1. Data Processing for Machine Learning

1.1 Data Understanding/Exploration

The first step in working with machine learning is to have the necessary libraries such as pandas, numpy, matplotlib, and seaborn loaded. These libraries are used for tasks like handling data and making graphs. Once these are loaded, the data is then loaded using pandas. It's important to understand the dataset well. This means knowing the size of the dataset, what the data represents, and checking for missing values or any data that seems unusual and could affect results. It must be determined whether the work involves regression (predicting a number) or classification (choosing a category). Understanding these details helps in preparing the data properly for machine learning.

1.2 Data Cleaning

Data cleaning is recognized as an important step in data preparation because it ensures that the data becomes accurate, complete, and of high quality before it is used for Machine Learning. This process involves multiple stages such as having missing values filled in and dealing with outliers.

1.2.1 Filling missing values

Filling missing values is considered a crucial step in the data preprocessing phase of a machine learning project. It is observed that in this dataset, missing values are present in the categories of mileage, registration, colour, year, body, and fuel.

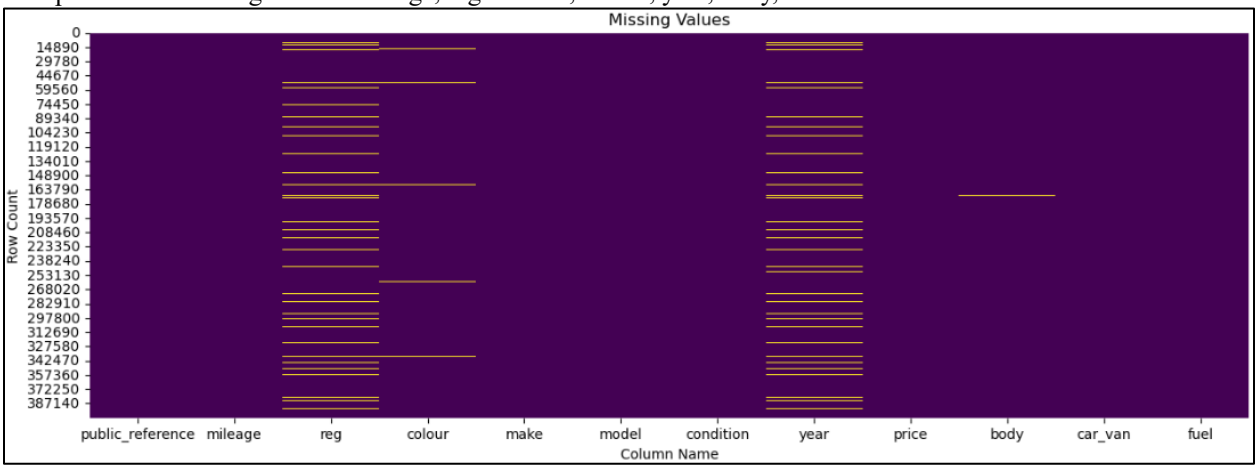

Fig. 1 Missing values in the dataset

To begin with mileage, 127 missing values are noted. The average mileage per year will be calculated. The reason for calculating the average mileage per year is that the dataset includes new cars with missing mileage data. If the mean or median were used to fill these, the value would not be accurate because the mileage of new cars is typically lower compared to used cars.

```
adv['mileage'] = adv.groupby('year')['mileage'].transform(lambda x: x.fillna(x.mean()))
```

Missing values in mileage will be filled by the above code based on average mileage per year.

In the reg column, 31,857 missing values are observed. Missing values in the reg column are being filled based on the registration year, according to the Vehicle registration plates of the United Kingdom. For the year 2020, the code used is either 20 or 69. For cars with mileage less than 1000, which are typically sold from March 1st to August 31st, the code 20 is used. Conversely, for cars with mileage greater than 1000, likely sold from September 1st to late February, the reg_code 69 is used, aligning with their respective registration periods.

```
con1 = adv.loc[(adv['reg'].isna() | (adv['reg'] == '')) &
        (adv['year'] == 2020) &
        (adv['mileage'] <= 1000), 'reg'] = '20'
```

```
con2 = adv.loc[(adv['reg'].isna() | (adv['reg'] == '')) &
        (adv['year'] == 2020) &
        (adv['mileage'] >= 1000), 'reg'] = '69'
```

To fill the year, it is noted that approximately 33,311 rows have missing values, and the latest year in the dataset is 2020. Therefore, under the condition that the vehicle is new and the year is null, a few rows will be filled with the year 2020.

```
1 adv.loc[(adv['vehicle_condition'] == 'NEW') & (adv['year_of_registration'].isnull()), 'year_of_registration'] = 2020
```

To fill the remaining missing values in the year column, a dictionary will be created by referring to Vehicle registration plates of the UK, and it will be mapped with reg. This mapping will assist in filling in missing values where the reg code is not missing.

```
adv['year'].fillna(adv['mapped_year'], inplace=True)
```

Missing values in body, colour, and fuel can be filled by using the most common value method. However, the missing values will be dropped, as there are only 5378 in colour, 837 in body, and 601 in fuel. Thus, all missing values have been filled, and the next step to be taken is to deal with outliers.

1.2.2 Dealing with outliers

To deal with outliers, I'll use the Inter Quantile Range method. For the price, I'm grouping standard make, vehicle condition, and price because this dataset includes cars ranging from the cheapest to the most expensive depending on the make. If I simply use the IQR method for the price, then I'll lose a valuable dataset of expensive cars the model will not get trained on the expensive cars dataset and maybe it won't be able to predict the correct price of expensive cars to deal with it, I'll create two variables to determine the 25th and 75th percentiles of the price column, and then calculate upper and lower bounds to remove outliers. But for mileage and year I will simply use IQR method which you can see in the below image.

```
q1_price = adv.groupby(['make', 'condition'])['price'].transform('quantile', 0.25)
q3_price = adv.groupby(['make', 'condition'])['price'].transform('quantile', 0.75)
price_iqr = q3_price - q1_price
lower_bound_price_val = q1_price - (1.5 * price_iqr)
upper_bound_price_val = q3_price + (1.5 * price_iqr)
adv = adv.query('price <= @upper_bound_price_val')
adv = adv.query('price >= @lower_bound_price_val')
```

Dealing with outliers in Price

```
q1_mileage,q3_mileage = np.percentile(mileage,[25,75])
mileage_iqr = q3_mileage - q1_mileage
lower_bound_mileage_val = q1_mileage - (1.5 * mileage_iqr)
upper_bound_mileage_val = q3_mileage + (1.5 * mileage_iqr)
adv = adv.query('mileage <= @upper_bound_mileage_val')
adv = adv.query('mileage >= @lower_bound_mileage_val')
```

Dealing with outliers in Mileage

```
q1_yor,q3_yor = np.percentile(yor,[25,75])
yor_iqr = q3_yor-q1_yor
lower_bound_yor_val = q1_yor - (1.5 * yor_iqr)
upper_bound_yor_val = q3_yor + (1.5 * yor_iqr)
adv = adv.query('year >= @lower_bound_yor_val')
adv = adv.query('year <= @upper_bound_yor_val')
```

Dealing with outliers in Year

After missing values have been filled and outliers dealt with, the length of the dataset is (356,518 rows and 12 columns), and the next step to be taken is feature engineering.

2. Feature Engineering

Feature engineering is recognized as the process of converting raw data into more useful and informative features for further analysis. In this process, new variables are created or existing ones are modified to better capture the underlying patterns and relationships in the data, resulting in more accurate and insightful analytical outcomes. This step is considered critical for preparing data for effective analysis and modelling.

2.1 Deriving Features

A new feature called age has been created to identify the age of a car. The current year is 2024, so subtracting 2024 from the year will result in the calculation of a car's age.

	public_reference	mileage	reg	colour	make	model	condition	year	price	body	car_van	fuel	age
0	202006039777689	0.00	20.00	Grey	Volvo	XC90	NEW	2020	73970	SUV	False	Petrol Plug-in Hybrid	4

In the condition column, only two values exist, namely 'New' and 'Used'. Cars with a 'New' condition will be assigned a value of 1, and those with 'Used' a value of 0, for converting categorical data into numeric. After the data was analyzed, it was discovered that every make corresponds to a specific model. Thus, to eliminate columns, the make and model will be merged, and individual columns will be dropped to reduce the number of features.

```
adv['make/model'] = adv['make'] +' / '+ adv['model']
```

This is how the new dataset now appears.

public_reference	make/model	body	car_van	colour	fuel	condition	mileage	reg	year	age	price
202006039777689	Volvo / XC90	SUV	False	Grey	Petrol Plug-in Hybrid	1	0	20	2020	4	73970

2.2 Categorical Encoding

One of the most crucial steps in the development of a machine-learning model is this one. In this process, features that are of the string type are encoded. The string values must be converted into the proper numeric values because string data cannot be accepted as input by machine learning algorithms. Two types of encoding are used to convert string data type into numeric data type:

1. **One-Hot Encoding**
2. **Target Encoding**

Columns such as make/model, colour, and body possess too many unique values. Should only One-hot Encoding be used, there would be too many features, and the model would become complex.

A variable for one-hot encoding is created, and pd.get_dummies is used for encoding.

```
ohe_cat_cols = ['fuel']
```

	fuel_Bi Fuel	fuel_Diesel	fuel_Diesel Hybrid	fuel_Diesel Plug-in Hybrid	fuel_Electric	fuel_Petrol	fuel_Petrol Hybrid	fuel_Petrol Plug-in Hybrid
0	0	0	0	0	0	0	0	1

One-Hot Encoding has been successful for the above column, and the next step is to encode the remaining columns with target encoding.

Target encoding, unlike one-hot encoding, can potentially leak data if not handled correctly, especially when information from the target variable is used to encode features. In target encoding, a categorical value is replaced with a number calculated from the average of the target variable for that category. This technique, useful for categorical features with high cardinality, must be implemented carefully to avoid overfitting and data leakage.

```
for col in ['make/model', 'body', 'colour']:
    target_means = adv.groupby(col)['price'].mean()
    adv[col + '_encoded'] = adv[col].map(target_means)
```

After the categorical columns have been encoded, both the variables of one-hot encoding and target encoding will be merged.

```
adv_encoded = pd.concat([adv.drop(ohe_cat_cols + ['make/model', 'body','colour', 'condition'], axis=1),
                         one_hot_encoded_df], axis=1)
```

After the encodings have been merged, the new dataset is shaped into 356566 rows and 19 columns.

2.3 Polynomial and Interaction Features

2.3.1 Splitting Data (Small Sample)

Just for the experiment with polynomial features, I have used 10,000 rows of cleaned datasets.

```
adv_pf = adv_encoded.sample(n=10000, random_state=1)
adv_pf.reset_index(drop=True, inplace=True)
```

Here, two variables, X and y, are being created, with all columns being assigned to the variable X except for the price column. The variable y will be assigned the target column, i.e., price.

```
X = adv_pf.drop('price', axis=1)
y = adv_pf['price']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
print('Shape of sample training data:', X_train.shape)
print('Shape of sample test data:', X_test.shape)

Shape of sample training data: (8000, 18)
Shape of sample test data: (2000, 18)
```

2.3.2 Polynomial Regression

The input features are first transformed into polynomial features that include only interactions between different variables by the poly_reg model, and then linear regression is applied.

```
poly_reg = Pipeline(steps=[('poly', PolynomialFeatures(interaction_only=True)),('est', LinearRegression())])
```

```
poly_reg.fit(X_train, y_train)
```

After the model has been trained on the training dataset, it will be tested using the predict() function, and the r2 score, mean absolute error, and mean squared error will be calculated. A higher r2 score indicates the most suitable model.

```
r2 score:  88.19265306453109 %
mean absolute error:  2973.5389982678353
mean squared error:  5191.5953433426985
```

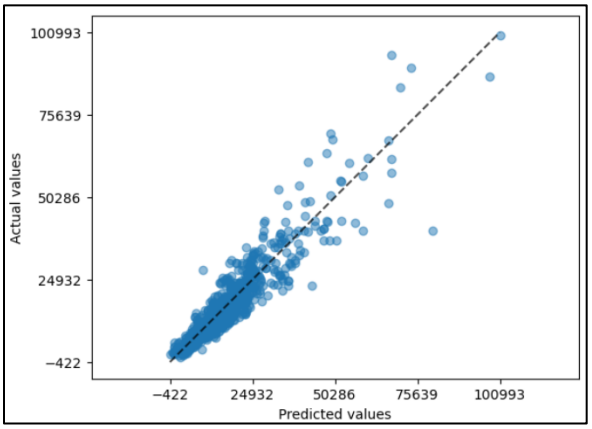

Fig. 2 Polynormal Features Accuracy and True vs Predicted

An accuracy of 88.19% is achieved by the model on the test data, which is considered good. A mean absolute error (MAE) of 2973.53 is indicative of the average magnitude of errors in a set of predictions, without consideration of their direction (positive or negative). This means that the predictions made by the model are, on average, off by 2973.53 units from the actual values. The closer the MAE is to zero, the closer the predictions are to the actual outcomes, indicating better model accuracy. A mean squared error (MSE) of 5191.59 is achieved when the deviations of the predicted values from the actual data points are squared and averaged, and the result is 5191.59. A lower MSE value suggests a closer fit of the model to the data, indicating better prediction accuracy. The MAE and MSE suggest that there might be room for improvement, especially in reducing the average error and the variance of errors in predictions. For a better understanding, the actual vs predicted value can be checked (scatter plot on the right side). The predicted points in blue are mostly clustered along the actual diagonal line, indicating that for many predictions, the model performs reasonably well, as the predicted values are close to the actual prices. Several predictions, especially in the higher price range, show the model underestimating the vehicle prices. Also noticeable are a few instances where negative values are predicted by the model, which are not feasible in the context of vehicle prices. This suggests that polynomial features are negatively impacting model performance, not by accuracy, but by predicting negative prices.

3. Feature Selection and Dimensionality Reduction

Feature selection and dimensionality reduction are considered essential processes in machine learning, playing a crucial role in the development of efficient and effective models. These processes are known to enhance model performance, decrease overfitting, speed up model training, reduce model complexity, and improve visualization.

3.1 Manual Feature Selection

Manual feature selection is performed based on domain knowledge, data exploration, etc.

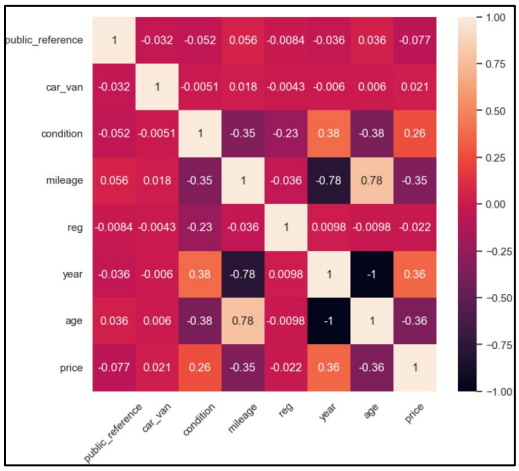

Fig. 3 Heatmap of Correlation metrics

Based on data exploration and correlation metrics, public reference and crossover_car_and_van are not considered important features. Therefore, they will be dropped to decrease model complexity. A feature called age has already been engineered, allowing for the year column to be dropped. This is because age captures similar information in a potentially more directly relevant manner.

3.2 Automated Feature Selection

There are many methods available for automated feature selection. Recursive Feature Elimination with Cross-Validation (RFECV) is utilized as a powerful technique to enhance model performance by systematically selecting the most important features from a dataset.
To begin, a sample dataset of 10,000 rows was created. This dataset was then split into training and testing subsets.

```
Shape of sample training data: (8000, 15)
Shape of sample test data: (2000, 15)
```

A simple linear algorithm is being used to train and test. ‘Step =1’ is used, given that only 15 columns are being dealt with. 'Step' defines the number of features to remove at each iteration. The number of folds in a k-fold cross-validation is specified by 'CV'. CV=5 is used, meaning that the dataset is split into 5 folds, and for each fold, the model is trained on 4 folds and validated on the 5th.

```
model = LinearRegression()
ref_selector_lf = RFECV(model, step=1, cv=5)
ref_selector_lf.fit(X_lf, y_lf)
```

```
r2 score:  87.33337966177005 %
mean absolute error:  3461.1146144243357
mean squared error:  6342.341341583058
```

An R2 score of 87.33% indicates that the model performs well on the test data. Below, the important features contributing most to higher accuracy are listed.

```
1 print("Features extended dataset: ", ref_selector_lf.get_feature_names_out())

Features extended dataset:  ['mileage' 'reg' 'age' 'make/model_encoded' 'condition' 'fuel_Bi Fuel'
 'fuel_Diesel' 'fuel_Diesel Hybrid' 'fuel_Diesel Plug-in Hybrid'
 'fuel_Electric' 'fuel_Petrol' 'fuel_Petrol Hybrid'
 'fuel_Petrol Plug-in Hybrid']
```

```
1 X_train_lf.keys()

Index(['mileage', 'reg', 'age', 'make/model_encoded', 'body_encoded',
       'colour_encoded', 'condition', 'fuel_Bi Fuel', 'fuel_Diesel',
       'fuel_Diesel Hybrid', 'fuel_Diesel Plug-in Hybrid', 'fuel_Electric',
       'fuel_Petrol', 'fuel_Petrol Hybrid', 'fuel_Petrol Plug-in Hybrid'],
      dtype='object')
```

As per RFECV, the important features contributing to the accuracy are shown above in the left-side image. The features that were given for training are displayed in the right-side image. It is suggested by Recursive Feature Elimination that body_encoded and colour_encoded can be dropped, as these features will not significantly affect model performance and will reduce model complexity. Below, the visualization of the mean test score with feature elimination is presented.

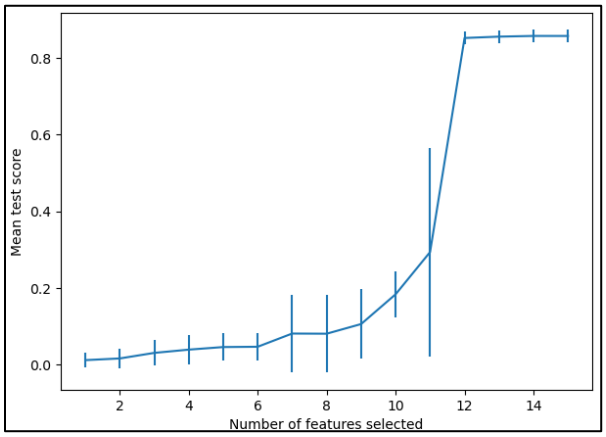

Fig. 4 RFECV Performance

From the graph, it is observed that the mean test score increases after 11 features, with a notable jump in performance from 11 to 12 features. However, no significant jump is seen from 12 to 15 features. This observation indicates that additional features beyond that point do not contribute significantly to the model's predictive power; it suggests that a set of 12 or 13 features provides a good balance between performance and model complexity. For model building, these columns (body_encoded and colour_encoded) will be dropped to decrease complexity.

4. Model Building

This step is considered one of the crucial steps in Machine Learning. The selection of the most appropriate algorithm to maximize the accuracy of the output for the task is determined here. The data will be trained and tested on four different models: a Linear model, a Random Forest, a Gradient-boosting regressor, and a Voting Regressor. A Voting Regressor, utilized specifically for regression tasks, is an ensemble machine-learning model. It functions by aggregating the predictions from multiple models, resulting in a final prediction that is usually more robust and accurate than that of any single model in the ensemble.

4.1 A Linear Model

A linear regression will be performed by fitting the X_train and y_train.

```
%%time
# Linear Regressor
lr = LinearRegression()
lr.fit(X_train, y_train)
```

After the model has been trained on the training dataset, it will now be tested, and calculations for the r2 score, mean absolute error, and mean squared error will be conducted. A higher r2 score denotes the model's suitability.

```
lr_y_pred = lr.predict(X_test)
lr_r2_Score = r2_score(y_test, lr_y_pred)*100
lr_mean_abs_err = mean_absolute_error(y_test, lr_y_pred)
lr_mean_sqr_err = np.sqrt(mean_squared_error(y_test, lr_y_pred))
```

```
r2 score:  86.57457414299401 %
mean absolute error:  3358.9964786247583
mean squared error:  5613.807843983003
```

The performance of the linear model is considered good based on the r2 score, yet the mean absolute error and mean squared error are found to be excessively high for a price prediction task. It is indicated by the MAE that, on average, the predictions made by the model deviate by 3358.99 units from the actual values.

4.2 Random Forest

A Random Forest regression will be performed by fitting the train data of X and y.

```
%%time
rf = RandomForestRegressor(random_state=7)
rf.fit(X_train, y_train)
```

After the model has been trained on the training dataset, it will now be tested by me, and the r2 score, mean absolute error, and mean squared error will be calculated. A higher r2 score indicates a more suitable model.

```
rf_y_pred = rf.predict(X_test)
rf_r2_Score = r2_score(y_test, rf_y_pred)*100
rf_mean_abs_err = mean_absolute_error(y_test, rf_y_pred)
rf_mean_sqr_err = np.sqrt(mean_squared_error(y_test, rf_y_pred))
```

```
r2 score:  95.71018325242817 %
mean absolute error:  1731.5972300814149
mean squared error:  3173.312759392183
```

An R^2 score of 95.71% signifies high accuracy, which is generally considered a strong performance indicator for any regression model, showing that it fits the data well. The MAE value of 1731.59 units reveals that, on average, the deviations of the model's predictions from actual prices are much lower compared to a linear model.

4.3 Gradient Boosted Tree

A gradient-boosted tree regression will be performed by fitting the X_train and y_train.

```
%%time
gb = GradientBoostingRegressor()
gb.fit(X_train, y_train)
```

After the model has been trained on the training dataset, it will now be tested by me, and the r2 score, mean absolute error, and mean squared error will be calculated. A higher r2 score indicates a more suitable model.

```
gb_y_pred = gb.predict(X_test)
gb_r2_Score = r2_score(y_test, gb_y_pred)*100
gb_mean_abs_err = mean_absolute_error(y_test, gb_y_pred)
gb_mean_sqr_err = np.sqrt(mean_squared_error(y_test, gb_y_pred))
```

```
r2 score:  91.94439343141565 %
mean absolute error:  2606.843236509099
mean squared error:  4348.531610830453
```

The gradient-boosted tree model is also performing well compared to the linear model based on the r2 score, mean absolute error, and mean squared error. However, the Random Forest model is predicting prices much more accurately compared to the gradient-boosted tree.

4.4 An Averager/Voter/Stacker Ensemble

An Average, Voter, or Stacker Ensemble is described as a collection of ensemble machine learning techniques that aggregate the outputs from several models to formulate predictions. In this instance, the outputs of three models are being utilized by me to create an ensemble. This approach is encompassed within a wider machine learning strategy known as ensemble learning, which enhances prediction accuracy and robustness by merging the results from multiple models instead of depending on a singular model.

A new model, termed the ensemble, is being created, incorporating my three algorithms: linear, random forest, and gradient booster. This model will be trained and tested by me.

```
from sklearn.ensemble import VotingRegressor
ensemble = VotingRegressor(
    [("gb", gb), ("rf", rf), ('lr', lr)
    ])
ensemble.fit(X_train, y_train)
```

```
r2 score:  93.59939757438782 %
mean absolute error:  2300.5438473520103
mean squared error:  3876.179975570488
```

Compared to the linear and gradient booster models, the ensemble's r2 score is superior, and both the mean absolute error and mean squared error are lower.

4.5 Model Ranking

4.5.1 Performance Analysis and Comparison

The performance of my models will be ranked based on accuracy, mean absolute error, and mean squared error.

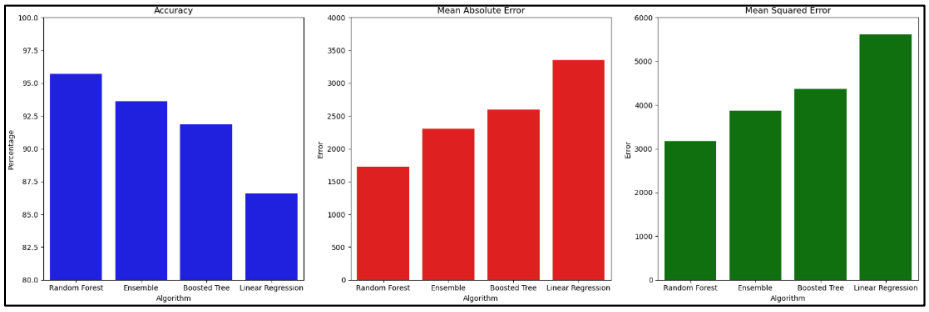

Fig. 5 Performance Comparison

The chart indicates that the Random Forest model excels above the others in all metrics, achieving the highest accuracy and registering the lowest MAE and MSE, which suggests it delivers more precise predictions with fewer large errors. The Ensemble and Boosted Tree methods demonstrate competitive performance but are slightly less accurate, possessing almost similar MAE values that are lower than those observed in Linear Regression. Conversely, Linear Regression falls behind in all metrics, exhibiting the lowest accuracy and the highest values in terms of MAE and MSE, which points to its greater and more frequent prediction errors compared to the other models.

4.5.2 Underfitting/Overfitting Analysis

Underfitting is observed when a model is overly simplistic, lacking the complexity needed to discern crucial patterns within the data, which results in poor performance on both the training and new data. Overfitting occurs when a model is excessively complex, adhering too closely to the training data and encompassing noise and outliers, thereby impairing its performance on new, unseen data. These conditions inhibit the models' ability to generalize effectively, leading to suboptimal predictive performance.

	Algorithm	Accuracy_test	Mean Absolute Error_test	Mean Squared Error_test	Accuracy_train	Mean Absolute Error_train	Mean Squared Error_train
0	Random Forest	95.71	1731.60	3173.31	99.09	756.64	1697.21
1	Ensemble	93.60	2300.54	3876.18	95.90	2071.81	3594.44
2	Boosted Tree	91.94	2606.84	4348.53	94.06	2598.05	4324.41
3	Linear Regression	86.57	3358.97	5613.83	88.86	3371.66	5923.31

The Random Forest model emerges as the superior choice, distinguished by its excellent accuracy and error metrics across both training and test evaluations. This model records the highest test accuracy at 95.71%, not only demonstrating its capability to effectively learn from the training data but also its robustness in generalizing to new, unseen data, as reflected by a high training accuracy of 99.09%. However, an indication of overfitting in the Random Forest is observed, evidenced by the decrease in test scores compared to training scores. Although some decline in performance is typical, the extent noted here suggests significant overfitting.

On the other hand, the Boosted Tree model exhibits remarkable consistency between its training and testing performances, signaling strong generalization capabilities and minimal overfitting compared to the Random Forest. It achieves a training accuracy of 94.06% and a closely matching testing accuracy of 91.94%, indicating effective adaptation to both known and new data without considerable performance drop. Nonetheless, its mean absolute error and mean squared error are higher compared to the Random Forest, which still positions the Random Forest as a more appropriate model for the price prediction task, despite its overfitting concerns.

5. Model Evaluation and Analysis

5.1 Overall Performance with Cross-Validation

	test_mae_mean	test_mae_std	train_mae_mean	train_mae_std
Linear	3365.60	2.96	3365.55	1.33
Random Forest	1745.22	3.21	756.86	3.63
Boosted Tree	2596.37	7.74	2583.00	9.29
Ensemble	2303.90	5.61	2065.10	2.02

The outcomes from the cross-validation process for all the models that were trained are summarized in the table. Among these, the Random Forest model was identified as the top performer. A lowest mean MAE of 1745.22 was recorded on the test data by this model, which clearly demonstrated its strong predictive accuracy. Additionally, with a mean MAE on the training data maintained at a comparably low 756.86, it demonstrated a solid fit to the training data and managed to avoid overfitting. These factors have established the Random Forest model as the most effective and dependable among those tested by me, making it particularly suitable for applications where robust performance on new, unseen data is crucial.

5.2 True vs Predicted

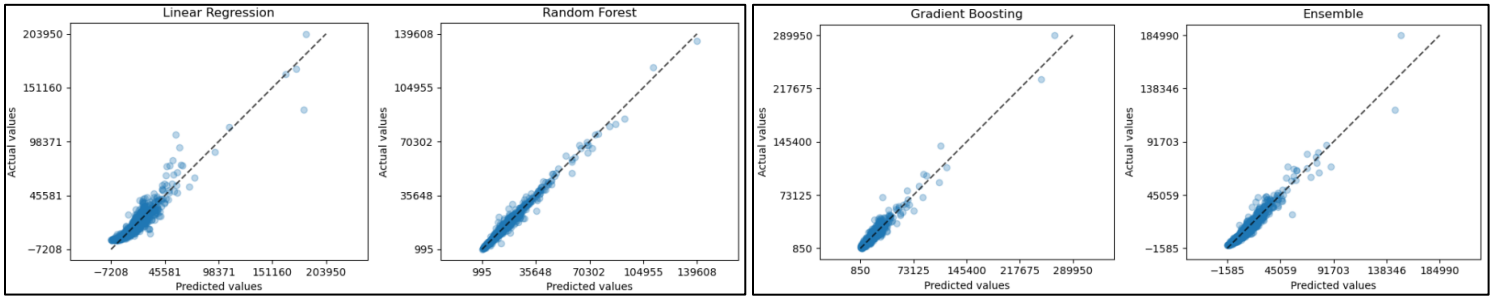

Fig. 6 Actual vs Predicted

The comparison of actual car prices with the values predicted by four different models is illustrated in the scatter plots provided. In each plot, a closer alignment of data points along the dotted line is indicative of better prediction accuracy. The Linear Regression model is shown to have a reasonable correlation between predicted and actual values, though notable variance is observed, especially at higher price points, and negative prices are also predicted, which is not feasible. A significant improvement is noted in the Random Forest model, with data points more tightly clustered along the line, suggesting higher accuracy and reduced prediction errors. A similar trend is observed in the gradient-boosting plot, indicating even more precise predictions with fewer outliers. The Ensemble model, combining all three models, also demonstrates high accuracy but includes predictions of negative values, possibly influenced by the linear model since the ensemble is an average of the three different models. Below, the visualization of 15 predictions is displayed, illustrating how much closer each model comes to predicting the true values of the prices.

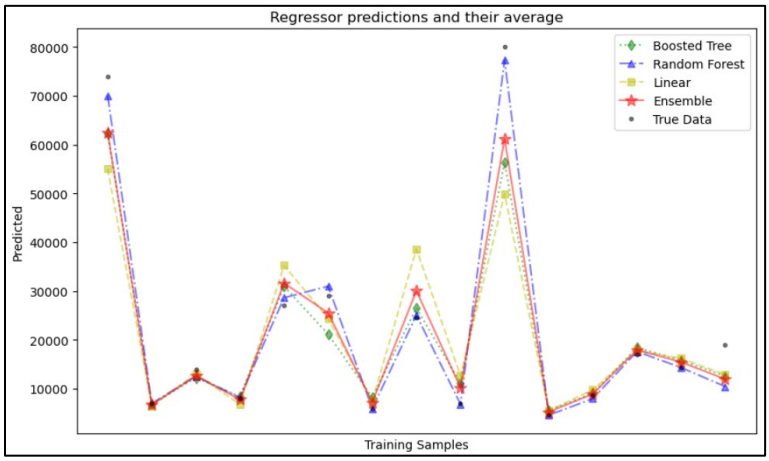

Fig. 7 True vs Predicted

Different model predictions are represented on the graph using various shapes and colours, each model striving to predict the actual price of a car based on several features. Black dots, which represent the true prices for the first 15 predictions, serve as the benchmarks that are aimed to be matched. Predictions made by an ensemble of all three models are depicted using a red line with star shapes. The predictions from the Random Forest model are indicated by blue lines and triangle shapes. Predictions from the Linear model are marked with a yellow line and square shapes, while predictions from the Boosted Tree model are shown in green with diamond shapes. Among these models, the blue Random Forest line closely

aligns with the black dots, indicating its strong predictive accuracy, as also reflected in the mean absolute error measurements, and the ensemble also shows commendable performance relative to the linear and gradient-boosted tree models.

5.3 Global and Local Explanations with SHAP

SHAP is a tool that helps explain how machine learning models make their decisions, providing both broad and specific insights.

5.3.1 Local Explanation with SHAP

Local SHAP analysis is employed to elucidate the prediction for a single instance by calculating how much each feature contributes to that prediction. This method offers in-depth insights into the influence of various features on the model's output for a particular data point.

5.3.1.1 Linear SHAP Analysis

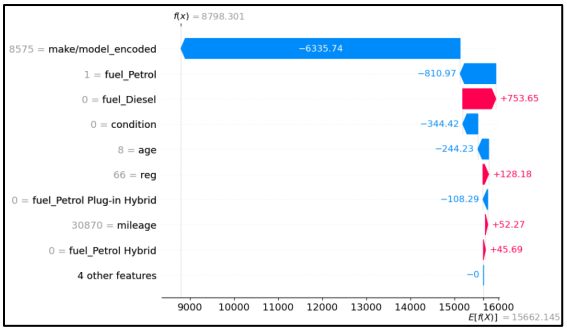

Fig. 8 Local SHAP of Linear

The SHAP analysis graph presented is a prediction analysis for index 1 and illustrates the influence of different car features on the estimated price according to our predictive model. It starts from a base prediction of 8,798.30, with various factors adjusting this base to arrive at the final predicted value of 15,662.15. The make or model of the car has the most significant impact, reducing the value by 6,335.74, which suggests that certain car types substantially lower the estimated price. In contrast, the presence of diesel fuel increases the car's value by 753.65, indicating a preference for diesel vehicles in this model. Factors such as the car's age and mileage also reduce the value, which reflects the typical wear and tear. This graph provides a visual summary of how each feature positively or negatively influences the car's price, clearly showing how different attributes contribute to the overall price estimation.

5.3.1.2 Random SHAP Analysis

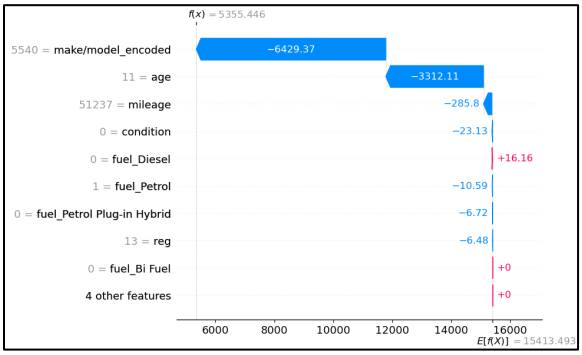

Fig. 9 Local SHAP of Random Forest

The SHAP analysis graph clearly shows that the most significant factors affecting the car's price prediction are its make or model and its age, both of which considerably reduce the value. Specifically, the make or model decreases the predicted price by 6,429.37, and the age of the car further reduces it by 3,312.11, underscoring how a car's value heavily depends on its make/model and age. Mileage also has a modest negative impact, decreasing the price by 285.8. Conversely, having a petrol engine slightly increases the car's price by 16.16. Other characteristics such as the car's condition, diesel fuel type, plug-in hybrid status, registration, and bi-fuel capability have minimal impacts on the final price estimation in this analysis.

5.3.1.3 Gradient Booster SHAP Analysis

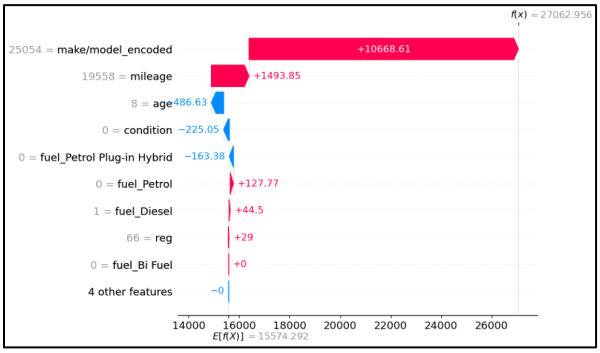

Fig. 10 Local SHAP of Gradient Booster

The SHAP analysis graph indicates that the feature make/model_encoded significantly increases the price by 10,668.61, highlighting the substantial value that specific car makes or models add. Interestingly, mileage also contributes positively, raising the predicted price by 1,493.85, which is atypical as higher mileage generally reduces a vehicle's value. This anomaly suggests the presence of unique characteristics or valuable classic cars in the dataset. On the downside, factors such as age, condition, and fuel_Petrol Plug-in Hybrid decrease the price by 486.63, 225.05, and 163.38, respectively. Collectively, these factors adjust the base prediction to the final predicted value of 15,574.292, demonstrating how premium makes/models and specific vehicle attributes can significantly influence car valuations in this model.

5.3.2 Global Explanation with SHAP

Global explanations are designed to elucidate the model's overall behavior, in contrast to local explanations which focus on individual predictions. They provide an overview of the most important features for the model across the entire dataset. Global explanations assist in understanding the model's decision-making on a broad scale and can be employed to verify if the model's logic corresponds with domain knowledge and intuition.

5.3.2.1 Linear SHAP Analysis

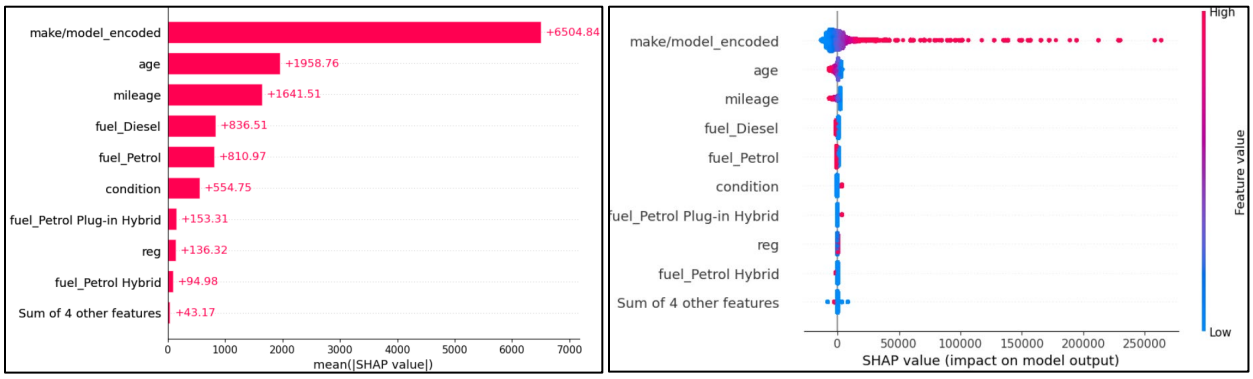

Fig. 11 Global Explanation of SHAP with Bar and Beeswarm of Linear

The SHAP graph of bar and beeswarm was used to show the distribution and impact of different features on a predictive model's output. Each dot on the beeswarm chart showed a SHAP value for an observation in the related feature. The most important feature, 'make/model_encoded', had a wide range of impact, with red indicating higher model outputs. Other key features were 'age' and 'mileage', where older and higher-mileage vehicles (shown in blue) usually lowered the model's predictions. Fuel types and vehicle conditions also had clear effects, with 'fuel_Diesel' and 'fuel_Petrol' making notable positive contributions. This plot made it clear which features were most important and how their high or low values affected the model's output, helping with decision-making.

5.3.2.2 Random Forest SHAP Analysis

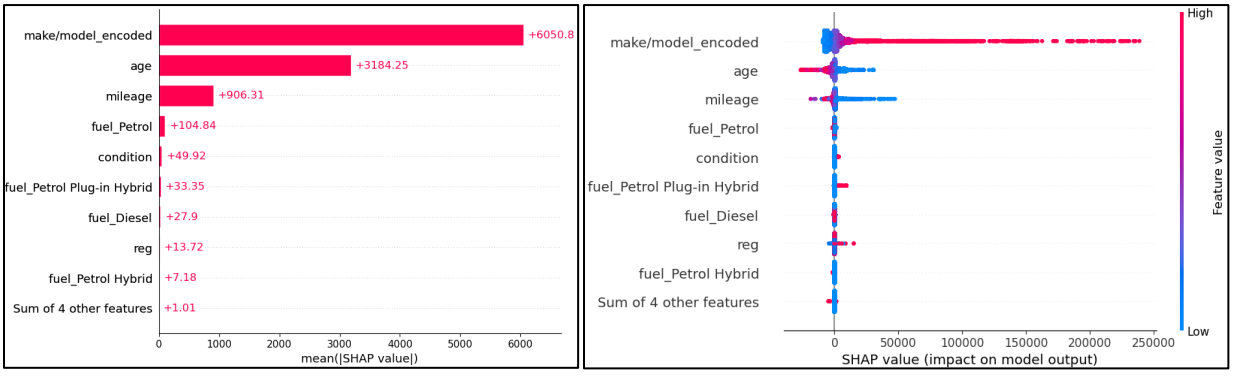

Fig. 12 Global Explanation of SHAP with Bar and Beeswarm of Random Forest

The bar chart was used to show that 'make/model_encoded' has the most significant average impact on the model's output, followed by 'age' and 'mileage'. This means that the make and model of a vehicle are the strongest predictors of the outcome being modelled, whether it be price, demand, or another metric. The second graph, a SHAP beeswarm, displayed the distribution of SHAP values for each feature. This plot used colour to show the value of the feature, with red indicating higher values and blue showing lower ones. It effectively highlighted the variance in the impact that different values of the 'make/model_encoded' feature have on the model's predictions, emphasizing the detailed role that vehicle characteristics play in the modelling process.

5.3.2.3 Gradient Booster SHAP Analysis

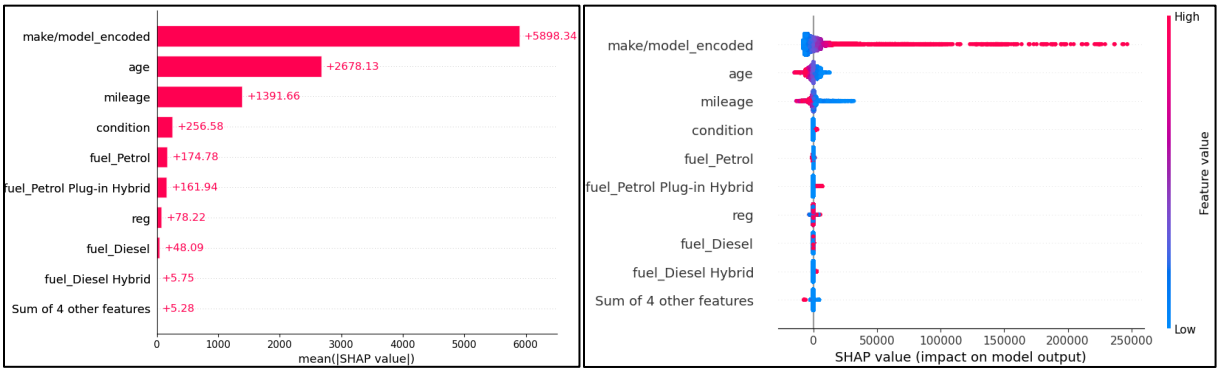

Fig. 13 Global Explanation of SHAP with Bar and Beeswarm of Gradient Booster

Similar to linear and random forest models, ‘make/model_encoded', 'age', and ‘mileage’ were identified as the top 3 influencers, significantly shaping the model's outputs. The beeswarm plot provided a deeper insight by illustrating how individual data points for each attribute were distributed and influenced model outcomes across their range of values. Notably, the plot revealed that higher values of 'make/model_encoded' tended to enhance the predicted outcomes, showcasing a consistent influence trend across different data points.

5.4 Partial Dependency Plots

A partial dependency plot (PDP) was used as a visualisation tool to show the effect of a specific feature on the predicted outcome of a model. It isolated the relationship between a selected feature and the target variable by averaging out the effects of all other features in the model. This was particularly useful when understanding the behaviour of a model concerning one or two features, regardless of what other features were doing.

5.4.1 Linear Model Partial Dependency

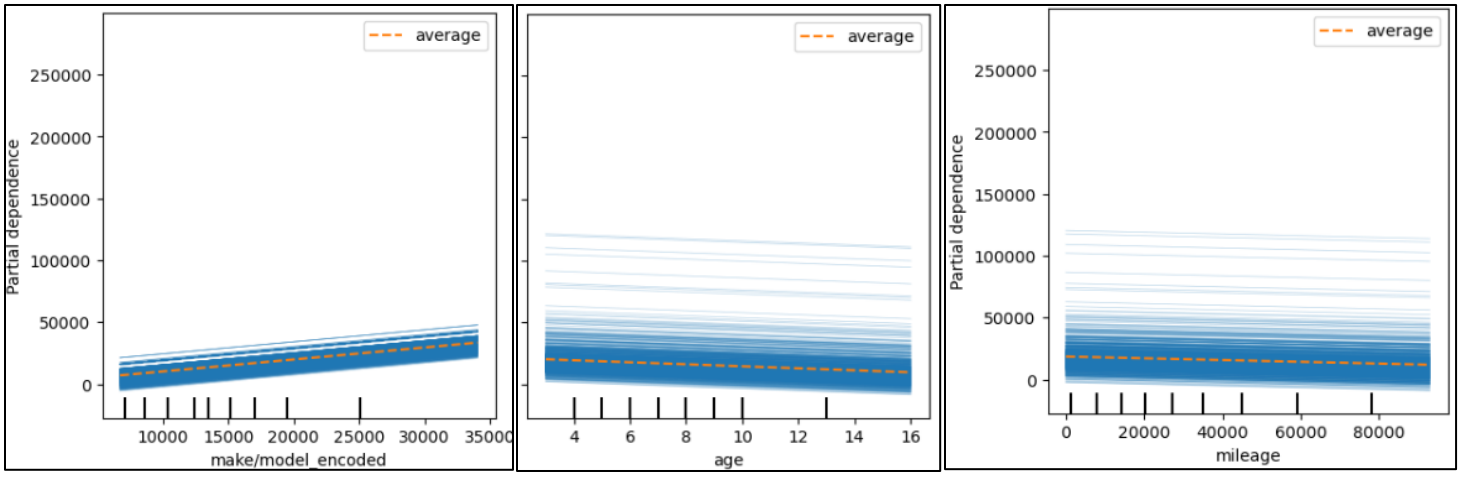

Fig. 14 PDP of Linear for make/model, age and mileage

Linear partial dependency plots of make/model, age, and mileage were used to show how they influence the predicted car prices in my linear model. The first plot, for the make/model of the car, indicated that the make/model has a relatively stable influence on the price, with slight variations suggesting some models might be slightly more or less valuable, but overall, the effect is subtle. The second plot illustrated how the age of a car affects its price, showing a clear trend where the price decreases as the car gets older, reflecting typical depreciation due to age. The third plot demonstrated the effect of mileage, which also showed a decrease in price as mileage increased, again reflecting depreciation due to wear and tear.

5.4.2 Random Forest Partial Dependency

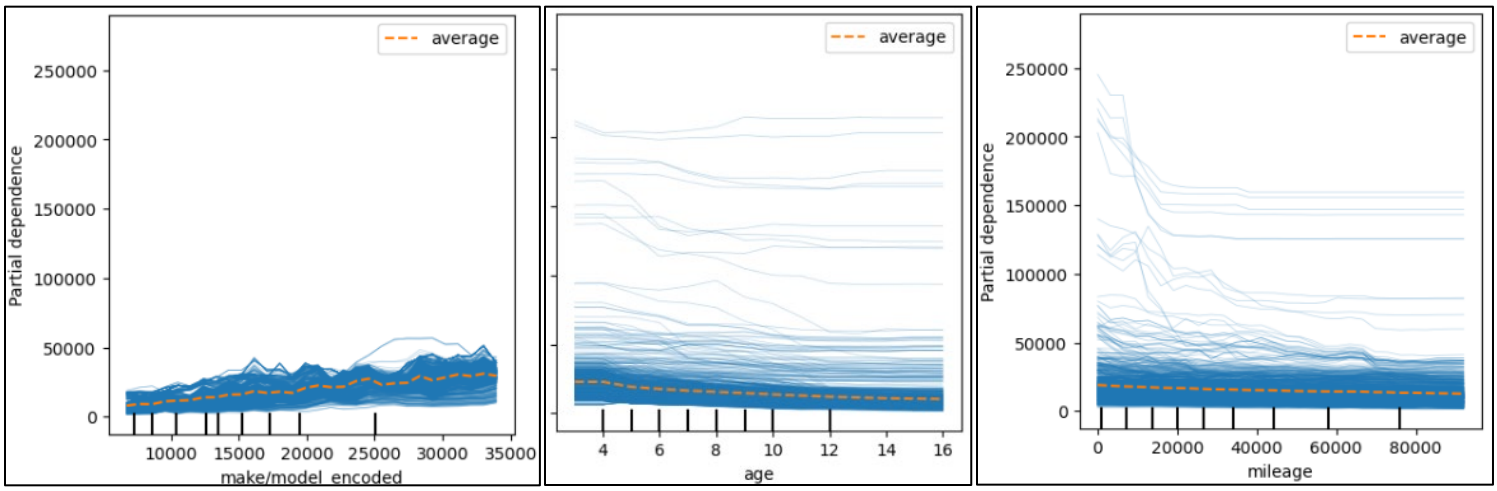

Fig.15 PDP of Random Forest for make/model, age and mileage

The make/model of the random forest model showed more variability than the linear model's plot. This suggested that this model picked up on nuances between different makes and models, indicating that some have a more pronounced effect on car prices than others. The second graph, focusing on age, revealed a gradual decline in car prices as age increased, consistent with general expectations about cars depreciating over time. However, this model highlighted a more nuanced depreciation pattern than the linear model, where the drop wasn't as pronounced. The third graph on mileage also indicated a decrease in car prices as mileage increased, showing a sharper drop in value initially, which then stabilized. This suggested that early mileage accumulation had a significant impact on value, a detail that the linear model's smoother curve might miss. These graphs made sense as they reflected more complex interactions in the data that a random forest model can capture compared to a linear model, providing deeper insights into how these features impact car pricing.

5.4.3 Gradient Booster Partial Dependency

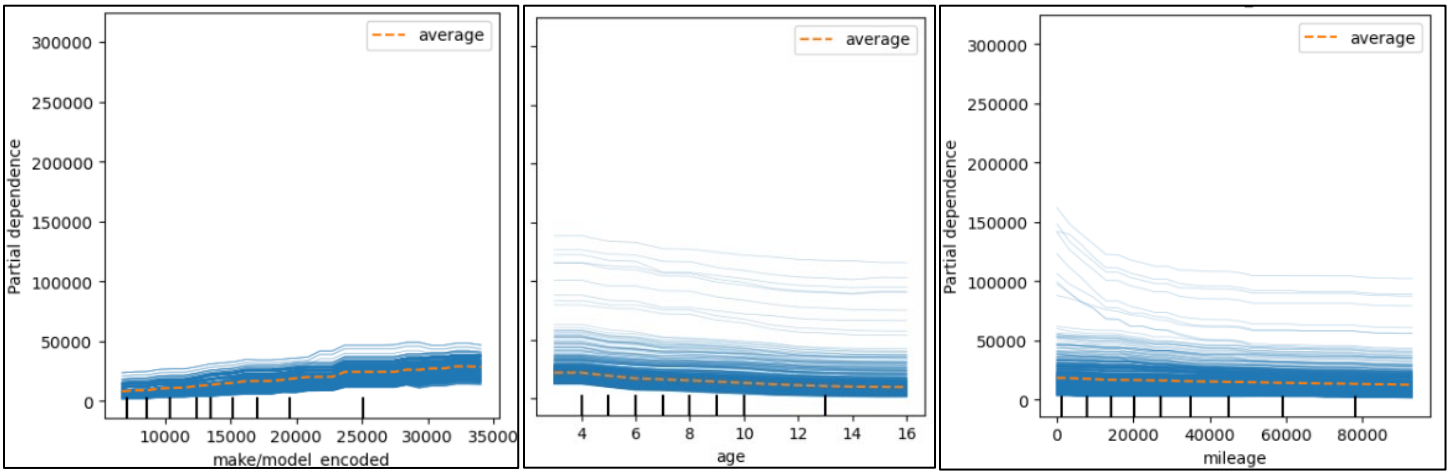

Fig.15 PDP of Gradient Booster for make/model, age and mileage

The Gradient Boosted Tree model's make/model graph displayed a consistent influence across different categories, with slight peaks indicating subtle differences in how certain makes/models impact the price. This was more stable compared to the more variable Random Forest model and more nuanced than the flat representation in the linear model. The age graph showed a steady decrease in price as cars aged, capturing a smooth, continuous depreciation that aligns well with typical car valuation trends where older cars are generally worth less. This depreciation curve was more detailed than the linear model, though less sharply defined than the Random Forest's, suggesting a balanced middle ground. The mileage graph presented a clear trend of decreasing price with increasing mileage, emphasizing initial sharp depreciation which then leveled off. This pattern was effectively captured by the Gradient Boosted Trees, possibly better than the linear model's simpler approach and similar to the Random Forest's insights but with less noise.