Abdirahman Abdullahi- 18062413

Advanced Machine Learning

By Abdirahman Abduulahi

I. DATA DESCRIPTION AND PREPROCCESSING

This The first step was to examine the target variable price. Although this column contained no missing values, the distribution was extremely right skewed, with a long tail of extreme values reaching into the millions, as seen in Figure 1. Left untreated, this would most certainly distort the model learning and would overemphasise outlier behaviour. To address the issue, I applied a clipping threshold at £100,000. This allowed expensive vehicles to remain in the dataset while reducing the influence of astonishingly expensive cars. Importantly, no rows were dropped. only capped. Figure 1 shows the distribution before and after clipping, with a clear shift towards a more usable target for regression

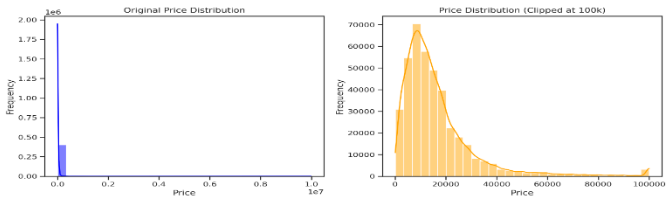


***Figure 1**: price distribution, original vs clipped at 100k*

Following this, I explored the mileage feature, which showed a similar long-tailed distribution (Figure 2). Since mileage is often a strong predictor of used car prices, it was important to correct this skew to avoid biasing the models. I applied a log transformation using log(x + 1), which compressed the tail and produced a much more symmetric distribution. This was done prior to any scaling to improve stability in later models.

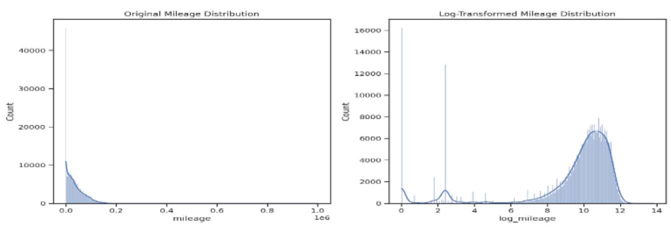


***Figure 2**: shows mileage distribution, original vs log +1 transformed*

Next came handling missing data. Most features, such as reg_code, body_type, and fuel _type, had low levels of missing data. Since it wasn't extreme loss, I opted for a simple imputation method, rather than dropping rows. For numerical features, I used the median (middle value) to avoid distortion from outliers. For categorical features, I used the mode (most frequent value) to maintain interpretability and consistency.
A particularly tricky feature was year _of_ registration, which had a mix of missing values, impossible outliers (e.g., values like the year 999), and inconsistent formatting. I cleaned it using a multi-step logic. First, any values outside the range 1909–2020 were adjusted using reg code, which maps registration suffixes to valid years (e.g., "A" for 1963). Where both values were missing, and the vehicle was listed as "new" (vehicle_condition = 1), I set the year to 2021(the youngest age in the dataset) for robustness. After these corrections, less than 1% of entries remained null, these nulls were then filled using the median. After this process, year_of_registration became a much more reliable feature ready for engineering.
Once the dataset was clean, I moved on to feature engineering. One of the most important additions was a new categorical combo feature I called **MMBF**, built by concatenating standard_make, standard_model, body_type, and fuel_type. The goal was to capture useful interactions between these attributes that affect pricing. I applied **K-fold target encoding** using 5 folds to convert MMBF into a numeric feature without leaking any information. The result was a new column, MMBF_target_encoded.Which represented the mean sale price for each group. About 655 values were missing possibly due to rare combinations unseen in training folds. These were then filled with the global mean price. As a result, these steps ensured the model would get well structured data that discriminated well into clean groups and preserved important variation.
I also derived a vehicle_age_years column by subtracting year_of_registration from 2021, then clipped any values above 50 years to remove extreme outliers, without losing rows. These engineered features MMBF_target_encoded and vehicle_age_years were

included in the final dataset and proved useful across multiple models. A summary and barchart of the distribution of the MMBF and vehicle age feature is shown below:

MMBF_target_encoded	
count	402005.00
mean	16543.10
std	12886.61
min	295.00
25%	8670.47
50%	13387.23
75%	18631.61

***Table 1**: shows the statistics behind the feature engineered groups in MMBF_target_encoded*

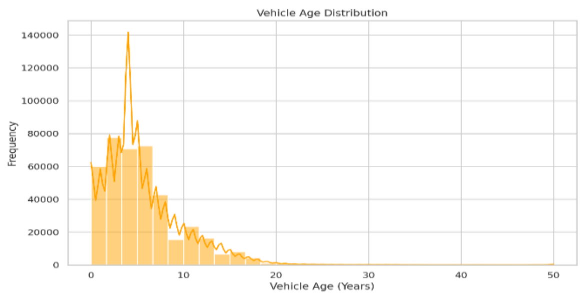
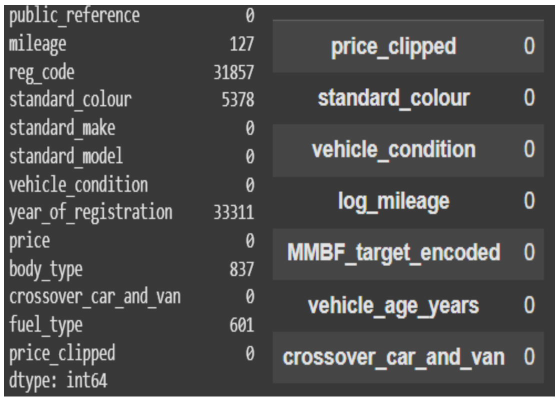


***Figure 3**: shows the distribution of the engineered feature vehicle age*

After all the cleaning and engineering, the final dataset was far smaller and focused, while utilising all of the features given in a meaningful way. These included: the target price (price_clipped), cleaned standard_colour (which is to be one-hot-encoded.), mileage (log_mileage), the new target-encoded combo feature (MMBF_target_encoded), calculated vehicle age (vehicle_age_years), and a binary flag identifying crossovers crossover_car_and_van and vehicle_codnition. All features were clean, with zero missing values. Ready for modelling. Lastly, Figure 4 shows the final dataset before and after. With all features utilised except for public reference.



```
public_reference          0
mileage                 127
reg_code              31857
standard_colour        5378
standard_make             0
standard_model            0
vehicle_condition         0
year_of_registration  33311
price                     0
body_type               837
crossover_car_and_van     0
fuel_type               601
price_clipped             0
dtype: int64
```

```
price_clipped           0
standard_colour         0
vehicle_condition       0
log_mileage             0
MMBF_target_encoded     0
vehicle_age_years       0
crossover_car_and_van   0
```

***Figure 4**: shows the final dataset before and after.*

II. Automated Feature Selection

I applied automated feature selection using Recursive Feature Elimination (RFE) with a Random Forest Regressor as the estimator. The goal here was to identify and confirm our suspicions on the most predictive features from the cleaned dataset, in the hopes of reducing redundancy and improving generalisation.

The dataset was first sampled down to 40% of the total (approximately 160,000 rows) to make the process more computationally efficient without losing too much variance. I then split this sample into training (60%), validation (20%), and test (20%) sets using stratified random sampling. While not at all strictly required for RFE itself, this early split was done to prevent any data leakage and support a consistent modelling pipeline in later stages. As we already know the target variable was price clipped, and the input features included all engineered and encoded attributes found in Figure 4. notably, we assumed log_mileage, MMBF_target_encoded, and vehicle_age_years would be the most informative features found in our dataset, as they are the closest features linked in our domain knowledge to depreciation or appreciation in price.

To confirm this assumption, we used RFE with a Random Forest model (100 trees, default hyperparameters). The algorithm recursively removed the least important features based on the model's internal importance scores. retraining at each iteration. Which allowed the selection process to consider non-linear interactions and correlated inputs. something simpler methods like univariate filtering for example can miss.

The results are visualised in Figure 5, which ranks all candidate features from most to least useful. As expected, high-performing variables included log_mileage, MMBF_target_encoded, vehicle_age_years and also vehicle_condition,. These align well with our early assumptions as a car's mileage, age, and make-model combination are core predictors of value. Interestingly, several one-hot encoded colours also received relatively strong rankings, with shades like black, silver, and grey appearing more predictive than other colours most likely reflecting mainstream desire for more common monotone colours.

On the other hand, more niche colours like colour_Indigo, colour_Magenta, and colour_Multicolour were ranked lower by the RFE process. However, rather than removing these lower-ranked features outright with vehicle crossover and van, that also scored low. I chose to keep them in the modelling pipeline. My reasoning for this was that even less favourable or less common attributes might carry some subtle signals that ensemble models most especially tree based ones, can pick up. This decision aimed to preserve diversity in the feature set and explore whether these "weaker" inputs could still play a role in creating a more robust and generalisable model. Despite the larger amount of seemingly unimportant features in isolation.

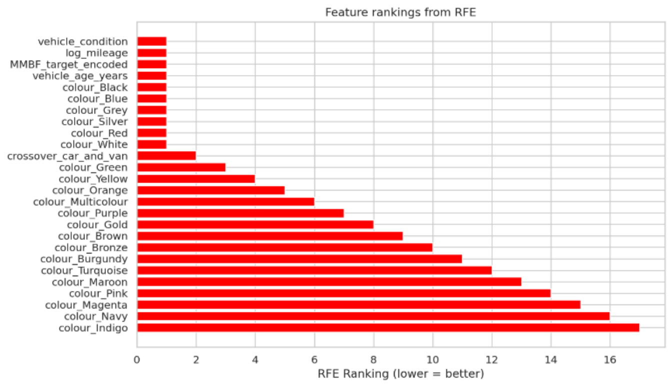


Figure 5*: shows the Recursive Feature Elimination. Showing the most and least impactful features.*

III. Tree Ensembles

To build a strong baseline for the regression task, I trained and evaluated two tree ensemble models: a **Random Forest Regressor** and a **Gradient Boosting Regressor**. Both are well-suited for tabular data, however random forest does have a slight edge on gradient boosting as random forest is more robust to overfitting as the tress grows. On the other hand despite this limitations they both could model non-linear relationships and handle feature interactions making them ideal candidates for this dataset.

Random Forest Regressor

I first trained a Random Forest model using 100 trees and a maximum depth of 10. On the validation set, it achieved a **MAE of 2739**, **RMSE of 4520**, and an **R² score of 0.91**. This was already a really strong out-of-the-box performance. After tuning via GridSearchCV (testing multiple combinations of tree depth, estimators, and minimum split size), the best model with 200 estimators, max depth 20 , and min split of 5, improved to a **MAE of 2216**, **RMSE of 3968**, and **R² of 0.93**. This confirmed the benefit of even basic tuning in ensemble methods.
The top features in both default and tuned Random Forests were consistent: MMBF_target_encoded was by far the most dominant, followed by log_mileage and vehicle_age_years as seen in Figure 6. This aligned well with the assumption made earlier in task 2 as it visually confirms the significant impact of the engineered features. As well as this Less predictive features like individual colour encodings still showed small contributions, which justified their inclusion in the modelling pipeline but didn't have close to the same effect as the big three did.

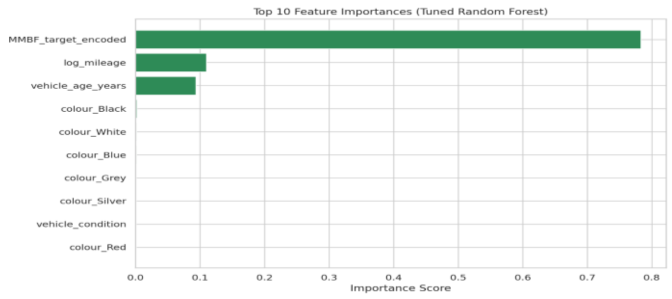


Figure 6*: shows the top 10 most important features used in predictions for tuned random forest regressor*

Gradient Boosting Regressor

I also trained a Gradient Boosting Regressor using 200 trees, a learning rate of 0.1, and max depth of 5 with 80% subsampling. On the validation set, it achieved a **MAE of 2696**, **RMSE of 4434**, and an **R² of 0.91**, which was only slightly better than the untuned Random Forest but not quite at the level of the tuned version. To get a broader view, I also ran 5-fold cross-validation set. The average scores came out to be: **MAE ~2694**, **RMSE ~ 4495**, and **R² ~ 0.911**, suggesting the model was fairly stable across different data splits.
Same as with the Random Forests, the feature importance Figure 7 showed MMBF_target_encoded leading by a wide margin, followed by vehicle_age_years and log_mileage. The pattern here was slightly different as vehicle_condition dropped in importance compared to the Random Forest, and some colour categories registered more strongly. Potentially due to how boosting fits residuals iteratively as it learns from error made by previous trees, while random forest averages out those predictions.

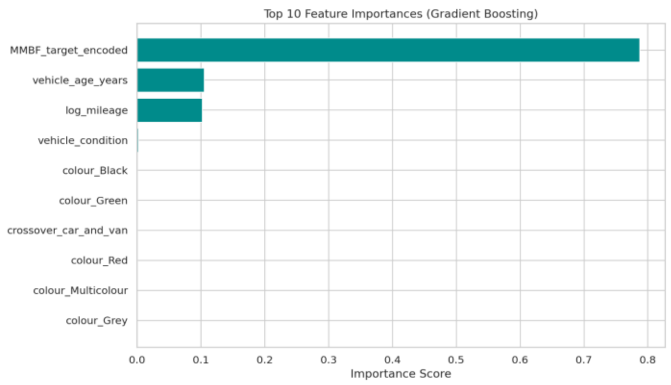


Figure 7*: shows the top 10 most important features used in predictions for gradient boosting regressor*

Overall, to conclude Both tree ensemble methods performed well, with Gradient Boosting giving solid results with fewer hyperparameters and Random Forest showing more flexibility once tuned. Feature importance was consistent across both models, reaffirming the strength of the engineered features. These models set a strong foundation for the next task combining their strengths in an ensemble-of-ensembles approach in task 4.

IV. Ensemble Of Tree Ensembles

After evaluating the Random Forest and Gradient Boosting models individually, I explored ensemble methods to combine their strengths. The intuition here was that each model captures slightly different aspects of the data. As previously mentioned Random Forests are generally more stable and less sensitive to noise and overfitting, while Gradient Boosting often fits finer residual patterns. By combining them, I aimed to reduce variance and improve overall predictive accuracy.

Weighted Average Ensemble

The first ensemble method I tested was a simple weighted average. I blended the predictions from the **tuned Random Forest** (the slightly more accurate) and the **Gradient Boosting Regressor**, assigning 70%/30% split. This ratio was based on experimentation. as it slightly favoured the better-performing model but still allowed gradient boosting to contribute.

The combined model produced a **MAE of 2276**, **RMSE of 3975**, and **R^2 of 0.931** on the validation set. While the improvement was relatively modest, the ensemble consistently outperformed either individual models. The predicted vs. actual plot (Figure 8) showed a strong alignment, especially across the bulk of the distribution, with some noise at the upper end most likely due to target (price) clipping during preprocessing.

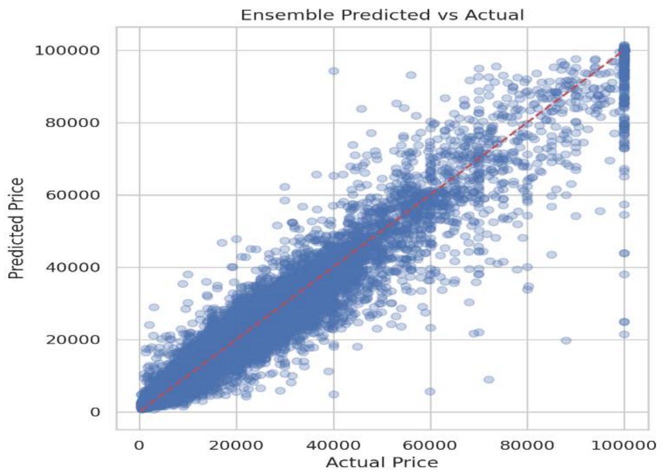


Figure 8*: shows the actual vs predicted pot for weighted average ensemble*

Stacked Ensemble

To push it further, I also implemented and tested a stacked ensemble using a Stacking-Regressor. This approach combines base models (here, the tuned Random Forest and Gradient Boosting) and feeds their predictions into a simple linear regression. The stacking regressor was trained with 5-fold cross-validation and passthrough disabled to avoid overfitting on raw inputs.

The stacked ensemble performed slightly better than the weighted average, achieving a **MAE of 2234**, **RMSE of 3957**, and an **R^2 of 0.9312**. While the gains were incremental, they showed that stacking added value by learning how to best combine the base predictions rather than relying on fixed weights (70/30). The predicted vs. actual plot (Figure 9) also looked very similar, notably at the extremes.

V. Feature Importance

To gain a deeper understanding of how different features were contributing to model performance, I used **permutation importance**. a method that quantifies how much the prediction error increases when a feature's values are randomly shuffled. Unlike tree-based feature importance scores (which rely on how often a feature is used in splits) that we have relied on so far. permutation importance directly reflects impact on model accuracy, making it more interpretable and robust measurement.

I applied this analysis to both the **Gradient Boosting Regressor** and the **tuned Random Forest** models, using 5 repeats on the validation set to reduce variance in the estimates. For both models, the results were consistent with earlier assumptions and evidence as it once again highlighted the dominance of engineered features.

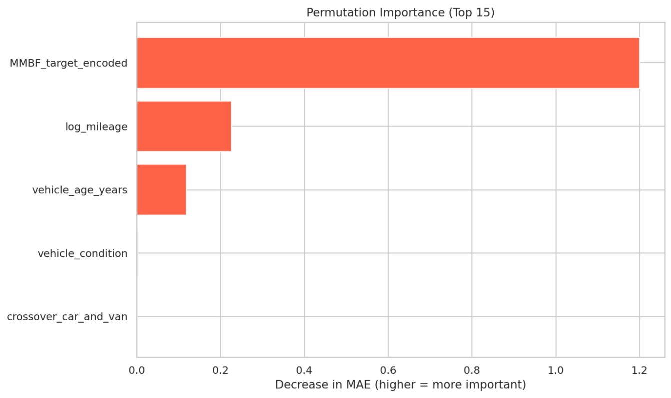


***Figure 10**: shows the permutation importance for the gradient boosting regressor.*

As seen in Figure 10 and 11, MMBF_target_encoded across the board stood out clearly as the most important feature. Permuting this column alone led to the largest increase in Mean Absolute Error (MAE), with a score nearly **5x higher** than the next-best feature. Reinforcing that the combination of make, model, body type, and fuel type with each group encoded with the groups mean price made it incredibly valuable as it showed the pricing patterns the models were heavily relying on.

Following that, log_mileage and vehicle_age_years consistently ranked second and third in importance. This matches intuitive expectations we had. cars with high mileage or advanced age tend to sell for less and vice versa, and that these effects compound when combined with other factors.

A few other features like vehicle_condition, crossover_car_and_van, and specific colour categories showed smaller but still non-trivial contributions. Interestingly, the tuned Random Forest placed slightly more emphasis on the colour White than the Gradient Boosting model, which could suggest it found minor residual patterns there that boosting didn't prioritise or eliminated.

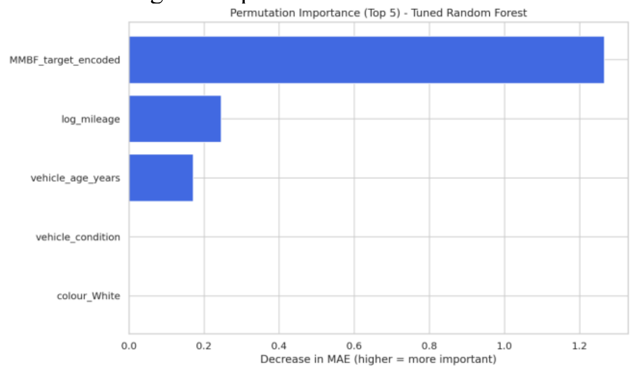


***Figure 11**: shows the permutation importance for the tuned random forest regressor.*

Figure10 and 11 both show the top five features by permutation importance for both models. The alignment between them confirms that the modelling pipeline is robust and that the key engineered features are doing the heavy lifting across different ensemble approaches.

This step provides further evidence on which features the models truly relied on not just for accuracy, but for decision making. It also helped validate the earlier feature engineering choices and gave me much needed confidence moving into interpretability techniques like SHAP and PDP.

VI. SHAP And PDPs

To better understand how the models were making decisions beyond just looking at accuracy, I used two key interpretability methods. Those being: SHAP (SHapley Additive Explanations) and Partial Dependence Plots (PDPs). Both are designed to make black-box models more transparent, but they approach the problem from different angles. SHAP breaks down individual predictions into feature contributions, assigning each variable a value that represents how much it pushed the prediction up or down. While PDPs, on the other hand, show how the model's average output changes as a single feature or pair of features varies, holding everything else constant.

I started by looking at global feature importance using SHAP values generated from the Gradient Boosting model. The summary bar plot (Figure 12) provided more evidence and made it immediately clear just how dominant the MMBF_target_encoded feature was. On average, this variable alone contributed over £7,000 to the predicted price. Far exceeding any other feature in the dataset. This made intuitive sense and provides further confirmation to the assumptions made previously. This is because MMBF_target_encoded represents a grouped encoding of make, model, body type, and fuel type, effectively condensing a lot of useful market signal into a single value. The next most important features was of no surprise. Log_mileage and vehicle_age_years, both of which are typical indicators of depreciation. Their contributions were smaller, but still meaningful.

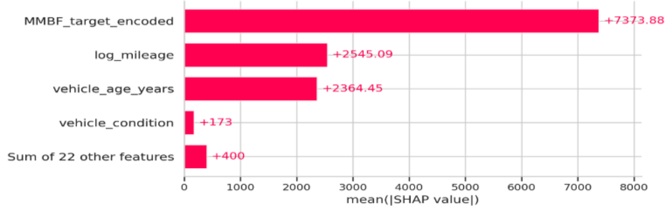


***Figure 12**: shows global SHAP feature importance Barplot for gradient boosting*

Going beyond just which features mattered most, I used a SHAP beeswarm plot (Figure 13) based on our best individual model the tuned Random Forest. This helped reveal how different feature values influenced predictions across the dataset. In this plot, each dot represents a single vehicle, and its horizontal position shows how much that specific feature affected the price. Colour gradients were used to indicate whether the feature value was high (red) or low(blue). What stood out was how consistently high MMBF_target_encoded values increased predicted price, while high mileage and older vehicles had the opposite effect, pulling predictions downward. Even the colour features, which ranked lower in importance, showed small pockets of influence, suggesting they may matter more in edge cases or for specific car types.

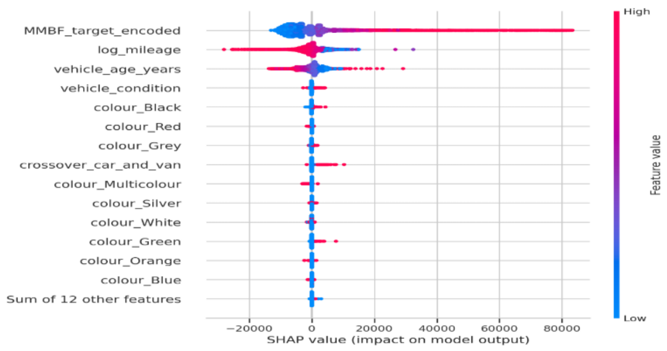


***Figure 13**: shows SHAP Beeswarm for the tuned random forest*

To see how this played out in a single case, I used a SHAP waterfall plot seen in Figure 14 to break down one prediction from the tuned Random Forest. For this particular vehicle, the model predicted a price of £19,502, compared to the actual price of £16,390. Figure 14 showed how the model got there: the car's low age (just two years) pushed the price up by over £3,400, while a modest MMBF_target_encoded value actually reduced the predicted price by nearly £1,000. Other features like log_mileage, vehicle_condition, and a few colour indicators made smaller contributions. It was satisfying to see the model's reasoning align with what would make sense from a buyer's perspective, as newer cars go for more, and popular make/models tend to carry premium pricing.

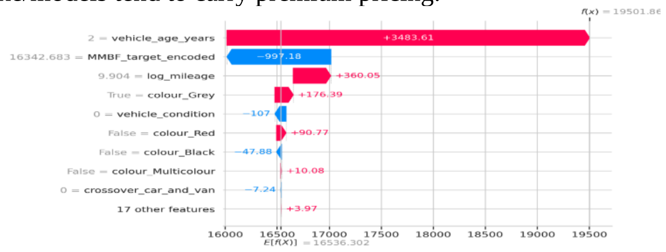


***Figure 14**: shows SHAP Waterfall for a single prediction in the tuned random forest*

While SHAP helped explain specific predictions and feature interactions, I also wanted to understand the average behaviour of the model when key variables changed. For that, I turned to Partial Dependence Plots (PDPs). These plots let you see how the model's output shifts as one feature changes across its range, while the rest are averaged out. I focused again on MMBF_target_encoded, since it was so influential. The resulting PDP showed a strikingly linear trend: as the encoded group price increased, so did the model's predicted price, with a steady upward slope. There was very little noise or non-linearity, which confirmed that the model was using this feature in a smooth, predictable way.

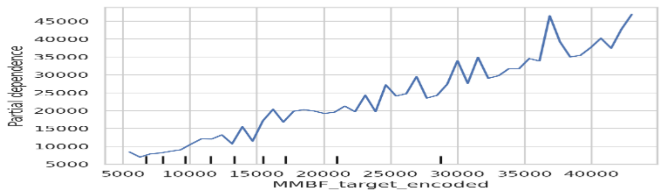


***Figure 15**: shows Partial Dependence Plot for MMBF_target_encoded*

To conclude all of this. The global SHAP summary, the beeswarm distribution, the waterfall breakdown for a specific case, and the PDP for our most important feature. All gave a strong sense that the model wasn't just accurate, but also logical. As it behaved in ways that were consistent with domain knowledge and relied on the features we'd expect. These interpretability tools and methods helped confirm that the learned relationships weren't at all arbitrary, and that the model's decisions and outputs could be understood, trusted and most importantly interpretable. something that's especially valuable in real-world applications, where transparency matters most.

VII. Dimensionality reduction, Linear

After completing feature engineering and encoding, the final dataset included 27 features. However, it's worth noting that only 5 of these were genuine continuous or meaningful engineered features (like mileage, age, condition, etc.). The rest were one-hot encoded colour categories which while it could've been potentially useful, has so far not done much except producing a sparse, high-dimensional structure without necessarily adding deep complexity.

To assess the underlying structure of the feature space and explore much needed dimensionality reduction, I applied **Principal Component Analysis (PCA)**. Because PCA is sensitive to scale, all features were standardised beforehand using StandardScaler.

Figure 16 shows the **cumulative explained variance**. This tracks how much of the dataset's overall information is retained as we include more principal components. The curve steadily rises, and importantly, it levels out after around 22 components. This means that 95% of the total variance in the data can be preserved using just 22 transformed components, instead of the full original 27. This confirms that some of the encoded colour dimensions may be redundant or overlapping in the way they contribute to the dataset's variance, and also means that some encoded colours added to the feature space.

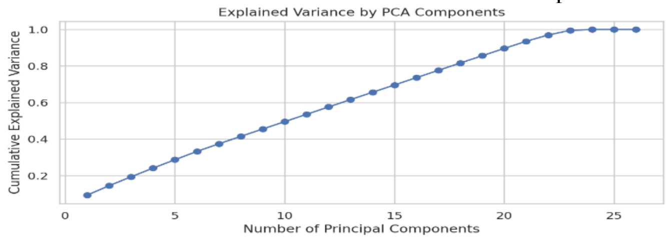


***Figure 16**: shows Cumulative explained variance by number of PCA components*

Figure 17 shows a **2D scatterplot** of the dataset projected onto the first two principal components. Each point represents a row of data, positioned according to how it scores on these new axes. The components themselves are "abstract" they're combinations of the original features but they help highlight general structure or spread in the data. While the plot doesn't reveal strong clustering patterns, it still suggests that some broad variation is being captured in just two dimensions.

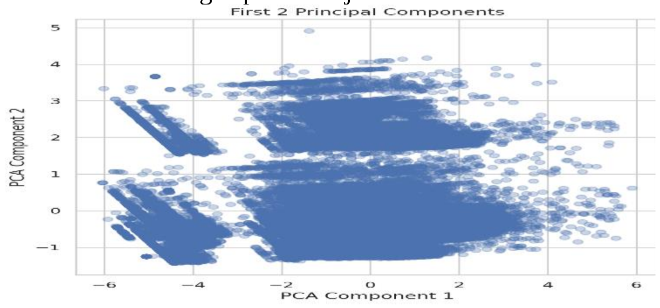


Figure 17*: shows 2D PCA projection using the first two principal components*

Overall, this PCA analysis showed that while the data is moderately high-dimensional, much of the meaningful variation is driven by a relatively small set of informative features. This supports the idea that colour encodings ,which we had retained for modelling, may have played a minor role in interpretation and didn't contribute significantly to the dataset's underlying structure.

VIII. Dimensionality reduction, Non Linear

To explore whether there were any hidden patterns in the data, I used a technique called **Isomap**. This is a type of dimensionality reduction method. Whch in simple terms,helps take complex, high-dimensional data and map it down to just two dimensions so we can visualise it more easily. Unlike PCA, which looks for straight-line patterns in the data, Isomap is better at picking up more complex, curved relationships between points. Before running it, I randomly selected 5000 rows from the dataset to keep the computation manageable. Then I used Isomap to project the data into two dimensions, while trying to keep the local structure between points intact. Figure 18 shows the result of that 2D projection, where each point is coloured based on the actual price of the vehicle it represents. What stood out was how the prices didn't appear randomly spread, in fact they formed visible bands or zones across the plot. This suggests to us that the data naturally falls into regions that correspond to different price levels, even though the Isomap algorithm wasn't actually given any price information. That's encouraging, because it means the features in the dataset contain real structure that's linked to the target variable.

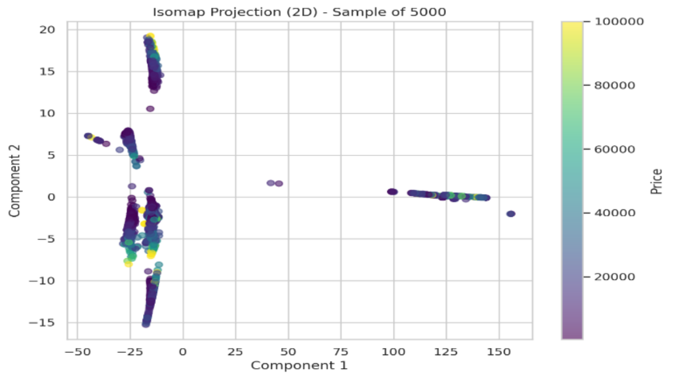


Figure 18*: shows Isomap projection (2D)*

I then applied **K-Means clustering** to the 2D projection. I chose to group the data into five clusters, which felt like a good balance between simplicity and detail. Figure 19 shows how these clusters were divided, and again, the result lined up well with the price-based patterns from earlier. Each cluster covered a specific region of the plot, suggesting that the clustering found meaningful groupings. I also checked the **silhouette score**, a measure of how distinct the clusters are, and got a score of **0.77**, which is quite high. That means the points in each cluster were close together, and different clusters stayed well separated.

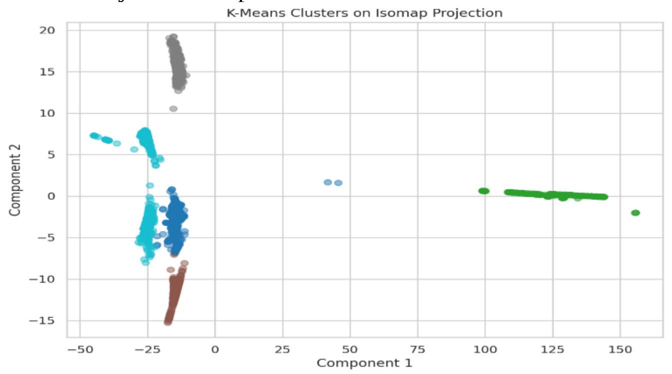


Figure 19*: shows K-Means clustering applied to Isomap 2D projection (k = 5)*

Overall, this process showed that even after reducing the dataset to just two dimensions, the structure in the data remained strong. Differences in price weren't random that they followed smooth, clustered shifts that matched what the model had been learning all along. It reinforced that the features used in this project were informative and captured something real and natrual about the vehicles being priced.

IX. Polynomial Regression

To see whether adding some simple non-linear effects could improve the model's ability to predict price, I tested both a regular linear regression and a polynomial regression using three of the most important continuous features: log_mileage, vehicle_age_years, and MMBF_target_encoded. These were chosen based on earlier results. As they consistently showed a strong relationship with the target variable and as a result were well-suited to this kind of modelling.

The basic linear regression assumes that the relationship between each input and the predicted price is straight for example, that every extra mile on the car lowers the price by the same amount, no matter what. Using this approach, the model achieved a Mean Absolute Error (MAE) of **3884**, a Root Mean Squared Error (RMSE) of **6302**, and an R^2 score of **0.825**, which means it explained about 82.5% of the variation in price.

Next, I used a polynomial regression with degree 2, which adds extra features based on squared values and interactions between the original inputs. This lets the model capture curved trends for example, that the effect of mileage might get worse at higher values, or that mileage and age together might have a different effect than either on their own. With these extra terms, the model improved noticeably: the MAE dropped to **3514**, the RMSE to **5703**, and the R^2 rose to **0.857**. So overall, it was able to explain more of the variation in price and made smaller errors on average.

These results support what earlier plots and model explanations suggested that the relationship between price and features like mileage or age isn’t purely linear. Adding some non-linearity helped the model better reflect how price actually behaves in the data, even without switching to more complex methods like decision trees.

X. Clustering For Feature Engineering

To explore whether clustering could help improve model performance, same as earlier(Figure 19) I used K-Means to group vehicles into five clusters based on the full set of scaled features. The idea here was to see if patterns in the data not tied directly to price could still carry useful information about vehicle type, condition, or configuration. If so, a model might benefit from knowing which "cluster" a car belongs to, as an extra feature during prediction.

After generating cluster labels using K-Means, I added them to the dataset as a new categorical feature. Then, I trained two separate Random Forest models for comparison. The first model used all features **except** the cluster label, while the second included the cluster labels alongside everything else. Both models were trained and validated using the same split, and evaluated using the same metrics: MAE, RMSE, and R^2.

Interestingly, the results were nearly identical. The model **without** the cluster label achieved an MAE of **2167.92** and R^2 of **0.932**, while the model **with** the cluster feature performed almost the same, with an MAE of **2170.01** and the same R^2. This suggests that the clustering didn’t introduce any useful new signal at least not beyond what the model was already learning from the raw features.

This outcome makes sense in context. Random Forest is already good at handling complex, non-linear patterns and combinations of features. Since the clustering was based on the same input data, it didn’t really provide anything the model couldn’t figure out on its own. Still, the exercise was worth doing as it confirmed that while clustering revealed some structure (as seen earlier with Isomap), that structure didn’t add predictive power in this particular case.