Price Prediction using Autotrader Dataset

1. Data Processing for Machine Learning:

For data processing, visualization, and manipulation, I started off importing some important packages such as pandas, NumPy, scikit-learn, seaborn, matplotlib and many others. Then loading the dataset, I took sample of 30000 values as it's efficient and I stored it in a variable called 'adv'.

For understanding the given Autotrader UK car dataset, I used some common functions as info(), columns, describe(), shape etc. Proceeding to understand the data more I plotted some boxplots using seaborn to see the distribution of some features such as mileage, price etc. over the selected data frame.

- **Dealing with missing values:**

To find the null values in data frame 'adv' I have used isnull() and sum() functions to get the total number of null values in each of columns in data frame. I have compared the snapshots of before and after removing outliers values below at the end of this point.

I have extracted one new feature called 'missing_year_with_not_missing_reg' from the features 'year_of_registration' and 'reg_code'. The approach to this one is, if we find the years with null values but have registration code then we can get the year of registration for those years using following steps:

```
missing_year_with_not_missing_reg = adv.loc[adv['year_of_registration'].isna() & ~adv['reg_code'].isna(), :]
```

After this, using the UK registration codes, I have assigned the respective years where the year of registration was missing. For example, if the reg_code is 16, then we will assign the year of registration as 2016 and so on. To do this, I used following code snippet:

```
# Function to convert UK registration code to year
def registration_code_to_year(reg_code):
    if 0 <= int(reg_code) <= 50:
        return "20" + reg_code
    elif 51 <= int(reg_code) <= 99:
        return "20" + str(int(reg_code) - 50)
    else:
        return None
```

Then to fill the null values for features mileage, standard_colour, year_of_registration, body_type, and fuel_type I have used fillna() method to fill null values with mean and mode values according to their data type. If it's numerical we will apply mean() method to that feature and mode() to the categorical features.

```
adv['mileage']=adv['mileage'].fillna(adv['mileage'].mean())
adv['body_type']=adv['body_type'].fillna(adv['body_type'].mode()[0])
adv['fuel_type']=adv['fuel_type'].fillna(adv['fuel_type'].mode()[0])
adv['year_of_registration']=adv['year_of_registration'].fillna(adv['year_of_registration'].mode()[0])
adv['standard_colour']=adv['standard_colour'].fillna(adv['standard_colour'].mode()[0])
```

- **Dealing with Outliers:**

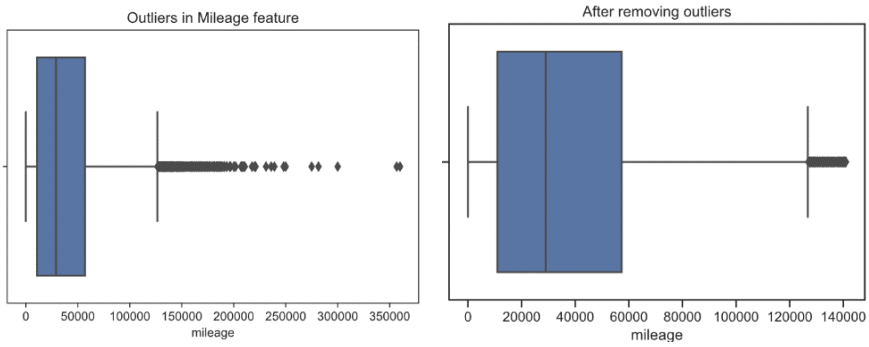
To deal with the outliers in mileage and price feature, I set the upper limit and lower limit for this. The mileage column's mean is subtracted from three times its standard deviation to determine the lower limit, which has a minimum value of 0 to prevent negative results. And for upper limit I took the 99the percentile using quantile() method. Then I performed following steps:

```
# Calculate the upper and lower limits for outlier detection
upper_limit = adv.mileage.quantile(0.99)
lower_limit = max(adv.mileage.mean() - 3 * adv.mileage.std(), 0)

# Find the outliers
outliers = adv[(adv.mileage > upper_limit) | (adv.mileage < lower_limit)]

# Cap the upper and lower limits at the desired values for the mileage column
adv['mileage'] = adv['mileage'].apply(lambda x: min(max(x, 0), upper_limit))
```

I applied lambda function on mileage column to cap the values between 0 and upper limit. This makes sure that any mileage values that are outliers are replaced with their corresponding limit values. We can compare the plots of before and after removal of outliers:



So, as we can see, this code has done well job on this feature to remove outliers. The minimum mileage is 0 and maximum mileage is around 130000-140000. Before removing outliers, the maximum mileage was 360000.

This method helps in producing good results, so I applied the same feature 'Price'. Let's compare the before and after snapshots of data frame when outliers and null values are dealt.

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

```
public_reference          0
mileage                   7
reg_code               2351
standard_colour         393
standard_make             0
standard_model            0
vehicle_condition         0
year_of_registration   2454
price                     0
body_type                56
crossover_car_and_van     0
fuel_type                46
dtype: int64
```

1.Before dealing with outliers and null values

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

```
public_reference          0
mileage                   0
reg_code               2351
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
Price Category            0
dtype: int64
```

2.After dealing with outliers and null values

I have not done anything for 'reg_code' as I am going to drop this column as it is not related to the features that I will be using for model building and it does not affect any further analysis/task.

- **Encoding:**

For categorical encoding, I have used One Hot Encoding as it shows every unique category in a categorical variable as a separate binary column. I did not use Target Ordinal Encoding as there is some chance of data leakage which might affect model performance and its predictive abilities.

The columns that I took for encoding are 'standard_make' 'vehicle_condition', 'fuel_type', and 'body_type'. Using get_dummies function we got the following result:

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

	mileage	year_of_registration	price	Price Category	age_of_car	standard_make_Audi	standard_make_BMW	standard_make_Citroen	standard_make_DS AUTOMOBILES	sta
0	0.00	2017	73970	6	3	0	0	0	0	
1	108230.00	2011	7000	1	9	0	0	0	0	

2 rows × 56 columns

I have added some new columns named ‘Price Category’ and ‘age_of_car’ which I’ll explain in feature engineering section.

Then I performed splitting of data into predicted and target data and obtained train, test folds using train_test_split() function. I have performed the encoding, scaling while creating a pipeline and done it separately as well just to have better understanding.

- **Scaling:**

I am using StandardScaler to standardize our numeric features of adv data frame. Scaling is used for achieving better model performance and interpretation. Scaling brings all the features to a similar scale, which results in stopping the dominance of features with more magnitude. Then I performed fit_transform on X_train and Y_train data set.

So, in this 1st point of Data Processing for Machine Learning, we cleaned, processed and manipulated the data to process further and now we have the train and test data which we can feed to models for analysis and predictions.

2. Feature Engineering:

We use feature engineering for creating or extracting new features from existing data to improve the performance of models. We extract relevant information from the available data and create new features that can help to identify the underlying pattern and relationships present in the data.

A) Feature Creation:

I have created new features called ‘age_of_car’ and ‘Price Category’ and added them to the adv data frame. The approach I used for extracting ‘age_of_car’ feature from year_of_registration is, I set the current year as 2020, as the given dataset has the same year as current year, then I subtracted the year_of_registration from the current year and then we have the column of year telling its age.

```
adv['current_year']=2020
```

```
adv['age_of_car']=adv['current_year']-adv['year_of_registration']
```

Then, I have added a new feature called ‘Price Category’ where the I have mentioned some price ranges and assigned a value to each range, for example, if price range is between 0 to 10000 then it’s category1, if it’s 10000 to 20000 then it’s category 2, …, and if it’s 50000 onwards then that’s category 6 being the most expensive.

```
# Define the range of prices for each class
class_ranges = [(0, 10000), (10000, 20000), (20000, 30000), (30000, 40000), (40000, 50000), (50000, float('inf'))]
# Define a function to map each price to a category
def get_price_category(price):
    for i, (lower, upper) in enumerate(class_ranges):
        if price >= lower and price < upper:
            return i+1
    return len(class_ranges)
# Apply the get_price_category function to the 'price' column to get the category
adv['Price Category'] = adv['price'].apply(get_price_category)
adv.head()
```

Next, I worked on the feature ‘standard_make’, I took the count of top 30 standard_make using valuecounts() and nlargest() functions. It resulted in returning the top 30 car brands that have the largest number of cars in adv data frame.

```
adv['standard_make'].value_counts().nlargest(30)
```

```
BMW              2828
Audi             2648
Volkswagen       2545
Vauxhall         2525
Mercedes-Benz    2419
```

This code has returned top 30 rows, we can tell that BMW has more cars than any other standard makes then we have Audi and so on, due to limited page access I have attached small snapshot.

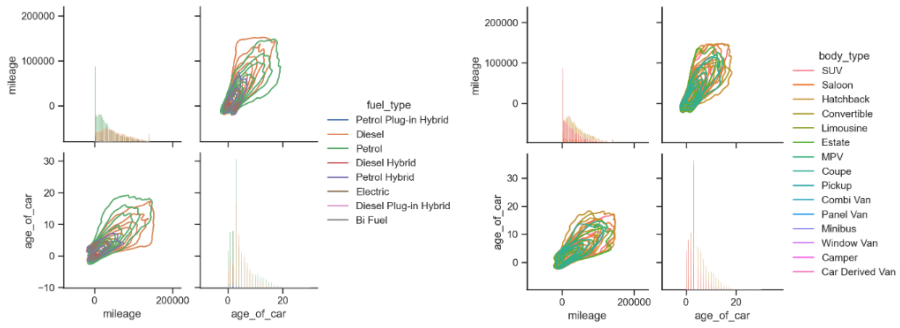
This isin() method will only keep the rows where the column value is in that top 30 values.

```
top_30 = adv['standard_make'].value_counts().nlargest(30).index
adv = adv[adv['standard_make'].isin(top_30)]
```

The adv data frame will be filtered once this code has been run to only include the rows whose values for the'standard_make' column are among the top 30 most frequent values.

B) Interaction between features:

So, these were some features that I worked on as a part of feature engineering. To visualize the relationship and interaction between these features I have plotted some graphs such as pairplot of 'mileage', 'age_of_car', 'fuel_type' and 'body_type'. The plot explains the relationship between these features:



The histogram for ‘age_of_car’ depicts the distribution of car age, the histogram for ‘mileage’ reflects the distribution of mileage values. These histograms give insights on each variable's frequency and range of values. The scatterplot shows how ‘mileage’ and ‘age_of_car’ are related considering the feature ‘fuel_type’, we can say cars using petrol and diesel have more patterns and shifts. This was for the first pairplot, same goes with the second plot where I considered ‘mileage’ and ‘age_of_car’ with respect

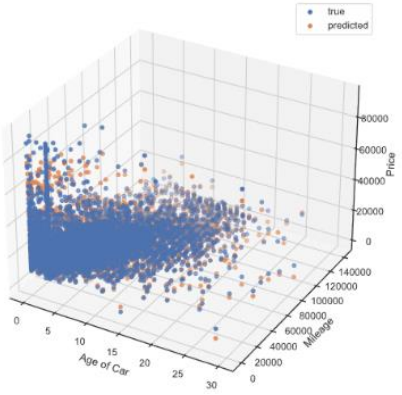
to ‘body_type’. So, this is one way to analyze the interaction between features. I have also plotted the scatter plot for the ‘mileage’ and ‘price’ considering the ‘age_of_car’, where the interaction is more understandable.

C) Polynomial Function:

Polynomial basis functions are a way to transform input features into higher-order polynomials that capture non-linear relationships with the target variable. In this case, we applied polynomial basis functions in combination with a random forest model to create a highly accurate predictive model for car prices. The preprocessed features included standardized numerical features and one-hot encoded categorical features, as well as polynomial interactions between certain features. And created a pipeline using the preprocessed transformer i.e., preprocessor and then worked with Random Forest Regressor for training and prediction. The random forest model was used to learn the relationships between the features and the target variable, and it achieved an impressive R-squared score of 0.95 on the test set, which means that it can explain 95% of the variance in the target variable using the selected features.

Visualization using 3D Scatter Plot:

The plot is very clustered between true and predicted values which explains that our model’s prediction is accurate, and it closely matches with predicted values. The tightly clustering also indicates that model has identified underlying patterns between target and predicted features.



3. Feature Selection and Dimensionality Reduction:

1)Manual Selection of features:

In feature we extract most relevant and informative features and reduce the other less relevant features to improve model performance, interpretability, computation compatibility and other many reasons. In our dataset adv, I have dropped some features like ‘crossover_car_and_van’, ‘reg_code’ and ‘public_reference’. I used ‘reg_code’ feature to fill possible null values for column ‘year_of_registration’ then I dropped it. Then the columns ‘crossover_car_and_van’ and ‘public_reference’, I decided to drop them as they are not directly connected to the feature that affect the model performance and predictions.

Also, considering the Domain knowledge factor here, we cannot retrieve any valuable information using these features, suppose if I want to know the price, mileage of any sepecific car, I cannot extract it using these features so its better option to drop them. Domain experts would think about the impact of adding/keeping less relevant features into dataset on model complexity, and its quite clear that these features do not have that much relevant information so its manageable without them.

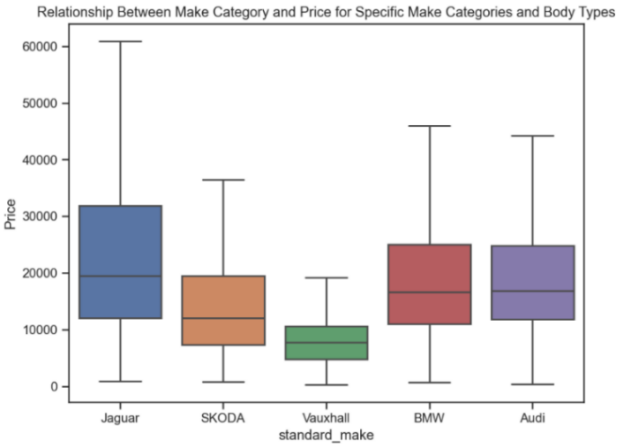
2) Automated Feature Selection using SelectKBest:

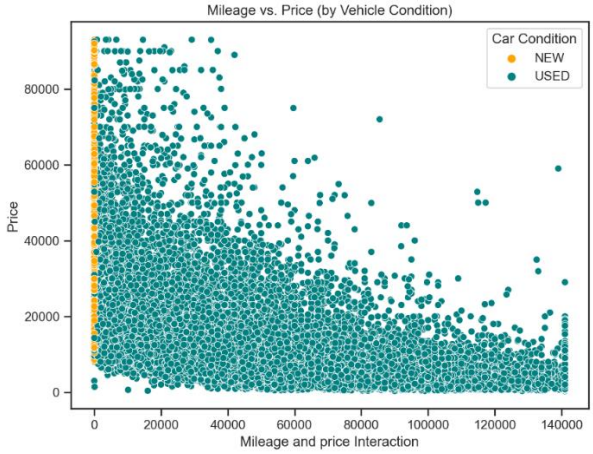
To automatically select the most useful predictors we can use SelectKBest method. It helps to pinpoint the most relevant features to the target variable and overall predictive power. I will be able to select the top K features with highest scores using specified functions such as f_classif.

f_classif is a statistical scoring function for feature selection which calculates the ANOVA F-value (Analysis of Variance F-value) between the means of two or more features. I am using this method to obtain top 10 features hence I will set the k=10, and I plotted the boxplot for the k features and sorted them for better visualization.

3) EDA:

Exploratory Data Analysis is an important step in Machine Learning which is used for analyzing and understanding the data before feeding it to the models. It identifies the outliers, helps to uncover the underlying patterns, provides insights of univariate and bivariate relationships among variables, understands the distribution of the features and so on. I have plotted some graphs to visualize this:





Due to limited page numbers I have not added all the plots of such as violin plot of price and price catergory, boxenplot price and fuel_type and some more plots visualising the distribution and relationships between them.

Dimensinality Reduction: PCA

PCAs may be useful in determining which factors contribute to the price of a vehicle. However, a balance between reducing the number of features and retaining enough information to accurately represent the original dataset will have to be struck when using PCA for dimensional reduction. If the reported variance ratio is highly skewed in favor of several components, further dimensionality reduction techniques may be better suited. In this case, it looks like the first principal component (PC1) explains almost all the variance in the dataset (99%), while the remaining components explain very little variance.

```
# Print the explained variance ratio of each principal component
print(pca.explained_variance_ratio_)
```

```
[9.99999985e-01 1.17108136e-08 1.00061662e-09 3.45516678e-10
 2.22867830e-10 1.36975670e-10 8.22412564e-11 7.72673110e-11
 7.56151888e-11 7.19239673e-11]
```

4. Model Building:

As per requirements of assignment, I will be working on Linear, Random Forest, Gradient Boost and Ensemble Regression Models for our target variable 'price'.

1) Linear Regression Model:

After performing PCA on dataset, I splitted the data into train and test folds and fitted the linear regression model on training set and used predict() on testing set as we normally do for every model. Then to assess the performance of the model I calculated MSE (Mean Squared Error), MAE (Mean Absolute Error) and r2 (R- squared) score which are as follows:

```
MSE: 13956032.316370253
MAE: 2606.589282983895
R-squared: 0.9139911015901474
```

MSE calculates the average squared difference between the target variable's predicted and actual values. A lower MSE implies that the model's predictions are closer to the true values, signifying improved prediction accuracy. Same goes with the MAE, lower the value, better the model performance. Then we have R-squared value which actually helps to determine the model performace and prediction, it represents the proportion of variance in the target variable that the model can explain. A score closer to 1 implies that the model explains a large proportion of the variance in the target variable, implying a strong fit to the data.

Based on this result, Linear Regression model shows promising performance. The MSE and MAE values suggest low prediction errors, however the high R-squared value indicates that the model explains a considerable percentage of the variance in the 'price' variable. These findings imply that the linear regression model, which was trained on the reduced-dimensional features acquired using PCA, is effective at predicting the 'price' in the provided dataset.

2) Random Forest Regression Model:

Performing the same steps as I have done above, so the MSE, MAE and r2 values are as follows:

```
MSE: 8189084.2765230695
MAE: 1774.1269057031543
R-squared: 0.9495319227096501
```

The MSE and MAE values obtained from this model are relatively smaller than the ones obtained in Linear Regression model which is quite impressing. R-squared score of this model is 0.949 i.e. 0.95 which indicates 95% of the variance in the target variable, this makes Random Forest Regression model a slightly better fit compared to previous model.

3) Gradient Boost Regression Model:

Performing the same steps as I have done above, so the MSE, MAE and r2 values are as follows:

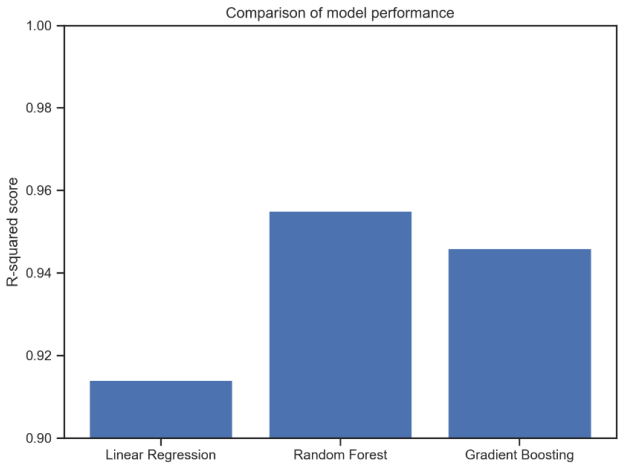
```
MSE: 8849404.413287656
MAE: 1973.9970617421588
R-squared: 0.9454624704274032
```

The MSE and MAE values obtained from this model are relatively smaller than the ones obtained in Linear Regression model which explains this model is better fit to the data than Linear Regression. R2 score of Gradient boost is also better than linear one, its 0.94 i.e., 94% of the variance in the target variable, which is almost same as the Random Forest.

When the R-squared (R2) scores of three regression models, linear regression, random forest, and gradient boost regression, are compared, it is clear that all models describe the variance in the target variable effectively. The random forest model had the greatest R2 score of 0.95, suggesting a good fit to the data and accounting for a significant percentage of the variance. The R2 score for the linear regression model was 0.91, indicating a good fit and explaining a significant percentage of the variation. The R2 value for the gradient boost regression model was 0.94, indicating a high degree of explained variation. Although the random forest model beat the other two models somewhat, all three models are capable of making accurate predictions.

Ranking:

I am going to rank models based on their R2 scores, here is the small plot just to visualise it.



Based on the model rankings and performance indicators, the Random Forest model had the strongest predictive capability of the three models tested. It has the lowest mean squared error (MSE) and mean absolute error (MAE), signifying better data fit and lower prediction errors. The Random Forest model also gets the highest R-squared (R2) score, suggesting that it explains a higher fraction of the variance in the target variable than the other models.

4) A Voter Ensemble:

Voting Regressor combines predictions of multiple regression models and aggregates their result to make final prediction. When I have number of regression models and want to exploit their combined knowledge, the VotingRegressor is ideal. When I have similar-performing models and want to combine their predictions equally, an Averager is handy. Stacking is more complicated, but it allows the meta-model to understand complex correlations between the predictions of the base models. In our case, as we want to leverage the predictions of best performing models, I am going with VotingRegressor. I am considering Linear and Random forest models as they are better performing.

```
# Fit a linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Fit a random forest regressor model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Create a voting regressor that combines the predictions of the linear regression and random forest regressor
voting = VotingRegressor([('lr', lr), ('rf', rf)])
voting.fit(X_train, y_train)

# Predict on the test set
y_pred = voting.predict(X_test)
```

I have fitted the linear and random regressor model on training data set and created a voting regressor which will combine the predictions of these two models and then again fit the model and then prediction. The MSE, MAE and r2 scores of this model:

```
MSE: 8559933.113845672
MAE: 1906.2214299856066
R-squared: 0.9472464378919289
```

In conclusion, the VotingRegressor ensemble model produced better results, with an MSE of 8,559,933.11 and an MAE of 1,906.22. These metrics show low prediction errors and promising predictions. Furthermore, the R-squared score of 0.947 indicates that the model can explain approximately 94.7% of the variance in the target variable. Overall, the VotingRegressor ensemble performs well, identifying patterns and correlations in the data to create accurate predictions for the target variable.

Hyperparameter Tuning: RandomizedSearchCV

A random search CV helps us in selecting the appropriate hyperparameters for our model, resulting in greater performance and prediction accuracy. It enables us to rapidly investigate a large range of hyperparameter values and select the combination that maximizes the model's performance on the supplied dataset.

I am choosing Random Forest regressor because of its versatility, ability to handle non-linear behavior, better interpretability and robustness for outliers and many reasons. It is performed with the help of 'RandmizedSearchCV' class, which includes Random Forest regressor, number of iterations and cross-validation folds. It goes through the parameter combinations and uses cross-validation to evaluate the performance of each combination. Then we predict it on test set and calculate the MSE, MAE and R2 scores.

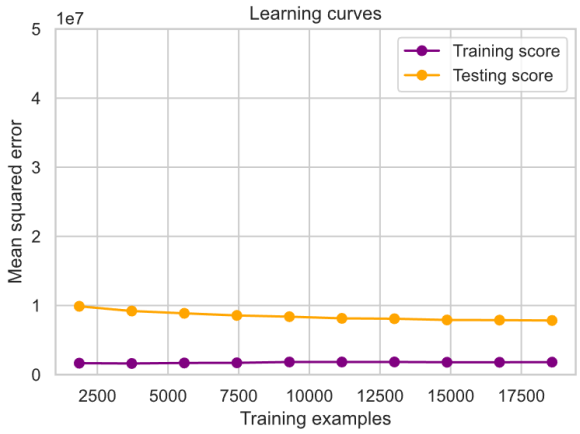
```
MSE: 7187805.335909034
MAE: 1720.0155192101404
R-squared: 0.9557026520925408
```

Based on the scores, the random forest regressor performs well on the test data after being trained with the hyperparameters obtained via randomized search CV. The model has modest prediction errors, with an MSE of 7,187,805.34 and an MAE of 1,720.02. Furthermore, the model explains about 95.57% of the variance in the target variable, as indicated by the high R-squared value of 0.9557. These findings indicate that the random forest regressor with optimized hyperparameters can capture underlying patterns and effectively predict the target variable in the dataset.

Underfit/Overfit Trade-off:

Underfitting and overfitting are classic machine learning difficulties that occur when models are either very simple or overly complicated, resulting in an inability to adequately capture the underlying relationships between input variables and the target variable. If we see the following plot, the curve of the learning curve of a random forest model reveals that the model's mean squared error (MSE) reduces as the number of training samples grows. However, the improvement eventually plateaus and levels out around 1,000 examples on the training set, indicating that future improvements beyond this point are limited.

Looking at the curves, we can say that the testing score is only slightly lower than the training score, it indicates that the model has a low degree of overfitting or that overfitting is not a serious worry. However, as the size of the training set grows, the model's performance on testing data stabilizes, showing that the model has learned to generalize well and is not too impacted by noise or random changes. This stabilization indicates that the model has struck a reasonable balance between fitting the training data and generalizing to previously unknown data.



5. Model Evaluation and Analysis:

A) Cross-Validation:

Let's use Cross-validation to identify whether a model is overfitting the training data. If the performance of the training and testing sets differs significantly, it indicates that the model is not generalising properly and may be overfitting. I am choosing the Random Forest regressor model as it has best score and loss-metrics. To determine the most suitable hyperparameters for a random forest regressor, the code uses randomised search cross-validation. It then uses cross-validation to assess the model's performance, selects the best hyperparameters, and fits the final model to the full dataset.

```
# Define the parameter grid for the random forest regressor
param_dist = {
    'n_estimators': [50, 100, 150],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3]
}

# Create a random forest regressor object
rf = RandomForestRegressor(random_state=42)

# Use randomized search CV with cross-validation to find the best hyperparameters for the random forest regressor
rand_search = RandomizedSearchCV(rf, param_distributions=param_dist, n_iter=10, cv=5, random_state=42, n_jobs=-1)
scores = cross_val_score(rand_search, X, y, cv=5, scoring='r2')
print('Cross-validation scores:', scores)
print('Mean R-squared:', scores.mean())

# Fit the best estimator found by randomized search CV on the entire dataset
rand_search.fit(X, y)
best_rf = rand_search.best_estimator_
```

I setted the number of iterations to 10, which means that 10 different hyperparameter combinations will be tested and number of cross-validation folds to to 5, suggesting that 5-fold cross-validation would be conducted. The cross-validation scores for these 5 folds are:

[0.95762276 0.95908772 0.9569289 0.95595627 0.95589362]

These scores indicate that the cross-validation scores, ranging between 0.9559 and 0.9591, are relatively high and close to each other and closer to 1. This implies that the random forest regressor model performs consistently well across multiple data subsets, showing steady and reliable performance.

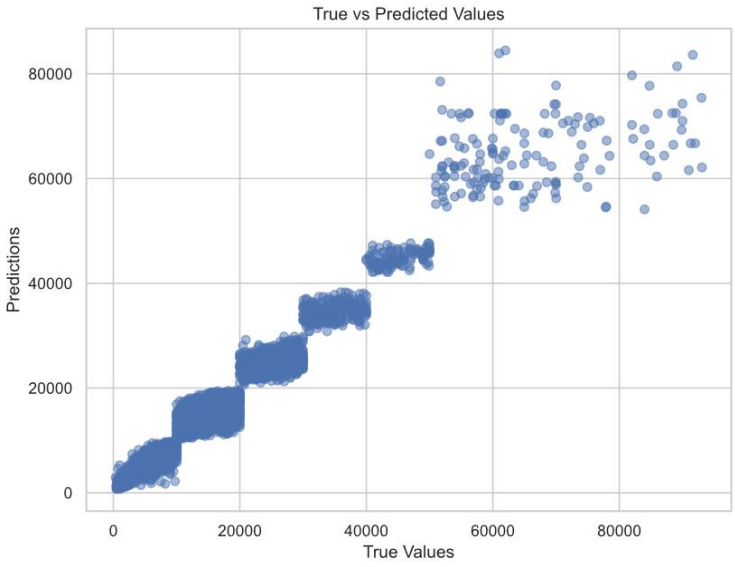
Then we have MSE, MAE and r2 scores:

```
Cross-validation scores: [0.95762276 0.95908772 0.9569289  0.95595627 0.95589362]
Mean R-squared: 0.9570978536549368
MSE: 4406791.384409189
MAE: 1392.4767448829627
R-squared: 0.9728416168791416
```

The R-squared score on the test set is 0.973, suggesting that the model explains about 97.3% of the variance in the target variable. These indicators indicate that the model works well and makes accurate predictions.

B) Analysing True and Predicted plot:

I plotted a scatterplot for analysing the true and predicted values using Random Forest regressor. The plot showed the alignment of values as follows:



Looking at distribution of the points around a diagonal line indicates a relatively strong positive linear relationship between the true and predicted values. This means that the model can capture the underlying patterns and trends in the data. In addition, there are several scattered spots deviate from the diagonal line. These scattered points represent cases in which the model's predictions do not exactly match the true values. The degree of scatter or spread in the points represents the prediction errors of the model.

The scattered spots represent instances where the model's predictions differ from the true values. While these points reveal areas of prediction error, they also provide useful insights into the data's complexity and unpredictability. Examining and interpreting these outliers might reveal underlying trends and potential model improvement areas.

C) SHAP Analysis:

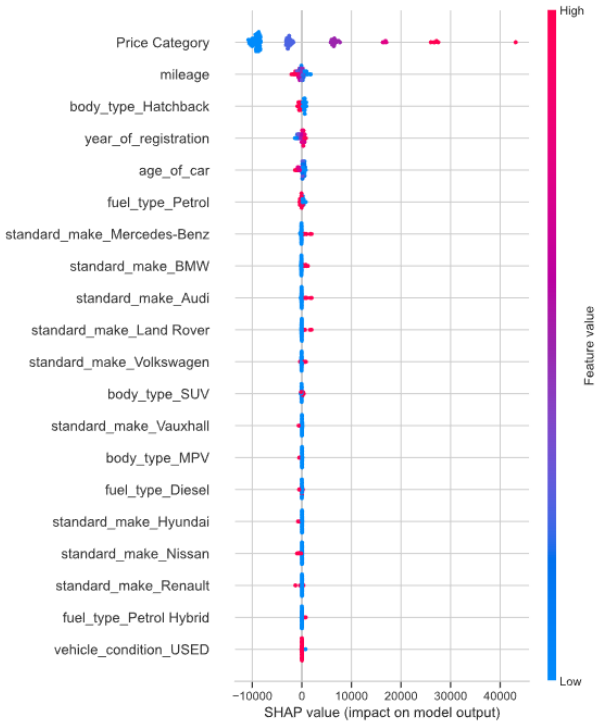
I am using this Mathematical model for explaining further predictions of Random Forest regression model. And the approach is to use SHAP Global and Local Explainability.

1. SHAP with Global Explainability using Summary Plot:

To provide the Global Explainability, SHAP calculates the average impact of each feature on model predictions. This helps in identifying the most important features and how they contribute to the model's output. A SHAP global explanation plot provides an overview of feature importance, allowing us to determine which features have the greatest influence on the model's predictions.

I am considering Random Forest regression model as it has ranked first in our model rankings, we can use any model though. After creating the SHAP explainer object, then SHAP values are calculated for the first 100 rows as I specified it using iloc in the test set using the explainer object. These SHAP values show the contribution of each feature in prediction for each value.

Summary Plot:



As we can see in this summary plot, each feature is placed on a horizontal bar, and its place along y-axis shows its importance. The color bar indicating the value of feature explains that the red color shows the higher values where blue indicates lower values. The bar length represents the amount of feature impact on model prediction. Features having higher impacts are position at the top of the plot.

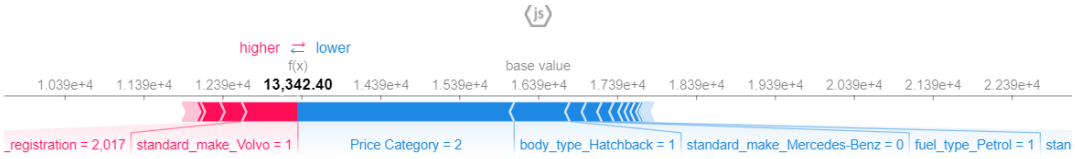
In our summary plot generated for random forest regressor model, the feature at bottom 'vehicle_condition_USED' has biggest red bar which means it has significant impact on model prediction. Then we have 'mileage', 'body_type_Hatchback' and so on.

The feature 'year_of_registration', 'age_of_car' and 'fuel_type_Petrol' also have major impact on model predictions. Whereas, 'Price Category', 'standard_make_Volkswagen'etc features comparatively have lower impact than other features. With this summary plot, we got the insights of which features have major influence on model prediction.

2. SHAP with Local Explanability using Force plot:

For local explanation representation focuses on explaining a specific prediction. It shows the individual feature contributions to the prediction. Force plot explains why the model generated a given prediction by displaying how each feature value affects the result.

There is very little difference in code for Force plot, in global explanability I took first 100 rows from test set, but here in local explanability we take any random instace from test set, then create explainer object and then plot graph using that. Let's have a look into the Force Plot,

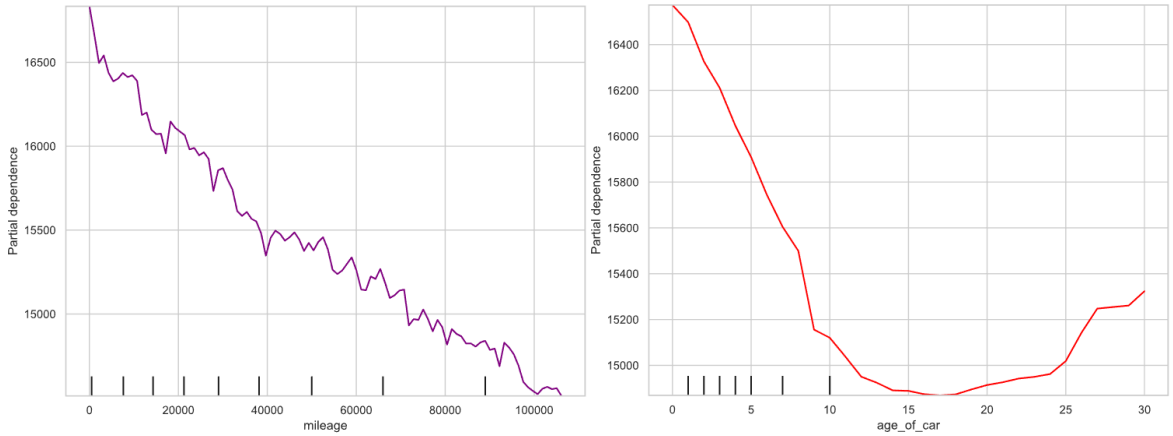


The force plot shows a horizontal bar chart that shows how each feature contributes to the final prediction. A bar represents each feature, and its length and colour reflect the magnitude and direction of its contribution. Positive contributions (in blue) raise the prediction, whereas negative contributions (in red) reduce it. The difference between the expected and predicted values is equal to the sum of the contributions from all features. The force plot explains how each variable influences the forecast for the chosen instance and provides insight into the model's decision-making process. So, we can conlude that

‘Price Category’, ‘body_type_Hatchback’ push the prediction, whereas ‘standard_make’ reduces the prediction.

D) Partial Dependency Plot: PDP

In a machine learning model, partial dependence plots (PDPs) are a useful tool for analysing the relationship between a feature and the target variable. I have done two PDPs, one for mileage and one for automobile age, and their interpretation is explored below.



The first plot where we have plotted the mileage with target variable i.e. price, its shown that as mileage of car increases, the predicted price for that car decreases. So we can say mileage has negative influence on price. The purple line's steepness reflects the amount of the influence, with a steeper line suggesting a larger effect. As a result, the plot reveals that mileage is a one of the important features in predicting car prices.

Whereas in second plot, as age_of_car increases the predicted price drops, which is an expected result. as the curve is not linear, it shows that the relationship between age_of_car and price is not straightforward like mileage. There might be some effect of other features like standard_make or model of the car are causing such results. The area around the curve shows the prediction's uncertainty, with a larger area suggesting greater uncertainty. This can be used to evaluate the accuracy of the model's predictions for various ages of car.