Student Name: Tooba Zahid (23721219)

1. Data/Domain Understanding and Exploration

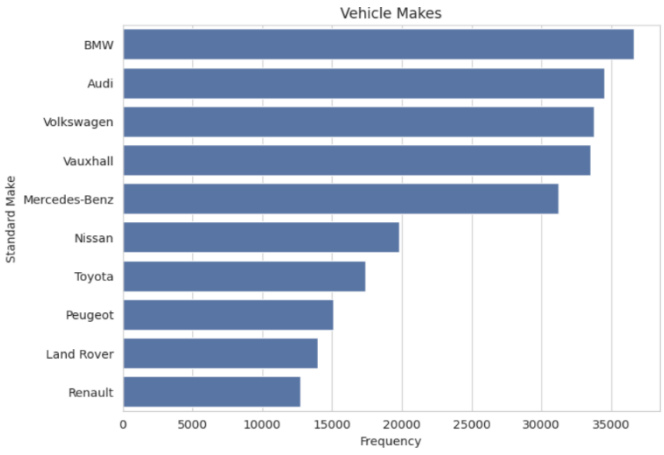
1.1 Meaning and Type of Features; Analysis of Distributions:

Here is the dataset having 402005 entries and 12 columns. Below is the brief initial exploration for each feature.

```
[8]  df_car.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
```

1. **Public_reference: Numerical (Integer)**; It is just a unique identifier for each vehicle.
2. **Mileage: Numerical (Float);** The mileage of the vehicle.
3. **Reg_code: (Categorical);** Registration code of the vehicle.
4. **Standard_colour**: **(Categorical)**; The colour of the vehicle. Such as black white
5. **Standard_make**: **(Categorical);** The manufacturer or brand of the vehicle.
6. **Standard_model**: **(Categorical);** The specific model of the vehicle.
7. **Vehicle_condition: (Categorical);** Condition of the vehicle, such as NEW or USED.
8. **Year_of_registration: (Numerical (Float);** year in which vehicle was registered.
9. **Price: (Numerical (Float);** The listed price of the vehicle.
10. **Body_type: (Categorical);** Type of the vehicle body, like SUV, Saloon, etc.
11. **Crossover_car_and_van: (Boolean);** Tells whether vehicle is a crossover between a car and a van.
12. **Fuel_type: (Categorical);** Type of fuel the vehicle uses, like Diesel, Petrol, etc.

Analysed the distributions of some selected feature, including year_of_registration, price, mileage, and standard_make. Basic statistics and distribution charts for these features is examined.

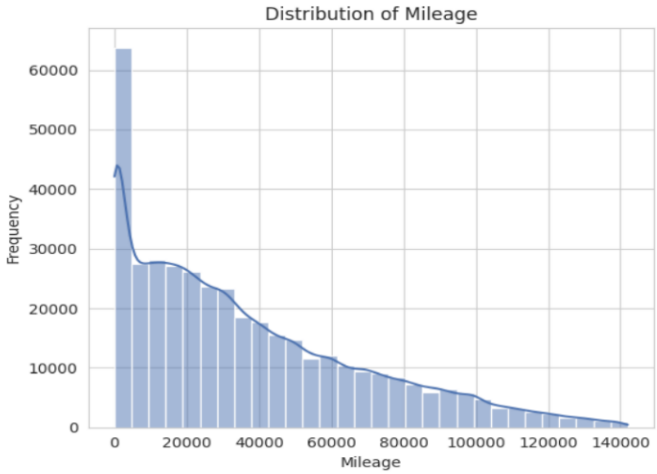


Distribution of Standard_Make:

The frequency of various brands in the dataset is displayed in the bar plot for the top ten car manufacturers. This provides information about the most popular brands on the market, which is helpful in figuring out consumer preferences.

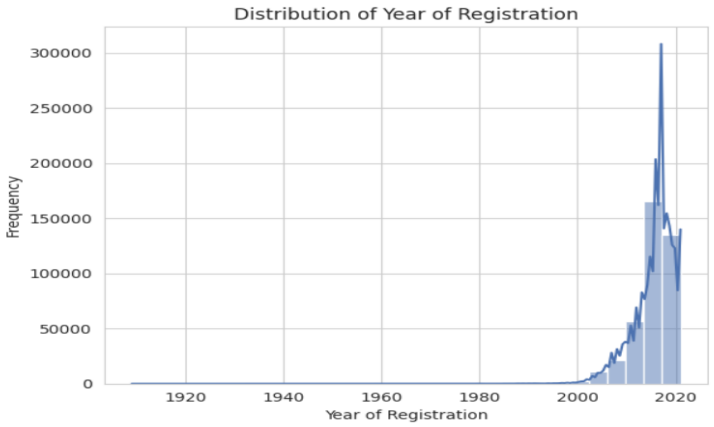
Distribution of Mileage:

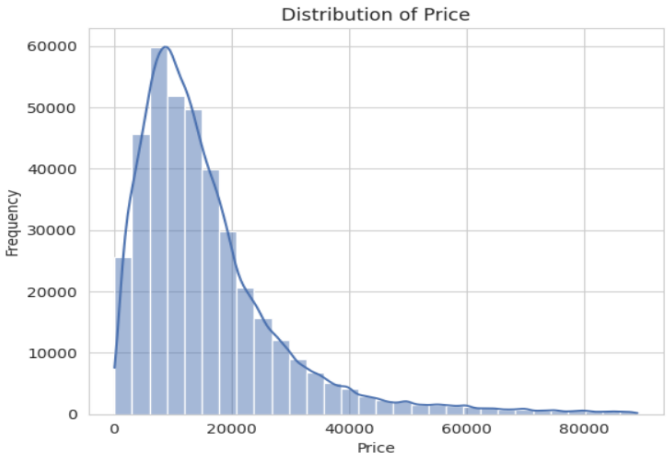
The "mileage" histogram has a right-skewed distribution, meaning that few cars have extremely high mileage and the majority have lower mileage. When it comes to used cars, this pattern is normal; newer cars with lower mileage are more common.



Year_of_registration:

Histogram shows that in this dataset, it seems that newer cars are more common, and that the frequency gradually decreases as the registration year moves further back in time. This may reflect a general trend in the used automobile market or the dataset's emphasis on more recent models.



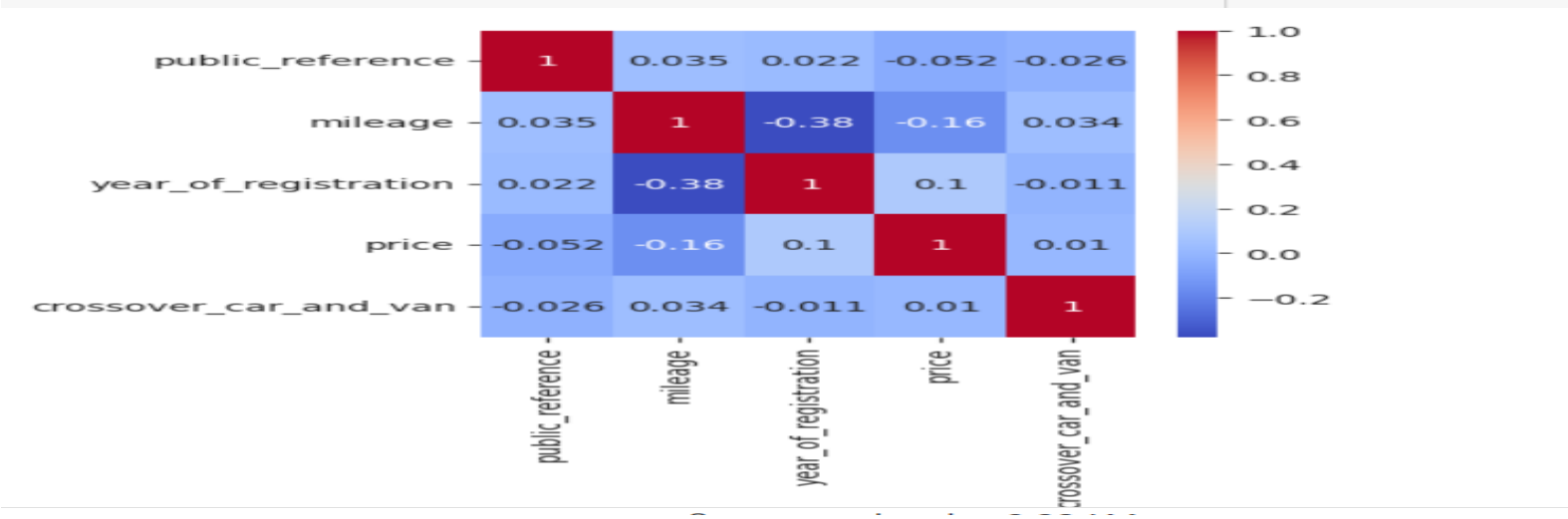


Price Distribution:

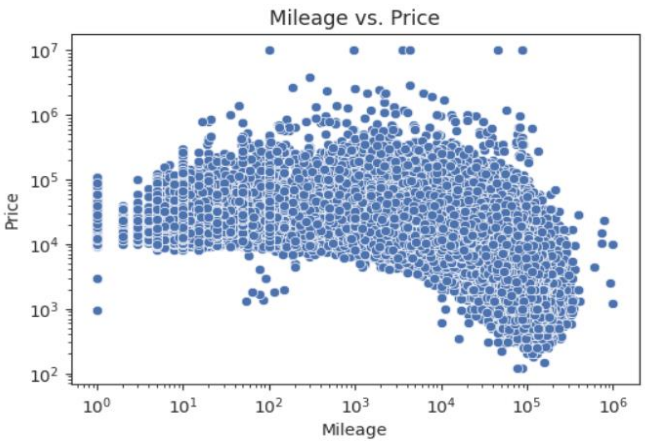
The price distribution has also a right skew. A few cars are significantly more expensive than most vehicles. This skewness is typical in vehicle datasets, reflecting a wide range of vehicle types and conditions.

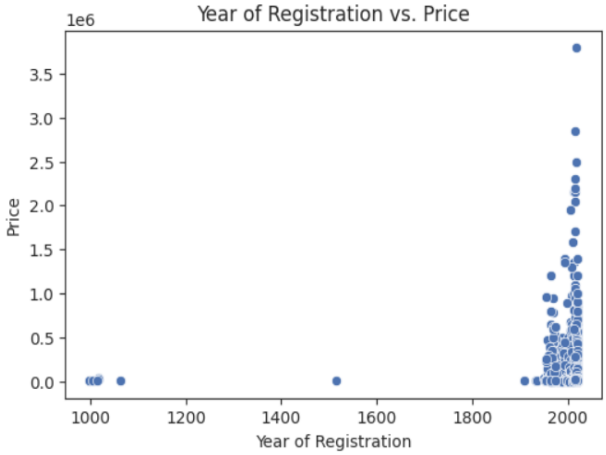
1.2. Analysis of Predictive Power of Features:

Relationships between features are shown by the correlation matrix and related scatter plots.



- Newer cars are positively correlated with higher prices, according to the price scatter plot and Year_of_registration, however data anomalies imply that data cleaning is required.
- A weak positive correlation between price and registration year, a weak negative correlation with mileage, and a substantial negative correlation between mileage and registration year are all visible in the correlation matrix heatmap.





- Mileage vs. price scatter plot shows price decreases when mileage increases, indicating predictive power but may not be linear.
- Mileage and the year of registration are predictive to some extent. However, more data cleaning, analysis, and potentially the acquisition of new descriptive variables is necessary to create a strong prediction model.
- However, there is little association between crossover_car_and_van and public_reference, suggesting that their predictive value is limited.

2. Data Processing for Machine Learning

Data Cleaning:

- **Identify and handle Missing Values:**
  The given image shows that dataset has missing values in several columns.

```
[16] df_car.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
```

- Like in mileage there is only 127 missing values some columns like price standard_make has no missing values but some have huge amount of it like reg_code, year_of_reg.
- I filled these values by median in case of mileage and for some categorical features I filled it with most frequent category.
- Standard_color, Body_type and Fuel_type is filled by White, SUV and Diesel respectively as these are their most frequent categories shown in the image below.

```
df_car[["standard_colour", "body_type", "fuel_type"]].isnull().sum()

standard_colour    5378
body_type           837
fuel_type           601
dtype: int64
```

```
df_car["standard_colour"] =df_car["standard_colour"].fillna('White')
df_car[["body_type","fuel_type"]]=df_car[["body_type","fuel_type"]].fillna({'body_type': 'SUV', 'fuel_type': 'Diesel'})
df_car[["standard_colour","body_type","fuel_type"]].isnull().sum()

standard_colour    0
body_type          0
fuel_type          0
dtype: int64
```

As there are 33311 missing values in year_of_registration from which missing value of NEW Car is filled by 2021 assuming it is recent year while rest are filled by most frequent one (2016), shown in image below.

```
[194] df_car["year_of_registration"].isnull().sum()
```

```
33311
```

```
#Fill the NEW cars with 2021
df_car.loc[df_car['vehicle_condition'] == 'NEW', 'year_of_registration'] = 2021
df_car["year_of_registration"].isnull().sum()
```

```
2062
```

```
[196] #and rest are filled by 2016
df_car["year_of_registration"]=df_car["year_of_registration"].fillna(2016).astype(int)
df_car["year_of_registration"].isnull().sum()
```

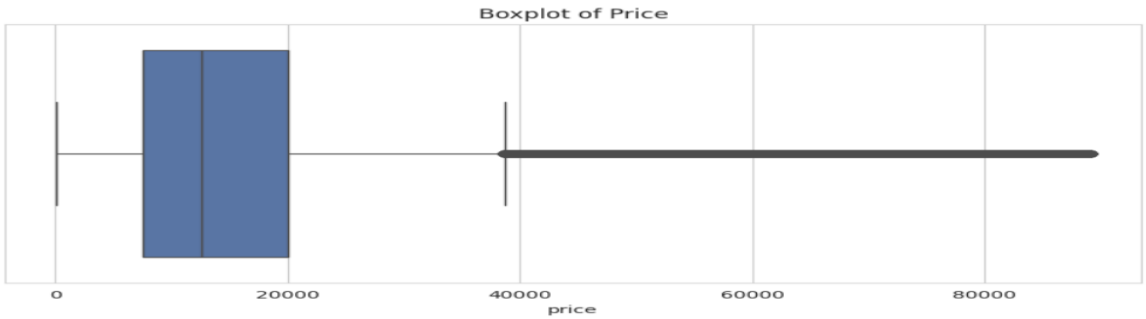
```
0
```

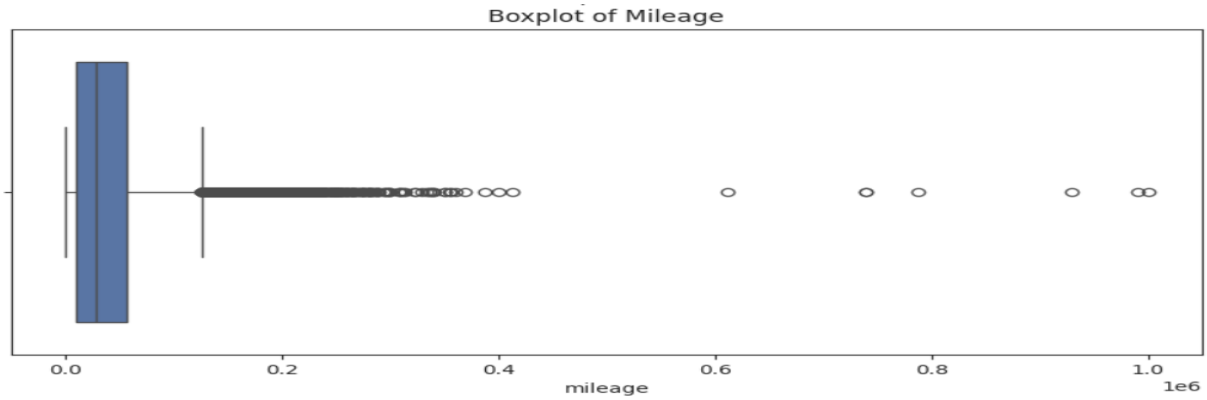
Also, dtype for year_of_registration is changed from float to integer.

Detect Outliers:

Some columns are checked for outliers and erroneous values such as in year of registration as some cars are registered before 1886 such as year 1073 etc, is also in dataset so I removed these values from this column. Some visualization is performed as shown below.

The given below images shows distribution of the data is right skewed, with outliers at the higher end of the price range, suggesting that most cars fall into a more reasonable price range. A tiny percentage of cars have substantially more mileage than the bulk, which has lower mileage. There are certain anomalies in the year of registration, which point to some extremely old cars. With a few earlier exceptions, most cars are grouped around more recent registration years.





By applying thresholds, the code removes extreme values (potential outliers) from the dataset. This method assumes that the highest 1% of values in both 'mileage' and 'price' are outliers and may not be representative of the typical data distribution. The 'crossover_car_and_van' feature's weak correlations suggest it may not be particularly predictive for the variables, while 'public_reference' appears to be a non-informative feature.

After handling missing values and outliers, dropping, reordering, and renaming the columns,
The cleaned dataset which is prepared is shown as,

```
[43] df.head(5)
```

	mileage	colour	make	model	condition	reg_year	body_type	fuel_type	price
0	0.0	Grey	Volvo	XC90	NEW	2021	SUV	Petrol Plug-in Hybrid	73970
1	108230.0	Blue	Jaguar	XF	USED	2011	Saloon	Diesel	7000
2	7800.0	Grey	SKODA	Yeti	USED	2017	SUV	Petrol	14000

Encoding:

Onehotencoding:

categorical features are encoded using 2 techniques onehotencoding for the ones having low categories and target encoding for high cardinality.
Onehotencoding is applying on 2 columns condition and fuel_type because of low cardinality which is successfully done and merged with whole dataset. The encoded columns are shown as:

condition_NEW	condition_USED	fuel_type_Bi Fuel	fuel_type_Diesel	fuel_type_Diesel Hybrid	fuel_type_Diesel Plug-in Hybrid	fuel_type_Electric	fuel_type_Petrol	fuel_type_I
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	

Splitting and target encoding:

```
#target-encoding
target_encoder = ce.TargetEncoder()
X_train[cat_feat_target] = target_encoder.fit_transform(X_train[cat_feat_target], y_train)
X_val[cat_feat_target] = target_encoder.transform(X_val[cat_feat_target])
X_test[cat_feat_target] = target_encoder.transform(X_test[cat_feat_target])
```

```
X_train.head(5)
```

	mileage	colour	make	model	reg_year	body_type	condition_NEW	condition_USED	fuel_type_Bi Fuel	fuel_type_Diesel	fuel_type_Diesel Hybrid	fuel_type_Diesel Plug-in Hybrid	f
287231	23000.0	15696.879893	8442.786976	5680.556164	2015	11157.328465	0.0	1.0	0.0	1.0	0.0	0.0	
93176	10.0	16221.458665	25986.738326	16414.022121	2021	18730.088305	1.0	0.0	0.0	0.0	0.0	0.0	
206882	22458.0	17163.517589	9979.053959	12328.932546	2017	21772.110212	0.0	1.0	0.0	1.0	0.0	0.0	

Splitting of dataset into test train and validation set is done before target encoding as encoding after the split, ensures pervention from data Leakage.After splitting target encoding is performed as shown above figure.

Scaling:

Scaling is performed using minmax scaler.

```
X_train.head(2)
```

	mileage	colour	make	model	reg_year	body_type	condition_NEW	condition_USED	fuel_type_Bi Fuel	fuel_type_Diesel	fuel_type_Diese Hybri
287231	0.161981	0.722902	0.080387	0.060601	0.946429	0.032274	0.0	1.0	0.0	1.0	0.
93176	0.000070	0.774402	0.344734	0.204708	1.000000	0.373389	1.0	0.0	0.0	0.0	0.

Model Building:

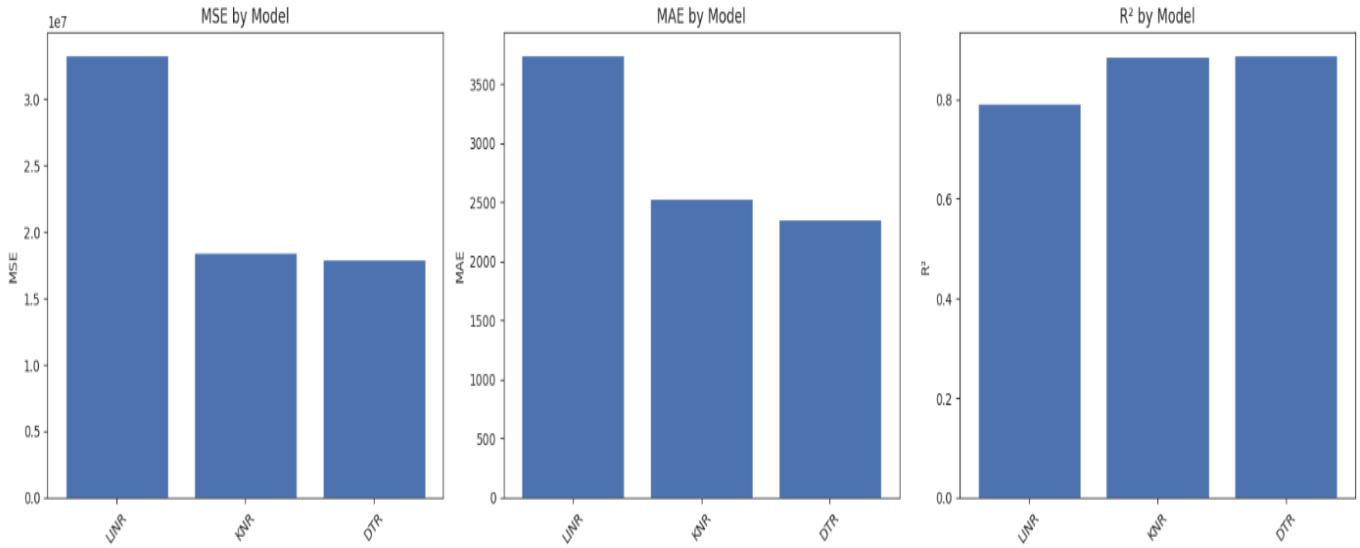
For model building KNN, Decision Tree and linear regression is used.

Model Instantiation and Configuration:

Firstly,performed Cross validation to assess the model performance which shows results as

```
LINR:
  MSE: 33326066.779   MAE: 3750.516   R2: 0.793

KNR:
  MSE: 18431490.093   MAE: 2531.005   R2: 0.885

DTR:
  MSE: 17976491.782   MAE: 2363.474   R2: 0.89
```

With the lowest MSE and MAE and a high R2 value, the Decision Tree Regression model performs better than the other two measures, suggesting that it is appropriate for the data. In terms of MSE and MAE, the k-Nearest Neighbours model performs better than linear regression; nonetheless, it is nearly similar in R2. The lowest R2 and highest error metrics indicate that linear regression may not be a superior fit for the complexity of the data.bar plots shows their comparison more precisely as:



DecisionTreeRegressor is initialized with a maximum depth of 3 and a **random_state** of 42 for reproducibility. The model is then fitted to the training data (**X_train** and **y_train**). A parameter grid is defined for the gridsearch.

GridSearchCV uses 5-fold cross-validation to test every possible combination of parameters in a grid. The model evaluation and comparison process use the negative mean squared error, with lower values desirable. The return_train_score option yields training fold scores for overfitting analysis. The grid search results are displayed in the top 5 results of a dataframe sorted by rank_test_score as:

_depth	param_min_samples_leaf	param_min_samples_split	mean_train_score	std_train_score	mean_test_score	std_test_score	rank_test_score
20	4	10	-5.629120e+06	18833.011116	-1.081020e+07	91455.428825	1
20	4	2	-5.444207e+06	19464.064845	-1.093848e+07	73454.508399	2
20	4	5	-5.444208e+06	19464.427515	-1.094029e+07	61727.304318	3
20	2	10	-5.024811e+06	33069.089148	-1.106686e+07	63944.202326	4
20	1	10	-4.629033e+06	49587.854201	-1.136921e+07	185091.059523	5

```
Grid_result.best_estimator_
```

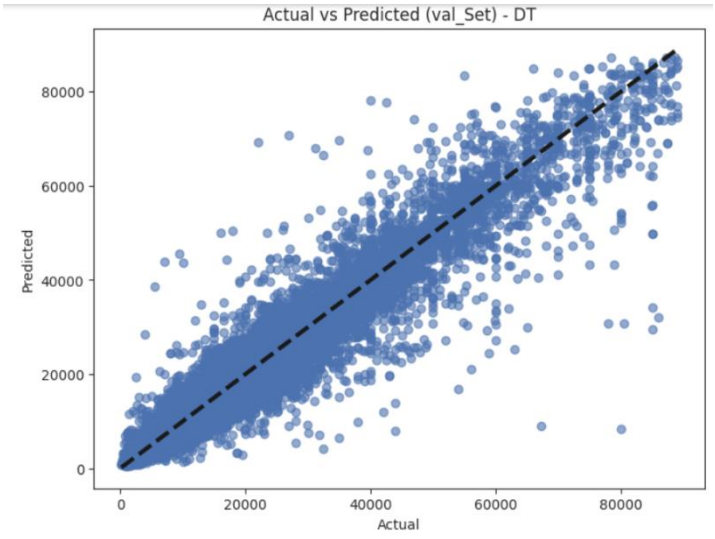
```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=20, min_samples_leaf=4, min_samples_split=10)
```

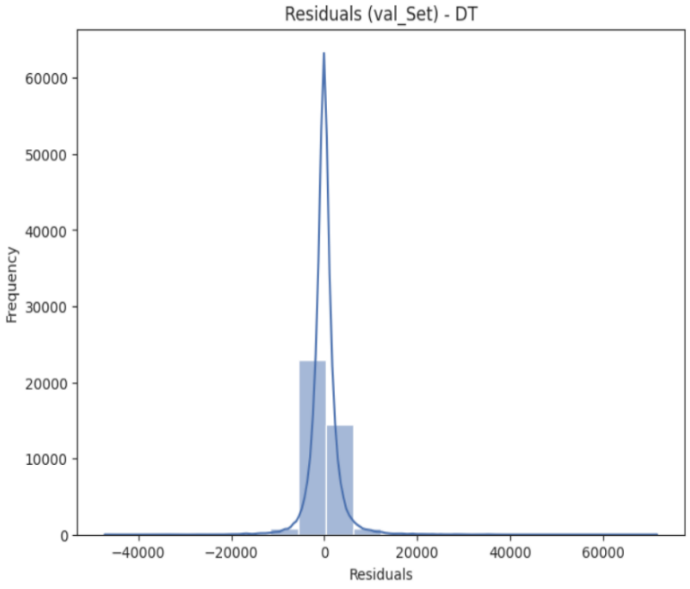
Validation-Set Evaluation

The model instance with the best performance throughout the grid search, known as the best_estimator_, already had its hyperparameters tuned to the most optimal value. Which makes predictions on validation set. Plot shows a good fit suggested by most of the data points clustering around the diagonal. There are, however, certain spots that deviate from the line, suggesting some predictive error.

Scores: The R-squared (R^2) score, Mean Absolute Error (MAE), and Mean Squared Error (MSE) are the metrics we utilised to assess performance. The following are the outcomes:

```
MSE for Decision Tree(val_set): 10445780.771
MAE for Decision Tree(val_set): 1830.606
R2 for Decision Tree(val_set): 0.935
```





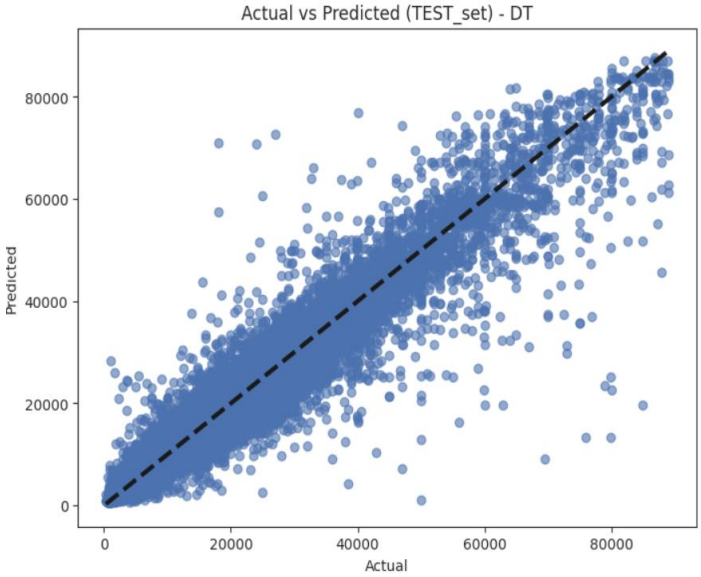
The distribution of residuals for Decision Tree model on the validation set is shown by the histogram. The distribution of the residuals reveals both the model's strong points and its room for development. Resolving the skewness and outliers may improve the predicted accuracy of the model. These above info shows overall performance of model is good.

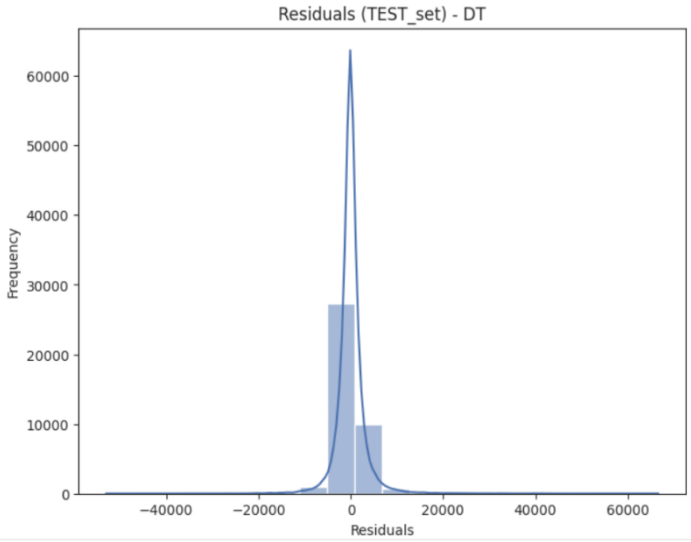
Evaluation on Test Set

Finally, moved from the validation to the test set, assessing how well model works with fresh, unseen data to ensure its accuracy and validate its usefulness. The Mean Squared Error (MSE) is 10,983,845.244, indicating variances between predicted and actual vehicle prices. The Mean Absolute Error (MAE) was 1,840.054, providing insights into average prediction errors. Model's R-squared score of 0.933, indicating a strong fit to data. Results from the test set's evaluation of the Decision Tree model shows that it is in line with those from the validation set

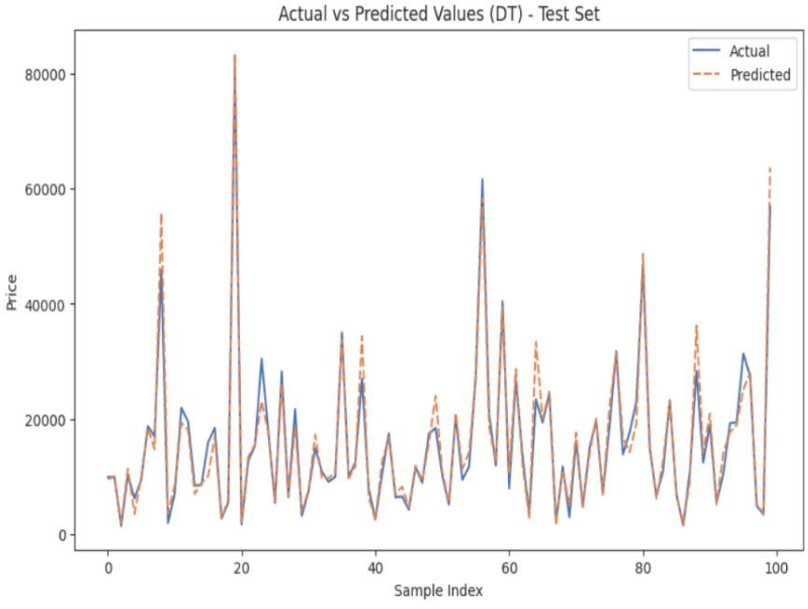
```
best_DT=Grid_result.best_estimator_
#now Use the best model to make predictions on the test set
y_pred_DT = best_DT.predict(X_test)

#Evaluate the performance
mae_dt=mean_absolute_error(y_test, y_pred_DT)
R2_dt=r2_score(y_test, y_pred_DT)
mse_dt=mean_squared_error(y_test, y_pred_DT)
print("MSE for Decision Tree(test_set):",round(mse_dt, 3))
print("MAE for Decision Tree(test_set):",round(mae_dt, 3))
print("R2 for Decision Tree(test_set):",round(R2_dt, 3))
```

```
MSE for Decision Tree(test_set): 10983845.244
MAE for Decision Tree(test_set): 1840.054
R2 for Decision Tree(test_set): 0.933
```





This scatter plot of Decision Tree model actual vs. anticipated values shows that the model predicts the target variable well. The model's accuracy, especially for higher-value predictions, is variable due to the dispersion of points around the line of best fit."



	Actual	Predicted
309308	11985	6873.714286
110487	11915	11248.714286
59106	25699	23683.333333
264255	27888	24969.400000
105277	36990	29191.250000
233892	13995	14618.142857
1231	14950	15084.698000
311296	17949	18164.411765
117604	15000	25885.600000
338580	11495	12127.333333

The test set's residual plot has a pattern that is comparable to the validation set's, with a concentration of errors near zero and a few significant outliers. The Decision Tree model's predictions are shown in the graph and table, with some deviations, showing how well they match actual values across test set samples.

KNN Regression:

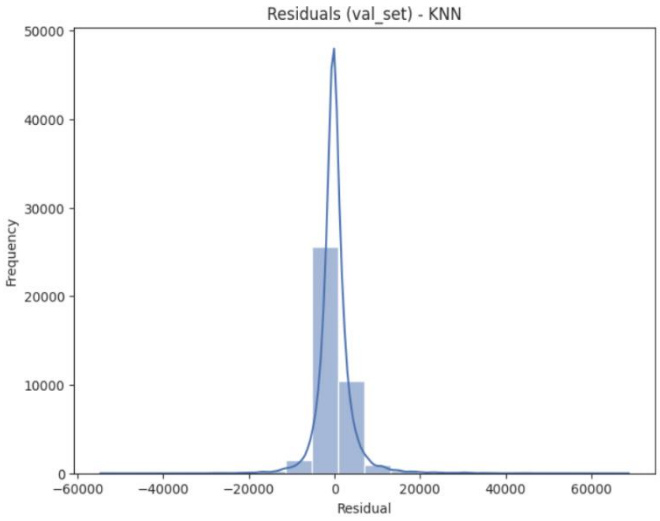
A manageable size sampling of 30% of the data as our training set is done for effective model training. In its initial configuration, the kNN model had 15 neighbours. Next, using a grid search across a predetermined set of neighbour values 3, 5, 7, 10, and 15, performed hyperparameter tuning. 5-fold cross-validation was used to test this optimisation. The grid search results showed that the best performance was produced by a kNN regressor with 5 neighbours, as indicated by the lowest mean test score.The optimal estimator was determined to be this model.

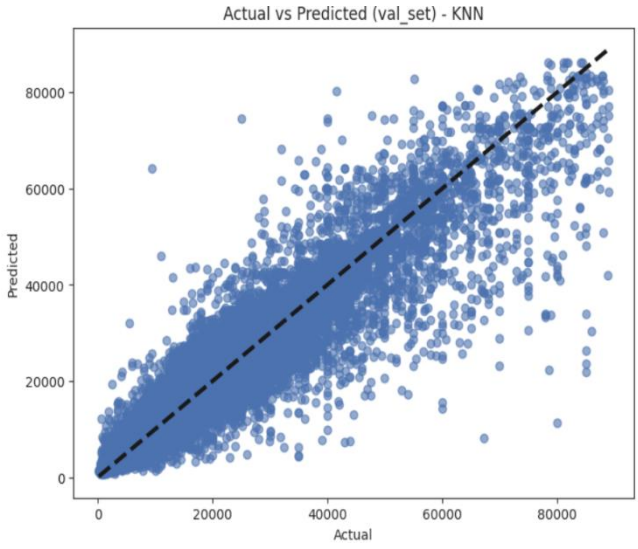
Validation-Set:

```
knn_pred= best_knn.predict(X_val)
mae=mean_absolute_error(y_val,knn_pred)
R2=r2_score(y_val,knn_pred)
mse=mean_squared_error(y_val, knn_pred)
print("MSE for KNN(val_set):",mse)
print("MAE for KNN(val_set):",mae)
print("R2 for KNN(val_set):",R2)
```

Validated performance of the best (kNN) model from the hyperparameter tuning phase on the validation set. After determining the optimal number of neighbours, the model produced scores (R^2) value of 0.8868, a Mean Absolute Error (MAE) of 2,469.17, and a Mean Squared Error (MSE) of 18,296,727.23.

```
MSE for KNN(val_set): 18296727.23120398
MAE for KNN(val_set): 2469.168836642462
R2 for KNN(val_set): 0.8868374726966302
```



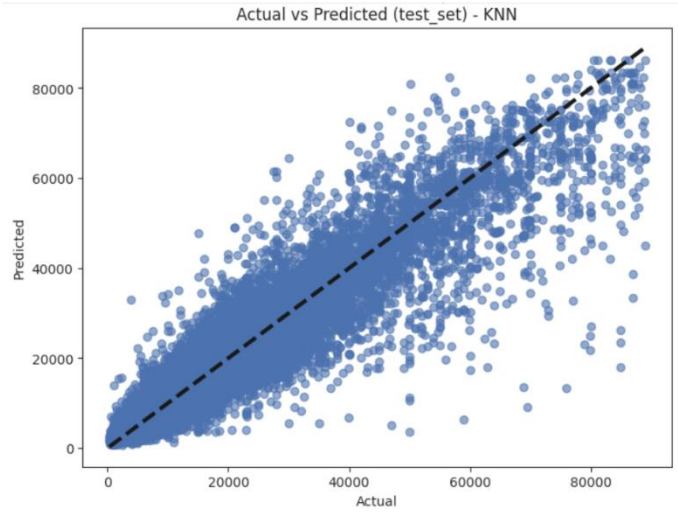


The quantity of errors indicates outliers or variability that the model might not be capturing, but these metrics show the model's ability to predict with a high degree of variance explanation.

Test-Set:

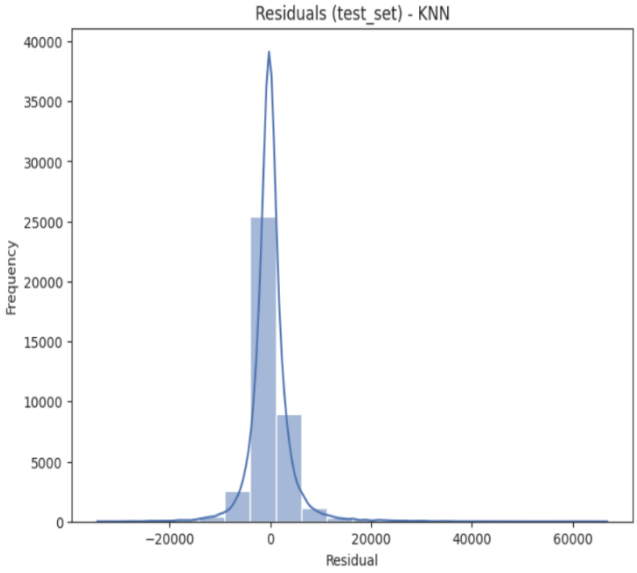
When the k-Nearest Neighbours (kNN) model was evaluated on the test set, the results showed a R^2 of 0.8879, an MAE of 2,470.04, and an MSE of 18,230,191.43. These measures, which show the model's consistent performance across several data subsets, closely resemble those found in the validation set.Furthermore, a mean cross-validation (CV) score of 0.9111 was obtained by applying the optimal kNN model to cross-validate on the training set. This high score further

validates the robustness of the model by indicating that it is stable and performs well across different folds of the training data



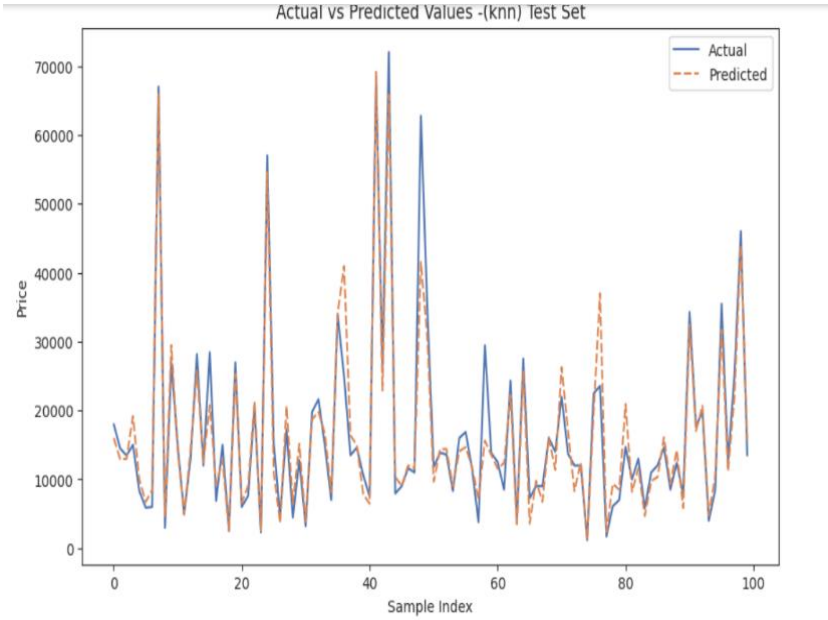

```
knn_Pred= best_knn.predict(X_test)
mae_knn=mean_absolute_error(y_test,knn_Pred)
R2_knn=r2_score(y_test,knn_Pred)
mse_knn=mean_squared_error(y_test, knn_Pred)
print("MSE for KNN(test_set):",mse_knn)
print("MAE for KNN(test_set):",mae_knn)
print("R2 for KNN(test_set):",R2_knn)
```

```
MSE for KNN(test_set): 18230191.431965675
MAE for KNN(test_set): 2470.0488346112834
R2 for KNN(test_set): 0.8879923790125539
```



.

- The residual plot for the test set shows a concentration of values around zero, indicating that the kNN model makes many predictions near to the actual values, with some noticeable exceptions resulting in bigger residuals.
- The 'Actual vs Predicted Values' plot for the test set compares predicted and actual values The table shows a subset of real and predicted values to compare the model's predictions to the true data. A granular view can reveal regions where the model needs calibration.



	Actual	Predicted
326801	10499	12514.8
261058	3495	3591.2
353729	6946	5900.8
136008	4295	3674.0
117979	8295	8456.8
296249	25295	20762.2
314182	4500	5564.0
383052	12749	12789.4
373238	13699	10136.0
42359	39825	36380.4

Training and test scores are consistent, indicating that the model generalised well without overfitting to training data.

Linear Regression Model:

A Linear Regression model was instantiated and trained on training set.

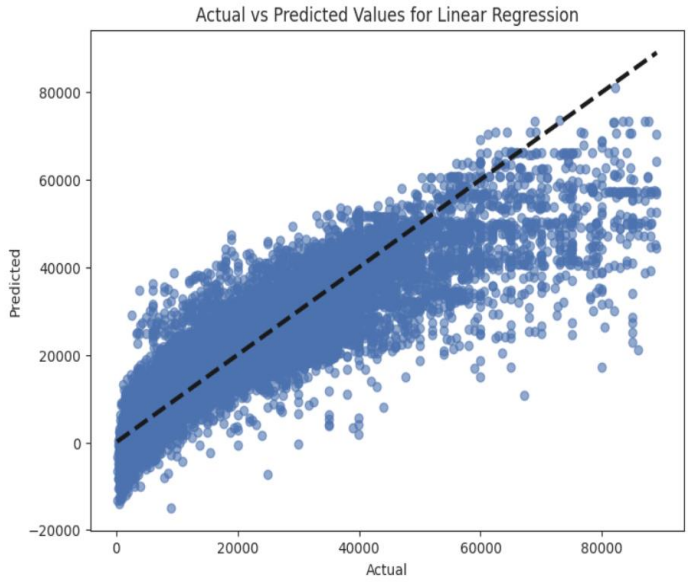
Validation-set

The model was validated post-training to assess performance.

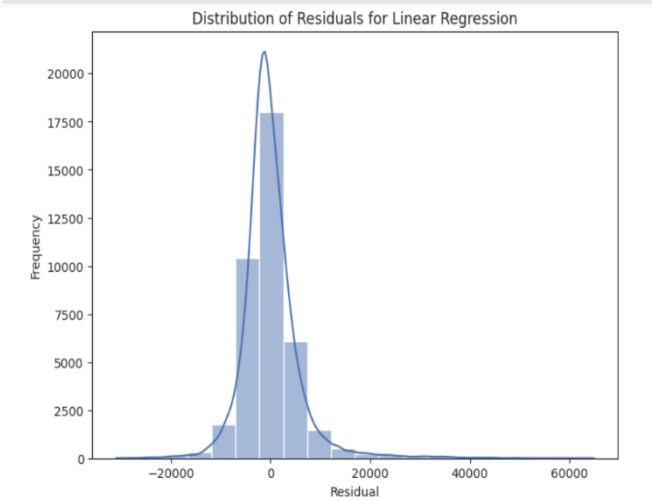
```
# Validate the model
y_val_pred = linear_model.predict(X_val)
mae=mean_absolute_error(y_val, y_val_pred)
R2=r2_score(y_val, y_val_pred)
mse_LR=mean_squared_error(y_val, y_val_pred)
print("MSE for LR(Val_set):",mse)
print("MAE for LR(VAL_set):",mae)
print("R2 for LR(VAL_set):",R2)
```

```
MSE for LR(Val_set): 18296727.23120398
MAE for LR(VAL_set): 3803.526320908129
R2 for LR(VAL_set): 0.7892767973562145
```

The model's validation set predictions had an MSE of 18,296,727.23, the average of the prediction errors' squares This performance evaluation shows that the Linear Regression model captures a significant percentage of the dataset's variance, although the MSE and MAE indicate that the model's prediction accuracy might be improved.



The scatter plot compares predicted and actual values. This line's density of points implies the model's predictions on validation set are generally accurate, but higher values show significant volatility.



The residuals histogram, which compares predicted and actual values, is centred around zero, which is good. The data shows a skew with a right tail, indicating that the model underestimates some values. The narrow peak reflects minimal residuals for most predictions, but the lengthy tail reveals outliers when the model's predictions are less accurate.

Test-Set Evaluation:

Evaluation of the Linear Regression model on the test set showed metrics scores. These results indicate that the model explains a considerable percentage of the target variable's volatility, yet prediction errors are significant. The model performed consistently across data subsets in cross-validation on the whole training set, with scores ranging from 0.7921 to 0.7991. The training and test set scores' close alignment supports the model's generalisation.

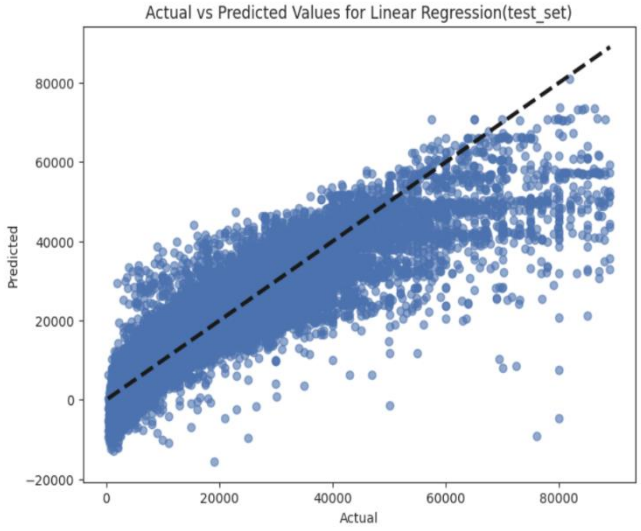
```
# Perform cross-validation on the entire training set using the best model
cv_scores_lin = cross_val_score(linear_model, X_train, y_train)
cv_scores_lin
```

```
array([0.79213345, 0.79178894, 0.79146461, 0.79322357, 0.79099622])
```

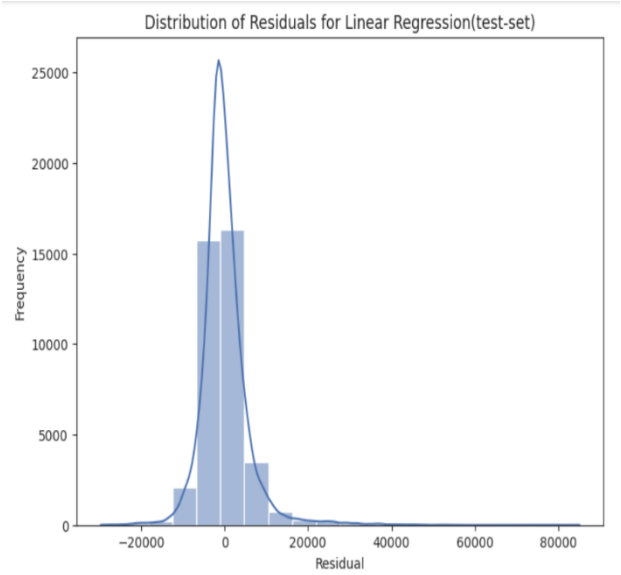
```
]7] # test the model
    y_LR_pred = linear_model.predict(X_test)
    mae_LR=mean_absolute_error(y_test, y_LR_pred)
    R2_LR=r2_score(y_test, y_LR_pred)
    mse_LR=mean_squared_error(y_test, y_LR_pred)
    print("MSE for LR(test_set):",mse_LR)
    print("MAE for LR(test_set):",mae_LR)
    print("R2 for LR(test_set):",R2_LR)
```

```
MSE for LR(test_set): 34320268.48115836
MAE for LR(test_set): 3790.1502877712487
R2 for LR(test_set): 0.7891337763198416
```

A 34,320,268 MSE implies high model prediction average squared errors.
The MAE is 3,790.15, indicating that model predictions regularly differ by this much.
The R^2 Score of 0.7891 indicates that the model explains 78.91% of the dependent variable's variability. Since the model cannot explain for 21% of variability, this score is high but might be improved. The MSE and MAE values demonstrate that the Linear Regression model is predictive but might be improved to reduce prediction errors.



The points are scattered around the identity line (dashed line), where perfect predictions lie. The spread shows that model accuracy decreases with actual values.



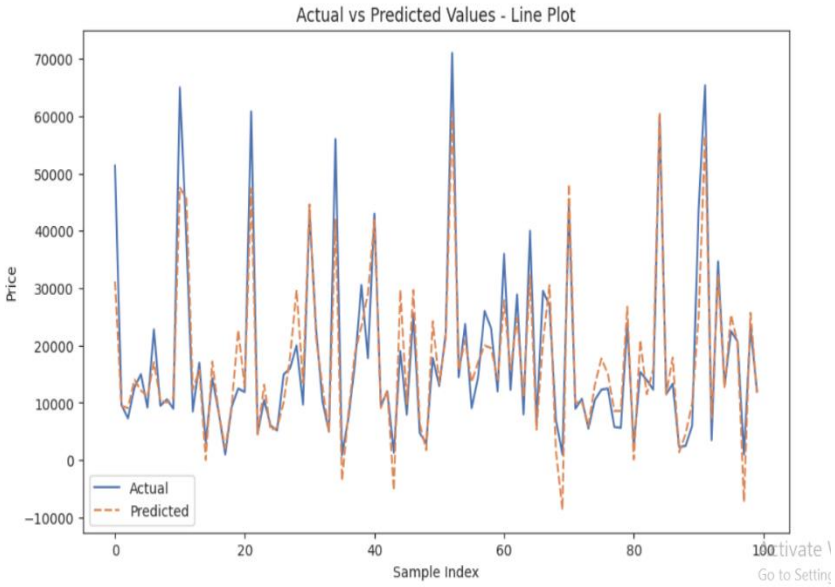
The residuals are centred around zero but have right skewness, suggesting the model underestimates. The residuals near 0 indicate that the model works well for many predictions, whereas the tails indicate severe prediction mistakes.

	Actual	Predicted
247800	11450	11396.197410
306198	14000	15772.632694
336746	17000	17476.716266
153381	4999	5822.583474
45404	11822	10953.511348
201847	5999	7908.463841
47126	25495	21060.026024
31200	13500	9731.976359
90593	9980	9454.565733
203250	20656	19087.103418

A table of the actual and anticipated values reveals where the model's predictions differ from the actual values, with some being near and others greater.

This plot shows that the model follows the trend of the actual values, although there are peaks and troughs where it predicts differently.

The Linear Regression model captures the broad trend in the data but struggles with accurate predictions across the dataset, especially at higher actual values, as seen by the residual distribution and line plot divergence.



Evaluation of Different Models:

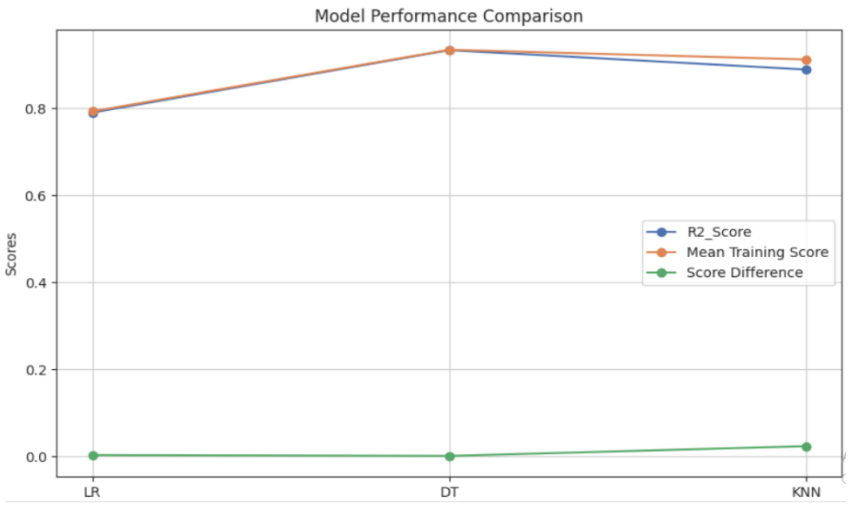
The investigation examined Linear Regression (LR), Decision Tree (DT), and k-Nearest Neighbours models. The R-squared (R^2) score on the test set and the mean training score from cross-validation were used to evaluate each model's performance. A higher R^2 score indicates a better fit between the model's predictions and actual data.

```
model_scores = {
    'Model': ['LR','DT', 'KNN'],
    'R2_Score': [R2_LR,R2_dt, R2_knn],
    'Mean Training Score': [cv_scores_lin.mean(), cv_scores_dt.mean(), cv_scores_knn.mean()],
    'Score Difference': [cv_scores_lin.mean() - R2_LR,
                         cv_scores_dt.mean() - R2_dt,
                         cv_scores_knn.mean() -R2_knn,
                        ]
}


scores_df = pd.DataFrame(model_scores)
print(scores_df)
```

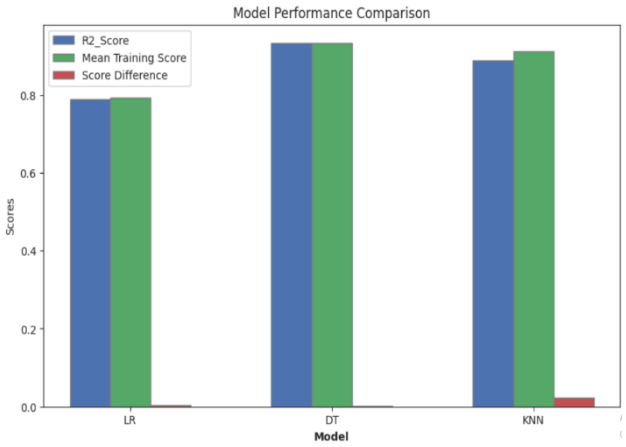
```
  Model  R2_Score  Mean Training Score  Score Difference
0    LR  0.789134             0.791921          0.002788
1    DT  0.932514             0.933322          0.000808
2   KNN  0.887992             0.911134          0.023142
```

The linear regression model having a slight score difference of 0.0028, indicating consistency across training and test sets. A Decision Tree model showed similar performance during cross-validation and on the test set, with a slight difference of 0. 0008.The k-Nearest Neighbours (KNN) model resulting in a 0.0231 score difference.

This disparity is significantly bigger than for the Linear Regression and Decision Tree models, demonstrating a gap between training performance and test data generalisation. Out of Linear Regression, Decision Tree, and KNN, Decision Tree performed best. It has the best R^2 score, showing a better fit to test data, and the smallest difference between training and testing results, indicating robust and consistent prediction power. The graph shows that DT has the highest R^2 Score, followed by KNN and LR.



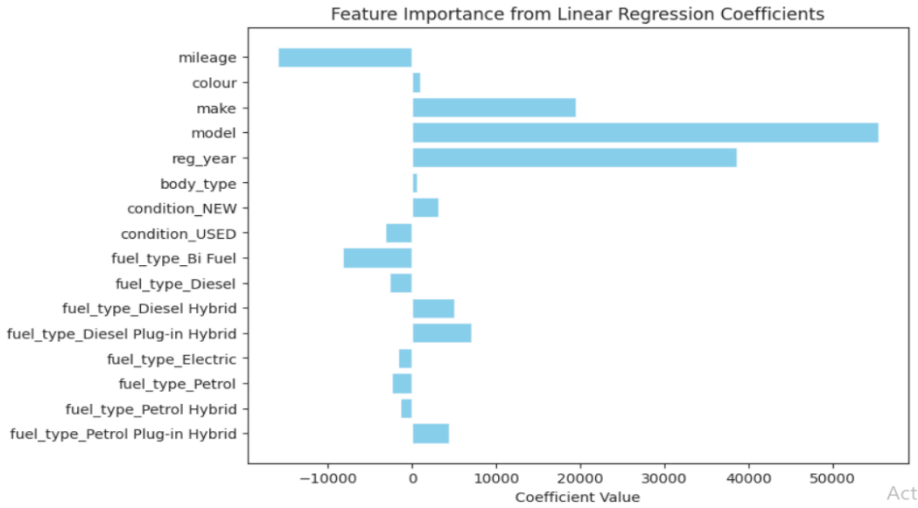
The Mean Training Score for DT is higher than KNN and LR. The Score Difference statistic compares cross-validation and test scores, showing low scores, with LR larger than DT and KNN. Bar Chart also shows comparison between Decision Tree KNN and Linear Regression models, and they have high R^2 scores, indicating good fit.



Feature Importance:

Linear Regression:

The 'model' feature is the most influential predictor of the target variable, with a rise in 'model' indicating an increase. 'Reg_year' is also positively affecting the target variable, while 'mileage' has a negative coefficient. Other attributes like 'condition_NEW' and 'condition_USED' also impact the target variable.




```python
# Get the coefficients
coefficients = linear_model.coef_
features = X_train.columns

for feature_name, coef in zip(features, coefficients):
    print(f"Coef for {feature_name}: {coef.round(3)}")
```

```
Coef for mileage: -15974.626
Coef for colour: 942.151
Coef for make: 19489.515
Coef for model: 55350.62
Coef for reg_year: 38609.965
Coef for body_type: 654.201
Coef for condition_NEW: 3170.505
Coef for condition_USED: -3170.505
Coef for fuel_type_Bi Fuel: -8306.11
Coef for fuel_type_Diesel: -2618.313
Coef for fuel_type_Diesel Hybrid: 5075.946
Coef for fuel_type_Diesel Plug-in Hybrid: 7028.039
Coef for fuel_type_Electric: -1683.642
Coef for fuel_type_Petrol: -2443.894
Coef for fuel_type_Petrol Hybrid: -1428.7
Coef for fuel_type_Petrol Plug-in Hybrid: 4376.674
```

Decision Tree Regression:

Decision Tree's most important predictors are 'model', 'mileage', and 'Reg_year'. Make, colour, and body_type is moderately important. Fuel types like 'Bi Fuel', 'Diesel Hybrid', 'Electric', and 'Petrol' matter nothing to the model's decision-making

```python
#feature importance from the Decision Tree
importances = best_DT.feature_importances_
features = X_train.columns
for feature_name, imp in zip(features, importances):
    print(f"{feature_name}: {imp.round(4)}")
```

```
mileage: 0.1079
colour: 0.0037
make: 0.0497
model: 0.5995
reg_year: 0.2114
body_type: 0.0107
condition_NEW: 0.0008
condition_USED: 0.0021
fuel_type_Bi Fuel: 0.0
fuel_type_Diesel: 0.0049
fuel_type_Diesel Hybrid: 0.0002
fuel_type_Diesel Plug-in Hybrid: 0.0
fuel_type_Electric: 0.0013
fuel_type_Petrol: 0.0046
fuel_type_Petrol Hybrid: 0.0006
fuel_type_Petrol Plug-in Hybrid: 0.0025
```