CAR SELLING PRICE PREDICTION

22533657

LAVANYA SREEDHAR

TABLE OF CONTENTS

1. Data processing for Machine Learning

Data processing for machine learning entails preparing and recovering raw data into a viable ML Algorithm. We have an AutoTrader dataset with 402006 rows and 10 columns. It includes both categorical and numerical data. For prediction, we use a sample set of data of 20000.

Analysis of Distributions

Matplotlib and Seaborn were used to analyse and visualise the distribution of the adverts data set. visualized price, mileage and year_of_registration.

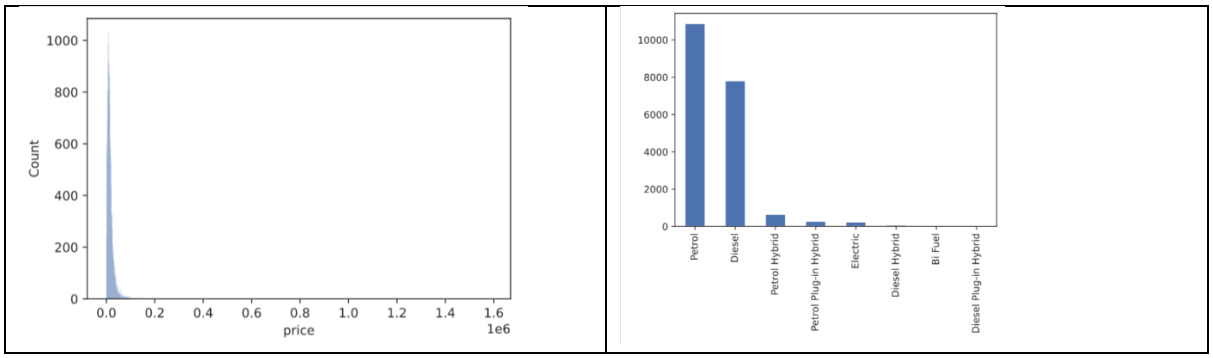

Fig 1: Boxplot of price and fuel type

Identification and dealing of missing values

To detect missing values I used the code **'adv.isnull().sum()**'. It determines the number of missing values in each column as well as the total number of missing values. Mileage, reg_code, standard_color, year_of_registration, body_type, and fuel_type are all missing data. With the help of mean, median, and mode, all missing values were removed.
I Visualized some features like price and mileage to find outliers and noise. Removed outliers from price and mileage using quantile method.

Noise

One problem with the vehicle's condition was discovered. When a car's condition status is USED, the mileage and registration year are both zero. It is erroneous because when the mileage is zero, the car should be brand new, in NEW condition, and the registration year should be the current year. Fixed the noise with a condition based on year of registration, mileage and vehicle condition.

Categorical Encoding with Ordinal Encoding

Utilizing ordinalEncoder to convert all categorical variables to numeric values. Converted standard_make, standard_colour, standard_model, body_type and fuel_type. Vehicle condition value USED and NEW changed to 0 and 1 with replace method.

Splitting Data into Predictors and Target

Splitting the dataset into predictors (features) and the target variable (variable to be predicted) is essential when training a machine learning model. Typically, the target is denoted by the output variable (y), whereas the predictors are denoted by the input variables (X). The desired result is price. So, features are separated into X and Y categories. price in the Y variable and others in the x variable.

2. Feature Engineering

The creation of new features or the transformation of existing ones to enhance the performance of a model is known as feature engineering. Here are some typical methods for feature engineering, including features derived from domain knowledge, polynomial/basis functions, and interaction features.

Polynomial/Basis Functions and Domain Knowledge-Based Features

Build features that capture relevant information using domain-specific insights or expert knowledge. By multiplying or exponentiating previously existing features, polynomial features involve the creation of new features.
Introducing new feature based on current year and year of registration, ie. **Age** of vehicle

```
[69] adv['Current_Year']=2023
     adv['year_of_registration'] = adv['year_of_registration'].astype(int)
     adv['age'] = adv['Current_Year']-adv['year_of_registration']
```

Fig 2: New Feature Age

After integrating a new feature, I removed the year of registration and the current year and some other features from data set. Some features that aren't necessary in my perspective for predicting car sales were removed that are public_reference, year_of_registration, Current_year, reg_code, croosover_car_and_van

The Scikit-Learn Way for Generating Interaction Features

Scikit-learn's preprocessing tools and feature engineering methods can be used to produce interaction features.

A PolynomialFeatures transformer instance should be created.

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

Fig 3: Polynomial Feature pipeline

The polynomial transformer is used to fit and change the data. The PolynomialFeatures transformer will generate new features by combining these features raised to different powers.

```
[74] preprocessing_pipe.fit_transform(X_train).head()
```

	mileage	standard_colour	standard_make	standard_model	vehicle_condition	body_type	fuel_type	age	mileage standard_colour	mileage standard_make	...	standard_model vehicle_condition
14223	0.056968	-1.207174	1.176505	-0.889372	-0.258189	-1.560455	0.755056	0.554959	-0.068770	0.067023	...	0.229626
4518	-1.209583	-1.207174	0.531955	-0.499910	3.873134	-0.632908	-1.242934	-0.578885	1.460177	-0.643444	...	-1.936219
10252	1.570918	0.813290	0.263392	0.170193	-0.258189	1.222188	-1.242934	0.451882	1.277612	0.413768	...	-0.043942
9138	-0.030295	1.279551	1.230218	0.714294	-0.258189	-0.632908	0.755056	0.039575	-0.038764	-0.037269	...	-0.184423
14532	-0.504758	-1.207174	-1.509121	-1.296016	-0.258189	1.531370	-1.242934	-0.166578	0.609331	0.761741	...	0.334617

5 rows × 36 columns

Fig 4: Feature Transformation

It basically broadens the feature space, allowing the model to learn more complicated patterns and potentially increase its prediction ability. Each column represents an interaction feature.

3. Feature Selection and Dimensionality Reduction

Dimensionality reduction and feature selection are essential stages of both machine learning and data analysis. Section 1 has already done some dimensionality reduction. I used a category encoder with ordinal encoder to reduce the number of categories. To get the most from both domain knowledge and algorithmic approaches, it is frequently advised to mix manual selection and automated processes

Automated Feature Selection (AFS): Univariate

AFS (Univariate Automated Feature Selection) is a machine learning technique that selects the most relevant features from a dataset based on their unique relationship with the target variable. It entails individually analysing each feature and picking the top-k features based on specific criteria.

Imported necessary modules for AFS which is selectKBest and f_regression. For regression prsoblems, the `f_regression` scoring function is commonly used.

```
[88] # could create a pipeline simply to get a pandas DataFrame as output
     selector = make_pipeline(
         SelectKBest(f_regression, k=7)
     ).set_output(transform='pandas').fit(X, y)
     X_sel = selector.transform(X)

[89] X_sel.head()
```

	mileage	standard_make	standard_model	vehicle_condition	body_type	fuel_type	age
0	95253.0	5.0	26.0	0.0	5.0	1.0	7.0
1	21900.0	21.0	579.0	0.0	5.0	5.0	9.0
2	18855.0	38.0	296.0	0.0	11.0	5.0	6.0
3	47000.0	34.0	151.0	0.0	11.0	5.0	6.0
4	88000.0	28.0	241.0	0.0	11.0	1.0	9.0

Fig 5: f_regression and output

Sequential Feature Selection (SFS) (Forward/Backward)

Another widely used strategy in machine learning for feature selection is sequential feature selection (SFS). SFS can be carried out in a forward or a backward direction, each with a different methodology.

Forward Sequential Feature Selection

The forward sequential feature selection iteratively selects the best feature at each stage. Here it selected essential and main features for the algorithm to get the best accurate.

```
[90] from sklearn.feature_selection import SequentialFeatureSelector

     sfs_forward = SequentialFeatureSelector(
         Ridge(), n_features_to_select=7, direction="forward"
     ).fit(X, y)

[92] sfs_forward.get_feature_names_out()

     array(['mileage', 'standard_colour', 'standard_make', 'standard_model',
            'vehicle_condition', 'body_type', 'fuel_type'], dtype=object)
```

Fig 6: forward sequential features

4. Model Building

The process for building models is known as machine learning. Model selection is the process of selecting an appropriate algorithm based on the type of issue (regression or classification) and data properties

4.1 Linear Model

A linear model is a sort of model that assumes a linear relationship between the input features and the target variable.

Pre-Processing and Model Fitting Pipelines

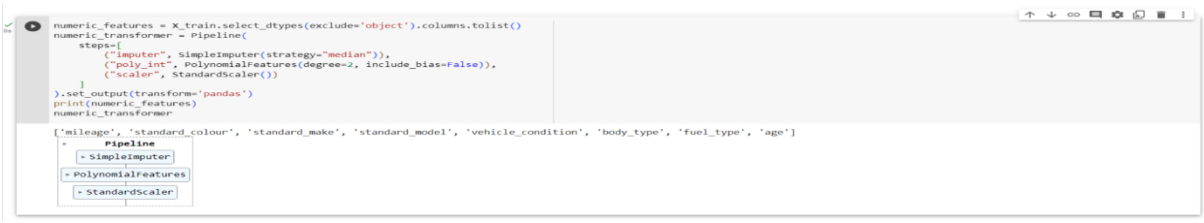

Fig 7: numeric transformer pipeline

For the model, I defined a numerical transformer pipeline. The pipeline consists of three steps: imputing missing data using the median, applying polynomial features with a degree of 2, and scaling the features using standardisation.

Also created another pipeline for categorical features, The pipeline categorical_transformer is intended for preprocessing categorical features. SimpleImputer replaces missing values in categorical features with the most frequent value (mode) of each feature.

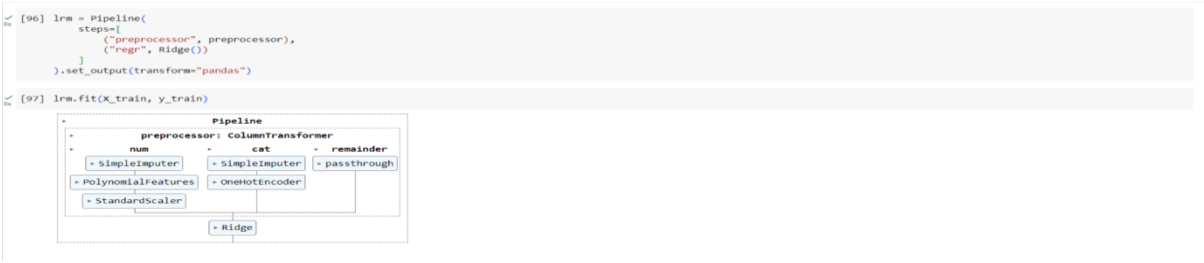

Fig **8**: pipeline of preprocessor and Ridge regression

Here, I developed a pipeline for the linear model Ridge, which conducts reprocessing steps, and the Ridge Regression model.

Linear Model Evaluation

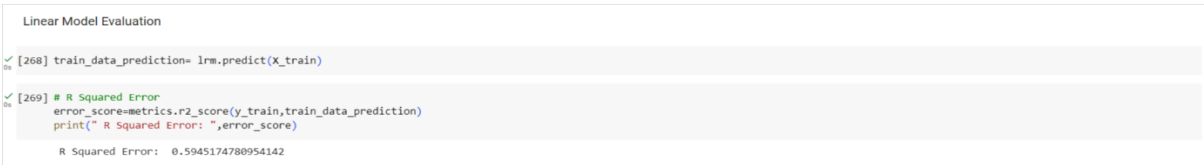

Fig 9: Linear model evaluation with R squared error

The mean absolute error calculated between y_test and X_test is 3866.42, while between y_train and X_train, it is 3816.66. The Ridge regression model can explain about 59.45% of the variance in the target variable with an R-squared value of 0.5945.

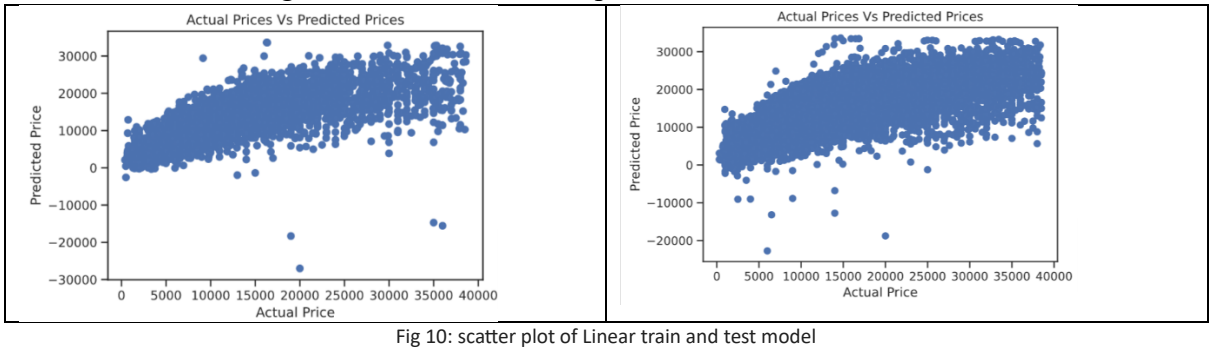

Fig 10: scatter plot of Linear train and test model

Based on these measures, it appears that the Linear model performs similarly on test and train data. However, that the model's may be relatively low.

4.2 Random Forest Regression

Random Forest Regression is a machine learning algorithm in the ensemble learning category.

```
[279] #importing randomForestRegression
      from sklearn.ensemble import RandomForestRegressor

[280] rfr = Pipeline(
          steps=[
              ("preprocessor", preprocessor),
              ("regr", RandomForestRegressor())
          ]
      ).set_output(transform="pandas")

[281] rfr.fit(X_train, y_train)
```

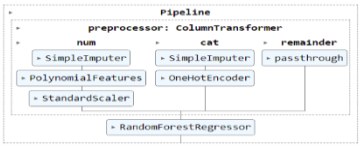

Fig 11: Random Forest Regressor

Here, I developed a pipeline for the Random Forest Regression model, which conducts preprocessing steps, and the Random Forest Regression.

```
[282] mean_absolute_error(y_test, rfr.predict(X_test))
      1959.0513021547972

[283] mean_absolute_error(y_train, rfr.predict(X_train))
      748.2423718901729

[286] rfg_y_pred= rfr.predict(X_test)
      rfg_y_pred.shape
      (4571,)

[287] rfg_slr_score = rfr.score(X_test,y_test)
      rfg_slr_score
      0.8608679585311688

[288] rfg_slr_score = rfr.score(X_train,y_train)
      rfg_slr_score
      0.9800954140393
```

Fig 12: Mean absolute error and R squared score of rfr

The average absolute error of the y_test and X_test is 1959.051. The y_train and X_train mean absolute error is 748.242.

The combined score of y_test and X_test is 0.86, while the combined score of y_train and X_train is 0.98. It considered good for our dataset.

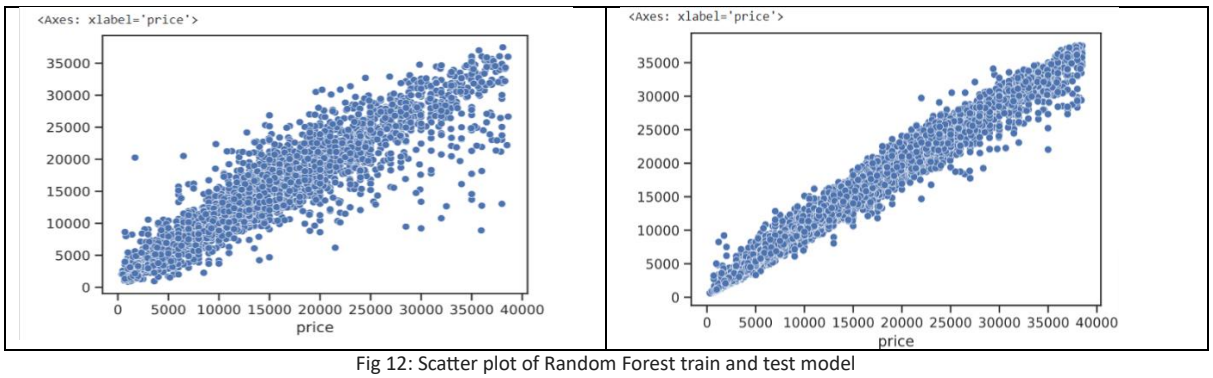

Fig 12: Scatter plot of Random Forest train and test model

The Random Forest Regression model appears to perform better on train data than on test data, as indicated by the lower MAE and higher R-squared score on train data. This could be

an indication of overfitting, in which the model fits the training data very well but does not generalise well to test data.

4.3 A Boosted Tree

A boosted tree model, sometimes referred to as gradient boosting, is an ensemble machine learning technique that combines different decision trees to produce an effective prediction model.

Model Fitting with Gradient Boosting Regressor

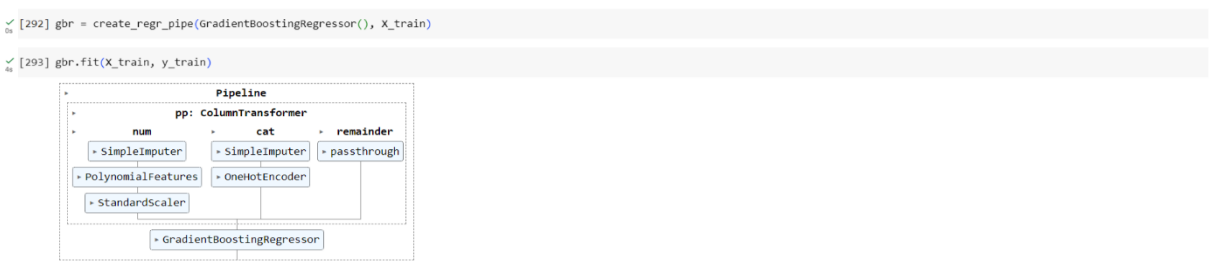

Fig 13: Gradient Boosting Regressor

Defined Gradient Boosting Regressor in pipeline

Fig 14: Score of Gradient Boosting Regressor

The mean and standard deviation of the Gradient Boosting Regressor's cross-validation score were determined. The average value is 3792.4980, while the standard deviation is 133.455.

As a result, with a mean score of 3792.4980 and a standard deviation of 133.455, the Gradient Boosting Regressor appears to have reasonably consistent performance on the training data, with an average RMSE of about 3792.4980.

Model Fitting with HistGradientBoostingRegressor

HistGradientBoostingRegressor is a gradient boosting variation that use histogram-based methods for effective training and prediction.

Fig 15: Hist Gradient Boosting Regressor

Created HistGradientBoostingRegressor with the data X_train and defined it.

The mean and standard deviation of the Hist Gradient Boosting Regressor's cross-validation score were determined. The average value is 2884.6062, while the standard deviation is 96.360.
As a result, with a mean score of 2884.6062 and a standard deviation of 96.360, the HistGradientBoostingRegressor model looks to have reasonably consistent performance on the training data, with an average RMSE of about 2884.6062.

4.4 An Averager / Voter / Stacker Ensemble

An Averager/Voter/Stacker ensemble is a mixture of ensemble techniques used in model construction. It entails merging the predictions of numerous models using averaging, voting, and stacking approaches.

Ensembing Voting regressor

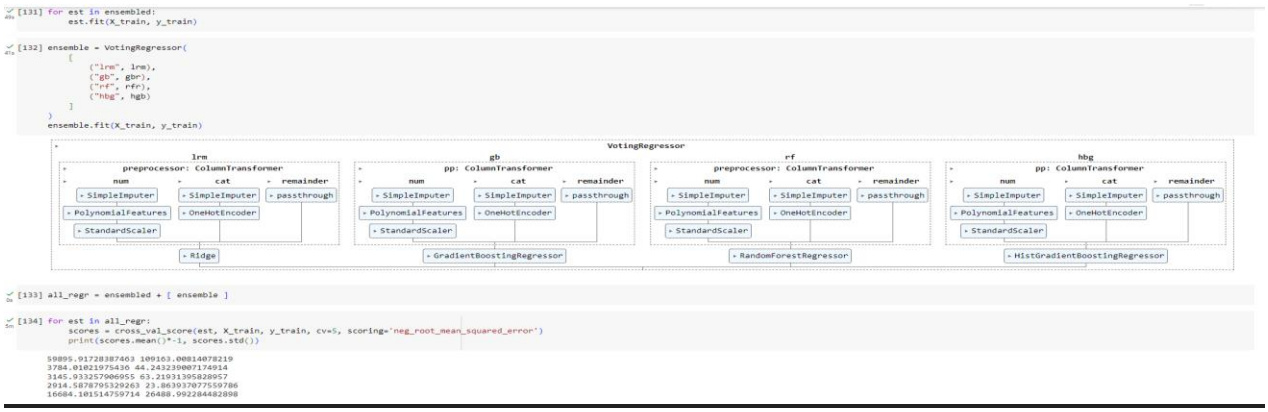

Fig 16: Ensemble models

Here I ensemble 4 models, which are a linear model, a random forest regression, a gradient boosting regression, and a histogram gradient boosting regression.

The linear model with Ridge Regression (lrm) has a mean RMSE of 3784.01021975436.
The Gradient Boosting Regressor (gb) has a mean RMSE of 3145.933257906955.
The Random Forest (rf) has a mean RMSE of 2914.5878795329263.
The Hist Gradient Regressor (hgb) has a mean RMSE of 16684.101514759714.
The ensemble model has a mean RMSE of 59895.91728387463. In rough work, grid search and ranking are performed.

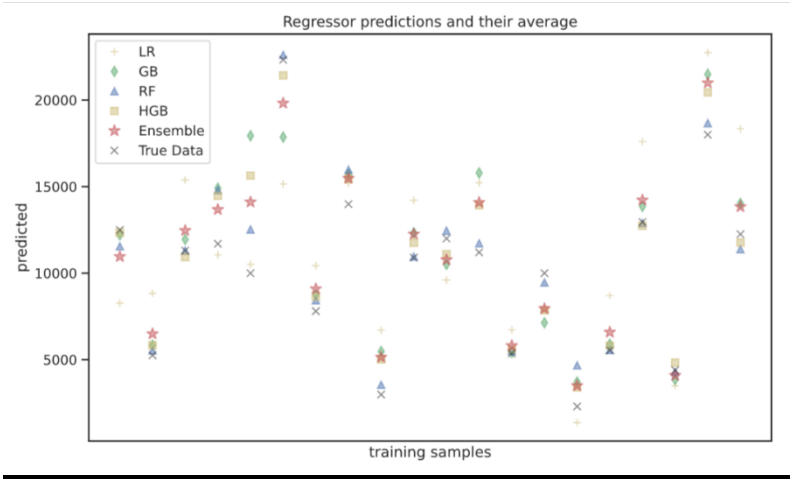

Fig 16: Regressor predictions and their average

5. Model Evaluation and Analysis

5.1 Overall Performance with Cross-Validation

In section 4.4, I already performed cross validation analysis on four models (Linear Model, Random Forest Regressor, Gradient Boosting Regressor, and Hist Gradient Boosting Regressor). I'm assessing and analysing the models here.

For the ensemble model:
Mean neg_root_mean_squared_error: 59895.91728387463
Standard deviation: 109163.00814078219

For the linear model with Ridge Regression (lrm):
Mean neg_root_mean_squared_error: 3784.01021975436
Standard deviation: 44.243239007174914

For the Gradient Boosting Regressor (gb):
Mean neg_root_mean_squared_error: 3145.933257906955
Standard deviation: 63.21931395828957

For the Random Forest (rf):
Mean neg_root_mean_squared_error: 2914.5878795329263
Standard deviation: 23.863937077559786

For the Hist Gradient Regressor (hgb):
Mean neg_root_mean_squared_error: 16684.101514759714

Standard deviation: 26488.992284482898

To determine the best model, we typically look for the one with the lowest mean neg_root_mean_squared_error (RMSE) value. Based on the above results the Random Forest (rf) has the lowest mean RMSE, indicating higher performance in predicting the target variable. As a consequence of the supplied results, the Random Forest Regressor model looks to be the best model.

Recursive Feature Elimination (RFE)

A feature selection method frequently used in machine learning is called recursive feature elimination (RFE). Imported RFECV method from feature selection and implemented on Random Forest Regression.

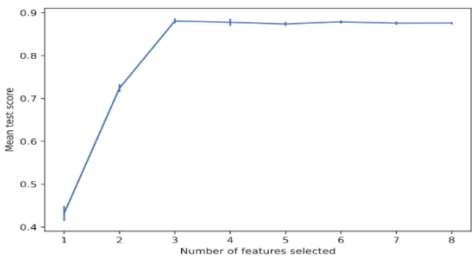

Fig 17: Plot of RFE

5.2 True vs Predicted Analysis

The scatter plot will visualize the relationship between the true and predicted values.

I made a plot with subplots for the true vs predicted analysis. The axes in the scatterplot above are designated True Target and Predicted Target.

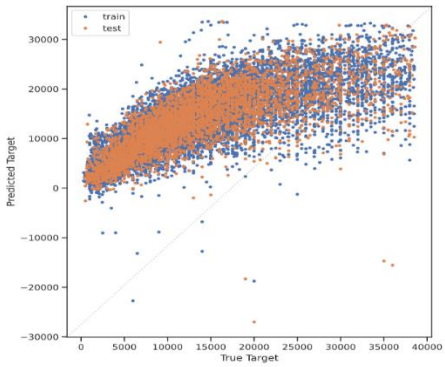

Fig 18: True vs Predicted scatter plot

This plot's pattern appears to be clustered. It indicates a good relationship by pointing from the left to the upper right. However, when it comes to estimating accuracy, train and test data are similar. The majority of data points are not near to the diagonal line, showing a significant correlation between true and expected values. Based on the results of this scatterplot, I can conclude that the Random Forest regressor can produce significant accuracy.

5.3 Global and Local Explanations with SHAP

A technique called SHAP (SHapley Additive exPlanations) is used to justify the predictions made by machine learning models. By giving input characteristics relevance values that reflect their contribution to the model's output, it offers both global and local explanations.

```
[151] X_pp = rfr['preprocessor'].transform(X_test)

[152] X100 = shap.utils.sample(X_pp, 100) # 100 instances for use as the background distribution

[153] explainer = shap.Explainer(lrm['regr'].predict, X100)
      shap_values = explainer(X_pp)

      Permutation explainer: 4572it [02:01, 34.93it/s]
```

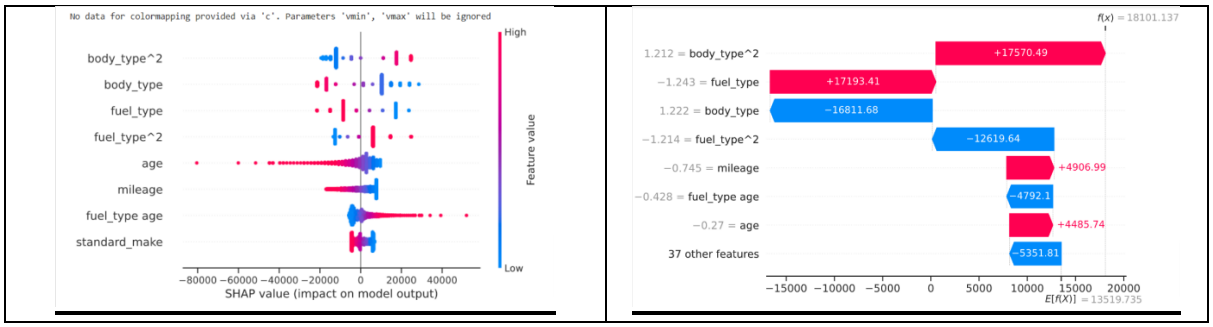

Fig 19: SHAP Tree and waterfall

The SHAP summary plot above explains the high and low values as well as their ramifications. It has both bad and positive characteristics. The feature value of fuel_type age is high. This suggests that fuel type and new car features are important in predicting selling price. Body_type2 and fuel_type2 are also important features. Age has a strong negative

SHAP value; older age has a negative impact on car prices. The importance of standard_make in forecasting selling price is lower.

5.4 Partial Dependency Plots

Individual Conditional Expectation (ICE)

Individual Conditional Expectation (ICE) plots are a type of partial dependence plot (PDP) that shows the relation between a given feature and the predicted outcome in a machine learning model in greater detail.

Partial dependence plots (PDPs) are used in machine learning models to assess the relationship between a single feature and the projected outcome. They illustrate the marginal effect of a feature on the model's predictions while maintaining other features constant.

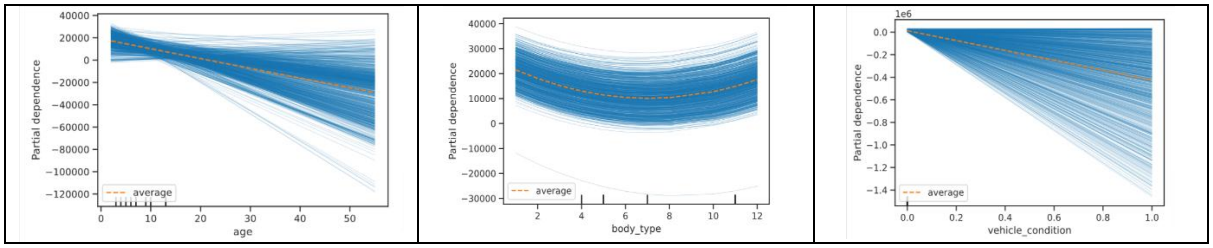

Fig 20: Partial dependency plot of Age

When the age is under 10, there is a high level of dependency in this segment. It demonstrates the positive reliance. However, as one gets older, that is after the age of ten, dependency decreases.

A 2D one gives us the possibility of analysis feature interactions (just the PDP curve; not feasible to show ICEs as well).

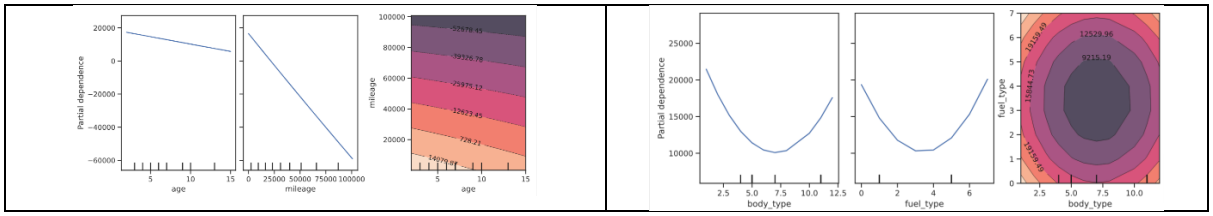

Fig 21: 2D partial dependency plot of age and mileage

Random Forest performs better at predicting the target variable, according to the score findings, as indicated by its lowest mean RMSE.

Therefore, I have decided that Random Forest Regression is the best predictive algorithm for this data set to estimate the selling price of a car using the provided features. The primary characteristics are fuel_type, body_type, age, mileage, and vehicle_condition.