Preliminary Task

1. Data Description and Pre-Processing

Based on the data exploration that has been done on the course works of “Principles of Data Science” and “Machine Learning Concepts” modules, we got a thorough overview of the data.
In the Initial Analysis, we got this insight about these key features listed below:

- **year_of_registration** and **reg_code**: They can be inferred from each other by using a mapping dictionary which is defined based on the UK Vehicle Age Identifiers.
- **mileage**: This numerical feature is right-skewed, shows a strong negative correlation with price, making it a key target predictor.
- **vehicle_conditions**: This binary categorical feature, only is assigned “NEW” value in case of having null values both in year_of_registration and reg_code features.

In the Data Processing section of the project, several procedures were applied to make the datasets prepared to build the models on them. A summary of these steps is grouped into the following categories:

- **Handling Missing Values, Outliers and Noise**: Techniques such as filling values based on other features, capping, dropping the records, and imputation via grouping procedures are applied on the data records.

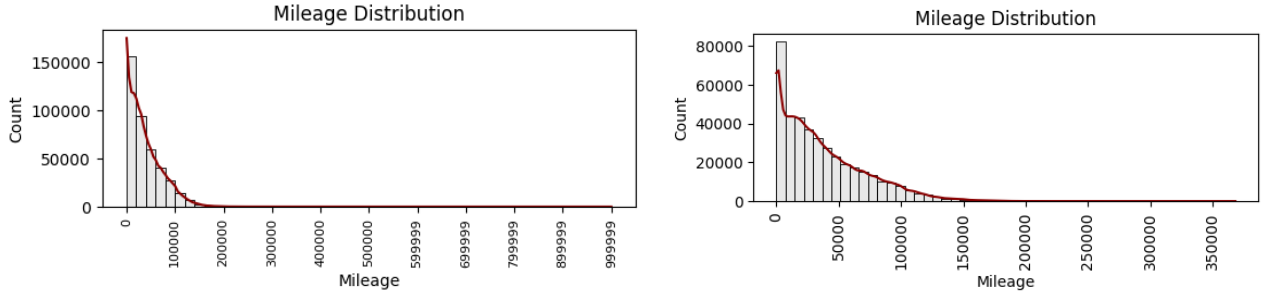
- **Feature Engineering, Data Transformation, and Feature Selection**: Implementing binning method, applying log transform method, and evaluating the features with higher association with the target price by using statistical testing and correlation function among the features.

	mileage	year_of_registration	price
mileage	1.000000	-0.861406	-0.645275
year_of_registration	-0.861406	1.000000	0.704944
price	-0.645275	0.704944	1.000000

table1. Correlation of numerical features with the target

After these steps, before building models other procedures are also have been done on the data such as encoding and scaling.
Mileage is one of the features that required multiple pre-processing approaches including:

- Filling missing values using group median based on standard_make, standard_model, and fuel_type
- Capping extreme values using the IQR method
- And eventually applying log transformation for better model performance and interpretability



plot1. An Example of Features Distribution Before and After Data Pre-processing

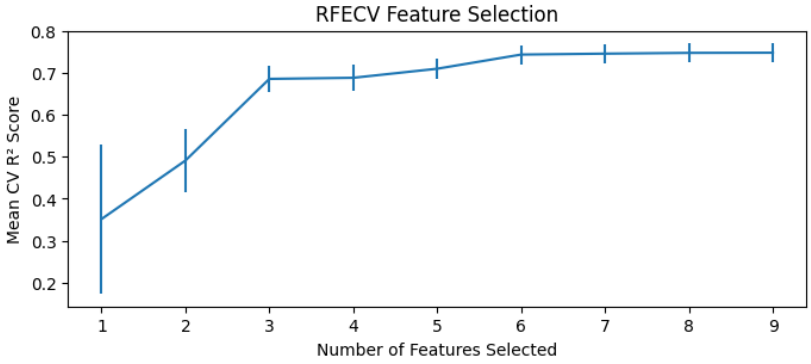
While using the entire dataset would likely provide more accurate and robust results, for reducing the computational costs and speeding the process up, a random 10% sample of the dataset is used for all experiments from this part.

Part I

2. Automated Feature Selection

Feature selection is performed using Recursive Feature Elimination with Cross-validation(RFECV) applied to a linear regression model. The reason behind preferring this method over other methods such as SelectKBest is due to the reason that it doesn't require a predefined number if features and also it combines with cross-validation which enhance the robustness of the feature selection process. It starts with minimum number of features required which is by default equal to 1 and determines the strategy of cross-validation splitting. Then, it will automatically tune the number of selected features by fitting an RFECV selector in different cross-validation splits.

Plot.1 illustrates the relationship between the number of features selected and the model's cross-validated R^2 score. It also displays that going from 1 feature to 3 features causes a huge improvement in R^2 score, meaning the first three features are the most important ones. According to the plot, the optimal feature number is 6, the model achieves a good balance of performance by having the highest R^2 score and stability by having the lowest standard deviation.



plot2. RFECV Feature Selection

	Number of Features Selected	Mean CV R² Score	Standard Deviation
0	1	0.3517	0.1784
1	2	0.4912	0.0752
2	3	0.6858	0.0316
3	4	0.6884	0.0307
4	5	0.7100	0.0242
5	6	0.7438	0.0228
6	7	0.7460	0.0225
7	8	0.7478	0.0224
8	9	0.7481	0.0223

output1. RFEVC Results

All 9 features were selected as optimal features based on the ranking of them which provided in the table1 which means that we need to keep all the 9 features due to their well contribution to the model performance.

```
1 rfecv.ranking_
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1])
```

code snippet1. RFECV Ranking Results (A)

	0
standard_colour	1
standard_make	1
standard_model	1
year_of_registration	1
body_type	1
fuel_type	1
log_mileage	1
vehicle_condition_NEW	1
crossover_car_and_van_True	1

dtype: int64

output2. RFECV Ranking Results (B)

3. Tree Ensembles

In this section, two ensemble models are applied on the dataset. All the features are used in this process because the RFECV method identified all of them as important.

3.1. Random Forests

This model uses grid search with 5-fold cross-validation to tune hyper parameters such as number of trees, maximum depth of each tree, and minimum samples to split an internal node. The best model best_rf is selected based on the highest cross-validated R^2 score.

```python
# --- Random Forest ---
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
}
rf = RandomForestRegressor(random_state=42)

rf_grid = GridSearchCV(rf, rf_params, cv=5, scoring='r2', n_jobs=-1)

start = time.time()
rf_grid.fit(X_train_scaled, y_train)
end = time.time()

rf_time = end - start

best_rf = rf_grid.best_estimator_
rf_preds = best_rf.predict(X_test_scaled)

```

code snippet2. Random Forest Model Hyperparameters

```python
print("Random Forest")
print("Best Params:", rf_grid.best_params_)
print("R² Test Score:", r2_score(y_test, rf_preds))
print("MAE Test:", mean_absolute_error(y_test, rf_preds))

print(f"\nTraining time: {rf_time:.2f} seconds")
```

```
Random Forest
Best Params: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 200}
R² Test Score: 0.8820153421031787
MAE Test: 0.18615069839385084

Training time: 180.00 seconds
```

```python
print("Train R²:", best_rf.score(X_train_scaled, y_train))
print("Test  R²:", best_rf.score(X_test_scaled, y_test))
```

```
Train R²: 0.9876055309321063
Test  R²: 0.8820153421031787
```

code snippet3. Random Forests Results

The Random Forest shows overfitting due to the better performance of the model on the training set with score of 0.98 in comparison to the test set with score of 0.88.

3.2. Gradient Boosting

This model uses grid search with 5-fold cross-validation to tune hyperparameters such as number of trees which describes the boosting stages, the contribution of each tree, and maximum depth of the individual regression estimator. The best-performing model best_gb is selected based on the highest average cross-validated R^2 score.

```python
# --- Gradient Boosting ---
gb_params = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5],
}
gb = GradientBoostingRegressor(random_state=42)

gb_grid = GridSearchCV(gb, gb_params, cv=5, scoring='r2', n_jobs=-1)

start = time.time()
gb_grid.fit(X_train_scaled, y_train)
end = time.time()

gb_time = end - start

best_gb = gb_grid.best_estimator_
gb_preds = best_gb.predict(X_test_scaled)

```

code snippet4. Gradient Boosting Model Hyperparameters

```python
print("\nGradient Boosting")
print("Best Params:", gb_grid.best_params_)
print("R² Test Score:", r2_score(y_test, gb_preds))
print("MAE Test:", mean_absolute_error(y_test, gb_preds))

print(f"\nTraining time: {gb_time:.2f} seconds")
```

```
Gradient Boosting
Best Params: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
R² Test Score: 0.9111622701782363
MAE Test: 0.17048728765248614

Training time: 73.51 seconds
```

```python
print("Train R²:", best_gb.score(X_train_scaled, y_train))
print("Test  R²:", best_gb.score(X_test_scaled, y_test))
```

```
Train R²: 0.9623385223837213
Test  R²: 0.9111622701782363
```

code snippet5. Gradient Boosted Trees Results

The Gradient Boosting model shows better generalising due to the gap between performance of the test (0.98) and train (0.88) sets as the model maintains consistent performance across both sets. While Random Forest shows stronger training performance, Gradient Boosting model offers a better balance between bias and variance regarding to the gap between test (0.91) and train (0.96) sets, making it preferred choice for deployment due to its generalisability and robustness.

4. Ensemble of Tree Ensembles

In this section, two methods of combining ensembles models are applied on the dataset. All the features are used in this process because the RFECV method identified all of them important.

4.1. Voting Method

Voting/ Averaging method takes the average of predictions from multiple base models which are random forests and gradient boosting in our case.

```python
# --- Averaging Ensemble (VotingRegressor) ---
averaging_ensemble = VotingRegressor(
    estimators=[('rf', best_rf), ('gb', best_gb)], n_jobs=-1
)

start = time.time()
averaging_ensemble.fit(X_train_scaled, y_train)
end = time.time()

avg_time = end - start

avg_preds = averaging_ensemble.predict(X_test_scaled)

```

code snippet6. Voting Model Hyperparameters

```python
print("\nAveraging Ensemble")
print("R² Test Score:", r2_score(y_test, avg_preds))
print("MAE Test:", mean_absolute_error(y_test, avg_preds))
print(f"\nTraining time: {avg_time:.2f} seconds")
```

```
Averaging Ensemble
R² Test Score: 0.9030236895631905
MAE Test: 0.17339995324457572

Training time: 13.68 seconds
```

```python
print("Train R²:", averaging_ensemble.score(X_train_scaled, y_train))
print("Test  R²:", averaging_ensemble.score(X_test_scaled, y_test))
```

```
Train R²: 0.9790620901342619
Test  R²: 0.9030236895631905
```

code snippet7. Voting Method Results

The Voting model achieved good performance with a high train R^2 of 0.97 and test R^2 of 0.90, which shows that the model effectively fits the training data and generalizes well.

4.2. Stacking Method

Stacking method feeds the predictions of multiple base models into a higher-level model, which in our case this model is ridge regression; then it combines the base models with each other to obtain the final prediction.

```python
# --- Stacking Ensemble ---
stacking_ensemble = StackingRegressor(
    estimators=[('rf', best_rf), ('gb', best_gb)],
    final_estimator=Ridge(random_state=42),
    cv=5,
    n_jobs=-1
)

start = time.time()
stacking_ensemble.fit(X_train_scaled, y_train)
end = time.time()

stack_time = end - start

stack_preds = stacking_ensemble.predict(X_test_scaled)

```

code snippet8. Stacking Model Hyperparameters

```python
print("\nStacking Ensemble")
print("R² Test Score:", r2_score(y_test, stack_preds))
print("MAE Test:", mean_absolute_error(y_test, stack_preds))
print(f"\nTraining time: {stack_time:.2f} seconds")
```

```
Stacking Ensemble
R² Test Score: 0.9102741984924191
MAE Test: 0.17026653309841713

Training time: 51.73 seconds
```

```python
print("Train R²:", stacking_ensemble.score(X_train_scaled, y_train))
print("Test  R²:", stacking_ensemble.score(X_test_scaled, y_test))
```

```
Train R²: 0.9690367717002502
Test  R²: 0.9102741984924191
```

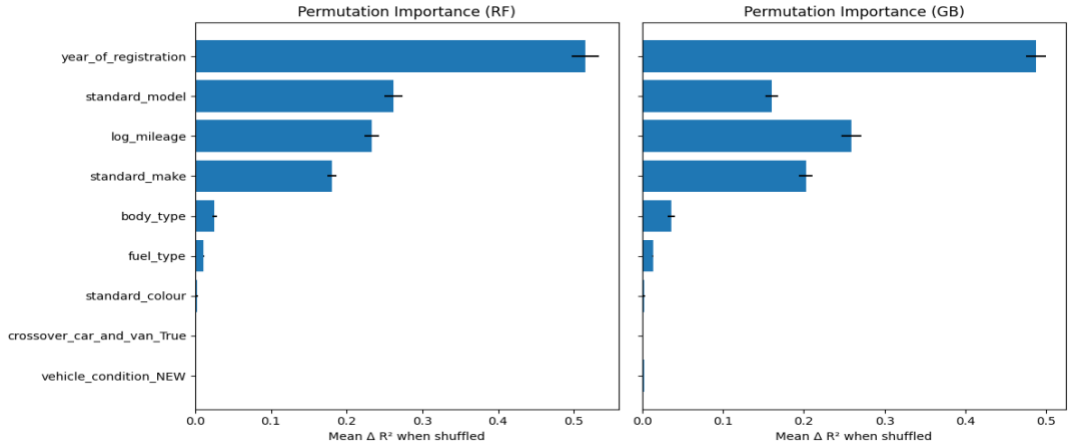
code snippet9. Stacking Method Results

The Stacking model also achieved good performance with a train score of 0.96 and test score of 0.91, indicating that it generalised well without overfitting, as it shows a balance between train and test performance.

The Voting method is faster to train; however, it shows a slightly larger gap between training and test scores that suggests a mild tendency to overfitting. In contrast, the Stacking method suggests a more balanced fit between training and test sets. Although it requires more training time, it's a more suitable choice when accuracy and robustness are the top priorities.

5. Feature Importance

In this section, permutation importance is applied to Random Forest and Gradient Boosting models, which evaluates the contribution of each feature to the model's predictive performance. Both plots indicate that feature year_of_registration is the most effective feature as it causes the greatest reduction in R^2 when shuffled, followed by standard_model, standard_make, and log_mileage suggesting that both models rely on a few key predictive features.



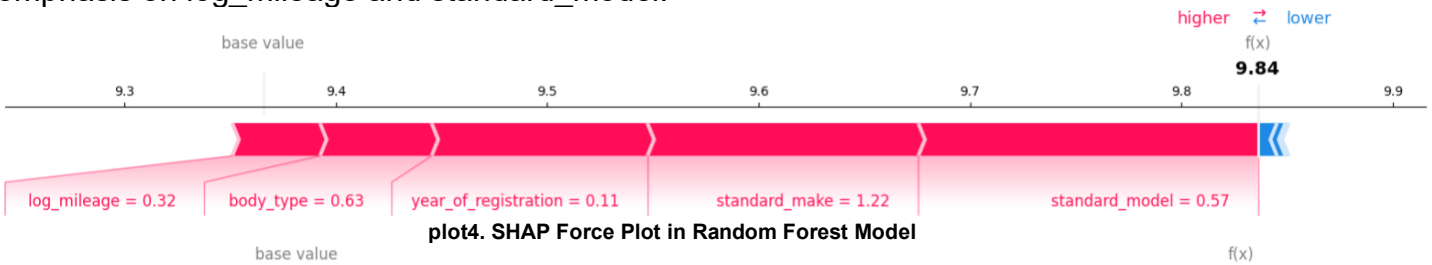
plot3. Permutation Feature Importance in Random Forest and Gradient Boosting Models

6. SHAP/PDP Model Explanations

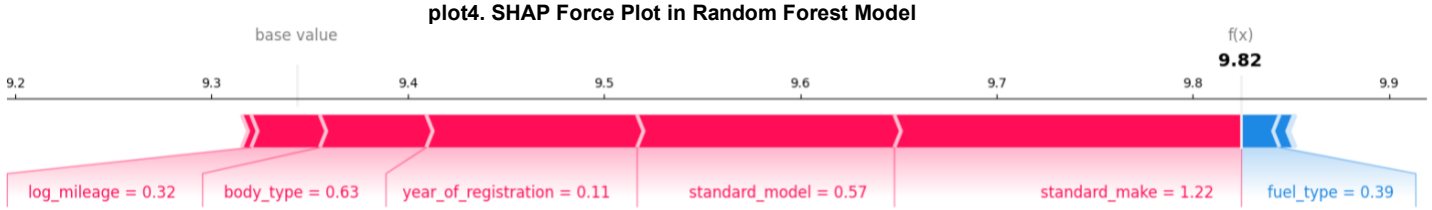
SHAP values provide a structured approach to assign a numerical value to each feature and showcase how much that particular feature influenced the model's prediction for a specific data point.

6.1. Local SHAP: Explain the model's prediction for a specific instance or a small subset of instances

- SHAP Force plots below visualise how individual features contributed to the model predictions for the first record in the test set, highlighting the effect of each feature on the final output. Both models rely strongly on the same features to make their predictions. However, Gradient Boosting shows a more negative contribution from the fuel_type, while Random Forest places slightly more emphasis on log_mileage and standard_model.



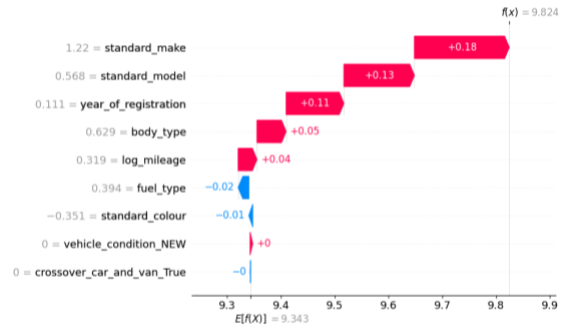
plot4. SHAP Force Plot in Random Forest Model



plot5. SHAP Force Plot in Gradient Boosting Model

- SHAP Waterfall plots below illustrates how the prediction for the first record in the test set was built up from the base value, showing the cumulative impact of each feature on the final models' output. Both models rely strongly on the same features of standard_model, standard_make, and year_of_registration with minor differences in the feature contributions' magnitude.

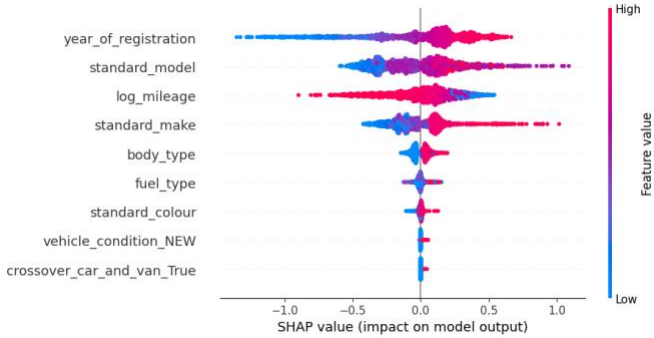
plot6. SHAP Waterfall Plot in Random Forest Model



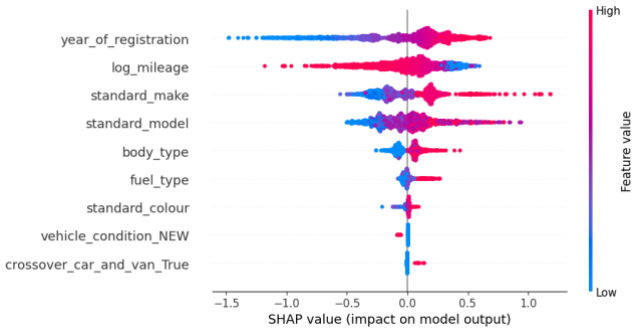
plot7. SHAP Waterfall Plot in Gradient Boosting Model

6.2. Global SHAP: Provide an overview of the model's behaviour across the entire dataset

SHAP Beeswarm plots show how each feature affects model prediction across the test set highlighting overall importance and impact direction. Both models agree on the year_of_registration and log_mileage contribution but differ slightly differ in how they weigh others like standard_make and standard_model. Gradient Boost shows stronger non-linear relationships between the features due to observing wider spread of SHAP values and colour gradient overlaps. Random Forests in the other hand, shows a tighter concentration of SHAP values around a narrower range, reflecting a more linear behavior compared to the more flexible, nonlinear interactions captured by Gradient Boosting



plot8. Global SHAP Plot in Random Forests Model



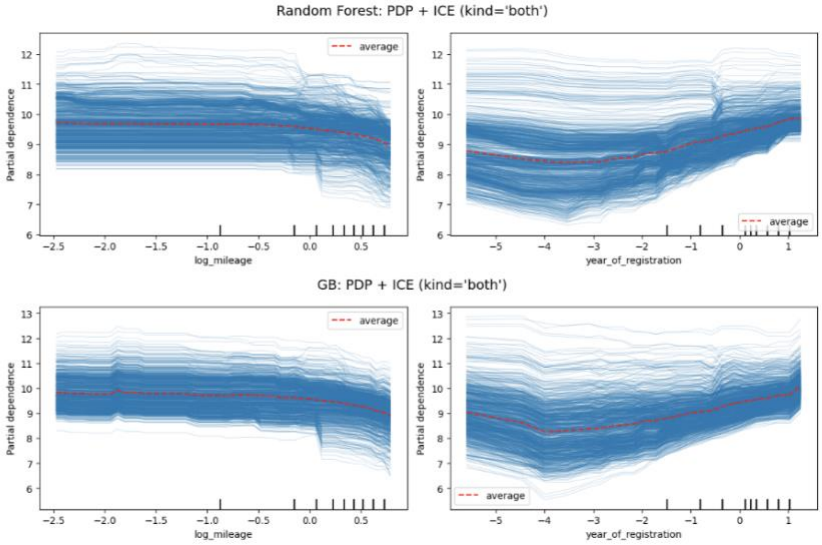
plot9. Global SHAP Plot in Gradient Boosting Model

6.3. PDP and ICE: Provide insights into how the model's predictions change as a particular feature varies while holding other features constant.

In this section, two features of log_mileage and year_of_registration are selected because they appeared among the high-impact features, and are continuous, and interpretable.

In both Random Forests and Gradient Boosting, as mileage increases, the average trend (PDP) shows a slight decrease in prediction. However Gradient Boosting shows more variation across ICE curves, indicating stronger interactions.

Both models show an upward PDP trend for year_of_registration, but Gradient Boosting ccaptures a clearer nonlinear pattern with greater ICE variability, while Random Forests displays a more stable and consistent response.



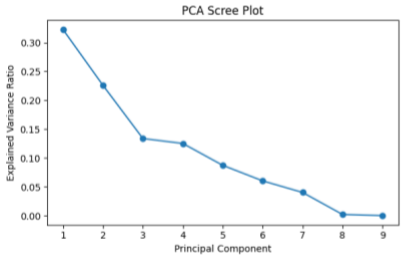
plot10. PDP and ICE Plot for log_mileage and year_of_registration in Random Forests and Gradient Boosting Models

Part II

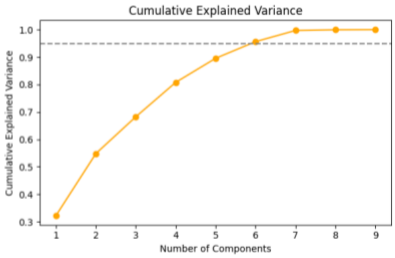
7. Dimensionality Reduction, Linear

PCA as a linear dimensionality reduction tool, is used to extract information from a high-dimensional space by projecting it into a lower-dimensional subspace.

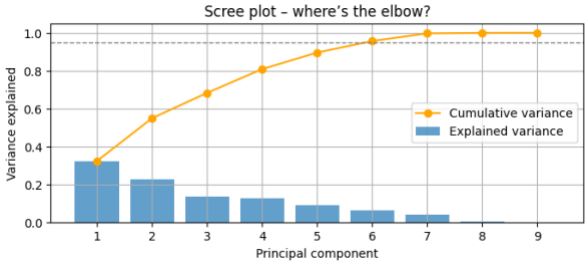
The scree plot displays the explained variance ratio for each principal component, and its elbow shape indicates the first few components (first 3-6 PCs) explain most of the variance and regarding to the Cumulative Explained Variance illustrates that 6 components are sufficient to retain at least 95% of the variance, making them a suitable choice for dimensionality reduction while maintain predictive information.



plot11. PCA Scree plot



plot12. Cumulative Explained Variance



plot13. Explained and Cumulative Variance

To evaluate the impact of PCA on model performance, all the models that were built, in addition to linear regression were trained and tested before and after applying PCA.

Model	Stage	Train Time (s)	Train R^2	Test R^2	Train MAE	Test MAE
Random Forest	Before PCA	11.1102	0.9876	0.8820	0.0652	0.1862
	After PCA	20.1113	0.9772	0.8495	0.0862	0.2311
Gradient Boosting	Before PCA	5.5924	0.9623	0.9112	0.1234	0.1705
	After PCA	8.4442	0.9247	0.8410	0.1746	0.2396

output3. Random Forests and Gradient Boosting Models Performance before and after applying PCA

Ensemble	Stage	Train Time (s)	Train R^2	Test R^2	Train MAE	Test MAE
Voting	Before PCA	15.9741	0.9791	0.9030	0.0910	0.1734
	After PCA	22.6041	0.9580	0.8503	0.1279	0.2313
Stacking	Before PCA	42.1814	0.9688	0.9103	0.1123	0.1703
	After PCA	88.8547	0.9697	0.8513	0.1059	0.2302

output4. Voting and Stacking Models Performance before and after applying PCA

Regarding to output3, applying PCA reduces the model performance for both Random Forests and Gradient Boosting. The difference between Train R^2 and Test R^2 is increased, indicating a drop in generalisation. Training time also increased slightly after this process. Although, PCA simplified the input space, it also led to information loss and had a negative impact on prediction accuracy of both models. In output4, we can also observe the information loss and how the performance is negatively affected. The training time also increased, especially in the Stacking model, which it doubled (~42s to ~98s).

Stage	Train Time (s)	Train R^2	Test R^2	Train MAE	Test MAE
Before PCA	0.0064	0.7493	0.7916	0.2810	0.2758
After PCA	0.0031	0.7168	0.7634	0.2963	0.2905

output5. Linear Regression Model Performance before and after applying PCA

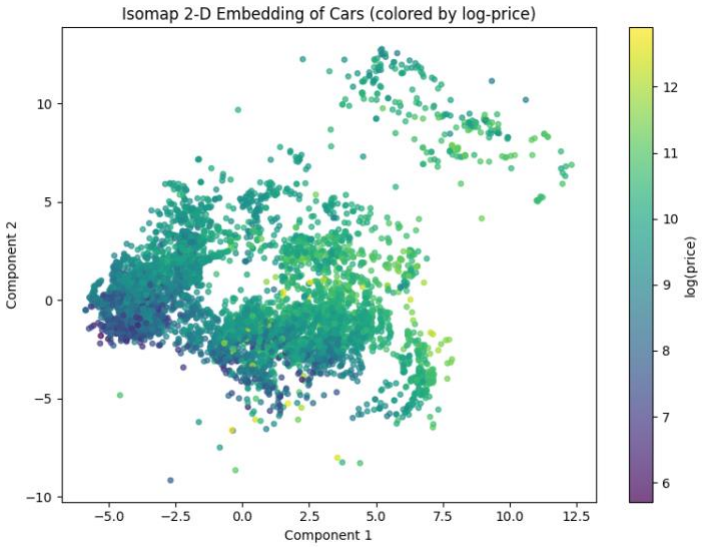
In contrast to other models, for Linear Regression, we can observe a slight decrease in Test R^2 and a small increase in Test MAE. Training time is also reduced by half.

In conclusion, PCA hurt the ensemble models both in predictive power and in training speed. Applying PCA led to a consistent drop in model performance across all ensemble methods, with lower Test R^2 and higher MAE indicating reduced predictive accuracy.

8. Dimensionality Reduction, Non-Linear

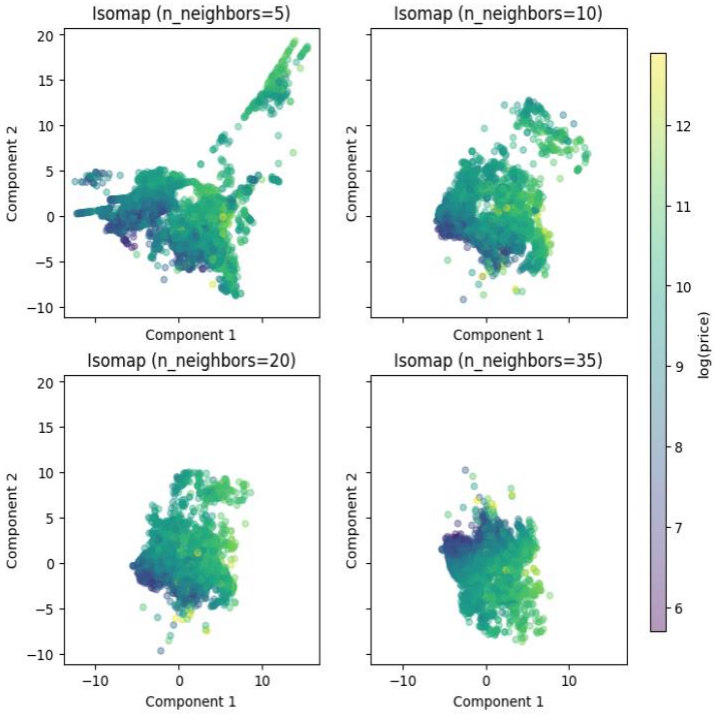
Isomap is a non-linear dimensionality reduction technique that aims to preserve the intrinsic geometric structure of high-dimensional data in a lower-dimensional space.

The Isomap plot below, with n_neighbors=10 and n_components=2, shows how cars group based on non-linear combinations of their features, which is coloured by the logarithm of the car prices. While the Isomap plot reveals non-linear structure and price-related trends, it does not clearly separate distinct cluster or indicate which features drive the embedding.



plot14. Isomap 2D Projection Colored by log(price)
Lower-priced vehicles are displayed (darker data points) and are more densely clustered in specific regions, while higher-priced vehicles (brighter data points) appear more dispersed across the embedding.

These Isomap plots below, illustrate the impact of varying the n_neighbors hyperparameter on the Isomap embedding of the data, where each data point is a vehicle and is coloured according to its log(price). By comparing embeddings across different values, we can assess the trade-off between capturing local variance and preserving the overall structure of the dataset.



plot15. Comparison of Isomap Embeddings with Different Neighbourhood Sizes
Smaller neighbours values (e.g., 5) focus too much on local relationships between nearby points. As a result, the global shape of the data may get broken into disconnected pieces in the plot, while larger values (e.g., 35) prioritise global relationships at the expense of local variation.

For instance, at n_neighbors=35, the 2D embedding becomes tightly compressed, with less visible structure. It makes it hard to tell where the high-priced and low-priced vehicles separate, and the data points look have overlaps.

9. Polynomial Regression

Polynomial Regression is a form of regression analysis in which the relationship between the independent variables and dependent variables are modelled in the nth degree polynomial.

In this experiment, two key features of year_of_registrayion and log_mileage are selected to use in both linear and polynomial regression. Based on the results shown below, Polynomial Regression performed better in both accuracy and explanatory power.

Model	Train R^2	Test R^2	Train MAE	Test MAE
Linear Regression	0.4660	0.5518	0.4585	0.4362
Polynomial Regression (deg=2)	0.5605	0.5989	0.4361	0.4164

output6. Comparison of Linear Regression and Polynomial Regression Performance

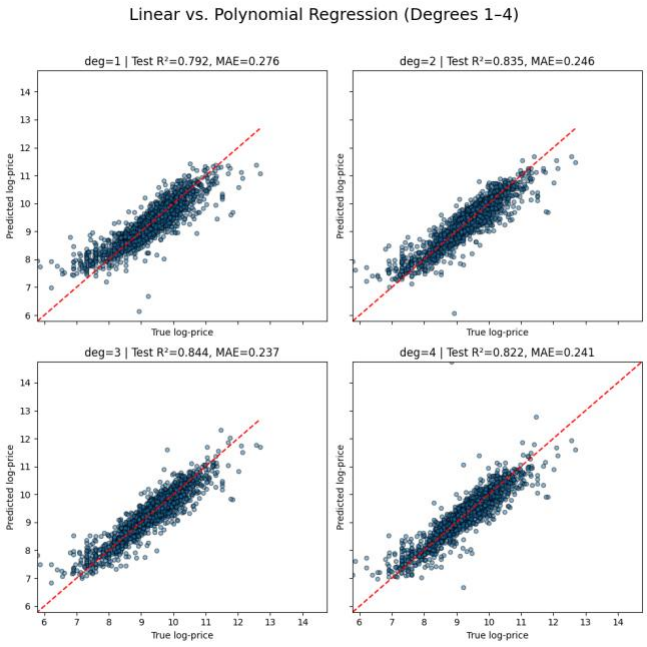
To explore the effectiveness of Polynomial Regression for predicting the target, models of degree 1 through 4 are trained using a pipeline that expands the input features with polynomial terms before fitting a linear regression model. As the polynomial degree increased, the models captured more complex, non-linear relationships, leading to improved performance up to degree 3.

Degree	Train R^2	Test R^2	Train MAE	Test MAE
1	0.7493	0.7916	0.2810	0.2758
2	0.8219	0.8348	0.2453	0.2457
3	0.8499	0.8437	0.2280	0.2374
4	0.8713	0.8219	0.2142	0.2406

output7. Comparison of Polynomial Regression Model Performance

The degree 3 model indicates the best balance by achieving the highest Test R^2 and the lowest test MAE and also the rational difference between Train R^2 and Test R^2.

While the degree 4 model continued to improve on the training data, it showed a slight drop in test performance, suggesting mild overfitting.



plot16. Polynomial Regression Predictions (Degrees 1-4)

The plots confirm that increasing polynomial degree improves prediction accuracy up to a point (degree 3). Degree 3 model shows that the predicted values closely are matched the actual values due the tight packed dots around the red dashed line which shows where predicted values= actual values.

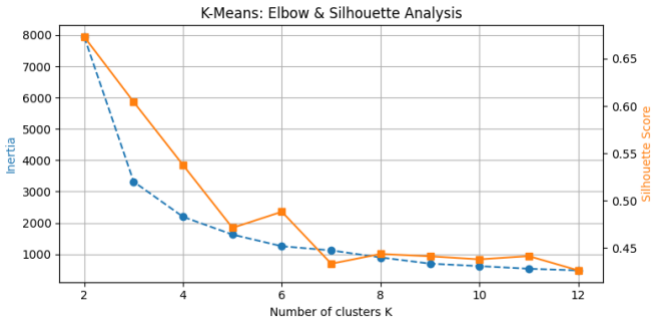
10. Clustering for Feature Engineering

By applying these unsupervised learning algorithms, clusters within the data can be identified and then they can be used as new features. These features can help models to learn group-specific patterns that might not be evident when using the raw features alone.

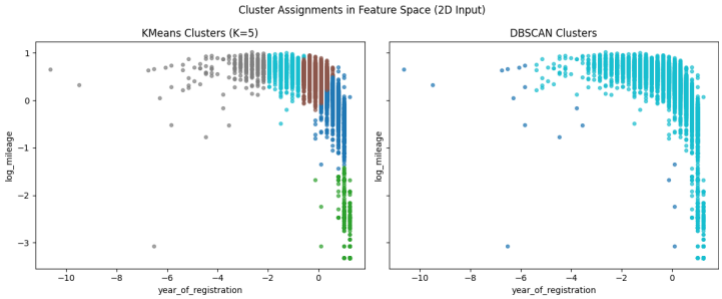
In this section, the goal is to determine the most appropriate number of clusters (k) to use for the KMeans Clustering. Selecting the right k helps us in the process of capturing the meaningful structure in the data. Two key metrics are used in this procedure:

- Inertia: Measures how compact the clusters are. (The lower the inertia, the better the model)
- Silhouette Score: Evaluates how well data points fit within their assigned cluster. (A higher score indicates a better clustering result)

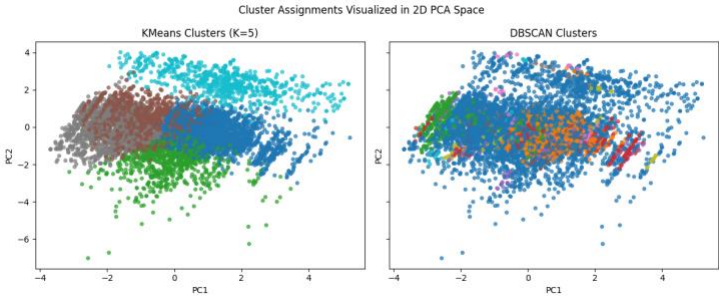
In our case, the input to the KMeans is limited to two key features of log_mileage and year_of_registration that are selected to ensure the clustering focused on a meaningful and interpretable structure. Eventually, k=5 is chosen because of displaying a sharp drop in Inertia and a relatively high Silhouette Score, suggesting well-formed and distinct clusters



plot17. K Selection (Elbow and Silhouette)



plot18. Comparison of KMeans and DNSCAN Clustering (k=5)
Compares the KMeans and DBSCAN Clustering results based on only two features of year_of_registration and log_mileage.
Because the year_of_registration is discrete, even after scaling, the data appears in vertical bands, causing KMeans to form column-like clusters and limiting DBSCAN's ability to detect meaningful density-based groupings.



plot19. Comparison of KMeans and DNSCAN Clustering (k=5)
Applies Clustering using all scaled features and then projects the results into 2D using PCA for visualisation. Kmeans produces better-separated clusters than DBSCAN, while DBSCAN identifies multiple irregular-shaped groups and outliers.
However, even in this plot, some clusters appear to overlap or lack sharp separation.