Machine Learning Model for Car Price Prediction

Coursework Specification (1CWK100)

6G7V0017

Advanced Machine Learning

Submitted by

Jesty Sebastian (22555483)

1. Data Processing for Machine Learning

The aim of our project is to discover associations and group differences that significantly affect the valuation of cars by using the Car Sale Adverts information given by AutoTrader. It is a .csv file having around 400K rows. The dataset includes a collection of anonymous advertisements for vehicles that include brand, type, colour, mileage, and the selling price.

1.1.Handling missing values, outlier and noise

The first step in the data preprocessing is to fill missing values. For the 'mileage' column, the missing values are filled using the mean value of the column.

For the 'year_of_registration' column, a custom function 'find_year' is defined to extract the year of registration from the 'reg_code' column. If the vehicle condition is not new and the 'reg_code' is a digit, then the year of registration is extracted using a certain logic. Otherwise, the year of registration is left as it is. Missing values are then filled with zero.

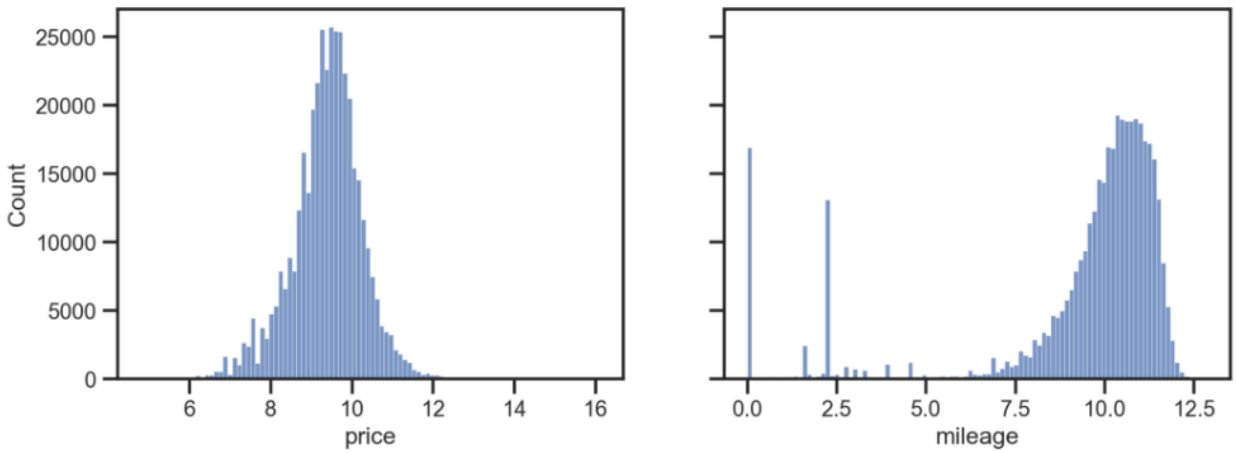
```
# Defining a function to find year_of_registration
def find_year(year_of_registration,reg_code,vehicle_condition):
    if(vehicle_condition!="NEW"):
        if reg_code.isdigit():
            if 0< int(reg_code) < 23 :
                out = int(reg_code)+2000
            elif 23 <= int(reg_code) < 73:
                out = int(reg_code)+1950
            else:
                  out=year_of_registration
        else:
            out=year_of_registration
    else :
            out=year_of_registration
    return out

# Calling the function
car_adv['year_of_registration'] =car_adv.apply(
    lambda x: find_year((x['year_of_registration']),
                        str(x['reg_code']),(x['vehicle_condition'])),
    axis=1)
```

Next, the 'fuel_type', 'standard_colour', and 'body_type' columns are filled with the mode value (most frequently occurring value) of the respective columns.

The categorical columns are then filtered to keep only those categories that have a count of at least 100. This is done to reduce the number of categories and to remove the categories that have very few occurrences in the dataset. This step is important to avoid overfitting and to improve the model's generalization capability.

1.2.Rescaling data:

Several machine learning methods may benefit from rescaling the data. This can be done by either normalizing the data to a particular range, such as [0,1] or [-1,1], or by standardizing the data to



have a zero mean and unit variance. The mileage column is transformed using the natural logarithm function, except for the values equal to 0, which are left unchanged. The price column is also transformed using the natural logarithm function. These transformations can help to improve the performance of machine learning models trained on this dataset, especially if the original data was skewed.

1.3. Automated Preprocessing:

1.3.1. Numeric Transformation

Missing Value Imputation: The missing values in numeric features are filled with the median value of each feature using the SimpleImputer with the strategy="median".

Feature Scaling: The numeric features are standardized using the StandardScaler, which subtracts the mean and scales the features to have unit variance.

1.3.2. Categorical Transformation:

Missing Value Imputation: The missing values in the categorical features 'standard_colour', 'standard_make', 'body_type', 'fuel_type', and ‘vehicle_condition’ are filled with the most frequent value using the SimpleImputer with strategy="most_frequent".

1.4. Splitting data:

A fraction (10% in this case) of rows from the car_adv dataset is randomly selected using the sample function. The random_state=42 ensures reproducibility by fixing the random seed. Features (X) and the target variable (y) are separated from the sampled data. The train_test_split function from scikit-learn is used to split X and y into training and testing sets. The test_size parameter is set to 0.25, indicating that 25% of the data will be used for testing, and the random_state=0 ensures reproducibility by fixing the random seed.

2. Feature Engineering

2.1. Manual Feature Engineering

Age of a car is an important feature to predict its price. So a function is defined to calculate the age of a car based on its year of registration. It then applied to the 'year_of_registration' column of the car_adv dataset using a lambda function and assigns the resulting values to a new column 'age'.

2.2. Automated Feature Engineering

Polynomial Features: Polynomial features up to degree 2 are generated using the PolynomialFeatures transformer. This allows capturing non-linear relationships between the numeric features and the target variable.

One-Hot Encoding: The OneHotEncoder is applied to the vehicle_condition feature to convert it into binary columns representing each unique category. Unknown categories are ignored, and if the feature is binary, one of the binary columns is dropped.

Target Encoding: The TargetEncoder from the category_encoders library is applied to the categorical features 'standard_colour', 'standard_make', 'body_type', and 'fuel_type'. Target encoding replaces each category with the average target value for that category. It helps capture the relationship between the categorical features and the target variable. A regularization parameter (smoothing=10) is used to prevent overfitting.

3. Feature Selection and Dimensionality Reduction

3.1. SelectKBest

SelectKBest is a feature selection method in scikit-learn that selects the top k features based on a specified scoring function. It is commonly used to reduce the dimensionality of the feature space by selecting the most informative features for a given task.

```
selector = SelectKBest(f_regression, k=8).fit(X_train_transformed, y_train)
X_sel = selector.transform(X_train_transformed)
```

```
selector.get_feature_names_out()
```

```
array(['mileage', 'age', 'mileage^2', 'vehicle_condition_USED',
       'standard_colour', 'standard_make', 'body_type', 'fuel_type'],
      dtype=object)
```

The best eight features are selected using SelectKBest.

The R-squared score indicates the proportion of the variance in the target variable that can be explained by the selected features and the regression model. Here, the R-squared score is approximately 0.7957, suggesting a reasonably good fit of the model to the test data. RMSE is a measure of the average prediction error, with lower values indicating better predictive performance. Here, the RMSE is approximately 0.3872.

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

```
0.7957529844958138
```

```
rmse(y_test, regr_pipe.predict(X_test))
```

```
0.38721915166128773
```

3.2. Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a feature selection method that aims to identify the most relevant features for a given machine learning model. It works by recursively eliminating features from the feature set and evaluating the model's performance after each elimination.

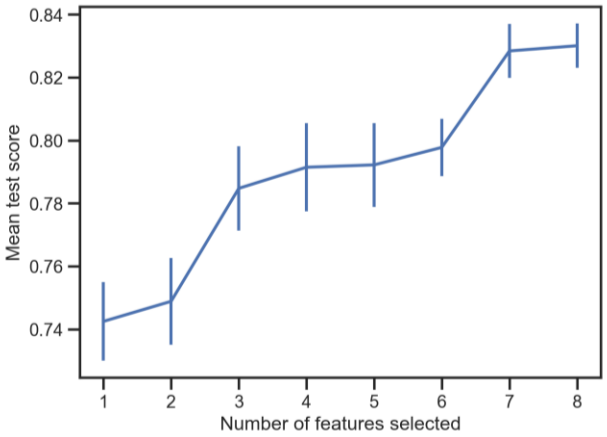
```
model = Ridge()
rfe_selector = RFECV(model, min_features_to_select=3, step=1, cv=5)
```

```
rfe_selector.get_feature_names_out()
```

```
array(['mileage', 'age', 'mileage^2', 'mileage age', 'age^2',
       'vehicle_condition_USED', 'standard_colour', 'standard_make',
       'body_type', 'fuel_type'], dtype=object)
```

The algorithm determined that all the features are important for the model's performance. So, RFECV does not remove any features and considers all the available features as valuable.

The figure shows a plot with error bars representing the mean test scores obtained during the RFECV process. It provides insights into the model's performance as the number of selected features changes. It is clear from the figure that the performance of the model is increasing by selecting each of the features. That means all the features are important.



The obtained R-squared score is approximately 0.8259, indicating a good fit of the model to the test data. The RMSE is approximately 0.3575, indicating the average prediction error of the model on the test data.

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

```
0.8258622168489789
```

```
rmse(y_test, regr_pipe.predict(X_test))
```

```
0.35754069106236547
```

3.3.Sequential Feature Selection (SFS) (Forward/Backward)

Sequential Feature Selection (SFS) is a feature selection technique that evaluates subsets of features by iteratively adding or removing features based on a specified criterion. SFS with forward and backward directions are performed using the Ridge regression model.

The best eight features are selected using SFS. Both forward and backward methods give the

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

```
0.8258509565571339
```

```
rmse(y_test, regr_pipe.predict(X_test))
```

```
0.3575522507232658
```

same set of features. The R-squared value obtained is approximately 0.8258, indicating that the fitted model explains about 82.58% of the variance in the test data. The RMSE value obtained is approximately 0.3576, which indicates that, on average, the model's predictions are off by approximately 0.3576 units from the actual target values in the test data.

```
sfs_forward = SequentialFeatureSelector(
    Ridge(), n_features_to_select=8, direction="forward"
).fit(X_train_transformed, y_train)
```

```
sfs_forward.get_feature_names_out()
```

```
array(['mileage', 'age', 'mileage^2', 'mileage age', 'age^2',
       'standard_make', 'body_type', 'fuel_type'], dtype=object)
```

```
sfs_backward = SequentialFeatureSelector(
    Ridge(), n_features_to_select=8, direction="backward"
).fit(X_train_transformed, y_train)
```

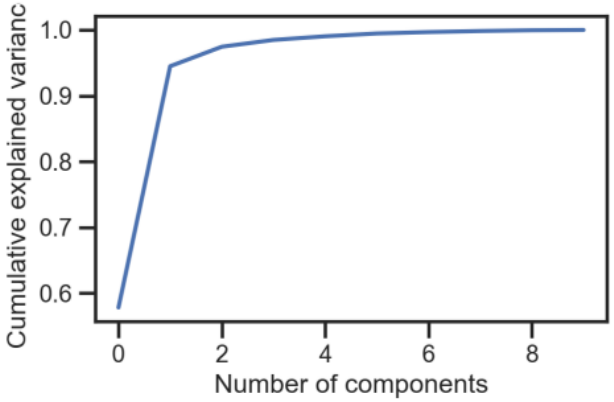
```
sfs_backward.get_feature_names_out()
```

```
array(['mileage', 'age', 'mileage^2', 'mileage age', 'age^2',
       'standard_make', 'body_type', 'fuel_type'], dtype=object)
```

3.4. Dimensionality Reduction

The basic objective of dimensionality reduction is to escape the constraints and difficulties that come with working with high-dimensional data. High-dimensional data can be challenging to visualise, comprehend, and analyse, and it can overfit machine learning models due to computational inefficiencies. Dimensionality reduction techniques address these problems by lowering the number of features or variables in the data while keeping as much relevant data as feasible.

Principal Component Analysis (PCA) is used for dimensionality reduction. It transforms high-dimensional dataset into a lower-dimensional space while preserving the most important patterns and relationships in the data.



The figure shows the cumulative explained variance plot. It is a visual representation of the cumulative contribution of each principal component to the total variance in the data. It helps in determining the number of principal components needed to capture a significant portion of the variance. It is clear that maximum variance is determined by four principal components

Four principal components are calculated based on the covariance matrix of the data.

```
# keep the first four principal components of the data
pca = PCA(n_components=4)
# fit PCA model
pca.fit(X_sel_rfe)

# transform data onto the first two principal components
X_pca = pca.transform(X_sel_rfe)
print("Original shape: {}".format(str(X_sel_rfe.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))
```

```
Original shape: (30077, 10)
Reduced shape: (30077, 4)
```

The data which is transformed with Recursive Feature Elimination is given to PCA for dimensionality reduction.

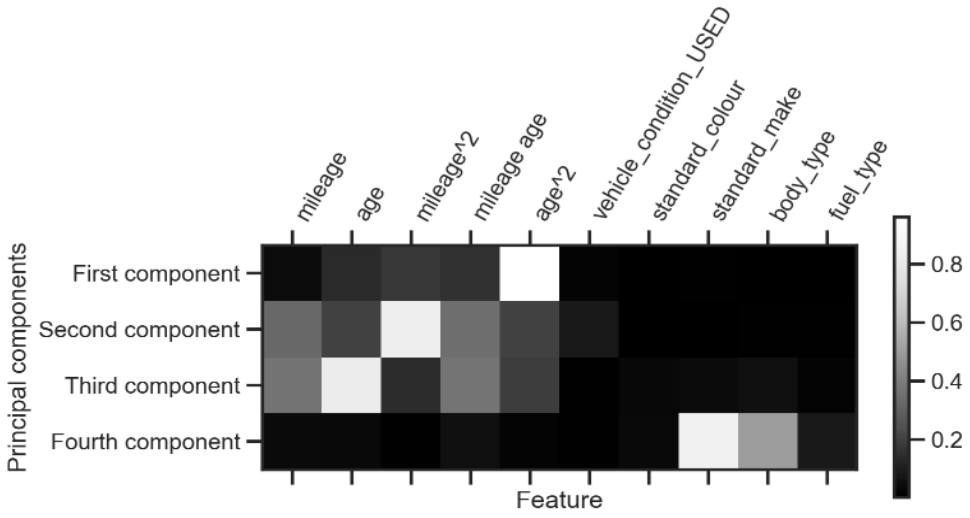


Figure shows the influence of the features in the PCA components.

After adding the PCA components to the proposed model pipeline, the performance is reduced slightly. So PCA is not selected for model building.

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

```
0.820015106635343
```

```
rmse(y_test, regr_pipe.predict(X_test))
```

```
0.3634937911529965
```

4. Model Building

The create_regr_pipe function is designed to generate a pipeline for regression modeling. This function takes two arguments: est, representing the regression estimator or model to be utilized, and X, which represents the input data.

```
def create_regr_pipe(est, X):
    """ """
    regr_pipe = Pipeline(
        steps=[
        ("preprocessor", preprocessor),
        ("featsel", RFECV(model, step=1, cv=5)),
        ("regr", est)
        ]
    ).set_output(transform="pandas")

    return regr_pipe
```

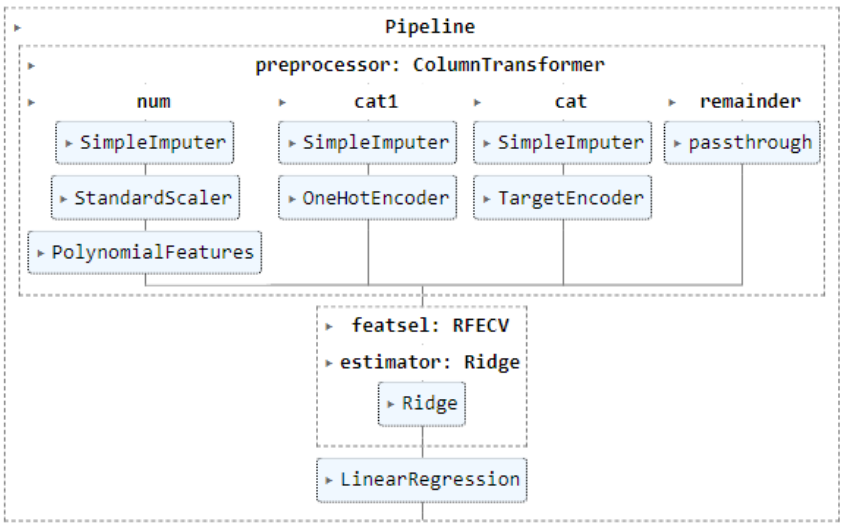
The preprocessor handles necessary data preprocessing tasks such as imputation, encoding, and scaling. Recursive Feature Elimination with Cross-Validation (RFECV recursively eliminates features to determine the most relevant ones for predicting the target variable. The "regr" step represents the regression model specified by the est argument. It can be any valid regression estimator, such as Ridge regression, Linear regression, or any other suitable model.

4.1.A Linear Model

Linear regression is a common statistical modelling technique for forecasting a continuous target variable based on one or more predictor variables. It assumes that the predictors and the target variable have a linear relationship. The R-squared score is approximately 0.8254, which suggests that around 82.54% of the variance in the target variable is explained by the linear regression model. The RMSE is approximately 0.3580, which means, on average, the linear regression model's predictions deviate from the actual values by approximately 0.3580 units.

```
lr = create_regr_pipe(LinearRegression(), X_train)
```

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




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

```
0.8253956452906963
```

```
rmse(y_test, lr.predict(X_test))
```

```
0.35802828189341207
```

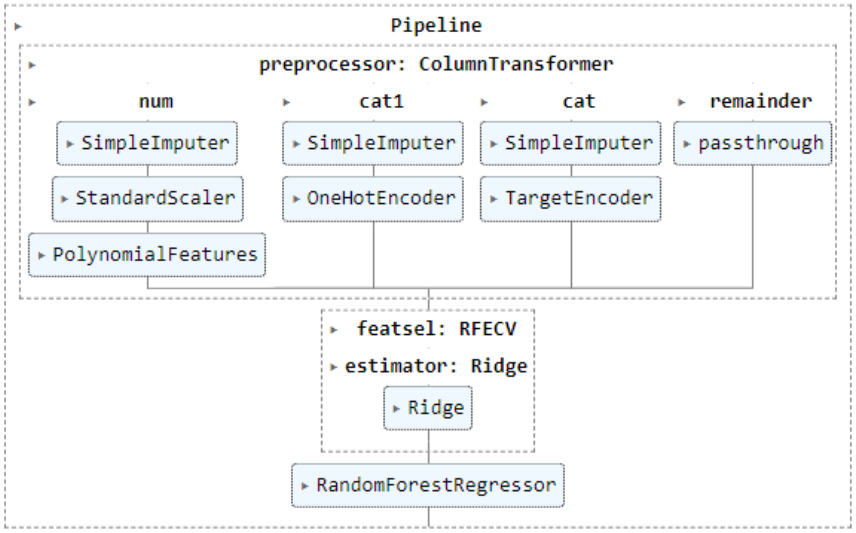
4.2.A Random Forest Model

The Random Forest Regressor is a powerful regression technique that excels at handling complex correlations in data because of its adaptability, robustness, and speed.

Specific hyperparameter values are used to generate the Random Forest Regressor (rfr). The training data (X_train and Y_train) are then used to train it. The test data are used to assess the model's performance; the result is

```
rfr = create_regr_pipe(RandomForestRegressor(
    max_depth=8, min_samples_split=15, min_samples_leaf=8, n_estimators=400
), X_train)
```

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



an R-squared score of roughly 0.8690, suggesting a satisfactory match to the test data. The average prediction error of the model is reportedly quite low, as indicated by the Root Mean Squared Error (RMSE), which is roughly 0.3101.

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

```
0.8690434667196603
```

```
rmse(y_test, rfr.predict(X_test))
```

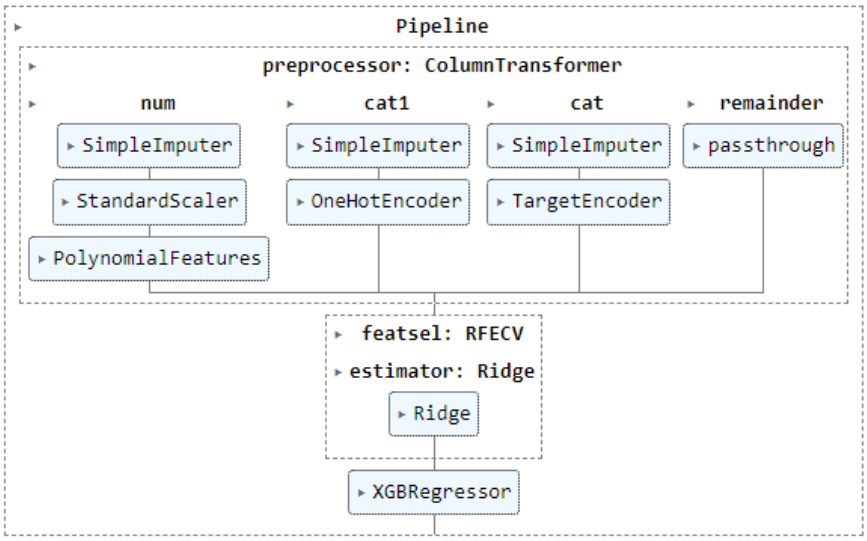
```
0.3100577237670887
```

4.3.A Boosted Tree

The XGBoost Regressor is a popular machine learning algorithm based on the gradient boosting framework. It stands for eXtreme Gradient Boosting, which refers to the algorithm's ability to leverage gradient boosting techniques to optimize the model's performance. The default hyperparameter values are used to generate the XGBoost Regressor (xgb). On the training data (X_train and Y_train), it is then trained. Using the test data to assess the model's performance, an R-squared score of roughly 0.8962 is obtained, suggesting a satisfactory match to the test data. The average prediction error of the model is thought to be quite low given that the Root Mean Squared Error (RMSE) is close to 0.2761.

```
xgb = create_regr_pipe(XGBRegressor(), X_train)
```

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




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

```
0.8961809151125124
```

```
rmse(y_test, xgb.predict(X_test))
```

```
0.27606895273299
```

4.4.An Averager/Voter/Stacker Ensemble

4.4.1. Voting Regressor

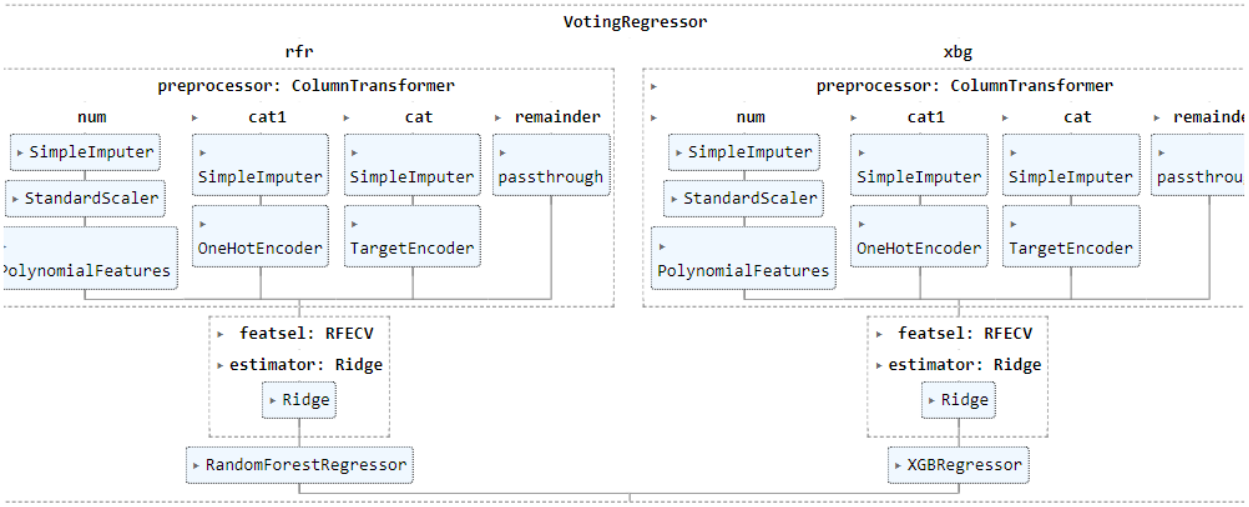
An ensemble model is created by combining RandomForestRegressor (rfr) and XGBRegressor (xgb) using the VotingRegressor class, which combines the predictions of multiple individual regression models by averaging their outputs. Both models are trained on the training data (X_train and y_train). After fitting the individual models and the ensemble model, their performance is evaluated using the test data (X_test and y_test).

```
ensembled = [rfr, xgb ]
```

```
for est in ensembled:
    est.fit(X_train, y_train)
```

Voter

```
voter_ensemble = VotingRegressor(
    [
        ("rfr", rfr),
        ("xbg", xgb)
    ]
)
voter_ensemble.fit(X_train, y_train)
```



The ensemble model achieves an R-squared score of approximately 0.8904, indicating a good fit to the test data. The Root Mean Squared Error (RMSE) is approximately 0.2836, suggesting that the ensemble model has a relatively low average prediction error.

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

```
0.8904402295016578
```

```
rmse(y_test, voter_ensemble.predict(X_test))
```

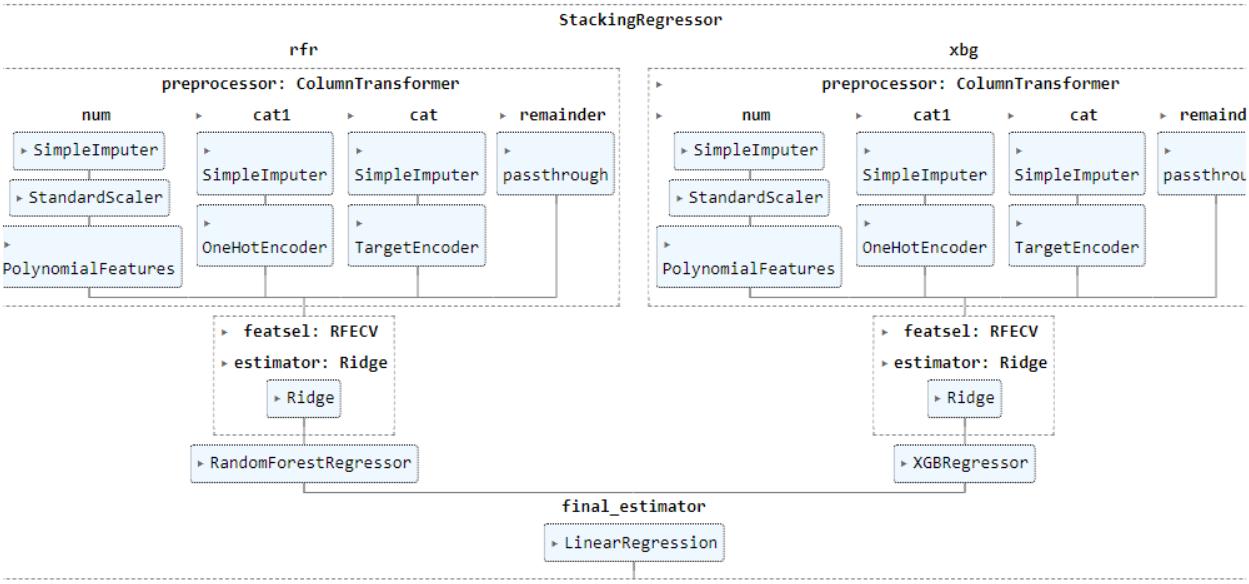
```
0.28359888983423054
```

4.4.2. Stacking Regressor

A stacked ensemble model is created using the StackingRegressor class. The final predictions of these models are combined using a final estimator, which is a LinearRegression model in this case. Stacked ensembles work by training multiple base

```
stacker_ensemble = StackingRegressor(
    [
        ("rfr", rfr),
        ("xbg", xgb)
    ], final_estimator=LinearRegression()
)
stacker_ensemble.fit(X_train, y_train)
```

models (RandomForestRegressor and XGBRegressor) on the training data and then combining their predictions using a meta-model (LinearRegression).



The ensemble model is trained on the training data (X_train and y_train), and its performance is evaluated using the test data (X_test and y_test).

The stacked ensemble model achieves an R-squared score of approximately 0.8963, indicating a good fit to the test data. The Root Mean Squared Error (RMSE) is approximately 0.2759, suggesting that the ensemble model has a relatively low average prediction error.

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

```
0.8962702462753596
```

```
rmse(y_test, stacker_ensemble.predict(X_test))
```

```
0.27595015536593676
```

4.4.3. Grid Search, and Model Ranking and Selection

Best parameters for Linear Model

```
print("Best hyperparameters: ", lr_grid_search.best_params_)
print("Best cross-validation score: ", lr_grid_search.best_score_)
```

```
Best hyperparameters:  {'regr__fit_intercept': True, 'regr__n_jobs': -3}
Best cross-validation score:  -0.3550577556092246
```

Best parameters for Random Forest Model

```
print("Best hyperparameters: ", rfr_grid_search.best_params_)
print("Best cross-validation score: ", rfr_grid_search.best_score_)
```

```
Best hyperparameters:  {'regr__max_depth': 20}
Best cross-validation score:  -0.29812243595248683
```

Best parameters for XG Boost Model

```
print("Best hyperparameters: ", xgb_grid_search.best_params_)
print("Best cross-validation score: ", xgb_grid_search.best_score_)
```

```
Best hyperparameters:  {'regr__alpha': 1.0}
Best cross-validation score:  -0.2848679567074721
```

Comparison of R-squared Score & RMSE

Model	R-squared Score	RMSE
Linear Model	0.8254	0.3580
Random Forest Model	0.8690	0.3101
XG Boost Model	0.8962	0.2761
Voter Ensemble Model	0.8904	0.2836
Stacker Ensemble Model	0.8963	0.2759

The cross-validated RMSE of each model is calculated using the cross_val_score function with 5-fold cross-validation and 'neg_root_mean_squared_error' as the scoring metric. Based on the R-squared values, it appears that the best model in this case is the stacked ensemble (0.8962702462753596) followed closely by the XGBoost model (0.8961809151125124).

However, it's also worth considering the root mean squared error (RMSE) values and cross-validation scores. The stacked ensemble has the lowest RMSE on the test set (0.27595015536593676), followed by the XGBoost model (0.27606895273299). The cross-validation scores also show that the XGBoost model and the stacked ensemble perform similarly and outperform the Random Forest model.

So overall, it seems that the XGBoost model or the stacked ensemble would be the best choice for this particular dataset and problem.

5. Model Evaluation and Analysis

Model evaluation and analysis involve assessing the performance and analyzing the results of a predictive model. Cross-validation is used to estimate the model's performance on unseen data. The dataset is splitted into multiple folds, then the model is trained on a subset of folds, and evaluated it on the remaining fold. This helps to assess the model's generalization ability and reduces the impact of data variability.

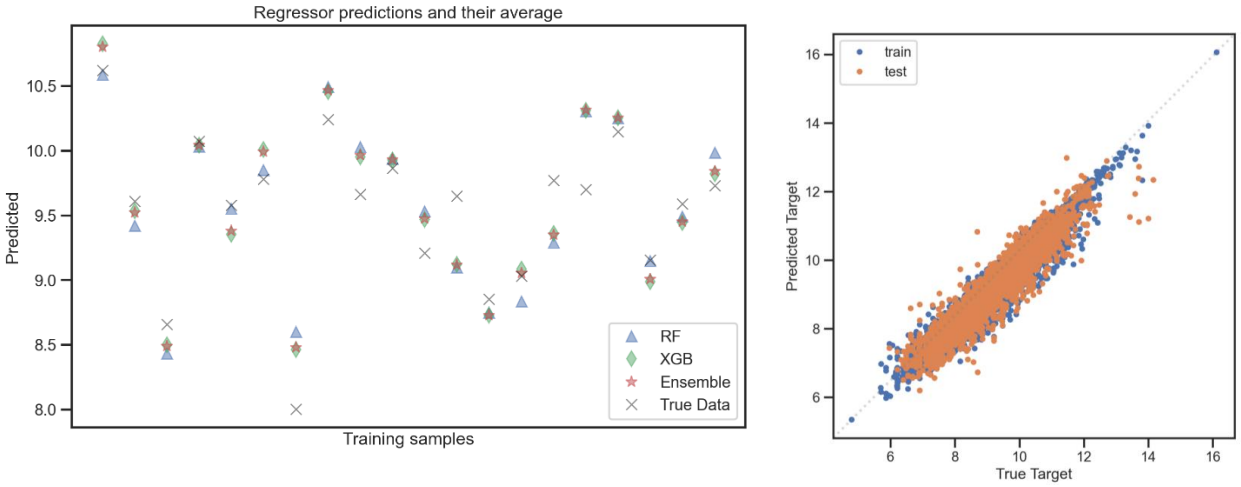
5.1. Overall Performance with Cross-Validation

Model	Cross_val_score (mean)	Cross_val_score (sd)
XG Boost Model	0.2832	0.0113
Stacker Ensemble Model	0.2849	0.0124

XG Boost Model: The XG Boost model has a cross-validation score mean of 0.2832 with a standard deviation of 0.0113. This indicates that, on average, the XG Boost model performs reasonably well in predicting the target variable. The small standard deviation suggests that the model's performance is consistent across different folds of the cross-validation.

Stacker Ensemble Model: The stacker ensemble model has a cross-validation score mean of 0.2849 with a standard deviation of 0.0124. The mean score is slightly higher than the XG Boost model, indicating a slightly better performance on average. The standard deviation suggests that the ensemble model's performance may vary across different cross-validation folds.

5.2. True vs Predicted Analysis

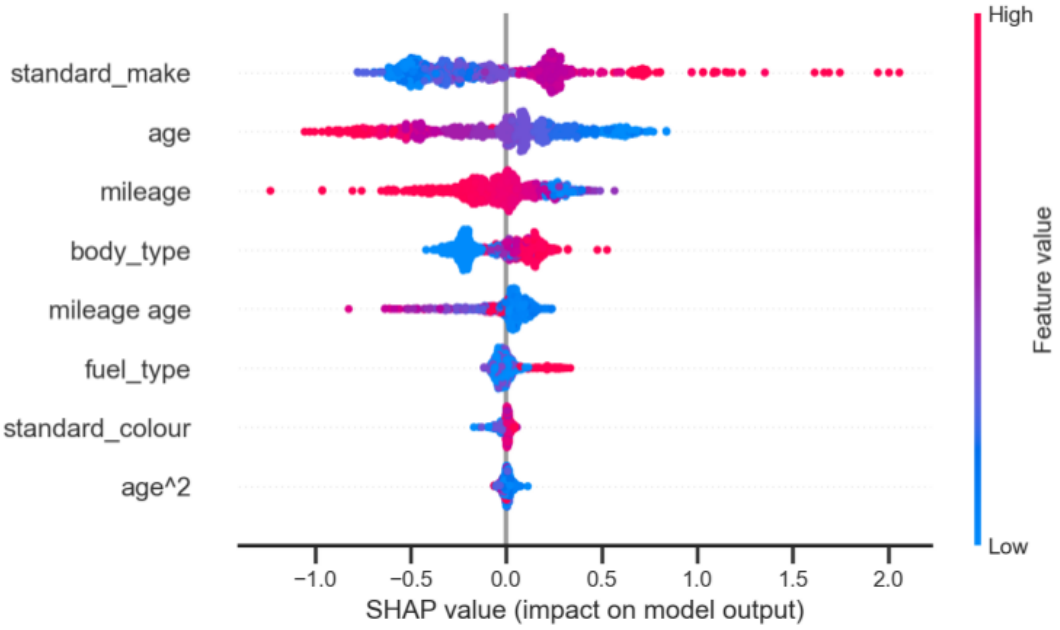


The first plot visualizes the predictions of the three regression models, along with the true data values, for the first 20 samples. The second plot shows the scatter points of the true target values versus the predicted target values for both the training and testing data. The diagonal line represents the ideal case where the predicted values perfectly align with the true values. It is clear that the predictions and true values are almost matching.

5.3. Global and Local Explanations with SHAP

SHAP (SHapley Additive exPlanations) is a powerful technique for providing both global and local explanations of machine learning models.

Global SHAP



Global SHAP values measure the average contribution of a feature across every possible feature combination. From figure we can assume that standard_make, age and mileage contributes more for the prediction.

Local SHAP

Local SHAP values quantify the contribution of each feature for a specific instance's prediction.

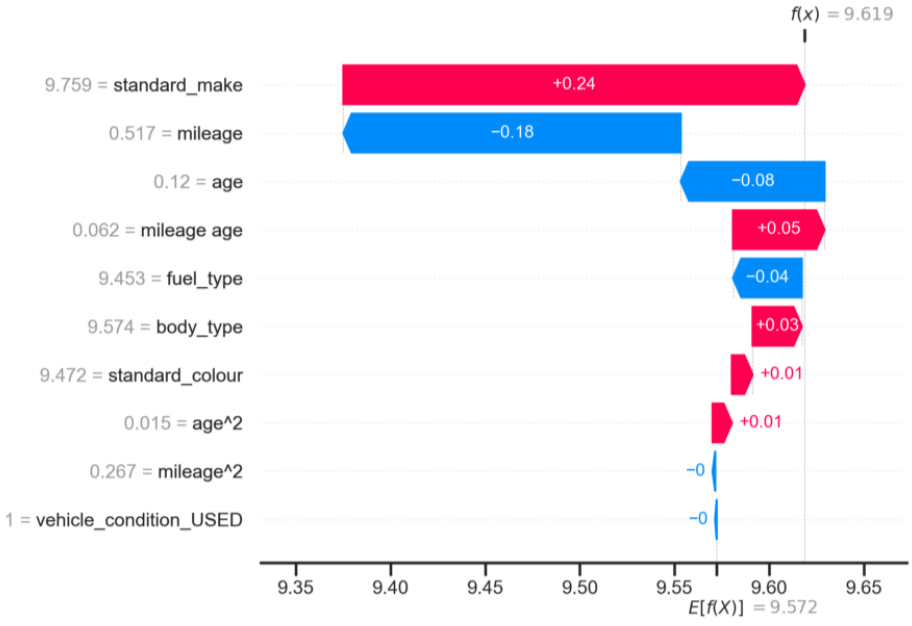


Figure shows the local SHAP of an instance in the dataset. Here stsndard_make, mileage-age, body_type, standard_color and age^2 contributes positively while mileage, age and fuel_type contributes negatively for this instance.

5.4.Partial Dependency Plots

Individual Conditional Expectation (ICE)

Individual Conditional Expectation (ICE) is a technique used for interpreting and visualizing the relationship between a feature and the target variable in a machine learning model. It allows us to understand how the target variable's expected value changes as a specific feature varies while holding other features fixed.

ICE - Mileage

The minimum mileage is 0.0 and maximum is 12.86876061860541. Synthetic data is created according to this values.

	standard_colour	standard_make	vehicle_condition	body_type	fuel_type	age	mileage
0	Bronze	Peugeot	USED	SUV	Diesel	5.0	0.000000
1	Bronze	Peugeot	USED	SUV	Diesel	5.0	0.129987
2	Bronze	Peugeot	USED	SUV	Diesel	5.0	0.259975
3	Bronze	Peugeot	USED	SUV	Diesel	5.0	0.389962
4	Bronze	Peugeot	USED	SUV	Diesel	5.0	0.519950
...	...	...	...	...	...	...	...
95	Bronze	Peugeot	USED	SUV	Diesel	5.0	12.348811
96	Bronze	Peugeot	USED	SUV	Diesel	5.0	12.478798
97	Bronze	Peugeot	USED	SUV	Diesel	5.0	12.608786
98	Bronze	Peugeot	USED	SUV	Diesel	5.0	12.738773
99	Bronze	Peugeot	USED	SUV	Diesel	5.0	12.868761

```
ax = sns.lineplot(x=synth_df['mileage'], y=pred);
ax.set_ylim(0, 20);
```

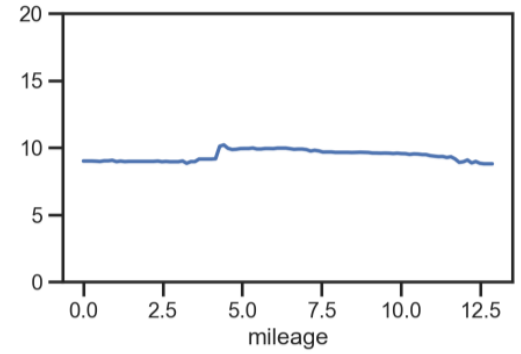


Figure clearly shows that the target value is somewhat decreasing if the mileage varies while all other aspects are kept constant. The negative relationship between mileage and car price in the ICE plot aligns with the general understanding of how mileage impacts the value of a vehicle. Higher mileage often implies more usage, which can lead to increased wear and tear, potential mechanical issues, and a perceived decrease in the car's reliability. Consequently, buyers may be less willing to pay higher prices for cars with higher mileage.

ICE - Age

Minimum age is 0 and maximum is 66. Synthetic data is created according to this values.

	mileage	standard_colour	standard_make	vehicle_condition	body_type	fuel_type	age
0	9.15451	Bronze	Peugeot	USED	SUV	Diesel	0.000000
1	9.15451	Bronze	Peugeot	USED	SUV	Diesel	0.666667
2	9.15451	Bronze	Peugeot	USED	SUV	Diesel	1.333333
3	9.15451	Bronze	Peugeot	USED	SUV	Diesel	2.000000
4	9.15451	Bronze	Peugeot	USED	SUV	Diesel	2.666667
...	...	...	...	...	...	...	...
95	9.15451	Bronze	Peugeot	USED	SUV	Diesel	63.333333
96	9.15451	Bronze	Peugeot	USED	SUV	Diesel	64.000000
97	9.15451	Bronze	Peugeot	USED	SUV	Diesel	64.666667
98	9.15451	Bronze	Peugeot	USED	SUV	Diesel	65.333333
99	9.15451	Bronze	Peugeot	USED	SUV	Diesel	66.000000

```
ax = sns.lineplot(x=synth_df['age'], y=pred);
ax.set_ylim(0, 20);
```

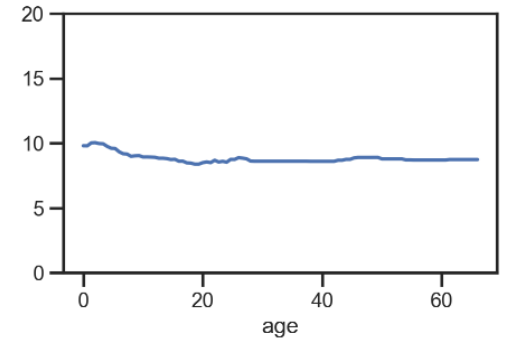
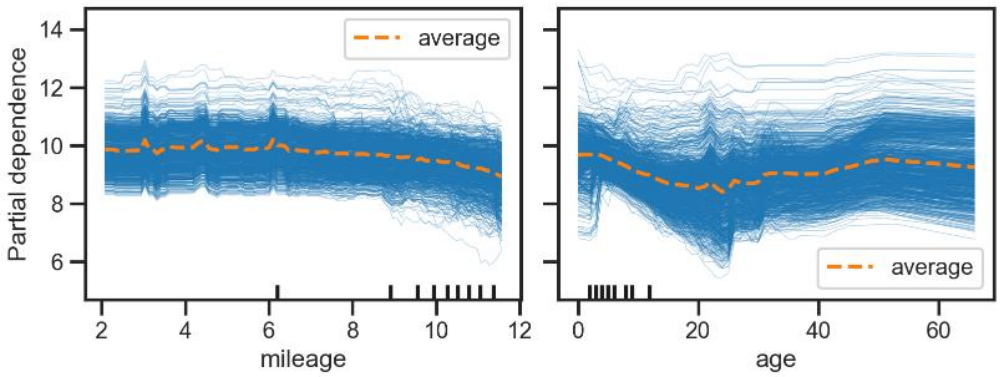
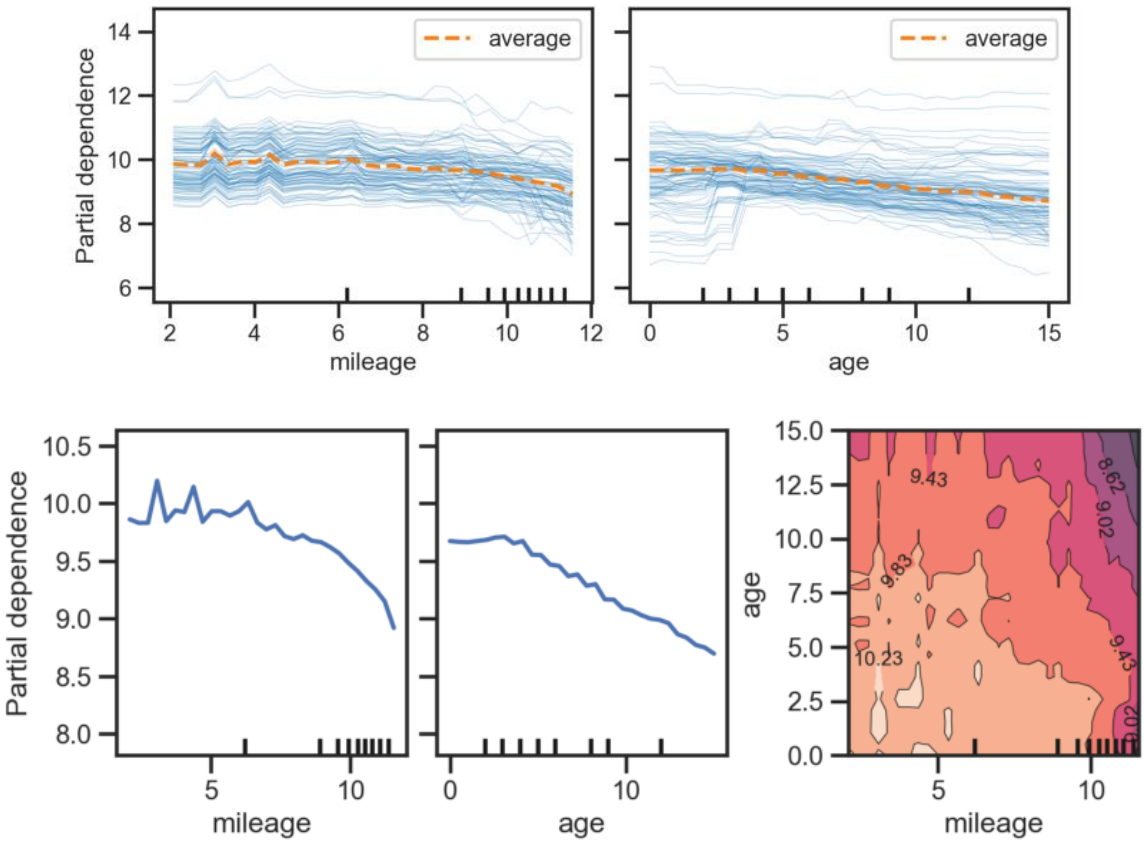


Figure clearly shows that the target value is somewhat decreasing if the age varies while all other aspects are kept constant. The negative relationship between age and car price in the ICE plot aligns with the general understanding of how age affects the value of a vehicle. As cars get older, they typically experience more wear and tear, potential mechanical issues, and a decrease in overall reliability. These factors can contribute to a decrease in the perceived value of the car, leading to lower prices.

Partial Dependency Plot (PDP)

Partial Dependency Plot (PDP) is a technique used for interpreting the relationship between a feature and the target variable in a machine learning model. It allows us to understand how the target variable's average value changes as a specific feature varies while holding other features fixed.





When analyzing the PDP plot shown above, it is evident that the average price of the car decreases as these features change while other features are held constant, it implies the following:

Age: The PDP plot shows how the average price of the car changes as the age of the car varies while keeping mileage and other features fixed. If the average price decreases as the age increases, it suggests that older cars tend to have lower prices. This can be attributed to factors such as depreciation, wear and tear, and the perception that older cars may have a higher likelihood of requiring maintenance or repairs.

Mileage: Similarly, the PDP plot demonstrates the relationship between the average price of the car and its mileage while keeping age and other features constant. If the average price decreases as the mileage increases, it indicates that higher mileage is associated with lower prices. Higher mileage typically suggests more usage and potential wear on the car, which can impact its value and pricing.

These relationships are based on general observations, but it's important to note that other factors may also influence car prices, such as brand, model, condition, market demand, and location. The PDP plot allows you to isolate the impact of age and mileage on the car price, providing insights into how these specific features affect the average price while controlling for other variables.