1.Data/Domain Understanding and Exploration

1.1. Meaning and Type of Features and Analysis of Distributions

Initially there are 12 columns/features in dataset. Each column has unique meaning. The below are some features meaning and type.

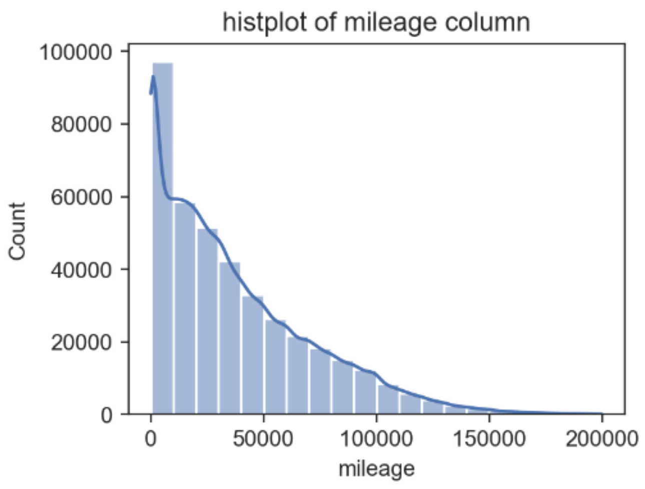
Standard make: This column gives information about the manufacturer who produced, designed, and branded the vehicle. Some of the brands like BMW, Audi, Benz, Ferrari etc. It is a Categorical column. This is a predictor.

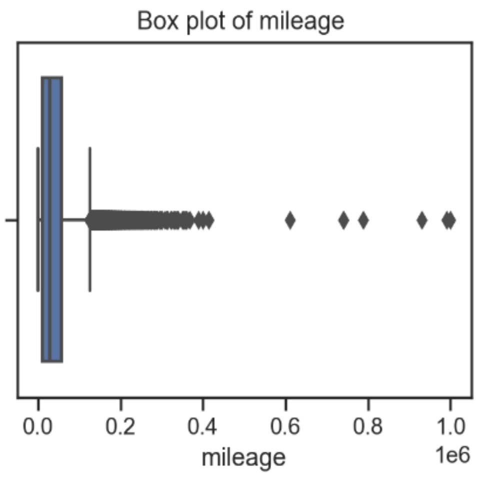
Price: This feature provides information about the costs involved in buying a car. This is a Numerical type of feature. This is the target variable.

Analysing the distribution of Mileage column:

```
#histogram for mileage
sns.histplot(data = df.sample(n=40000), x= 'mileage', bins = 30,kde = True)
```

```
#boxplot of mileage column
sns.boxplot(data = df, x = 'mileage')
```



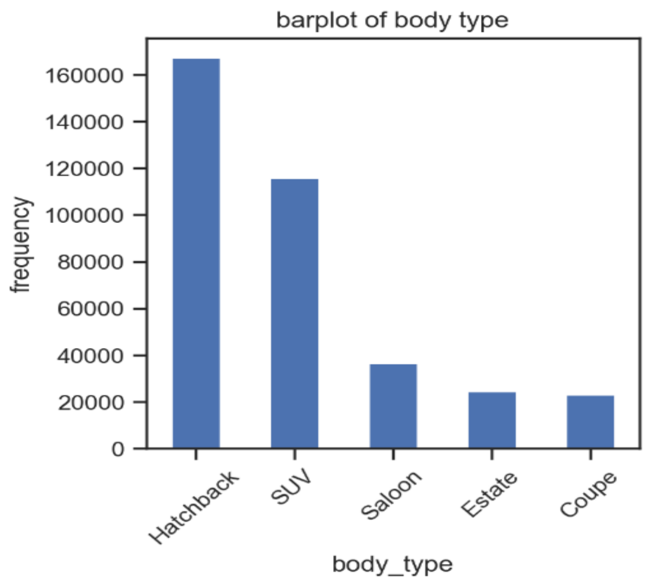


The above boxplot and hist plot tell that there are many outliers in the data set and the data is right skewed which is indicating that there are cars with exceptionally high mileage. The hist plot is plotted on the subset of the whole data i.e. mileage less than 2000.There are many zero values in the mileage column.

Analysing the distribution of Body type column:

The below bar plot is showing distribution of the top 5 body types of cars while there are 16 different body types because to get good understanding. From the plot, there are majority of cars with Hatchback body type followed by SUV. Saloons, Estates and coupes have significantly lower frequencies in descending order.

```
#barplot of body type
df['body_type'].value_counts().head().plot.bar()
```



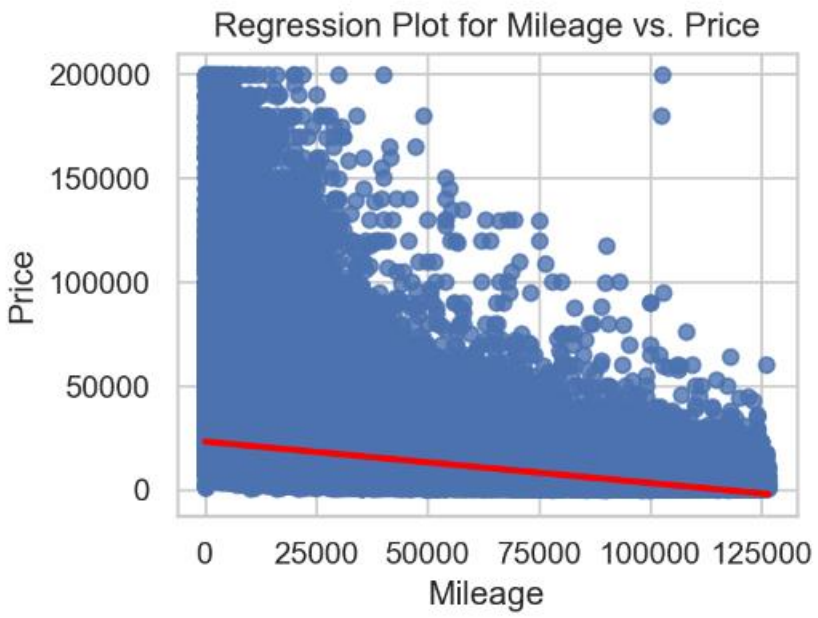
1.2. Analysis of Predictive Power of Features

Mileage vs price:

```
#correlation between 'mileage' and 'price.'
correlation_mileage = df['mileage'].corr(df['price'])
#-0.393325708552187966
```

There is a negative correlation i.e. -0393 between mileage and price columns.

```
# Scatter plot with regression line for 'mileage' vs. 'price'
sns.regplot(x='mileage', y='price', data=df,scatter_kws={'s': 30}, line_kws={'color': 'red'})
```

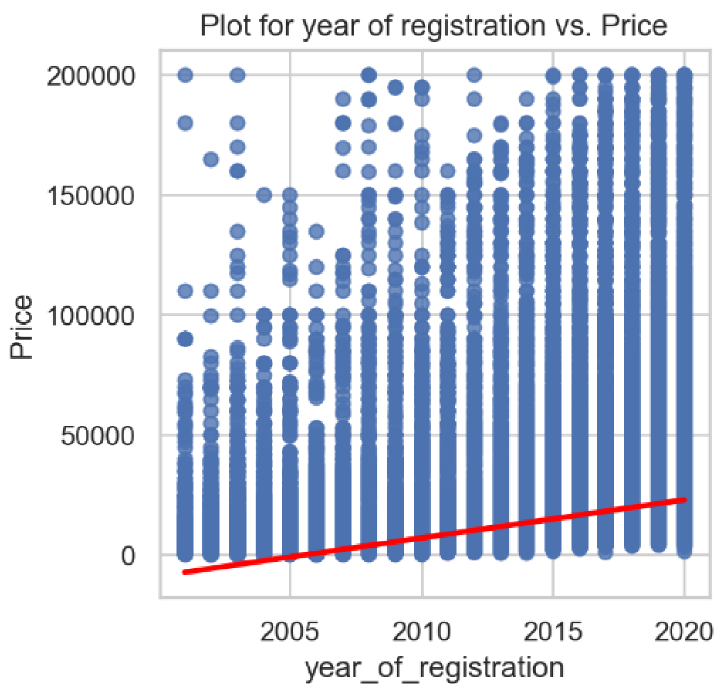


From the above graph and correlation code snippet, it is clear that as the mileage increase price decreases which is a slight negative correlation. The red trend line the graph tells that the price of cars decreased when mileage increased. In reality it is true that the car with more miles driven has lower price and vice versa. So therefore, mileage column is useful in predicting price of a used car.

Year of registration vs Price:

```
#correlation between ' year_of_registration' and 'price'
correlation_year_of_registration = df['year_of_registration'].corr(df['price'])
#0.3215146242839339
```

```
# Scatter plot with regression line for 'mileage' vs. 'price'
filtered_df = df[df['year_of_registration'] > 2000]
sns.regplot(x='year_of_registration', y='price', data=filtered_df,scatter_kws={'s': 30},  line_kws={'color': 'red'})
```



The correlation between year of registration and price column is 0.3215 which is a positive correlation. The above scatter plot is showing the relationship between the two features. The graph tells that the prices of cars are generally increased over the years and the red trend line indicates that the price of cars has increased over time. So, year of registration can be used for predicting the price of a used car.

1.3. Data Processing for Data Exploration and Visualisation

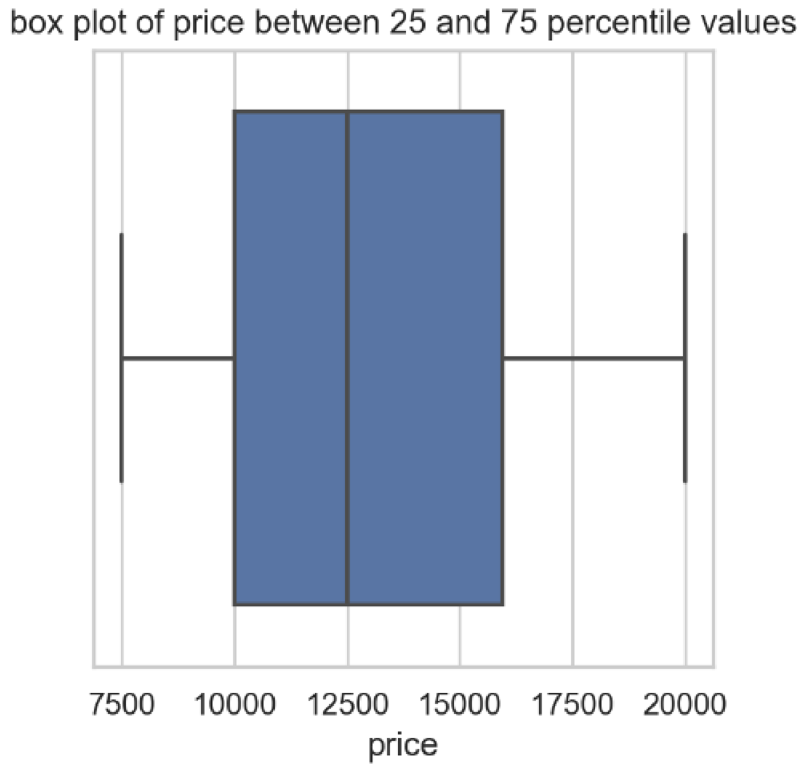
The goal of this section is to ensure that data is in suitable format for exploration and to uncover patterns, trends, and relationships through various visualization techniques. Let's have look at some of the visualizations.

Price feature:

```
df_new = df[(df['price'] >= q1) & (df['price'] <=  q3)]
```

The below box plot is plotted on the subset (i.e. between 25th and 75th percentiles) of the data to get good understanding and insight of the price column. The minimum price is 7500 and the maximum is 20000. The distribution is slightly right skewed which means the majority of the prices are on left side of the plot. The median is 12500.

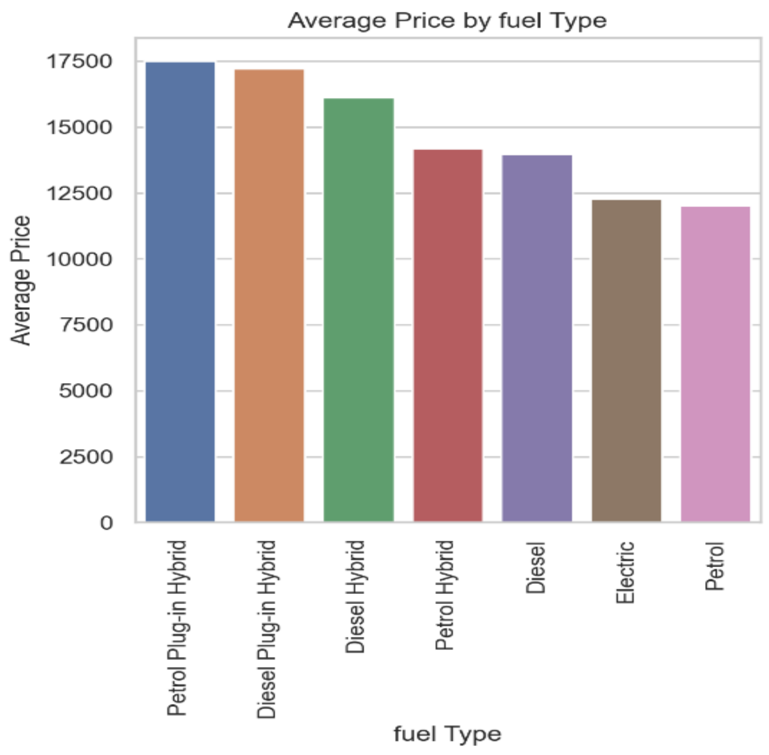
```
sns.boxplot(data =df_new , x = 'price');
```
2.



Price vs fuel type:

```
# Group by 'fuel_type' and calculate average price
avg_price_by_fuel_type = df_new.groupby('fuel_type')['price'].mean().sort_values(ascending=False)
```

```
sns.barplot(x=avg_price_by_fuel_type.index, y=avg_price_by_fuel_type.values)
```



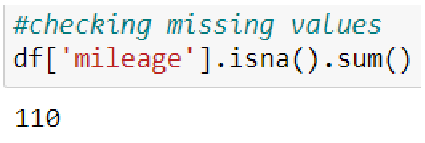
The above bar plot the average price of Petrol Plug in Hybrid and Diesel plug in hybrid are bit similar and higher followed by diesel hybrid fuel type of cars. Cars with petrol type are relatively cheaper followed by Electric compared to all other fuel types. Petrol hybrid and diesel are prices looks both have similar average price.

2. Data Processing for Machine Learning

2.1. Dealing with Missing Values, Outliers, and Noise

Cleaning mileage column:

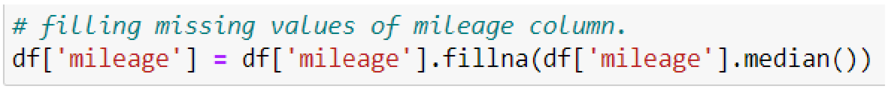
Let's check whether there are any missing values in the mileage column.



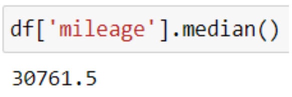
```
#checking missing values
df['mileage'].isna().sum()
```

```
110
```

There are a total of 110 missing values in the mileage column. Let's fix it by using the median value. We can also fill it with the mean value, but the mean may be affected by outliers. However, even if there are outliers, the median value is not affected by them.



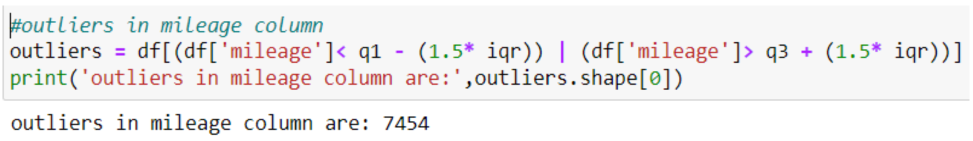
```
# filling missing values of mileage column.
df['mileage'] = df['mileage'].fillna(df['mileage'].median())
```



```
df['mileage'].median()
```

```
30761.5
```

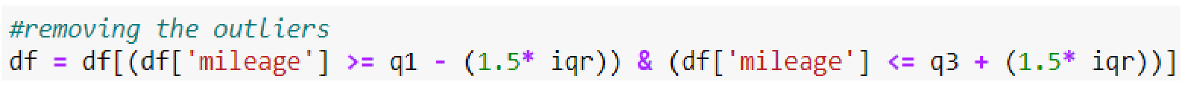
All the missing values are filled with the median value 30761.5 using the fillna() function. Now let's deal with outliers.



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

```
outliers in mileage column are: 7454
```

From the above code, there are total of 7454 outliers are present. Remove 7454 outliers with IQR method which we followed for price column.

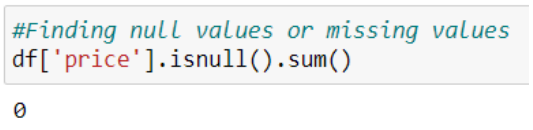


```
#removing the outliers
df = df[(df['mileage'] >= q1 - (1.5* iqr)) & (df['mileage'] <= q3 + (1.5* iqr))]
```

All the outliers are removed from mileage column and stored in the original data frame again.

Cleaning price column:

Let's check whether there are any missing values in the price column.



```
#Finding null values or missing values
df['price'].isnull().sum()
```

```
0
```

There are no missing values in the price column and no need to fill anything in the price column.

Dealing outliers in price column:

The summary statistics of price column tells that the max value is 9999999 and the 75th percentile value is 20000. Statistically the distribution is highly right skewed and removing all the values above than 20000 is also not a good method. So let's assume to fix a cap of 100000 because in reality people can afford upto 100000 pounds. Even though there are some outliers after caping, but they are acceptable.

```
df['price'].describe()
```

```
count    4.020050e+05
mean     1.734197e+04
std      4.643746e+04
min      1.200000e+02
25%      7.495000e+03
50%      1.260000e+04
75%      2.000000e+04
max      9.999999e+06
```

```
# removing some outliers in price column manually which are greater than 100000
df = df[df['price'] < 100000]
#Reset the index
df = df.reset_index(drop= True)
```

Now, the outliers greater than 100000 are removed from the price column. The distribution of the price column is still right skewed and there are some outliers, but those outliers can be considered for now. Only 2832 columns are removed which are most far away from other values in the price column.

2.2. Feature Engineering, Data Transformations, Feature Selection

Feature engineering:

Creating a new feature called age_of_car based on the existing year_of_registration feature. This new feature indicates the age of the car, which is used to understand how old the car is. This feature is used in predicting the price.

```
#creating age_of_car column
current_year = datetime.now().year
df['age_of_car'] = current_year - df['year_of_registration']
```

Created age of car by subtracting the year of registration from the current year (i.e. 2024). This feature tells how old the car is.

Data transformation:

Transforming categorical variables into numerical variables by Target encoding which is better for high-cardinality features. Data transformation is important for machine learning as it only takes numerical values to fit or train the model.

```
cols = ['standard_colour', 'standard_make', 'body_type', 'fuel_type', 'price_category']
target = 'price'
te = TargetEncoder()
for col in cols:
    transformed_col = te.fit_transform(df[col].astype(str), df[target])
    df[col + '_encoded'] = transformed_col  # Creating new columns with encoded values
```

The above code is of Target encoding. The features ‘standard_colour’, ‘standard_make’, ‘body_type’, ‘fuel_type', 'price_category’ are encoded with respect to the target variable ‘price. The TargetEncoder() function is used to perform this transformation. Adding the transformed data columns to the data frame (df) again.

After transforming, original categorical columns which are transformed are removed from the data frame.

Feature Selection:

Selecting some important features and then separating predictors and target variables. Features are being selected based on high correlation (positive or negative) between price and other features.

```
df= df[['mileage','age_of_car','is_luxury','standard_make','body_type','fuel_type','price_category','price']]
```

```
#The X/y Spilt
X = df.drop(columns = 'price')
y = df['price']
```

```
#The Train/Test Spilt
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state = 2)
```

Splitting the 80 percent of data into training set and 20 percent into testing test. Training data is for training the model and testing data is is to test how accurate the model is running on the training data.

3. Model Building

3.1. Algorithm Selection, Model Instantiation and Configuration

Algorithm Selection:

For model building in machine learning, let’s Choose

a) Decision tree Regressor algorithm,
b) Knn regressor algorithm,
c) linear Regression algorithm.

Model Instantiation and Configuration:

a) Decision tree Regression

```
#decision tree regressor
dtr = DecisionTreeRegressor(max_depth=5,min_samples_split=6)
dtr.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, min_samples_split=6)
```

The Decision tree Regressor model is instantiated with some random parameters and configured with max depth as 5 and min samples split as 6. Then fitted the model with training data.

b) KNN regressor algorithm:

```
#knn regressor
knnr = KNeighborsRegressor(n_neighbors = 4)
knnr.fit(preprocessor.fit_transform(X_train),y_train)
```

```
▾ KNeighborsRegressor
KNeighborsRegressor(n_neighbors=4)
```

The KNN Regressor Model is initially instantiated with random parameter n_neighbor and configured n_neighbor with 4. Then the model fitted with trained scaled data.

c) Linear Regression:

```
#linear Regression
lr = LinearRegression(fit_intercept=True)
lr.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

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

All the three above models are initially instantiated with random parameters and with random configurations. One can does not rely on such random parameters. Need to try with different parameters. For that we can use gridsearchcv method.

3.2. Grid Search, and Model Ranking and Selection:

In this phase hyper tuning is performed using GridSearchCV. In the below codes, GridSearchCV is performed using pipelines on each model. Each pipeline for each model. So, all the below codes are performed using pipelines.

- Grid search for Decision tree:

```
# Define parameters for GridSearchCV
param_grid = {
    'regressor__max_depth': [None, 5, 10,15,20],
    'regressor__min_samples_split': [2, 5, 10,12]
}
# Perform GridSearchCV with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=KFold(n_splits=5), scoring='neg_mean_absolute_error')

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

The above code is of a grid search which is performed to explore various hyperparameter combinations for the Decision tree Regressor, specifically varying the maximum depth, using K-fold cross-validation with 5 folds. After performing grid search, the best parameters of the model are max depth: 15 and min samples split:12 which has first rank among other combinations of max depth and min samples split.

Grid search for KNN :

```
# Define parameters for GridSearchCV
param_grid = {
    'regressor__n_neighbors': [ 5, 7, 9,11]
}

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

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

The above code is performing grid search to explore different values of the n_neighbors hyperparameter for KNN regressor., using K-fold cross-validation with 5 folds. After performing grid search cv, the best parameter of KNN model is n_neighbor with value 11.

- Grid Search for Linear Regression:

```
# Define parameters for GridSearchCV
param_grid = {
    'regressor__fit_intercept': [True, False]
}
# Perform GridSearchCV with cross-validation
grid_search = GridSearchCV(pipeline_lr, param_grid, cv=KFold(n_splits=5), scoring='neg_mean_absolute_error')

# Fit the model directly since Linear Regression doesn't need hyperparameter tuning
grid_search.fit(X_train, y_train)
```

The above code is performing grid search to explore a search over the specified hyperparameter for fit_intercept, using K-fold cross-validation with 5 folds. After performing grid search cv, the best parameter of the model is fit_intercept as True.

Model Ranking:

Mean absolute Error of Decision tree Regression on Trained Data is,

```
Train MAE: 1862.00622240536
```

Rank :1

Mean absolute Error KNN Regressor Trained Data is,

```
Train MAE: 1949.7506527629628
```

Rank: 2

Mean absolute Error of Linear Regression on Trained Data is,

```
Train MAE: 3369.237193908575
```

Rank: 3

Selection:

On comparing the above Mean absolute Errors of three models, Decision tree Regressor has the lowest mean absolute error followed by KNN regressor. The best model among the three models is Decision tree model because of the lowest error. The lowest the error, the better the model.

4. Model Evaluation and Analysis

4.1. Coarse-Grained Evaluation/Analysis

Decision Tree Regression:

```
y_pred_test = best_model_dt.predict(X_test)
```

```
test_mae = mean_absolute_error(y_test , y_pred_test)
```

```
Test MAE: 2126.9807662713933
```

The above code showing Mean absolute Error (2126.9) which is a measure of the average absolute differences between the actual and predicted values of Decision tree Regression on Test data.

KNN Regression:

```
y_pred_test = best_model_knn.predict(X_test)
```

```
test_mae = mean_absolute_error(y_test , y_pred_test)
```

```
Test MAE: 2149.3263284010654
```

The above code showing Mean absolute Error(2149.3) which is a measure of the average absolute differences between the actual and predicted values of KNN Regression.

Linear Regression:

```
y_pred_test = best_linear_reg.predict(X_test)
```

```
test_mae = mean_absolute_error(y_test, y_pred_test)
```

```
Test MAE: 3372.650611117086
```

The above code showing the Mean absolute Error(3372.6) which is a measure of the average absolute differences between the actual and predicted values of Linear Regression.

Among the three models , Decision Tree Regressor seems to perform the best followed by KNN regressor, considering the trade-off between Mean absolute error on train data and test data.

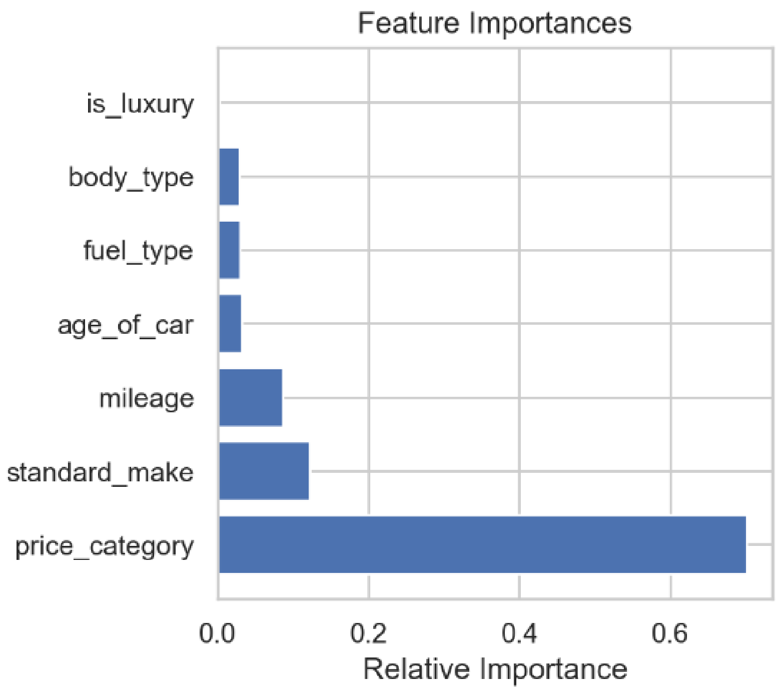
4.2. Feature Importance

Important Feature in Decision Tee Regressor:

In decision tree Regressor, there is attribute called feature_importances_ which is used to tell measure of importance of each feature in making predictions. Below is the code for finding feature importance and plotting feature importances.

```
# 'best_model_dt' is Decision Tree Regressor pipeline
importances_dt = best_model_dt.named_steps['regressor'].feature_importances_
```

```
plt.barh(range(X.shape[1]), importances_dt[indices], align='center')
```



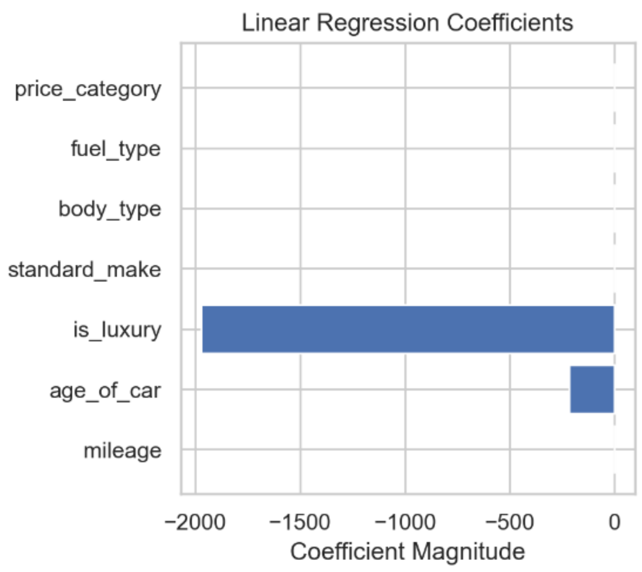
From the above horizontal bar plot, the most important feature in predicting price using decision tree regression model is Price category followed by standard make and mileage. Is luxury feature being not much useful in predicting price in decision tree regressor model.

Important feature in Linear Regression model:

Unlike the Decision tree model, the linear regression model does not have a built-in mechanism to find feature importance. However, the importance of features can be found indirectly by using the coefficients associated with each feature. Below is the code for finding coefficients and plotting the graph.

```
coefficients = best_linear_reg.named_steps['regressor'].coef_
```

```
plt.barh(feature_names, coefficients)
```



The above bar graph shows the relative importance of each variable in the linear regression model. According to the graph, the variables "luxury" and "age of car" have negative coefficients, indicating that as these variables increase, the dependent variable decreases when holding other factors constant. The larger the magnitude of the coefficient, the more important the variable is in predicting the dependent variable.

4.3. Fine-Grained Evaluation

Decision tree Regressor:

```
actual_value = y_test.iloc[6]
actual_value
```

```
5600
```

```
predicted_value = best_model_dt.predict(X_test.iloc[[6]])
predicted_value
```

```
array([5553.06557377])
```

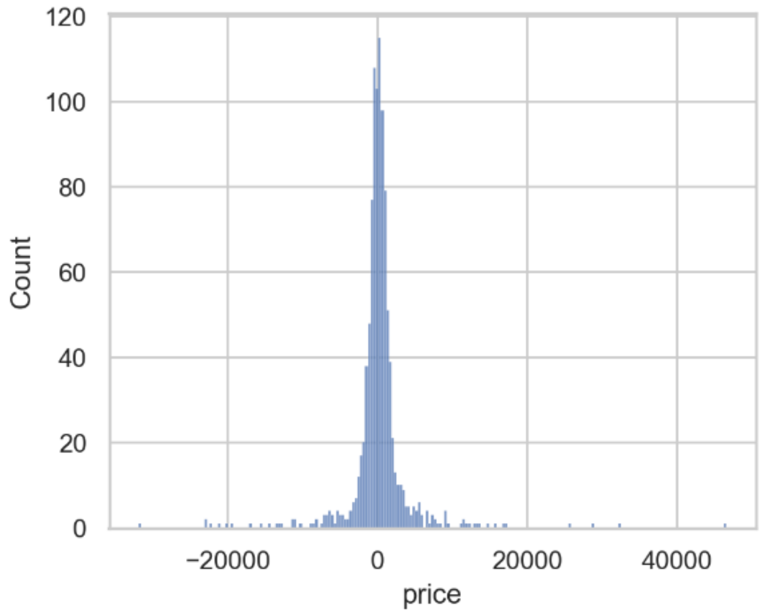
```
predicted_value - actual_value
```

```
array([-46.93442623])
```

From the above code snippets, the difference between predicted and actual values for the 7th instance -46.93 which is negative and not close to zero. So It is under predicted for this instance.

Generally, If error(i.e. difference between predicted and actual value) is positive, then that is over predicted. If error is 0 or close to zero, that indicates that the model made a good prediction for that particular instance.

```
#residual plot for sample of 1000 points
sns.histplot(residuals.sample(1000));
```



In the above histogram, Most of the residuals lies around zero which means that the model is predicting better for most of the data points as the difference between predicted values and actual values of test data set is 0, but only with few instances, the model is underpredicting and for some instances overpredicting. The above graph is plotted on 1000 samples of data to get good visualization and understanding.

KNN regressor:

```
actual = y_test.iloc[6]
actual
```

```
5600
```

```
predicted = best_model_knn.predict(X_test.iloc[[6]])
predicted
```

```
array([5495.18181818])
```

```
predicted - actual
```

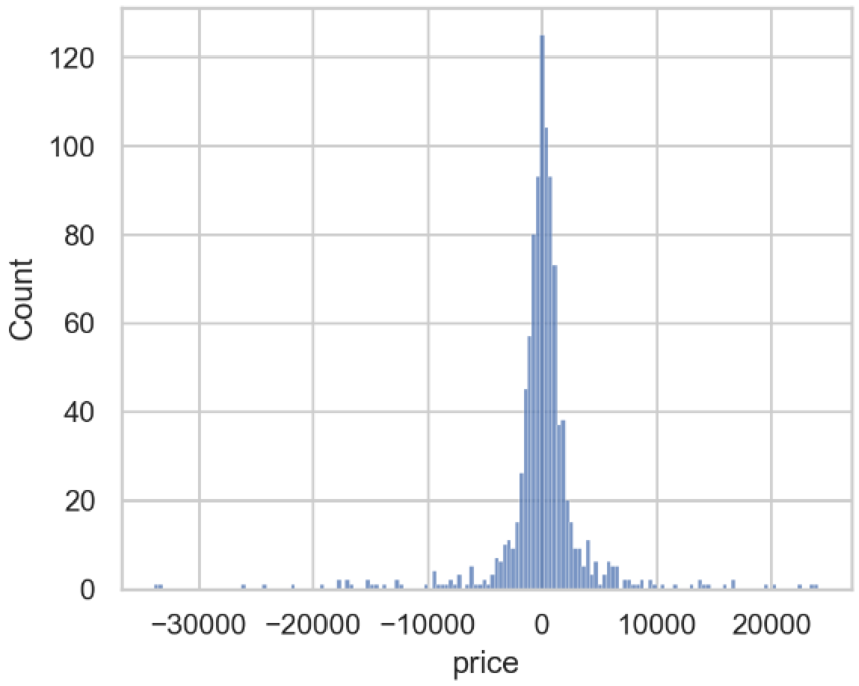
```
array([-104.81818182])
```

The difference between the predicted and actual value of KNN regressor model called as error which is -104 which is more far from 0 compared to decision tree model. So, the model underpredicted for this level of instance.

```
residuals = best_model_knn.predict(X_test) - y_test
```

```
plt.figure(figsize = (5,4))

#residual plot
sns.histplot(residuals.sample(1000));
```



From the above histogram, it is clear that most points lie around zero which means that the difference between predictive price and actual price is 0. But there are few errors with -30000 and 20000 which are underpredicted and overpredicted respectively which is not good.