NAME: TEMILOLUWA AKHIGBE

STUDENT ID: 22534133

DATA PROCESSING FOR MACHINE LEARNING

Missing Values, Outliers and Noise: The dataset was divided into two based on the vehicle condition. The unique values of the registration code and year of registration column are viewed, some noise is observed and the rows containing these values are dropped. The missing registration codes can be reverse engineered from the years of registration according to the information on this Wikipedia Page. The rows with null values for both year of registration and registration code are deleted. Next, a dictionary is created with the years of registration as the keys and the registration codes as the values, the missing registration codes were then mapped to the corresponding years of registration.

The missing values in the mileage column were replaced by the mean value. The missing values in the standard colour, body type and fuel type columns were replaced by their modes grouped by their standard model.

Outliers were identified as data points that fall outside 1.5 multiplied by the interquartile range below the first quartile and above the third quartile and they are deleted.

```
cols = ['mileage', 'price']
#Outliers are identified and deleted
Q1 = Old_cars[cols].quantile(0.25)
Q3 = Old_cars[cols].quantile(0.75)
IQR = Q3 - Q1
Old_cars = Old_cars[~((Old_cars[cols] < (Q1 - 1.5*IQR)) | (Old_cars[cols] > (Q3 + 1.5*IQR))).any(axis = 1)]
```

It is observed that all the rows in the year of registration and registration code columns are null, which is understandable as the cars are new and unregistered. Therefore, the years of registration are replaced with the latest year of registration (2020) and the registration code is replaced by 0. The Old and New cars subset are then concatenated into a new dataframe called auto_trader.

Categorical Encoding: Target encoding is used to transform the categorical variables into numerical representations based on their relationship with the target (price). The TargetEncoder class was imported from the category_encoders library and applied to the categorical columns. The resulting encoded dataset is named auto_trader_target.

```
target_encode_features = ["standard_colour", "standard_make", "standard_model", "body_type",
                          "fuel_type", "vehicle_condition", "crossover_car_and_van"]
encoder = TargetEncoder(cols=target_encode_features)
encoder.fit(auto_trader_sample, auto_trader_sample["price"])
auto_trader_target = encoder.transform(auto_trader_sample)
```

Rescaling: The data is rescaled to ensure that the features are on a similar scale using StandardScalar(). The StandardScalar() method was put in a preprocessing pipeline which was applied to the dataset before fitting it into the model.

Splitting Dataset into Predictors and Target: The auto_trader_target dataset is split into input features (X) and the target variable (y). The X DataFrame contains all the features from the dataset that will be used to predict the target variable (price). The target variable (y) is obtained by selecting only the 'price' column from the auto_trader_target dataset. This represents the dependent variable that will be predicted based on the input features.

```
X = auto_trader_target.drop('price', axis=1)
y = auto_trader_target['price']
```

Obtaining Train and Test Folds: The train_test_split function is used to split the data into two sets: the training set and the testing set. 25% of the data will be used for testing and 75% will be used for training. The training set (X_train and y_train) is used to train the machine learning model, while the testing set (X_test and y_test) is used to evaluate the model's performance on unseen data.

FEATURE ENGINEERING

Due to the size of this dataset, a representative subset of the data was created. 10% of the data from each group of the "standard_model" column in the auto_trader dataframe was collected and stored in a new dataframe named auto_trader_sample.

```
#use stratified sampling to take a sample of the dataset for modelling
auto_trader_sample = auto_trader.groupby("standard_model", group_keys=False).apply(lambda x:x.sample(frac=0.1))
```

```
#change the datatype of the mileage column to integer
auto_trader['mileage'] = auto_trader['mileage'].astype(int)
```

```
#change the datatype of the year of registration column to integer
auto_trader['year_of_registration'] = auto_trader['year_of_registration'].astype(int)
```

The data type of mileage and year of registration was changed from float to integer as shown in the images above.

```
#create a new column called car_age by subtracting 2020 from the year of registration
year = 2020
auto_trader['car_age'] = year - auto_trader['year_of_registration']
```

A new column called car age was created by subtracting 2020 from the year of registration as seen in the code snippet above.

Polynomial and Interaction Features: The data is passed into the PolynomialFeatures() transformer, the result is a transformed dataset with additional features capturing the relationships and interactions between the original features.

```
preprocessing_pipe = Pipeline(
    steps=[
        ('scaler', StandardScaler()),
        ('poly', PolynomialFeatures(interaction_only=True, include_bias=False))
    ]
).set_output(transform='pandas')
```

```
X_pp = preprocessing_pipe.fit_transform(X)
```

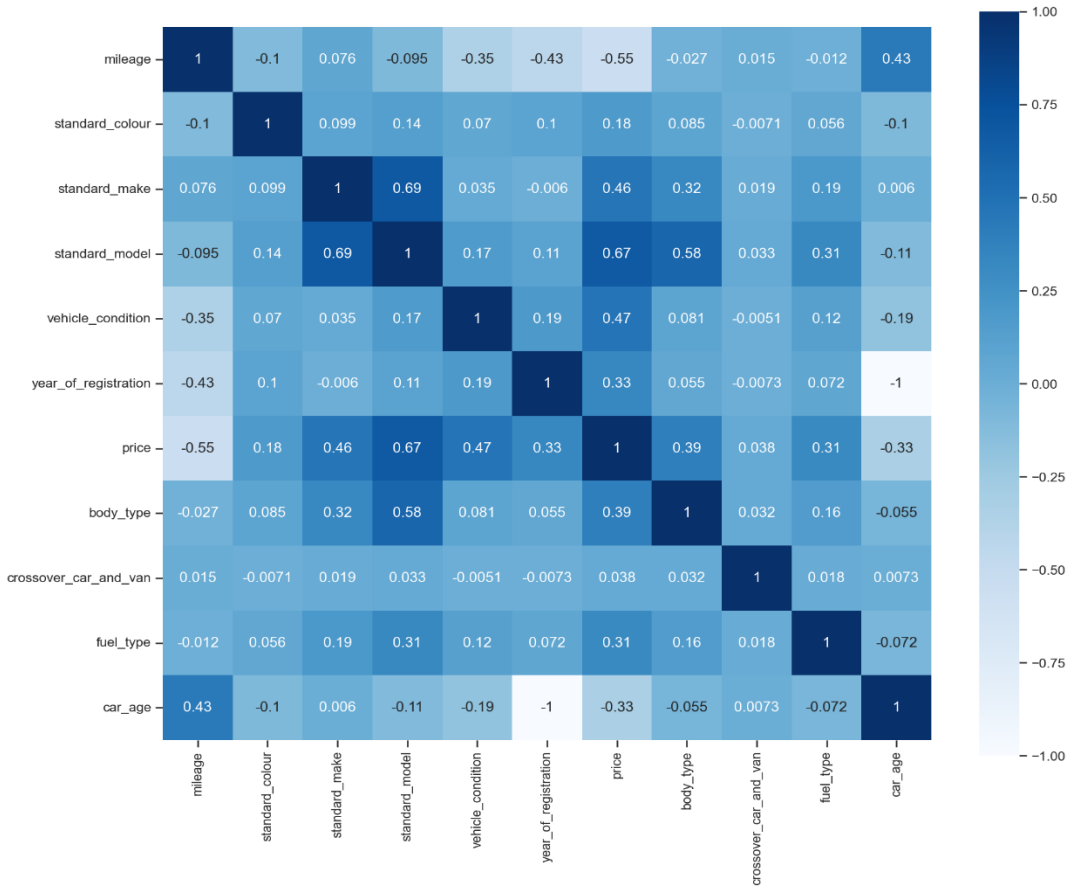
```
preprocessing_pipe.fit_transform(X).columns
```

```
Index(['mileage', 'standard_make', 'standard_model', 'vehicle_condition',
       'body_type', 'crossover_car_and_van', 'fuel_type', 'car_age',
       'mileage standard_make', 'mileage standard_model',
       'mileage vehicle_condition', 'mileage body_type',
       'mileage crossover_car_and_van', 'mileage fuel_type', 'mileage car_age',
       'standard_make standard_model', 'standard_make vehicle_condition',
       'standard_make body_type', 'standard_make crossover_car_and_van',
       'standard_make fuel_type', 'standard_make car_age',
       'standard_model vehicle_condition', 'standard_model body_type',
       'standard_model crossover_car_and_van', 'standard_model fuel_type',
       'standard_model car_age', 'vehicle_condition body_type',
       'vehicle_condition crossover_car_and_van',
       'vehicle_condition fuel_type', 'vehicle_condition car_age',
       'body_type crossover_car_and_van', 'body_type fuel_type',
       'body_type car_age', 'crossover_car_and_van fuel_type',
       'crossover_car_and_van car_age', 'fuel_type car_age'],
      dtype='object')
```

36 columns are observed with new columns including interactions between the original features.

FEATURE SELECTION AND DIMENSIONALITY REDUCTION

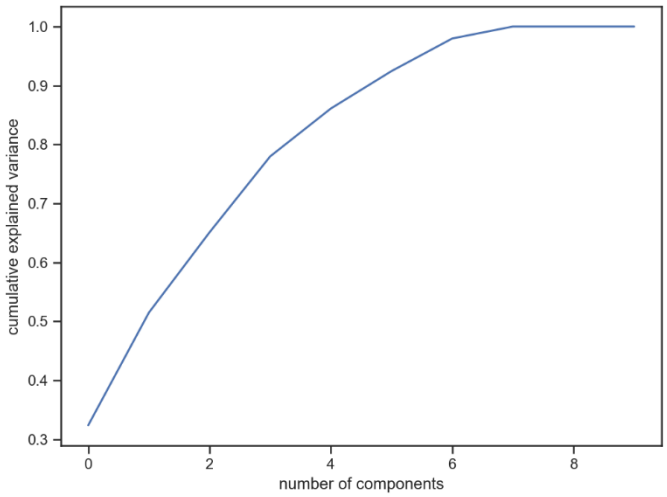
The correlation coefficient was calculated and visualized using a heatmap. Features with high correlation coefficients are noted as they are likely to be important predictors. The following features were dropped as they have a low correlation with the target (price): standard colour, registration code. The year of registration column was also dropped as it understandably has the same correlation as the car age.



The SelectKBest method was used on the predictors, which selects the most influential features to the target variable. The top 10 features were selected and they are observed in the image below.

mileage	standard_make	standard_model	vehicle_condition	body_type	fuel_type	car_age	mileage vehicle_condition	standard_model vehicle_condition	vehicle_condition car_age
-1.106093	0.840496	-0.162375	-0.295733	-0.995283	-0.560490	0.484301	0.327109	0.04802	-0.143224
1.793898	0.840496	-0.162375	-0.295733	-0.995283	-0.560490	0.763751	-0.530515	0.04802	-0.225867
-0.570732	0.840496	-0.162375	-0.295733	-0.995283	0.298749	-0.214323	0.168784	0.04802	0.063383
-0.186041	0.840496	-0.162375	-0.295733	0.138073	-0.560490	0.484301	0.055019	0.04802	-0.143224
0.472838	0.840496	-0.162375	-0.295733	-0.995283	0.298749	0.204852	-0.139834	0.04802	-0.060581

Dimensionality Reduction: Principal Component Analysis (PCA) was used to transform the dataset into a lower dimensional space. The cumulative explained variance ratio was plotted as a line plot to help determine the appropriate number of components to retain for dimensionality reduction and seven components were retained: mileage, standard make, standard model, vehicle condition, body type, fuel type and car age.



MODEL BUILDING:

The data set was split into training and test sets after feature selection and dimensionality reduction has been performed. The test size was 25% of the data and the random state was set to 0 to ensure reproducibility.

```
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.25, random_state=0)
```

Three models were selected for this regression problem: Linear Regression, Random Forest Regressor and Hist Gradient Boost Regressor.

```
lgr = LinearRegression()
rfr = RandomForestRegressor()
hgbr = HistGradientBoostingRegressor()
```

The linear regression model is selected as the baseline model, a random forest regressor and a hist gradient boost regressor were also initiated and the training data was fit into the three models.

```
lgr.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
rfr.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
hgbr.fit(X_train, y_train)
```

```
▾ HistGradientBoostingRegressor
HistGradientBoostingRegressor()
```

The RMSE and the r2 scores of the models are:

	R2 score	RMSE
LinearRegression()	0.77428	4494.03
RandomForestRegressor()	0.91035	2832.18
HistGradientBoostingRegressor()	0.89424	3076.14

A parameter grid to tune the hyperparameters was defined and a grid search was performed with a specified set of possible values and then the model is evaluated for each combination of hyperparameter values.

```
# Define the parameter grid for hyperparameter tuning
param_grid_lgr = {'fit_intercept':[True,False], 'copy_X':[True, False]}
param_grid_rfr = {'n_estimators': [50, 100, 150, 200, 300], 'max_depth': [None, 5, 10, 15, 20, 25, 30]}
param_grid_hgbr = {'learning_rate': [0.5, 0.1, 0.05, 0.01], 'max_depth': [2, 3, 5, 7, 9]}
```

```
# Perform GridSearchCV to tune the hyperparameters
grid_lgr = GridSearchCV(lgr, param_grid_lgr, scoring='neg_mean_squared_error', cv=5)
grid_rfr = GridSearchCV(rfr, param_grid_rfr, scoring='neg_mean_squared_error', cv=5)
grid_hgbr = GridSearchCV(hgbr, param_grid_hgbr, scoring='neg_mean_squared_error', cv=5)
```

The best parameters for each model were obtained using the best_params_ attribute; this returns a dictionary with the best hyperparameters identified by the grid search. A new model was created with the best hyperparameters and the training data was fit into it.

```
best_lgr_params = grid_lgr.best_params_
best_rfr_params = grid_rfr.best_params_
best_hgbr_params = grid_hgbr.best_params_
```

```
# Select the best performing models
best_lgr = LinearRegression(**best_lgr_params)
```

```
best_rfr = RandomForestRegressor(**best_rfr_params)
```

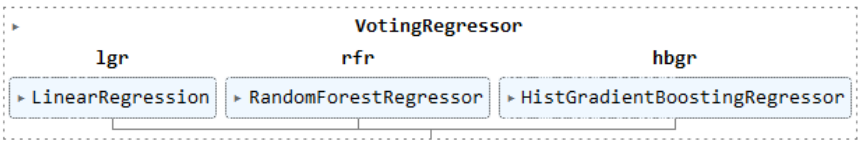
```
best_hgbr = HistGradientBoostingRegressor(**best_hgbr_params)
```

An ensemble was created with the best performing models.

```
ensembled = [ best_lgr, best_rfr, best_hgbr ]
```

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

```
ensemble = VotingRegressor(
    [
        ("lgr", best_lgr),
        ("rfr", best_rfr),
        ("hbgr", best_hgbr)
    ]
)
ensemble.fit(X_train, y_train)
```



MODEL EVALUATION AND ANALYSIS:

Evaluation of Models Based on Popular Scores and Metrics: The selected models are evaluated using the root mean square error and r2 score.

	R2 score	RMSE
LinearRegression()	0.77428	4494.03
RandomForestRegressor()	0.91117	2819.20
HistGradientBoostingRegressor()	0.90771	2873.58
Ensemble	0.89674	3039.53

After getting the best hyperparameters and getting the best performing models, a slight improvement is observed in the r2 scores and RMSE scores. The Random Forest Regressor is the best performing model as it has the lowest RMSE score and the highest r2 score. The HistGradientBoosting Regressor is the next best performing model according to these metrics.

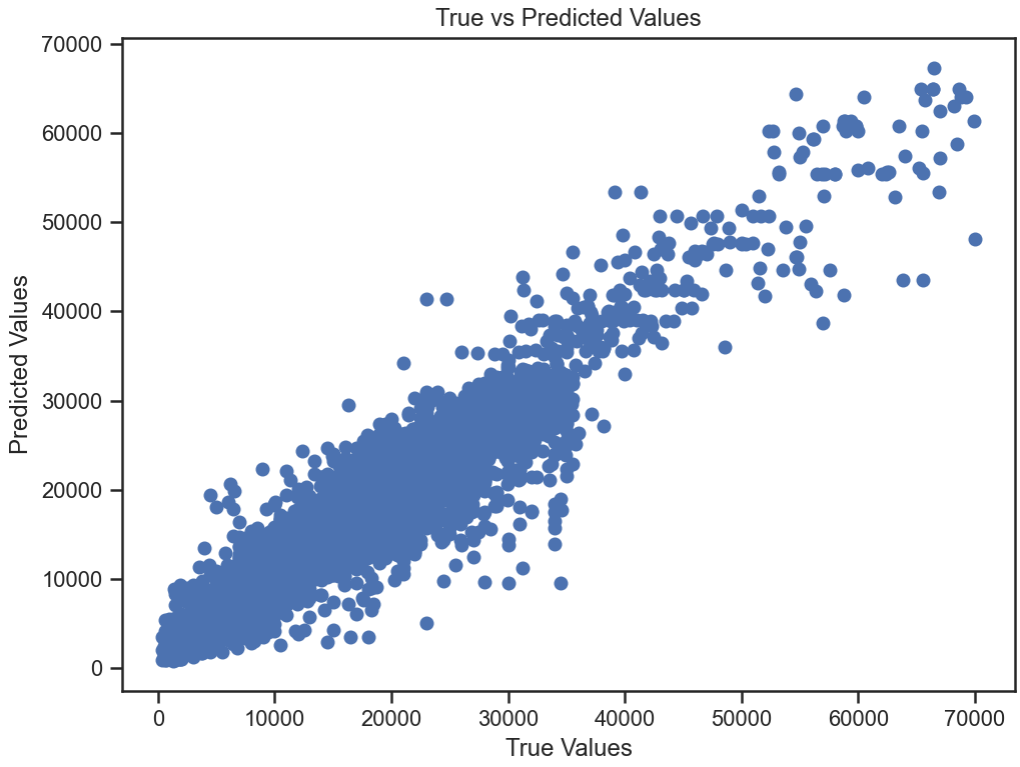
Cross Validation: The overall performance of the random forest regressor is calculated with cross validation across 10 folds using r2 scoring as the scoring parameter.

```
scores = cross_val_score(best_rfr, X_train, y_train, cv=10, scoring='r2')
scores.mean(), scores.std()
```

```
(0.9021588110794525, 0.0055634885696033524)
```

The mean value of the cross validation score is 0.9021, meaning the model averagely explains about 90.21% of the variance in the target variable across the 10 folds; This high score indicates a good predictive performance. The standard deviation of the scores is 0.0056, this low standard deviation score indicates that the model's performance is consistent across different subsets of the data. From the cross validation scores, we can conclude that the Random Forest Regressor performs well on both the training and the test datasets.

A scatter plot is created to visually compare the true and predicted values.



The plot above shows that majority of the data points fall close to a diagonal line indicating that the predicted values are close to the true values. This tells us that the Random Forest Regressor is performing well and

accurately capturing the underlying patterns in the data. The linearity of the plot along with the low Root Mean Square Error and the high r2 score of the Random Forest Regressor indicates that the model is performing well and is neither overfitting nor underfitting.

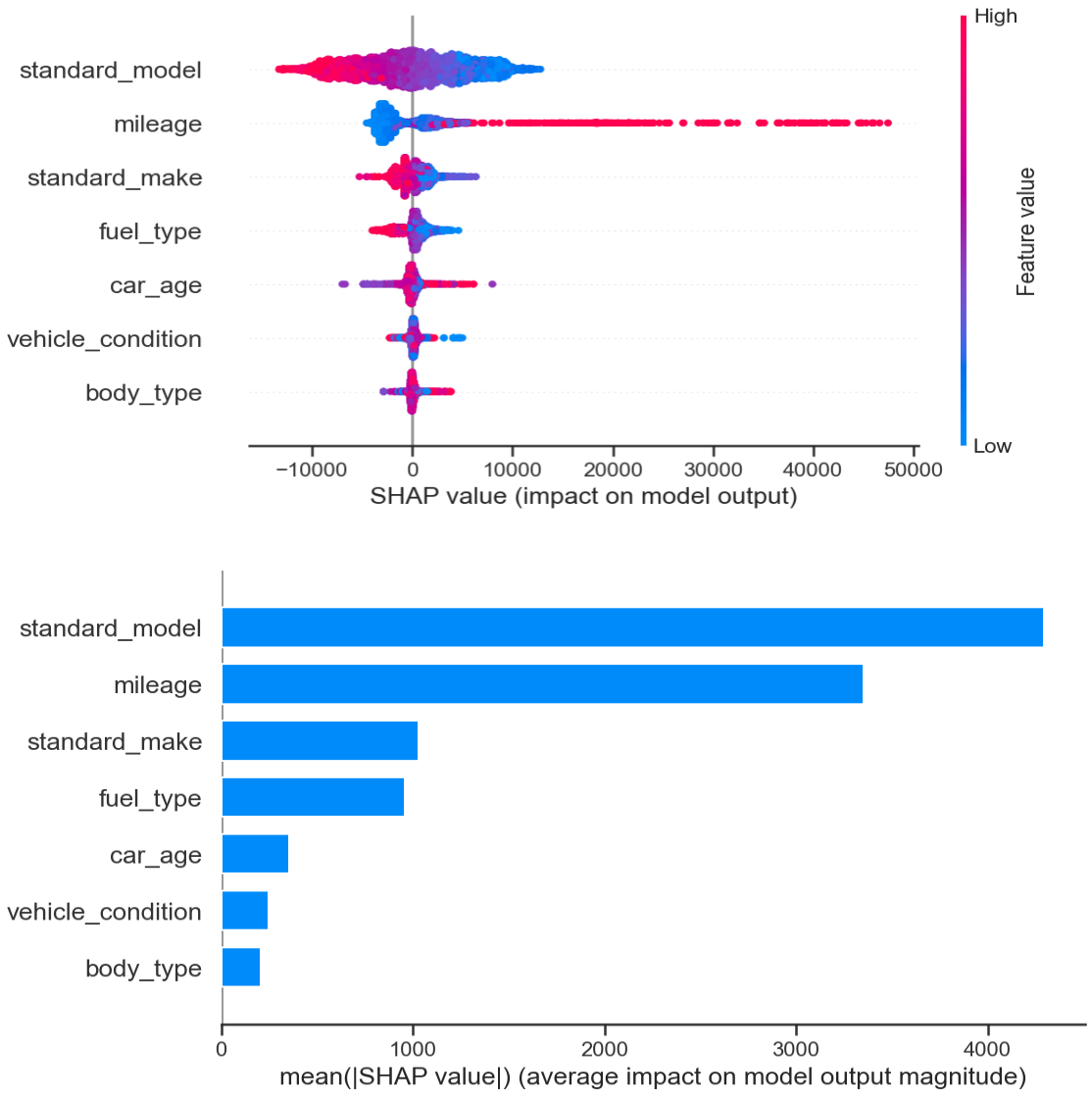
Global Explanations with SHAP: A TreeExplainer object is created, taking the random forest model as input. The SHAP values for the test set is calculated. The SHAP values represent the contribution of each feature to the prediction for each instance in the dataset.

```
# Create a TreeExplainer for the Random Forest Regression model
explainer = shap.TreeExplainer(best_rfr)
```

```
# Compute SHAP values for the test data
shap_values = explainer.shap_values(X_test)
```

```
# Global Explainability: Plot summary of SHAP values
shap.summary_plot(shap_values, X_test)
```

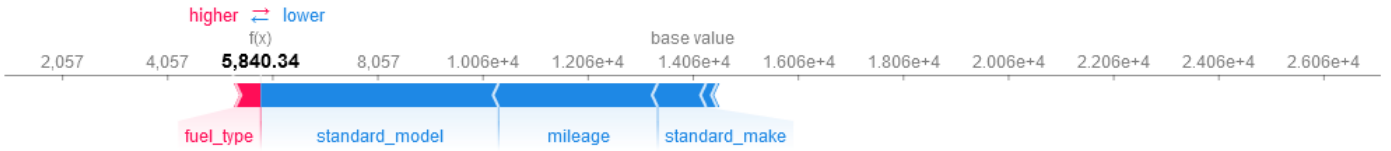
A summary plot is then generated to display the positive and negative relationships of the predictors with the target variable and to identify which features have significant impact on the model's output.



We can identify the features along with their impact from the plots above. The model of the vehicle has the most significant impact on the model output. The mileage also has a significant impact, and it is observed that it has a high impact on the target price of the vehicle.

Local Explainability with SHAP: An instance is specified using an index and the SHAP values for that instance is obtained and the contributions of the individual features to the prediction for that instance is visualized using force_plot.

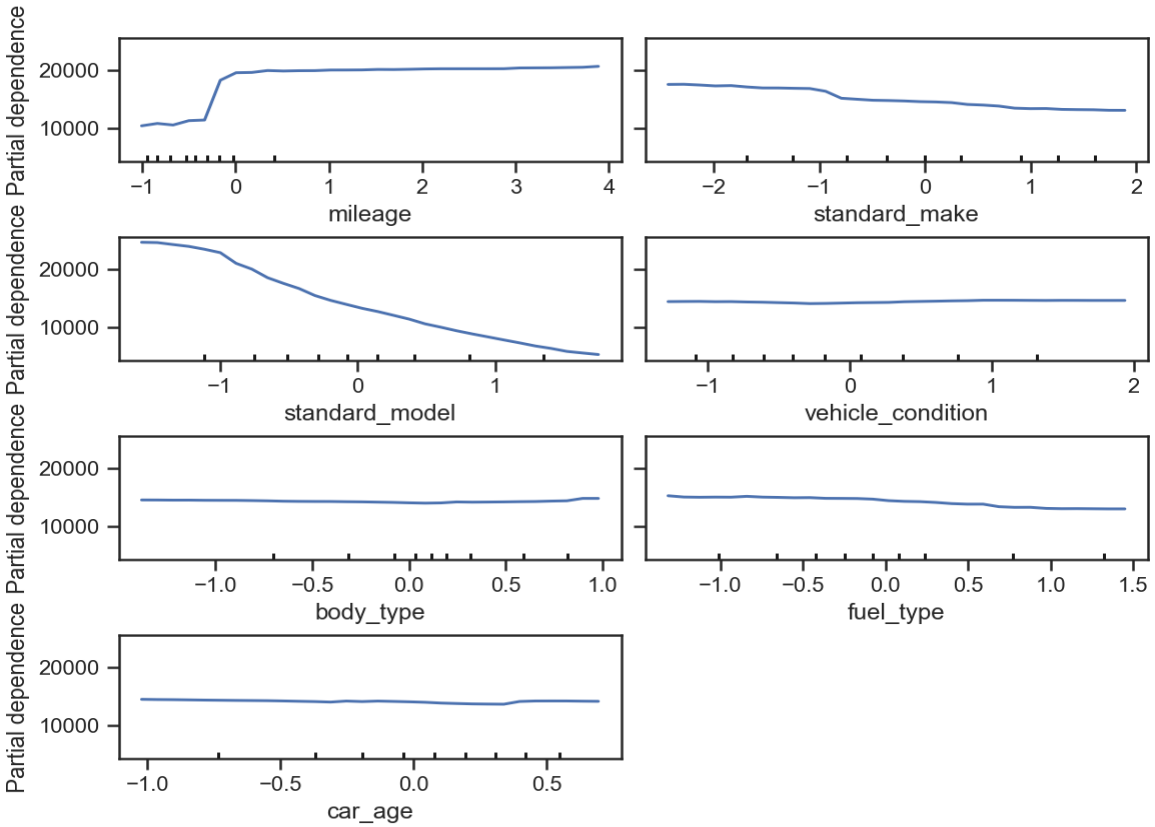
```
row_idx = 0
shap.force_plot(
    explainer.expected_value,
    shap_values[row_idx],
    features=X_test.columns
)
```



The output value for the prediction above is 5,850.34. It is noticed that the fuel type pushes the prediction to the right. While the standard model, mileage and standard make of the vehicle pushes the prediction in this instance to the left.

Partial Dependency Plots: To understand the relationship between the vehicle price and the variables, partial dependency plots are plotted.

```
fig, ax = plt.subplots(figsize=(8,6), constrained_layout=True)
PartialDependenceDisplay.from_estimator(
    best_rfr, X_test, features=X_test.select_dtypes(exclude='object').columns,
    kind='both',
    subsample=100, grid_resolution=30, n_jobs=2, random_state=0,
    ax=ax, n_cols=2
);
```



From the plots above, we can see the relationship that each of the variables have with the target (price).

In conclusion, the Random Forest Regression model is the best performing model and the most significant features in determining the price of a vehicle is the mileage and the standard make of the car.