MACHINE LEARNING ANALYSIS FOR PREDICTING CAR PRICE

TAWAKALIT AGBOOLA OMOLABAKE

DATA/DOMAIN UNDERSTANDING AND EXPLORATION

1.1. Meaning and Type of Features: Analysis of Distributions

During the stage of our analysis, we imported a dataset from a CSV file into a pandas DataFrame called 'car'. This dataset contains details of car advertisements.

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	crossov
0	202006039777689	0.0	NaN	Grey	Volvo	XC90	NEW	NaN	73970	SUV	
1	202007020778260	108230.0	61	Blue	Jaguar	XF	USED	2011.0	7000	Saloon	
2	202007020778474	7800.0	17	Grey	SKODA	Yeti	USED	2017.0	14000	SUV	
3	202007080986776	45000.0	16	Brown	Vauxhall	Mokka	USED	2016.0	7995	Hatchback	
4	202007161321269	64000.0	64	Grey	Land Rover	Range Rover Sport	USED	2015.0	26995	SUV	

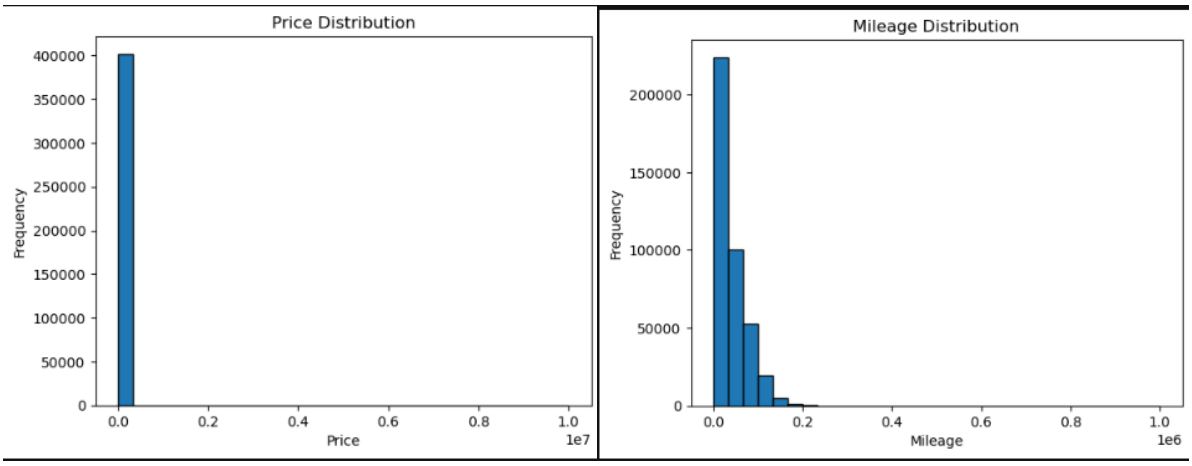
We categorized the features into two groups: quantitative and qualitative. The quantitative features consist of 'mileage' 'year_of_registration' and 'price' while the qualitative features include 'standard_colour', 'standard_make' and 'fuel_type'.

```
qnf = car.select_dtypes(exclude = 'object').columns.tolist()
print('Quantitative Features:',qnf)
print()
qlf = car.select_dtypes(include = 'object').columns.tolist()
print('Qualitative Features:', qlf)
```

```
Quantitative Features: ['public_reference', 'mileage', 'year_of_registration', 'price', 'crossover_car_and_
van']

Qualitative Features: ['reg_code', 'standard_colour', 'standard_make', 'standard_model', 'vehicle_conditio
n', 'body_type', 'fuel_type']
```

To understand the distribution of features we used histograms. The histogram, for 'price' gave us insights into the range average value and spread of car prices in the dataset. Similarly, by examining the histogram for 'mileage' we were able to grasp how car mileages are distributed. These visual representations help us understand how data is spread out and identify any outliers or irregularities.



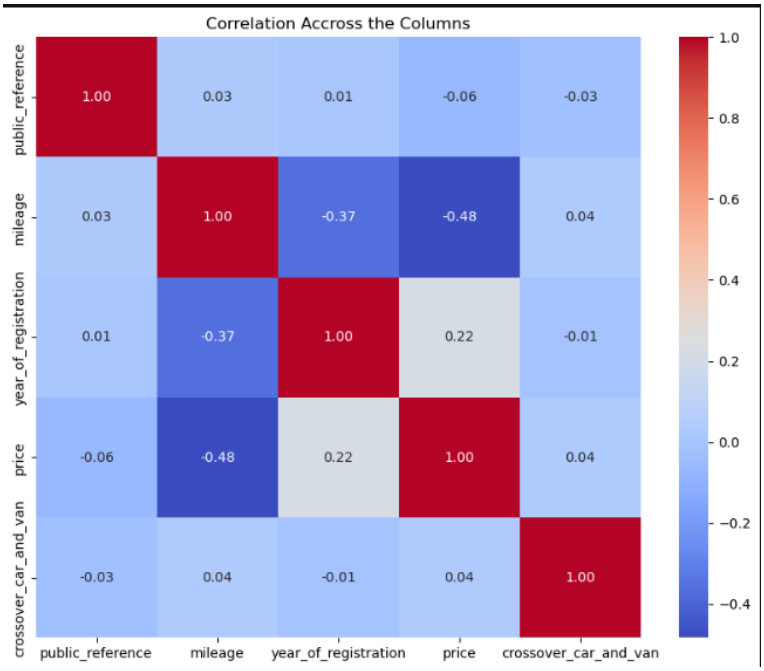
Furthermore, we conducted a summary analysis on the features providing numerical summaries of their distribution. This analysis included metrics such as count (number of observations) mean (value) deviation (measure of variability) minimum value and maximum value. It helped us gain an understanding of tendencies and variations present in the dataset. This exploratory data analysis is crucial for our modeling and predictive analytics stages.

Summary Statistics:

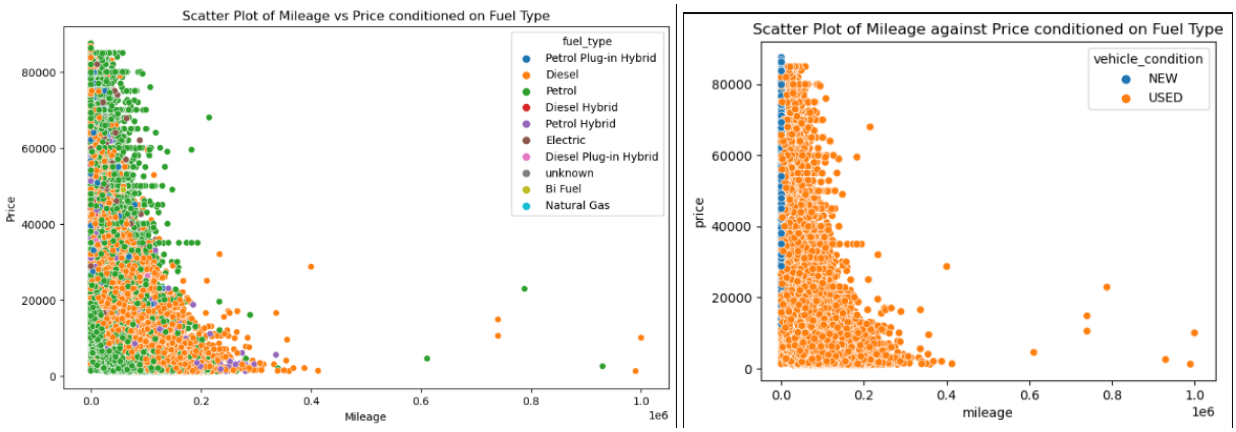
	public_reference	mileage	year_of_registration	price
count	4.020050e+05	401878.000000	368694.000000	4.020050e+05
mean	2.020071e+14	37743.595656	2015.006206	1.734197e+04
std	1.691662e+10	34831.724018	7.962667	4.643746e+04
min	2.013072e+14	0.000000	999.000000	1.200000e+02
25%	2.020090e+14	10481.000000	2013.000000	7.495000e+03
50%	2.020093e+14	28629. 500000	2016.000000	1.260000e+04
75%	2.020102e+14	56875.750000	2018.000000	2.000000e+04
max	2.020110e+14	999999.000000	2020.000000	9.999999e+06

1.2. Analysis of Predictive Power of Features

Additionally, we analyzed the strength of each feature in our dataset using correlation metrics and scatter plots.



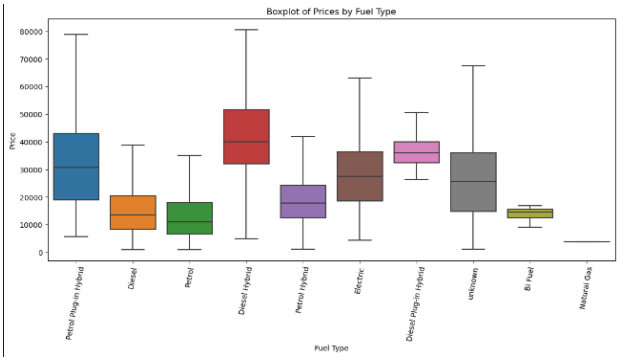
The correlation matrix gave us a way to measure how different features are connected to each other especially when it comes to the target variable 'price'. We paid attention to features, like 'mileage' 'year_of_registration'. Derived features such as 'vehicle_age' and 'mileage_per_year' to understand their relationship with the price.

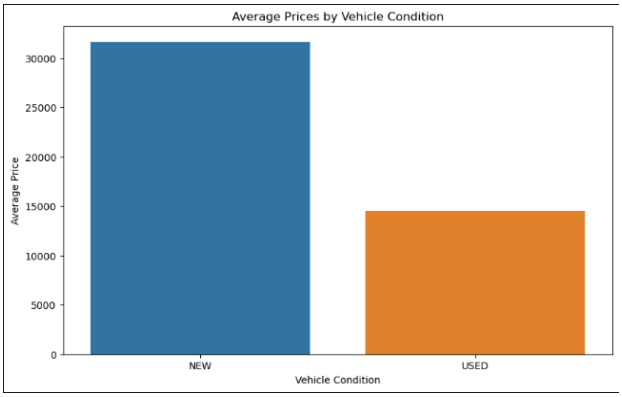


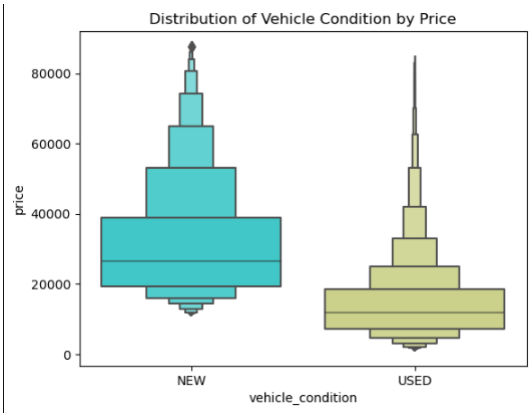
We used scatter plots to explore these relationships. For example, scatter plots of 'mileage' vs. 'price' and 'vehicle_age' vs. 'price' helped us identify trends and potential nonlinear connections. These plots also considered features like 'fuel_type', ‘Vehicle_condition' offering deeper insights into how these categories influence the price. This analysis is crucial for identifying predictors in the regression model and guiding feature selection and engineering.

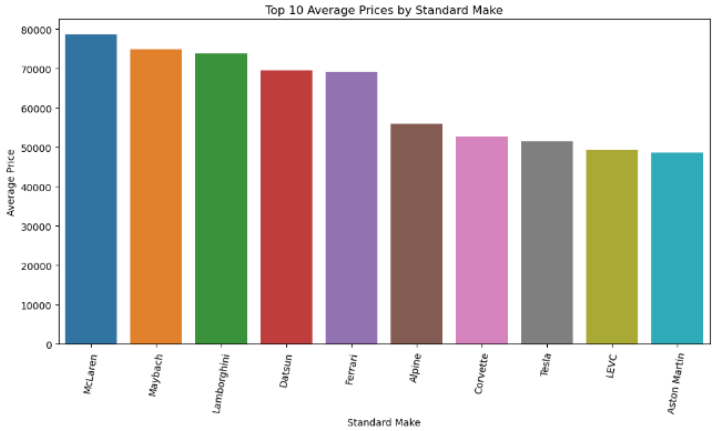
1.3. Data Processing for Data Exploration and Visualization

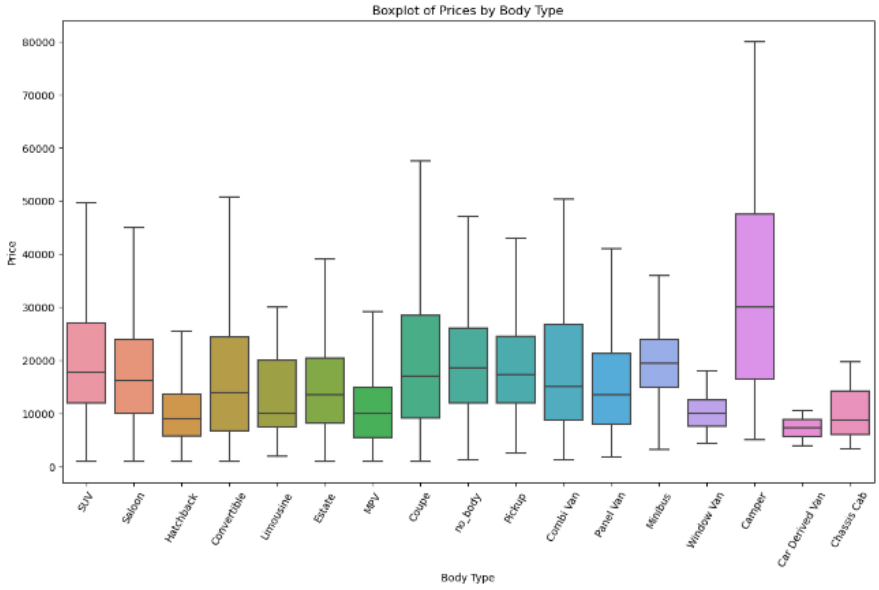
To ensure data visualization we took preprocessing steps. One important step involved handling missing values before creating representations of the dataset. After our dataset is cleaned, we visualize our data to show the trends in each features.







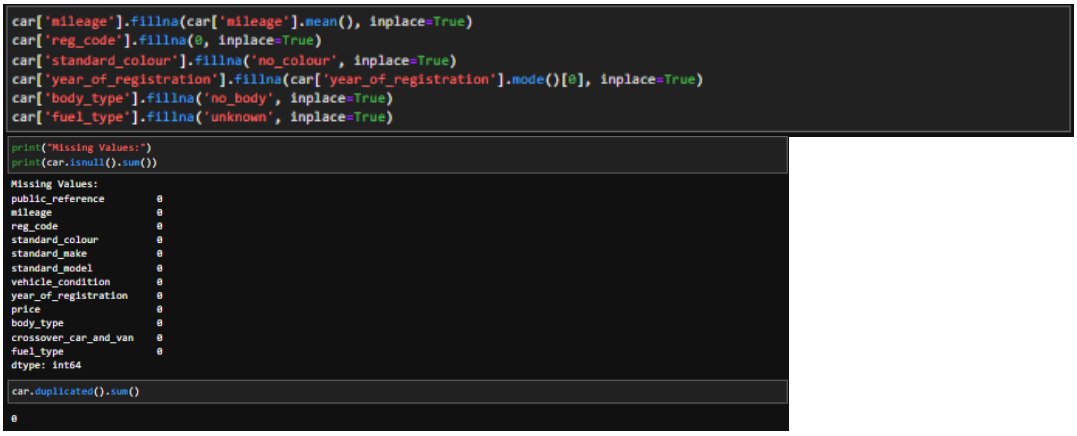




DATA PROCESSING FOR MACHINE LEARNING

2.1 Dealing with Missing Values, Outliers, and Noise

Missing Values: When dealing with values our strategy was based on the type of feature. For variables, like 'mileage' we used the value for imputation, which gives a representative measure of central tendency. To maintain the integrity of variables we used the mode to ensure that the most common category was preserved. This approach helps us minimize any distortion in the distribution of data.



```
car['mileage'].fillna(car['mileage'].mean(), inplace=True)
car['reg_code'].fillna(0, inplace=True)
car['standard_colour'].fillna('no_colour', inplace=True)
car['year_of_registration'].fillna(car['year_of_registration'].mode()[0], inplace=True)
car['body_type'].fillna('no_body', inplace=True)
car['fuel_type'].fillna('unknown', inplace=True)
```

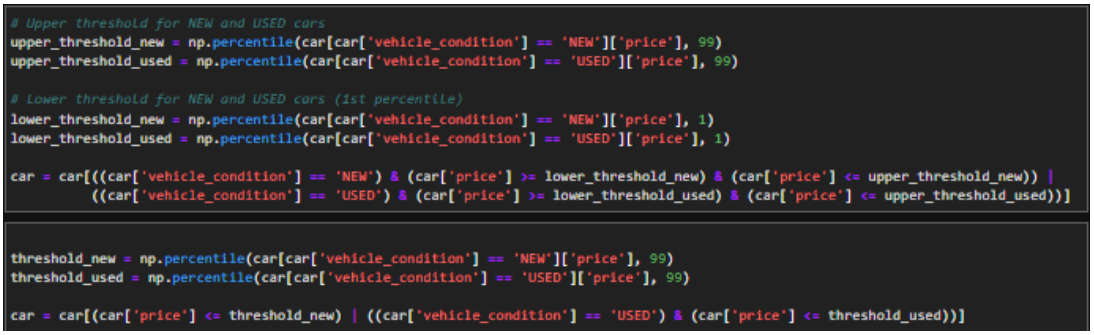
```
print("Missing Values:")
print(car.isnull().sum())
```

```
Missing Values:
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
```

```
car.duplicated().sum()
```

```
0
```

Outliers: When handling outliers we implemented a threshold method based on the vehicle's condition ('NEW' or 'USED'). We specifically chose this method because it considers the characteristics of vehicle conditions. Traditional outlier removal techniques may mistakenly eliminate data points, such as prices for new cars, which are not actually outliers but are inherent to the vehicles value. By setting condition thresholds we effectively preserved the attributes of new and used cars in our dataset. This ensures a representative analysis while maintaining data integrity particularly for features like 'price' that are significantly influenced by the vehicles condition.



```
# Upper threshold for NEW and USED cars
upper_threshold_new = np.percentile(car[car['vehicle_condition'] == 'NEW']['price'], 99)
upper_threshold_used = np.percentile(car[car['vehicle_condition'] == 'USED']['price'], 99)

# Lower threshold for NEW and USED cars (1st percentile)
lower_threshold_new = np.percentile(car[car['vehicle_condition'] == 'NEW']['price'], 1)
lower_threshold_used = np.percentile(car[car['vehicle_condition'] == 'USED']['price'], 1)

car = car[((car['vehicle_condition'] == 'NEW') & (car['price'] >= lower_threshold_new) & (car['price'] <= upper_threshold_new)) |
          ((car['vehicle_condition'] == 'USED') & (car['price'] >= lower_threshold_used) & (car['price'] <= upper_threshold_used))]
```

```
threshold_new = np.percentile(car[car['vehicle_condition'] == 'NEW']['price'], 99)
threshold_used = np.percentile(car[car['vehicle_condition'] == 'USED']['price'], 99)

car = car[(car['price'] <= threshold_new) | ((car['vehicle_condition'] == 'USED') & (car['price'] <= threshold_used))]
```

2.2 Feature Engineering, Data Transformations, Feature Selection

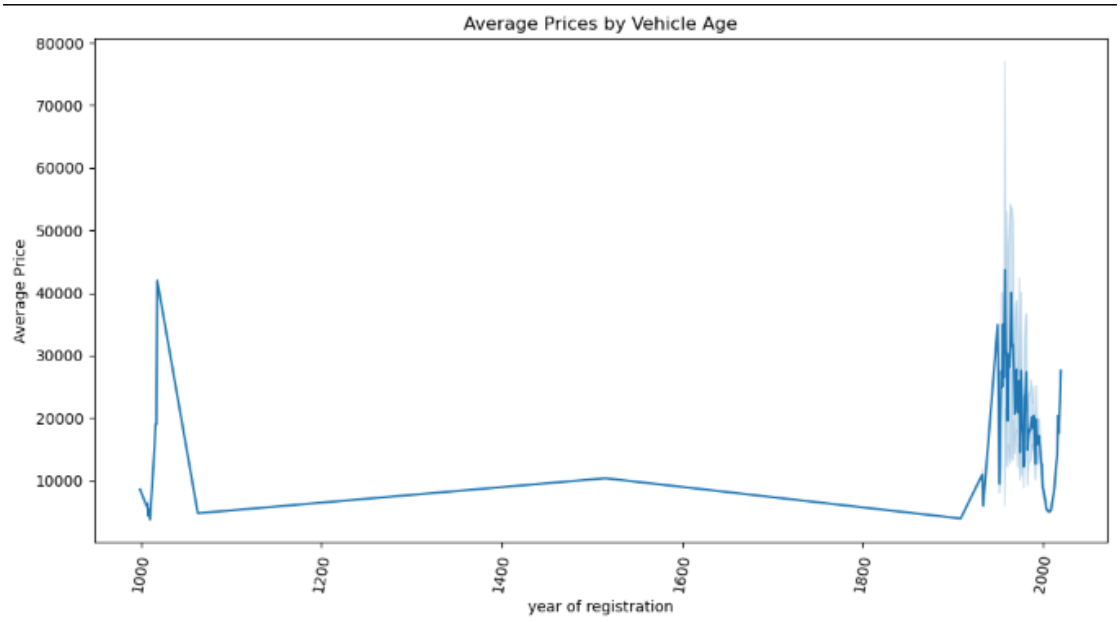
During the feature engineering phase, we introduced features to enhance model performance. The 'Vehicle_age' feature was created by subtracting the year of registration from the year providing a measure of how old a vehicle is—a critical factor in car valuation. Additionally, we computed 'mileage_per_year' by dividing mileage by the vehicle's age. This offers a normalized perspective on vehicle usage.

```
car['vehicle_age'] = 2024 - car['year_of_registration']
car['vehicle_age'].fillna(car['vehicle_age'].mean(), inplace=True)
car['vehicle_age'].head()
```

```
0     7.0
1    13.0
2     7.0
3     8.0
4     9.0
Name: vehicle_age, dtype: float64
```

```
car['mileage_per_year'] = car['mileage'] / car['vehicle_age']
car['mileage_per_year'].head()
```

```
0       0.000000
1    8325.384615
2    1114.285714
3    5625.000000
4    7111.111111
Name: mileage_per_year, dtype: float64
```



These engineered characteristics capture nuanced aspects of the data, which likely provide the model with relevant information to make accurate predictions about car prices. Including these features is expected to enhance the model's ability to capture variations in prices that may not be evident through the features.

One Hot Encoding: This step involves transforming variables into a format that can be used by machine learning algorithms to improve prediction accuracy. It converts data, such as 'fuel_type' and 'body_type' into format by creating new binary columns for each category.

```
# Encoding the categorical variables using one-hot encoder
columns = ['reg_code', 'standard_colour', 'standard_make', 'standard_model','vehicle_condition', 'body_type', 'crossover_car_and_van', 'fuel_type']
car_encoded = pd.get_dummies(car, columns=columns, drop_first=True)
```

```
car_encoded.head()
```

r	reg_code_02	reg_code_03	reg_code_04	reg_code_05	...	crossover_car_and_van_True	fuel_type_Diesel	fuel_type_Diesel Hybrid	fuel_type_Diesel Plug-in Hybrid	fuel_type_Electric	fuel_type_Natural Gas	fuel_type_Petrol	fuel_type_Petrol Hybrid	fuel_type_ Plug-in H
)	0	0	0	0	...	0	0	0	0	0	0	0	0	
;	0	0	0	0	...	0	1	0	0	0	0	0	0	
4	0	0	0	0	...	0	0	0	0	0	0	1	0	
)	0	0	0	0	...	0	1	0	0	0	0	0	0	
1	0	0	0	0	...	0	1	0	0	0	0	0	0	

Normalization: In this step numeric features like 'mileage' and 'year_of_registration' are standardized. This process scales the feature values so that they have a mean of 0 and variance of 1. It ensures that no single feature dominates the learning process of the model. Standardization is crucial for models that're sensitive to feature scales.

Splitting Data into Feature and Target Variable: Here the dataset is divided into two parts. Features (X). Target variable (y) which represents 'price'. This separation is essential in learning scenarios where the model needs to learn how input features relate to the desired outcome.

Train Test Split: The information is divided into two sets; the training set and the testing set. The training set is used to construct and fine tune the model while the testing set is utilized to assess how well it performs. This effective split approach is crucial in avoiding overfitting and underfitting allowing for an evaluation of the model's ability to make predictions.

```
from sklearn.model_selection import train_test_split

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

# Splitting the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

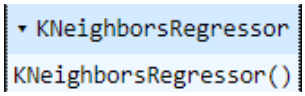
	Training set	Validation_set	Test set
X	275695, 1298	59078, 1298	59078, 1298
y	275695,	59078,	59078,

MODEL BUILDING

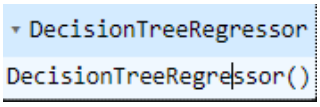
3.1 Algorithm Selection, Model Instantiation and Configuration

In our analysis of predicting vehicle prices, we selected three algorithms: K Nearest Neighbors (KNN) Decision Tree Regressor and Linear Regression. Each algorithm serves a purpose in our approach.

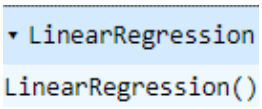
- **K Nearest Neighbors (KNN);** We chose to use KNN because of its effectiveness in capturing patterns through its proximity-based learning method. This model is well suited for identifying linear relationships that may exist within our dataset.



- **Decision Tree Regressor**; This algorithm plays a role in our analysis as it can handle linear data using a structured tree like model. Its interpretability is especially valuable as it allows us to understand how different features influence vehicle prices.



- **Linear Regression**: We utilized this algorithm due to its efficiency in analyzing relationships. Its simplicity is advantageous for establishing a foundation understanding of how various predictors correlate with vehicle prices.



These approaches enable us to evaluate and optimize each model ensuring an accurate prediction mechanism. Our approach combines both confirmatory data analysis methods following established industry standards to gain an understanding of the dataset's properties. This systematic methodology is crucial in creating a precise model that can be effectively used for predicting prices, in real world scenarios.

3.2 Grid Search, and Model Ranking and Selection.

The section involved exploring configurations of hyperparameters for the Decision Tree model using GridSearchCV. The grid search included parameters such as 'criterion' (mse and friedman_mse) 'max_depth', 'min_samples_split' 'min_samples_leaf' and 'max_features. Our objective was to find the combination of parameters for the Decision Tree model.

To tune the model, we iteratively trained it with parameter combinations and evaluated its performance using 5-fold cross validation. This approach allowed us to thoroughly explore the parameter space and obtain an assessment of how the model performed.

After tuning we identified the hyperparameters as follows; 'criterion'; 'friedman_mse' [1] 'max_depth'; None, 'max_features; 'sqrt' 'min_samples_leaf'; 1 and 'min_samples_split'; 10. Using these settings resulted in a Mean Squared Error of 20.9 million on the validation set along with an R Squared value of 0.87. These metrics indicate an improvement in performance compared to the configuration.

```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(dt_model, X, y, cv=5, scoring='neg_mean_squared_error')
print("CV Scores for Decision Tree:", cv_scores)
```

```
CV Scores for Decision Tree: [-13990571.14437133 -13672421.34501992 -13924647.79432839
 -14209290.63092761 -14319732.38779733]
```

```
from sklearn.model_selection import GridSearchCV

param_grid_dt = {
    'criterion': ['mse', 'friedman_mse'],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 10],
    'min_samples_leaf': [1, 4],
    'max_features': ['auto', 'sqrt']
}
dt_tuned = DecisionTreeRegressor(random_state=42)

grid_search_dt = GridSearchCV(dt_tuned, param_grid_dt, cv=5, scoring='neg_mean_squared_error', verbose=2)
grid_search_dt.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

GridSearchCV

estimator: DecisionTreeRegressor

DecisionTreeRegressor

DecisionTreeRegressor(random_state=42)

```
best_dt_model = grid_search_dt.best_estimator_
dt_pred_tuned = best_dt_model.predict(X_val)
dt_mse_tuned = mean_squared_error(y_val, dt_pred_tuned)

# Output
print(f'Best Decision Tree Model - Mean Squared Error: {dt_mse_tuned}')
print(f'Best Decision Tree Hyperparameters: {grid_search_dt.best_params_}')
```

```
Best Decision Tree Model - Mean Squared Error: 20865316.792056903
Best Decision Tree Hyperparameters: {'criterion': 'friedman_mse', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1,
'min_samples_split': 10}
```

```
cv_scores_tuned = cross_val_score(best_dt_model, X, y, cv=5, scoring='neg_mean_squared_error')
print("CV Scores for Tuned Decision Tree:", cv_scores_tuned)

rmse_scores_tuned = np.sqrt(np.abs(cv_scores_tuned))
print("CV RMSE Scores for Tuned Decision Tree:", rmse_scores_tuned)
```

```
CV Scores for Tuned Decision Tree: [-19217948.25755391 -20515651.15017591 -20520269.37860696
 -21859898.6892091  -20992807.65533224]
CV RMSE Scores for Tuned Decision Tree: [4383.82803695 4529.4206197  4529.93039445 4675.45705672 4581.79087861]
```

Although we planned to tune the KNN model, computational limitations prevented us from completing that step. As a result, our focus shifted towards utilizing the Decision Tree model with its optimized parameters, for analysis and application.

The meticulous and systematic approach taken when selecting models and tuning hyperparameters highlights the dedication to creating a reliable model.

MODEL EVALUATION AND ANALYSIS

4.1. Coarse-Grained Evaluation/Analysis

- **kNN Regressor**: kNN Regressor model was not satisfactory. It resulted in a Mean Squared Error (MSE) of approximately 183,796,125 and a negative R^2 value (0.1699). These metrics indicate that the model struggled to predict the target variable, which could mean that it was overfitting to the training data.

Mean Squared Error	183796125.4013345
R Squared	-0.16991999992666829

- **Decision Tree Regressor**: Decision Tree Regressor model showed improvement in its predictive capabilities. It achieved an MSE of 14,243,793 and an R^2 value of 0.9093. These results suggest that the model successfully captured the underlying patterns in the data and performed well.

Mean Squared Error	14243793.80080765
R Squared	0.9093337837998006

- **Linear Regression**: Linear Regression it demonstrated performance with an MSE of 20,490,559 and an R^2 value of 0.8695. Although it didn't perform as the Decision Tree model it still managed to capture a considerable amount of variance in the data.

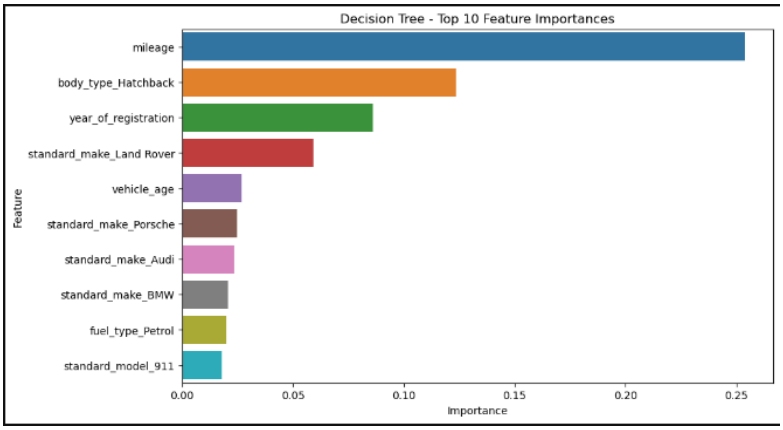
Mean Squared Error	20490559.157138184
R Squared	0.8695711625298362

These evaluations provide an overview of each model's performance by highlighting their strengths and limitations within this dataset context. The superior performance of the Decision Tree model can be attributed to its ability to handle relationships effectively. On the hand lower

performance from kNN suggests issues with parameter settings or a mismatch, between kNN algorithm characteristics and dataset attributes.

4.2. Feature Importance

The analysis suggests that the influential factor in determining the models' predictions is the "mileage" of a vehicle. This makes sense because typically a vehicle's mileage is closely related to its value. Moreover, specific body types such as "Hatchback" and certain brands like "Land Rover" also appear to have an impact. This indicates that the type of vehicle and its brand can greatly influence its price. Additionally, the year of registration, which reflects the age of the vehicle, plays a role in determining its price. This aligns with observations in the market where newer vehicles tend to have value. Other features like "vehicle_age" also contribute to the accuracy of the model but to an extent. The analysis of feature importance highlights how various factors such as mileage, make, model and age collectively affect a vehicle's price in the market.



4.3. Fine-Grained Evaluation

The detailed assessment of the regression models as shown by the scatter and residual plots gives us insights into how these models predict vehicle prices.

Before Fine Tuning:

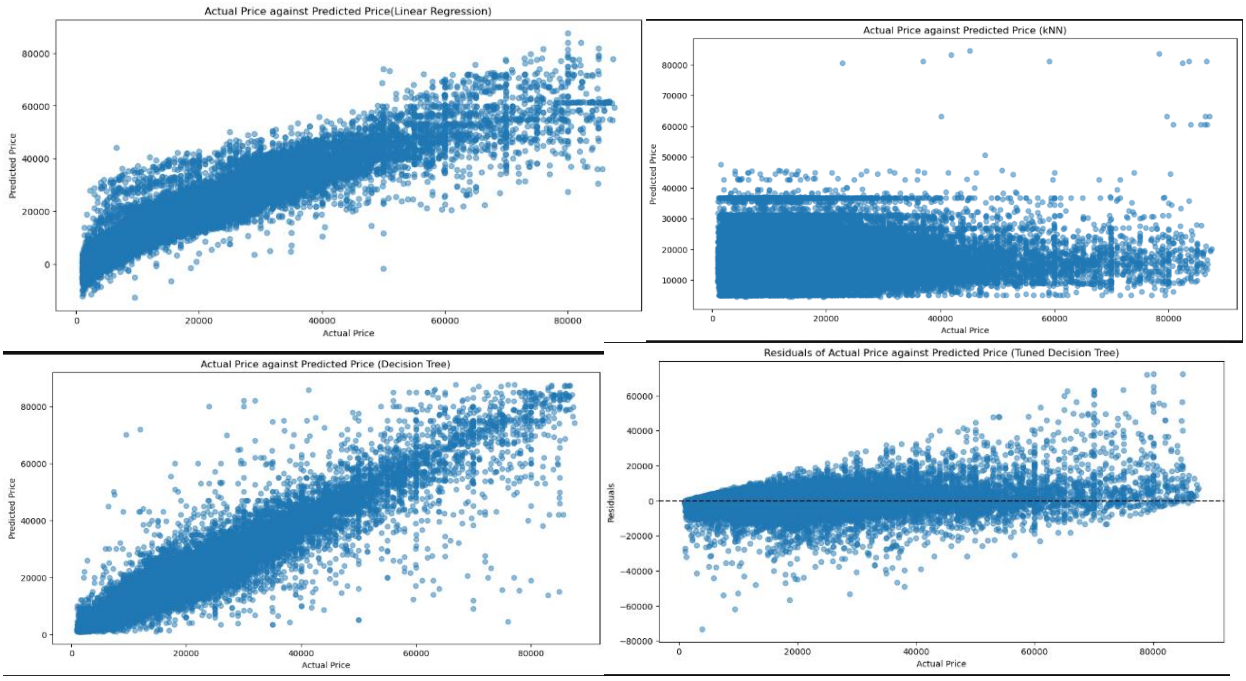
The scatter plots for KNN and Decision Tree exhibit a range especially for higher actual prices. This indicates that the accuracy of these models differs across price ranges.

The Linear Regression plot has a funnel shaped spread, which suggests that it might not be as accurate in predicting prices.

After Fine-Tuning the Decision Tree Model:

The residuals plot after tuning the Decision Tree model shows a concentration around the zero line. However, there are still some deviations at higher price points.

Analyzing residuals through a grained evaluation provides an understanding of how accurate the models predictions are. In the case of the Decision Tree model after tuning we can see that the residual plot displays data points clustering closely around the horizontal zero line. This indicates that its predictions are closer to the prices. However, there is still dispersion among priced cars suggesting that the model's predictive performance varies depending on the price range.



Reference:

[1] *Sklearn.tree.DecisionTreeRegressor*. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html (Accessed: January 2, 2024).