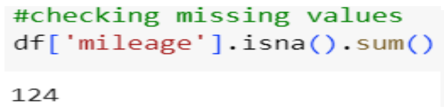
Paramesh Kumar Bukya | 23670090

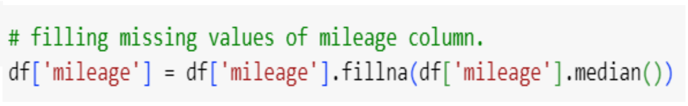
1.Data Processing for Machine Learning

Missing Handling Value:

- Used the isna().sum() method to identify missing values in the dataset.
- Filled missing values in the 'mileage' column with the median value,



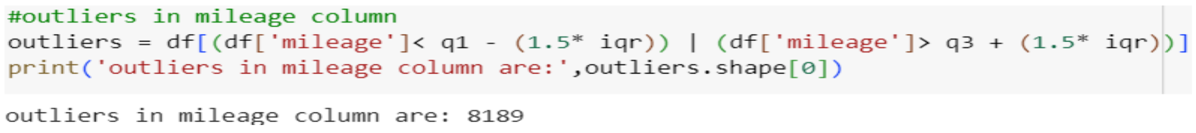
```
#checking missing values
df['mileage'].isna().sum()

124
```



```
# filling missing values of mileage column.
df['mileage'] = df['mileage'].fillna(df['mileage'].median())
```

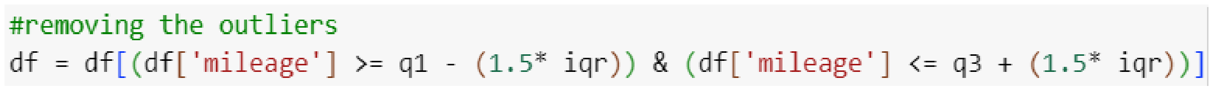
Outlier Detection and removal:

- Detected outliers in the 'mileage' column using the Interquartile Range (IQR) technique, identifying values below q1 - (1.5 * IQR) and above q3 + (1.5 * IQR) as outliers.



```
#outliers in mileage column
outliers = df[(df['mileage']< q1 - (1.5* iqr)) | (df['mileage']> q3 + (1.5* iqr))]
print('outliers in mileage column are:',outliers.shape[0])

outliers in mileage column are: 8189
```

- Removed the identified outliers from the 'mileage' column using the condition.

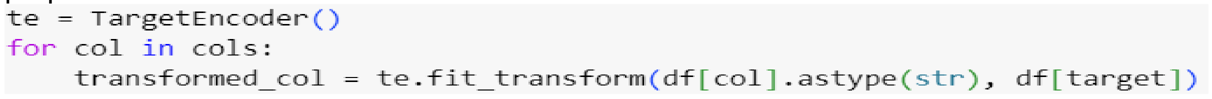


```
#removing the outliers
df = df[(df['mileage'] >= q1 - (1.5* iqr)) & (df['mileage'] <= q3 + (1.5* iqr))]
```

The chosen techniques for handling missing values (median for numerical, mode for categorical) are robust to outliers, computationally efficient, simple, and preserve the distribution and stability of the data. While other techniques like KNN imputation exist, the statistical approach is preferred due to its robustness and simplicity.

Categorical Feature Encoding:

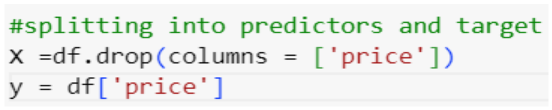
Transformed categorical features into numerical values using Target Encoding, which is better suited for high-cardinality features. Employed the TargetEncoder() class for this purpose.



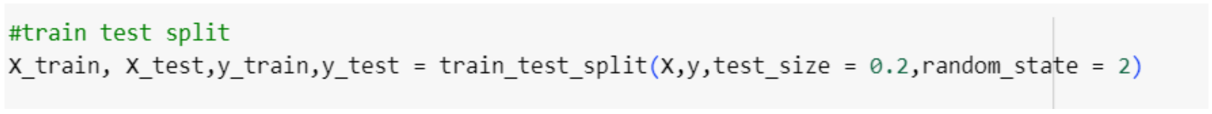
```
te = TargetEncoder()
for col in cols:
    transformed_col = te.fit_transform(df[col].astype(str), df[target])
```

Data Spitting:

Split the dataset into predictor variables (X) and target variable (y).
Performed a train-test split, allocating 80% of the data for training (X_train, y_train) and 20% for testing (X_test, y_test), with a random state set for reproducibility.



```
#splitting into predictors and target
X =df.drop(columns = ['price'])
y = df['price']
```



```
#train test split
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state = 2)
```

Feature Scaling:

Normalized feature values using MinMax scaling, scaling them to a range between 0 and 1. Feature scaling improves model convergence and equalizes feature importance.

```
# Fit the scaler to your data
X_train_scaled = scaler.fit_transform(X_train)
```

```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()
```

2.Feature Engineering

In feature engineering, new and informative features derived based on domain knowledge or polynomial/basis functions to enhance the performance of machine learning models.

Based on domain knowledge:

a): is_luxury feature:

- A new binary feature is_luxury is created based on a predefined list of luxury car brands.
- The code checks if the standard_make column of the DataFrame df belongs to the luxury_brands list.
- The result of the isin() operation is converted to an integer (0 or 1) using astype(int), representing whether the car make is a luxury brand or not.

```
#creating is_luxury column
luxury_brands = ['Mercedes-Benz', 'BMW', 'Audi', 'Rolls-Royce', 'Land Rover']
df['is_luxury'] = df['standard_make'].isin(luxury_brands).astype(int)
```

Using polynomial features method:

- The PolynomialFeatures class from scikit-learn is used to generate new features based on polynomial combinations of the existing features.
- An instance of PolynomialFeatures is created with degree=2 (quadratic polynomials), interaction_only=False (including both interaction and non-interaction terms) and include_bias=False (excluding the bias term).
- The fit_transform() method is applied to the training data X_train_scaled_df to generate the new polynomial features, and the result is stored in X_train_poly.
- The transform() method is applied to the test data X_test_scaled_df to generate the new polynomial features, and the result is stored in X_test_poly

```
# Generated polynomial features
poly = PolynomialFeatures(degree=2,interaction_only=False, include_bias=False)
X_train_poly = poly.fit_transform(X_train_scaled_df)
X_test_poly =poly.transform(X_test_scaled_df)
```

3.Feature selection and Dimensionality Reduction

Feature selection is a critical step in machine learning to improve model performance, interpretability, and prevent overfitting.

Based on Domain knowledge:

The below code shows the features which are selected based on domain knowledge. All the features which I selected are useful in training the model and in further work.

```
#selecting customer based on domain knowledge
df=df[['mileage','year_of_registration', 'price', 'age_of_car', 'is_luxury',
        'standard_colour', 'standard_make', 'standard_model', 'body_type',
        'crossover_car_and_van', 'fuel_type']]
```

Automated Feature selection(AFS):

The main goal of AFS is to choose features according to how each one relates to the target variable. AFS finds the top k features that have the strongest univariate association with the target by using statistical tests like f_regression. AFS is easy to use and effective.

```
# Select top k features
selector = SelectKBest(f_regression, k=6)
X_train_selected = selector.fit_transform(X_train_poly_df,y_train)
X_test_selected = selector.transform(X_test_poly_df)
```

Recursive Feature Elimination (RFE)

Recursive feature elimination (RFE) is a wrapper method that fits a model and eliminates features with low relevance scores until the target number is retained. In addition to RFE cross-validation on five folds, a linear regression model is employed.

```
model = LinearRegression()
ref_selector = RFECV(model, step=1, cv=5)
ref_selector.fit(X_train_poly_df, y_train)
```

Principle Component analysis:

PCA is a technique used to reduce the dimensionality of data. The main idea of PCA is to obtain a lower dimensionality projection of the original data that preserves as much as possible the variance in the data. The below code initializes a Principal Component Analysis (PCA) object with 5 components.

```
# Initialize PCA with desired number of components
pca = PCA(n_components=5)
# Fit PCA on selected features
X_train_selected_pca = pca.fit_transform(X_train_selected_df)
# Transform test data
X_test_selected_pca = pca.transform(X_test_selected_df)
```

PCA is not showing much difference after using it. So, I am not using PCA in my work.

4.Model Building

Let's use A Linear Model(Linear Regression),Random Forest and Boosted Tree algorithms for building models.

a) A Linear Model (Linear Regression):

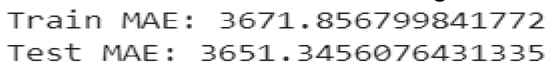
```
#linear regressor
lr = LinearRegression(fit_intercept = True)
lr.fit(X_train_selected_df,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

The Linear Regression model is instantiated with fit intercept parameter and configured as True initially. The model is fitted with the training data.

Grid search is unnecessary for the Linear Regression model because the fit_intercept parameter, which determines whether to calculate the intercept, has only two possible values: True or False. Since there are only two options, there is no need for a grid search to explore different parameter values.

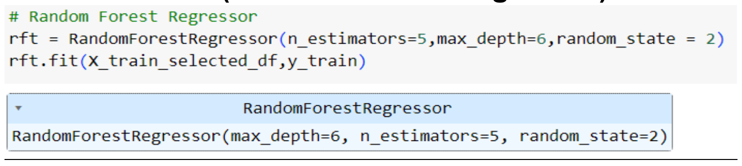
Mean absolute Error of Linear Regression model on trained data is ,



```
Train MAE: 3671.856799841772
Test MAE: 3651.3456076431335
```

Rank : 2

b) Random Forest(Random Forest Regressor)

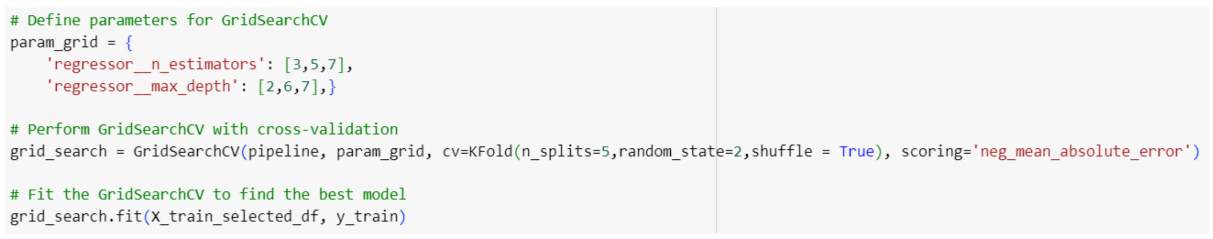


```
# Random Forest Regressor
rft = RandomForestRegressor(n_estimators=5,max_depth=6,random_state = 2)
rft.fit(X_train_selected_df,y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=6, n_estimators=5, random_state=2)
```

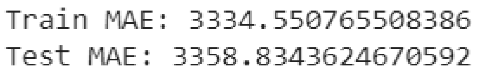
The Random Forest regressor Model is initially instantiated with random parameters n_estimators with 5, max_depth with 6 and random state = 2 then the model fitted on training data.

The below code is of a grid search which is performed to explore various hyperparameter combinations for the Random Forest Regressor, specifically varying the n_estimators and max_depth, using K-fold cross validation with 5 folds. After performing grid search cv, the best parameter value of random forest regressor is n_estimators : 7 , max_depth : 7.



```
# Define parameters for GridSearchCV
param_grid = {
    'regressor__n_estimators': [3,5,7],
    'regressor__max_depth': [2,6,7],}

# Perform GridSearchCV with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=KFold(n_splits=5,random_state=2,shuffle = True), scoring='neg_mean_absolute_error')

# Fit the GridSearchCV to find the best model
grid_search.fit(X_train_selected_df, y_train)
```

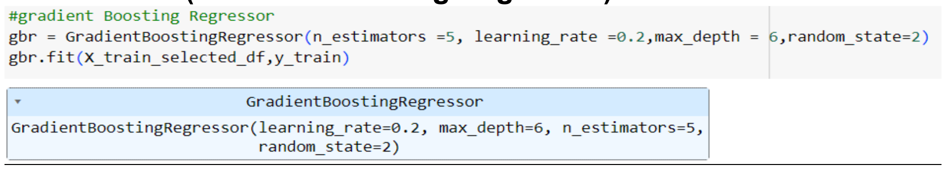
Mean absolute Error of Random Forest Regressor on trained and test data is ,



```
Train MAE: 3334.550765508386
Test MAE: 3358.8343624670592
```

Rank : 1

c) Boosted tree(Gradient Boosting Regressor)



```
#gradient Boosting Regressor
gbr = GradientBoostingRegressor(n_estimators =5, learning_rate =0.2,max_depth = 6,random_state=2)
gbr.fit(X_train_selected_df,y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.2, max_depth=6, n_estimators=5,
                          random_state=2)
```

The Gradient Boosting Regressor model is initially instantiated with random parameters n_estimators with 5, learning_rate with 0.2 and max_depth with 6, then the model is fitted on training data.

The below code is performing grid search to explore different values of the n_estimators, learning_rate and max_depth hyperparameter for Gradient boosting regressor., using K-fold cross-validation with 5 folds. After performing grid search cv, the best parameter value of Gradient boosting regressor model is n_estimators : 7 , learning rate : 0.2 , max_depth : 5.

```
# Define parameters for GridSearchCV
param_grid_gb = {
    'regressor__n_estimators': [5,7,3],  # Number of trees
    'regressor__learning_rate': [0.05, 0.1, 0.2],  # Step size shrinkage
    'regressor__max_depth': [3, 4, 5],  # Maximum depth of the individual estimators
}
# Perform GridSearchCV with cross-validation
grid_search_gb = GridSearchCV(pipeline_gb, param_grid_gb, cv=KFold(n_splits=5), scoring='neg_mean_absolute_error')

# Fit the GridSearchCV to find the best model
grid_search_gb.fit(X_train_selected_df, y_train)
```

Mean absolute Error of Gradient Boosting Regressor on trained and test data is ,

```
Train MAE: 4031.7101278596333
Test MAE: 4037.9944572929166
```

Rank 3

On comparing the above Mean Absolute Errors of training data and testing data of three models, Random Forest Regressor is performing best as it has the lowest MAE. The lower the MAE, the better the model.

4.4 An Averager/Voter/Stacker Ensemble

An ensemble model created by combining the best-performing Random Forest, Linear Regression, and Gradient Boosting models using a Voting Regressor. This ensemble technique can potentially improve predictive performance by leveraging the strengths of each individual model.

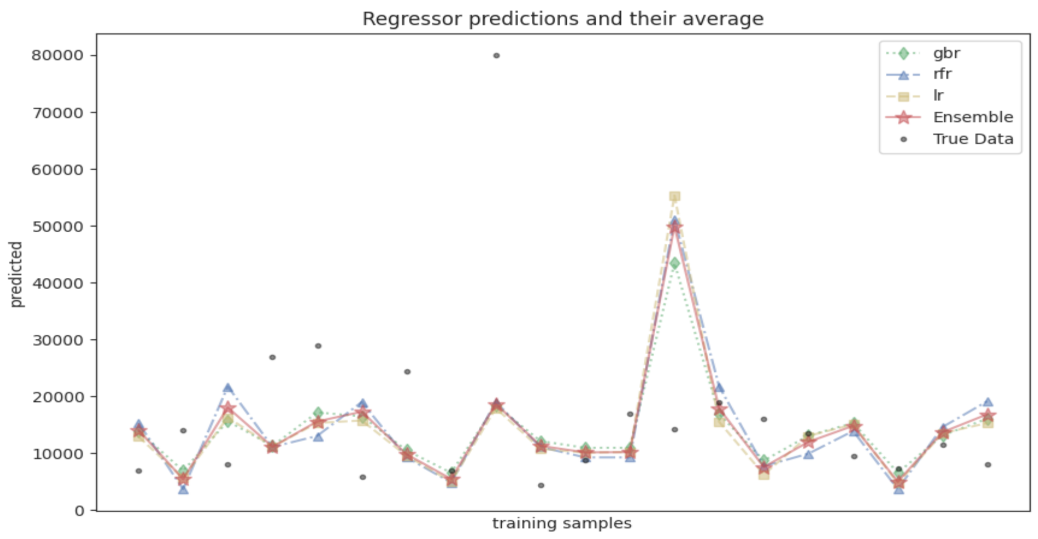
```
ensemble = VotingRegressor([("gb",best_model_gb),("rf",best_model_rf),('lr',best_model_lr)])
ensemble.fit(X_train_selected_df, y_train)
```

A comparison table provided below, showing the cross-validated mean and standard deviation of MAE for each individual model and the ensemble model. This table allows for a direct comparison of the models' performance. Among the evaluated models, the ensemble model emerges as the most important and relevant. By combining the predictions from individual models (Random Forest, Linear Regression, and Gradient Boosting), the ensemble achieves a relatively low mean absolute error (MAE) on both the test and training datasets. The performance suggests that the ensemble can provide accurate predictions while minimizing errors.

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

	test_mae_mean	test_mae_std	train_mae_mean	train_mae_std
best_model_rf	2550.791391	17.314287	2537.516345	12.393716
best_model_lr	3510.251622	18.870026	3510.019612	6.586160
best_model_gb	3132.659478	14.814680	3126.569678	5.416021
ensemble	2653.069006	14.972620	2646.959093	6.296154

The below plot shows the predictions of the individual models and the ensemble model, along with the true data points. This provides insights into how the ensemble combines the predictions of the individual models.



gbr (Gradient Boosting Regressor) predictions appear as blue diamond markers connected by dashed lines. rfr (Random Forest Regressor) predictions are represented by red square markers connected by dashed lines. lr (Linear Regression) predictions are visualized with green triangle markers connected by dashed lines. The ensemble average of all predictions is denoted by yellow plus markers connected by solid lines. True Data points are indicated by black circular markers without connecting lines.

Interestingly, around the midpoint of the plot, all predictions converge closely with the True Data. This convergence suggests that the ensemble average provides a more accurate prediction.

5. Model Evaluation and Analysis

5.1 Overall Performance with Cross-Validation

Cross-validation is a reliable method for evaluating a model's performance. It helps in making sure that the particular train-test split does not skew the model's performance measures. In this part, we will compare the mean absolute error scores of each model and assess its performance using cross-validation.

Linear Regression:

```
# Cross-validated score
cv_scores_lr = cross_val_score(best_model_lr, X_test_selected_df, y_test, scoring='neg_mean_absolute_error')
cv_mean_mae_lr = -1 * cv_scores_lr.mean()
cv_std_mae_lr = cv_scores_lr.std()
```

```
Cross-Validated MAE: 3576.954432174824 +/- 25.13210714048551
```

Random Forest:

```
# Cross-validated score
cv_scores = cross_val_score(best_model_rf, X_test_selected_df, y_test, scoring='neg_mean_absolute_error')
cv_mean_mae = -1 * cv_scores.mean()
cv_std_mae = cv_scores.std()
```

```
Cross-Validated MAE: 3301.0696855184387 +/- 37.34384904683838
```

A Boosted Tree:

```
# Cross-validated score
cv_scores_gb = cross_val_score(best_model_gb, X_test_selected_df, y_test, scoring='neg_mean_absolute_error')
cv_mean_mae_gb = -1 * cv_scores_gb.mean()
cv_std_mae_gb = cv_scores_gb.std()
```

```
Cross-Validated MAE: 4015.208880567459 +/- 33.24258795714202
```

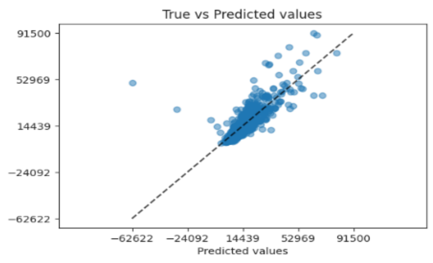
On comparing all the Cross validations of all three models, Cross validated Mean absolute error is lower than other models. So, among three models, Random Forest is performing better followed by Linear Regression model.

5.2 True vs Predicted Analysis:

Analysing the true vs. predicted values visually can help one understand how well the model performs across the target variable's range. To compare actual target values with the predicted ones for each model, we'll make scatter plots in this section.

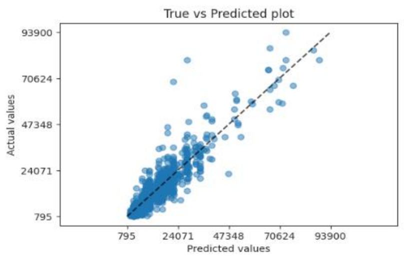
Linear regression:

```
PredictionErrorDisplay.from_estimator(
    best_model_lr, X_test_selected_df, y_test, kind="actual_vs_predicted", scatter_kwargs=dict(alpha=0.5));
```



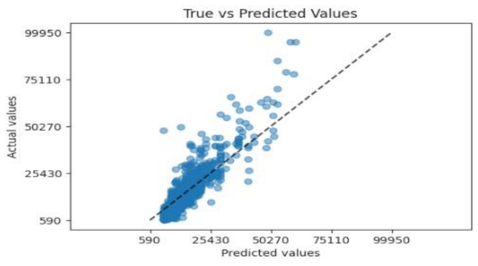
Random forest:

```
PredictionErrorDisplay.from_estimator(
    best_model_rf, X_test_selected_df, y_test, kind="actual_vs_predicted", scatter_kwargs=dict(alpha=0.5));
```



Gradient Boosting Regressor:

```
PredictionErrorDisplay.from_estimator(
    best_model_gb, X_test_selected_df, y_test, kind="actual_vs_predicted", scatter_kwargs=dict(alpha=0.5));
```



In all the above True vs Predicted plots, the x-axis represents predicted values, while the y-axis represents actual values. Each blue dot corresponds to an individual data point, reflecting predictions made by a model. The dashed line across the plot represents perfect prediction, where true values match predicted values. Most dots cluster around the dotted line, indicating reasonably accurate predictions. However, some outliers deviate from the ideal line, highlighting areas for improvement in the model's accuracy.

5.3 Global and Local Explanations with SHAP

Global and local explanations with SHAP provide insights into how individual features contribute to model predictions on both a specific instance (local explanation) and across the entire dataset (global explanation). Now I am SHAP(local explanation and global explanation) on Linear regression and Random Forest Regressor.

Linear regression:

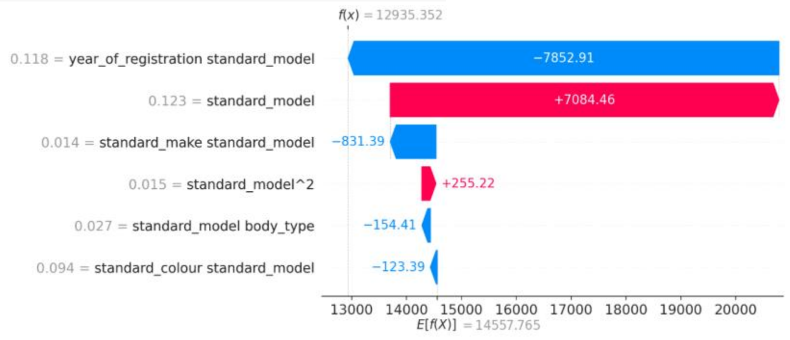
'LinearExplainer' is used to generate SHAP values for linear regression model.

```
explainer = shap.LinearExplainer(best_lr_regressor, X_train_selected_df)
```

a) Local Explanations:

This below code generates waterfall plot for the first instance in my test set, showing how each feature contributes to the difference between the base value and the model's prediction for that instance.

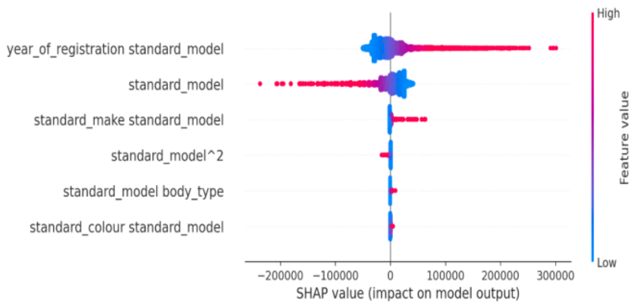
```
# for a random instance
shap.plots.waterfall(shap_values[0])
```



b) Global Explanation:

The below code generates a beeswarm plot, where each point represents a SHAP value for a feature in a particular data point, by giving an overview of feature importances across the entire dataset.

```
# global model
shap.plots.beeswarm(shap_values)
```



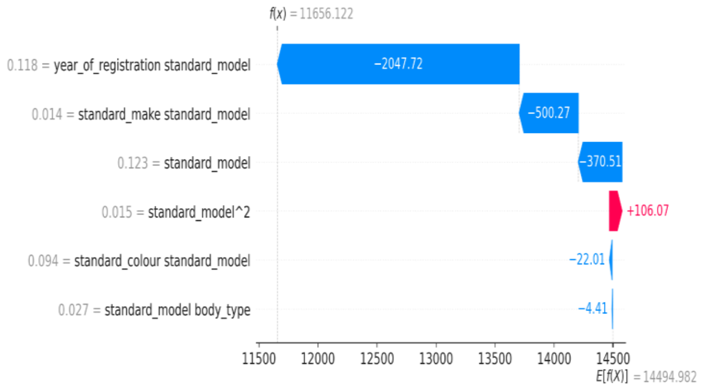
From the above plot, ‘year_of_registration_standard_model’(positive values) and ‘standard_model’(negative values) shows significant impact on the model’s prediction. Features with dots clustered close to the center (around 0) on the x-axis have a minimal influence on the model's prediction.

Random forest:

TreeExplainer is used to generate SHAP values for Random Forest Regressor,

```
explainer = shap.TreeExplainer(best_rf_regressor, X_train_selected_df)
```

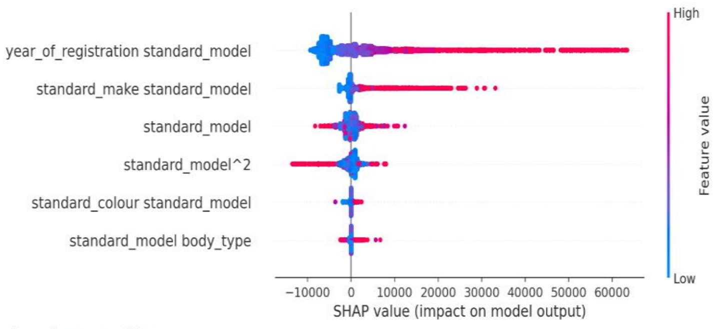
a) Local Explanations:



```
# for a random instance
shap.plots.waterfall(shap_values[0])
```

b) Global Explanations:

Beeswarm plot is used for visualizing global explanations,



```
# global model
shap.plots.beeswarm(shap_values)
```

```
Figure size 600x400 with 0 Axes>
```

From the above plot, ‘year_of_registration_standard_model’(positive values) and ‘standard_make_standard_model’(positive values) shows significant impact on the model’s prediction. Features with dots clustered close to the center (around 0) on the x-axis have a minimal influence on the model's prediction.

Gradient boosting Regressor:

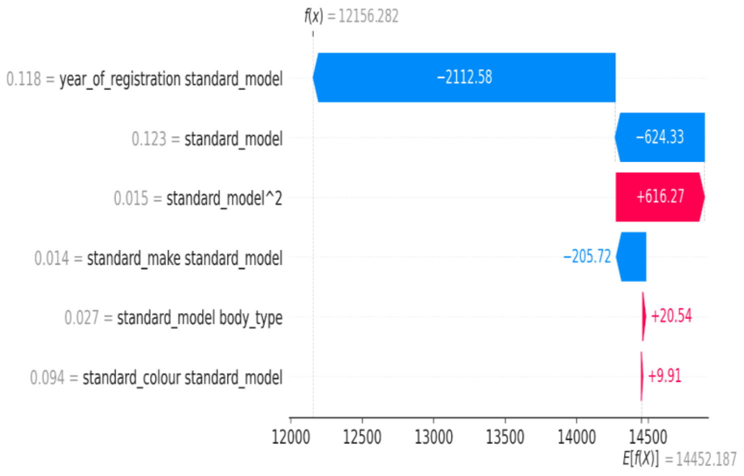
TreeExplainer is used to generate SHAP values for Gradient Boosting Regressor,

```
explainer = shap.TreeExplainer(best_gb_regressor, X_train_selected_df)
```

a) Local Explanations:

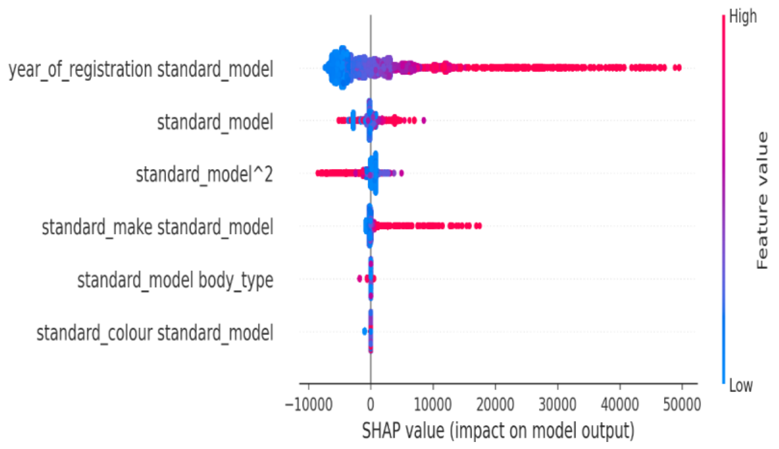
Water Fall plot is used to visualize local explanations,

```
# for a random instance
shap.plots.waterfall(shap_values[0])
```



b) Global Explanations:

```
# global model
shap.plots.beeswarm(shap_values)
```



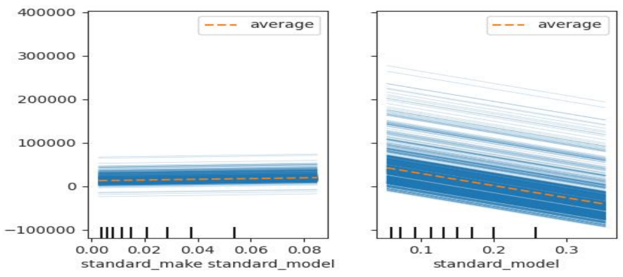
Similar to the random forest regressor , features 'year_of_registration_standard_model'(positive values) and 'standard_make_standard_model'(positive values) shows significant impact on the model's prediction.

5.4 Partial Dependency Plots

Partial Dependency Plots (PDPs) generated for the Random Forest, Linear Regression, and Gradient Boosting models. These plots help understand the marginal effect of one or two features on the predicted outcome, while averaging out the effects of all other features.

Linear Regression:

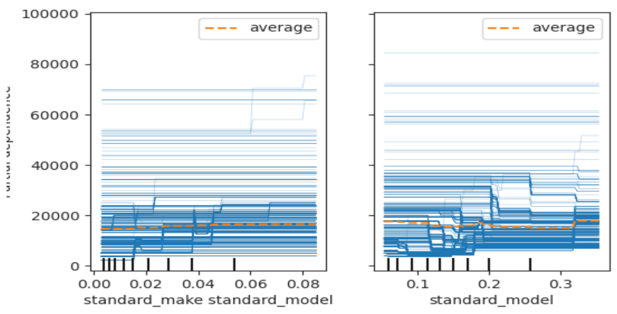
```
# Partial Dependency Plots for Linear Regression
features_indices = [X_train_selected_df.columns.tolist().index(feature) for feature in ['standard_make standard_model', 'standard_model']]
PartialDependenceDisplay.from_estimator(best_model_lr, X_test_selected_df, features=features_indices,kind='both');
```



The PDPs both show increasing trends, with "standard_model" having a steeper increase than "standard_make". This suggests the model is more sensitive to changes in "standard_model" than "standard_make".

Random Forest:

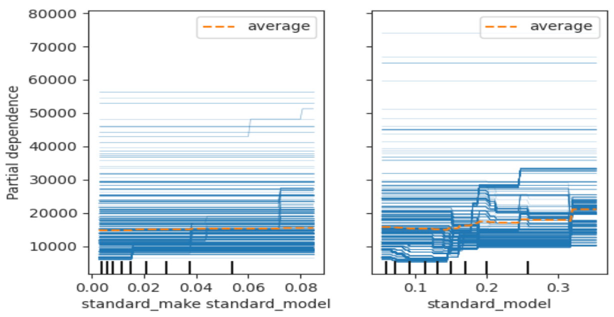
```
# Partial Dependency Plots for Random Forest
PartialDependenceDisplay.from_estimator(best_model_rf, X_test_selected_df, features=features_indices,kind='both');
```



According to these plots as "standard_model" increases, the average value of "average" also increases.

Gradient Boosting Regressor:

```
# Partial Dependency Plots for Gradient Boosting Regressor
PartialDependenceDisplay.from_estimator(best_model_gb, X_test_selected_df, features=features_indices,kind='both');
```



According to these plots, there's a positive correlation: as "standard_model" increases, "partial dependence" also increases. Most points show higher variability than average, indicating fluctuation around the mean.