Machine Learning Project

6G7V0015 Machine Learning Concepts

Name: Ishwari Jigajinni

MMU-ID: 23707865

1. Data/Domain Understanding and Exploration

1.1. Meaning and Type of Features; Analysis of Distributions

The dataset *adverts* is used to train the machine learning model to predict the price of the car by using the features given. The dataset is saved in a pandas data frame called '**data**'. The dataset given for car prediction using regression models is a huge dataset with 402005 rows and 12 columns. A sample data is inspected using ***data.sample(5),*** it displays us the 5 randomly selected rows or instances. Using ***data.head(),*** we can observe the first 5 instances.

```
[9]: data.sample(5)
```

[9]:

rence	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	crossover_car_and_van	fuel_type
47700	59000.0	Y	Red	SKODA	Fabia	USED	2001.0	995	Estate	False	Petrol
97980	7046.0	69	Black	Mercedes-Benz	C Class	USED	2019.0	27195	Estate	False	Diesel
75338	5000.0	68	Grey	Fiat	500	USED	2018.0	8295	Hatchback	False	Petrol
05008	69532.0	64	Black	Mercedes-Benz	SL Class	USED	2014.0	21500	Convertible	False	Petrol
62682	26000.0	64	Red	MINI	Hatch	USED	2014.0	8490	Hatchback	False	Petrol

After checking the shape of the data using ***data.shape***, the ***data.info()*** is used to know about the type of data stored in each of the columns, there are 11 features and 1 target variable. Price of the car is our target variable. The other features are: *'public_reference', 'mileage', 'reg_code', 'standard_colour', 'standard_make', 'standard_model', 'vehicle_condition', 'year_of_registration', 'body_type', 'crossover_car_and_van', 'fuel_type'*.

- The numerical features *like 'public_reference', 'mileage', and 'year_of_registration'* represent the 15-digit reference id of each car, miles run by car till date, and year the car was first registered respectively.
- The categorical features like *'reg_code', 'standard_colour', 'standard_make', 'standard_model', 'vehicle_condition', 'body_type', and 'fuel_type'* represents the registration code of the car, the color of the car, the car brand, car model in that specific brand, Vehicle condition as in used or new, the car body, fuel type of the car respectively.
- *crossover_car_and_van* is boolean which says if the car is crossover or not.

data.describe() is used for knowing the descriptive analysis of numerical features. The *mileage* has a maximum of 999,999 while the mean is 37,743, this says that there are error values in *mileage*. Similarly, *year_of_registration* and *price* also have many error values. They need to be handled by trimming the data.

```
[10]: print(data.describe())
```

```
       public_reference        mileage  year_of_registration         price
count      4.020050e+05  401878.000000         368694.000000  4.020050e+05
mean       2.020071e+14   37743.595656           2015.006206  1.734197e+04
std        1.691662e+10   34831.724018              7.962667  4.643746e+04
min        2.013072e+14       0.000000            999.000000  1.200000e+02
25%        2.020090e+14   10481.000000           2013.000000  7.495000e+03
50%        2.020093e+14   28629.500000           2016.000000  1.260000e+04
75%        2.020102e+14   56875.750000           2018.000000  2.000000e+04
max        2.020110e+14  999999.000000           2020.000000  9.999999e+06
```

After running ***data.isnull().sum()***, we see that the data also has null values in mileage, year_of_registration, reg_code, standard_colour, body_type, and fuel_type. After running ***data['categorical_features'].nunique()***, we see the number of unique values present in each categorical column. We see that there are 1168 unique values in *standard_model* and 110 unique values in *standard_make*. The data has high cardinality due to these features.

1.2. Analysis of Predictive Power of Features

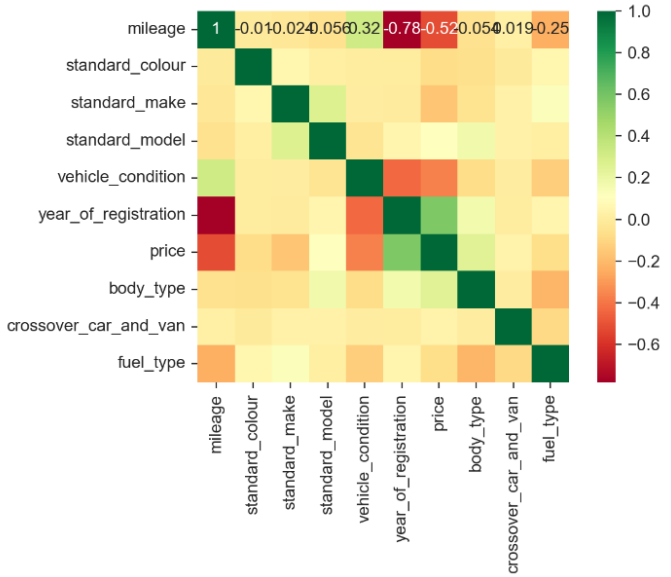
By understanding the feature meaning and domain knowledge, the predictive power of features can be analyzed.

- The *public_reference* which contains the 15-digit car ID is not of much use. So, we are dropping the column using ***data.drop('public_reference', axis=1)***
- The registration code of the car does not influence the price. It only tells us the age of the car. So, this is also dropped after using it to fill the null values in *year_of_registation*.
- Mileage the car has run has a great impact on the car, as it indicates the shelf life left for the engines.
- Year_of_registration gives the age of the car, so it influences the price.
- Vehicle_condition defines if the car is used or new. New cars are costlier than the used ones.
- Features like standard_model, standard_make, and standard_colour affect the price, some luxury brands like Tesla, Mercedes, BMW, Audi, etc have higher prices than normal brands. Also, some specific models have higher prices. Some rare colours like silver will be expensive.

To know how one feature is affecting the other one we use correlation function ***data.corr().*** It gives us the matrix which represents the effect of one feature on one another. From plotting the heatmap of the matrix, we can see the correlation between features.

```
sns.heatmap(data_sample.corr(), square=True, annot=True, cmap='RdYlGn')
```

```
<Axes: >
```



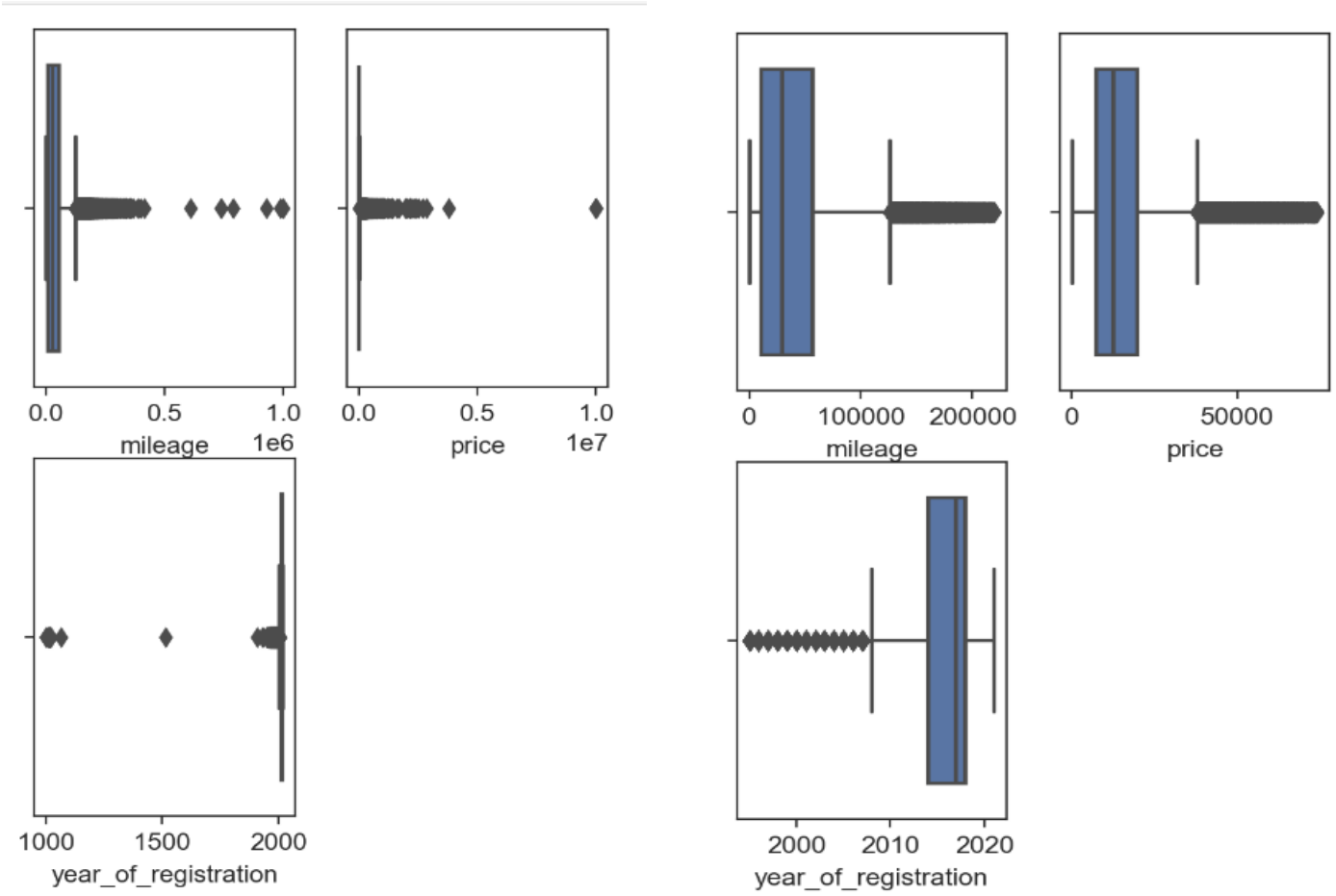
From the plot above, it is evident that *mileage, vehicle_condition, year_of_registration,* and *body_type* are the 4 important features that affect the price.

The red in the heatmap for *mileage* represents that the *price* and *mileage* are inversely proportional, similarly the orange in *vehicle_condition* and *standard_make* represents the inversely proportional relation with price.

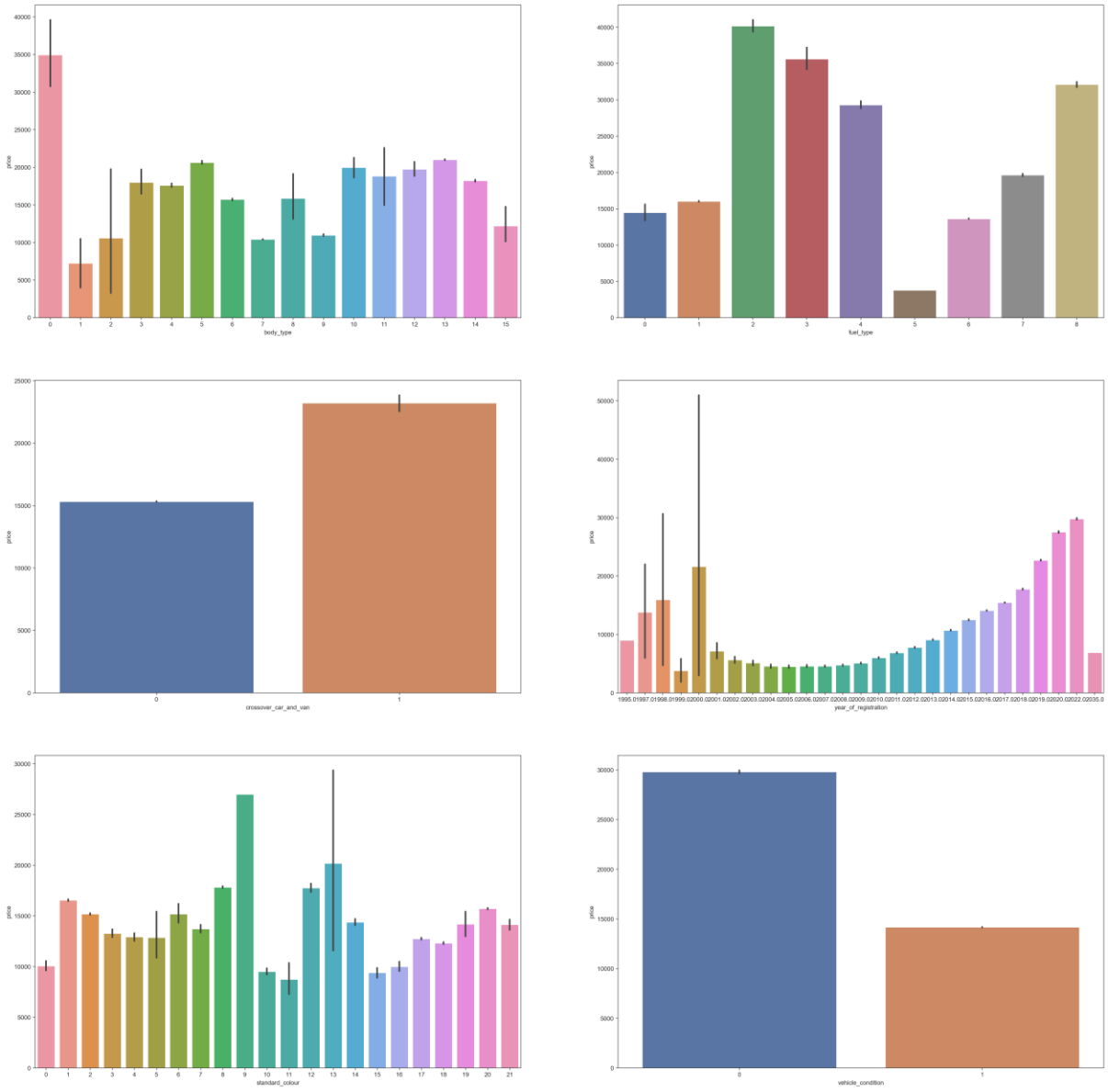
The green in the heatmap for *year_of_registration* and *body_type* presents the directly proportional relation with price.

1.3. Data Processing for Data Exploration and Visualisation

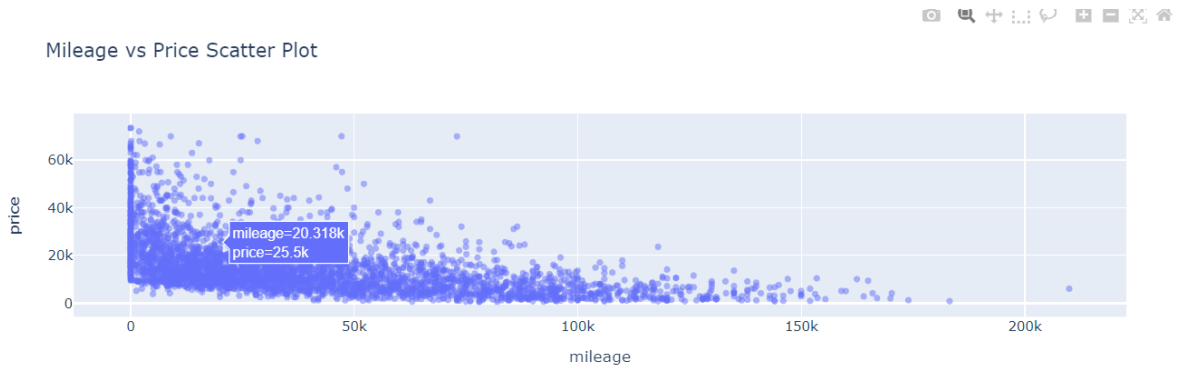
The box plot of numerical features on the left is the one before handling outliers the box plot of silver features can be seen on the right after the outliers are being handled using IQR method.



After encoding categorical features using a label encoder, The dependency of features on the target variable price is Visualized using a barplot for all the features as shown below



The interactive scatter plot for mileage versus price is Visualized as shown below.



2. Data Processing for Machine Learning

2.1. Dealing with Missing Values, Outliers, and Noise

Null Values:

The 127 null values in the *mileage* column are filled with the mean values. The 601 null values in column *fuel_type* are filled with the most frequently occurred values. This is done using the mode function.

Similarly, *body_type* and *standard_color* null values are also built with the most frequently occurring values

In the dataset, some of the new cars usually have null values in the year of registration and registration code as they have not been registered yet. So, the value 2021 has been filled for the new cars that have null in the year of registration.

Car year of registration can be found in the registration code in the UK. After 2001, if the car has been brought in the first six months of a year, then the registration code will be the last two digits of the year. If the car has been brought in the last six months of a year, then the registration core will be the last two digits of the year plus 50. Using this logic, the null values in the year of registration can be found from *reg_code* the code for this is given below.

The rest of the null values are dropped from the data set.

```
replacement_value = 2021
data.loc[(data['year_of_registration'].isnull()) &  (data['vehicle_condition'] == 'NEW'), 'year_of_registration'] = replacement_value

# Iterate over  all rows
for index, row in data.iterrows():
    reg_code = row['reg_code']
    if pd.notna(reg_code):
        if reg_code.isnumeric() and int(reg_code) > 50:
            data.at[index, 'year_of_registration'] = int(reg_code) + 1950
        elif reg_code.isnumeric() and int(reg_code) <= 50:
            data.at[index, 'year_of_registration'] = int(reg_code) + 2000
        else:
            # If 'reg_code' is a letter, delete the row
            data = data.drop(index)
```

Outliers:

The outliers are detected using the IQR method. IQR is a statistical method to define the quarters and set a IQR range, and only keep the data when it is in between the upper bound and lower bound.

All the outliers of mileage, price, and year_of_registration are deleted from the data, using the code below.

```
columnss =data[['mileage', 'price','year_of_registration']]
for column in columnss:
        Q1 = data[column].quantile(0.10)
        Q3 = data[column].quantile(0.90)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        data = data[(data[column] >= lower_bound) & (data[column] <= upper_bound)]
```

Around 6000 outliers were deleted from the data. And the data is now cleaned and has no noise in it.

2.2. Feature Engineering, Data Transformations, Feature Selection.

Encoding All the categorical features of the data must be encoded so that models like linear regression, K-nearest neighbor Regression, and decision tree regression. If we use one hot encoding, the data will have 1000+ columns due to the high cardinality of *standard_model* and *standard_make*. The computation time will increase exponentially. So, the label econder becomes the ideal type of encoding for our data.

In label encoding, the labels are given to each different value in the categorical features. The labels are usually numbers. The Boolean type cross_over_car_and_van is converted to an integer.

Standard_make has values from 1- 93, standard_colour has 22, and standard_model has 906 values

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
categorical_features = data.select_dtypes(include=['object']).columns
for column in categorical_features:
    data[column] = le.fit_transform(data[column])

data['crossover_car_and_van'] = data['crossover_car_and_van'].astype(int)
```

Scaling:

Features like *year_of_registration* and *mileage* are on a different scale than our target variable price. If the model is trained on unscaled data, the linear regression and KNN regression will be less efficient. So, it is always advised to scale the data before Linear regression or KNN regression.

StandardScaler is a preprocessing technique in sci-kit-learn (sklearn) that standardizes a dataset by scaling each feature to have a mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler

scaler_data= StandardScaler()
data_scaled = scaler_data.fit_transform(data)
```

3. Model Building

3.1. Algorithm Selection, Model Instantiation and Configuration

This is a very crucial step in machine learning models. The linear regression, KNN regression, and decision tree regression are instantiated.

KNN regression works by predicting the value of a new data point based on the average or weighted average of its k-nearest neighbors target values. It measures similarity using distance metrics and assigns a prediction by aggregating the target values of the nearest neighbors in the feature space.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score, GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error, r2_score

knn_model = KNeighborsRegressor()
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train, y_train)

knn_predictions = knn_model.predict(X_val)
knn_rmse = mean_squared_error(y_val, knn_predictions, squared=False)
knn_r2 = r2_score(y_val, knn_predictions)
print("KNN RMSE:", knn_rmse)
print("KNN R^2:", knn_r2)
```

Linear regression models the relationship between a dependent variable and one or more independent variables by fitting a straight line to the observed data.

```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()

linear_model.fit(X_train,y_train)
lr_predictions=linear_model.predict(X_test)
lr_rmse = mean_squared_error(y_test, lr_predictions, squared=False)
lr_r2 = r2_score(y_test, lr_predictions)
print("RMSE:",lr_rmse)
print("R2 score:", lr_r2)
```

Decision tree regression works by recursively splitting the dataset into subsets based on features, identifying optimal breakpoints at each node to minimize the mean squared error. The final prediction is the average of the target variable within the leaf node reached by a new data point.

```
from sklearn.tree import DecisionTreeRegressor
dt_model = DecisionTreeRegressor()

dt_model = DecisionTreeRegressor(random_state=0)
dt_model.fit(X_train, y_train)
tree_predictions = dt_model.predict(X_val)
tree_rmse = mean_squared_error(y_val, tree_predictions, squared=False)
tree_r2 = r2_score(y_val, tree_predictions)
print("\nDecision Tree RMSE:", tree_rmse)
print("Decision Tree R^2:", tree_r2)
```

The target variable *price* is stored in a variable called *y*, all other features are stored in *X*, this data is split using the test train split function. It is split into 80% of the data for training and 20% of the data for testing.

The models are trained with *X_train* and *y_train*. After training the models the prices are predicted for test data. The RMSE and R2 scores are computed to check the efficiency of the models.

The values we get are tabulated below.

Model	RMSE	R2 score
KNN Regression	0.3644	0.8664
Decision tree Regression	0.3018	0.9083
Linear Regression	0.7597	0.4260

The decision tree model is highly efficient with 90% efficiency and the linear model is least efficient with 42% efficiency.

3.2. Grid Search, and Model Ranking and Selection

Grid search is a hyperparameter optimization technique where a predefined set of hyperparameter values is systematically tested to find the optimal combination. It involves creating a grid of hyperparameter values and evaluating the model's performance for each combination through cross-validation. The goal is to identify the configuration that maximizes the model's effectiveness on the given dataset.

- During the grid search procedure, a KNN regression model was optimized through fine-tuning to maximize its performance. The hyperparameter grid included n_neighbors values, which have three options: 3;5;7. By the negative mean squared error, GridSearchCV Cv explored these combinations through cross-validation. It was determined that the best configuration had ‘n_neighbors’ set at 3. This optimal model was then stored as ‘knn_best_model’. The RMSE of 0.3600 and R2 score equal to 87.10%.

```
# Grid search
knn_param_grid = {'n_neighbors': [3, 5, 7] }
knn_grid_search = GridSearchCV(knn_model, knn_param_grid, scoring='neg_mean_squared_error', cv=5, n_jobs=-1)
knn_grid_search.fit(X_train, y_train)
knn_best_params = knn_grid_search.best_params_
knn_best_model = knn_grid_search.best_estimator_
```

```
print("Best Parameters:", knn_best_params)
```

```
Best Parameters: {'n_neighbors': 3}
```

- During the grid search process, a decision tree regression model was optimized based on hyperparameter combinations. The grid exhibited options for ‘max_depth’ (maximum depth of the tree) and ‘min_samples_split'(min number required to split an internal node) among others. Using cross-validation, the best configuration was determined to be max_depth=20 and min_samples split 10. The optimal model saved as dt_best_model, shown to yield an RMSE of 0.2592 and an R2 score equal to 93.29%.

```
# grid search
dt_param_grid = {'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10]}
dt_grid_search = GridSearchCV(dt_model, dt_param_grid, scoring='neg_mean_squared_error', cv=5, n_jobs=-1)
dt_grid_search.fit(X_train, y_train)
dt_best_params = dt_grid_search.best_params_
dt_best_model = dt_grid_search.best_estimator_
```

```
print("Best Parameters:", dt_best_params)
```

```
Best Parameters: {'max_depth': 20, 'min_samples_split': 10}
```

- The linear regression does not have a parameter so there is no hyperparameter tuning, or grid search.

So, after this, the model efficiency is ranked in the following order, Decision tree with 93.29% then the KNN model with 87.10%, and the linear model with 42%.

4. Model Evaluation and Analysis

4.1. Coarse-Grained Evaluation/Analysis (3) (e.g., with model scores) 10% (1 page)

Cross-validation is a technique used in machine learning to assess a model's performance and generalization ability. It involves dividing the dataset into multiple subsets, typically k-fold.

All the models are trained on 4 folds and tested on the remaining fold, repeating this process 5 times. The performance metrics from each iteration are then averaged to provide a more reliable estimate of the model's effectiveness. Cross-validation helps ensure that the model is not overfitting to a specific subset of the data, providing a more robust evaluation of its ability to perform on unseen data.

RMSE is a measure of the average magnitude of errors between predicted and actual values in regression analysis. It is a measure of the quality that can be achieved by matching model predictions with observed data, where lower RMSE values correspond to better accuracy.

R-squared (R2) score, which is expressed as a proportion of the variance in the dependent variable that can be predicted from an independent. It varies from 0 to 1, whereby 1 shows the perfect fit indicating that everything in terms of variability is explained by a model and zero indicates no explanatory power. A larger value of R2 indicates a better fit for the model.

The RMSE values of all the models with the best parameters during cross-validation are noted in the table below.

For linear regression:

```
[0.75148856 0.75449476 0.75372919 0.76119868 0.75440698]
0.7550636332933518
```

For KNN regression:

```
K-Nearest Neighbors CV RMSE: 0.3746338288450649
[0.37446151 0.37532044 0.37760876 0.37589768 0.36983542]
```

For Decision tree:

```
[0.26727604 0.2701457  0.27146797 0.27173104 0.26528536]
Decision Tree CV RMSE: 0.26918122184307725
```

The R2 score of the KNN best model and the Decision tree best model, linear regression is given below, this table interprets that the decision model is best for our data.

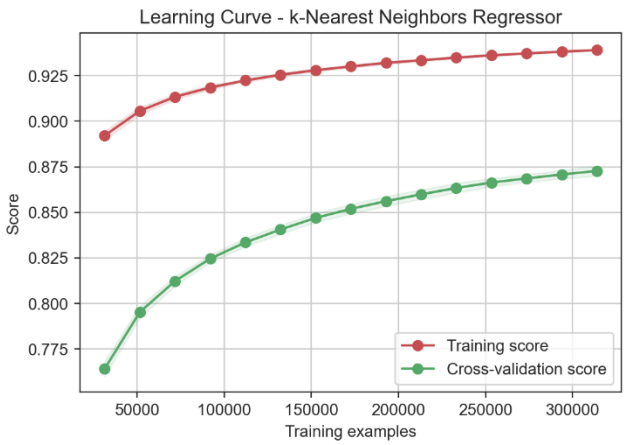
Model	R2 score
KNN_best_model (3 neighbors)	0.8710
Decision_tree _best model (max_depth- 20 and max_split- 10)	0.9329
Linear regression	0.7550

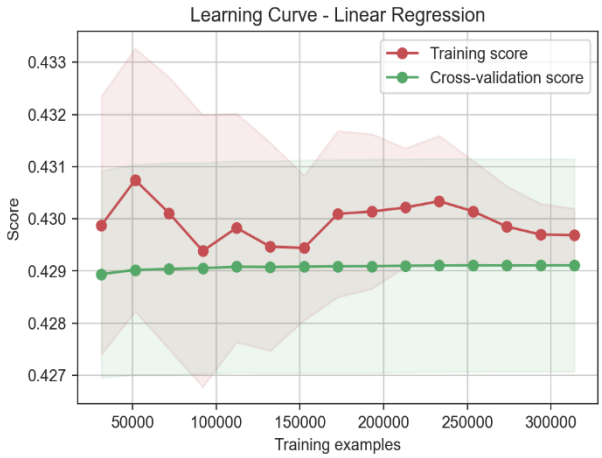
Learning curve:

The learning curve of decision tree shows that the training score keeps on increasing and becomes as table in later values of training examples.

Cross-validation score increases exponentially.

Both the curves seem converging, this means the model is training well.





The curve for KNN is like a decision tree curve but the values are a little lower for KNN. The training score for linear regression is very unstable while the validation score is constantly low throughout.

4.2. Feature Importance

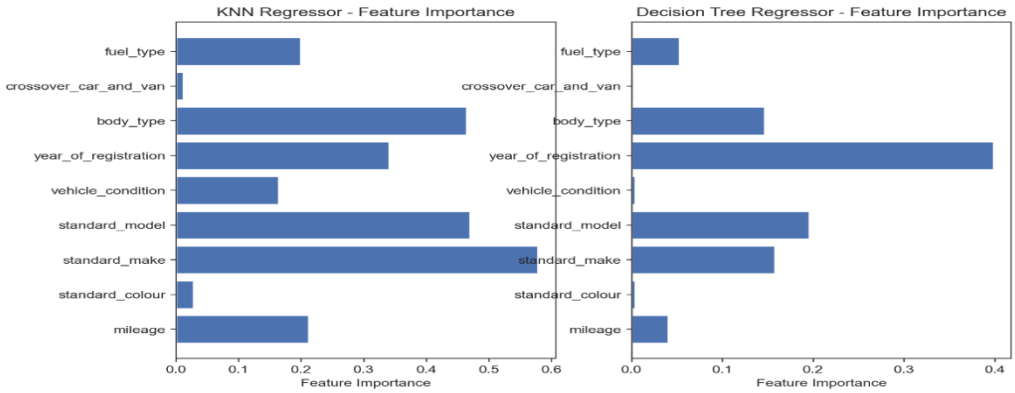
- The feature importance in the decision tree is checked for the model with the best parameters. The year of registration is the one which influences the price a lot. *Standard_model* and *standard_make* also have a great impact on the target variable.

	Coefficient
year_of_registration	0.398114
standard_model	0.195574
standard_make	0.157595
body_type	0.146442
fuel_type	0.052564
mileage	0.040104
standard_colour	0.004109
vehicle_condition	0.003622
crossover_car_and_van	0.001875

- In the linear regression, all the features are having same importance in our model, which makes it perform badly compared to others.
- The feature importance of KNN is checked and in that, the standard_make is of more importance, body_type and standard_model have 0.46 importance, and Crossover_car_and_van and standard_colour have the least importance in KNN.

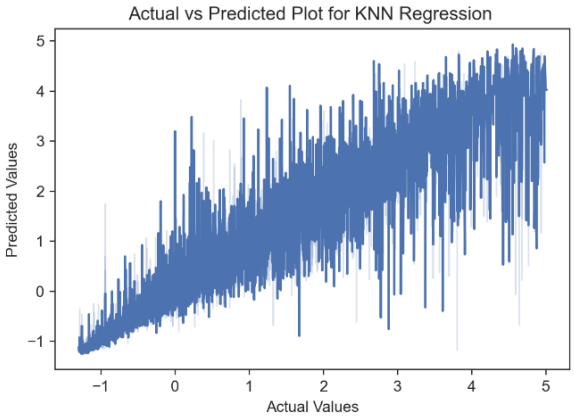
	Coefficient
standard_make	0.577493
standard_model	0.469372
body_type	0.464113
year_of_registration	0.339953
mileage	0.212086
fuel_type	0.198690
vehicle_condition	0.163050
standard_colour	0.027552
crossover_car_and_van	0.011533

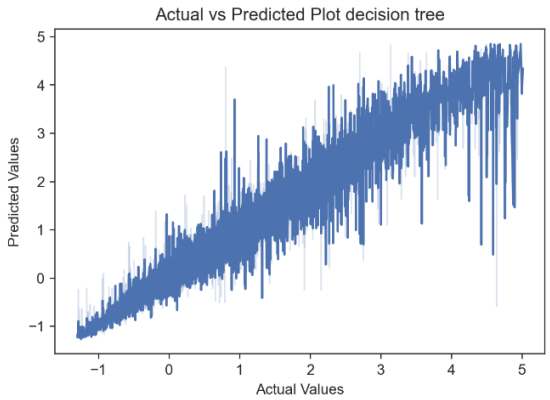
The feature importance of KNN and Decision tree are plotted below:

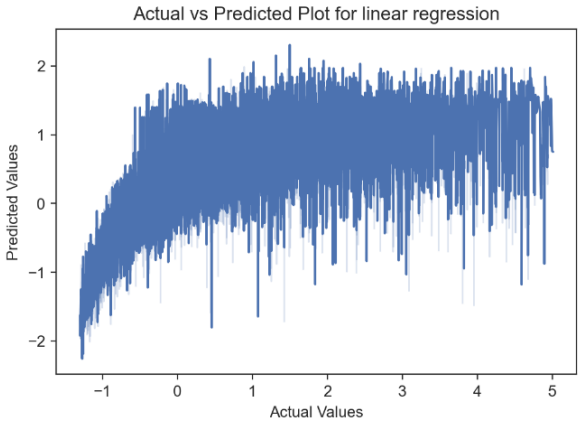


4.3. Fine-Grained Evaluation

Actual vs predicted values are plotted for all 3 models.

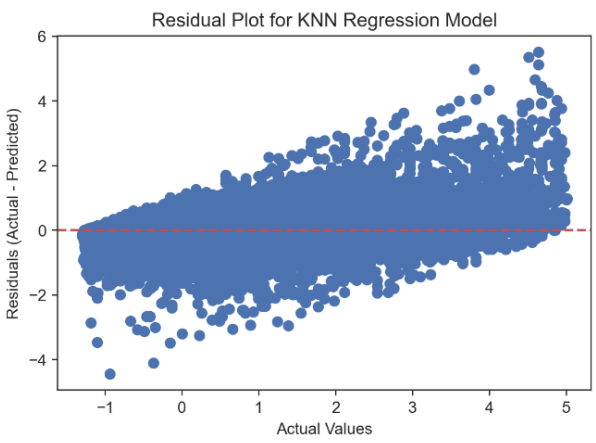


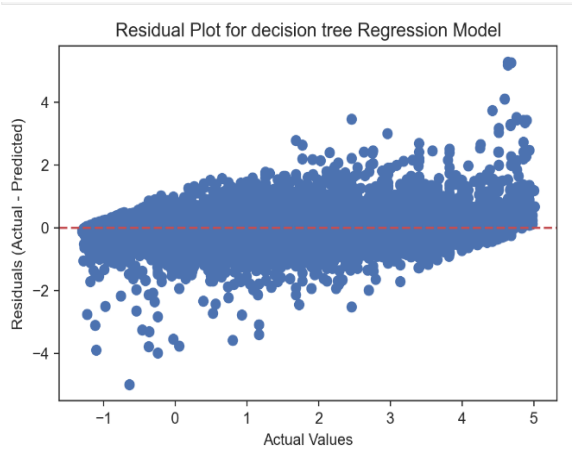


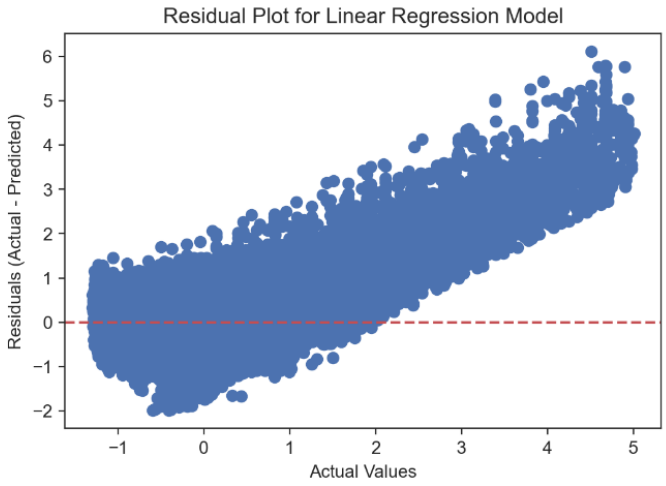


- The decision tree plot has a curve like a straight line with x=y. this shows that there is not much difference between actual and predicted values.
- The linear regression graph has a curve that is much less like a straight line passing through the center.

Residual plot







- The residual plot plots the difference between the actual and predicted values.
- Ideally model should have zero residual values, for it to predict all the values correctly.
- The plot of linear regression shows that the values are away from the y=0 line, it means the model is not predicting the correct values.
- For the Decision tree plot there are very few points away from the line, indicating that it is predicting correct values most of the times.
- The KNN also has many values near the y=0 line. It shows that even KNN model is having good efficiency.