Advanced Machine Learning Project

Contents

Notes:

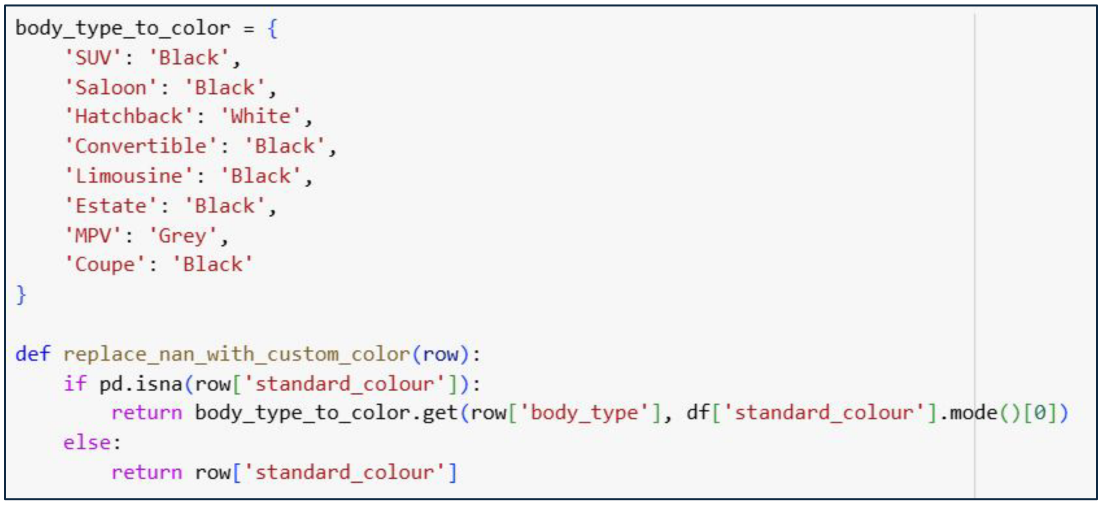
1- In this report, given the constraints of limited space, only the most significant components and concise explanations of the work are presented. For a thorough understanding and complete details of the methodologies employed, please refer to the accompanying Jupyter file.

2- The Python version and two key libraries used for the project were: Python 3.11.12, scikit-learn version 1.6.1, and numpy version 2.0.2. Other versions may cause errors or produce outputs with differing interfaces.

1 PRELIMINARY TASK

1.1 DATA DESCRIPTION AND PRE-PROCESSING

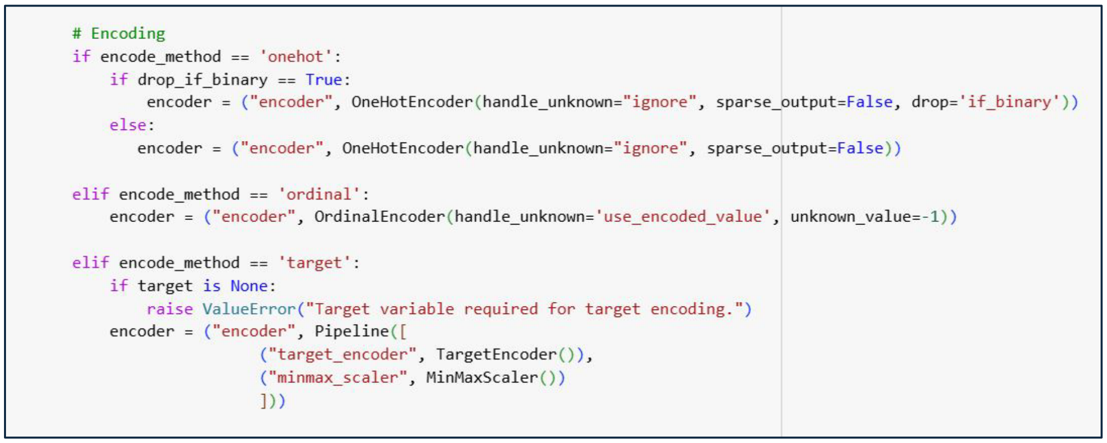
This step involves removing irrelevant and redundant columns, clearing out noise from the dataset, imputing missing values, and addressing outliers. A tailored approach is employed for each feature based on its specific type. These methods are determined through careful investigation to identify which strategies enhance the predictive power of the model most effectively. Pipelines are established for further preprocessing tasks, including the encoding of categorical features and the scaling of numerical ones. The function of these pipelines is highly flexible and reproducible, allowing users to easily modify the method names in the function to create pipeline instances which suits their needs best. All pipelines are efficiently implemented using the "ColumnTransformer" in a single step. Ultimately, a sample with a size of 10000 is randomly chosen from the preprocessed data for further exploration and modelling.



```
body_type_to_color = {
    'SUV': 'Black',
    'Saloon': 'Black',
    'Hatchback': 'White',
    'Convertible': 'Black',
    'Limousine': 'Black',
    'Estate': 'Black',
    'MPV': 'Grey',
    'Coupe': 'Black'
}

def replace_nan_with_custom_color(row):
    if pd.isna(row['standard_colour']):
        return body_type_to_color.get(row['body_type'], df['standard_colour'].mode()[0])
    else:
        return row['standard_colour']
```

Figure 1: Imputation of "standard_colour" based on the "body_type" of the row



```
    # Encoding
    if encode_method == 'onehot':
        if drop_if_binary == True:
            encoder = ("encoder", OneHotEncoder(handle_unknown="ignore", sparse_output=False, drop='if_binary'))
        else:
           encoder = ("encoder", OneHotEncoder(handle_unknown="ignore", sparse_output=False))

    elif encode_method == 'ordinal':
        encoder = ("encoder", OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))

    elif encode_method == 'target':
        if target is None:
            raise ValueError("Target variable required for target encoding.")
        encoder = ("encoder", Pipeline([
                        ("target_encoder", TargetEncoder()),
                        ("minmax_scaler", MinMaxScaler())
                       ]))
```

Figure 2: Encoding part of pipeline

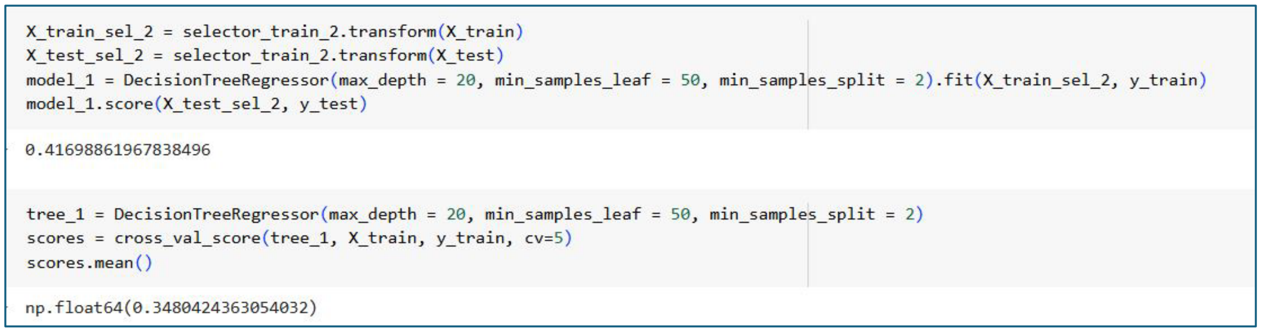
2 PART I

2.1 AUTOMATED FEATURE SELECTION

All automated feature selection methods, including "SelectKBest," "RFECV," and "SFS," are evaluated across various models. Different types of models are selected to facilitate a comparison of the results based on the methods employed.

SelectKBest is applied with k=10 on a linear regression model using the "f_regression" scoring function, which resulted in an improvement of approximately 6 percent in score accuracy.

SelectKBest is also implemented with k=10 on a decision tree model with "mutual_info_regression" scoring function. This also improved prediction score accuracy on the original data by almost 7 percent.



```
X_train_sel_2 = selector_train_2.transform(X_train)
X_test_sel_2 = selector_train_2.transform(X_test)
model_1 = DecisionTreeRegressor(max_depth = 20, min_samples_leaf = 50, min_samples_split = 2).fit(X_train_sel_2, y_train)
model_1.score(X_test_sel_2, y_test)

0.41698861967838496

tree_1 = DecisionTreeRegressor(max_depth = 20, min_samples_leaf = 50, min_samples_split = 2)
scores = cross_val_score(tree_1, X_train, y_train, cv=5)
scores.mean()

np.float64(0.3480424363054032)
```

Figure 1: SelectKBest results

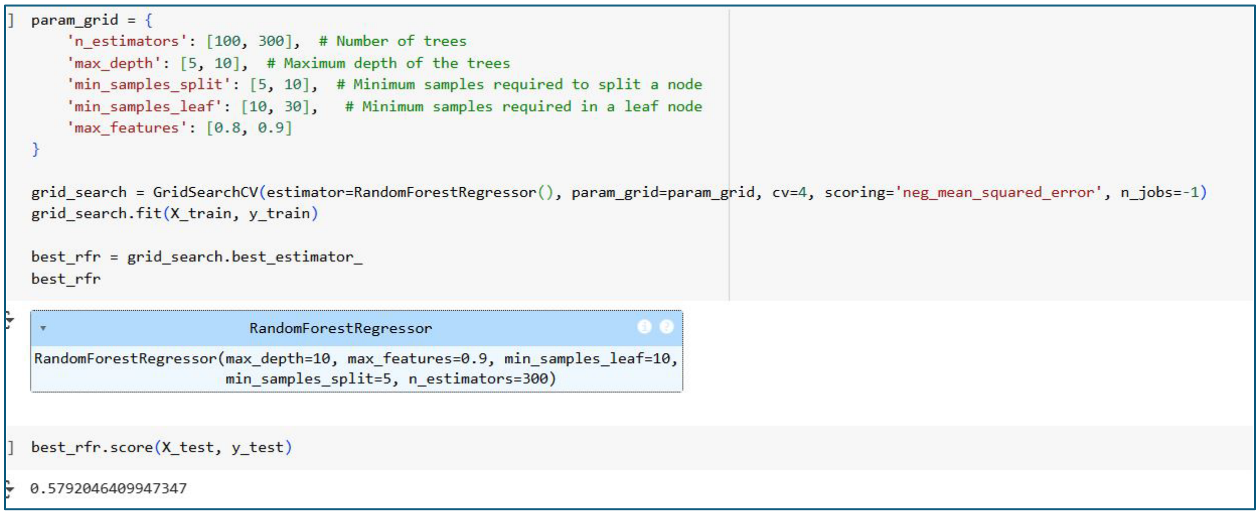
The features selected in each of these methodologies exhibit some variation, attributable to the distinct operational mechanisms inherent to each approach.

RFECV: This is implemented with a decision tree regression. Here, the number of features to select is handed over to the method. It concluded with just some features that have in common with other method's selected features. But the regressor's result was worse than the normal decision tree regressor on the original data. This suggests that while RFECV can identify key features, the SelectKBest method may offer a more balanced approach by considering additional relevant features.

SFS: This is implemented with a linear regression model. It had almost the same result as SelectKBest.

2.2 Tree Ensembles

Random forest regression is initialized and tuned. The best hyper parameters and result achieved was:



```
param_grid = {
    'n_estimators': [100, 300],  # Number of trees
    'max_depth': [5, 10],  # Maximum depth of the trees
    'min_samples_split': [5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [10, 30],   # Minimum samples required in a leaf node
    'max_features': [0.8, 0.9]
}

grid_search = GridSearchCV(estimator=RandomForestRegressor(), param_grid=param_grid, cv=4, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)

best_rfr = grid_search.best_estimator_
best_rfr
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=10, max_features=0.9, min_samples_leaf=10,
                      min_samples_split=5, n_estimators=300)
```

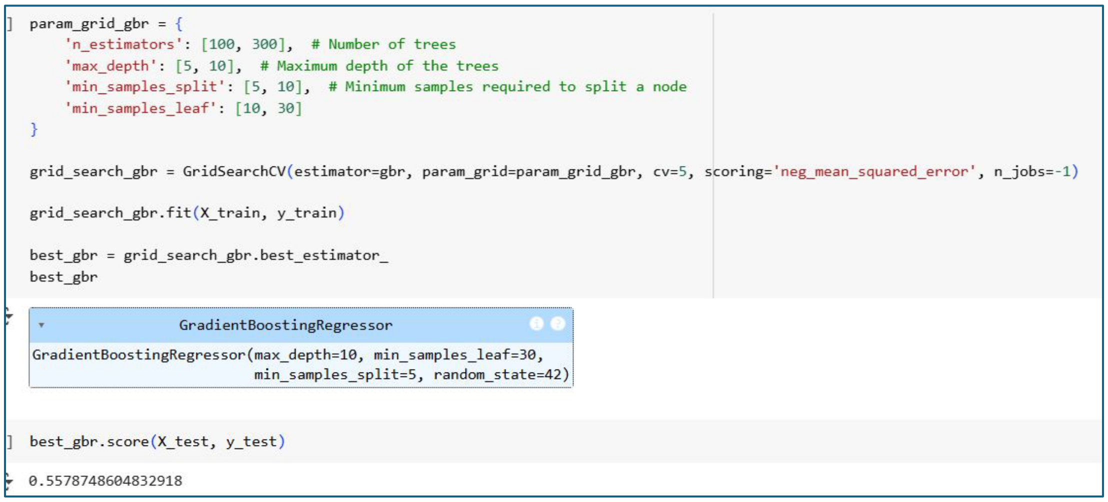
```
best_rfr.score(X_test, y_test)
```

```
0.5792046409947347
```

Figure 2: Tuning and results of random forest regressor

Random forest regressor has the best performance so far between all the models (0.579 score accuracy).

Gradient boosting regressor: the best one has the score of 0.558, which was 2 percent better than the best of random forest regressor, expectedly.



```
param_grid_gbr = {
    'n_estimators': [100, 300],  # Number of trees
    'max_depth': [5, 10],  # Maximum depth of the trees
    'min_samples_split': [5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [10, 30]
}

grid_search_gbr = GridSearchCV(estimator=gbr, param_grid=param_grid_gbr, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

grid_search_gbr.fit(X_train, y_train)

best_gbr = grid_search_gbr.best_estimator_
best_gbr
```

```
GradientBoostingRegressor
GradientBoostingRegressor(max_depth=10, min_samples_leaf=30,
                          min_samples_split=5, random_state=42)
```

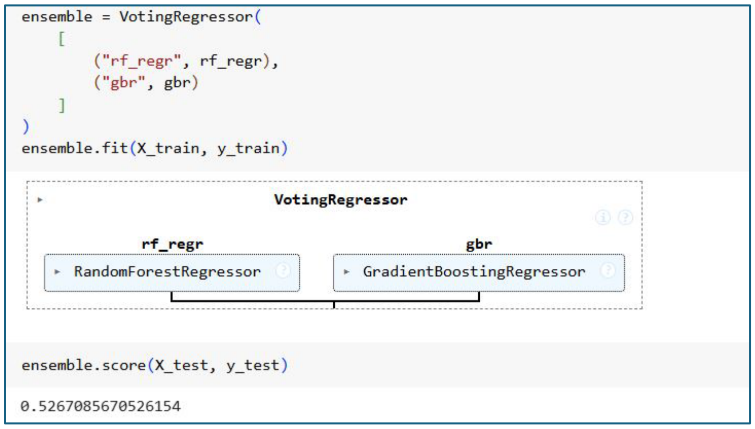
```
best_gbr.score(X_test, y_test)
```

```
0.5578748604832918
```

Figure 3: Tuning and results of gradient boosting regressor

2.3 Ensemble of Tree Ensembles

A voting regressor is implemented using previously defined normal versions of random forest regression and gradient boosting regression. As expected, the result of the voting ensemble is the average of the two models employed in this approach.



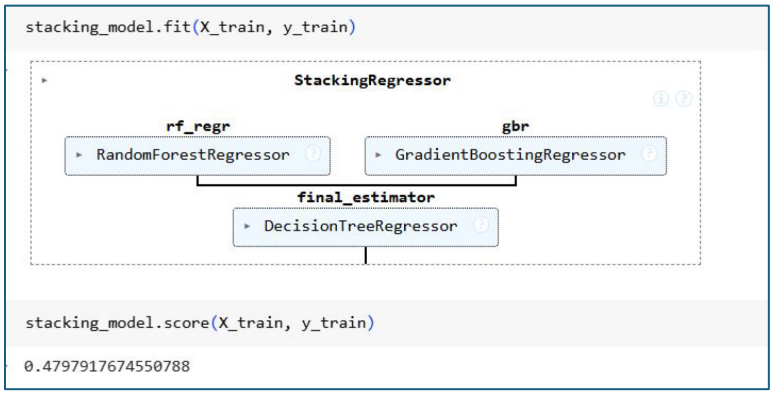
```
ensemble = VotingRegressor(
    [
        ("rf_regr", rf_regr),
        ("gbr", gbr)
    ]
)
ensemble.fit(X_train, y_train)
```

```
ensemble.score(X_test, y_test)
```

```
0.5267085670526154
```

Figure 4: The result of voting regressor

Stacking a regressor utilizing a linear regression model is executed in conjunction with conventional implementations of random forest regression and gradient boosting regression. The predicting power of it is almost the same as the voting regressor.



```
stacking_model.fit(X_train, y_train)
```

```
stacking_model.score(X_train, y_train)
```

```
0.4797917674550788
```

Figure 5: The result of stacking regression

2.4 Feature Importance

The Permutation importance method demonstrates that "mileage" is the most effective feature in our dataset.

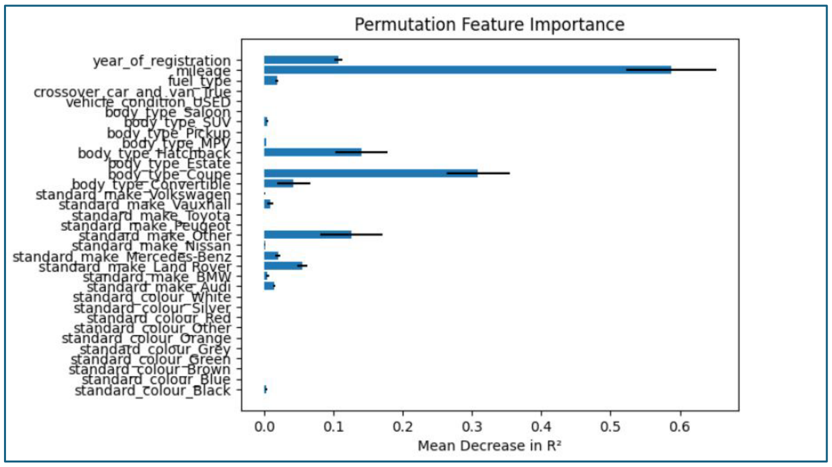
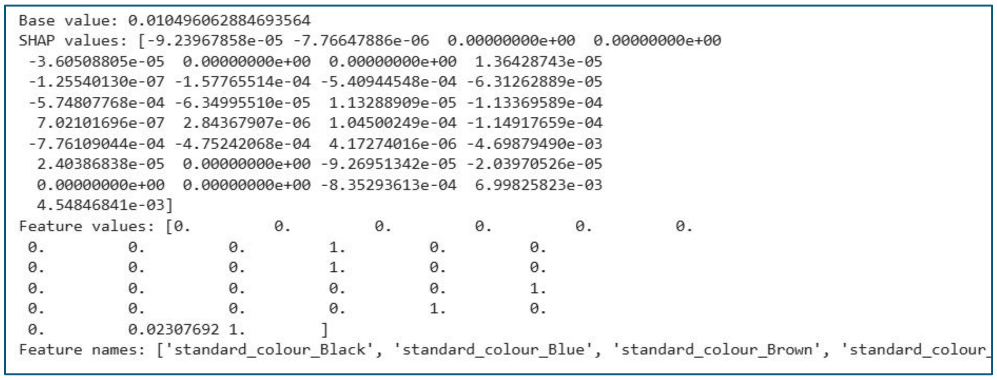


Figure 6: Permutation Importance Result

The SHAP is applied both locally and globally. In the local explanation, a waterfall plot is displayed in the first row to illustrate the impact of each feature on the prediction. Since many feature contributions round to zero, their values are presented with precision.



```
Base value: 0.010496062884693564
SHAP values: [-9.23967858e-05 -7.76647886e-06  0.00000000e+00  0.00000000e+00
 -3.60508805e-05  0.00000000e+00  0.00000000e+00  1.36428743e-05
 -1.25540130e-07 -1.57765514e-04 -5.40944548e-04 -6.31262889e-05
 -5.74807768e-04 -6.34995510e-05  1.13288909e-05 -1.13369589e-04
  7.02101696e-07  2.84367907e-06  1.04500249e-04 -1.14917659e-04
 -7.76109044e-04 -4.75242068e-04  4.17274016e-06 -4.69879490e-03
  2.40386838e-05  0.00000000e+00 -9.26951342e-05 -2.03970526e-05
  0.00000000e+00  0.00000000e+00 -8.35293613e-04  6.99825823e-03
  4.54846841e-03]
Feature values: [0.         0.         0.         0.         0.         0.
 0.         0.         0.         1.         0.         0.
 0.         0.         0.         1.         0.         0.
 0.         0.         0.         0.         0.         1.
 0.         0.         0.         0.         1.         0.
 0.         0.02307692 1.        ]
Feature names: ['standard_colour_Black', 'standard_colour_Blue', 'standard_colour_Brown', 'standard_colour_
```

Figure 7: Local explanation of feature contribution with SHAP

2.5 SHAP/PDP MODEL EXPLANATIONS

The exploration of global model interpretability employed a beeswarm plot, which effectively visualizes the distribution and importance of features within the model. This visualization confirms the findings from the permutation importance analysis, indicating that "mileage" is the paramount feature influencing the predictions. The beeswarm plot distinctly shows that "mileage" not only has the highest variance in SHAP values but also demonstrates a significant number of instances characterized by elevated SHAP values. This suggests a strong and consistent impact of mileage on the model's outputs across the dataset, highlighting its critical role in the predictive analytics conducted.

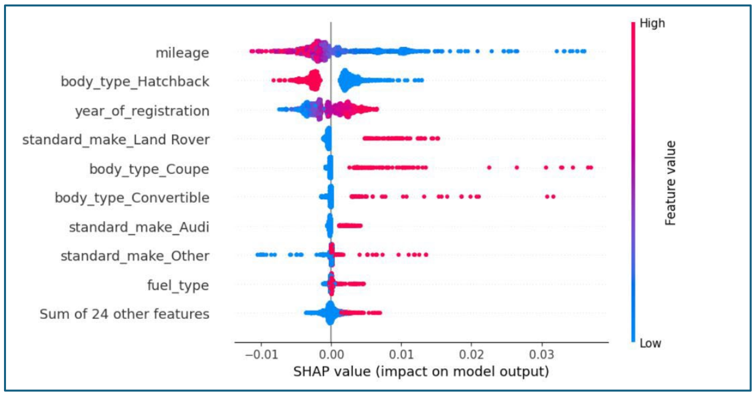


Figure 8: Global explanation of feature contribution with SHAP

PDP and ICE: They are implemented on 'mileage'. What the resulting plots show prove what has been explored so far from the model, such as 'mileage' with high values have less impact on the predictions than 'mileage' with low values. This also makes sense as one of the most important roles in the car market is the low mileage of a car.

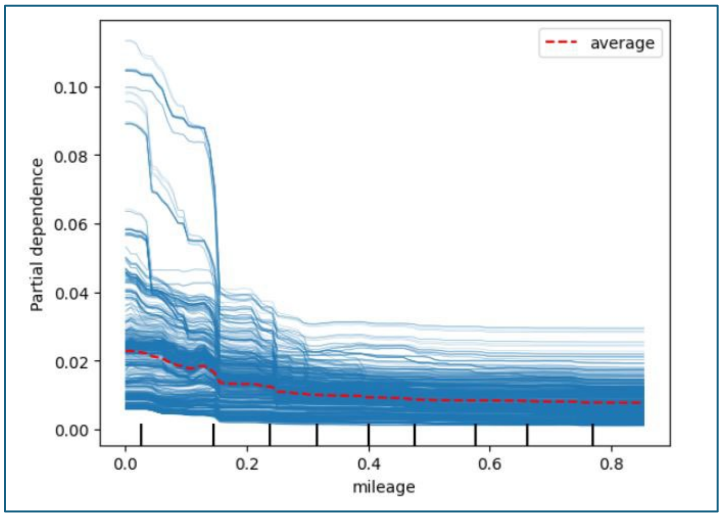


Figure 9: PDP and ICE of 'mileage' feature

3 PART II

3.1 DIMENSIONALITY REDUCTION (LINEAR)

PCA: PCA is implemented with 10 components. In this condition, it keeps 70 percent of variance, which would be enough not to fall into overfitting. The elbow method also proves that. In the scree plot, from the 9th and 10th principal components, the increase in the explained variance becomes too low. Testing the newly generated components, the random forest trained on them to predict the target feature performed much worse than the random forest trained with the original features. The reason is because the dimensions have been reduced, but most of the variance has been kept. So, the random forest has a lower ability to maneuver between features and arrives at the target.

PCA with two components is also implemented on the model to see if the two newly generated features can provide insight into their relationship to the target feature. However, as the distribution of price is mostly around 0 and is too low, helpful insights cannot be inferred from it.

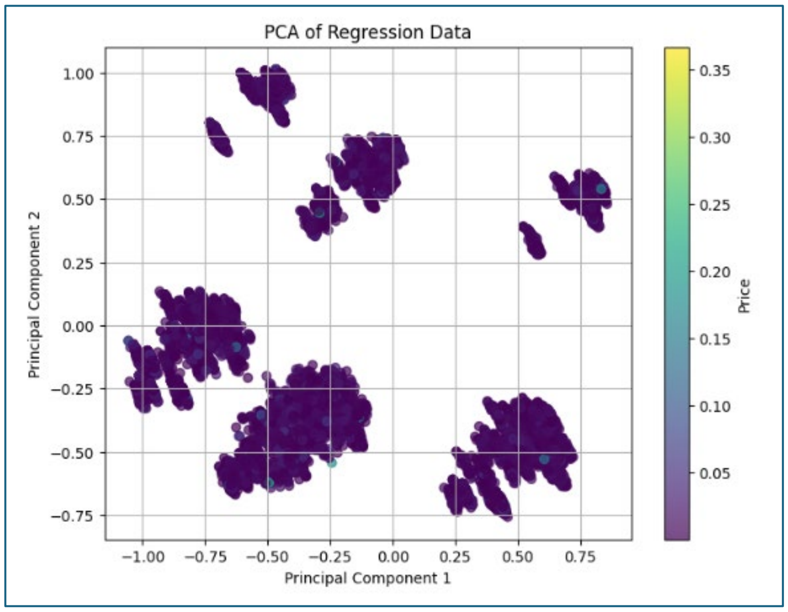


Figure 10: The result of PCA with two components

3.2 Dimensionality Reduction (Non-Linear)

Isomap and T-SNE: The implementation utilizes 15 neighbors. This specific choice is made to preserve the geodesic distances, which reflect meaningful patterns in the data, without introducing too much noise from a larger number of neighbors. Conversely, a smaller number of neighbors may fail to maintain the integrity of patterns present in the global structure. The resulting plot is generated from 300 randomly selected data points. Some patterns can be drawn from the plots.

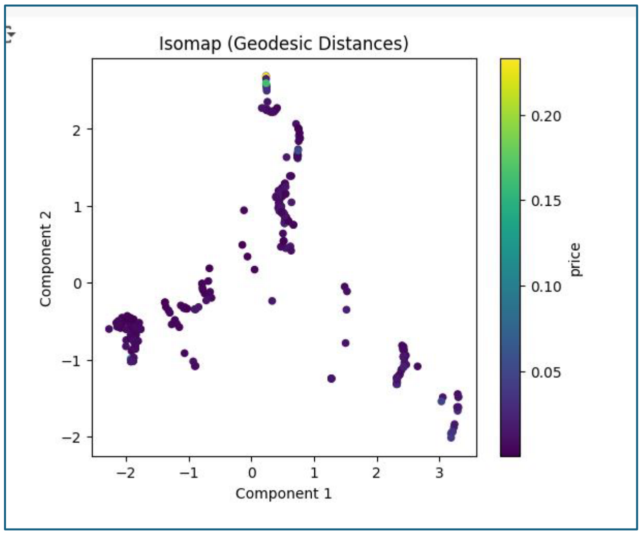


Figure 11: Isomap results

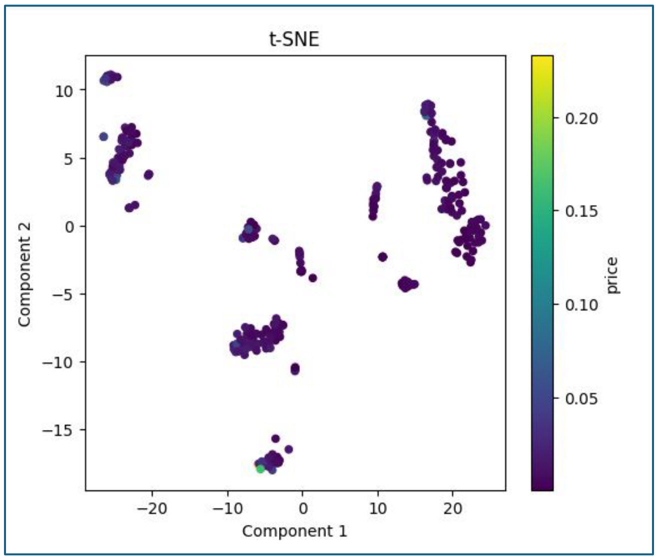


Figure 13: T-SNE results

3.3 Polynomial Regression

Polynomial regression enhances the ability of linear regression models to capture the underlying trends in data by enabling the model to fit more complex relationships. This increased flexibility is reflected in the R^2 score, which indicates the proportion of variance in the dependent variable that can be explained by the independent variables. In this context, the inclusion of polynomial features has resulted in a noteworthy improvement of 0.15 in the R^2 score for the linear regression model.

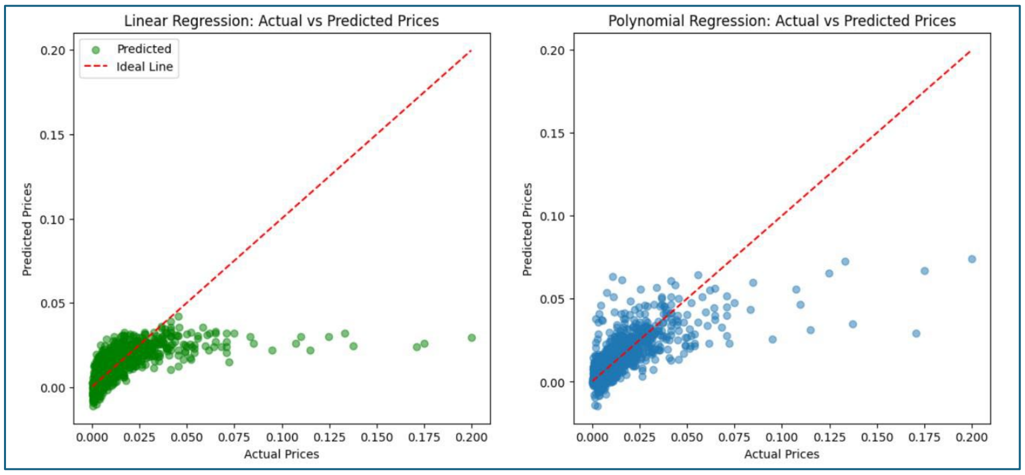


Figure 12: Compare Linear regression with regular and polynomial features

Bayesian Regression: Prior distribution of parameters (alpha and beta) are set to normal with 'mu' and 'sigma' set to 0 and 10, respectively. Based on the target data, these assumptions are modified with samples generated by different combinations of parameters. The trace and distribution plots:

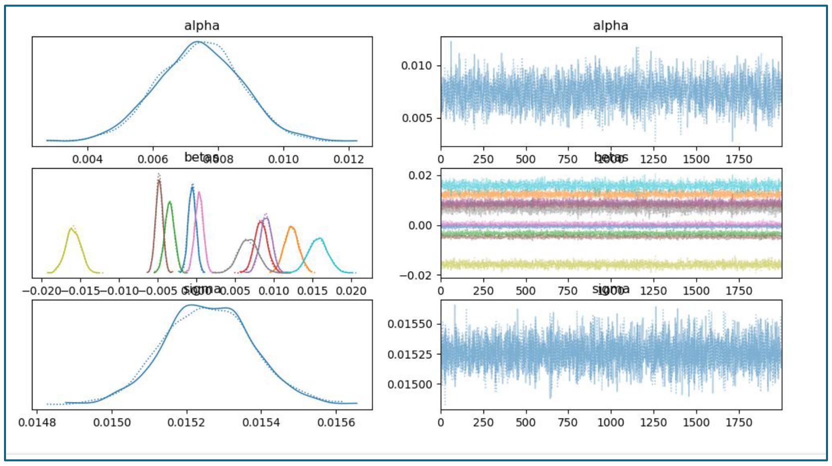


Figure 13: The trace and distribution plots of parameters

Result: The Bayesian regression model performs almost like regular linear, with a slight improvement.

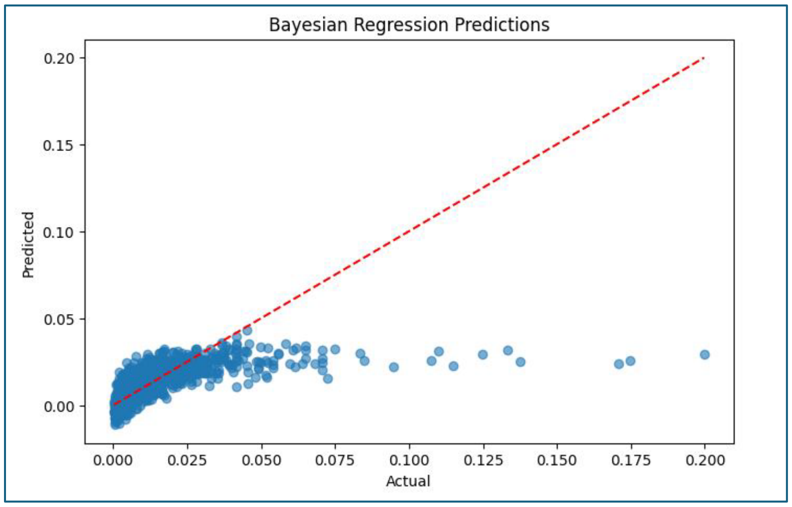


Figure 14: The predictions of Bayesian Regression

3.4 Clustering for Feature Engineering

K-means Clustering: Following the determination of the optimal value for “k,” the resulting clusters were visualized in terms of their corresponding “mileage” and “price” metrics. The resulting scatter plot again proves high mileage results in lower prices, and cars in cluster 0 (red) are more expensive ones.

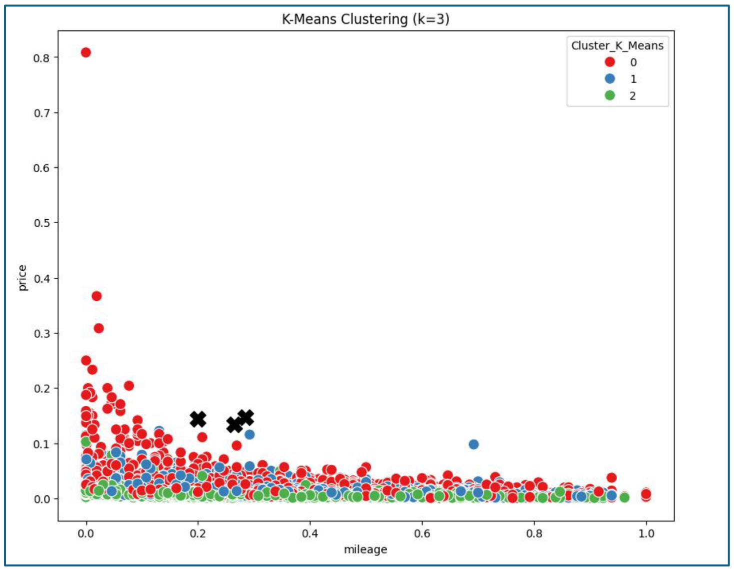


Figure 15: K-means clustering results

Additionally, the inclusion of the clusters column in the dataset did not enhance the model's predictive performance. The random forest algorithm exhibited comparable results with and without the clusters column.

DBSCAN (Density-Based Clustering): Another time elbow method is used to find the optimal value for EPS. Then, the clusters are visualized with their “mileage” and “price” value. Again, the analysis reveals that the clusters produced lack meaningful insights regarding their composition and the associated values.

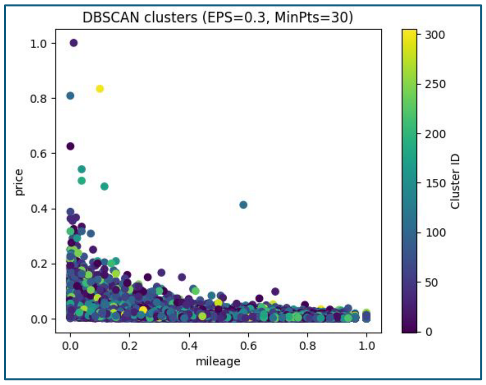


Figure 16: DBSCAN clustering results