Advanced Machine Learning
Name: Farrukh Nizam Arain

1. Data Processing for Machine Learning

During the data processing stage following exploration, the focus was on refining raw data for machine learning. Some anomalies in the dataset, like outliers and missing values, were identified and rectified. This not only ensured the dataset was well-prepared, but also preserved 99.97% of the data for further analysis.

a) Checking for Possible Duplicates:

In the process of ensuring integrity of the dataset, a thorough check was conducted to find any duplicates.

```
duplicate_rows = df[df.duplicated()]
print("Number of duplicate rows: ", duplicate_rows.shape)
```

b) Dealing with Outliers in Vehicle Registration Year and Related Features:

During the inspection of the vehicle registration year column, rows with dates equal to or before 1909 were deleted. This was done as the launch year of some vehicles did not align with their model year, or because vehicles had not yet invented at that time.

```
df.query('year_of_registration <= 1909.0')
df.drop(df[df['year_of_registration'] <= 1909.0].index, inplace=True)
```

Extreme values in the price and mileage features were filtered out using the same method mentioned above. This was necessary because such extreme values negatively affect the model's performance by skewing the distribution. By retaining other high values, I preserved as much data as possible, which is crucial for building robust machine learning models.

c) Imputation Techniques for Handling Missing Values:

Missing values in categorical features were filled by calculating the mode based upon a unique combination of two existing features of the vehicle. For instance, the mode (most frequently occurring value) of 'standard_colour' for each unique combination of 'standard_make' and 'standard_model' was calculated. If the mode was empty (i.e., all values are NaN), it returned pd.NA. Otherwise, it filled the missing values.

```
standard_model_mode_series = df.groupby(['standard_make','standard_model'])['standard_colour']
.transform(lambda x: x.mode().iloc[0] if not x.mode().empty else pd.NA)
df['standard_colour']=df['standard_colour'].fillna(standard_model_mode_series)
```

Moreover, for features that comprised numerical data, a similar strategy was utilized, but instead of mode, mean (average) was calculated.

```
mileage_mean_series = df.groupby(['standard_make','standard_model'])['mileage']
.transform('mean')
df['mileage'] = df['mileage'].fillna(mileage_mean_series)
```

d) Dropping Irrelevant Columns

The 'public_reference' and 'reg_code' columns were dropped from the dataset because they are unique identifiers that do not contribute to car price prediction. Including such irrelevant features in machine learning models can lead to overfitting, as the model might learn from noise rather than meaningful relationships.

```
df = df.drop(['public_reference', 'reg_code'], axis=1)
df = df[['standard_make', 'standard_model', 'standard_colour', 'body_type','vehicle_condition',
'crossover_car_and_van', 'fuel_type', 'mileage','year_of_registration', 'price_category', 'price']]
```

2. Feature Engineering

A few modifications in the feature engineering stage aided in the later stages of the project.

a) Re-categorization of the Vehicle Condition Feature:

A re-classification based on mileage was formulated to overcome a lacked pattern. This task involved creating Boolean values to mask rows where 'vehicle_condition' was 'USED' and mileage was less than or equal to 100. Those 'USED' entries in the 'vehicle_condition' column were replaced with 'NEW' and updated accordingly.

```
mask = (df['vehicle_condition'] == 'USED') & (df['mileage'] <= 100)
x = df[mask].sort_values(by='mileage',ascending=False)['vehicle_condition'].replace('USED', 'NEW')
df.update(x)
```

b) Simplification and Enhanced Interpretability of Price Data:

A new column for binning prices for old and new vehicles and labels was created to reduce complexity, aid in handling limited samples for certain price ranges, and better interpretability. For instance, for used cars, assumed price ranges (bins) and their corresponding labels were saved in variables. A new feature 'price_category', used the 'pd.cut' function to categorize and update the price labels in that new feature.

```
price_bins = [0, 13999, 17999, 21999, 25999, 29999, 33999, 37999,float('inf')]
price_labels = ['Very Low', 'Low', 'Medium-Low', 'Medium', 'Medium-High', 'High', 'Very High', 'Luxury']
df['price_category'] = pd.cut(df.query('vehicle_condition == "USED"')['price'], bins=price_bins,
labels=price_labels, right=True)
```

c) Application of Polynomial/Basis Functions and Interaction Features:

Since polynomial/basis functions and interaction features are typically applied to numerical rather than categorical features, several steps were taken before using these functions. First, a smaller sample was extracted from the data frame for each 'year_of_registration' to ensure a representative sample for every year.

```
df = df.groupby('year_of_registration').sample(frac=0.02, random_state=5).reset_index().copy()
```

Following that, the target variable was separated from the features, and the data was divided into training and testing sets.

```
X = df.drop(columns='price')
y = df['price'],X_train, X_test, y_train, y_test = train_test_split(X, y, random_state= 5, test_size=0.25)
```

Target encoding was applied to a set of categorical variables in the dataset. One-Hot Encoding was avoided because it increased dimensionality, and Label Encoding did not yield good scores.

```
te_cols = ['standard_make', 'standard_model', 'standard_colour', 'body_type', 'fuel_type',
'crossover_car_and_van', 'vehicle_condition', 'price_category']
encoder = ce.TargetEncoder(cols=te_cols),encoder.fit(X_train, y_train)
X_train_encoded = encoder.transform(X_train),X_test_encoded = encoder.transform(X_test)
```

Two linear models were compared for prediction improvements, with and without polynomial features.

```
normal_reg = Pipeline(steps=[('est', LinearRegression())])
poly_reg = Pipeline(steps=[('poly', PolynomialFeatures(interaction_only=True, include_bias=False)),
        ('est', LinearRegression())])
normal_reg.fit(X_train_encoded, y_train), poly_reg.fit(X_train_encoded, y_train)
normal_reg.score(X_test_encoded, y_test), poly_reg['poly'].fit_transform(X_train_encoded)
```

Utilizing polynomial features within the linear model led to an increase in R-squared values from 74% to 84%, indicating a 10% improvement in prediction accuracy. However, this inclusion also expanded the dataset to fifty-five features, adding complexity to model interpretation. These transformed datasets were replaced to be used in the next stages of the project.

3. Feature Selection and Dimensionality Reduction:

Before dimensionality reduction, three automated feature selection models were tested and selected:

a) Sequential Feature Selection (SFS) (Forward/Backward):
A pipeline was created with two stages: feature selection and applying a linear regression model for assessing the performance of best parameters. A forward greedy search was used to add features until a stopping condition was met. Linear Regression assessed feature importance, and 5-fold cross-validation determined the optimal number of features automatically because they were not explicitly defined.

```
regr_sfs = Pipeline(steps=[("featsel", SequentialFeatureSelector(LinearRegression(),
n_features_to_select="auto", direction="forward", cv=5)),("regr", LinearRegression())])
regr_sfs.fit(X_train_poly, y_train), len(X_train_poly.columns[regr_sfs['featsel'].get_support()])
regr_sfs.score(X_train_poly, y_train), regr_sfs.score(X_test_poly, y_test)
```

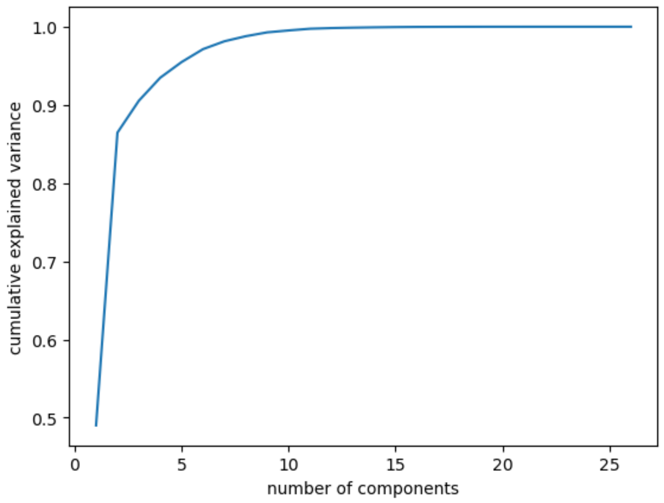
Model Name	No. of Features	Train Score	Test Score	Train & Test MSE
Select K-Best	45	86.7%	84.2%	3214 & 3225
RFECV	32	85.9%	82.8%	3226 & 3241
SFS	27	87.4%	84.0%	3207 & 3291

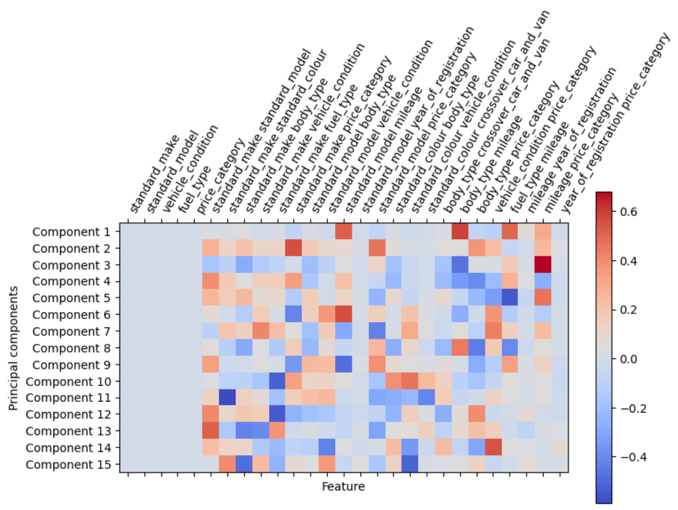
Sequential Feature Subtraction's datasets were used for PCA, yielding similar performance with fewer features.

For selecting the k value, a loop iterated over various numbers of components in the dataset before applying PCA, calculating explained variance, and comparing predictability via R-Squared and MAE values. Additionally, a plot visualized how total variance is explained with increasing components.

```
n_c = 15, pca = PCA(n_components=n_c), pca.fit(X_train_sfs)
X_train_pca = pca.transform(X_train_sfs), X_test_pca = pca.transform(X_test_sfs)
X_train_pca_df = pd.DataFrame(X_train_pca, columns=[f'PC{i+1}' for i in range(n_c)])
X_test_pca_df = pd.DataFrame(X_test_pca, columns=[f'PC{i+1}' for i in range(n_c)])
```

The number of components was chosen to be 15 because it captured 99% of the variance in the dataset, had a high test score of 83%, and the MAE values did not significantly increase with more components.





Summary of the contributions of the independent features on principal components are the following:

- **Mileage and vehicle condition:** They are critical factors when it comes to predicting price of the car, especially when interacting with body types and price categories to influence car prices.
- **Desirable makes and models:** consistently show positive impacts on car prices.
- **Fuel type and specific body types**: can both positively and negatively affect prices, depending on the combination with other factors.
- **Price categories:** have varying effects, with certain categories maintaining their value despite high mileage and less favourable body types.

4. Model Building:

Next, several single and ensemble models were analysed to select the most suitable algorithm for predicting the vehicle's price.

a) A Linear Model:

A linear model is a statistical tool for predicting a dependent variable based on the values of one or more independent variables. This model assumes a linear relationship between these variables. So, for predicting the target values, a pipeline was constructed for implementing the linear regression model. A parameter grid evaluated the impact of the 'fit_intercept' parameter through a 5-fold Grid Search CV to identify the optimal model based on the parameters of training, testing, and MAE values.

```
regr_pca = Pipeline(steps=[("regr", LinearRegression())])
param_grid = {"regr__fit_intercept": [True, False],}
grid_search = GridSearchCV(regr_pca, param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train_pca_df, y_train), best_params = grid_search.best_params_,
best_estimator = grid_search.best_estimator_, train_score = best_estimator.score(X_train_pca_df, y_train)
test_score = best_estimator.score(X_test_pca_df, y_test)
train_mae = mean_absolute_error(y_train, best_estimator.predict(X_train_pca_df))
test_mae = mean_absolute_error(y_test, best_estimator.predict(X_test_pca_df))
```

b) Random Forest Regressor:

Random Forest Regressor is a machine learning algorithm that uses multiple decision trees to predict a continuous outcome. For implementation of this algorithm, a pipeline for its execution was created. A parameter grid evaluated its performance with the number of trees in the forest and maximum depth because deeper tree can capture more information of the data. These parameters were used in a 5-Fold Grid Search CV to calculate best parameters, train, test, and MAE scores.

```
rfr = Pipeline(steps=[("regr", RandomForestRegressor())])
param_grid = {"regr__n_estimators": [10, 50, 100],"regr__max_depth": [None, 10, 20],}
grid_search = GridSearchCV(rfr, param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train_pca_df, y_train), best_params = grid_search.best_params_,
best_estimator = grid_search.best_estimator_, train_score = best_estimator.score(X_train_pca_df, y_train)
test_score = best_estimator.score(X_test_pca_df, y_test)
train_mae = mean_absolute_error(y_train, best_estimator.predict(X_train_pca_df))
test_mae = mean_absolute_error(y_test, best_estimator.predict(X_test_pca_df))
```

c) Boosted Tree Regressor:

Gradient Boosting Regressor involves building multiple decision trees in a sequential manner in such a way that each tree is trained to correct the mistakes of previous trees, hence it is referred a 'boosting.' For the application, a pipeline was created, and same parameter grid was utilized of number of trees and maximum depth. These parameters were used in a 5-fold Grid Seach CV, and best parameters, train, test, and MAE scores were calculated.

```
gb = Pipeline(steps=[("gb", GradientBoostingRegressor())])
param_grid = {"gb__n_estimators": [10, 50, 100],"gb__max_depth": [None, 10, 20],}
grid_search = GridSearchCV(gb, param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train_pca_df, y_train), best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_, train_score = best_estimator.score(X_train_pca_df, y_train)
test_score = best_estimator.score(X_test_pca_df, y_test)
train_mae = mean_absolute_error(y_train, best_estimator.predict(X_train_pca_df))
test_mae = mean_absolute_error(y_test, best_estimator.predict(X_test_pca_df))
```

The following results pertaining to the single models were calculated.

Name	Train Score	Test Score	Train & Test MAE Scores
Linear Regression	86.1%	82.5%	3589 and 3668

Random Forest Regression	98.5%	88.0%	897 and 2379
Gradient Boosting Regressor	99.7%	73.9%	666 and 2586

Random Forest Regressor emerged as the top single model, showcasing balanced accuracy, lower overfitting, and superior performance on unseen data compared to other models, based on train, test and MAE scores.

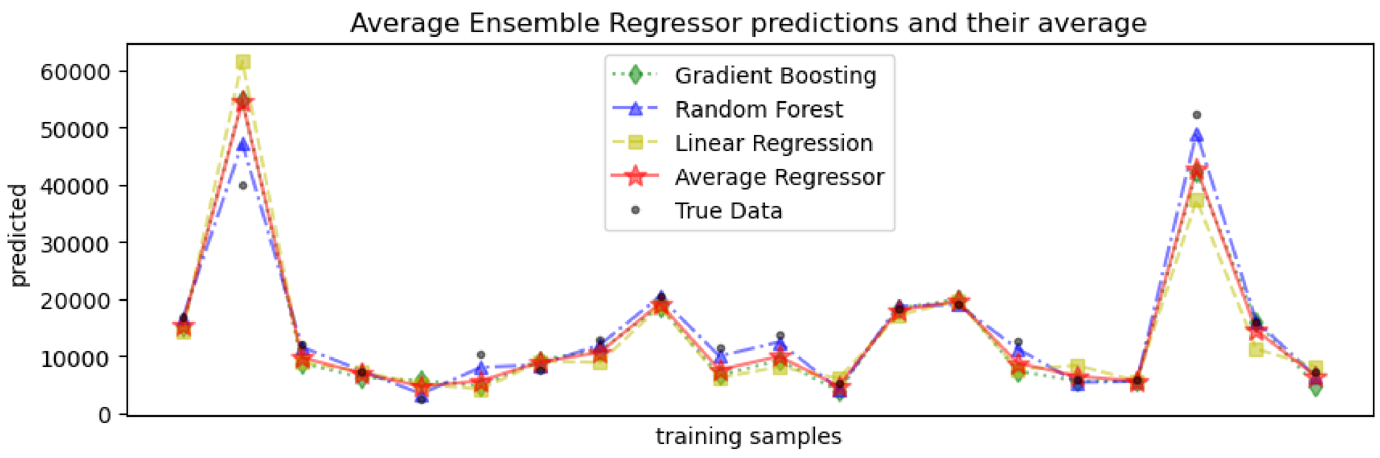
d) Average/Voter/Stacker Ensemble:

To enhance the predictive performance of the previous models, average, voting, and stacking ensemble techniques were adopted, combining the strengths of them to construct a more robust and accurate learner.

- **Average Ensemble:**

This technique involves training several models independently and averaging their scores to reduce variance and improve accuracy and robustness. A pipeline was constructed with the same models trained above, with equal weighting for each model's prediction. Using 5-fold cross-validation, the R-Squared and MAE values for training and testing sets were calculated and plotted on a subset of values.

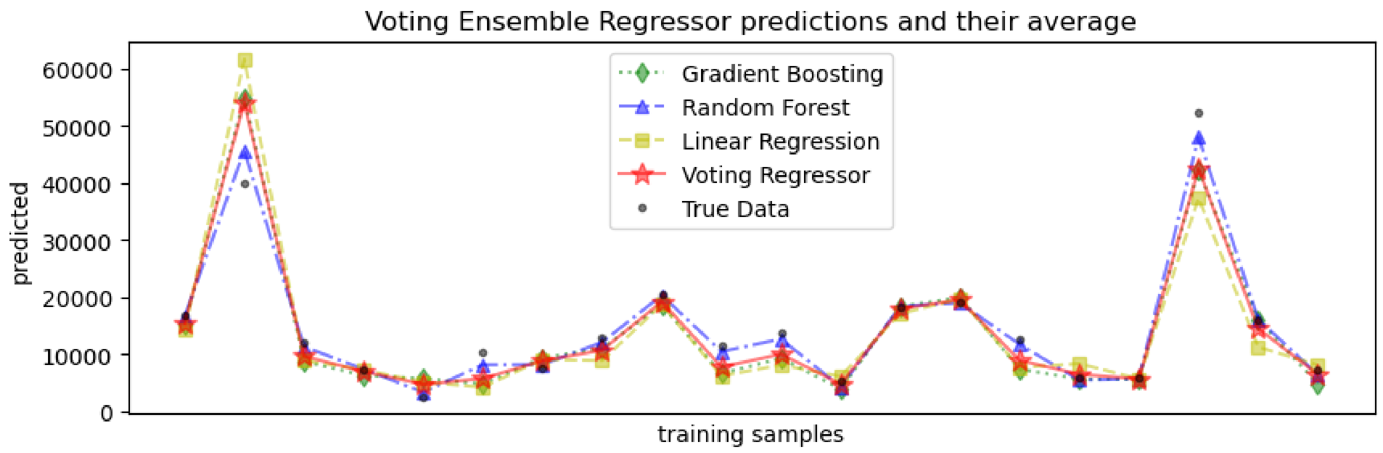
```
average = Pipeline(steps=[("average", VotingRegressor([('regr', LinearRegression()),
            ('rf', RandomForestRegressor()),('gb', GradientBoostingRegressor())], weights=[1, 1, 1]))])
eval_results = cross_validate(average, X_train_pca_df, y_train, cv=5,
    scoring='neg_mean_absolute_error',return_train_score=True)
average.fit(X_train_pca_df, y_train)
train_score = average.score(X_train_pca_df, y_train)
test_score = average.score(X_test_pca_df, y_test)
train_mae = mean_absolute_error(y_train, average.predict(X_train_pca_df))
test_mae = mean_absolute_error(y_test, average.predict(X_test_pca_df))
```



- **Voter Ensemble:**

For regression problems, voter ensemble takes an average of each model mentioned above to make the final prediction. Regarding its application, the same models were utilized, and a 5-fold cross-validation was fit the method. All the usual R-Squared and MAE values for training and testing set were calculated and plotted on a subset of values.

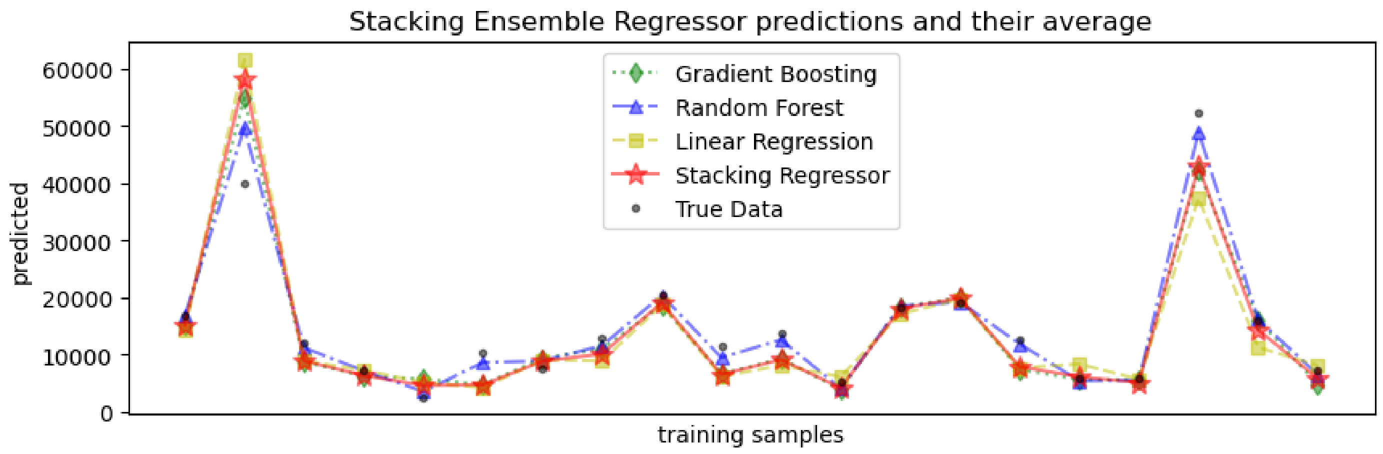
```
voting = Pipeline(steps=[("voting", VotingRegressor([('regr', LinearRegression()),
            ('rf', RandomForestRegressor()),('gb', GradientBoostingRegressor())]))])
eval_results = cross_validate(voting, X_train_pca_df, y_train, cv=5,
    scoring='neg_mean_absolute_error',return_train_score=True)
voting.fit(X_train_pca_df, y_train),train_score = voting.score(X_train_pca_df, y_train)
test_score = voting.score(X_test_pca_df, y_test),train_mae = mean_absolute_error(y_train,
voting.predict(X_train_pca_df)),test_mae = mean_absolute_error(y_test, voting.predict(X_test_pca_df))
```



- **Stacker Ensemble:**

Stacking ensemble technique combines multiple models via a meta-regressor, where base models are trained on the entire training set, and their outputs are used to train the meta-model, which makes the final prediction. As for the application, a pipeline was created for the same models mentioned above. Linear regression was selected as final estimator. The rest of the process and was the same as described above.

```
stacking = Pipeline(steps=[("stacking", StackingRegressor([('regr', LinearRegression()),
            ('rf', RandomForestRegressor()),('gb', GradientBoostingRegressor())],
final_estimator=LinearRegression()))])
eval_results = cross_validate(stacking, X_train_pca_df, y_train, cv=5,
    scoring='neg_mean_absolute_error',return_train_score=True)
stacking.fit(X_train_pca_df, y_train),train_score = stacking.score(X_train_pca_df, y_train)
test_score = stacking.score(X_test_pca_df, y_test)
train_mae = mean_absolute_error(y_train, stacking.predict(X_train_pca_df))
test_mae = mean_absolute_error(y_test, stacking.predict(X_test_pca_df))
```



The following results pertaining to the ensemble methods were calculated.

Name	Train Score	Test Score	Train & Test MAE Scores
Average Ensemble	96.0%	88.3%	2032 and 2589
Voter Ensemble	96.1%	88.4%	2029 and 2577
Stacker Ensemble	95.8%	88.2%	2285 and 2716

The best ensemble according to the table and plots is the Voter Ensemble, as it has the highest test score (88.4%) and the lowest test MAE (2577). This indicates that it performed slightly better in terms of generalization and prediction accuracy on the test set compared to the other ensembles.

5. Model Evaluation and Analysis:

In the last phase of the project, all models' performance were analysed and contribution of all principal components in prediction of target variable.

a) Overall Performance with Cross-Validation:

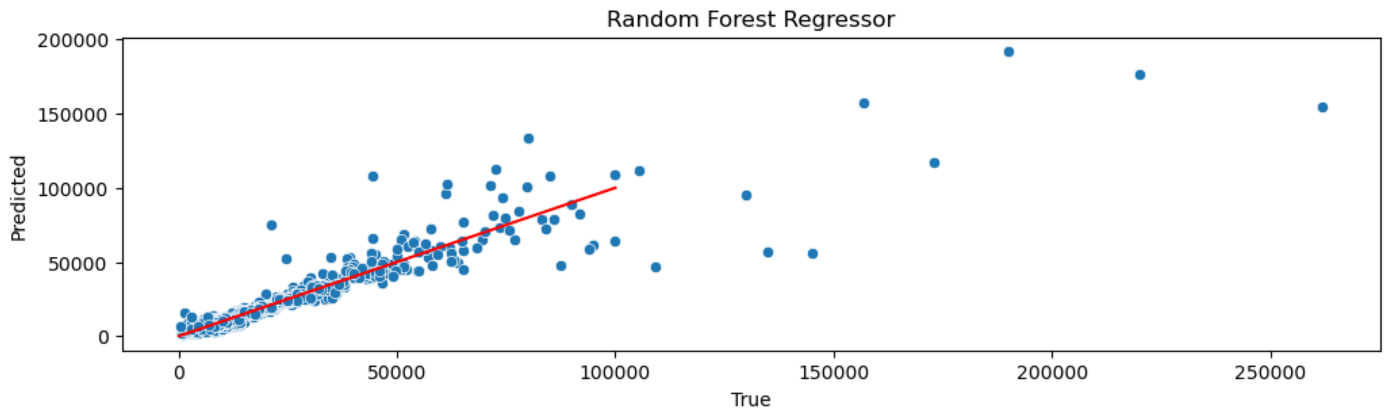
Cross-validation is a statistical method to evaluate model performance by dividing the data into training and validation sets. For Gradient Boosting Regressor, a pipeline was created for 5-fold Cross Validation on training data, and negative mean absolute error was used as the scoring metric. The model was fitted on that data, and R2 score and MAE values for both training and testing was calculated.

```
gbr = Pipeline(steps=[("regr", GradientBoostingRegressor())])
scores = cross_val_score(gbr, X_train_pca_df, y_train, cv=5, scoring='neg_mean_absolute_error')
gbr.fit(X_train_pca_df, y_train)
train_score = gbr.score(X_train_pca_df, y_train)
test_score = gbr.score(X_test_pca_df, y_test)
train_mae = mean_absolute_error(y_train, gbr.predict(X_train_pca_df))
test_mae = mean_absolute_error(y_test, gbr.predict(X_test_pca_df))
```

After Grid Search CV, the Gradient Boosting Regressor displayed significant improvements or stability in R-squared scores, especially in the test score rising to 87.8% from 73.9%. However, Mean Absolute Error (MAE) scores also increased, with training MAE rising from 666 to 2133 and test MAE from 2586 to 2638. Overall, Grid Search CV notably improved the prediction of unseen data.

b) True vs Predicted Analysis:

A true vs. predicted plot visually represents the relationship between the predicted and actual values of a machine learning model. Regarding plotting Random Forest Regressor, the x-axis displayed the true (actual) values, while the y-axis represented the predicted values. It also included a line of perfect prediction, running diagonally from the bottom left to the top right, where ideal predictions would align.



The plot displayed a significantly denser clustering of points along the diagonal line, showcasing improved handling of higher values while still exhibiting outliers. Although the spread of points is particularly narrower, suggesting excellent overall performance, some noticeable deviations remained evident in the visualization.

c) Global and Location Explanations of SHAP:

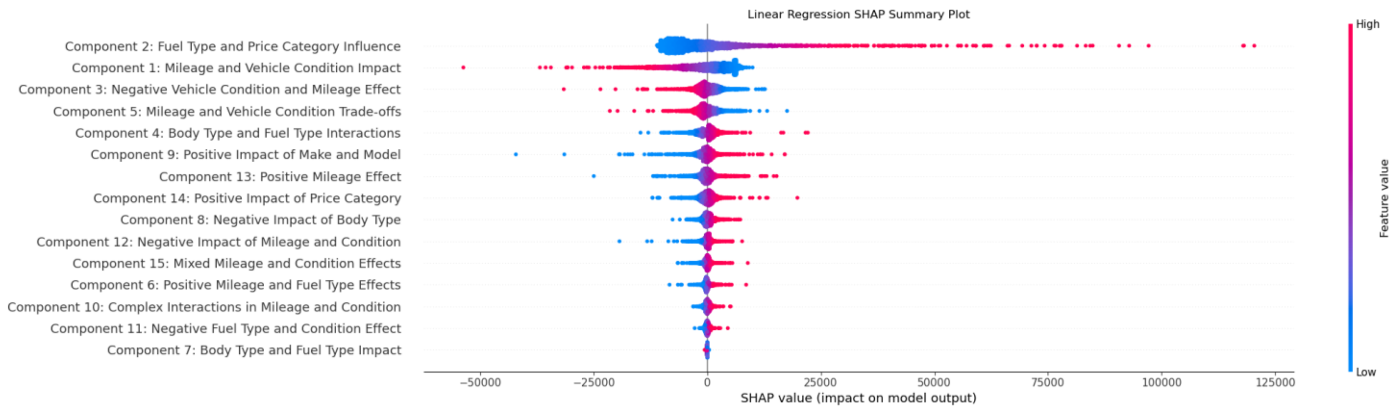
SHAP (Shapley Additive Explanations), is a method applied for understanding the individual contribution of features in a dataset towards predictions made by a machine learning model. Before plotting SHAP values, the columns of X_test_pca_df were renamed to make it more descriptive, indicating what each principal component represents in terms of original features.

- **Global Explanations:**
  Global Explanations helped to understand which features are important for models across all predictions.

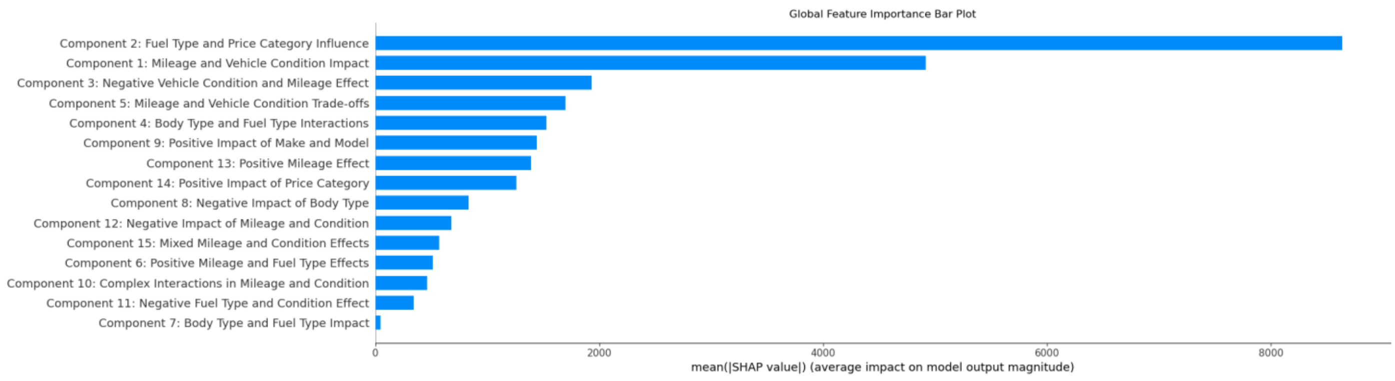
Linear Regression:

Summary and global feature importance bar plot was created to understand each feature's importance.

```
plt.figure(figsize=(22, 6))
explainer = shap.LinearExplainer(regr.named_steps['regr'], X_train_pca_df)
shap_values = explainer.shap_values(X_test_pca_df)
plt.title('Linear Regression SHAP Summary Plot')
shap.summary_plot(shap_values, X_test_pca_df, plot_size=None, show=False, sort=True)
```



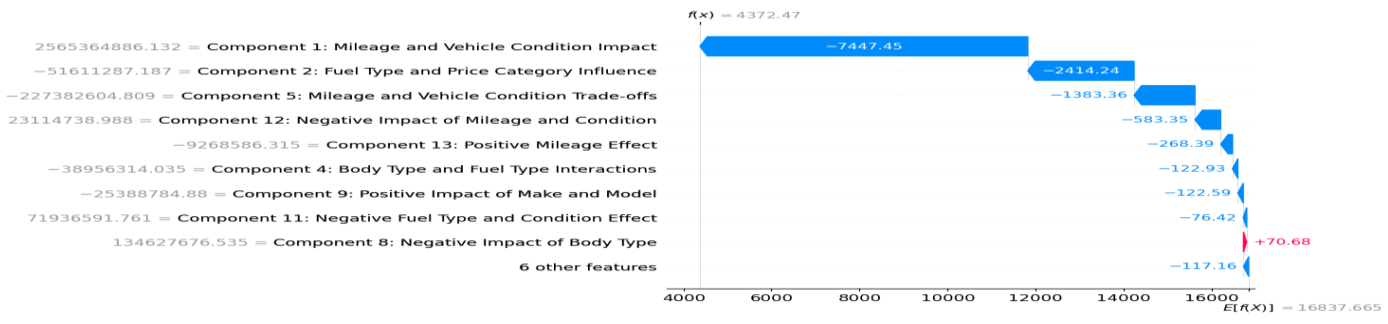
The analysis of vehicle price predictions highlights significant factors: Fuel type and price category (Component 2) are influential, along with mileage and vehicle condition (Component 1), which decreased predictions. Poor vehicle condition and mileage (Component 3) further lowered values. Certain make and model combinations (Component 9) positively affected predictions. Mixed effects are observed, indicating complex interactions. Overall, diverse factors shaped price predictions, including fuel type, condition, and make/model considerations.



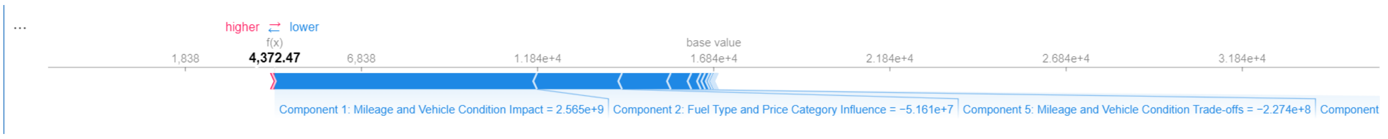
The global feature importance bar plot, using average SHAP values, shows that Component 2 (fuel type and price category) is the most significant factor, while Component 1 (mileage and vehicle condition) is the second most influential, both significantly impacting vehicle price predictions.

- **Local Explanations:**
  Local explanations help us understand why the model made a specific prediction for a single instance based on all the features contribution in making that prediction pertaining to vehicle's price.

Random Forest:



The SHAP waterfall plot explains the predicted car price of £4,372.47, starting from an average base value of £16,837.665. The largest negative impact came from Component 1 (Mileage and Vehicle Condition Impact) at -7,447.45, indicating that high mileage and poor vehicle condition greatly reduced the car's value. Other notable negative impacts included Component 2 (Fuel Type and Price Category Influence) at -2,414.24, and Component 5 (Mileage and Vehicle Condition Trade-offs) at -1,383.36. Additional negative effects came from Components 12, 13, 4, 9, and 11. Component 8: Negative Impact of Body Type contribute slightly to favour of increase in vehicle's price. Moreover, the combined effect of other features provides a minor negative impact of -117.16 to decrease the price.



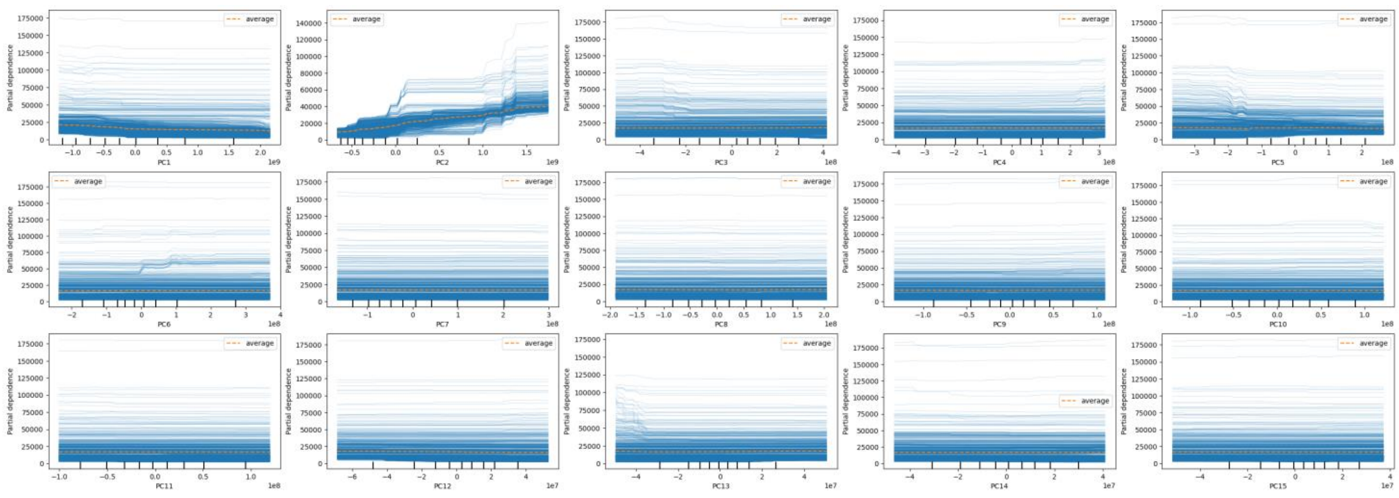
This SHAP force plot illustrates the predicted vehicle price of £4,372.47, starting from an average base value of £16,837.665. Key factors that decrease the price (in blue) include Component 1: Mileage and Vehicle Condition Impact (-7,447.45), Component 2: Fuel Type and Price Category Influence (-2,414.24), and Component 5: Mileage and Vehicle Condition Trade-offs (-1,383.36). On the other hand, positive influences (in red) are not explicitly highlighted in this plot, suggesting that the negative impacts dominate in determining the lower predicted price.

d) Partial Dependency Plots:

Partial Dependency plots visually display the marginal effect one or two features have on a machine learning model's predicted outcome, offering insights into the relationship between predictors and the predicted outcome.

- **Gradient Forest Regressor:**

  For visualizing the effects of all the features in the dataset, a 3x5 grid of subplot was created, each showing a partial dependence plot for a different feature from a Random Forest Regressor model. The kind='both' parameter means both individual and averaged effects of the features were calculated and displayed.



In the partial dependency plots, several principal components significantly impact car price predictions. Component 2 (Fuel Type and Price Category Influence) shows a strong positive impact, while Component 1 (Mileage and Vehicle Condition Impact) has a minor negative effect. Components 3, 8, and 13 highlights negative effects from poor condition, certain body types, and high mileage. Conversely, Component 9 indicates a positive effect from desirable makes and models, and Component 10 underscores complex mileage-condition interactions. Component 14 shows a balanced effect of mileage within specific price categories.