



Department of Computing and Mathematics

Advanced Machine Learning

6G7V0017

*MSC DATA SCIENCE*

*ASSESSMENT*

KOLAWOLE MARY BIOLA | 23659564 | MAY 2024

## Data Processing for Machine Learning

### Identification and Handling of Incorrect, missing values and outliers

Upon initial analysis of the dataset, several incorrect and missing values, as well as some potential outliers, were found. The year of registration had some incorrect values ranging from the early 1990s to an unrealistic value of 999. Considering that the first car was registered in 1904, values predating this period were treated as erroneous. To address this, Cars earlier than 1904 were marked as null values and subsequently replaced with the mean of the year of registration while replacing the missing value. Incorrect values were also discovered in the mileage column, where the mileage for used cars was zero. To address this, the columns were marked null and replaced with mean mileage calculated from year and vehicle condition, as shown in Figure 1.

```
#replacing mileage with null vaules
autos.loc[(autos['mileage'] == 0.0) & (autos['vehicle_condition'] == 'USED'), 'mileage'] = pd.NA

autos.loc[autos['year_of_reg'] < 1904, 'year_of_reg'] = pd.NA
```

Figure 1. The code implementation for replacing the incorrect values for year and mileage with null values.

The data set was examined for missing values, and six columns within the dataset were found to have some missing values. The imputation method was used to replace the missing values, a crucial step to ensure the dataset is accurately represented. The numeric features were replaced using the mean, as illustrated in Figure 2. The categorical features were replaced using the simple imputer from the pre-processed pipeline design to handle both numerical and

```
autos['year_of_reg'].fillna(autos['year_of_reg'].mode()[0], inplace=True)
# Calculate the mean mileage by grouping by year of reg and vehicle condition
mean_mileage_by_group = autos.groupby(['year_of_reg', 'vehicle_condition'])['mileage'].transform('mean')
# Replace the missing values for mileage with the mean mileage
autos['mileage'].fillna(mean_mileage_by_group, inplace=True)
```

categorical values.

Figure 2. The Code snippet for replacing the year of registration and mileage column.

Potential outliers in price and mileage were identified and adjusted using the percentile method. Any data points beyond the 95th percentile were capped to minimise their impact, as shown in Figure 3.

```
# Calculate the 95th percentile for both 'mileage' and 'price'
percentile_95_mileage = np.percentile(autos['mileage'], 95)
percentile_95_price = np.percentile(autos['price'], 95)

# Filter the DataFrame to include values within the 95th for both mileage, price
autos = autos[
    (autos['mileage'] <= percentile_95_mileage) &
    (autos['price'] <= percentile_95_price)]
```

Figure 3. The code Implementation for removing outliers from the mileage and price features using percentile.

### Subset and Train/Test Split

A random subset of 40% of the dataset was used to build the model and speed up iteration for model development. As shown in Figure 4, the dataset was split into training and testing sets in an 80-20 ratio.

```
#subset dataset, starting with 40% of the dataset
#sample the data with frac=0.4
autos = autos.sample(frac=0.4, random_state=42)

## the X/y split
X = autos.drop(columns='price')
y = autos['price']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Figure 4. The code snippet for sub-setting and splitting the dataset into train and test.

Preprocessing Pipeline

A Preprocessing pipeline was built to prepare the dataset for machine learning modelling. The pipeline is designed to handle both numeric and categorical features. The process involves handling missing values, which was implemented to replace the missing categorical features with the most frequent, scale numerical features, encode features, and optionally add polynomial features to capture non-linear relationships. The categorical features were encoded using a target encoder, which transformed each category into the mean value of the target for that feature. A target encoder was used to help handle high cardinality in features like make and model. When enabled, the numeric features were scaled using MinMaxScaler, which transforms the data to fall within the range of 0 and 1.

Feature Engineering and Polynomial Features

For feature engineering, car age was derived from the year of registration by subtracting the year from the current year as of when this report was written. This new continuous variable captures the depreciation factors more directly than the year of registration. Most cars in this dataset are less than ten years old, emphasising age's significance in determining a vehicle's market value. This highlights age as a significant predictor of car price.

```
from datetime import datetime

current_year = datetime.now().year
autos['age'] = current_year - autos['year_of_reg'].astype(int)
```

Figure 5. The code implementation for deriving age from the year of registration.

To enhance the accuracy of the linear regression model, polynomial features and interaction terms were employed to capture the non-linear correlation between the features and the target variable. The two-degree polynomial transformation was applied, and it derived three features, age squared, mileage squared, and age mileage, from the joint interaction of the two features. Upon cross-validation and assessing the model performance using the R squared and mean absolute error MAE metrics, the linear regressor model with polynomial features slightly outperformed the normal linear regression model, as shown in Figure 6. Despite the slight difference in performance, using polynomial features led to enhanced predicted accuracy. Based on these findings, polynomial features will be included in the linear regression model for future analysis.

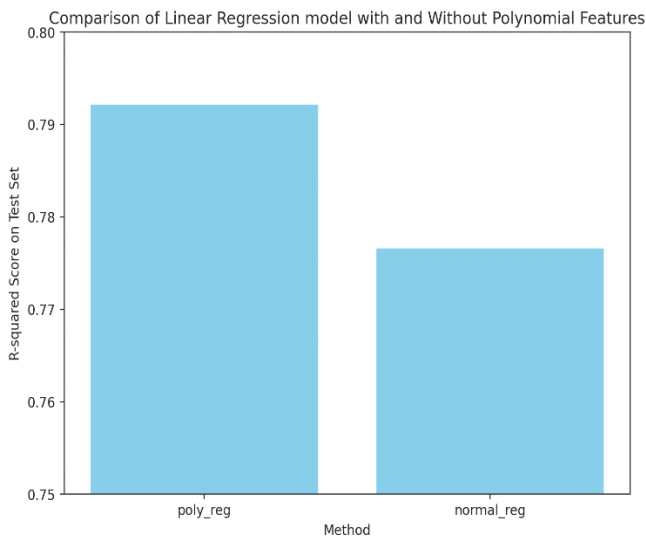


Table 1.The MAE Scores for with and without polynomial on the linear regression model

Model	Test MAE	Train MAE
Normal Linear Regression	3038.52	3029.94
Polynomial with Linear Regression	2910.04	2901.22

Figure 6.Comparison of Linear Regression with and without Polynomial Features.

## Feature Selection and Dimensionality Reduction

The Registration code and public reference were dropped based on domain knowledge and their relationship to other features, as shown in Figure 4. During exploratory analysis, it was observed that the reg code strongly correlated with age, indicating a car's age. Thus, including both features could result in multicollinearity. Public reference serves as a unique identifier for each vehicle but does not provide any predictive value to the model. Therefore, it was deemed unnecessary to consider these features in this analysis.

```
# Drop columns
columns_to_drop = ['public_ref', 'reg_code']
autos= autos.drop(columns=columns_to_drop)
```

Figure 7. The code Snippet for dropping the feature Reg code and public reference.

### Automated Feature selection

The three automated Feature selections were employed to determine the most predictive features for the linear regression model. The tree-based models were not utilised due to their ability to capture non-linear relationships and interactions between features. The select K best method was initially used where the  $k = 5$ . The RFECV was then applied, which led to the retention of all features indicative of their collective importance in the linear regression model. The RFECV plot in Figure 8 indicates that the model's performance improves by adding subset features up to the 8<sup>th</sup> feature, after which it stabilises, indicating that additional features added beyond the 8<sup>th</sup> feature did not significantly contribute to the model's performance. The sequential feature identified six key features, make, mileage, mileage squared, vehicle condition and model, that collectively provide the best predictive power for the model, but the R\_squared score obtained after cross-validation was low compared to that of the RFECV method, as shown in Figure 9. Therefore, all the features will be applied to the linear model moving forward in this analysis.

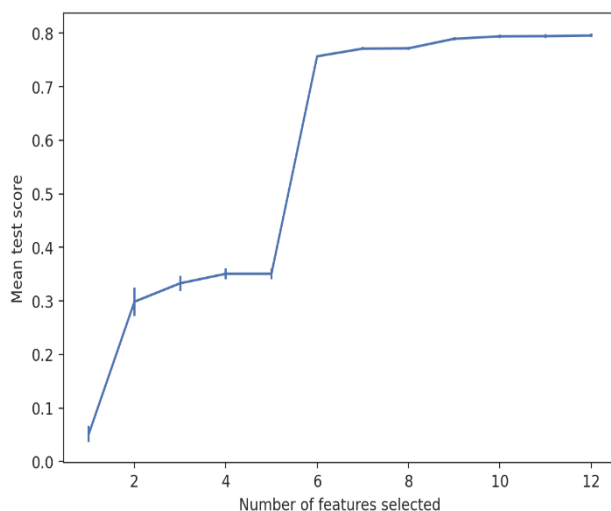


Figure 8. The RFECV Plot

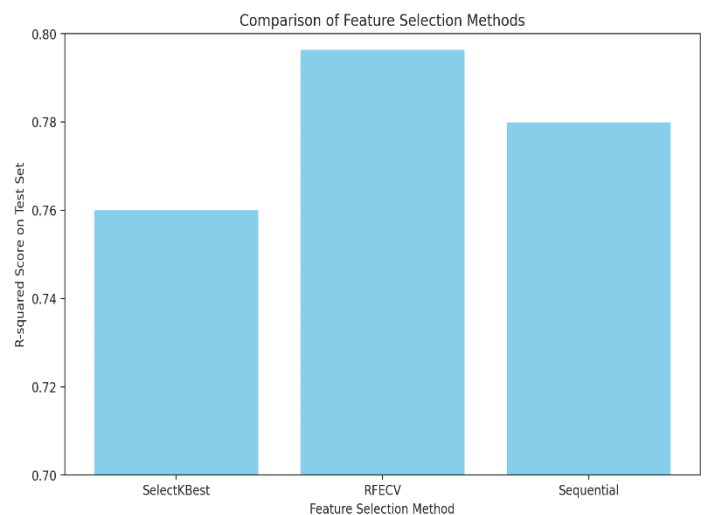


Figure 9. The Comparison of the Feature Selection Methods

### Principle Component Analysis

The Principal component analysis (PCA) reduces the dimensionality of a large dataset. It was employed on the linear regression model using components 2, 6, and full. The sixth component had the lowest MAE score on the linear regression model but was not as good as the linear regression with polynomial features without PCA. Therefore, Given that the polynomial regression alone yields a better MAE result than the PCA, the PCA doesn't contribute to my model performance; hence, focusing on polynomial features without PCA would be the best choice moving forward.

## Model Building

The regression function was applied because the target price is a continuous variable using algorithms like linear Regression, Gradient boosting Regression, and Random Forest tree Regression. A Grid search using cross-validation (CV=5) was used to determine which hyperparameter setting provides the best results for the Random Forest tree and Gradient-boosting Regressors. The regressor is then re-initialized with the optimal hyperparameters and trained on the dataset. Linear Regression does not use a hyperparameter, so it was not utilised. The models were trained with the training set, while the test set was used to evaluate the final model.

### Random Forest Tree Model

```
# Random Forest
rf_pipeline = create_est_ppln(RandomForestRegressor(), X_train, scaling=False, poly=False)

# Define your parameter grid for GridSearchCV
param_grid_rfr = {
    'est__max_depth': [3, 4, 8],
    'est__min_samples_leaf': [10, 20, 30],
}

grid_search_rfr = GridSearchCV(rf_pipeline, param_grid_rfr, cv=5, return_train_score=True)
grid_search_rfr.fit(X_train, y_train)
print("Best parameters found for RandomForestRegressor: ", grid_search_rfr.best_params_)

Best parameters found for RandomForestRegressor: {'est__max_depth': 8, 'est__min_samples_leaf': 10}

results_rfr = grid_search_rfr.cv_results_
gs_df_rfr = pd.DataFrame(results_rfr)
gs_df_rfr['rank_test_score'] = gs_df_rfr['rank_test_score'].astype(int)
gs_df_relevant_rfr = gs_df_rfr.loc[:, [
    'param_est__max_depth', 'param_est__min_samples_leaf',
    'mean_train_score', 'std_train_score',
    'mean_test_score', 'std_test_score', 'rank_test_score'
]]
# Print the sorted grid search results
gs_df_relevant_rfr
```

```
best_max_depth_rfr = grid_search_rfr.best_params_['est__max_depth']
best_min_samples_leaf_rfr = grid_search_rfr.best_params_['est__min_samples_leaf']

# the RandomForestRegressor with the best parameters
best_rfr = RandomForestRegressor(
    max_depth=best_max_depth_rfr,
    min_samples_leaf=best_min_samples_leaf_rfr
)

# Creating a new pipeline with this regressor
best_rfr_pipe = create_est_ppln(best_rfr, X_train, scaling=False, poly=False)

# Fit the pipeline with the best estimator on the training data
best_rfr_pipe.fit(X_train, y_train)
```

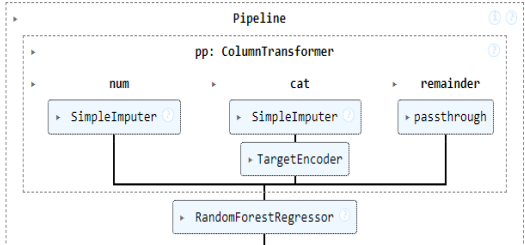


Figure 10. The Code Implementation for Grid Search on Random Forest Model

### Gradient Boosting Model

```
# Gradient Boosting
gb_pipeline = create_est_ppln(GradientBoostingRegressor(), X_train, scaling=False, poly=False)

# Define your parameter grid for GridSearchCV
param_grid_gb = {
    'est__max_depth': [3, 4, 6],
}

grid_search_gb = GridSearchCV(gb_pipeline, param_grid_gb, cv=5, return_train_score=True)
grid_search_gb.fit(X_train, y_train)
print("Best parameters found for GradientBoostingRegressor: ", grid_search_gb.best_params_)

Best parameters found for GradientBoostingRegressor: {'est__max_depth': 6}

# Retrieve the grid search results
results_gb = grid_search_gb.cv_results_
gs_df_gb = pd.DataFrame(results_gb)
gs_df_gb['rank_test_score'] = gs_df_gb['rank_test_score'].astype(int)
# Select relevant columns to display
gs_df_relevant_gb = gs_df_gb.loc[:, [
    'param_est__max_depth',
    'mean_train_score', 'std_train_score',
    'mean_test_score', 'std_test_score', 'rank_test_score'
]]
# Print the sorted grid search results
gs_df_relevant_gb
```

```
best_max_depth = grid_search_gb.best_params_['est__max_depth']
best_gbr = GradientBoostingRegressor(
    max_depth=best_max_depth
)

# Creating a new pipeline with this regressor
best_gbr_pipe = create_est_ppln(best_gbr, X_train, scaling=False, poly=False)
# Fit the pipeline with the best estimator on the training data
best_gbr_pipe.fit(X_train, y_train)
```

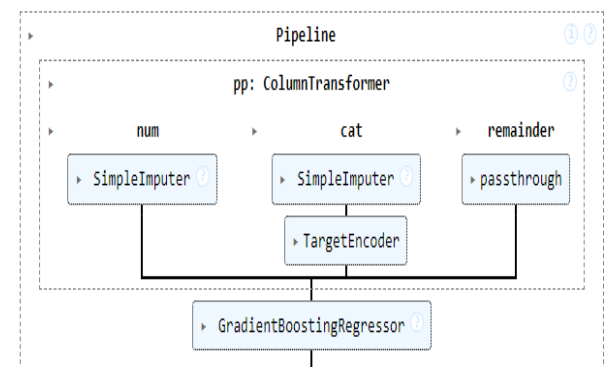


Figure 11. The code Implementation of Grid Search on Gradient Boosting Model

## Linear Regression

```
def create_est_ppln(est, X, scaling=True, poly=True):
    est_pipe = Pipeline(
        steps=[
            ("pp", create_pp_ppln(X, scaling, poly)),
            ("est", est),
        ]
    )

    return est_pipe

lr_regr = create_est_ppln(LinearRegression(), X_train)

# Linear Regression with linear_model=True to add scaling and polynomial features
lr_regr = create_est_ppln(LinearRegression(), X_train, scaling=True, poly=True)

lr_regr.fit(X_train, y_train)
```

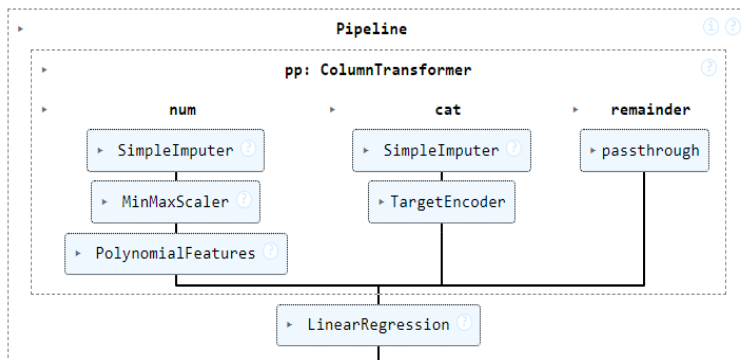


Figure 12. The Code Implementation for the Linear Regression Model

From the grid search results, the random forest model with a `max_depth` of 8 and `min_samples_leaf` of 10 was the best performing based on the test score, while the gradient boosting regressor had a `min_depth` of 8 as the best hyperparameters, as displayed in Table 1.

Table 2. The best hyperparameters obtained after Grid Search for the models.

Model	Hyperparameters	Best Hyperparameter	Rationale
Linear Regression	Default	Default	No hyperparameters are needed for this model.
Random Forest	Max_depth: 2, 5, 8 min_samples_leaf: 10, 20, 30	Max_depth: 8 Min_sample_leaf: 10	Chosen based on grid search
Gradient Boosting	Max_depth : 3, 4, 6, 8 n-estimators:	Max_depth: 8	Chosen based on grid search

## Ensemble Model

The ensemble method enhances predictive performance beyond what individual models could achieve. This approach capitalises on the strengths of the three models to achieve enhanced accuracy and more stable predictions. The ensemble was constructed using a simple voting regressor method, where each model is trained first according to its algorithm. Then, the voting regressor makes predictions by averaging all the models using simple voting, as shown in Figure 13. By utilising this approach, we aim to obtain more reliable and precise results than any single model could provide.

```
for est in my_models:
    est.fit(X_train, y_train)
```

```
ensemble = VotingRegressor(
    [
        ("gb", best_gbr_pipe),
        ("rf", best_rfr_pipe),
        ('lr', lr_regr)
    ]
)
ensemble.fit(X_train, y_train)
```

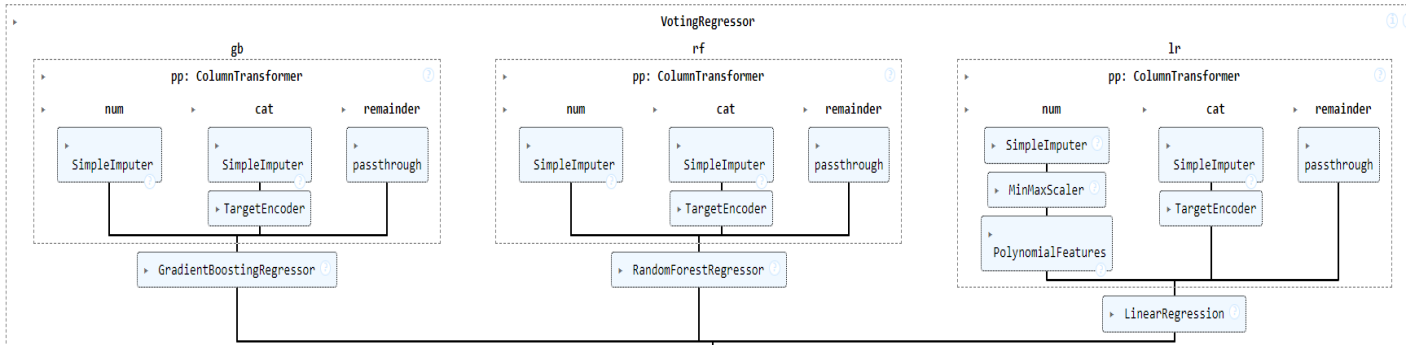


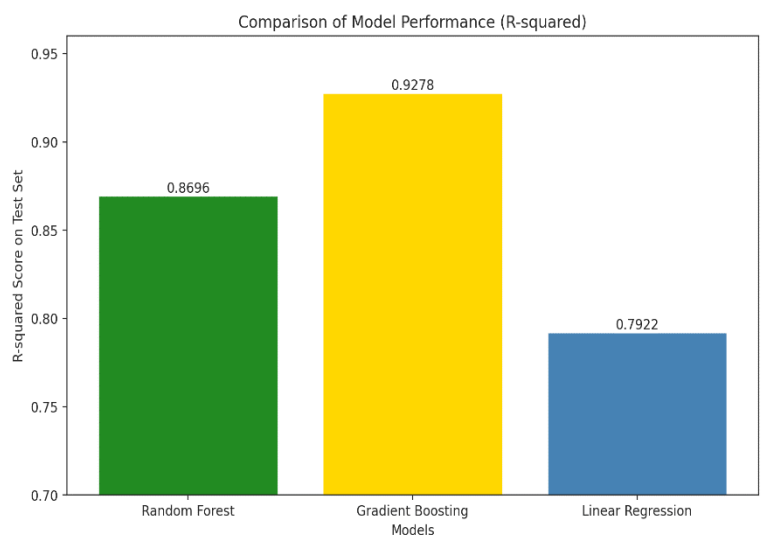
Figure 13.The Code Implementation for building the ensemble *model*.

## Model Evaluation and Analysis

The performance of the machine learning models was evaluated using Mean Absolute Error (MAE) and R-squared metrics through cross-validation. Table 3 presents each model's R-squared values derived from the test set, while Figures 14 and 15 present the mean absolute error obtained from the different regression models.

Table 3.The Performance of the different models on the Test and Train set for R-squared value.

Model	Test set	Train set
Gradient Boosting	0.9278	0.9435
Random Forest Tree	0.8695	0.8772
Linear Regression	0.7922	0.7958



Based on the R-squared values displayed in Table 3, gradient-boosting regression demonstrated the highest predictive performance on the test set with an R-squared value of 92% when compared to the other models, which suggests that the model was the most effective in predicting the variability in the target variable. Linear regression yielded the lowest R-squared value of 79%, suggesting that the model displayed a lower level of predictive performance when compared to gradient boosting and random forest models. The difference between the train and test sets of the three models was relatively small, which suggests that the model generalises well with the unseen data with no form of overfitting or underfitting.

```
eval_results = cross_validate(
    ensemble, X, y, cv=5,
    scoring='neg_mean_absolute_error',
    return_train_score=True
)
ensemble_result = (
    -eval_results['test_score'].mean(), eval_results['test_score'].std(),
    -eval_results['train_score'].mean(), eval_results['train_score'].std()
)
```

```
model_results.loc['ensemble'] = ensemble_result
```

```
model_results
```

	test_mae_mean	test_mae_std	train_mae_mean	train_mae_std
<b>best_gbr</b>	1572.300462	13.832781	1460.834232	3.662608
<b>best_rfr</b>	2135.187650	17.247506	2098.333460	11.360417
<b>lr_regr</b>	2902.833062	21.423692	2893.979091	4.874003
<b>ensemble</b>	2007.278600	14.118029	1960.682107	5.105803

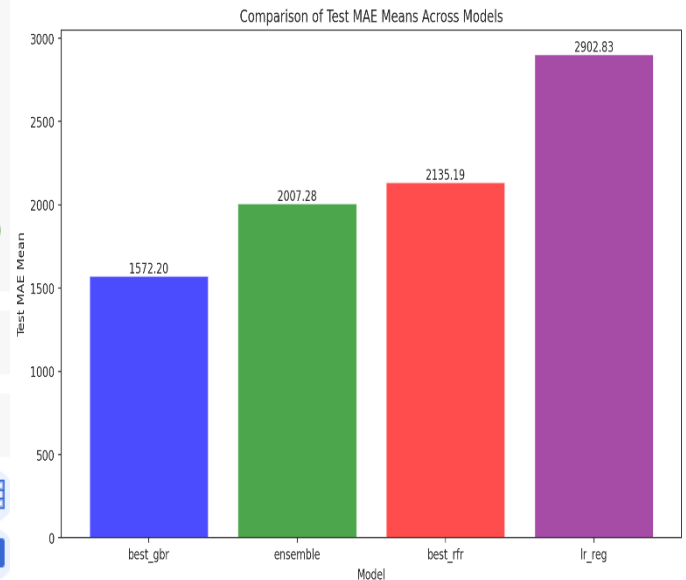


Figure 14. Visualising the *test set of the* Mean absolute error for the different models.

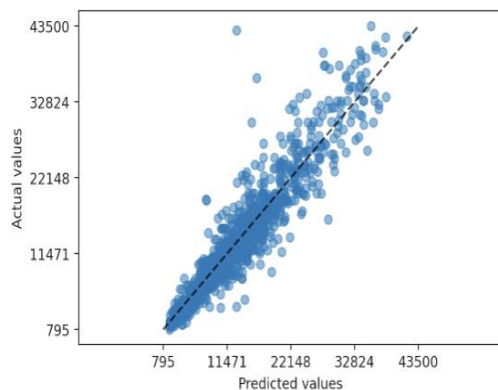
Figure 15. The mean absolute error obtained by cross-validation from the different regression models on the test and train set.

After employing the voting ensemble method, cross-validation was carried out to determine the mean absolute error for the test and train set on the different regression models, as displayed in Figure 15. The gradient boosting model had the lowest MAE on the test and train set, indicating that it has a good fit and is making predictions that are close to the true values in the dataset. The ensemble model had good generalisation, as indicated by the low MAE score compared to the random forest and linear regression, but was less effective than the gradient boosting model. The linear model had a higher mean MAE, indicating less accuracy. The difference between the test and train MAE was significantly low, indicating no overfitting or underfitting in the models.

## True vs Predicted Analysis

```
from sklearn.metrics import PredictionErrorDisplay
PredictionErrorDisplay.from_estimator(
    best_rfr_pipe, X_test, y_test, kind="actual_vs_predicted", scatter_kwargs=dict(alpha=0.5)
)
```

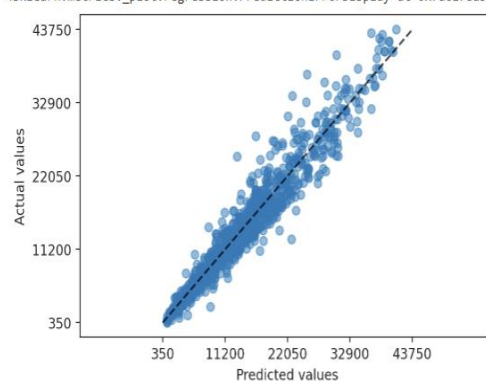
<sklearn.metrics.\_plot.regression.PredictionErrorDisplay at 0x7a6183808460>



Random Forest Regressor Model

```
from sklearn.metrics import PredictionErrorDisplay
PredictionErrorDisplay.from_estimator(
    best_gbr_pipe, X_test, y_test, kind="actual_vs_predicted", scatter_kwargs=dict(alpha=0.5)
)
```

<sklearn.metrics.\_plot.regression.PredictionErrorDisplay at 0x7a6178a58e80>

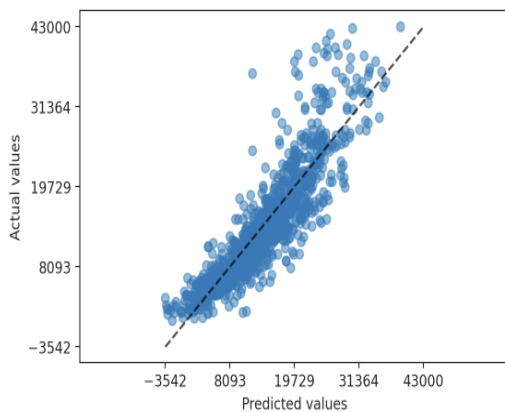


Gradient Boosting Regressor Model



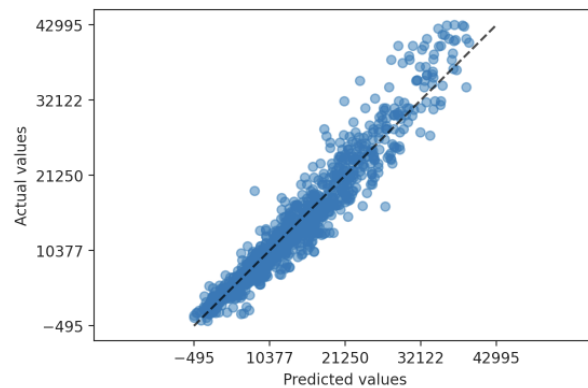
```
from sklearn.metrics import PredictionErrorDisplay
PredictionErrorDisplay.from_estimator(
    lr_regr, X_test, y_test, kind="actual_vs_predicted", scatter_kwargs=dict(alpha=0.5)
)
```

<sklearn.metrics.\_plot.regression.PredictionErrorDisplay at 0x7a616adfc850>



```
from sklearn.metrics import PredictionErrorDisplay
PredictionErrorDisplay.from_estimator(
    ensemble, X_test, y_test, kind="actual_vs_predicted",
    scatter_kwargs=dict(alpha=0.5)
)
```

<sklearn.metrics.\_plot.regression.PredictionErrorDisplay at 0x7a618ba74280>



### Linear Regression Model

### Ensemble Model

The plot above compares the predicted and actual values for the test set from the four different regression models used in this analysis. This is used to evaluate the performance of the models. The plot above showed that the gradient-boosting model had the most accurate predicting values, with tight clustering around the prediction line and consistent results across all value ranges. The random forest model predicted well for the lower range but had less precision with higher values, suggesting it performed best on cars with lower prices. The linear regression model had the largest variability, especially for higher-value predictions. The ensemble of the three models was more accurate than the random forest and linear regression but did not outperform the gradient-boosting model. Notably, all models demonstrated an increasing prediction spread for higher values. The gradient boosting was the best on both the train and test, followed by the ensemble, random forest, and linear regression, as shown in the above figure.

### SHAP

Shap is a tool that allows for a detailed analysis of individual predictions by identifying the factors that contributed to each prediction. To perform this analysis, the local explanation and global plot were utilised. The local explanation helped interpret the model's prediction for individual instances, while the global plot sorted features based on their importance in predicting the model's overall output. The most important features are listed at the top, and features with large negative and positive Shap values can be visualised.

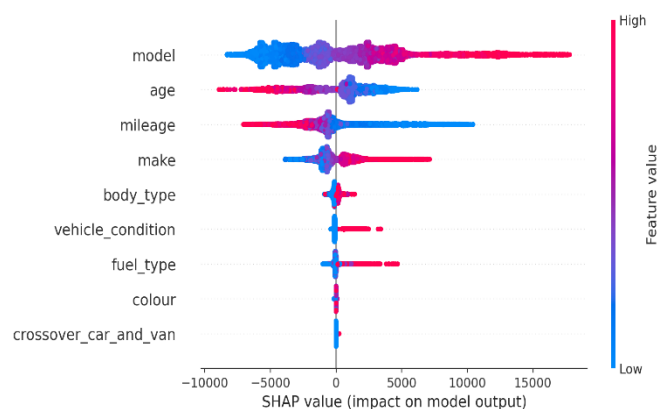
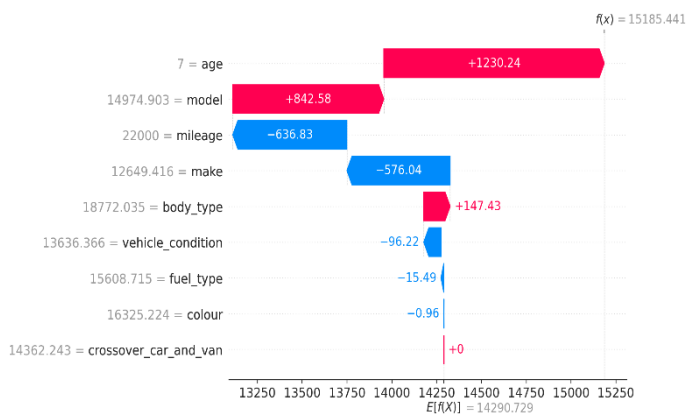


Figure 16. Local Explanation and Global Plot for Random Forest Model

The Shap local explanation plot above for the random forest model indicates that the age (7) and model of this vehicle positively affect the model prediction by 1230.24 and 842.58, respectively, with age having the largest positive influence. This suggests that newer vehicles are more expensive, and the specific car model is associated with higher values in the target variable or might be premium. However, mileage and make decreased the model's prediction by 636.83 and 576.04, suggesting that higher mileage decreases the car price, and this vehicle brand decreases the target value. The body type and vehicle condition showed minimal impact; body type increased the prediction by 147.43, and vehicle condition decreased the prediction by 96.22, indicating the car might be used and the body type might be expensive. Other features like fuel type and colour had less impact on the overall value. In contrast, crossover\_car\_and\_van did not impact the model prediction, indicating that these factors do not significantly drive the model's predictions.

The global plot indicates that the standard model, age, and mileage are the most important features of the random forest model. The model feature displayed a wide range of positive and negative Shap values, indicating that certain models significantly reduce or increase the car. While for age and mileage, it's the opposite; large feature values are associated with smaller Shap values, suggesting that lower age and mileage enhance the car value. The make indicates higher feature values are associated with higher Shap values, suggesting low feature values associated with makes are less preferred. While vehicle condition, body type and fuel type displayed higher Shap values with high feature values. The colour and crossover\_car\_and\_van features displayed no significant influence.

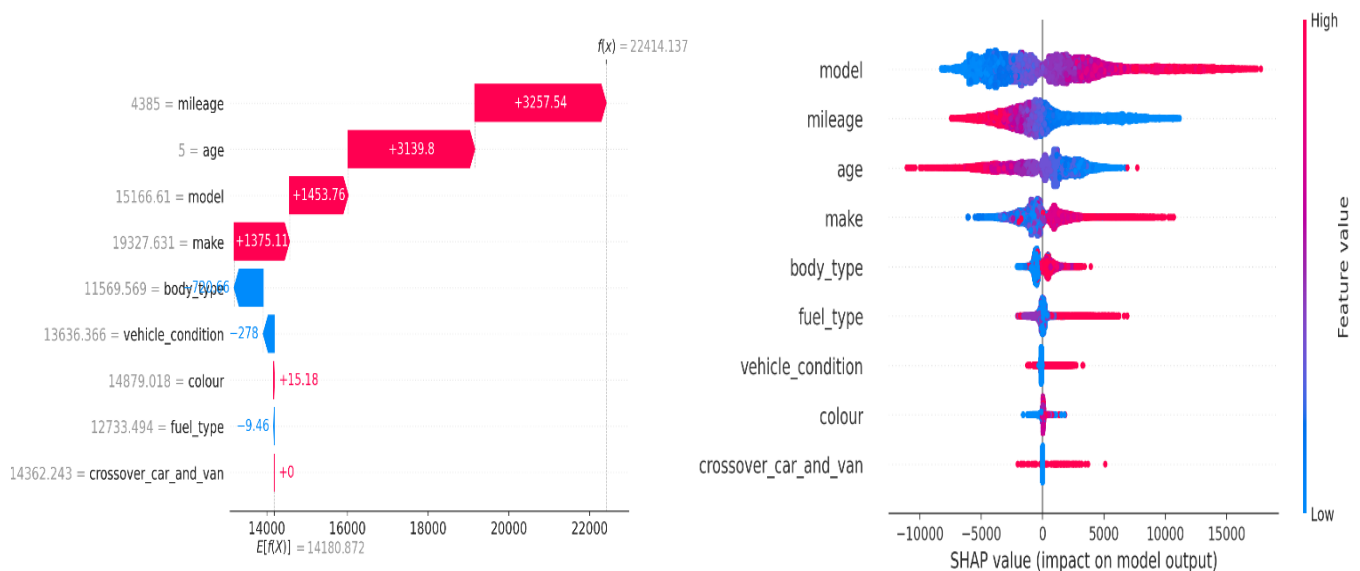


Figure 17. Gradient Boosting Regressor Model Local Explanation and Global Plot

The waterfall plot for the gradient boosting model In this particular instance, the car's mileage, age, model, and make significantly increased the prediction by 3257.54, 3139.8, 1453.76, and 1375.11, respectively. However, the body type and vehicle condition decrease the prediction. The bee warm plot for the gradient boosting model showed a more pronounced effect on top key features like model, mileage, and age. The standard model displayed a wide spread of negative and positive Shap values with large feature values associated with larger Shap values. Age displayed broadly distributed Shap values, with high feature values associated with lower Shap values. Crossover\_van\_car, vehicle condition and colour had minimal effect on the model prediction, indicating that these features are not significant determinants of car price in this model.

Kolawole Mary Biola

23659564

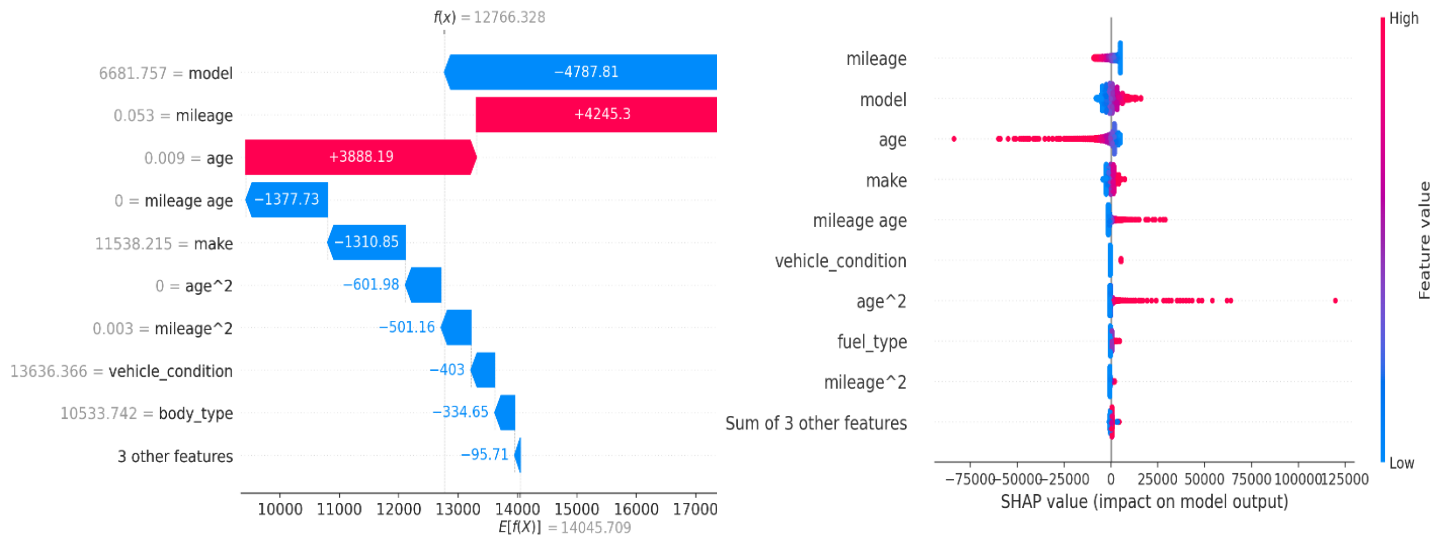


Figure 18. Linear Regression Local Explanation and Global Plot

The waterfall plot for the linear regression model shows that age and mileage increased the prediction of the log odds by 4245.3 and 3888.19. However, the standard model had an extremely negative impact on prediction, decreasing it by 4587.81. Additionally, the remaining features had a negative impact, indicating that these features typically decrease the vehicle price for this particular instance. The linear regression global plot displayed a larger range of Shap values than the tree models. Across the different models employed in this analysis, standard model, mileage, and age are the most important features, although the feature importances differ in magnitude and directionality for each model.

## Partial Dependency Plots

### Gradient Boosting Regressor model

The partial dependency plot visualises the marginal effect of a predicted outcome on the predictive feature of interest by plotting the average model outcome at different predictive variable values. It indicates how a target outcome affects the predictive variable on average.

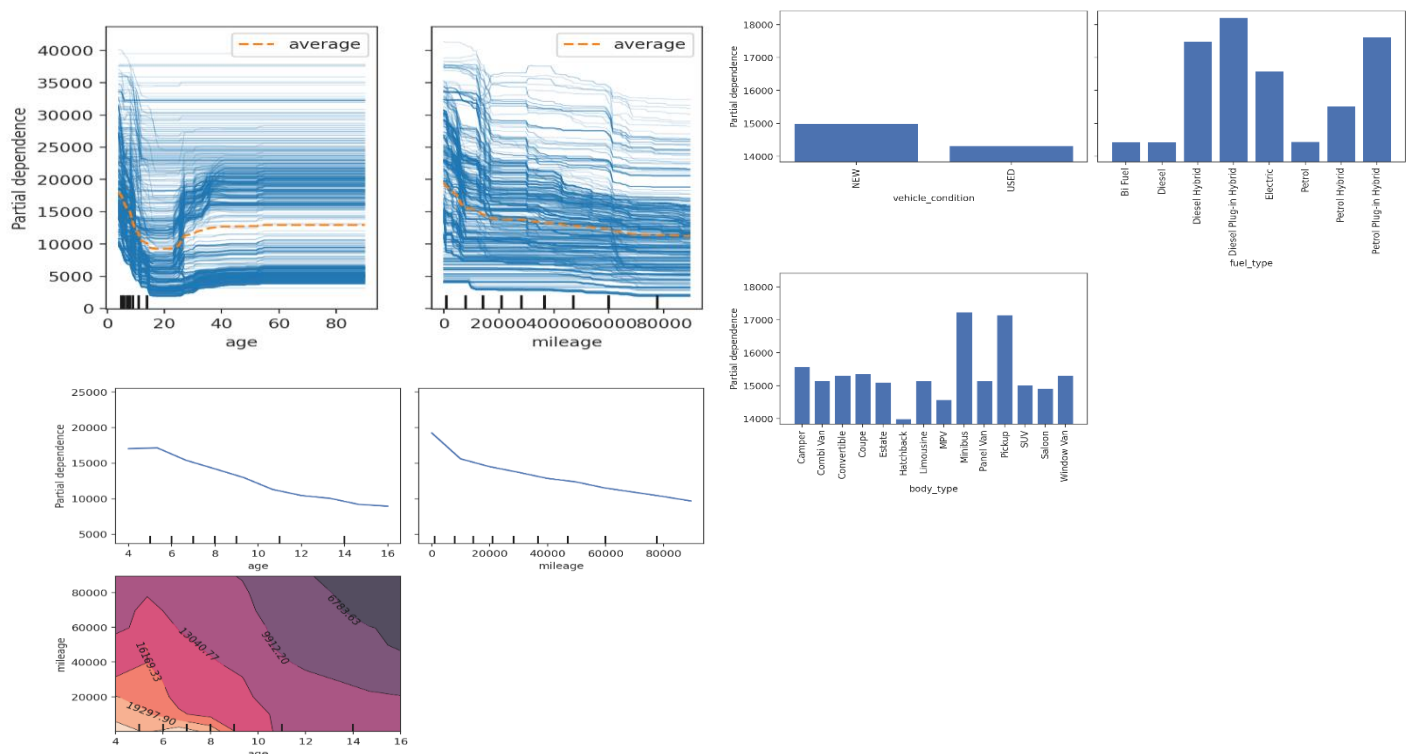


Figure 19. The Partial dependency and contour plot for the Gradient Boosting regressor Model

From the PDP plot above in Figure 19, it can be observed that the average predicted price exhibits a nonlinear trend for the age of the vehicle. As age increases, the average predicted price decreases, and then there is a slight increase in the predicted price as the vehicle approaches 20 years, stabilising at 59 years. The increase in the average marginal effect may be attributed to the fact that these vehicles within that range are vintage models. The PDP plot suggests that the predicted price generally decreases as mileage increases. The PDP plot for vehicle condition indicates that the new car has a higher predicted value than the used car, and fuel types petrol, bi-fuel and diesel were associated with lower predicted prices. The contour plot visualises the interaction between features against the predicted price. It can be observed that older cars with high mileage decrease the predicted price.

### Random Forest Model

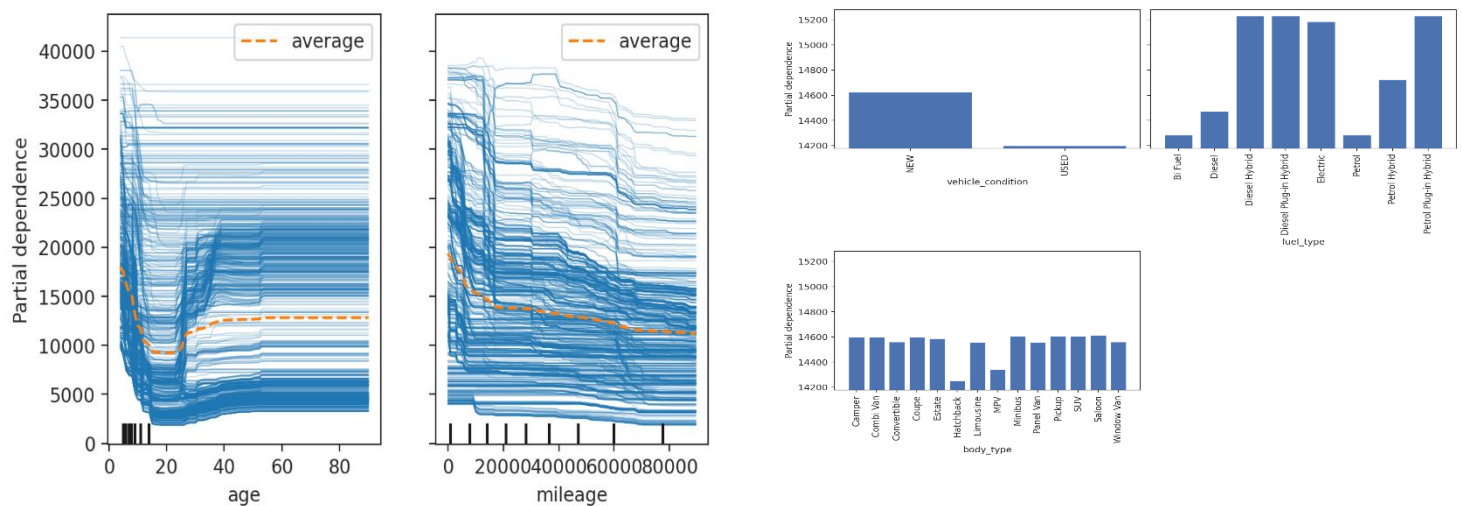
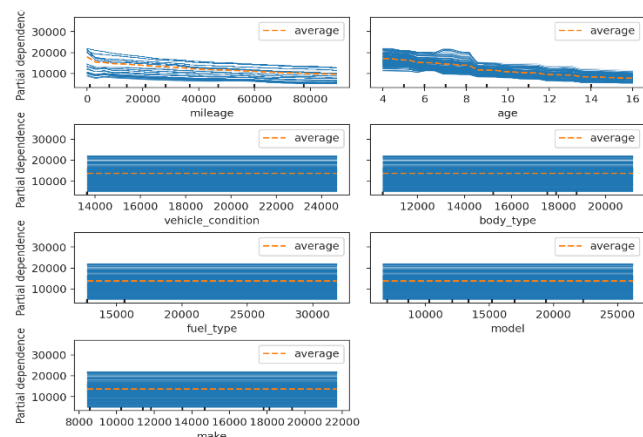


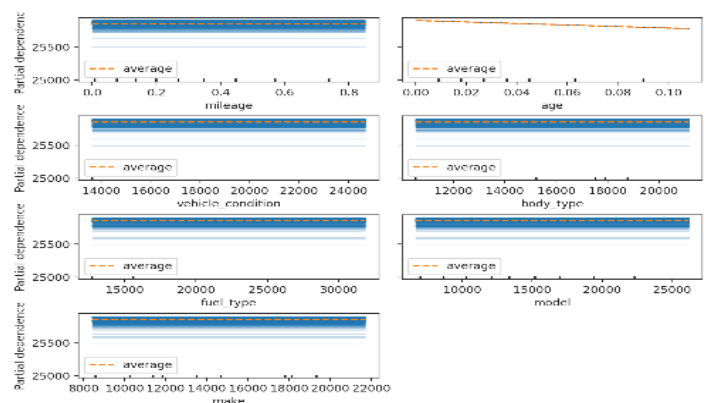
Figure 20. The Partial dependency plot for the Random Forest regressor Model

Similar trends were observed in the PDP plot for the random forest model. Age and mileage independently contributed to the predicted price depreciation. The PDP plot for vehicle condition indicates that the new car has a higher predicted value than the used car, and fuel types petrol and diesel were associated with lower predicted prices. However, for the body type, they were less pronounced, suggesting that this feature does not significantly impact the predicted outcome for this model.

### Ensemble Model



### Linear regression Model



The linear regression model displayed a straight line for the feature because it's a linear model, while the ensemble model displayed a non-linear relationship for age and mileage against the predicted value.

To conclude, out of the four models utilised in this analysis, the gradient-boosting regressor model was the best, with an R-squared value of 92% and an MAE of 1572.19 compared to other models.