Machine Learning Concepts
Name: Farrukh Nizam Arain

1. Data Understanding and Exploration:

1.1 Meaning and Type of Features:

At the start of the project, data was loaded in a variable to know the number of features and size of the dataset on which the project will be initiated.

```
# Importing the dataset
df = pd.read_csv('../Dataset/adverts.csv')
```

```
Size of the Dataset is:  4824060
Shape of the Dataset is:  (402005, 12)
```

Furthermore, in order to know which features are quantitative and qualitative in the dataset, it’s relevant function was ran and a list was generated.

```
# Extracting a list of Categorical and Numerical Columns from the dataset
cat_col = df.select_dtypes(include=['object']).columns.tolist()
num_col = df.select_dtypes(exclude=['object']).columns.tolist()

# Printing the list of Categorical and Numerical Columns
print("Categorical Columns:", cat_col)
print("Numerical Columns:", num_col)
```

```
Categorical Columns: ['reg_code', 'standard_colour', 'standard_make',
'standard_model', 'vehicle_condition', 'body_type', 'fuel_type']
Numerical Columns: ['public_reference', 'mileage', 'year_of_registration',
'price', 'crossover_car_and_van']
```

Although there were twelve features in the dataset, but type and nature of few of the features will be described for further understanding.

a) Car’s Manufacturer Name (standard_make):
It is a qualitative feature and it has an object datatype. This feature conveys the name of the manufacturer of a car.

b) Car’s Model Name (standard_model):
It is a categorical column and data is stored in this column of object datatype. It conveys model name of a car which is manufactured by a manufacturer.

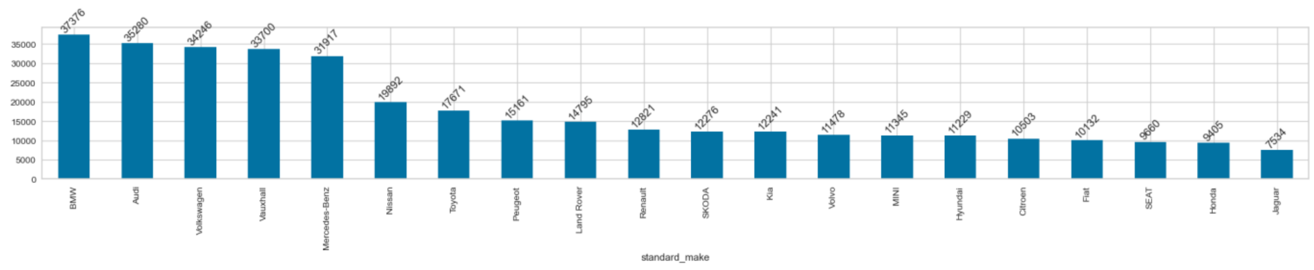
c) Vehicle Condition (vehicle_condition):
Just like the other two features, it is also a categorical column with an object datatype. This feature conveys whether the condition of the vehicle is NEW or USED in the dataset of AutoTrader.

Analysis of Distributions

All the features in the dataset were extensively analysed through statistical, mathematical, and other methods. Some of the key analysis through visualizations are given below:

a) Car's Manufacturer Name (standard_make):

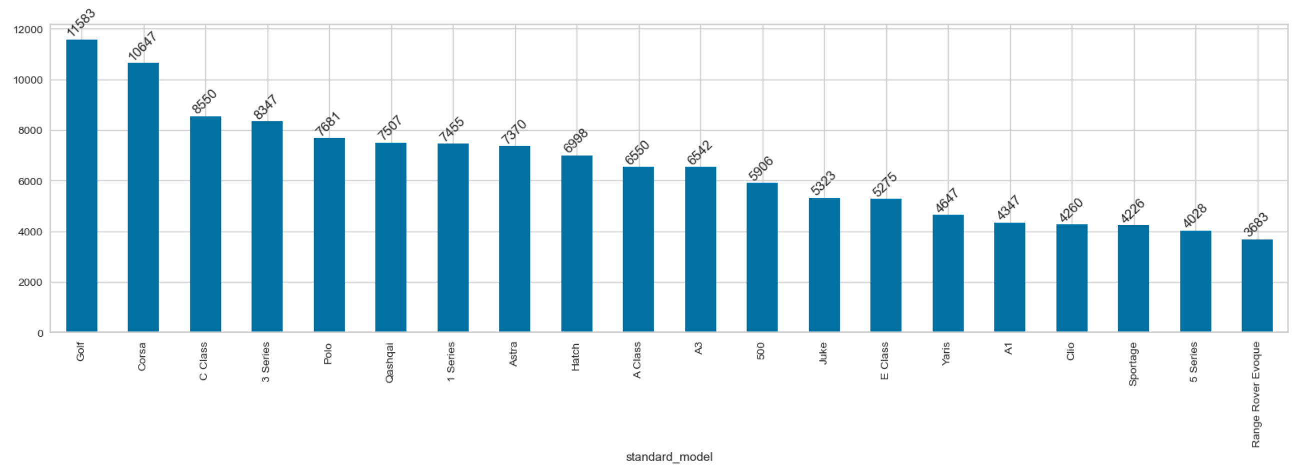
```
# A bar plot for standard_make feature's dispersion
plt.figure(figsize=(25, 3))
plt.ticklabel_format(style='plain')
ax = df['standard_make'].value_counts().nlargest(20).plot(kind='bar')
for i in ax.containers:
    ax.bar_label(i, rotation= 45, label_type="edge")
plt.show()
```



According to the bar plot, most of the vehicles pertained to those manufacturers which are generally known to make vehicles which target customers of different price budgets. There were some vehicles which were known to produce luxurious or sports cars but the number of them were very low in the dataset.

b) Car's Model Name (standard_model):

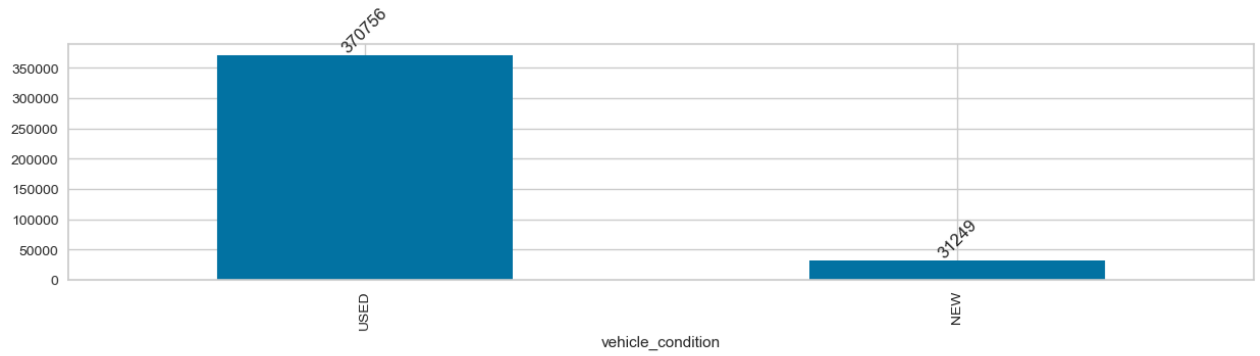
```
# Detailed Visualization the dispersion measures of standard_model column via
bar plot
plt.figure(figsize=(20, 5))
plt.ticklabel_format(style='plain')
ax = df['standard_model'].value_counts().nlargest(20).plot(kind='bar')
# Labelling bars in the barplot
for i in ax.containers:
    ax.bar_label(i, rotation= 45, label_type="edge")
plt.title("Barplot of Standard Model")
plt.show()
```



This categorical feature revealed another interesting insight that BMW, Volkswagen, Nissan, Toyota, and similar brands are prevalent in numbers which are known to be the ones which people from every class and budget can afford. This information corresponded with the price column's earlier findings which showed that most of the cars in the inventory are cheaper cars.

c) Vehicle's Condition (vehicle_condition):

```python
# Value Dispersion Visualisation vehicle_condition column via bar plot
plt.figure(figsize=(15, 3))
plt.ticklabel_format(style='plain')
ax = df['vehicle_condition'].value_counts().plot(kind='bar')
# Labelling bars in the barplot
for i in ax.containers:
    ax.bar_label(i, rotation= 45, label_type="edge")
plt.show()
```



More than 92 percent of the vehicles comprised of used cars, which illustrated that AutoTrader deals majorly with them. Additionally, it also depicted that customers with used vehicles prefers their company more than customers who want to buy new cars.

1.2 Analysis of Predictive Power of Features:

Fortunately, there were quite a few predictors which were considered to being a good enough feature to predict price of a vehicle. Primarily, understanding of the problem, domain, nature of the features were considered when selecting them.

a) Car's Manufacturer Name (standard_make):
Although it may seem that the name of the manufacturer might not be a feature that will help in predicting the price of the car; but it does play a key role because generally a reputation attached to the name have an influence on not only the price of the vehicle, but also it helps in retaining a vehicle's value for a significant amount of time.

b) Car's Model Name (standard_model):
There are numerous vehicle manufacturers which launch models of the cars with different price points just to attract customers of different price segments. For instance, recently Suzuki launched Maruti Suzuki K10 which targeted customers with low price point, and on the other hand, it also launched Maruti Suzuki Grand Vitara to attract customers with higher budget range.

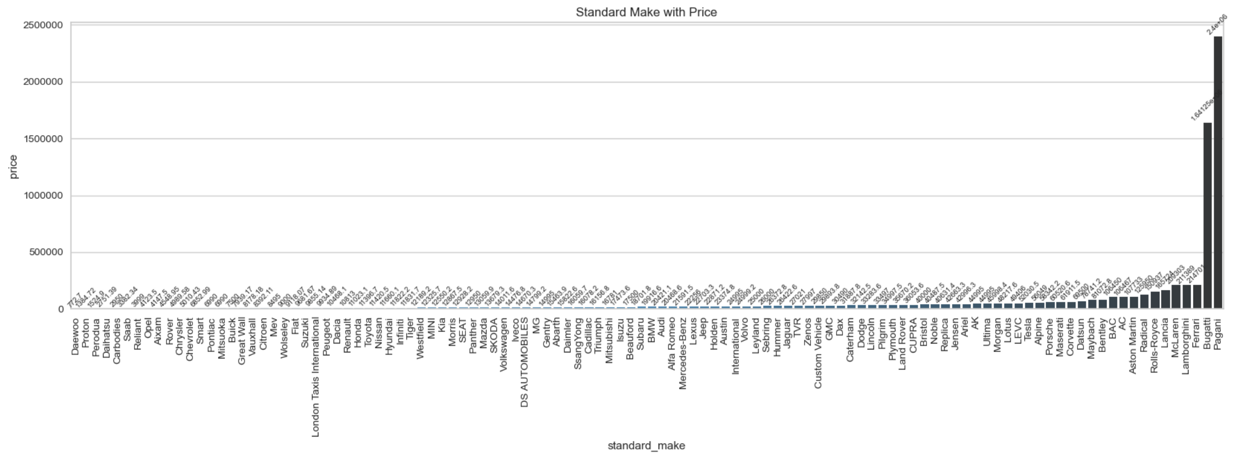
1.3 Data Processing for Data Exploration and Visualization:

Almost every feature revealed a good association with the price in the dataset, however there were a few which stood out above all the rest.

a) Car's Manufacturer Name (standard_model):

```python
# Bar plot of standard_make with price
plt.figure(figsize=(20, 5))
plt.ticklabel_format(style='plain')
```

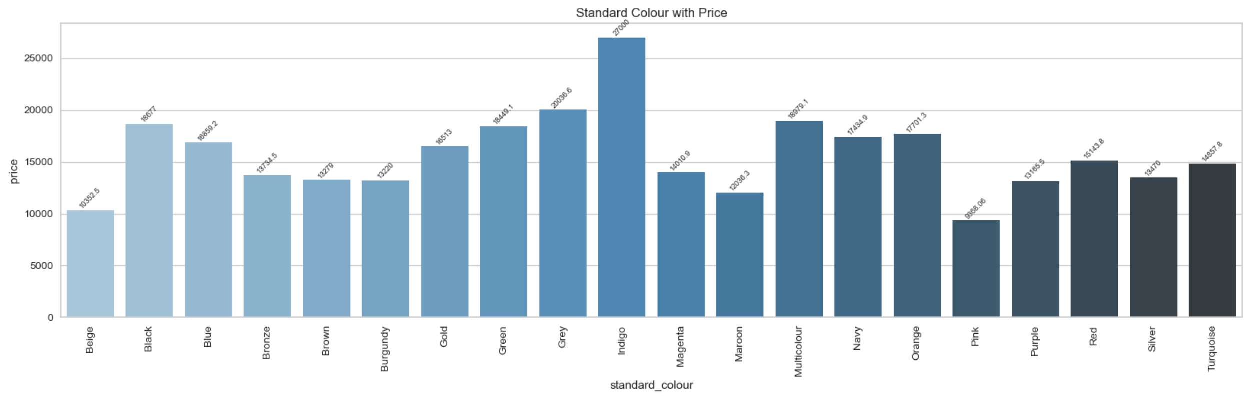
```
ax = sns.barplot(data=df.groupby('standard_make')['price'].mean().reset_in-
dex().sort_values(by='price', ascending=True), x='standard_make', y='price',
errorbar=None, palette='Blues_d')
plt.xticks(rotation=90)
for i in ax.containers:
    ax.bar_label(i, rotation=45 ,label_type="edge", size=7)
plt.title("Standard Make with Price")
plt.show()
```



In the first plot, it was observed that the mean price of expensive brands like Bugatti, Ferrari, Lamborghini are priced higher than cheaper ones like Toyota and Honda because cars of popular brands with reputation of building excellent vehicles are priced higher than others in the market.

b) Car's Model (standard_model):

```
# Bar plot of standard_colour with price
plt.figure(figsize=(20, 5))
plt.ticklabel_format(style='plain')
ax=sns.barplot(data=df.groupby('standard_colour')['price'].mean().reset_index(
).sort_values(by='standard_colour', ascending=True).head(20),
x='standard_colour', y='price', errorbar=None, palette='Blues_d')
plt.xticks(rotation=90)
for i in ax.containers:
    ax.bar_label(i, rotation=45 ,label_type="edge", size=7)
plt.title("Standard Colour with Price")
plt.show()
```



Next bar plot in between a vehicle's colour and price showed a clear indication that mean values of different colours of vehicles were prices differently in the dataset and it confirms that some colours more popular than others and there is a high demand for them from the customers.

2. Data Processing for Machine Learning:

2.1 Dealing with Missing Values, Outliers, and Noise:

a) Missing Values:

```
# Removing Missing Values in the standard_colour column
standard_model_mode_series = df.groupby(['standard_make',
'standard_model'])['standard_colour'].transform(lambda x: x.mode().iloc[0] if
not x.mode().empty else pd.NA)
df['standard_colour'] =
df['standard_colour'].fillna(standard_model_mode_series)
```

```
# 24 missing values in standard_colour remains which does not have any
standard_make and standard_model combination.
df.isna().sum()
```

```
24
```

```
# Deleting all the remaining missing values in the dataset
df.dropna(inplace=True)
```

In this code snippet, features of make and model were grouped together and colour value of its relevant mode was taken & updated in the relevant observations with NaN values. Rest of the values were deleted.

b) Error Values:

```
# Detecting possible outliers in the mileage column and deleting them
mileage_counts =
df['mileage'].value_counts().to_frame().reset_index().sort_values(by='mileage'
, ascending=True)
mileage_counts.columns = ['mileage', 'count']
pd.set_option('display.max_rows', None)
print(mileage_counts)
```

```
        mileage  count
0           0.0  16207
...
68053  990000.0      1
78985  999999.0      1
```

```
df.drop(df[df['mileage'] == 999999.0].index, inplace=True)
```

In the column of mileage of car, value of 999999.0 was removed because it was assumed that it was for a typing error which was either from a customer or a malfunction.

2.2 Feature Engineering, Data Transformations, Feature Selection:

a) Segregating NEW and USED Cars in the Dataset:

```
# As per assumption, mileage up to 100 miles is considered as new car, so we
will replace status with `NEW` for mileage less than or equals to 100 miles
mask = (df['vehicle_condition'] == 'USED') & (df['mileage'] <= 100)
x = df[mask].sort_values(by='mileage',
ascending=False)['vehicle_condition'].replace('USED', 'NEW')
df.update(x)
```

```
# Checking whether the vehicle_condition has been updated
df.query('vehicle_condition == "USED" and mileage <= 100')
```

Some of the research was conducted regarding the status of vehicle and in relation to that it was assumed that cars having `mileage` value till 100 miles were considered as 'NEW' in the relevant column and the rest were considered to be ‘USED’ cars.

b) New Column for Price Categories:

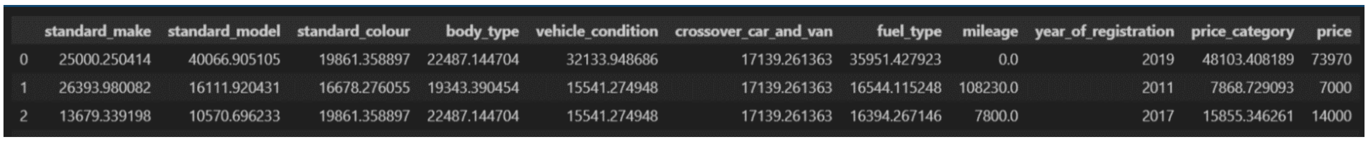
```
# Declaring price bins and its related labels for the used cars
price_bins = [0, 13999, 17999, 21999, 25999, 29999, 33999, 37999,
float('inf')]
price_labels = ['Very Low', 'Low', 'Medium-Low', 'Medium', 'Medium-High',
'High', 'Very High', 'Luxury']
# Selecting used cars and inserting the price category in the dataset
df['price_category'] = pd.cut(df.query('vehicle_condition ==
"USED"')['price'], bins=price_bins, labels=price_labels, right=True)
```

```
# Declaring price bins and its related labels for new cars
price_bins = [0, 11999, 16999, 21999, 35999, 49999, 69999, 99999,
float('inf')]
price_labels = ['Very Low', 'Low', 'Medium-Low', 'Medium', 'Medium-High',
'High', 'Very High', 'Luxury']
df.loc[df['vehicle_condition'] == 'NEW', 'price_category'] =
pd.cut(df.loc[df['vehicle_condition'] == "NEW", 'price'], bins=price_bins,
labels=price_labels, right=True)
```

A new column was created using a new function pd.cut after distributing and labelling the price values into almost equal-sized quantiles / bins. All those categories were assumed after obtaining the price ranges from Honest John for real-world categories for analysis or visualization for old car price ranges. Furthermore, similar code was executed for newer cars where and its data was researched and assumed from different websites like Motors.co.uk.

c) Categorically-Encoding the Relevant Columns:

```
# Creating a list of the categorical variables
te_cols = ['standard_make', 'standard_model', 'standard_colour', 'body_type',
'fuel_type', 'crossover_car_and_van', 'vehicle_condition', 'price_category']
# Applying the target_encoder function to the categorical variables
encoder = ce.TargetEncoder(cols=te_cols)
df = encoder.fit_transform(df, df['price']).copy()
df.head(5)
```



```
   standard_make  standard_model  standard_colour     body_type  vehicle_condition  crossover_car_and_van     fuel_type   mileage  year_of_registration  price_category  price
0   25000.250414    40066.905105     19861.358897  22487.144704       32133.948686           17139.261363  35951.427923       0.0                  2019    48103.408189  73970
1   26393.980082    16111.920431     16678.276055  19343.390454       15541.274948           17139.261363  16544.115248  108230.0                  2011     7868.729093   7000
2   13679.339198    10570.696233     19861.358897  22487.144704       15541.274948           17139.261363  16394.267146    7800.0                  2017    15855.346261  14000
```

One-Hot Encoding was not used here because it would have created a lot of columns and made the dataset very sparse. Label Encoding also did not give good scores. So, Target Encoding was applied which was closer and more relevant to the categorical features in the dataset and it did not output any issues like overfitting or unfitting of data.

3. Model Building:

3.1 Algorithm Selection, Model Instantiation and Configuration:

a) KNN Regression:

In the first model pertaining to regression problem, a random KN value was chosen which was then fitted against the train dataset and an accuracy score was calculated which gave an initial prediction level as whether the model was underfit or overfit.

```
# Declaring value for nearest neighbours
kn_value = 1
# For accuracy scores, fitting the dataset and obtaining the scores
knnr = KNeighborsRegressor(n_neighbors=kn_value)
knnr.fit(X_train_scaled, y_train)
# Accuracy scores of the traning and testing in the dataset
knn_train_Score = knnr.score(X_train_scaled, y_train); knn_Test_Score =
knnr.score(X_test_scaled, y_test);
print('KNN Regression Train Score is: ', knn_train_Score); print('KNN
Regression Test Score is: ', knn_Test_Score)
```

```
KNN Regression Train Score is:  1.0
KNN Regression Test Score is:  0.8576632222880993
```

b) Decision Tree Regression:
As the training and testing data was already split and rescaled, an instance of decision tree regressor was initiated, fitted and accuracy scores were analysed before it was processed through the usual `GridSearchCV` and `Rank` functions.

```
# Initializing the maximum depth of the tree
depth = 2
# Creating the decision tree regressor instance and fitting the model
dtr = DecisionTreeRegressor(max_depth=depth, random_state=rs)
dtr.fit(X_train_scaled, y_train)
# Accuracy scores of the traning and testing in the dataset
dtr_train_Score = dtr.score(X_train_scaled, y_train); dtr_Test_Score =
dtr.score(X_test_scaled, y_test);
print('Decision Tree Regression Train Score is: ', dtr_train_Score);
print('Decision Tree Regression Test Score is: ', dtr_Test_Score)
```

```
Decision Tree Regression Train Score is:  0.7470835799044246
Decision Tree Regression Test Score is:  0.6291818309704147
```

c) Linear Regression:
All of the steps involving splitting, scaling, and fitting the data were executed in the previous steps. So, for now, the model was fitted, and the scores were obtained.

```
# Creating an instance of the Linear Regression Model
lr = LinearRegression()
# Fitting the model
lr.fit(X_train_scaled, y_train)
# Accuracy scores of the traning and testing in the dataset
lr_train_Score = lr.score(X_train_scaled, y_train); lr_Test_Score =
lr.score(X_test_scaled, y_test);
# Displaying the scores
print('Linear Regression Train Score is: {}'.format(lr_train_Score));
print('Linear Regression Test Score is: {}'.format(lr_Test_Score))
```

```
Linear Regression Train Score is: 0.8402123114855913
Linear Regression Test Score is: 0.8242438320253085
```

3.2 Grid Search, Model Building and Selection:

a) KNN Regression:

```
# Creating a dictionary of parameters to be searched
param_grid = {'n_neighbors': np.arange(1, 21)}
# Instantiating GridSearchCV
knnr_cv = GridSearchCV(knnr, param_grid, cv=5, return_train_score=True)
# Fitting the model
knnr_cv.fit(X_train_scaled, y_train)
```

```
# Ranking the scores by the best score
knnr_cv_results =
pd.DataFrame(knnr_cv.cv_results_).sort_values(by='rank_test_score')
knnr_cv_results = knnr_cv_results[['param_n_neighbors', 'mean_train_score',
'std_train_score', 'mean_test_score', 'std_test_score',
'rank_test_score']].head(5).reset_index(drop=True)
knnr_cv_results
# Displaying the best parameters and the best score
print("Tuned KNN Regression Parameter: {}".format(knnr_cv.best_params_))
print("Best score is {}".format(knnr_cv.best_score_))
```

```
Tuned KNN Regression Parameter: {'n_neighbors': 5}
Best score is 0.8572790598381215
```

```
# Extracting the best estimator
kn_value = knnr_cv.best_params_['n_neighbors']
# For accuracy scores, fitting the dataset and obtaining the scores
knnr = KNeighborsRegressor(n_neighbors=kn_value)
knnr.fit(X_train_scaled, y_train)
# Accuracy scores of the traning and testing in the dataset
knn_train_Score = knnr.score(X_train_scaled, y_train); knn_Test_Score =
knnr.score(X_test_scaled, y_test);
print('Best KNN Regression Train Score is: ', knn_train_Score); print('Best
KNN Regression Test Score is: ', knn_Test_Score)
```

```
Best KNN Regression Train Score is:  0.914071003960354
Best KNN Regression Test Score is:  0.8748839111978068
```

The significant improvement in scores of Train and Test proved not only the successful implementation of the model, but also `GridSearchCV` and `Rank` algorithms improved the scores further.

b) Decision Tree Regression:

```
# For finding the best fit for the decision tree regressor, a list of
max_depth values are created
param_grid = {'max_depth': np.arange(1,21)}
# Instantiating GridSearchCV
dtr_cv = GridSearchCV(DecisionTreeRegressor(random_state=rs), param_grid,
cv=5, return_train_score=True)
# Fitting the model
dtr_cv.fit(X_train_scaled, y_train)
# Ranking the scores by the best score
dtr_cv_results =
pd.DataFrame(dtr_cv.cv_results_).sort_values(by='rank_test_score')
dtr_cv_results = dtr_cv_results[['param_max_depth', 'mean_train_score',
'std_train_score', 'mean_test_score', 'std_test_score',
'rank_test_score']].head(5).reset_index(drop=True)
# Printing the best parameters and the best score
print("Tuned Decision Tree Regression Parameter: {}".for-
mat(dtr_cv.best_params_))
```

```
Tuned Decision Tree Regression Parameter: {'max_depth': 16}
```

```
# Extracting the best value and then fitting the model
depth = dtr_cv.best_params_['max_depth']
dtr = DecisionTreeRegressor(max_depth=depth, random_state=rs)
dtr.fit(X_train_scaled, y_train)
# Accuracy scores of the training and testing in the dataset
```

```
dtr_train_Score = dtr.score(X_train_scaled, y_train); dtr_Test_Score =
dtr.score(X_test_scaled, y_test);
print('Decision Tree Regression Train Score is: ', dtr_train_Score);
print('Decision Tree Regression Test Score is: ', dtr_Test_Score)
```

```
Decision Tree Regression Train Score is:  0.9995553079829906
Decision Tree Regression Test Score is:  0.8346250419305771
```

Previously, the training and testing scores were abysmal, but the significant improvements proved the successful implementation `GridSearchCV` and `Rank` algorithms for the Decision Tree Regression.

c) Linear Regression:

```
# Declaring parameters for the ridge regularization
param_grid = {'fit_intercept': [True, False], 'alpha': [1e-15, 1e-10, 1e-8,
1e-3, 1e-2, 1, 5, 10, 20, 30, 35, 40, 45, 50, 55, 100]}  # Alpha values are
the regularization values
ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=5, return_train_score=True)
ridge_cv.fit(X_train_scaled, y_train)
# Print grid search results
lr_cv_results =
pd.DataFrame(ridge_cv.cv_results_).sort_values(by='rank_test_score')
lr_cv_results = lr_cv_results[['params','mean_train_score', 'std_train_score',
'mean_test_score', 'std_test_score',
'rank_test_score']].head(5).reset_index(drop=True)
# Extract best parameters
best_params = ridge_cv.best_params_
fit_intercept = best_params['fit_intercept']
alpha = best_params['alpha']
# Fit final model with best parameters
```

```
ridge = Ridge(fit_intercept=fit_intercept, alpha=alpha)
ridge.fit(X_train_scaled, y_train)
# Evaluate model performance
ridge_train_Score = ridge.score(X_train_scaled, y_train)
ridge_Test_Score = ridge.score(X_test_scaled, y_test)
# Applying the best parameters to lr which is our main model
lr = Ridge(fit_intercept=fit_intercept, alpha=alpha)
lr.fit(X_train_scaled, y_train)
# Accuracy scores of the training and testing in the dataset
lr_train_Score = lr.score(X_train_scaled, y_train); lr_Test_Score =
lr.score(X_test_scaled, y_test);
# Displaying the scores
print('Linear Regression Train Score is: {}'.format(lr_train_Score));
print('Linear Regression Test Score is: {}'.format(lr_Test_Score))
```

```
Linear Regression Train Score is: 0.8400572053245074
Linear Regression Test Score is: 0.8263666607153445
```

Before applying Ridge, Lasso function was applied to tried to increase the score, but it did not show a major difference in the accuracy percentage(s). However, Ridge showed an increase of one percent for target variable only but not for training variable.

4. Model Evaluation and Analysis:

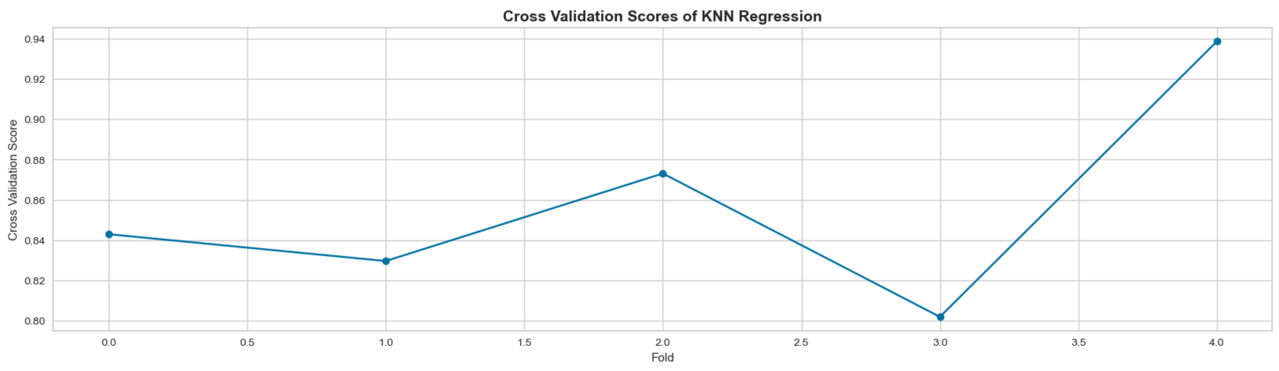
4.1 Coarse-Grained Evaluation / Analysis:

a) KNN Regression:

```
# Applying Related Cross Validation function to the KNN Regression Model
cv_scores = cross_val_score(knnr, X_train_scaled, y_train, cv=5)
# Displaying the scores
print('Scores for Each Fold:', cv_scores)
print('Average CV: ', cv_scores.mean())
```

```
Scores for Each Fold: [0.84297133 0.82966303 0.87309481 0.80197027 0.93869586]
Average CV:  0.8572790598381215
```

```
# Line plot of the cross-validation scores
plt.figure(figsize=(8,6))
plt.plot(cv_scores, marker='o')
plt.xlabel('Fold')
plt.ylabel('Cross Validation Score')
plt.title('Cross Validation Scores of KNN Regression', fontweight='bold',
fontsize=14)
plt.show()
```



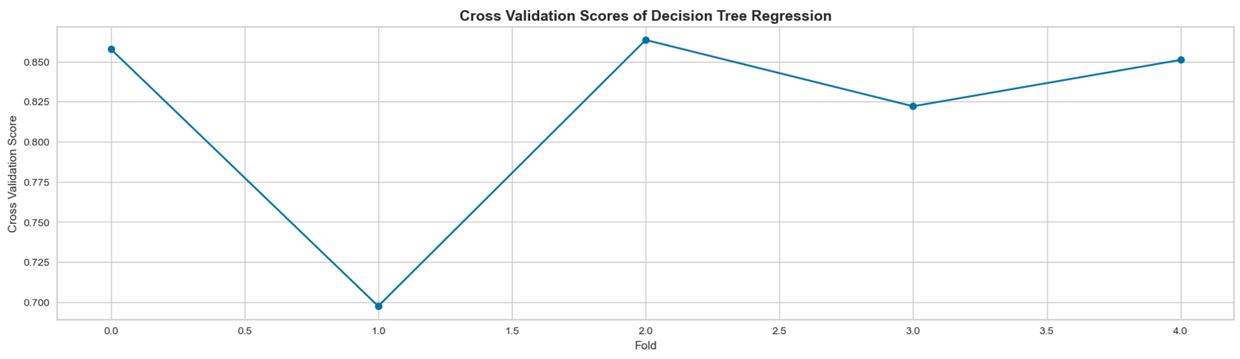
After analysing the multiple accuracy scores through five folds and taking mean of the means, it is to be proved that the model and dataset are compatible with each other and have no issues whatsoever.

b) Decision Tree Regression:

```
# Applying Related Cross Validation function to the DT Regression Model
cv_scores = cross_val_score(dtr, X_train_scaled, y_train, cv=5)
# Displaying the scores
print('Scores for Each Fold:', cv_scores)
print('Average CV: ', cv_scores.mean())
```

```
Scores for Each Fold: [0.85781427 0.69745594 0.86348626 0.82212891 0.8510394 ]
Average CV:  0.8183849560196492
```

```
# Line plot of the cross-validation scores
plt.figure(figsize=(8,6))
plt.plot(cv_scores, marker='o')
plt.xlabel('Fold')
plt.ylabel('Cross Validation Score')
plt.title('Cross Validation Scores of Decision Tree Regression',
fontweight='bold', fontsize=14)
# Displaying the plot
plt.show()
```



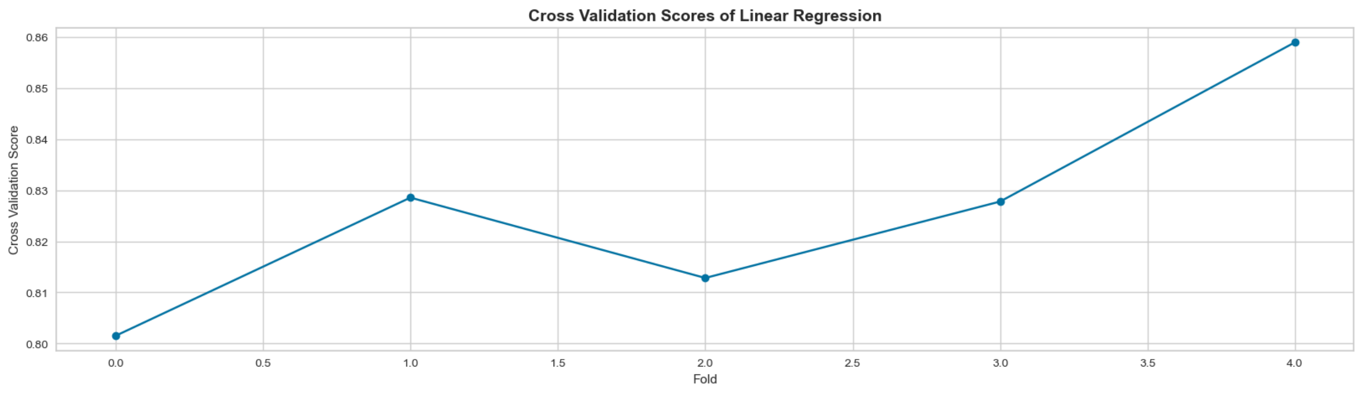
Multiple instances of the K-Fold CV were executed to make sure if the accuracy is according to the client's requirements or not. Fortunately, all the values and even the value of mean suggested that the model and dataset are compatible with each other and have no issues whatsoever.

c) Linear Regression:

```
# Applying Related Cross Validation function to the Linear Regression Model
cv_scores = cross_val_score(lr, X_train_scaled, y_train, cv=5)
# Displaying the scores
print('Scores for Each Fold:', cv_scores)
print('Average CV: ', cv_scores.mean())
```

```
Scores for Each Fold: [0.80151328 0.82854524 0.81282695 0.8277806  0.8589277 ]
Average CV:  0.8259187548659804
```

```
# Line plot of the cross-validation scores
plt.figure(figsize=(8,6))
plt.plot(cv_scores, marker='o')
plt.xlabel('Fold')
plt.ylabel('Cross Validation Score')
plt.title('Cross Validation Scores of Linear Regression', fontweight='bold',
fontsize=14)
plt.show()
```



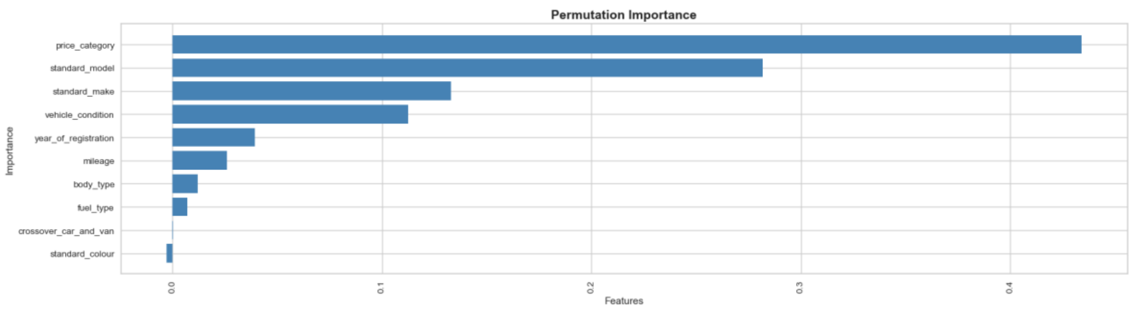
Just like with the other two models, the accuracy of K-Fold CV was according to the client's requirements. Fortunately, all the values and even the value of mean suggested that the model and dataset are compatible with each other and have no issues.

4.2 Feature Importance:

a) KNN Regression:

```
# Calculating permutation importance
p_importance = permutation_importance(knnr, X_test_scaled, y_test)
# Creating a DataFrame of feature names and permutation importance
df_perm_importance = pd.DataFrame({'feature': X.columns.tolist(),
'importance': p_importance.importances_mean})
# Sorting the DataFrame in descending order of importance
df_perm_importance.sort_values(by='importance', ascending=True, inplace=True)
```

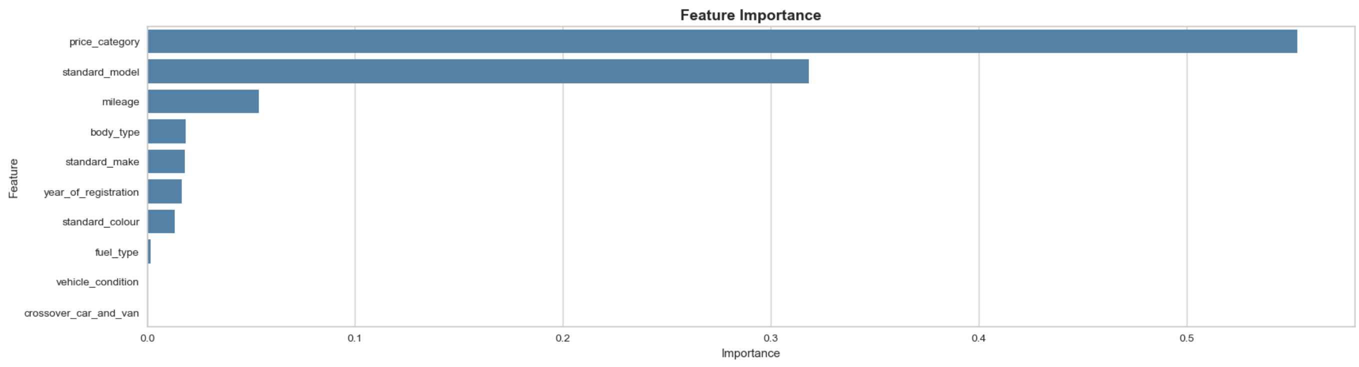
```
# Plotting the permutation importance
plt.figure(figsize=(8, 6))
plt.barh(df_perm_importance['feature'], df_perm_importance['importance'],
color='steelblue')
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Permutation Importance', fontweight='bold', fontsize=14)
plt.show()
```



As per the plot above, column of price category was the most important regarding the dataset because price specific categories based on the real world data makes it easier for the model to predict them conveniently and with higher precision. Furthermore, on second and third place, car's model, and vehicle's condition were placed because generally model of any automobile manufacturer represent a category of customers with different price points, and categorization of used and new cars based on the requirements by AutoTrader also makes it easier for the model to predict the prices as per the standards of the employer.

b) Decision Tree Regression

```
# For decision tree regressor, calculating the feature importance
dtr_importance = dtr.feature_importances_
# Creating a DataFrame of feature importance
dtr_feature_importance = pd.DataFrame({'feature': X.columns, 'importance':
dtr_importance})
# Sorting the DataFrame in descending order of feature importance
dtr_feature_importance.sort_values(by='importance', ascending=False,
inplace=True)
# Plotting the feature importance
plt.figure(figsize=(8,6))
sns.barplot(x='importance', y='feature', data=dtr_feature_importance,
color='steelblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance', fontweight='bold', fontsize=14)
plt.show()
```

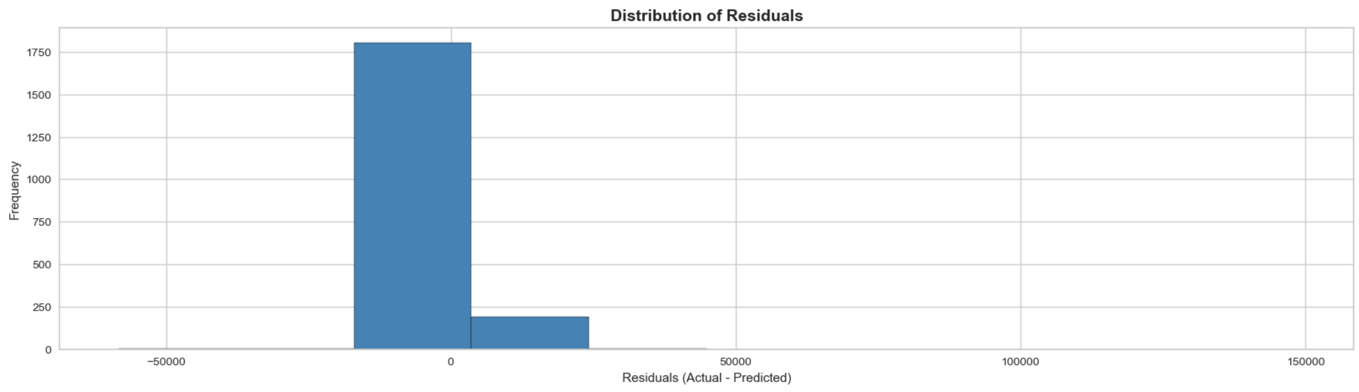


Regarding Decision Tree Regressor, price category was again on top of the list due to the categorization of price points based on the real world data which was applied due to feature engineering. On the second place, standard model was again one of the most important features because as mentioned before, manufacturers around the world use model names to target cars specific to customers having unique needs and budget. On the third place, mileage is also a universal feature which effects the price of the vehicle because the higher mileage numbers effect the price of the vehicle negatively because it denotes that the overall condition of the car.

4.3 Fine-Grained Evaluation:

a) KNN Regression:

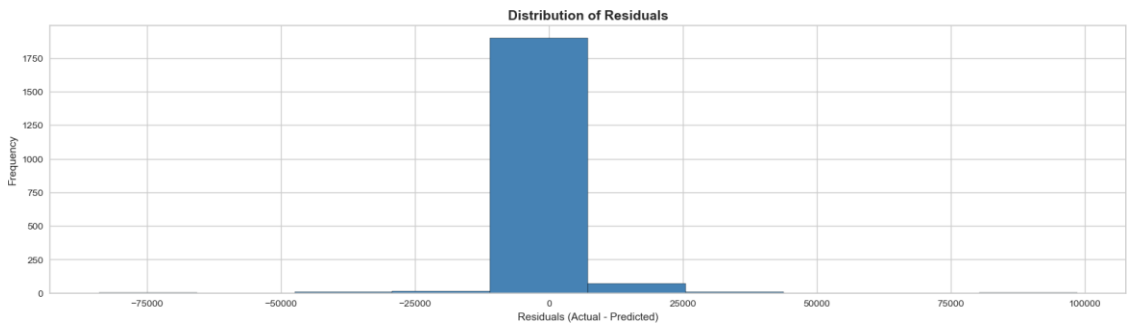
```
# Calculate the residuals
y_pred = knnr.predict(X_test_scaled)
residuals = y_test - y_pred
# Plotting the residuals
plt.figure(figsize=(8, 6))
ax = plt.hist(residuals, bins=10, edgecolor='black', color='steelblue')
plt.xlabel('Residuals (Actual - Predicted)')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals', fontweight='bold', fontsize=14)
plt.show()
```



According to the histogram, it clarifies the excellent performance of the KNN regressor on the dataset, even after its complexity. It forms a normal distribution where most of the values cluster around zero, with few number of instances of large positive or negative residuals.

b) Decision Tree Regression:

```
# For Decision Tree Regressor, calculating the residuals
y_pred = dtr.predict(X_test_scaled)
residuals = y_test - y_pred
# Plotting the residuals
plt.figure(figsize=(8,6))
ax = plt.hist(residuals, bins=10, color='steelblue', edgecolor='black')
plt.xlabel('Residuals (Actual - Predicted)')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals', fontweight='bold', fontsize=14)
plt.show()
```



For Decision Tree Regressor, not only the accuracy scores mentioned above were excellent, but also the residuals(the differences between the observed and predicted values) hovered around zero. It suggests that the model is a good fit, and, on an average, the model is neither systematically overestimating nor underestimating the target variable. Overall, it indicates an excellent performance of the model.