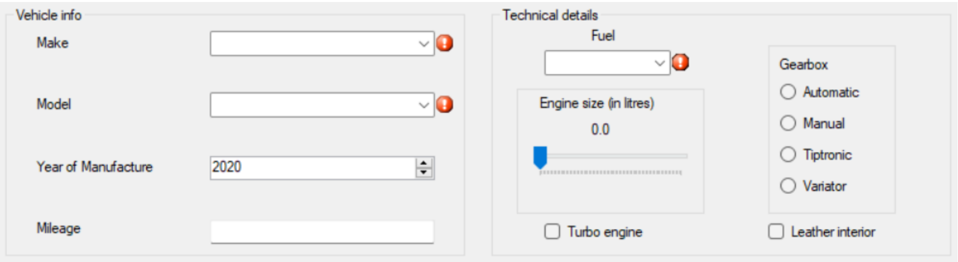
Re-testing

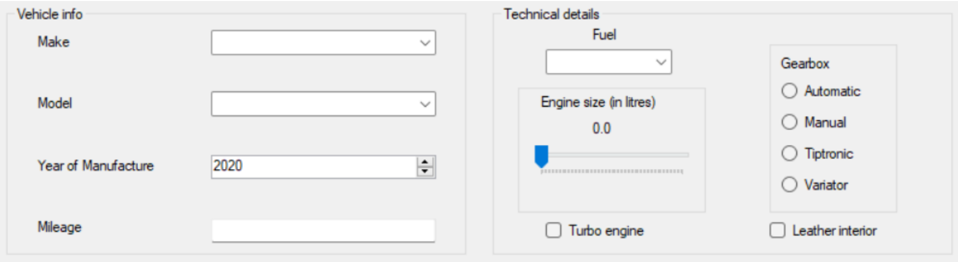
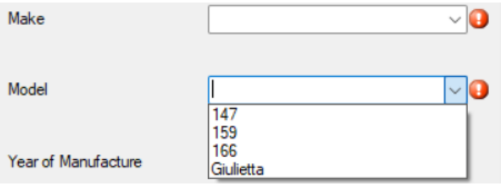
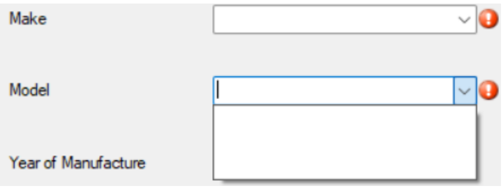
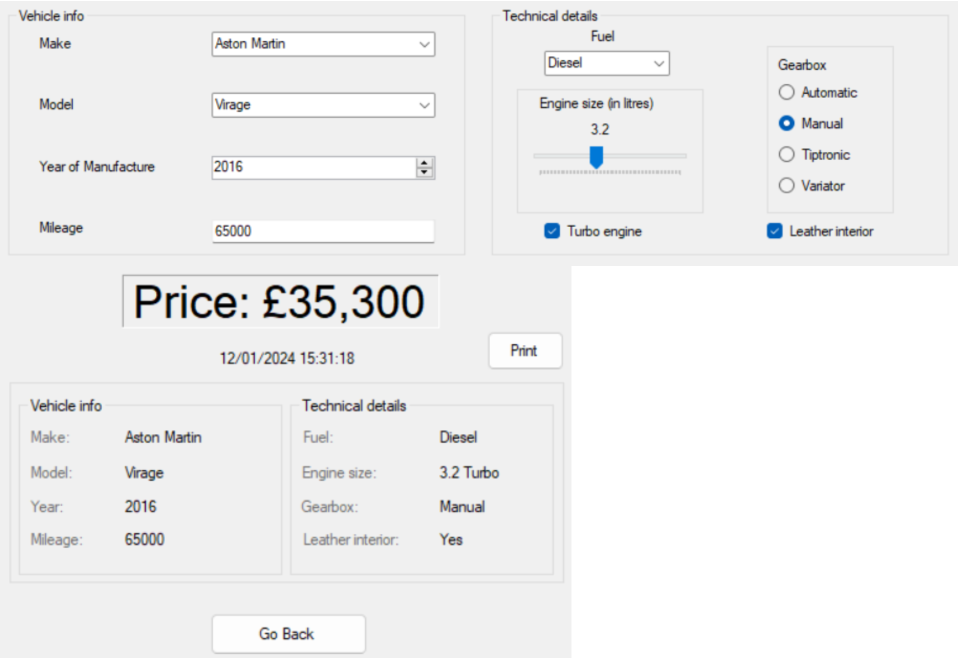
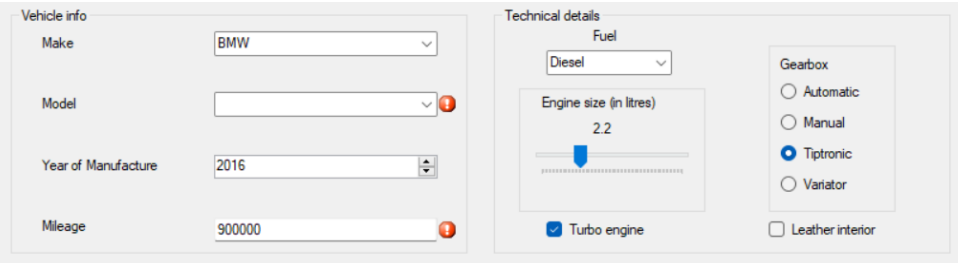
Test no.	Corrective actions	Test data	Expected outcome	Actual outcome	Test passed (Y/N)
6	Modified the ResetForm() method by adding lines to set the error to null for the make, model, mileage and fuel respective controls.	Form in same state as before - See evidence 6.1	All inputs are reset to their initial state, and all error indications are removed.	As expected. (evidence 6.2)	Y
7	Modified the event handler method that gets called when the user leaves the make	Form in same state as before- See	Suitable errors indicated and model	As expected (evidence 7.2)	Y

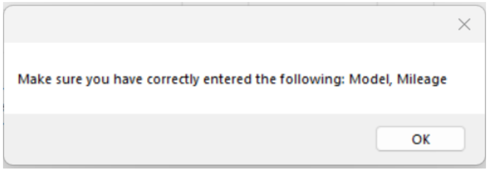
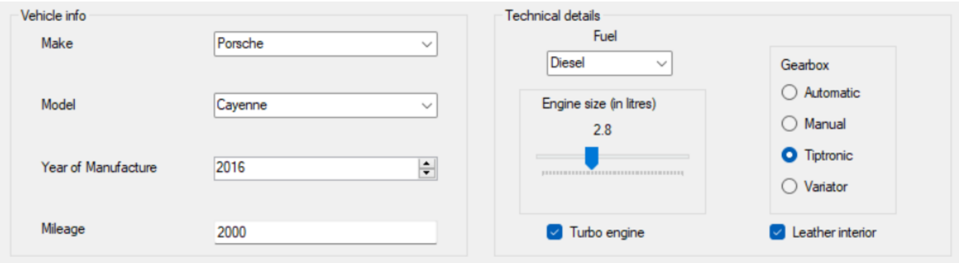
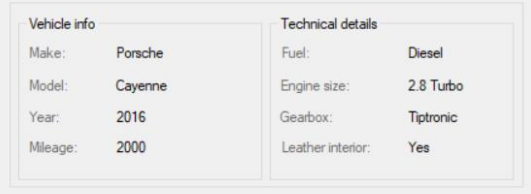
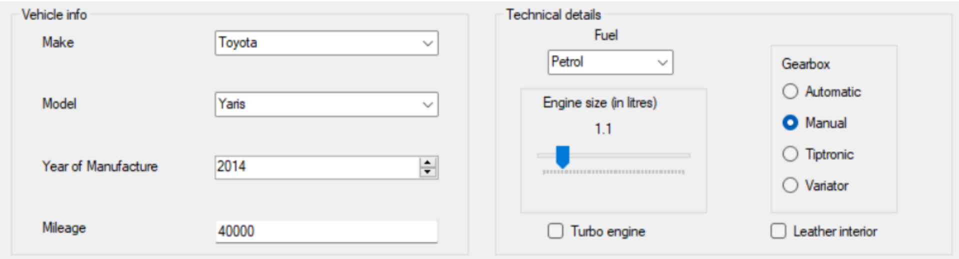
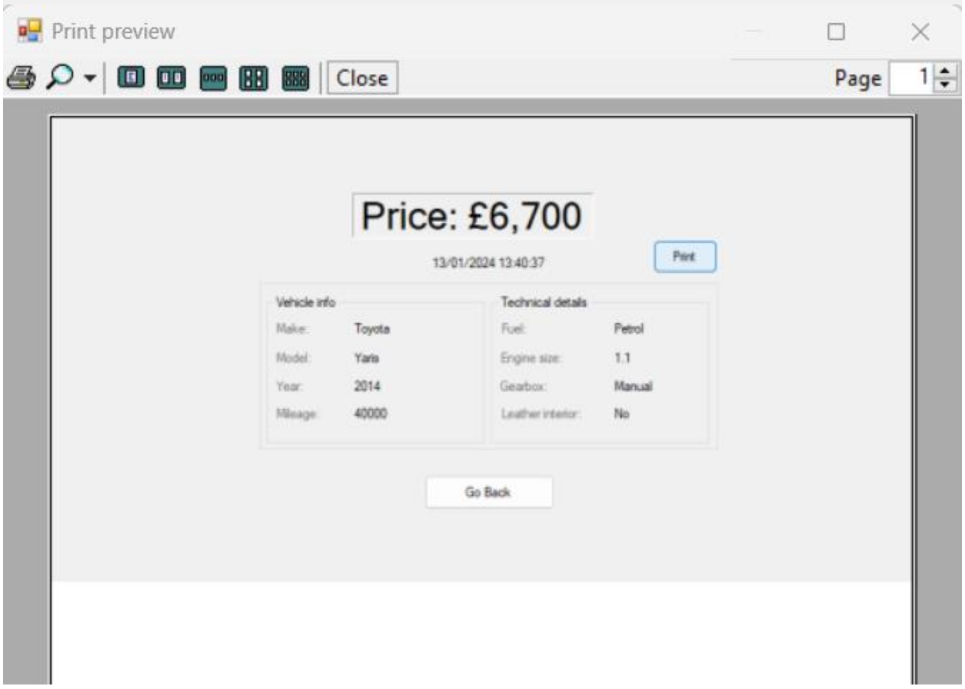
	dropdown control. Included a selection statement that checks that the user’s input text matches one of the makes in the dropdown, and, if not, the model dropdown options are cleared.	evidence 7.1	dropdown is empty.		

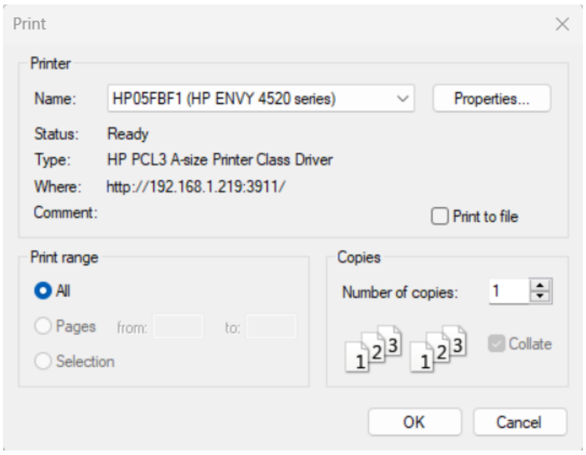
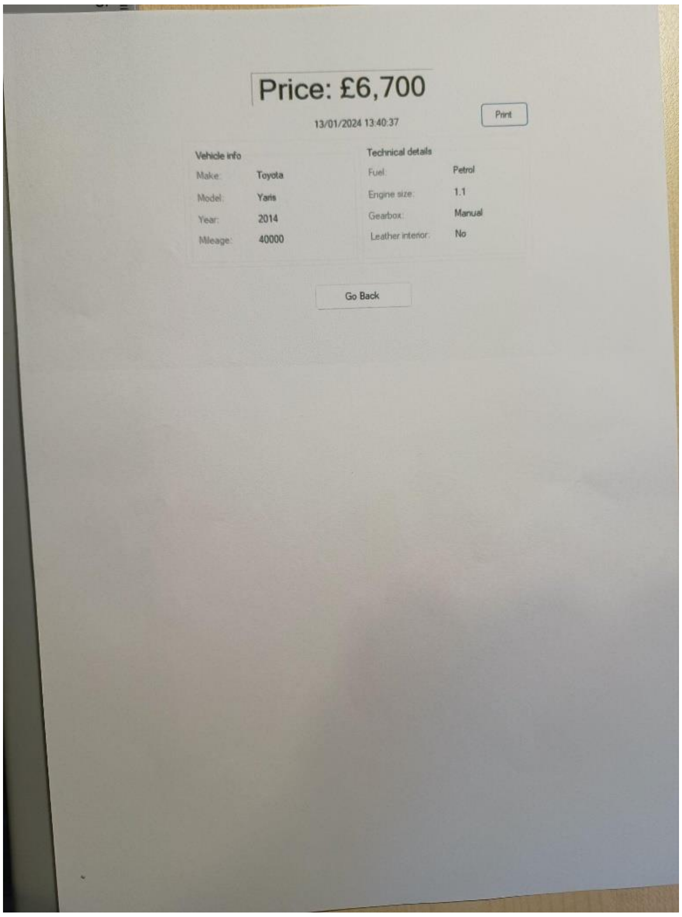
Test evidence

Evidence no.	Screenshot
1.1	 Make: Audi Model: Year of Manufacture Mileage 100 50 A3 A3 4X4 A3 PREMIUM A4 A4 B5 A4 B6 A4 B7 A4 premium A4 premium plus A4 S line A4 S4 A4 Sline A5 A5 Sportback A6 A6 C7 A6 evropuli A6 premium plus A6 QUATTRO A6 UNIVERSAL A7 A7 Prestige A8 Allroad Allroad Allroad Q3 Q5 Q5 Prestige
1.2	 Make: Fuel: Please select a make from the dropdown.
2.1	 Make: Audi Model: A4

2.2	 Model Year of Manufacture 2020 Mileage Engine size (in litres) 0.0 ○ Automatic ○ Manual ○ Tiptronic ○ Variator □ Turbo engine □ Leather interior Get Price Estimate The make must be selected before the model. OK
2.3	 Model Engine size (in litres) Please select a model from the dropdown.
3.1	Before:  Year of Manufacture 1950 After:  Year of Manufacture 1990
4.1	 Mileage 10000
4.2	 Mileage 250000
4.3	 Mileage 400000 □ Turbo engine □ Lea Mileage must be a number less than or equal to 250000
4.4	 Mileage abcdef □ Turbo engine Mileage not in valid format.
5.1	 Fuel Gearbox Engine size (in litres) Please select a fuel from the dropdown.
6.1	Before:  Vehicle info Make Model Year of Manufacture 2017 Mileage 10000 Technical details Fuel Engine size (in litres) 2.1 Gearbox ○ Automatic ● Manual ○ Tiptronic ○ Variator ☑ Turbo engine ☑ Leather interior After:  Vehicle info Make Model Year of Manufacture 2020 Mileage Technical details Fuel Engine size (in litres) 0.0 Gearbox ○ Automatic ○ Manual ○ Tiptronic ○ Variator □ Turbo engine □ Leather interior

6.2	 Vehicle info Make: Model: Year of Manufacture: 2020 Mileage: Technical details Fuel: Engine size (in litres): 0.0 Gearbox: ○ Automatic ○ Manual ○ Tiptronic ○ Variator □ Turbo engine □ Leather interior
7.1	 Make: Model: 147 159 166 Giulietta Year of Manufacture
7.2	 Make: Model: Year of Manufacture
8.1	 Vehicle info Make: Aston Martin Model: Virage Year of Manufacture: 2016 Mileage: 65000 Technical details Fuel: Diesel Engine size (in litres): 3.2 Gearbox: ○ Automatic ● Manual ○ Tiptronic ○ Variator ☑ Turbo engine ☑ Leather interior Price: £35,300 12/01/2024 15:31:18 Vehicle info Make: Aston Martin Model: Virage Year: 2016 Mileage: 65000 Technical details Fuel: Diesel Engine size: 3.2 Turbo Gearbox: Manual Leather interior: Yes
8.2	 Vehicle info Make: BMW Model: Year of Manufacture: 2016 Mileage: 900000 Technical details Fuel: Diesel Engine size (in litres): 2.2 Gearbox: ○ Automatic ○ Manual ● Tiptronic ○ Variator ☑ Turbo engine □ Leather interior

	 Make sure you have correctly entered the following: Model, Mileage OK
9.1	 Vehicle info Make: Porsche Model: Cayenne Year of Manufacture: 2016 Mileage: 2000 Technical details Fuel: Diesel Engine size (in litres): 2.8 Gearbox: ○ Automatic ○ Manual ● Tiptronic ○ Variator ☑ Turbo engine ☑ Leather interior  Vehicle info Make: Porsche Model: Cayenne Year: 2016 Mileage: 2000 Technical details Fuel: Diesel Engine size: 2.8 Turbo Gearbox: Tiptronic Leather interior: Yes
10.1	 Vehicle info Make: Toyota Model: Yaris Year of Manufacture: 2014 Mileage: 40000 Technical details Fuel: Petrol Engine size (in litres): 1.1 Gearbox: ○ Automatic ● Manual ○ Tiptronic ○ Variator □ Turbo engine □ Leather interior  Print preview Close Page 1 Price: £6,700 13/01/2024 13:40:37 Print Vehicle info Make: Toyota Model: Yaris Year: 2014 Mileage: 40000 Technical details Fuel: Petrol Engine size: 1.1 Gearbox: Manual Leather interior: No Go Back

10.2	 Print Printer Name: HP05FBF1 (HP ENVY 4520 series) Status: Ready Type: HP PCL3 A-size Printer Class Driver Where: http://192.168.1.219:3911/ Comment: □ Print to file Print range ● All ○ Pages from: to: ○ Selection Copies Number of copies: 1 ☑ Collate
10.3	 Price: £6,700 13/01/2024 13:40:37 Vehicle info Make: Toyota Model: Yaris Year: 2014 Mileage: 40000 Technical details Fuel: Petrol Engine size: 1.1 Gearbox: Manual Leather interior: No

Video

Here is a link to a short video (1 min 37 secs) that demonstrates that the car price estimator program I have created works as described: https://youtu.be/H5B1dWf65uo

The video demonstrates how the form works as well as the print and reset functions. It also highlights how the price changes when changing the inputs; for example, when the mileage decreased, the price goes up and when the year of production decreases, the price goes down.

Evaluation

Comparison of final project against the objectives

Inputs:

1.a.i. *Suitable control for an enumeration user input.* This was achieved through the use of a combo box (dropdown) for the make, model and fuel as these inputs consist of a large number of options. A set of radio buttons was used for the gearbox as this only comprised four options.

1.a.ii. *Suitable control for a boolean user input.* This was achieved through the use of checkboxes for the turbo engine and leather interior inputs.

1.a.iii. *Suitable control for an integer user input.* This was achieved through the use of a numeric incrementer for the production year due to its relatively small range. A text box was used for the mileage input due to its much larger range of possible values.

1.a.iv. *Suitable control for a single (float) user input.* This was achieved through the use of a slider for the engine size due its small range.

1.b.i. *Content is spaced and not too dense.* This was achieved by having two separate panels, one for input and one for output, and by leaving adequate space between labels, inputs and buttons.

1.b.ii. *Controls are sensibly organised.* This was achieved by separating the user inputs into two container boxes for vehicle information and technical details.

1.b.iii. *All inputs are necessary and impactful.* This was achieved by only choosing inputs that are predictor variables for the integrated regression model.

1.c.i. *Model dropdown displays only applicable models for the user's selected make.* This was achieved by using event handling methods and a make-model dictionary (populated from a json file) to populate the model dropdown.

1.d.i. *Reset button resets all user inputs.* This was achieved with an event handler method that is triggered once the reset button is pressed, and returns all input values to default and removes any error indications.

2.a.i. E*rror raised for incorrect data type for text-based inputs.* This was achieved through the use of a try-catch statement for the mileage input (the only text-based input) and the errorProvider class provided by Windows Forms.

2.a.ii. *Error raised for not in range for numeric inputs.* This was achieved by using the errorProvider.SetError() method when the input outside the acceptable range for the mileage. For the production year, an error does not need to be raised as the built-in functionality for the NumericUpDown class means that an input out of range is instantly changed to the least/greatest acceptable value within range.

2.a.iii. *Error raised if no option has been selected for dropdowns.* This was achieved by checking the value of the SelectedIndex attribute.

2.a.iv. *Error raised if no option has been checked for radio buttons.* This was achieved by checking the returned value of the Checked() method belonging to each radio button.

2.a.v. *Error raised if the user attempts to enter a value that is not an option in the dropdown.* This was achieved similarly to 2.a.iii by checking the value of the SelectedIndex.

2.a.vi. *Error raised if the user attempts to select a model before the make*. This was achieved by using a selection statement to check if the SelectedItem attribute of the make dropdown is null, and a built-in class MessageBox is used to display the error message.

2.b.i. *Displays a message if any inputs have been incorrectly completed when the form is submitted.* This was achieved by checking for the presence of errors and invalid inputs and using the MessageBox class to indicate any of these invalid inputs.

Processing:

3.a.i. *Dataset must be initially checked against a data dictionary to ensure data is in a valid form.* This was achieved by checking that each entry in the dataset matches the corresponding regular expression defined in the data dictionary, and replacing non-standard characters with underscores to notify later stages of regression analysis.

3.b.i. *Dataset must be checked for missing values.* This was achieved by defining a method that checks if each data entry is missing using the built-in method IsNullOrEmpty().

3.b.ii. *Selectively delete or impute missing values.* There were no missing values found so this was not necessary, however, if there were missing values, the same process would have been followed as explained in 3.b.v.

3.b.iii. *Check for duplicate records and remove them if found.* This was achieved by using LINQ to retrieve groups of rows where the values in the rows are the same, and then remove all but 1 of each of these rows in each group.

3.b.iv. *Locate outliers for each numerical feature.* This was achieved by defining three separate functions which all calculate and output outliers using different methods: z-scores, modified z-scores and interquartile range.

3.b.v *Selectively delete or impute outliers.* This was achieved by defining two methods, one for imputing binary columns with the mode and one for imputing numerical columns with the mean.

3.b.vi. *Engineer new features if suitable.* This was achieved by defining a method to split the engine volume column into a numerical and binary column: ‘engine displacement’ and ‘is engine turbo’.

3.b.vii. *Transform categorical variables into a numerical format.* This was achieved by assigning a 1 or 0 to new binary columns depending on the value at each row.

3.b.viii. *Scale numerical features.* This was achieved by defining two methods for scaling numerical columns: standardisation and min-max scaling.

3.c.i. *Calculate correlation coefficients for continuous and dichotomous variables.* This was achieved by defining a method to calculate the Pearson correlation coefficient for continuous variables, and likewise a method to calculate the point biserial correlation coefficient for dichotomous variables.

3.c.ii. *Visualise data correlation through scatter plots.* This was achieved by creating a scatter plot form, provided by the OxyPlot library, that plots the given data points.

3.c.iii. Visualise data distribution through box plots. This was achieved by creating a box plot form, provided by the OxyPlot library, that displays multiple box plots of a given predictor.

3.c.iv. *Obtain a ranking of feature importance.* This was achieved by defining a method for ranking continuous variables based on their pearson coefficients and one for ranking dichotomous variables based on their point biserial coefficients, in descending order of their coefficients.

3.d.i. *Simple linear regression coded from scratch.* This was achieved by training the model using a formulaic approach to calculate the model coefficients, and these coefficients were used to predict the price in the Fit() method.

3.d.ii. *Multi linear regression training coded from scratch.* This was achieved by using gradient descent optimisation to calculate the model coefficients.

3.d.iii. *K-nearest neighbours regression coded from scratch.* This was achieved by using the algorithm set out in the design section, to identify the k-nearest neighbours and take the mean price of these neighbours.

3.d.iv. *Decision tree training coded using a C# supported library.* This was achieved through the use of the FastTree regression trainer provided by ML.NET framework.

3.e.i. *The model's performance is assessed using cross validation.* This was achieved by implementing K-fold cross validation.

3.e.ii. *The error metrics used are selected such that there is minimised bias towards any one model.* This was achieved by measuring all models' performance with four different error metrics: MAE, RMSE, R-squared, Adjusted R-squared - each of which cater differently to outliers, multiple predictors etc.

3.f.i. *Each regression model implementation should include data preprocessing.* This was achieved by applying the techniques in 3.b differently for each regression model.

3.f.ii. *Split data into training and test.* This was achieved by defining a method that splits the data into three separate datatables for training, testing and holdout.

3.f.iii. *Perform feature selection.* This was achieved by applying the techniques in 3.c in order to comprehend what are the better features before doing any training and validation.

3.f.iv. *Train model with training data.* This was achieved through the use of different techniques as described in 3.d.

3.f.v. *Fit model with test data.* This was achieved through the use of different techniques, depending on the regression model.

3.f.vi. *Evaluate the model with a range of error metrics.* This was achieved by calculating the error metrics described in 3.e.ii, comparing the validation data's actual prices to the predicted prices.

3.f.vii. *Fine-tune the model by altering features and/or hyperparameters.* This was achieved by defining a set of possible hyperparameter values and running the model to determine the optimal value, similar to features, where the regression model's performance was measured with selective feature combinations.

3.g.i. *Each fine-tuned model is tested once more.* This was achieved by testing the fine-tuned models each on unseen, holdout data.

3.g.ii. *Best-performing model is determined.* This was achieved by viewing which of the models had the best error metrics when tested with the holdout data.

3.g.iii. *Best model integrated with the UI application.* This was achieved by saving the trained model to a file and then defining a prediction engine at run-time.

4.a.i. *Model initially saves to file.* This was achieved by training the model with the optimal features and hyperparameters and then saving it to a zipped file.

4.a.ii. *Model instantiated at run-time.* This was achieved by loading the model from the file in the form_load() event handler.

4.a.iii. *User's input data fed into model.* This was achieved by creating an instance of the CarData class and assigning the relevant attributes to the current input values on the form.

4.a.iv. *Model predicts the car price.* This was achieved by passing the user's data into the prediction function and calling the Predict() method provided by ML.NET.

4.a.v. *Price checked for validity.* This was not achieved because there was no feasible method of calculating the validity of the prediction due to the model being a decision tree.

4.a.vi. *Confidence level calculated.* This was not achieved due to the integrated model being a decision tree, and I could not find a feasible method of calculating the level of accuracy of a prediction.

5.a.i. *Program consists of a regression analysis console application.* This was achieved by using a .NET Console App for the entire regression analysis process.

5.a.ii. *Program consists of a GUI form application.* This was achieved by using a .NET Windows Forms application for the final system.

5.b.i. *Regression analysis split into sensible classes.* This was achieved by creating many classes, including abstract and static classes, as shown by the UML class diagram in the design.

5.b.ii. *Each class must have a clear responsibility.* This was achieved by creating new classes for different stages of the regression analysis process, and they all have identifiers that clearly indicate their responsibility.

5.b.iii. *Utilise inheritance and polymorphism where necessary.* This was achieved by having the regression model parent abstract class and each individual regression model class inheriting this functionality. Composition was also used throughout, as shown by the UML class diagram in the design.

Outputs:

6.a.i. *Price formatted well and in large lettering.* This was achieved by positioning the price at the top of the content on the output page, and in the largest font.

6.a.ii. *User's vehicle information is displayed.* This was achieved by presenting the information in the same way that it was inputted, with a container for vehicle info and a container for technical details.

6.a.iii. *Content organised nicely.* This was achieved by containing all vehicle information in a container box and by positioning all containers with adequate blank space around them.

6.a.iv. *There is a colour-coded confidence level regarding the accuracy of the accuracy of the estimation shown.* This was not achieved for the same reason as described in 4.a.vi.

6.a.v. *An error message is displayed if there was a problem with generating a price estimation.* This was achieved through the use of a try-catch statement.

6.a.vi. *The date of valuation is displayed.* This was achieved using the built-in DateTime.Now() funcion.

6.a.vii. *There is an option to 'Go Back'.* This was achieved by using the button control, and the BringToFront() method to display the form panel.

6.a.viii. *There is an option to print.* This was achieved by using the PrintDocument, PrintPageEventHandler, PrintPreviewDialog and PrintDialog classes provided by Windows Forms.

6.b.i. *The 'Go Back' button takes the user back to the form page.* This was achieved through the use of the BringToFront() method to display the form panel.

6.b.ii. *The form page still contains the user's details.* This was achieved by not altering the current values of any of the input fields in the form.

6.c.i. *Print preview appears initially.* This was achieved through the use of the printPreviewDialog.Show() method.

6.c.ii. *Print details are able to be customised according to the user's preferences.* This was achieved by instantiating the PrintDialog class.

6.c.iii. *Document should print if the user confirms so.* This was achieved through the use of the provided Print() method.

User feedback

I gave the program to my primary client, Bernhard Klinger, and below is a summary of the feedback he has provided.

Aspects of the program he liked:

- User interface is very easy to use.
- The ability to go back to the form page after getting a price estimation, changing one parameter and then you get an instant, new estimation.
- The fact that you can print the estimation with the details entered.
- It is very fast to estimate, after submitting car details.
- The design of the interface would be suitable for different screen sizes, e.g. mobile phone.
- Validation checks are implemented well - he could not break the system.
- Error messages are easy to understand.

Aspects of the program he thinks could be improved:

- The arrow of the dropdown must be pressed to drop down, so you cannot press anywhere on the dropdown.
- The technical detail inputs do not update when the model is selected. For example, it should not allow you to choose a 6 litre engine size for a Volkswagen Golf.
- The data could be retrieved from the live DVLA database, and you would only need to input your registration number.
- It does not provide a range of prices or a confidence level to indicate the accuracy of the price estimation.
- After entering the details of his Honda Civic, the estimator overpriced his car.

Analysis of user feedback

Overall, I am satisfied with the feedback from the primary client as it is evident that the vast majority of core requirements have been met. He has confirmed that the user interface is well-designed and easy to use, which were important aspects that he raised in the interview before creating the program. Additionally, he also commented about the speed of providing the price estimation being very fast, which was another key requirement mentioned in the initial interview. Whilst having a print option was one of the less crucial aspects according to the prospective user survey, providing the option to print also proved to be beneficial for the client’s experience and the button is positioned in a way that should not hinder a user’s experience if they do not want to print their price estimation.

The display of a confidence level was the only major requirement that I failed to provide in the final system. Unfortunately, the regression model that I implemented was a tree model and the implementation of the model means that I cannot easily provide a confidence level. If I were to have integrated a linear regression, there are formulae that allow you to obtain a confidence interval, however, the linear regression models that I developed did not perform very well with my data. I will discuss more about Bernhard’s suggested improvements in detail below.

Improvements

Below, I have highlighted some improvements I would make to the system if I were to revisit it in the future:

- If I had access to the DVLA database, I would integrate API requests and modify the form to have the registration number as the only input, and the registration number would be passed as a parameter for a GET request. This would mean there are far less inputs and would enhance the user experience and speed of obtaining a valuation.

- In order to drastically improve the accuracy of the price estimations, I would need to access a large database that holds live data, as opposed to the current dataset used which is from 2020 and only holds approximately 20,000 records, of which the accuracy of the data is unknown. This would allow me to conduct regression analysis on a larger scale and potentially could implement more powerful regression models that require large amounts of data such as neural networks. A larger dataset would also increase the model's ability to generalise to new, unseen data and reduce the issue of overfitting that is present in the current model. I could also consider selecting a regression model that provides confidence intervals in order to display a colour-coded confidence level on the output page.

- If the access to DVLA database is too expensive, I would implement some event handling that, once a model has been selected, it triggers the inputs like fuel and gearbox to only accept values that are possible for the selected model.

- I would modify the combo box control event handler methods to enable dropdown when the user clicks anywhere on the control box, as Bernhard mentioned above.

- The prospective user survey conducted before the creation of the program indicated that users would prefer to be able to change the currency. Hence, I would access a currency converter API to allow the user to customise the currency before pressing the 'Get Price Estimate' button.