Manchester Metropolitan University

Department of Computing and Mathematics

ASSESSMENT COVER SHEET 2023/24

Student Name:	**POPOOLA, AYOMIDE PETER**
Student Id:	**23655243**
Unit Code and Title	**6G7V0015 Machine Learning Concepts**
Assessment Set By:	**L Gerber**
Assessment ID:	**1CWK100**
Assessment Title:	**Machine Learning Project**
Type:	**Individual**
Environment:	**Jupyter Notebook**

1.0. DATA/DOMAIN UNDERSTANDING AND EXPLORATION

1.1. Meaning and Type of Features; Analysis of Distributions

The prediction of car price using machine learning is key in the car dealership industry, this concept will increase cars sales and profit by dealers over time. Various conditions determine the car price, using such prediction will help keep an up to date car sales price. In this project, the car sales advert dataset of 402,005 rows and 12 columns was provided by AutoTrader (AT) which is one of the university industry partners. The historical dataset used contains several information both the categorical features and numerical features which includes:

Public Reference: This is a unique value assigned to each vehicle advertised at a particular period of time, no two vehicles have the same public reference

Mileage: This is the distance (miles) the vehicle has travelled per gallon before its selling time

Registration Code: This refers to the code given to the vehicle at the point of registration, the registration codes for vehicles below year 2000 are represented using alphabet, while the registration code above year 2000 are numeric

Standard Colour: This is the colour of the vehicle at the selling point

Standard Make: This refers to the vehicle manufacturers for example BMW, Audi etc

Standard Model: This is the brand of the car given by the manufacturer based on the specifications and features

Vehicle Condition: This is the status of the vehicle at the point of selling, whether used or new

Year of Registration: This is the year the car was officially registered by the authority

Price: This is the amount the vehicle was sold in pounds

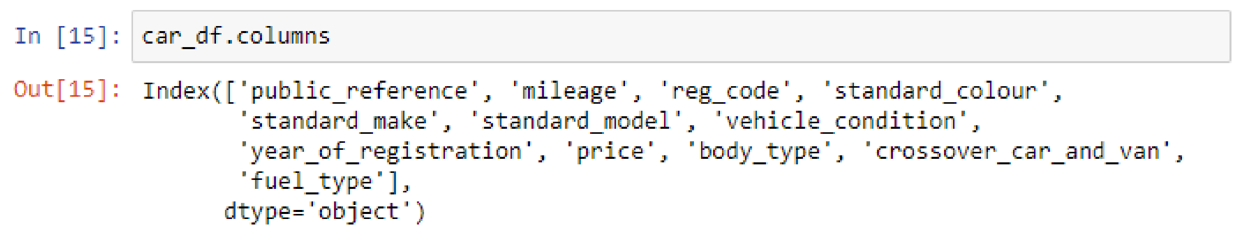
Body Type: This is the vehicle outer configuration or shape for example: hatchback, SUV etc

Crossover Car and Van: Crossover are vehicles that known for maintaining comfortability and fuel efficiency, it combines the features of SUV and passenger cars, while Vans are vehicles used for transporting goods and people

Fuel Type: This is the type of fuel that is used to power the vehicle’s engine to move for instance: petrol, diesel etc.

The categorical features are standard colour, standard make, standard model, vehicle condition, body type, crossover cars and van and fuel type

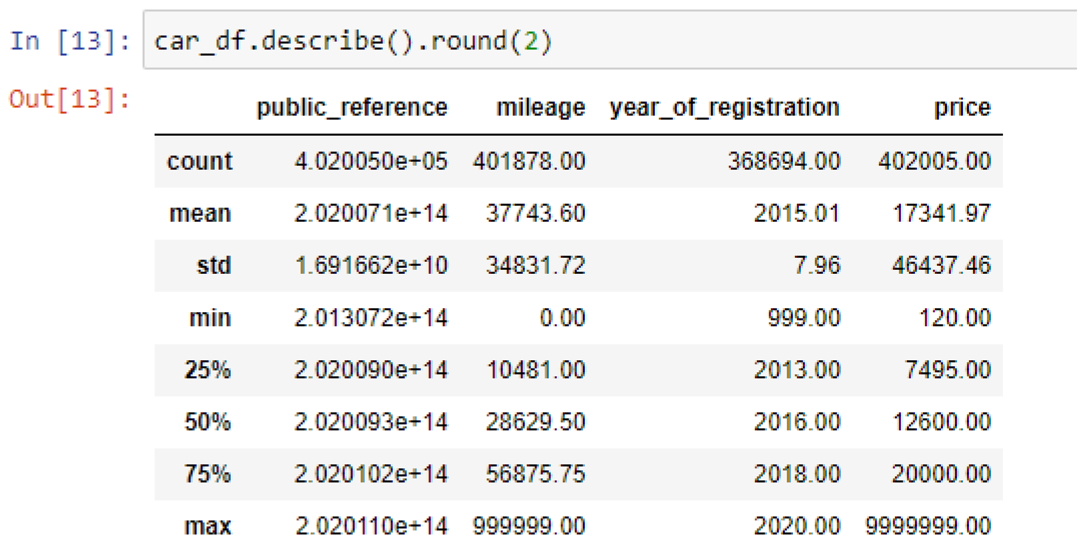
Numerical features are public reference, year of registration, price, reg code



```
In [15]: car_df.columns

Out[15]: Index(['public_reference', 'mileage', 'reg_code', 'standard_colour',
                'standard_make', 'standard_model', 'vehicle_condition',
                'year_of_registration', 'price', 'body_type', 'crossover_car_and_van',
                'fuel_type'],
               dtype='object')
```

Figure 1: Feature types

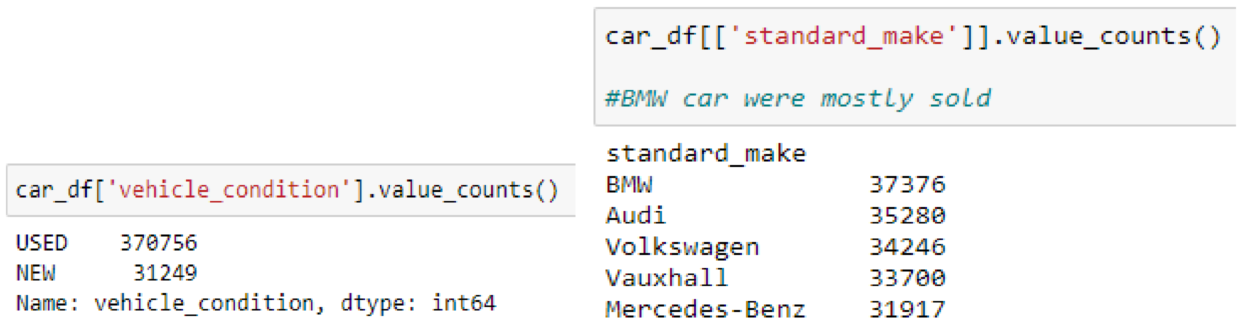
Analysis of Distribution: The numerical features are analysed in order to understand their distribution. A quick understanding of price indicates a mean of 17,342, and a standard deviation of 46,437, the interquartile range is between 7495 and 20000. The mean value for mileage and year of registration is 37,743 and 2015 respectively (figure 2). For categorical feature, the distribution of cardinality was carried out, indicating that the data contain more used vehicles and the most advertised car manufaturer vehicle is BMW.



```
In [13]: car_df.describe().round(2)

Out[13]:
```

	public_reference	mileage	year_of_registration	price
count	4.020050e+05	401878.00	368694.00	402005.00
mean	2.020071e+14	37743.60	2015.01	17341.97
std	1.691662e+10	34831.72	7.96	46437.46
min	2.013072e+14	0.00	999.00	120.00
25%	2.020090e+14	10481.00	2013.00	7495.00
50%	2.020093e+14	28629.50	2016.00	12600.00
75%	2.020102e+14	56875.75	2018.00	20000.00
max	2.020110e+14	999999.00	2020.00	9999999.00

Figure 2: Analysis of numerical feature distribution



```
car_df['vehicle_condition'].value_counts()

USED    370756
NEW      31249
Name: vehicle_condition, dtype: int64
```

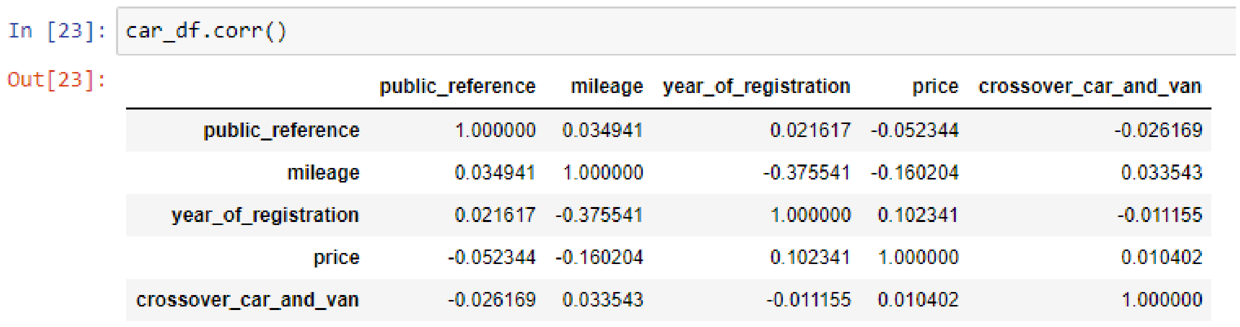
```
car_df[['standard_make']].value_counts()

#BMW car were mostly sold

standard_make
BMW              37376
Audi             35280
Volkswagen       34246
Vauxhall         33700
Mercedes-Benz    31917
```

Figure 3: Analysis of categorical feature distribution

1.2. Analysis of Predictive Power of Features

Based on the domain knowledge, it is guessed that features like mileage, standard model, standard make, fuel type , vehicle condition should have a high predictive power because these are the main determinant of the price value placed on a vehicle, but using the correlation matrix, there is a positive correlation between price and year of registration indicating that the price increases as the year increase, while the correlation between the mileage and price is negative, this indicates that the price decreases as the mileage increases.

Public reference and crossover car and van has no correlations with any of the other features, so they have a low predictive power

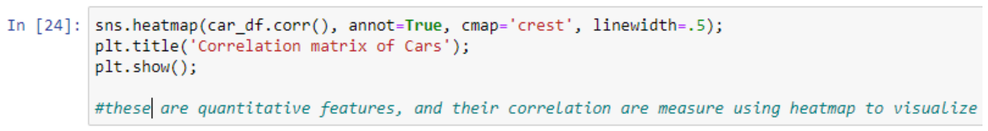


```
In [23]: car_df.corr()
```

Out[23]:

	public_reference	mileage	year_of_registration	price	crossover_car_and_van
public_reference	1.000000	0.034941	0.021617	-0.052344	-0.026169
mileage	0.034941	1.000000	-0.375541	-0.160204	0.033543
year_of_registration	0.021617	-0.375541	1.000000	0.102341	-0.011155
price	-0.052344	-0.160204	0.102341	1.000000	0.010402
crossover_car_and_van	-0.026169	0.033543	-0.011155	0.010402	1.000000

Figure 4: Correlation matrix



```
In [24]: sns.heatmap(car_df.corr(), annot=True, cmap='crest', linewidth=.5);
         plt.title('Correlation matrix of Cars');
         plt.show();

         #these are quantitative features, and their correlation are measure using heatmap to visualize
```

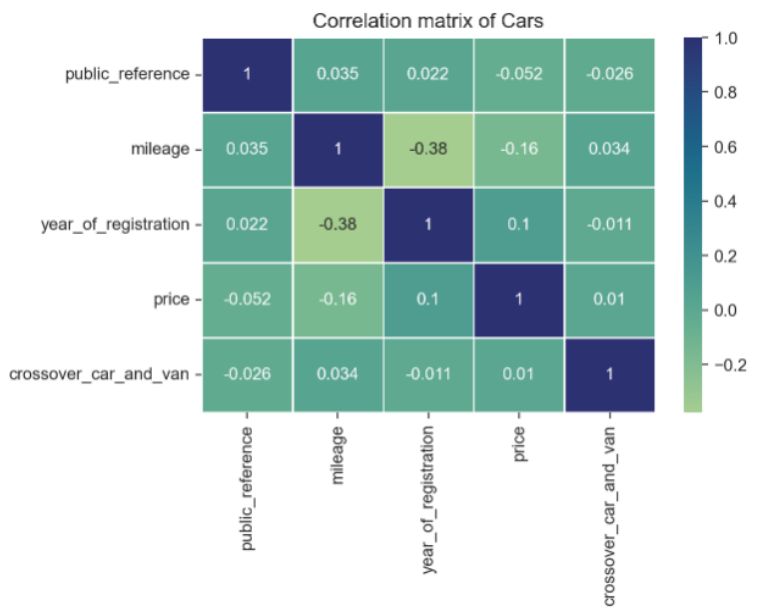
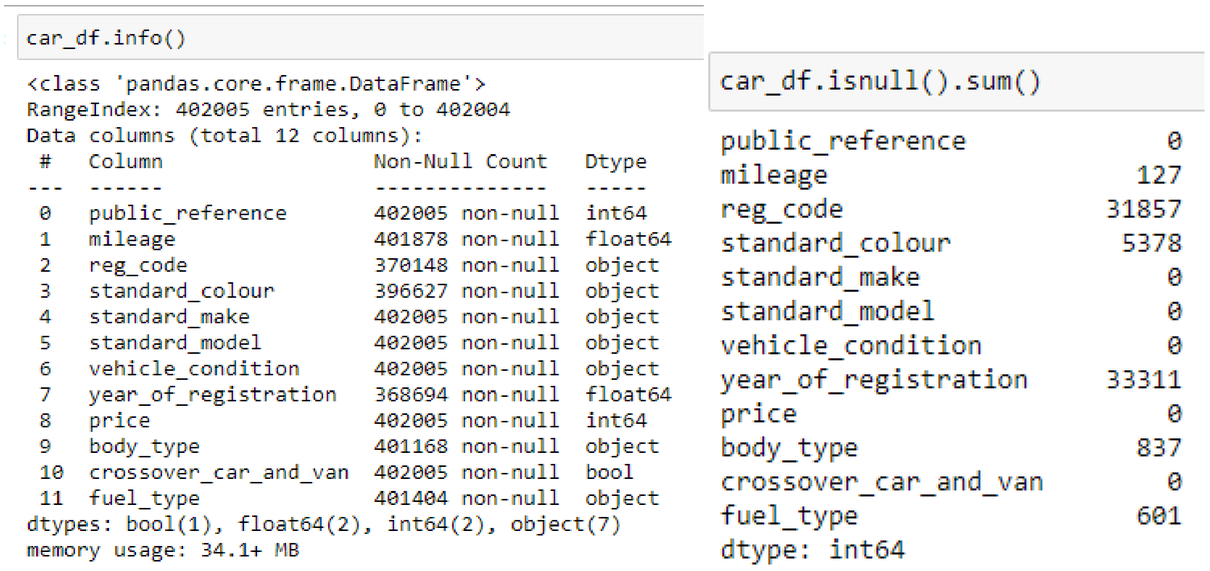


Figure 5: Heatmap showing the correlation matrix

1.3. Data Processing for Data Exploration and Visualisation

The dataset contains 4 datatypes which are integers (public reference, price), float (mileage, years of registration), object (reg code, standard colour, standard make, standard model, vehicle condition, body type and fuel type) and boolean (crossover and van). The dataset contains missing values which may cause breakage in the model so there is a need to deal with them accordingly. Reg code and year of registration has high number of null values (figure 6), hence there is a need for processing.



```
car_df.info()
```

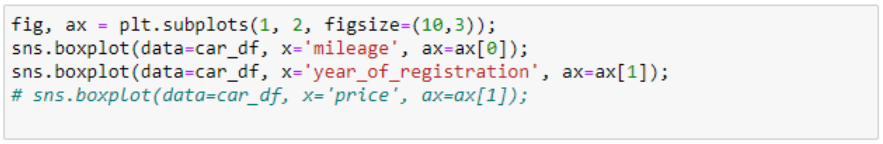
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 402005 entries, 0 to 402004
Data columns (total 12 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   public_reference        402005 non-null  int64
 1   mileage                 401878 non-null  float64
 2   reg_code                370148 non-null  object
 3   standard_colour         396627 non-null  object
 4   standard_make           402005 non-null  object
 5   standard_model          402005 non-null  object
 6   vehicle_condition       402005 non-null  object
 7   year_of_registration    368694 non-null  float64
 8   price                   402005 non-null  int64
 9   body_type               401168 non-null  object
 10  crossover_car_and_van   402005 non-null  bool
 11  fuel_type               401404 non-null  object
dtypes: bool(1), float64(2), int64(2), object(7)
memory usage: 34.1+ MB
```

```
car_df.isnull().sum()
```

```
public_reference              0
mileage                     127
reg_code                  31857
standard_colour            5378
standard_make                 0
standard_model                0
vehicle_condition             0
year_of_registration      33311
price                         0
body_type                   837
crossover_car_and_van         0
fuel_type                   601
dtype: int64
```

Figure 6: Data information and exploration

Furthermore, after considering the data distribution, it was discovered that the data set contain some outliers which give less meaningful information, these outliers need to be removed.



```
fig, ax = plt.subplots(1, 2, figsize=(10,3));
sns.boxplot(data=car_df, x='mileage', ax=ax[0]);
sns.boxplot(data=car_df, x='year_of_registration', ax=ax[1]);
# sns.boxplot(data=car_df, x='price', ax=ax[1]);
```

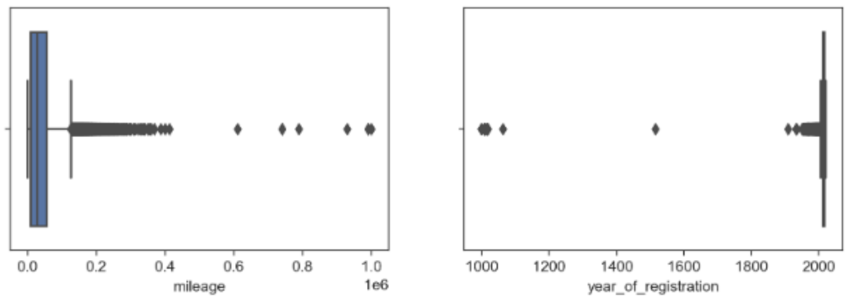


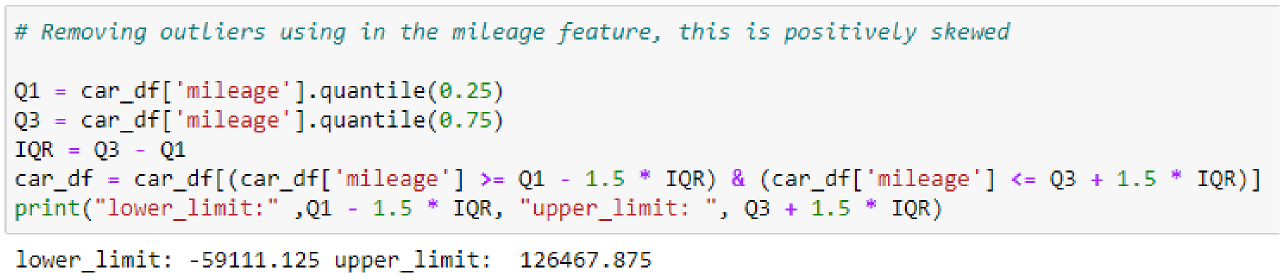
Figure 7: Boxplot of mileage and year of registration

In figure 7, boxplot explains an important distribution, indicating that mileage above 120000 are outliers, and also the year of registration, the extreme outliers need to be taken care of.

2.0. DATA PROCESSING FOR MACHINE LEARNING

2.1. Dealing with Missing Values, Outliers and Noise:

The boxplot analysis in figure 7 reveals the presence of outliers in the mileage data. To address this issue, any mileage values outside the range defined by the lower limit and upper limit will be treated as outliers and subsequently removed. The lower limit is computed as (Q1 – 1.5*IQR), and the upper limit is determined as (Q3 + 1.5*IQR). This same procedure is applied to handle outliers in the year of registration column.

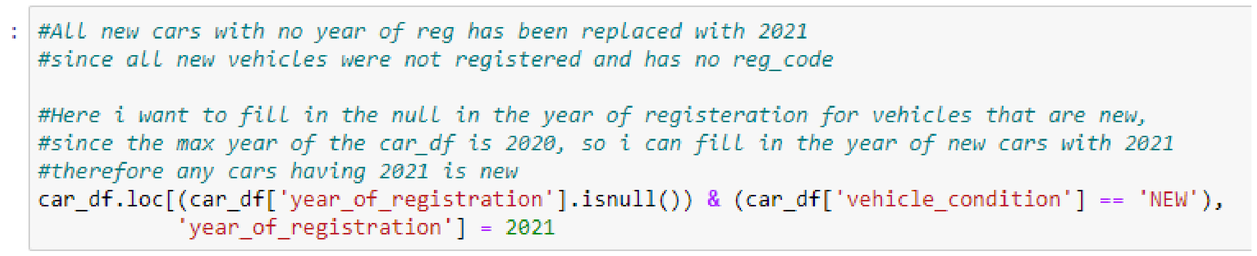


```
# Removing outliers using in the mileage feature, this is positively skewed

Q1 = car_df['mileage'].quantile(0.25)
Q3 = car_df['mileage'].quantile(0.75)
IQR = Q3 - Q1
car_df = car_df[(car_df['mileage'] >= Q1 - 1.5 * IQR) & (car_df['mileage'] <= Q3 + 1.5 * IQR)]
print("lower_limit:" ,Q1 - 1.5 * IQR, "upper_limit: ", Q3 + 1.5 * IQR)
```

```
lower_limit: -59111.125 upper_limit:  126467.875
```

Figure 8: Removal of outliers in mileage

During data exploration, it was confirmed that none of the new vehicles had been registered. Instead of disregarding all the new vehicles, the year of registration is replaced with 2021, assuming they will be registered in the upcoming year

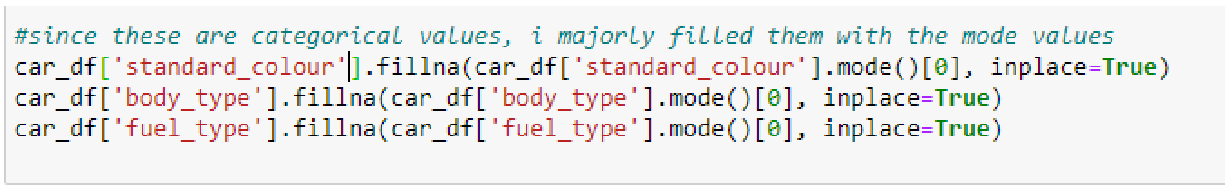


```
: #All new cars with no year of reg has been replaced with 2021
  #since all new vehicles were not registered and has no reg_code

  #Here i want to fill in the null in the year of registeration for vehicles that are new,
  #since the max year of the car_df is 2020, so i can fill in the year of new cars with 2021
  #therefore any cars having 2021 is new
  car_df.loc[(car_df['year_of_registration'].isnull()) & (car_df['vehicle_condition'] == 'NEW'),
            'year_of_registration'] = 2021
```

Figure 9: Filling missing values in year of registration

The missing values in the year of registration were replaced using the KNN imputer from scikit learn, k-neighbour of 2.

For categorical feature such as standard colour, body type and fuel type, the missing values are replaced with the most frequent value.

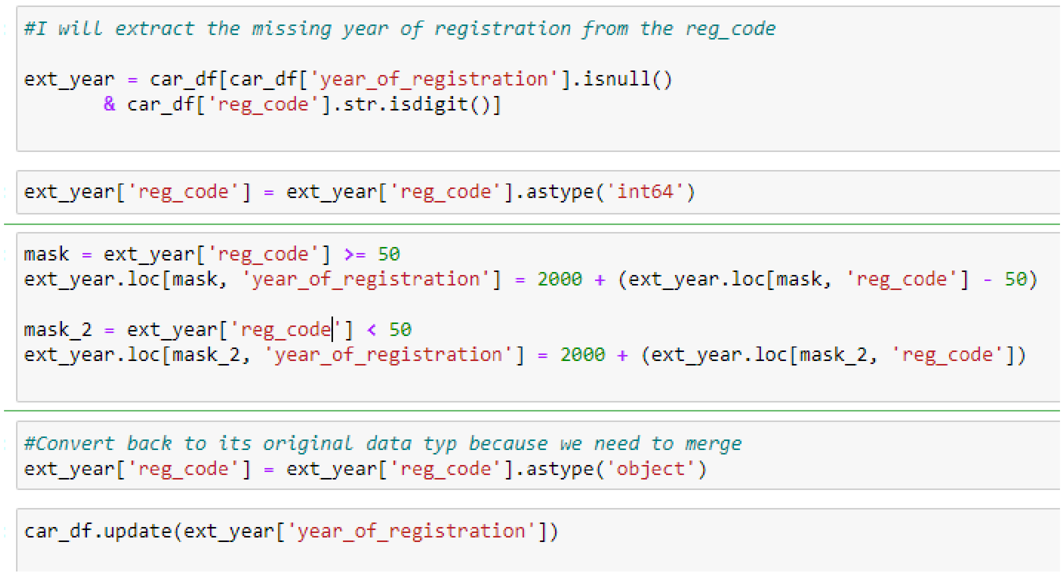


```
#since these are categorical values, i majorly filled them with the mode values
car_df['standard_colour'].fillna(car_df['standard_colour'].mode()[0], inplace=True)
car_df['body_type'].fillna(car_df['body_type'].mode()[0], inplace=True)
car_df['fuel_type'].fillna(car_df['fuel_type'].mode()[0], inplace=True)
```

Figure 10: Filling missing values in categorical features

2.2. Feature Engineering, Data Transformations, Feature Selection:

Considering the high number of missing values in registration code and year of registration, it has been identified that the reg code is derived from the year of registration, and vice versa. Therefore, there is a need to extract the missing year of registration from the registration code



```
#I will extract the missing year of registration from the reg_code

ext_year = car_df[car_df['year_of_registration'].isnull()
       & car_df['reg_code'].str.isdigit()]
```

```
ext_year['reg_code'] = ext_year['reg_code'].astype('int64')
```

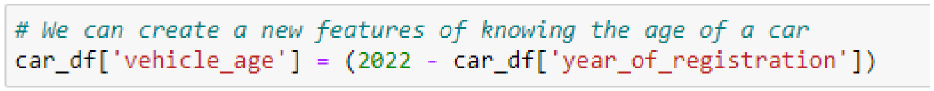
```
mask = ext_year['reg_code'] >= 50
ext_year.loc[mask, 'year_of_registration'] = 2000 + (ext_year.loc[mask, 'reg_code'] - 50)

mask_2 = ext_year['reg_code'] < 50
ext_year.loc[mask_2, 'year_of_registration'] = 2000 + (ext_year.loc[mask_2, 'reg_code'])
```

```
#Convert back to its original data typ because we need to merge
ext_year['reg_code'] = ext_year['reg_code'].astype('object')
```

```
car_df.update(ext_year['year_of_registration'])
```

Figure 11: Extraction car of missing year of registration from the reg code

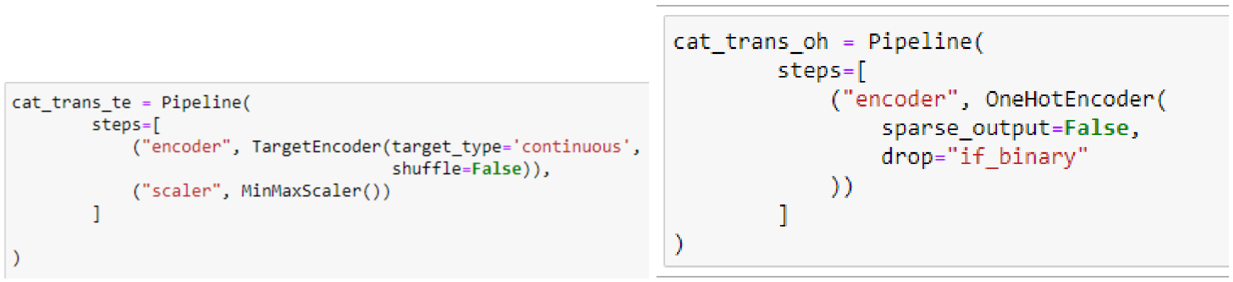
A “vehicle age” feature was created to provide a quick understanding of the age of a vehicle since its registration



```
# We can create a new features of knowing the age of a car
car_df['vehicle_age'] = (2022 - car_df['year_of_registration'])
```

Figure 12: creating a new column

Data transformation has been performed on the data according to the feature type. A pipeline was created to streamline the workflow. One -hot encoding was applied to categorical features with a cardinality of two, while features with cardinalities above 2 were transformed using target encoding. MinMaxScaler was employed to normalized the values of numerical features, ensuring a smoother model process.



```
cat_trans_te = Pipeline(
        steps=[
            ("encoder", TargetEncoder(target_type='continuous',
                                      shuffle=False)),
            ("scaler", MinMaxScaler())
        ]
)
```

```
cat_trans_oh = Pipeline(
        steps=[
            ("encoder", OneHotEncoder(
                sparse_output=False,
                drop="if_binary"
            ))
        ]
)
```

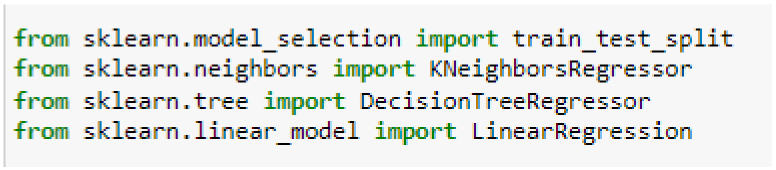
Figure 13: Data transformation using one hot encoder and target encoder through pipeline

For machine learning preparation, the dataset after cleaning and processing contains large numbers of row, hence there is a need to sample and make selection. The selection was based on standard make and standard model through row stratification. A 20% of the row were selected but the modeling was carried out on the 20 most frequent vehicle manufacturer

3.0. MODEL BUILDING

3.1. Algorithm Selection, Model Instantiation and Configuration

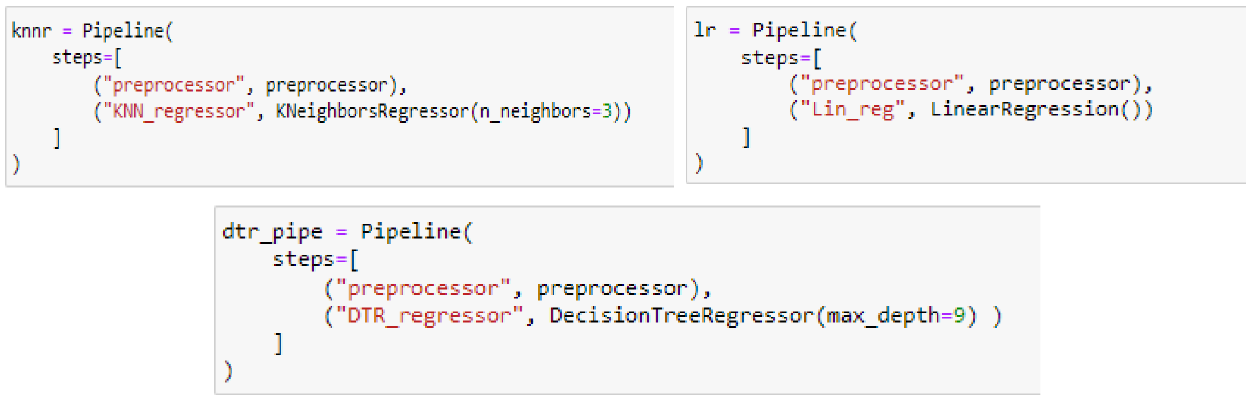
The target feature is continuous, this shows that it is a regression problem, so therefore regression algorithm such as K nearest neighbor regression, decision tree regression and linear regression will be the selected algorithm to be used



```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
```

Figure 14: Algorithm Selection

The selected algorithms were inserted into a pipeline which also contain all other transformation and configuration, such as MinMaxScaler, target encoders, and one hot encoder, all this transformation are loaded into the column transformer in a variable named preprocessor. In the figure below, the pipeline helps to streamline and automate the preprocessing steps along with the model training. The KNN regressor is instantiated with number of neighbours as 3, while the decision tree regressor with a max depth of 9. The linear regression is in its simple state



```
knnr = Pipeline(
    steps=[
        ("preprocessor", preprocessor),
        ("KNN_regressor", KNeighborsRegressor(n_neighbors=3))
    ]
)
```

```
lr = Pipeline(
    steps=[
        ("preprocessor", preprocessor),
        ("Lin_reg", LinearRegression())
    ]
)
```

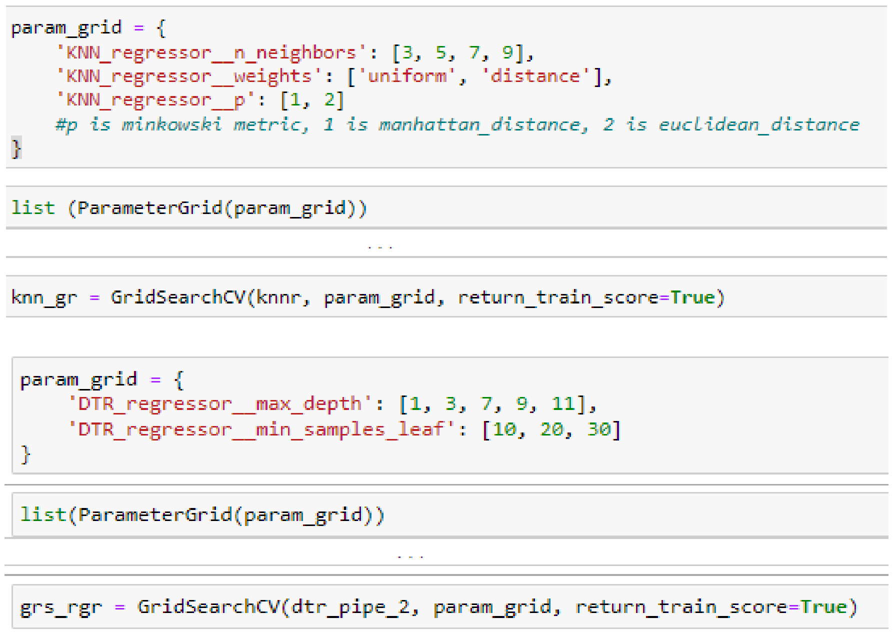
```
dtr_pipe = Pipeline(
    steps=[
        ("preprocessor", preprocessor),
        ("DTR_regressor", DecisionTreeRegressor(max_depth=9) )
    ]
)
```

Figure 15: Model Instantiation

The KNN regressor with a number of neighbours of 3 predicts the target variable (price) by averaging the target values of its k nearest neighbours. The decision tree regressor is also instantiated with a max depth of 9, a max depth of 9 is considered so that it can capture complex pattern and at the same time not being overfitting.

3.2. Grid Search and Model Ranking and Selection

Grid serach helps in tuning the hyperparameters inorder to get the best combination of hyperparameters from a predefined param grid of possible values. The grid search technique is performed for the K Nearest Neighbours Regression and Decision Tree Regression.



```
param_grid = {
    'KNN_regressor__n_neighbors': [3, 5, 7, 9],
    'KNN_regressor__weights': ['uniform', 'distance'],
    'KNN_regressor__p': [1, 2]
    #p is minkowski metric, 1 is manhattan_distance, 2 is euclidean_distance
}
```

```
list (ParameterGrid(param_grid))
```

```
...
```

```
knn_gr = GridSearchCV(knnr, param_grid, return_train_score=True)
```

```
param_grid = {
    'DTR_regressor__max_depth': [1, 3, 7, 9, 11],
    'DTR_regressor__min_samples_leaf': [10, 20, 30]
}
```

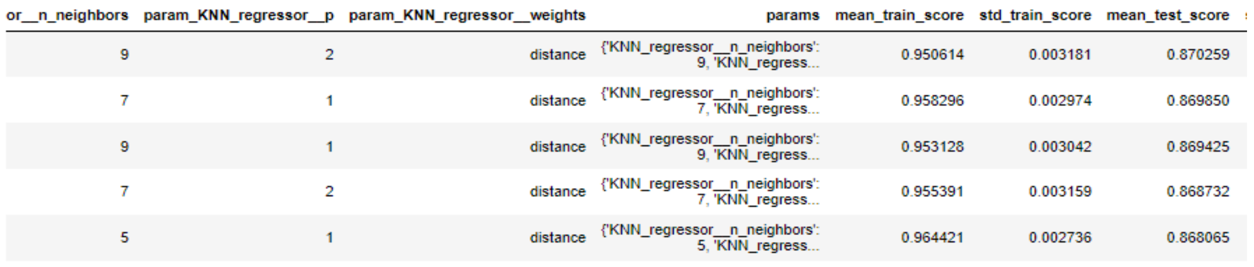
```
list(ParameterGrid(param_grid))
```

```
...
```

```
grs_rgr = GridSearchCV(dtr_pipe_2, param_grid, return_train_score=True)
```

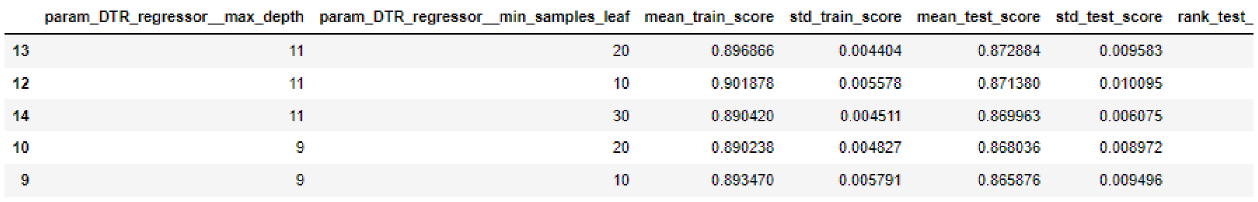
Figure 16: Grid Search (KNN and Decision Tree)

Considering the result from the grid search of both KNN and decision tree regressor, the KNN regressor performs well on the training data giving a difference of about 0.8 when compared with the mean test score. The decision tree has a low difference of about 0.3 between the mean train score and mean test score. So, decision tree regressor is more preferred



or__n_neighbors	param_KNN_regressor__p	param_KNN_regressor__weights	params	mean_train_score	std_train_score	mean_test_score
9	2	distance	{'KNN_regressor__n_neighbors': 9, 'KNN_regress...	0.950614	0.003181	0.870259
7	1	distance	{'KNN_regressor__n_neighbors': 7, 'KNN_regress...	0.958296	0.002974	0.869850
9	1	distance	{'KNN_regressor__n_neighbors': 9, 'KNN_regress...	0.953128	0.003042	0.869425
7	2	distance	{'KNN_regressor__n_neighbors': 7, 'KNN_regress...	0.955391	0.003159	0.868732
5	1	distance	{'KNN_regressor__n_neighbors': 5, 'KNN_regress...	0.964421	0.002736	0.868065

Figure 17: KNN Regressor grid search result



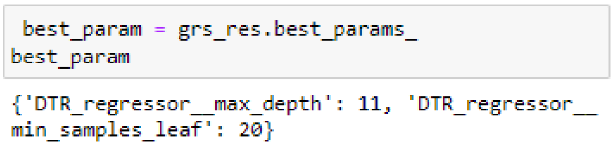
	param_DTR_regressor__max_depth	param_DTR_regressor__min_samples_leaf	mean_train_score	std_train_score	mean_test_score	std_test_score	rank_test_
13	11	20	0.896866	0.004404	0.872884	0.009583	
12	11	10	0.901878	0.005578	0.871380	0.010095	
14	11	30	0.890420	0.004511	0.869963	0.006075	
10	9	20	0.890238	0.004827	0.868036	0.008972	
9	9	10	0.893470	0.005791	0.865876	0.009496	

Figure 18: Decision Tree Regressor grid search result

4.0. MODEL EVALUATION AND ANALYSIS

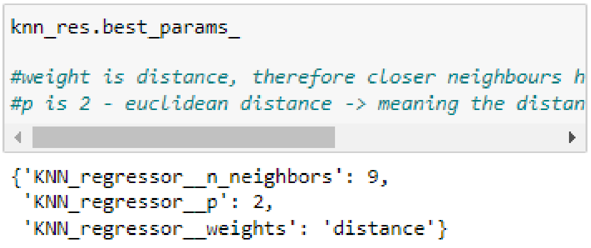
4.1. Coarse-Grained Evaluation/Analysis

The score values based on different model describes the performance of the model on the training and test dataset. To determine the best score value, a grid search analysis was done by tuning the hyperparameters, the best parameter was integrated to determine the performance score value.



```
best_param = grs_res.best_params_
best_param
```

```
{'DTR_regressor__max_depth': 11, 'DTR_regressor__
min_samples_leaf': 20}
```

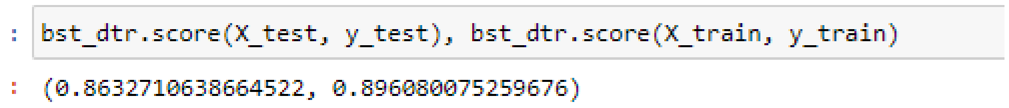


```
knn_res.best_params_

#weight is distance, therefore closer neighbours h
#p is 2 - euclidean distance -> meaning the distan
```

```
{'KNN_regressor__n_neighbors': 9,
 'KNN_regressor__p': 2,
 'KNN_regressor__weights': 'distance'}
```

Figure 19: The best parameter for KNN and Decision Tree Regression

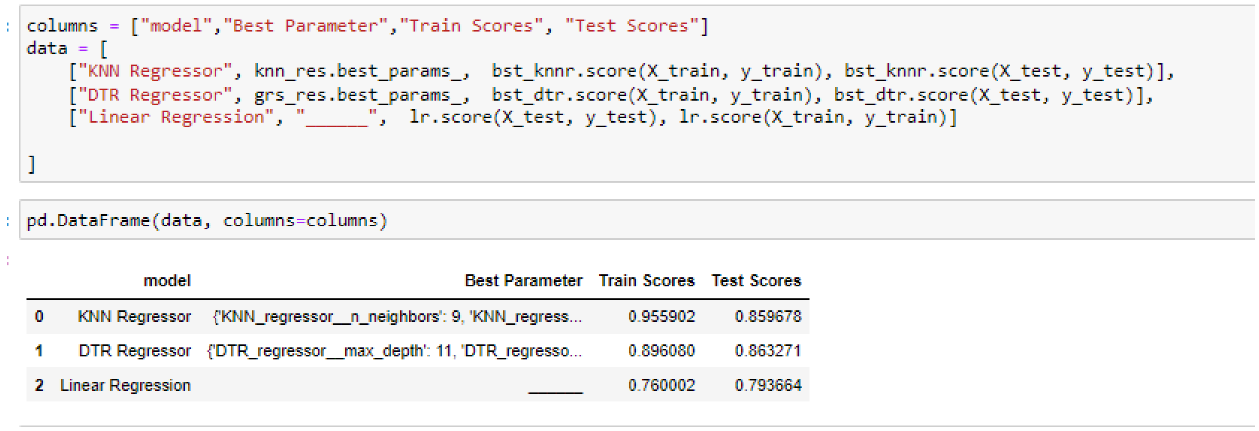
Applying this best parameters, the score values of the three models were compared. The KNN regressor performs well on the train dataset, the difference when compared with the test data was about 0.1 (10%). But for decision tree and linear regression, it is about 0.03 (3%) which is more lower. Between the two models with a lower differences, the decision tree regressor seems to be the best because its performance score value is high, and also with a low score difference between the train and test data set.



```
bst_dtr.score(X_test, y_test), bst_dtr.score(X_train, y_train)
```

```
(0.8632710638664522, 0.896080075259676)
```

Figure 20: Performance score using the best paramenter from decision tree hyparameter tuning



```
columns = ["model","Best Parameter","Train Scores", "Test Scores"]
data = [
    ["KNN Regressor", knn_res.best_params_,  bst_knnr.score(X_train, y_train), bst_knnr.score(X_test, y_test)],
    ["DTR Regressor", grs_res.best_params_,  bst_dtr.score(X_train, y_train), bst_dtr.score(X_test, y_test)],
    ["Linear Regression", "______",  lr.score(X_test, y_test), lr.score(X_train, y_train)]

]
```

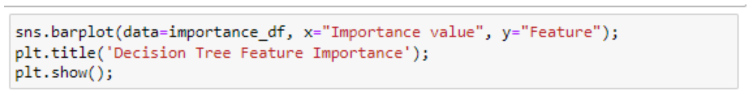
```
pd.DataFrame(data, columns=columns)
```

	model	Best Parameter	Train Scores	Test Scores
0	KNN Regressor	{'KNN_regressor__n_neighbors': 9, 'KNN_regress...	0.955902	0.859678
1	DTR Regressor	{'DTR_regressor__max_depth': 11, 'DTR_regresso...	0.896080	0.863271
2	Linear Regression	______	0.760002	0.793664

Figure 21: Performance score comparison

4.2. Feature Importance

In model evaluation, it is important that we understand the significance of each feature. The decision tree-based model explains the importance of each feature which is also essential for feature selection and gaining insight into the feature that contributes most to the model's prediction. In the figure below, the body type is the most important feature with about 63% importance value, followed by standard model and mileage which have an importance value of 14% and 13% respectively



```
sns.barplot(data=importance_df, x="Importance value", y="Feature");
plt.title('Decision Tree Feature Importance');
plt.show();
```

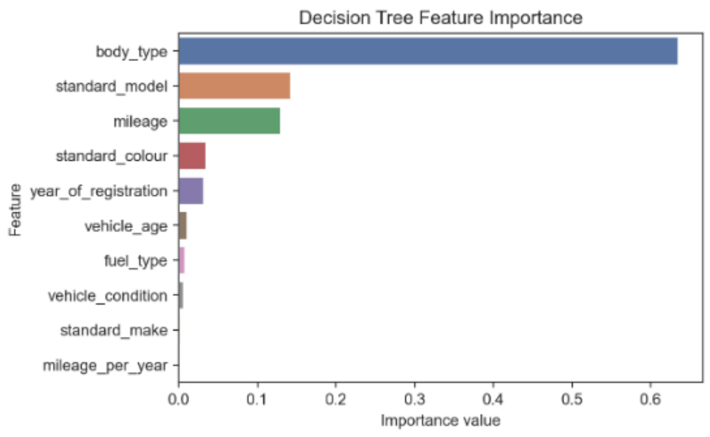
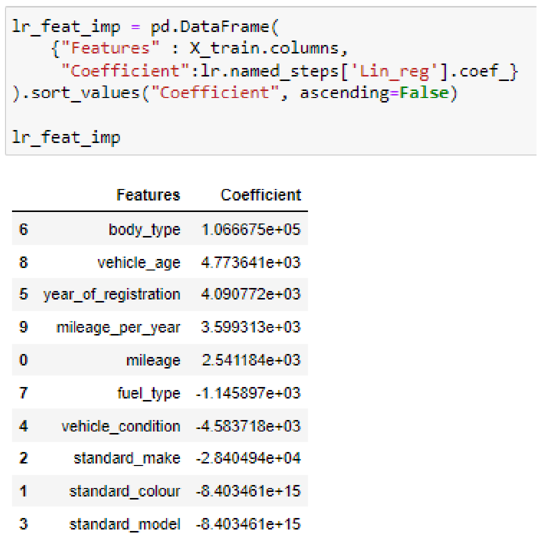


Figure 22: Feature importance in decision tree-based model

In Linear regression feature importance, the body type feature also is the most important followed by vehicle age, year of registration, mileage per year and mileage. vehicle age and mileage per year are derived from year of registration and mileage respectively.



```
lr_feat_imp = pd.DataFrame(
    {"Features" : X_train.columns,
     "Coefficient":lr.named_steps['Lin_reg'].coef_}
).sort_values("Coefficient", ascending=False)

lr_feat_imp
```

	Features	Coefficient
6	body_type	1.066675e+05
8	vehicle_age	4.773641e+03
5	year_of_registration	4.090772e+03
9	mileage_per_year	3.599313e+03
0	mileage	2.541184e+03
7	fuel_type	-1.145897e+03
4	vehicle_condition	-4.583718e+03
2	standard_make	-2.840494e+04
1	standard_colour	-8.403461e+15
3	standard_model	-8.403461e+15

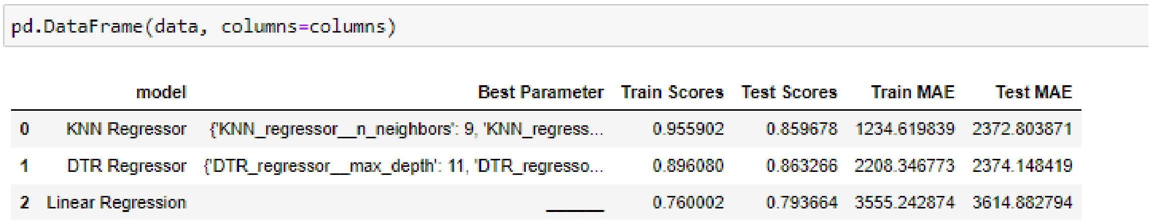
Figure 23: Feature importance in linear regression model

4.3. Fine-Grained Evaluation/Analysis (e.g. with instance-level errors)

The metric employed for this evaluation and analysis is Mean Absolute Error (MAE). This is the average of the difference between the predicted value and the actual value. This helps to understand how well the model performs. A larger MAE value indicate that the predicted value is much higher than the actual value, while a smaller MAE indicates that the predicted value is closer to the actual value, the formular to calculate the Mean Absolute Error is

$(1/n) * \sum|yi - xi|$

- *Σ: Greek symbol for summation*
- *yi: Actual value for the ith observation*
- *xi: Calculated value for the ith observation*
- *n: Total number of observations*



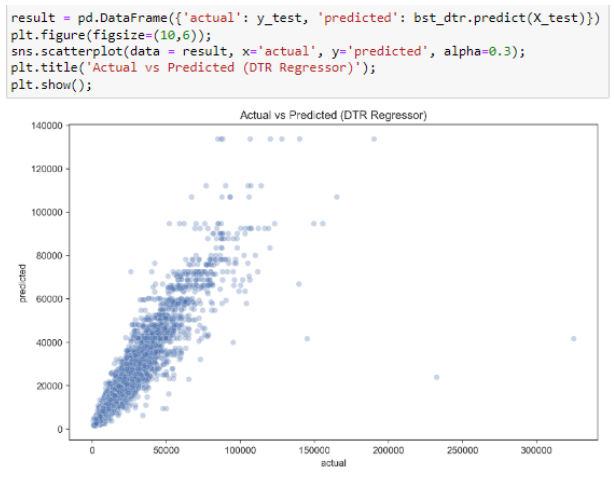
```
pd.DataFrame(data, columns=columns)
```

	model	Best Parameter	Train Scores	Test Scores	Train MAE	Test MAE
0	KNN Regressor	{'KNN_regressor__n_neighbors': 9, 'KNN_regress...	0.955902	0.859678	1234.619839	2372.803871
1	DTR Regressor	{'DTR_regressor__max_depth': 11, 'DTR_regresso...	0.896080	0.863266	2208.346773	2374.148419
2	Linear Regression	______	0.760002	0.793664	3555.242874	3614.882794

Figure 24: Summary of the MAE value for all models

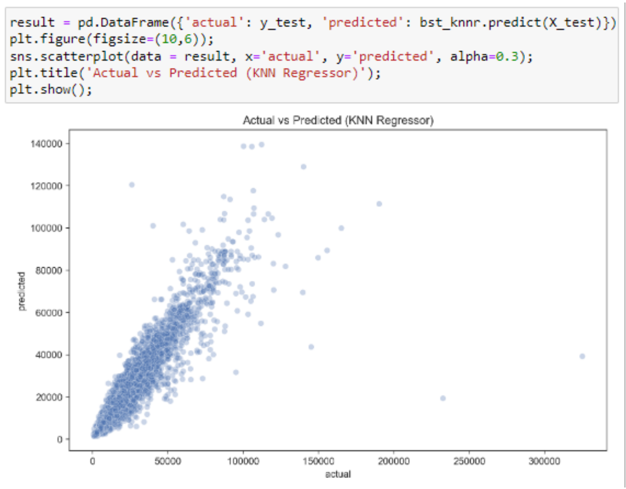
KNN regressor has a very low MAE on train data, but it is high on test data, having a difference of over 1000. When compare with decision tree regressor, the error difference is slightly above 100, which shows that the performance on the unseen data is almost the same as the train data. The mean absolute error of linear regression is too high both on the train and test data when compared with other regression model

The plot of actual and predicted value further explains how good the model is, the linear model plot doesn't show a linear correlation as others does.



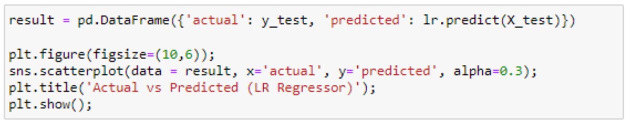
```
result = pd.DataFrame({'actual': y_test, 'predicted': bst_dtr.predict(X_test)})
plt.figure(figsize=(10,6));
sns.scatterplot(data = result, x='actual', y='predicted', alpha=0.3);
plt.title('Actual vs Predicted (DTR Regressor)');
plt.show();
```

Figure 25: Actual vs Predited plot for Decision tree regressor



```
result = pd.DataFrame({'actual': y_test, 'predicted': bst_knnr.predict(X_test)})
plt.figure(figsize=(10,6));
sns.scatterplot(data = result, x='actual', y='predicted', alpha=0.3);
plt.title('Actual vs Predicted (KNN Regressor)');
plt.show();
```

Figure 26: Actual vs Predited plot for KNN regressor



```
result = pd.DataFrame({'actual': y_test, 'predicted': lr.predict(X_test)})

plt.figure(figsize=(10,6));
sns.scatterplot(data = result, x='actual', y='predicted', alpha=0.3);
plt.title('Actual vs Predicted (LR Regressor)');
plt.show();
```

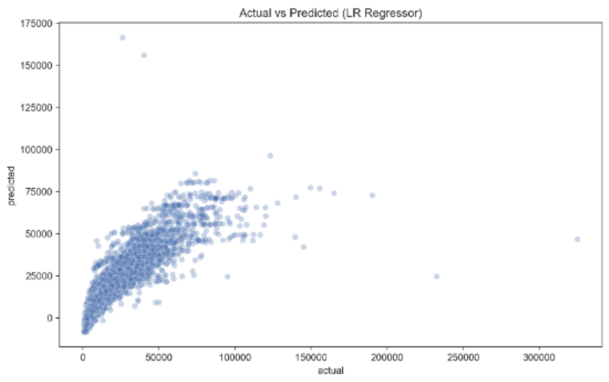


Figure 27: Actual vs Predited plot for linear regressor

In conclusion, the most preferred model is decision tree regression, because it produces a low difference between test and train data (in this case the score and mean absolute error), it also seems to perform well on both the test and train data than other type of model employed