1. Data/Domain Understanding and Exploration

1.1 Importing all the important packages

The first step is to import all the important packages such as pandas, numpy, matplotlib, seaborn, etc.

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
pd.options.display.float_format = '{:.2f}'.format
```

1.2 Load the Data

To load the dataset, I will use pandas.

```
adv = pd.read_csv(r"C:\Users\asus\Desktop\Data_Science\Principles of Data Science\adverts.csv")
```

1.3 Sample Observation

We can use ". head()" to get an understanding of the data frame.

```
adv.head(2)
```

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	crossove
0	202006039777689	0.00	NaN	Grey	Volvo	XC90	NEW	NaN	73970	SUV	
1	202007020778260	108230.00	61	Blue	Jaguar	XF	USED	2011.00	7000	Saloon	

1.4 Meaning and Type of Features

To identify the column names and types of each column we can use ".info()".

```
adv.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
```

So, we have 402,005 rows and 12 columns in this data set.

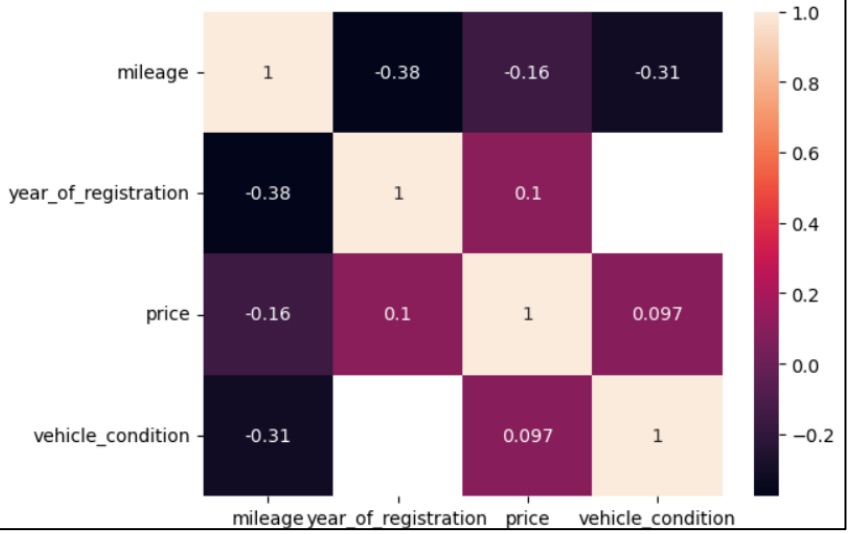
Meaning and types of each feature

1. **public_reference (integer)**: A unique identification number for every vehicle.
2. **mileage (float):** The total distance that the vehicle has travelled.
3. **reg_code (object):** The vehicle's registration code to identify the year of registration.
4. **standard_color (object):** The exterior colour of the vehicle.
5. **standard_make (object)**: The vehicle's brand or manufacturer.
6. **standard_model (object):** The vehicle's specific model within the brand.
7. **vehicle_condition (object):** Determines whether the vehicle is new or used.
8. **year_of_registration (float)**: The year when the vehicle was first registered.
9. **price (integer):** The selling price of a vehicle.
10. **body_type (object):** The shape or style of the vehicle's body, such as SUV or sedan.
11. **crossover_car_and_van (boolean):** To distinguish between a car and a van.
12. **fuel_type (object):** The type of fuel the vehicle uses (e.g., petrol, diesel, hybrid, etc).

1.5 Analysis of Predictive Power of Features

1.5.1 Correlation Matrix

```
numeric_data = adv[numeric_cols]
sns.heatmap(numeric_data.corr(), annot=True)
plt.show()
```



Based on the correlation matrix the best price predictors appear to be

1. year_of_registration (0.1): This correlates positively with price, indicating that newer vehicles are more expensive. This is most likely one of the strongest predictors in the matrix.

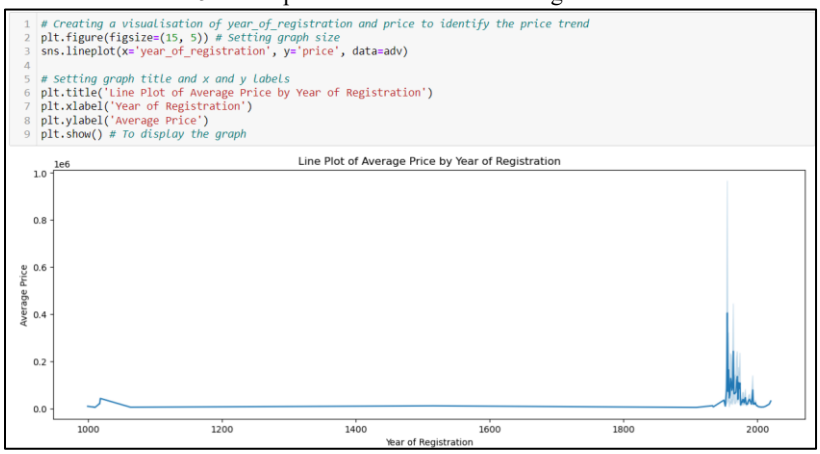
2. vehicle_condition (0.097): There is a weak positive correlation, suggesting that the condition of the vehicle has a less pronounced but still positive relationship with the price.

3. mileage (-0.16): There is a moderate negative correlation between mileage and price, indicating that vehicles with higher mileage are more likely to be cheaper.

1.5.2 Line plot of Price and Year of Registration

```
# Creating a visualisation of year_of_registration and price to identify the price trend
plt.figure(figsize=(15, 5)) # Setting graph size
sns.lineplot(x='year_of_registration', y='price', data=adv)

# Setting graph title and x and y labels
plt.title('Line Plot of Average Price by Year of Registration')
plt.xlabel('Year of Registration')
plt.ylabel('Average Price')
plt.show() # To display the graph
```

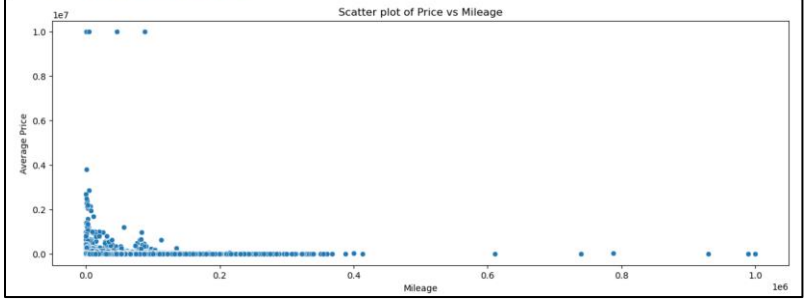


The line plot shows the average price of vehicles spiking significantly for certain years of registration, with most activity occurring near 2000.

1.5.3 Scatter-Plot of Mileage and Price

```
plt.figure(figsize=(15, 5)) # Setting graph size
sns.scatterplot(x='mileage', y='price',data=adv)

# Setting graph title and x and y labels
plt.title('Scatter plot of Price vs Mileage')
plt.xlabel('Mileage')
plt.ylabel('Average Price')
plt.show() # To display the graph
```

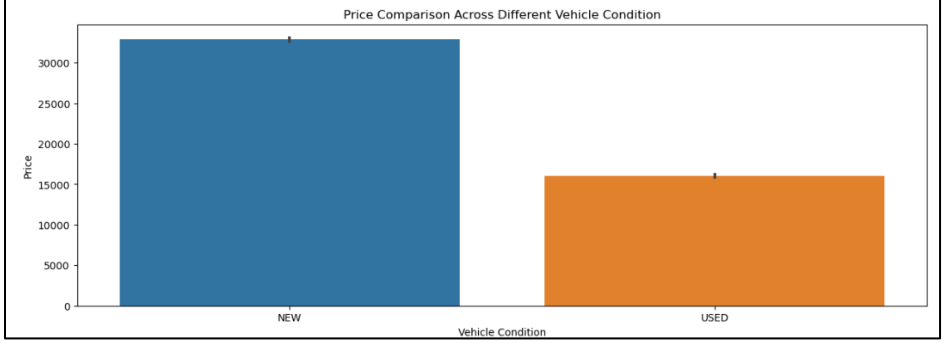


The scatter plot illustrates the relationship between vehicle mileage and price, with a concentration of data points at lower mileage and price ranges and a few high-priced vehicles as outliers across different mileage points.

1.5.3 Bar-Plot of Vehicle Condition based on Price

```
plt.figure(figsize=(15, 5)) # Setting graph size
sns.barplot(x='vehicle_condition', y='price', data=adv)

# Setting graph title and x and y labels
plt.title('Price Comparison Across Different Vehicle Condition')
plt.xlabel('Vehicle Condition')
plt.ylabel('Price')
plt.show() # To display the graph
```



The image displays a bar chart comparing the price of vehicles based on their condition, with the category "NEW" showing a significantly higher price than "USED", indicating that new vehicles are generally sold at a higher price point compared to used ones.

1.6 Data Exploration and Visualisation

1.6.1 Identifying Null Values

```
# Finding columns with missing values
adv.isnull().sum()
```

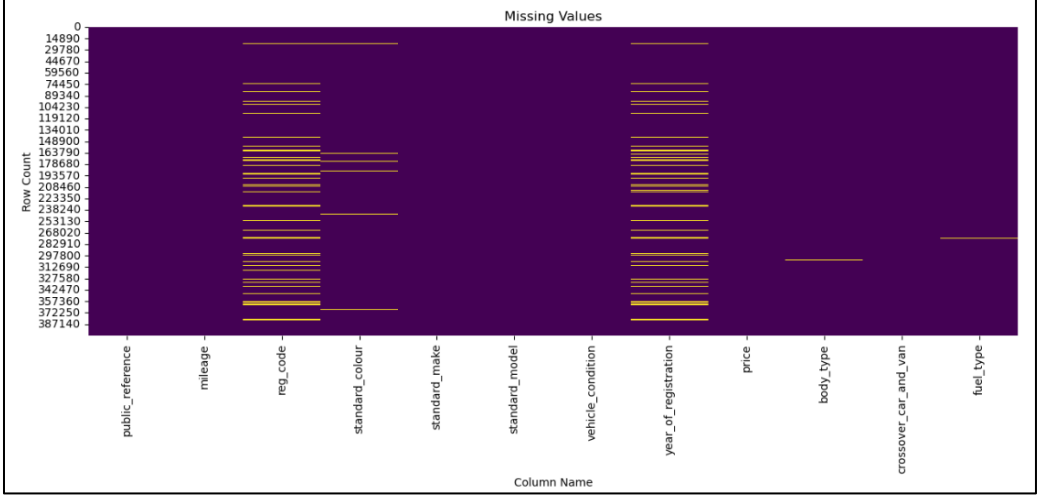
```
public_reference            0
mileage                   127
reg_code                31857
standard_colour          5378
standard_make               0
standard_model              0
vehicle_condition           0
year_of_registration    33311
price                       0
body_type                 837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

Let’s visualise the above numbers in a graph

1.6.1.1 Heatmap of Missing Values

```
plt.figure(figsize=(15,5)) # Setting graph size
sns.heatmap(adv.isnull(), cmap = 'viridis', cbar=False)

# Setting graph title and x and y labels
plt.title('Missing Values')
plt.xlabel('Column Name')
plt.ylabel('Row Count')
plt.show() # To display the graph
```



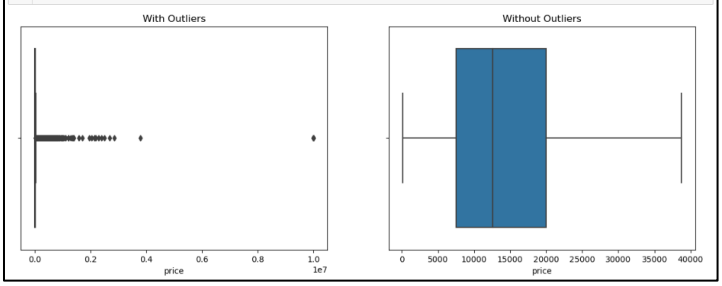
Now here we can see there are a few columns with missing values that are important for analysis. We need to fill those in for further and accurate analysis.

1.6.2 Identifying Outliers

1.6.2.1 Box-Plot of Price

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5)) # Setting graph size
# Visualization of Price column with outliers
sns.boxplot(data=adv, x='price', ax=axes[0]);
axes[0].set_title('With Outliers')

# Visualization of Price column without outliers
sns.boxplot(data=adv, x='price', showfliers=False, ax=axes[1]);
axes[1].set_title('Without Outliers')
plt.show() # To display the graph
```



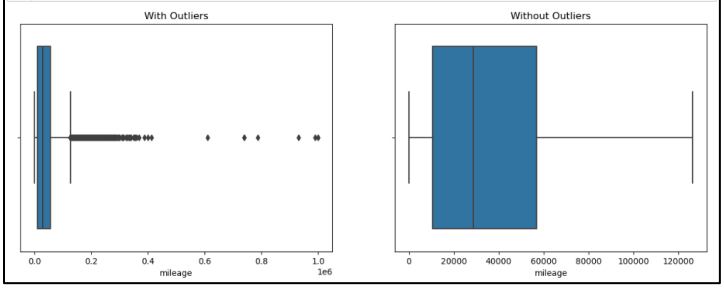
Price with Outliers: The boxplot displays the distribution of prices for the items, with a few extreme outliers showing higher prices and the majority of prices concentrated at the lower end of the scale. There appears to be a skew towards lower values as the median price is closer to the lower quartile.

Price without outliers: The majority of the prices are concentrated in a smaller range, as seen by this boxplot, which displays the range of prices excluding outliers. A line in the middle of the box represents the median price and suggests the dataset's central tendency.

1.6.2.2 Box-Plot of Mileage

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5)) # Setting graph size
# Visualization of Mileage column without outliers
sns.boxplot(data = adv, x='mileage', ax=axes[0])
axes[0].set_title('With Outliers')

# Visualization of Mileage column without outliers
sns.boxplot(data = adv, x='mileage', showfliers=False, ax=axes[1])
axes[1].set_title('Without Outliers')
plt.show() # To display the graph
```

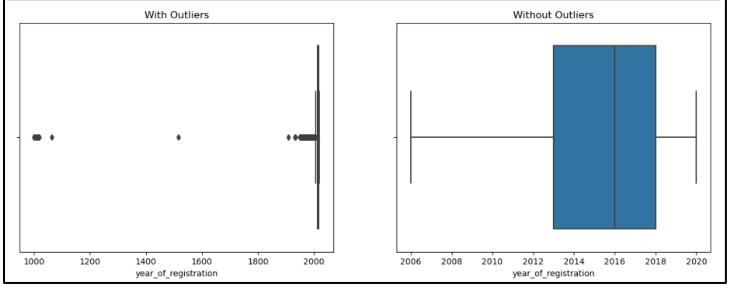


Mileage with Outliers: The vehicle mileage distribution is shown in a boxplot graph, with multiple outliers extending towards high mileage values and a median that is noticeably lower than the upper quartile. Up to a million mileage values are displayed on the scale, with the majority of the data being concentrated in the lower half of the range.
Mileage without Outlier: The mileage distribution of vehicles is shown in the boxplot with no outliers, where most data points fall into a moderate range. The vehicle mileage values appear symmetrically distributed, with the median value lying approximately in the middle of the range.

1.6.2.3 Box-Plot of Year of Registration

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5)) # Setting graph size
# Visualization of year_of_registration column with outliers
sns.boxplot(data=adv, x='year_of_registration', ax=axes[0])
axes[0].set_title('With Outliers')

# Visualization of year_of_registration column without outliers
sns.boxplot(data=adv, x='year_of_registration', showfliers=False, ax=axes[1]);
axes[1].set_title('Without Outliers')
plt.show() # To display the graph
```



Year of Registration with Outliers: The year of registration distribution for cars is shown in a boxplot, which places an average of registration years closer to the present and a few outliers representing much older cars.
Year of Registration without Outliers: The year of registration for a group of cars is displayed in this boxplot, which demonstrates a close grouping of years towards the most recent end of the scale.

2. Data Pre-Processing for Machine Learning

Data pre-processing is the process of preparing raw data for analysis by correcting or eliminating errors, inconsistencies, and irrelevant data points. This process involves recognizing and fixing issues such as missing values, outliers, duplicate entries, and errors in data entry or formatting. The goal of data cleaning is to ensure that the data is accurate, complete, and in an analysis-ready format, resulting in more reliable and meaningful insights from the data. This is an important step because data quality has a direct impact on the results of any Machine Learning Model.

2.1 Data Cleaning

Data cleaning is an important step in data preparation because it ensures that the data is accurate, complete, and of high quality before being used for Machine Learning.

2.1.1 Filling Missing Values

We have already seen we have a few columns with missing values. We need to fill those in for further and accurate analysis. To begin with year of registration column has around 33311 rows with missing values and the latest year in the data set is 2020 so by using condition I will fill a few rows with the year 2020. So, the condition would be vehicle condition is new and yor isnull.

```
adv.loc[(adv['vehicle_condition'] == 'NEW') & (adv['year_of_registration'].isnull()), 'year_of_registration'] = 2020
```

I am filling in missing values in the reg_code with 20 because as per the link cars that are registered in the year 2020 they have reg_code as 20 and 69. The reason I am using 20 here is that the cars that are sold from 1st March - 31st August the reg_code is 20 and as per the condition the cars that have mileage less than 1000 would be the cars that are registered in this period.

```
con1 = adv.loc[(adv['reg_code'].isna() | (adv['reg_code'] == '')) &
        (adv['year_of_registration'] == 2020) &
        (adv['mileage'] <= 1000), 'reg_code'] = '20'
```

I am filling in missing values in the reg_code with 69 because as per the link cars that are registered in the year 2020 they have reg_code as 20 and 69. The reason I am using 69 here is because for the cars that are sold from 1st September - 28/29th February the reg_code is 69 and as per the condition the cars that have mileage greater than 1000 would be the cars that are registered in this period.

```
con2 = adv.loc[(adv['reg_code'].isna() | (adv['reg_code'] == '')) &
        (adv['year_of_registration'] == 2020) &
        (adv['mileage'] >= 1000), 'reg_code'] = '69'
```

Now, to fill in the remaining values of the year of registration and mileage column I will be using a median number for each column.

```
median_mileage = adv['mileage'].median()
median_yor = adv['year_of_registration'].median()
print('Median of Mileage column is:', median_mileage)
print('Median of Mileage column is:', median_yor)
```

```
Median of Mileage column is: 28629.5
Median of Mileage column is: 2017.0
```

```
# Filling missing values using median for mileage and year of registration
adv['mileage'] = adv['mileage'].fillna(median_mileage)
adv['year_of_registration'] =adv['year_of_registration'].fillna(median_yor)
```

Now, I'll mark categorical columns like body_type, fuel_type, and standard_colour as Unknown. I could also fill in those with the most common value from each column, but I'm not doing that because it would be incorrect data and could affect our analysis, and these are the columns with the fewest missing values.

```
# Filling missing values of "body_type" column
adv['body_type'] = adv['body_type'].fillna('Unknown')
# Filling missing values of "fuel_type" column
adv['fuel_type'] = adv['fuel_type'].fillna('Unknown')
# Filling missing values of "standard_colour" column
adv['standard_colour'] = adv['standard_colour'].fillna('Unknown')
```

```
adv.isnull().sum()
```

```
public_reference          0
mileage                   0
reg_code                  0
standard_colour           0
standard_make             0
standard_model            0
vehicle_condition         0
year_of_registration      0
price                     0
body_type                 0
crossover_car_and_van     0
fuel_type                 0
dtype: int64
```

I have filled in all the missing values of each column the next step is to deal with outliers.

2.1.2 Dealing with Outliers

I will use the Interquartile Range (IQR) to deal with outliers in mileage, year of registration, and price column. The Interquartile Range (IQR) is a statistical dispersion measure calculated as the difference between a dataset's 75th (Q3) and 25th (Q1) percentiles. It represents the range in which the middle 50% of the data falls. Potential outliers include values that are significantly lower than Q1 – (1.5*IQR) or higher than Q3 + (1.5*IQR). This method is beneficial because it is less influenced by extreme values than other measures.

Step 1 (Price)

```
# Finding 25 and 75 percentile of price
prices = adv['price'].values
q1_price,q3_price = np.percentile(prices,[25,75])
print('Price range',q1_price,q3_price)
```

```
Price range 7495.0 20000.0
```

Step 2 (Price)

```
# Find IQR for price

price_iqr = q3_price - q1_price
print('IQR for price is',price_iqr)
```

```
IQR for price is 12505.0
```

Step 3 (Price)

```
# Finding Lower and Upper bound for price

lower_bound_price_val = q1_price - (1.5 * price_iqr)
upper_bound_price_val = q3_price + (1.5 * price_iqr)
print('lower bound value for price',lower_bound_price_val)
print('Upper bound value for price',upper_bound_price_val)
```

```
lower bound value for price -11262.5
Upper bound value for price 38757.5
```

Step 1 (Mileage)

```
# Finding 25 and 75 percentile of mileage
mileage = adv['mileage'].values
q1_mileage,q3_mileage = np.percentile(mileage,[25,75])
print('Mileage range', q1_mileage,q3_mileage)
```

```
Mileage range 10487.0 56852.0
```

Step 2 (Mileage)

```
# Find IQR for mileage

mileage_iqr = q3_mileage - q1_mileage
print('IQR for mileage is',mileage_iqr)
```

```
IQR for mileage is 46365.0
```

Step 3 (Mileage)

```
# Finding Lower and Upper bound for mileage

lower_bound_mileage_val = q1_mileage - (1.5 * mileage_iqr)
upper_bound_mileage_val = q3_mileage + (1.5 * mileage_iqr)
print('lower bound value for mileage',lower_bound_mileage_val)
print('Upper bound value for mileage',upper_bound_mileage_val)
```

```
lower bound value for mileage -59060.5
Upper bound value for mileage 126399.5
```

Step 1 (Year of Registration)

```
# Finding 25 and 75 percentile of year of registration
yor = adv['year_of_registration'].values
q1_yor,q3_yor = np.percentile(yor,[25,75])
print('Year of registration range', q1_yor,q3_yor)
```

```
Year of registration range 2014.0 2018.0
```

Step 2 (Year of Registration)

```
# Find IQR for Year of registration

yor_iqr = q3_yor-q1_yor
print('IQR for year of registration is', yor_iqr)
```

```
IQR for year of registration is 4.0
```

Step 3 (Year of Registration)

```
# Finding Lower and Upper bound for year of registration

lower_bound_yor_val = q1_yor - (1.5 * yor_iqr)
upper_bound_yor_val = q3_yor + (1.5 * yor_iqr)
print('lower bound value for year of registration',lower_bound_yor_val)
print('Upper bound value for year of registration',upper_bound_yor_val)
```

```
lower bound value for year of registration 2008.0
Upper bound value for year of registration 2024.0
```

By using the output of step 3, the upper and lower bounds of each column I will remove the outliers

```
# Using upper and lower bound to remove the outliers from the dataset
adv = adv.query('price <= @upper_bound_price_val')
adv = adv.query('mileage <= @upper_bound_mileage_val')
adv = adv.query('year_of_registration >= @lower_bound_yor_val')
```

After removing the outliers my new dataset length is 348523 rows and 12 columns.

```
adv.shape
```

```
(348523, 12)
```

2.2 Feature Engineering

In data science, feature engineering is the process of converting raw data into more useful and informative features for further analysis. It entails creating new variables or modifying existing ones to better capture the underlying patterns and relationships in the data, resulting in more accurate and insightful analytical outcomes. This step is critical in preparing data for effective analysis and modelling.

```
# Here, I am changing values of new and used vehicle condition for further analysis
adv['vehicle_condition'] = adv['vehicle_condition'].map({'NEW': True, 'USED': False})
```

```
# Changing data type of year of registration coloumn as integer
adv['year_of_registration'] = adv['year_of_registration'].astype(int)
```

I have changed the data type of the column vehicle condition and year of registration for further analysis.

```
adv.head(1)
```

index	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	cr
0	1	202007020778260	108230.00	61	Blue	Jaguar	XF	False	2011	7000	Saloon

I am creating a new column “vehicles_age” to identify the age of a vehicle by subtracting the current year (i.e. 2020) from the year of registration for further analysis.

```
adv['vehicles_age'] = 2023 - adv['year_of_registration']
adv.head(1)
```

:ode	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	crossover_car_and_van	fuel_type	vehicles_age
61	Blue	Jaguar	XF	False	2011	7000	Saloon	False	Diesel	12

2.3 Subsetting (e.g., feature selection and row sampling)

Subsetting is the process of selecting certain parts of your data for further analysis. This process can include two major aspects. Feature selection is the process of selecting only specific columns from a dataset, and row sampling is the process of selecting a subset of the dataset's rows. Both of these practices are important for making data more manageable and increasing analysis efficiency.

2.3.1 Feature Selection

I am removing the crossover_car_and_van, public_reference, and reg_code columns from the dataset because it is obvious that they play no role in predicting the vehicle's price. I am removing that column from the dataset because it is an irrelevant feature.

```
adv.drop('public_reference', axis=1, inplace=True)
adv.drop('crossover_car_and_van', axis=1, inplace=True)
adv.drop('reg_code', axis=1, inplace=True)
```

After removing the unnecessary columns, I will rearrange the dataset for better data framing and understanding.

```
# Rearranging the columns
temp = ['standard_make', 'standard_model', 'standard_colour', 'body_type',
        'fuel_type', 'vehicle_condition', 'mileage', 'year_of_registration', 'vehicles_age', 'price']
```

I have created a new variable to rearrange the columns. So, I will reassign the new variable to my actual dataset name (i.e. adv)

```
temp = adv[temp]
```

```
adv = temp
```

After doing feature engineering and feature selection below is the new data frame which we will use for Encoding.

```
adv.head(1)
```

	standard_make	standard_model	standard_colour	body_type	fuel_type	vehicle_condition	mileage	year_of_registration	vehicles_age	price
0	Jaguar	XF	Blue	Saloon	Diesel	False	108230.00	2011	12	7000

2.4 Encoding

One of the most crucial steps in the development of a machine-learning model is this one. The features that are of the string type are encoded as part of this process. It is crucial to convert the string values into the proper numeric values because machine learning algorithms cannot accept string data as input.

I am using 2 types of encoding to convert string data type into numeric data type.

1) One-Hot Encoding
2) Target Encoding

I have columns with numerous unique values, such as standard make and model, which is why I am using the target encoding method. Because of these two columns, there will be too many columns if I use one-hot encoding for them.

Importing important packages from scikit-learn.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

Creating a variable for one-hot encoding and using pd.get_dummies for encoding.

```
ohe_cat_cols = ['standard_colour','body_type','fuel_type','vehicle_condition']
```

```
one_hot_encoded_df = pd.get_dummies(adv[ohe_cat_cols])
```

Encoding is successful for the above columns and now the next step is to encode the remaining columns with target encoding.

```
one_hot_encoded_df.head(1)
```

	standard_colour_Beige	standard_colour_Black	standard_colour_Blue	standard_colour_Bronze	standard_colour_Brown	standard_colour_Burgundy	standard_c
0	0	0	1	0	0	0	

1 rows × 51 columns

The target encoding is applying mean encoding to categorical features, creating new columns that represent each category by the average price of the corresponding target variable.

```
for col in ['standard_make', 'standard_model']:
    target_means = adv.groupby(col)['price'].mean()
    adv[col + '_encoded'] = adv[col].map(target_means)
```

After encoding categorical columns, I will merge both the variables of one-hot encoding and target encoding.

```
data_encoded = pd.concat([adv.drop(ohe_cat_cols + ['standard_make', 'standard_model'], axis=1),
                          one_hot_encoded_df], axis=1)
```

```
data_encoded.head(1)
```

	mileage	year_of_registration	vehicles_age	price	standard_make_encoded	standard_model_encoded	standard_colour_Beige	standard_colour_Black	stand
0	108230.00	2011	12	7000	20966.08	15143.16	0	0	

1 rows × 57 columns

2.6 Splitting Data

After feature engineering and encoding categorical data, the next important step is to split data into training and testing data. I am dividing my dataset into 2 parts, 80% for training purposes and 20% for testing.
Here, I am creating 2 variables X and y and assigning all the columns to the variable X except the price column. Y variable will be the variable of my target column i.e. price.

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

Importing packages from sci-kit learn and dividing data into 2 (train and test)

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((278818, 56), (278818,), (69705, 56), (69705,))
```

Following data splitting, I have 69,705 rows and 56 columns for testing with 69,705 rows of my targeted column of price, 278,818 rows and 56 columns in the X train dataset, 278,818 rows with my targeted column price.

3. Model Building

This is an important step in Machine Learning. In this step machine learning engineer will decide the most suitable model to get the maximum accurate output of the task. I will train and test my data on 3 different models i.e. Linear regressor, Decision Tree regressor, and KNN regressor.
Importing all the important packages to train/test and to analyse the performance.

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
```

3.1.1 Linear Regressor

To perform a linear regression, I will fit my train data of X and y.

```
%%time
# Linear Regressor
lr = LinearRegression()
lr.fit(X_train, y_train)
```

After training my model on the training dataset, now I will test my model by using predict() and calculating the r2 score, mean absolute error and mean squared error. The higher r2 score indicates the best suitable model.

```
lr_y_pred = lr.predict(X_test)
lr_r2_Score = r2_score(y_test, lr_y_pred)*100
lr_mean_abs_err = mean_absolute_error(y_test, lr_y_pred)
lr_mean_sqr_err = np.sqrt(mean_squared_error(y_test, lr_y_pred))
```

3.1.1 R2 score, mean absolute and squared error of Linear Regression

```
r2 score:  80.52076150141063 %
mean absolute error:  2658.7951262799475
mean squared error:  3570.705889373439
```

3.1.2 Decision Tree Regressor

To perform a decision tree regression, I will fit my train data of X and y.

```
%%time
# Decision Tree Regressor
dtr = DecisionTreeRegressor(random_state=0)
dtr.fit(X_train, y_train)
```

After training my model on the training dataset, now I will test my model by using predict() and calculating the r2 score, mean absolute error and mean squared error. The higher r2 score indicates the most suitable model.

```
dtr_y_pred = dtr.predict(X_test)
dtr_r2_Score = r2_score(y_test, dtr_y_pred)*100
dtr_mean_abs_err = mean_absolute_error(y_test, dtr_y_pred)
dtr_mean_sqr_err = np.sqrt(mean_squared_error(y_test, dtr_y_pred))
```

3.1.2 R2 score, mean absolute and squared error of Decision Tree Regressor

```
r2 score:  88.69151041661999 %
mean absolute error:  1795.7760675769503
mean squared error:  2720.6357060584824
```

3.1.3 KNN Regressor

To perform a KNN regression, I will fit my train data of X and y.

```
%%time
# KNN Regressor
knnr = KNeighborsRegressor(n_neighbors=11)
knnr.fit(X_train, y_train)
```

After training my model on the training dataset, now I will test my model by using predict() and calculating the r2 score, mean absolute error and mean squared error. The higher r2 score indicates the most suitable model.

```
knnr_y_pred = knnr.predict(X_test)
knnr_r2_Score = r2_score(y_test, knnr_y_pred)*100
knnr_mean_abs_err = mean_absolute_error(y_test, knnr_y_pred)
knnr_mean_sqr_err = np.sqrt(mean_squared_error(y_test, knnr_y_pred))
```

3.1.3 R2 score, mean absolute and squared error of KNN Regressor

```
r2 score:  86.08674204105671 %
mean absolute error:  2140.093915266284
mean squared error:  3017.7448050504427
```

3.2 Model Selection

After model training and testing this step of model selection is important this step includes identifying the most suitable model for your task.

After performing all the regression models let's compare the performance by creating a data frame.

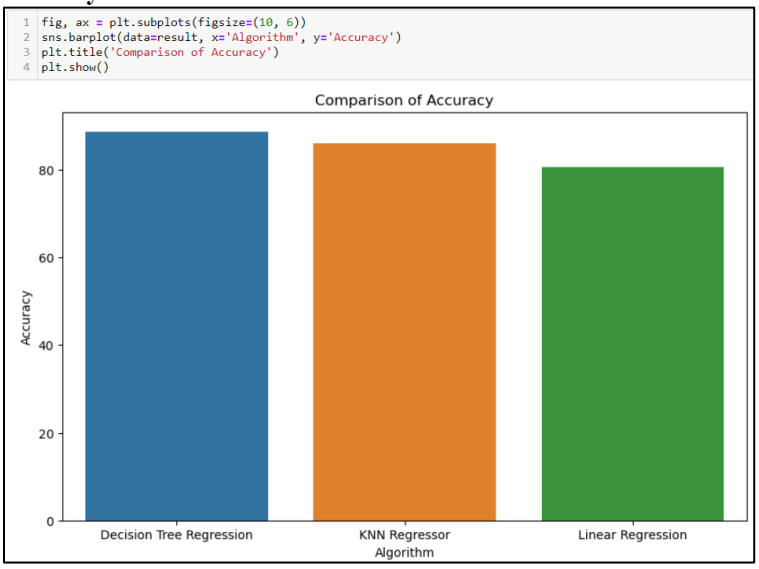
```
result = (pd.DataFrame({'Algorithm': ['Linear Regression','Decision Tree Regression','KNN Regressor'],
                        'Accuracy':[lr_r2_Score, dtr_r2_Score, knnr_r2_Score],
                        'Mean Absolute Error':[lr_mean_abs_err, dtr_mean_abs_err, knnr_mean_abs_err],
                        'Mean Squared Error':[lr_mean_sqr_err, dtr_mean_sqr_err, knnr_mean_sqr_err]
}).sort_values('Accuracy', ascending=False))
result
```

	Algorithm	Accuracy	Mean Absolute Error	Mean Squared Error
1	Decision Tree Regression	88.69	1795.78	2720.64
2	KNN Regressor	86.09	2140.09	3017.74
0	Linear Regression	80.52	2658.80	3570.71

Let's visualise the above data frame in a bar graph.

3.2.1 Comparison of Accuracy

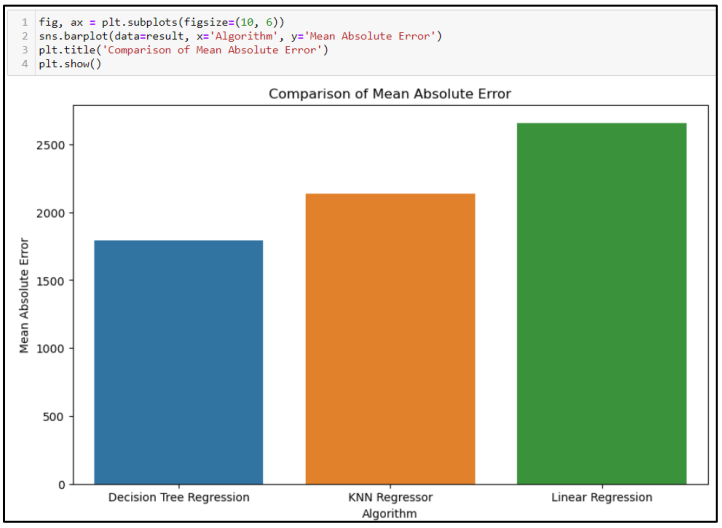
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=result, x='Algorithm', y='Accuracy')
plt.title('Comparison of Accuracy')
plt.show()
```



The graph indicates that the r2 score/Accuracy of decision tree regression is highest compared to KNN and Linear.

3.2.2 Comparison of Mean Absolute Error

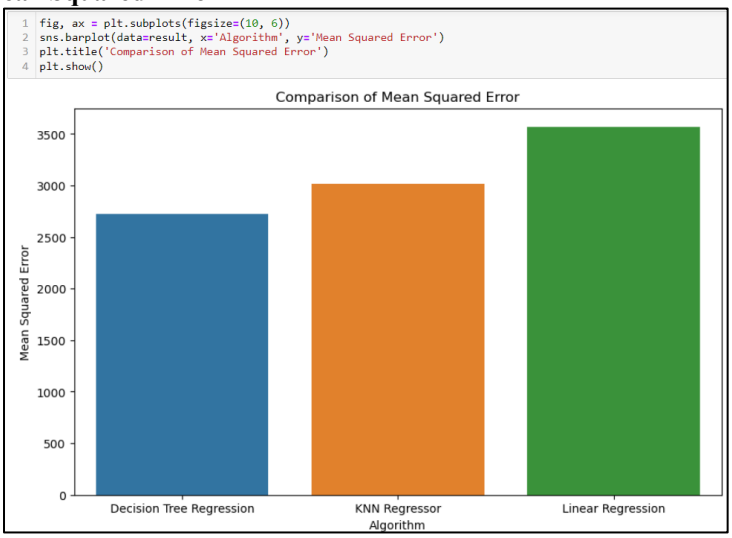
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=result, x='Algorithm', y='Mean Absolute Error')
plt.title('Comparison of Mean Absolute Error')
plt.show()
```



A lower Mean Absolute Error indicates that the differences between the predicted values and the actual values are smaller on average, suggesting a better performance of the predictive model. By looking at the graph we can say decision tree regressor is best compared to KNN and Linear regression.

3.2.3 Comparison of Mean Squared Error

```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=result, x='Algorithm', y='Mean Squared Error')
plt.title('Comparison of Mean Squared Error')
plt.show()
```



Similar to the Mean Absolute Error, a lower Mean Squared Error (MSE) is also indicative of better model performance. As per the graph decision tree is most effective for predicting a price compared to KNN and Linear regression.

The next step is to analyse the selected model. Here, I am selecting a decision tree regressor for producing a regression model for price prediction.

4. Model Analysis

This is a very important and final step for producing a regression model. In this step, we have to analyse the performance of our selected model.

4.1 Comparing Performance Metrics

This step is crucial because in this step we check and address underfitting and overfitting, this is essential for developing models that are robust and perform well when making predictions on new data.

```
dtr_y_pred_train = dtr.predict(X_train)
dtr_r2_Score_train = r2_score(y_train, dtr_y_pred_train)*100
```

```
dtr_r2_Score_train, dtr_r2_Score
```

```
(99.53241899178578, 88.69151041661999)
```

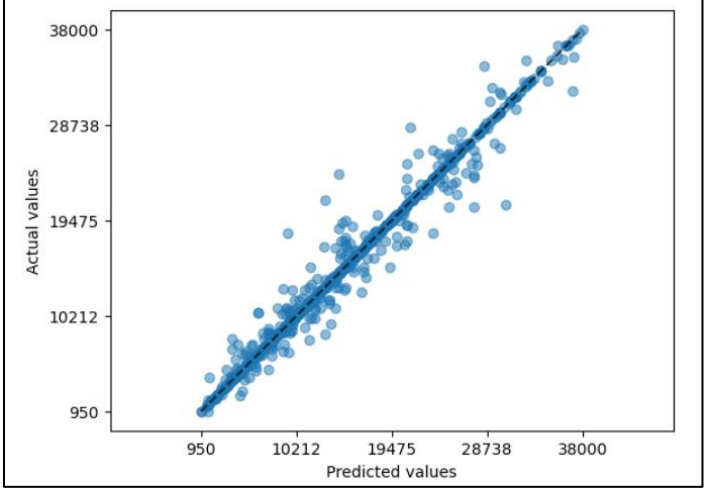
```
# Comparing Decision Tree Regressor's r2 Score of train and test data
if dtr_r2_Score_train > dtr_r2_Score:
    print("The model may be overfitting the training data.")
elif dtr_r2_Score_train < dtr_r2_Score or dtr_r2_Score_train < 0.5:
    print("The model may be underfitting the training data.")
else:
    print("The model has a balanced fit on the training and test data.")
```

```
The model may be overfitting the training data.
```

An R^2 score of 99.53% on the training data and 88.69% on the test data suggests that the Decision Tree Regressor model may be overfitting. While a certain degree of drop in performance from training to test data is normal.

4.2 Actual vs Predicted

```
from sklearn.metrics import PredictionErrorDisplay
PredictionErrorDisplay.from_estimator(
    dtr, X, y, kind="actual_vs_predicted", scatter_kwargs=dict(alpha=0.5)
);
```



The scatter plot shows a Decision Tree model's prediction closely aligning with the actual values, indicating accurate predictions, with a slight increase in prediction errors at higher value ranges.

4.3 Feature Importance

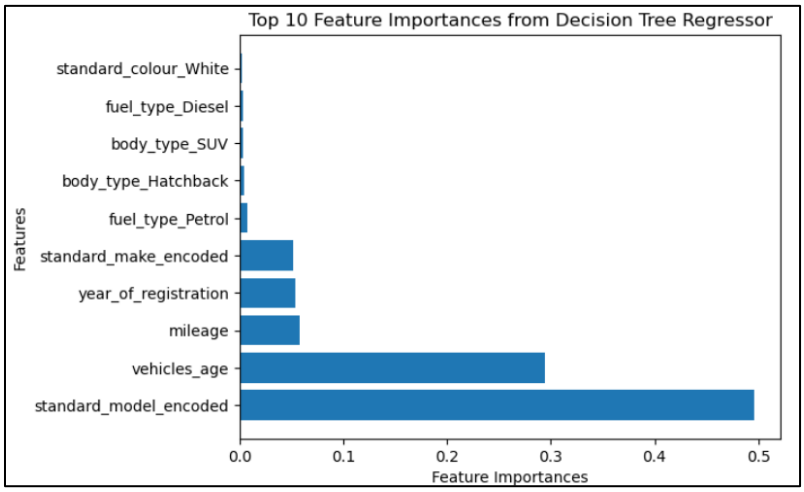
Feature importance seeks to figure out which different attributes of the data were most important when it comes to predicting the target variable (price)

Here, I am creating a variable to identify the top features that were used by the model for predicting the price.

```python
importances = dtr.feature_importances_
columns = list(X_train.columns)
df = pd.DataFrame({'features': columns, 'feature_importances': importances})

df_top10 = df.sort_values('feature_importances', ascending=False).head(10).reset_index(drop=True)
```

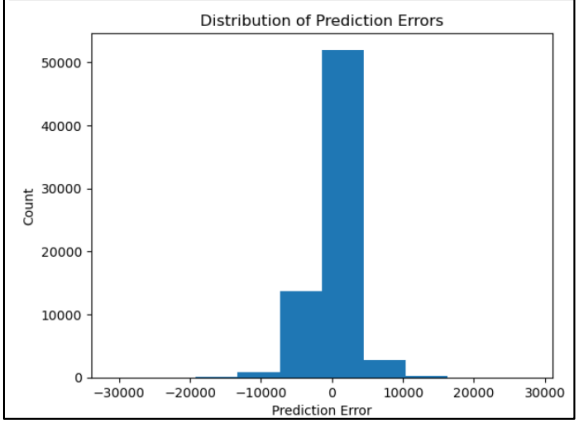
It is better to check the top 10 features because we have too many columns after we encoded the data for training and testing.



The key characteristics as identified by a Decision Tree Regressor are shown in the above graph. With 'standard_model_encoded' having the highest importance. This implies that the most important predictor of the target variable in the model is the model encoded version of the'standard_model' variable.

4.4 Analyse Individual Predictions and Distribution of Scores/Losses

```python
errors = y_test - dtr_y_pred
plt.hist(errors)
plt.xlabel("Prediction Error")
plt.ylabel("Count")
plt.title("Distribution of Prediction Errors")
plt.show()
```



The distribution of prediction errors for a regression model is shown by the histogram. The difference between the actual and predicted values (y_test - dtr_y_pred) is used to calculate the errors. Based on the graph, most predictions seem to be within a reasonable range of the actual values, with a peak located at the centre of the histogram (about zero error). However, the histogram also shows some errors spread out on both sides, indicating instances where the model under or overestimated the actual value.