ADVANCED MACHINE LEARNING PROJECT

RAMSES MORENO DE LA CRUZ – MMU ID: 23624012

1. Data Processing for Machine Learning

1.1. Stratified Splitting (Train/Validation/Test folds)

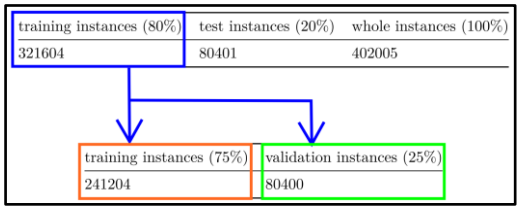
[Data used: raw dataset provided]

After loading the provided raw dataset, the same was split into a training dataset, a validation dataset, and a test dataset in a stratified way, using as a parameter for the split the "target variable" (i.e., the variable "price"). This discretization consisted of splitting the variable "price" into six bins. The lower and upper bounds of each bin were established based on the data distribution of the raw dataset provided for the variable "price." In addition, it is important to highlight that the first four bins hold around 90% of the instances.

Bin #1 (£)	Bin #2 (£)	Bin #3 (£)	Bin #4 (£)	Bin #5 (£)	Bin #6 (£)
0 - 20,000	20,000-40,000	40,000-70,000	70,000-100,000	100,000-400,000	400,000-10,000,000

The raw dataset provided (i.e., 402,005 instances) was split into a training dataset (60% of 241204 instances), a validation dataset (20% of 80,400 instances), and a test dataset (20% of 80,401 instances) using the following steps:

- Step 1: Stratification (as it was described in the first paragraph).
- Step 2: The raw dataset was split into the so-called training dataset (80% = 321,604 instances) and the test dataset (20% = 80,401 instances).
- Step 3: Stratification (as it was described in the first paragraph) in the so-called training dataset (80% = 321,604 instances).
- Step 4: The so-called training dataset (80% = 321,604 instances) was split again into a training dataset (75% = 241,204 instances) and a validation dataset (20% = 80,400 instances).



1.2. Fixing a feature with erroneous values

[Data used: stratified training dataset]

The metric feature "year_of_registration" had twelve values (999, 1006, 1007, 1008, 1009, 1010, 1015, 1016, 1017, 1018, 1063, and 1515) that were considered errors or noisy data (some of them shadowed in red in the image below).

```
[ 999. 1007. 1008. 1009. 1018. 1063. 1515. 1934. 1950. 1952. 1954. 1955.
 1956. 1957. 1958. 1959. 1960. 1961. 1962. 1963. 1964. 1965. 1966. 1967.
 1968. 1969. 1970. 1971. 1972. 1973. 1974. 1975. 1976. 1977. 1978. 1979.
 1980. 1981. 1982. 1983. 1984. 1985. 1986. 1987. 1988. 1989. 1990. 1991.
 1992. 1993. 1994. 1995. 1996. 1997. 1998. 1999. 2000. 2001. 2002. 2003.
 2004. 2005. 2006. 2007. 2008. 2009. 2010. 2011. 2012. 2013. 2014. 2015.
 2016. 2017. 2018. 2019. 2020.   nan]
```

Therefore, based on simplicity (considering the small percentage that 12 values represented in the training dataset [12/241204 = 0.00005%]) and with the approach to not drop any instance, an "element-wise operation" was developed to update those 12 values (listed previously) by 1000, if and only if they were below the year 1900 (before the updating process) and below the year 2020 (after the updating process).

```
data_c1.loc[data_c1['year_of_registration'] < 1900, 'year_of_registration'] =␣
↪data_c1.loc[data['year_of_registration'] < 1900, 'year_of_registration'] +␣
↪1000
```

1.3. Fixing the features with contradictory values

[Data used: stratified training dataset]

Some of the values of the metric feature "mileage" and the feature "vehicle_condition" described information considered contradictory, as the following table shows:

mileage	vehicle_condition	Number of Instances
equal to 0 km	used	215
greater than 0 km	new	9222

Therefore, based on convenience, considering the small percentage values represented in the training dataset [(9222+215) /241204 = 0.04%], and with the approach of not dropping any instance, the "loc" attribute was used to access a group of rows and columns and then fix the labels of specific instances.

```
data_c2.loc[(data_c2["mileage"]==0) &␣
↪(data_c2["vehicle_condition"]=='USED'),"vehicle_condition"] = "NEW"
```

```
data_c2.loc[(data_c2["mileage"]>0) &␣
↪(data_c2["vehicle_condition"]=='NEW'),"vehicle_condition"] = "USED"
```

1.4. Fixing the features with missing values

[Data used: stratified training dataset]

	Missing	Percentage %
public_reference	0	0.00
mileage	72	0.03
reg_code	19043	7.89
standard_colour	3257	1.35
standard_make	0	0.00
standard_model	0	0.00
vehicle_condition	0	0.00
year_of_registration	19933	8.26
price	0	0.00
body_type	502	0.21
crossover_car_and_van	0	0.00
fuel_type	365	0.15

The non-metric features (shadowed in blue in the image above) had the majority of missing values of the whole dataset (i.e., 9.60%) compared to the metric features (shadowed in red in the image above), which had a minority of missing values of the whole dataset (i.e., 8.29%).

Consequently, the way to handle missing values in non-metric features was through the "imputation" of the "most frequent value" for each feature, and for metric features, through the "imputation" of the "median value" for each feature. To facilitate this process, the "imputation" technique was applied in a pipeline together with other techniques that belong to the fields of "feature transformation" and "feature scaling".

1.5. Changing the datatype of a feature

[Data used: stratified training dataset]

Considering the approach of using all the features of the training dataset to gain deep insights into the data, the variable "crossover_car_and_van" is transformed from "boolean" into "strings" (i.e., Dtype: "object", as it is shown shadowed in red in the image below). Consequently, this procedure smooths future steps in encoding this new non-metric feature into a metric feature.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 241204 entries, 2 to 402003
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   public_reference      241204 non-null  int64
 1   mileage               241132 non-null  float64
 2   reg_code              222161 non-null  object
 3   standard_colour       237947 non-null  object
 4   standard_make         241204 non-null  object
 5   standard_model        241204 non-null  object
 6   vehicle_condition     241204 non-null  object
 7   year_of_registration  221271 non-null  float64
 8   price                 241204 non-null  int64
 9   body_type             240702 non-null  object
 10  crossover_car_and_van 241204 non-null  object
 11  fuel_type             240839 non-null  object
```

```
    #Step 2: Convert boolean to string
data_c1['crossover_car_and_van'] = data_c1['crossover_car_and_van'].astype(str)
```

2. Feature Engineering

2.1. Deriving features

[Data used: stratified training dataset]

Based on domain knowledge and the features "body_type", "mileage", and "year_of_registration", the following features were generated:

- number of doors: “doors”
- number of seats: “seats”
- “mileage” / “year_of_registration" = “mileage_year”

In the case of “doors” and “seats”, the steps to derive them are as follows:

- Step 1: identify the type of "body_type" present in the training dataset.
- Step 2: Build an “informative dataframe” that has the body type, number of doors, and number of seats according to domain knowledge (see image below, shadowed in red).
- Step 3: merge (based on the column “body_type”) the training dataset with the “informative dataset” (built in Step 2) to produce a new dataset.

	body_type	doors	seats
0	SUV	5	5
1	Hatchback	5	5
2	Convertible	2	4
3	Limousine	6	10
4	Saloon	4	5
5	Coupe	2	4
6	MPV	5	6
7	Estate	5	5
8	Pickup	4	5
9	Panel Van	5	5
10	Minibus	5	8
11	Combi Van	5	5
12	Window Van	4	12
13	Camper	4	5
14	Car Derived Van	4	2
15	None	4	5
16	Chassis Cab	2	2

In the case of the feature “mileage” or "year_of_registration", it was generated by dividing the variable “mileage” by the variable "year_of_registration".

```
data_c4["mileage_year"] = data_c4["mileage"] / data_c4["year_of_registration"]
print(data_c4.shape)
```

2.2. Producing polynomials and interacting features

[Data used: stratified training dataset]

The insights provided by the exploratory data analysis (EDA) were clear about the presence of a non-linear relationship between the metric features ("mileage", "year_of_registration", and “public_reference”) and the target value (“price”). For instance, low values of the “Pearson correlation” coefficient unequivocally point to the presence of non-linear relationships (see image below, shadowed in red).

Metric Variable	Pearson Correlation
price	1.000000
year_of_registration	0.125117
public_reference	-0.052097
mileage	-0.144140

Therefore, the generation of polynomial and interaction features to capture non-linear patterns and/or non-linear relationships between the independent features and the target variable was mandatory.

The approaches to obtaining the best “Polynomial Model” were the following:

- Approach 1: it generates polynomial and interaction features using only the metric features (i.e., “mileage”, “year_of_registration”, and “public_reference”). The output was the following:

Exponent	Coefficient of Determination (R^2 Score)	Mean Absolute Error (MAE) [£]	Number of Features Generated
2	0.42	10631.22	34
3	0.42	10613.38	91
4	0.43	10641.96	217

- Approach 2: It generates polynomial and interaction features using the metric features (i.e., “mileage”, “year_of_registration”, and “public_reference”) and the new metric features (i.e., the encoded non-metric features). The output was the following:

Exponent	Coefficient of Determination (R^2 Score)	Mean Absolute Error (MAE) [£]	Number of Features Generated
2	0.78	4519.03	71
3	0.89	4489.21	247
4	0.96	3668.68	703

It is important to note that the encoding of non-metric features was done through the technique of “target encoding” for the following reasons:

- These non-metric features are nominal variables.
- “Target Encoding” does not generate extra columns for the training dataset, and it is computationally cheaper than “One-Hot Encoding.”

The “Polynomial Model of Exponent 2” (to be specific, of Approach 2: considering metric and encoded non-metric variables) was chosen to develop the rest of the project because it is not overfitting (R^2=0.78), it has a reasonable MAE (4519.03 £) compared to the other polynomial models, and the number of features generated is the smallest one (71 features), which in the trade-off of performance and computational efficiency makes it the best candidate.

```
if poly_lin_model:
    metric_transformer.steps.extend([
        ("scaler", MinMaxScaler()),
        ('poly', PolynomialFeatures(2, include_bias=False))
    ])
    non_metric_transformer.steps.extend([
        ("scaler", MinMaxScaler()),
        ('poly', PolynomialFeatures(2, include_bias=False))
    ])
```

The striking difference in the predictive performance of the “Polynomial Model of Exponent 2” (to be specific, of Approach 2: considering metric and encoded non-metric variables) and a simple linear model can be seen in the following table, confirming again in a straightforward way the presence of non-linear relationships between the variables.

Model	Coefficient of Determination (R^2 Score)	Mean Absolute Error (MAE) [£]	Number of Features
Polynomial	0.78	4519.03	71
Linear	0.41	10569.89	14

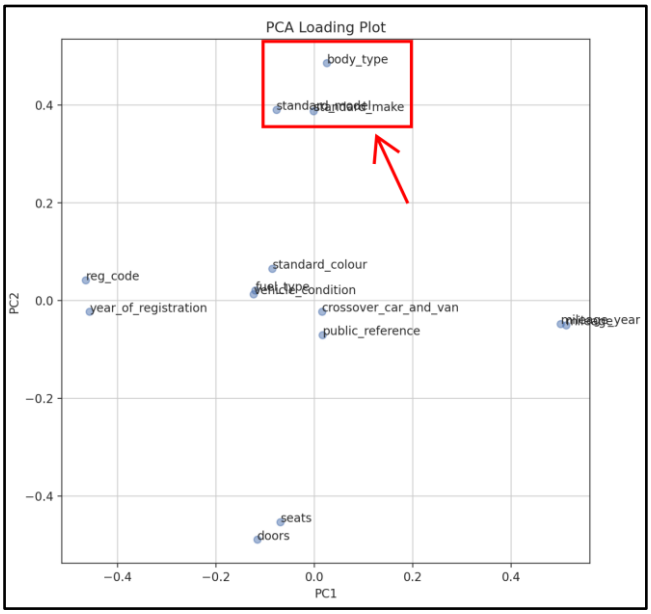
3. Feature Selection and Dimensionality Reduction

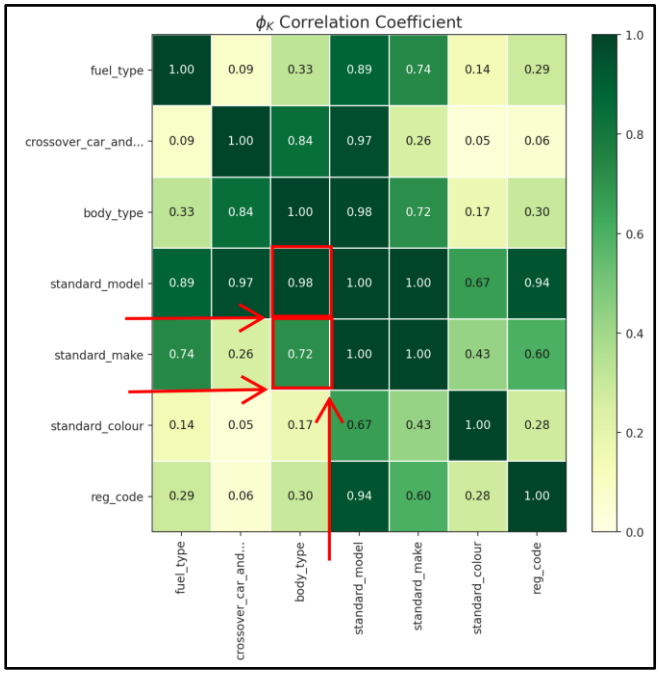
3.1. Principal Component Analysis (PCA)

[Data used: stratified training dataset]

Despite the fact that PCA does not work properly as an "automatic feature selector" because its main goal is to reduce the dimensionality of the feature space by creating new variables called "principal components", it was applied to spot positive (similar "loading values") or negative (one positive and one negative "loading value" on the same principal component) correlations among the features through the "PCA Loading Plot." For instance (see table below):

Variables	PCA Loading Correlation	Phi-k Correlation Coefficient
body_type, standard_make	Positive	0.98
body_type, standard_model	Positive	0.72





Furthermore, the information about correlations derived from PCA is very useful for "Shapley Values" because "Shapley Values" tend to be sensitive to "correlated features" as it becomes difficult to disentangle the "individual contributions" of each feature to the output of the model..

3.2. Manual Selection guided by domain knowledge and exploratory data analysis

[Data used: stratified training dataset]

Considering the information provided by several methods applied in the exploratory data analysis, such as "Pearson, Spearman, and Phi-k Correlation Coefficients", which pointed out the presence of non-linear relationships and/or complex relationships between the variables, plus the insights gained about correlations between features after applying PCA, the decision to keep all the variables to figure out the behavior of the model in this scenario was taken.

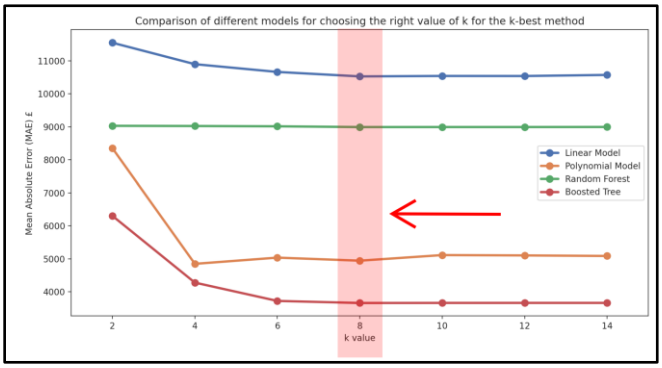
3.3. Select K-Best, Feature Elimination with Cross Validation (RFECV) and Sequential Feature Selector (SFS)

[Data used: stratified training dataset]

For this "Select K-Best" the "scoring function" chosen was "f_regression" because it is suitable for regression models, it is based on the "scoring function" called "r_regression" and it is less computationally expensive than the "scoring function" called "mutual_info_regression" (see image below, shadowed in red).

```
def regression_pipeline_k(estimator_model, X, lin_model=False, poly_lin_model=
 False):

    reg_pipeline = Pipeline(
        steps=[
            ("transformations", transformer_pipeline(X, lin_model,
 poly_lin_model)),
            ("feat_sel_k", SelectKBest(f_regression, k=8)),
            ("regressor", estimator_model),
```

In addition, several values of "k" were tried (k = 2, 4, 6, 8, 10, 12 and 14), and then model's performance was evaluated using the Mean Absolute Error (MAE) to select "k value" that provided a good balance between performance (MAE) and model complexity (number of variables). After comparing the MAE of all models (Linear Model, Polynomial Model, Random Forest and Boosted Trees) the best k value is k = 8 (i.e., this was the k value for which the models output the lowest MAE); therefore, for the feature selection process k = 8 was the value used.



Regarding RFECV and SFS, it is important to note that both methods were implemented in every model (Linear Model, Polynomial Model, Random Forest and Boosted Trees) using "LinearRegression()" as an "estimator" because it was faster and computationally inexpensive compared to other kinds of "estimators" (see images below, shadowed in red).

```
def regression_pipeline_RFECV(estimator_model, X, lin_model=False,
 poly_lin_model= False):

    reg_pipeline = Pipeline(
        steps=[
            ("transformations", transformer_pipeline(X, lin_model,
 poly_lin_model)),
            ("feat_sel_RFECV", RFECV(LinearRegression(), step=1, cv=5)),
            ("regressor", estimator_model),
```

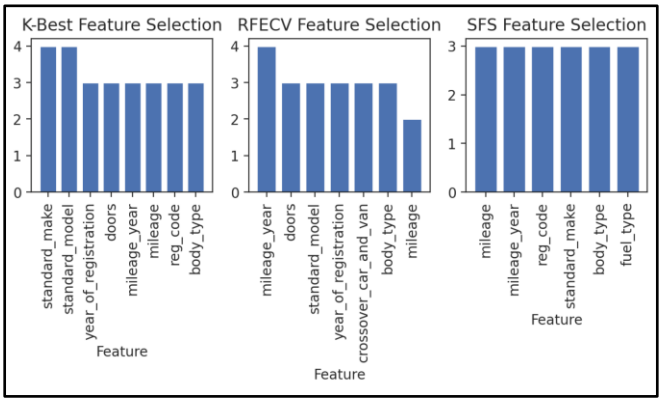
```
def regression_pipeline_SFS(estimator_model, X, lin_model=False,␣
↪poly_lin_model= False):

    reg_pipeline = Pipeline(
        steps=[
            ("transformations", transformer_pipeline(X, lin_model,␣
↪poly_lin_model)),
            ("feat_sel_SFS", SequentialFeatureSelector(LinearRegression(),␣
↪n_features_to_select='auto', direction="forward", tol=0.01)),
            ("regressor", estimator_model),
```

3.4. Summary of Features Selected

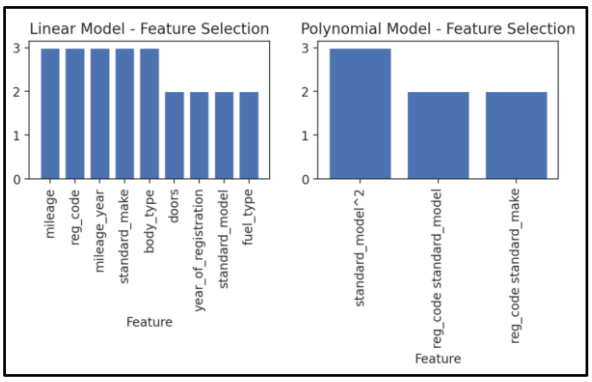
[Data used: stratified training dataset]

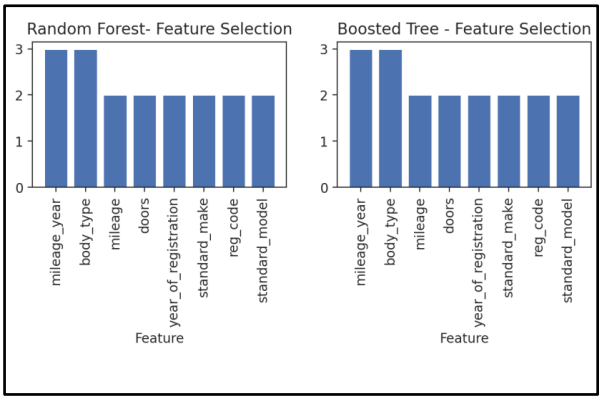
The approaches to summarizing the "feature selected" were the following:

- Approach 1: The "features selected" were grouped based on the "automatic feature selector" and their frequency (greater than one). For this analysis, we took into account the best features provided by every model (Linear Model, Polynomial Model, Random Forest and Boosted Trees). See the histograms below.



- Approach 2: The "features selected" were grouped based on every model (Linear Model, Polynomial Model, Random Forest and Boosted Trees) and their frequency (greater than one). For this analysis, we took into account the best features provided by every "automatic feature selector" (Select K-Best, Feature Elimination with Cross Validation, and Sequential Feature Selector). See the histograms below.





Finally, the features chosen to face the rest of the project (training, validation, and test phase) came from the "Approach 2", because it considers the features that are most suitable for the properties of the models and consequently increases their predictive performance. The following table summarizes key information about the models used during the feature selection process and the final list of features selected per model.

Model	Parameter	Number of Features (Before → After "feature selection")	List of Features
Linear	-	14 → 9	mileage, mileage_year, reg_code, standard_make, body_type, year_of_registration, doors, standard_model, fuel_type
Polynomial	Exponent = 2	71 → 3	standard_model^2, reg_code* standard_make, reg_code*standard_ model
Random Forest	max_depth = 2	14 → 8	mileage, mileage_year, reg_code, standard_make, body_type, year_of_registration, doors, standard_model
Boosted Trees	max_depth = 2	14 → 8	mileage, mileage_year, reg_code, standard_make, body_type, year_of_registration, doors, standard_model

It is extremely relevant to remark that from the "List of Features" selected (on the table above), three models (Linear Model, Random Forest and Boosted Trees) have among their features two derived features (i.e., "doors" and "mileage_year").

4. Model Building

4.1. Linear Model

[Data used: stratified training dataset, stratified validation dataset]

The "Linear Model" during the "training and validation phase" was implemented together with the regularization technique called "Elastic Net" because the goal was to try to fit the data as well as possible but at the same time keep the weights as small as possible (i.e., keep a balance between the "bias" produced by the L1 term and the "variance" produced by the L2 term), where:

- L1: LASSO (Least Absolute Shrinkage and Selection Operator). This method shrinks the weights of the less important features towards zero (i.e., it does perform an additional "feature selection process"). Additionally, this technique is very helpful in avoiding overfitting.
- L2: Ridge Regression. It shrinks the weights of the less important features towards zero but does not set any of them exactly to zero (i.e., it does not perform "a feature selection process").

```
from sklearn.linear_model import ElasticNet

training_lr = regression_pipeline(ElasticNet(), x_training_data_lr_after_fs,␣
↪lin_model=True, poly_lin_model= False)
```

```
lr_grid = dict(regressor__alpha=[0.1, 1, 10], regressor__l1_ratio=[0, 0.5, 1])

from sklearn.model_selection import GridSearchCV

lr_gridsearch_cv = GridSearchCV(
    regression_pipeline_val(ElasticNet(), x_validation_data_lr_after_fs,␣
↪lr_grid, lin_model=True, poly_lin_model=False),
    lr_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    return_train_score=True,
    n_jobs=-1,
    error_score='raise',
)
lr_gridsearch_cv
```

The Mean Absolute Error for the training set (MAE = £ 10729.71) and for the validation set (MAE = £ 8389.72) are huge amounts of money compared to the "mean value" of the target variable (i.e., "price") in the training dataset (μ = £ 17,376.12).

Phase	Model	Mean Absolute Error (MAE) [£]	Features
Before Grid SearchCV (training dataset)	Linear [ElasticNet]	10729.71	mileage, mileage_year, reg_code, standard_make, body_type, year_of_registration, doors, standard_model, fuel_type
During Grid SearchCV (validation dataset)	**Best Model** ↓ Linear [ElasticNet (alpha= 0.1, l1_ratio= 0.5)]	8389.72	

After applying the Linear Model on the stratified training dataset and stratified validation dataset, it is very clear that the Linear Model is seriously struggling to fit the data. It is important to recall that based on the information provided by the Exploratory Data Analysis (EDA) there are non-linear relationships and/or complex non-linear relationships between the variables of the dataset, making very difficult for a simple model such as Linear Model to capture proper insights of the dataset as it is reflected in the "Coefficient of Determination" during the training phase (R^2 = 0.006). Therefore, the poor behavior demonstrated by the Linear Model was expected.

4.2. Polynomial Model

[Data used: stratified training dataset, stratified validation dataset]

The "Polynomial Model" during the "training and validation phase" was the "Polynomial Model of Exponent 2" previously described in the section "2.2. Producing polynomials and interacting features" of this report.

```
if poly_lin_model:
    metric_transformer.steps.extend([
        ("scaler", MinMaxScaler()),
        ('poly', PolynomialFeatures(2, include_bias=False))
    ])
    non_metric_transformer.steps.extend([
        ("scaler", MinMaxScaler()),
        ('poly', PolynomialFeatures(2, include_bias=False))
    ])
```

```
poly_lr_grid = dict(regressor__fit_intercept=[True, False])

from sklearn.model_selection import GridSearchCV

poly_lr_gridsearch_cv = GridSearchCV(
    regression_pipeline_val(LinearRegression(), X_transformed_poly_lr_val_df2,␣
↪poly_lr_grid, lin_model=False, poly_lin_model=False),
    poly_lr_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    return_train_score=True,
    n_jobs=-1,
    error_score='raise',
)
poly_lr_gridsearch_cv
```

The Mean Absolute Error for the training set (MAE = £ 4698.60) and for the validation set (MAE = £ 4991.90) are reasonable amounts of money compared to the "mean value" of the target variable (i.e., "price") in the training dataset (μ = £ 17,376.12).

Phase	Model	Mean Absolute Error (MAE) [£]	Features
Before Grid SearchCV (training dataset)	Polynomial Exponent 2	4698.60	standard_model^2, reg_code* standard_make, reg_code*standard_ model
During Grid SearchCV (validation dataset)	**Best Model** ↓ Polynomial Exponent 2 [fit_intercept =True]	4991.90	

After applying the Polynomial Model to the stratified training dataset and stratified validation dataset, it is evident that the Polynomial Model is doing a very good job to fit the data compared to the Linear Model. This is because a Polynomial model is well suited to address scenarios in which there are non-linear relationships between the variables of the dataset, making easier for it to capture in a proper way many insights from the dataset as they are reflected in the "Coefficient of Determination" during the training phase (R^2 = 0.77).

4.3. Random Forest

[Data used: stratified training dataset, stratified validation dataset]

```
training_rf = regression_pipeline(RandomForestRegressor(max_depth=5,␣
↪random_state=21), x_training_data_rf_after_fs, lin_model=False,␣
↪poly_lin_model= False)
```

```
rf_grid = dict(regressor__n_estimators=[100, 110], regressor__max_depth=[10,␣
↪20])

from sklearn.model_selection import GridSearchCV

rf_gridsearch_cv = GridSearchCV(
    regression_pipeline_val(RandomForestRegressor(), x_validation_rf_after_fs,␣
↪rf_grid, lin_model=True, poly_lin_model=False),
    rf_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    return_train_score=True,
    n_jobs=-1,
    error_score='raise',
)
rf_gridsearch_cv
```

The Mean Absolute Error for the training set (MAE = £ 4671.63) and for the validation set (MAE = £ 2618.62) are small amounts of money compared to the "mean value" of the target variable (i.e., "price") in the training dataset (μ = £ 17,376.12).

Phase	Model	Mean Absolute Error (MAE) [£]	Features
Before Grid SearchCV (training dataset)	Random Forest [max_depth: 5]	4671.63	mileage, mileage_year, reg_code, standard_make, body_type, year_of_registration, doors, standard_model
During Grid SearchCV (validation dataset)	**Best Model** ↓ Random Forest [max_depth: 20, n_estimators: 110]	2618.62	

After applying the Random Forest Model to the stratified training dataset and stratified validation dataset, it is evident that the Random Forest Model excels at fitting the data compared to the Linear Model and Polynomial Model. This is probably due to the fact that Random Forest Model is well suited to address scenarios in which there are non-linear and/or complex relationships between the variables of the dataset, making it so easy for it to fit the dataset as it is reflected in the "Coefficient of Determination" during the training phase (R^2 = 0.92), which, by the way, could lead to potentially overfitting situations in future stages.

Furthermore, this kind of model is very hard to interpret (i.e., harder to interpret than individual decision trees) and slow to train and fine-tune; therefore, despite the great benefits in performance prediction it offers, it remains an option to take in a cautious way.

4.4. Boosted Trees

[Data used: stratified training dataset, stratified validation dataset]

```
training_bt = regression_pipeline(GradientBoostingRegressor(max_depth=5,␣
↪random_state=21), x_training_data_bt_after_fs, lin_model=False,␣
↪poly_lin_model= False)
```

```
bt_grid = dict(regressor__n_estimators=[100, 110], regressor__max_depth=[10,␣
↪20])

from sklearn.model_selection import GridSearchCV

bt_gridsearch_cv = GridSearchCV(
    regression_pipeline_val(GradientBoostingRegressor(),␣
↪x_validation_bt_after_fs, bt_grid, lin_model=True, poly_lin_model=False),
    bt_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    return_train_score=True,
    n_jobs=-1,
    error_score='raise',
)
bt_gridsearch_cv
```

The Mean Absolute Error for the training set (MAE = £ 2837.12) and for the validation set (MAE = £ 2590.03) are significant small amounts of money compared to the "mean value" of the target variable (i.e., "price") in the training dataset (μ = £ 17,376.12).

Phase	Model	Mean Absolute Error (MAE) [£]	Features
Before Grid SearchCV (training dataset)	Boosted Trees [max_depth: 5]	2837.12	mileage, mileage_year, reg_code, standard_make, body_type, year_of_registration, doors, standard_model
During Grid SearchCV (validation dataset)	**Best Model** ↓ Boosted Trees [max_depth: 10, n_estimators: 110]	2590.03	

After applying the Boosted Trees Model to the stratified training dataset and stratified validation dataset, it is irrefutable that the Boosted Trees Model surpasses the Linear Model, Polynomial Model and Random Forest in the context of fitting the data and prediction accuracy. This is probably due to the fact that the boosted tree model is well suited to address scenarios in which there are non-linear and/or complex relationships between the variables of the dataset. However, this also makes it extremely easy for it to overfit the dataset, as it is reflected in the "Coefficient of Determination" during the training phase (R^2 = 0.98).

4.5. A Voter Ensemble

[Data used: validation dataset]

The steps to fine tune the "Voter Ensemble" were the following:

1. Step 1: The best individual models (Linear Model, Random Forest and Boosted Trees) from the validation phase (described in the previous sections) were chosen.
2. Step 2: A series of different weights for the individuals' models were established as a way to check the performance (Mean Absolute Error) of the "Voter Ensemble" under several scenarios.

Model[1]	Mean Absolute Error (MAE) [£]
Scenario 1: 0.25 LR + 0.25 POLY_LR + 0.25 RF + 0.25 BT	3240.69
Scenario 2: 0.20 LR + 0.30 POLY_LR + 0.30 RF + 0.20 BT	3043.47
Best Model ↓ Scenario 3: 0.20 LR + 0.20 POLY_LR + 0.30 RF + 0.30 BT	2805.86

Scenario 1: All models have the same weight (25%) in the "Voter Ensemble". This was a less trustworthy scenario considering that the dataset has non-linear and/or complex relationships that could not be well addressed for the Linear Model. In other words, the Linear Model having the same importance than the rest of the models was hurtful for the performance of the "Voter Ensemble" as it is shown in the Mean Absolute Error for the validation set (MAE = £ 3240.69), which is far greater than the Mean Absolute Error for the validation set of the best Random Forest Model (MAE = £ 2618.62) or the best Boosted Trees Model (MAE = £ 2590.03).

Scenario 2: This scenario was better than Scenario 1. However, giving the same weight (20%) to the Linear Model (LR) and Boosted Trees in the "Voter Ensemble" was still hurting the "Voter Ensemble". In other words, doing this weight distribution did not harness the power of the Boosted Trees Model over a dataset that has non-linear and/or complex relationships. This is shown in the Mean Absolute Error for the validation set (MAE = £ 3043.47), which is greater than the Mean Absolute Error for the validation set of the best Random Forest Model (MAE = £ 2618.62) or the best Boosted Trees Model (MAE = £ 2590.03).

Scenario 3: This is the best of the tried scenarios because it gives more weight to the Random Forest Model and Boosted Trees Model (30%, respectively). Consequently, it has well harnessed the incredible fitting and predicting power of the Random Forest Model and Boosted Trees Model in a dataset that has non-linear and/or complex relationships. This is shown in the Mean Absolute Error for the validation set (MAE = £ 2805.86), which is very close to the Mean Absolute Error for the validation set of the best Random Forest Model (MAE = £ 2618.62) or the best Boosted Trees Model (MAE = £ 2590.03).

[1] LR: Linear Model, POLY_LR: Polynomial Model, RF: Random Forest, BT: Boosted Trees

5. Model Evaluation and Analysis

5.1. Overall Performance with Cross-Validation

[Data used: stratified test dataset]

The following table shows the summary of the Mean Absolute Error (MAE) and "Coefficient of Determination" (R^2) during the Test Phase of the best models.

Model	Coefficient of Determination (R^2 Score)	Mean Absolute Error (MAE) [£]
Best Linear Model	0.11	8493.63
Best Polynomial Model	0.52	4724.03
Best Random Forest	0.71	2406.05
Best Boosted Trees	0.71	2404.61
Best Voter Ensemble	-	3163.81

Some of the insights from the testing phase are the following:

- The performance of the Best Linear Model confirmed the findings made during the Validation Phase, in other words, the Linear Model was the worst model to address this dataset.
- The "Coefficient of Determination" (R^2) of the Best Polynomial Model sharply decreased from (R^2 = 0.77) during the Training Phase to (R^2 = 0.52) during the Test Phase; therefore, this model could only explain almost half of the variance in the variable "price," and it did not look anymore as a promising option to generate price predictions for this dataset.
- About the Best Random Forest and the Best Boosted Trees models, both models performed better than in the training and validation sets in the context of Mean Absolute Error (MAE). In addition, the "Coefficient of Determination" during the Test Phase (R^2 = 0.71) decreased compared to the Training Phase (R^2 = 0.92) and (R^2 = 0.98) respectively for Random Forest and Boosted Trees; therefore, this was evidence that both models were not overfitting anymore.
- The Best Voter Ensemble with a Mean Absolute Error (MAE = £ 3163.81) did perform slightly worse than in the Validation Phase (MAE = £ 2805.86).
- Finally, the evidence shown in several sections of this report suggests that the Best Random Forest, the Best Boosted Trees and the Best Voter Ensemble are the most appropriate models to address in a reasonable and responsible way (from the point of view of performance) the dataset provided by the company Audotrader.

The coefficients for the Best Linear Model are the following:

features_lr_test	coefficients_lr_test
mileage	-3528.440212
mileage_year	-2749.979065
year_of_registration	186.446223
doors	-8505.019995
reg_code	16801.833524
standard_make	10713.795964
body_type	11756.667732
standard_model	5152.638411
fuel_type	6073.333393

The coefficients for the Best Polynomial Model are the following:

features_poly_lr_test	coefficients_poly_lr_test
reg_code_standard_make	4.138054e+05
reg_code_standard_model	1.587407e+06
standard_model^2	9.435454e+06

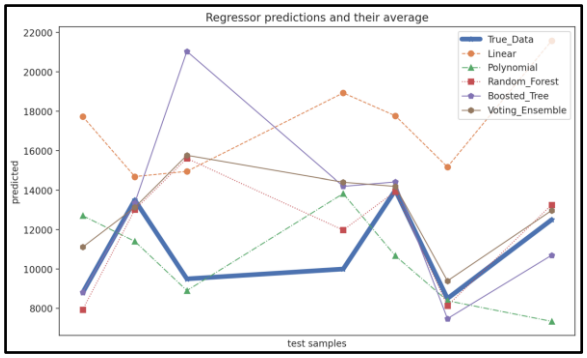
5.2. True vs Predicted Analysis

[Data used: stratified test dataset]

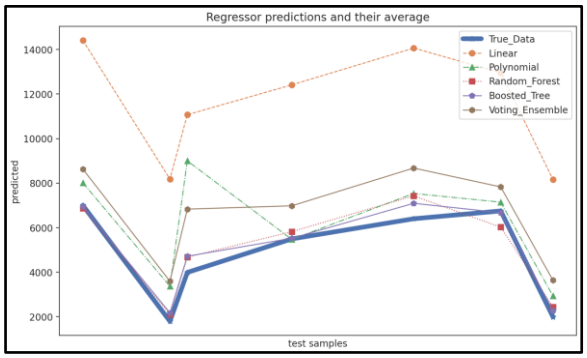
In this section, it was checked (using a data subset) the price differences among the predictions of all the best models (Linear, Polynomial, Random Forest, Boosted Trees, Voter Ensemble) in different regions or zones (see the following images and the table below) of the data distribution of the target variable (i.e., the variable "price"). This analysis was useful to back up the idea that there are models whose performance is better in specific price zones. Consequently, the concept of a single model, "'One Size Fits All," probably does not fit to address this situation of predicting prices in this dataset provided by the company Autotrader.

Region	Information	Best Models
Region 1	7,495 £ <= X <= 20,000 £ (between 25th and 75th percentile) This is the most important region of the data distribution, because this is the bulk of the dataset	Random Forest
Region 2	122 £ <= X <= 7,495 £ (between Minimum Value and 25th percentile) This region has the cheapest cars	Boosted Trees
Region 3	20,000 £ <= X <= 100,000 £ (between 75th percentile and 100,000 £)	Boosted Trees
Region 4	1,000,000 £ <= X <= 10,000,000 £ This region has luxury cars *Note: overall, in this region, all the best models shown poor predictive performance*	Boosted Trees

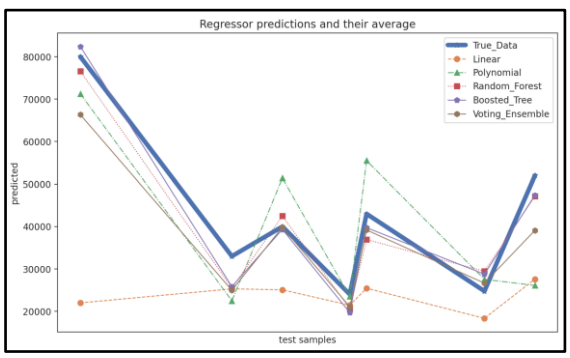
Region 1 (7,495 £ <= True Data [Price] <= 20,000 £) ↘



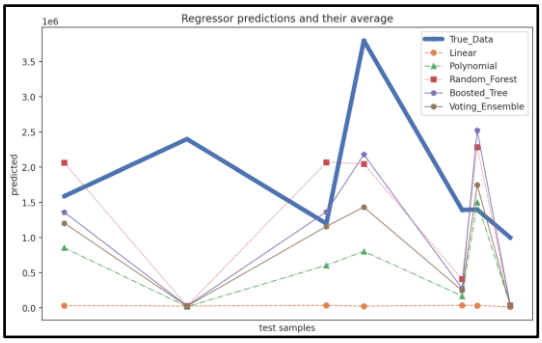
Region 2 (122 £ <= True Data [Price] <= 7,495 £) ↘



Region 3 (20,000 £ <= True Data [Price] <= 100,000 £) ↘



Region 4 (1,000,000 £ <= True Data [Price] <= 10,000,000 £) ↘



5.3. Global and Local Explanations with SHAP

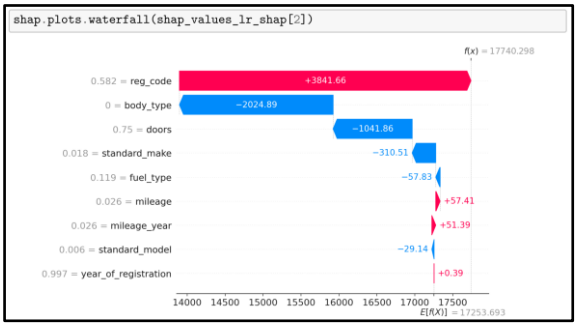
5.3.1. Local Explanations with SHAP

[Data used: stratified validation dataset, stratified test dataset]

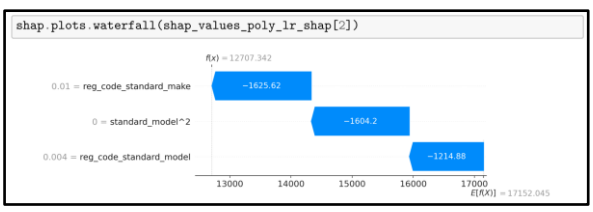
Despite the fact that Shapley Values can be really function to quantify feature importance in complex non-linear models such as "Random Forest, Boosted Trees and Voter Ensemble", in this case, they were applied to the "Best Linear Model" and "Best Polynomial Model" with the goal of comparing the most important features according to their feature's coefficients (as it was shown in the section "5.1. Overall Performance with Cross-Validation") with the most important features according to Shapley Values for the same models.

In this instance (which target price is 8,795 £), the local explanation through Shapley Values shows in a clear way that the features with higher magnitude are "reg_code" (pushing the prediction higher), "body_type" and "doors" (both pushing the prediction lower), which coincide with some of the most important features provided at the end of the Test Phase for the "Best Linear Model" (shadowed in red in the image below).

features_lr_test	coefficients_lr_test
mileage	-3528.440212
mileage_year	-2749.979065
year_of_registration	186.446223
doors	-8505.019995
reg_code	16801.833524
standard_make	10713.795964
body_type	11756.667732
standard_model	5152.638411
fuel_type	6073.333393



In this instance (which target price is 8,795 £), the local explanation through Shapley Values shows in a clear way that the features "standard_model^2", "reg_code*standard_make", and "reg_code*standard_ model" (all pushing the prediction lower) match with the most important features provided at the end of the Test Phase for the "Best Polynomial Model".



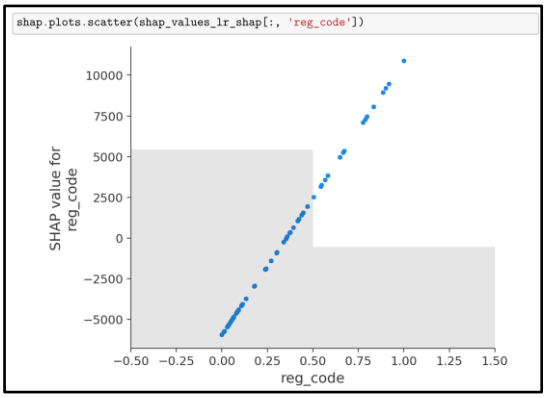
features_poly_lr_test	coefficients_poly_lr_test
reg_code_standard_make	4.138054e+05
reg_code_standard_model	1.587407e+06
standard_model^2	9.435454e+06

As it was seen in the previous two instances, model coefficients of the Best Linear and Best Polynomial Model provided a good estimate of feature importance; however, Shapley values gave us a deep insight into the model because they are able to capture non-linear relationships and interactions between features, which coefficients struggle to do. In addition, it is relevant to highlight (as it was shown in the previous instances) the usual divergences between the magnitude, sign, and rank of the coefficients and the most important features according to Shapley values.

5.3.2. Global Explanations with SHAP

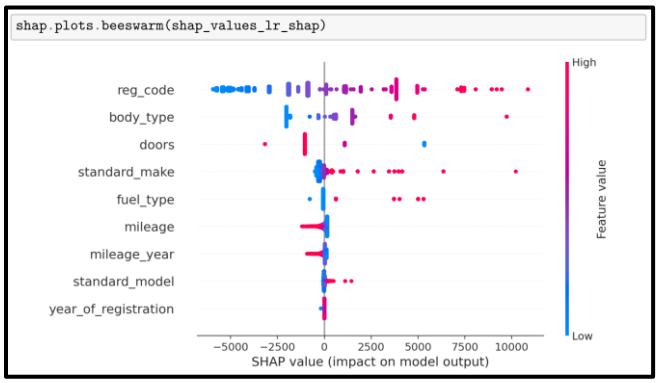
[Data used: stratified test dataset]

According to the coefficients and the Shapley Values, the feature "reg_code" is one of the most important features of the "Best Linear Model"; therefore, the relationship between the values (transformed and normalized) of "reg_code" and their corresponding "Shapley Values" was plotted (as seen in the following image). The main conclusion of this plot is that higher values of "reg_code", which are highly correlated with newer cars or not-so-old cars, mean higher car prices and, consequently, higher "Shapley Values" (i.e., great importance of this feature at the local and global level). This analysis makes sense considering the linear correlation between the values of the feature "reg_code" and the "Shapley Values" and also the lack of a signal of diminishing or even reversing in the scatterplot.



From the following "Bee swarm Plot of Shapley Values" provided by the "Best Linear Model", the most important insights are as follows:

- The features "reg_code", "body_type", "doors", and "standard_make", which have many dots further from the centre line, have a more significant impact on the model's output (as demonstrated in some way with the coefficients of the Best Linear Model and local Shapley values).
- Despite the fact that the variables "standard_model", "mileage_year" and "mileage" have a low impact on predictions, they are relatively consistent in the kind of impact that they have (i.e., they showed a clear dispersion of values in only one side of the X-axis).
- Ther impact of the features "reg_code" and "body_type" varies significantly across the instances, as shown in the plot with a wide dispersion of data points on both sides of the x-axis for both variables.



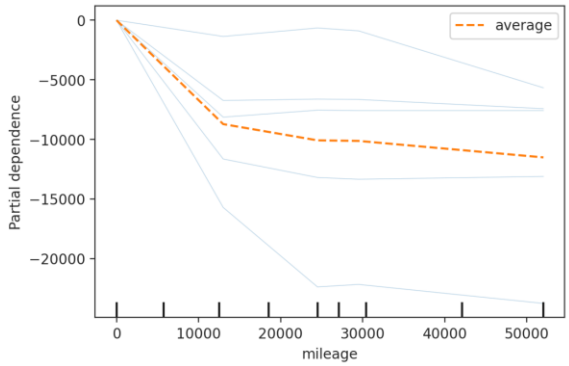
5.4. Partial Dependency Plot

[Data used: stratified test dataset]

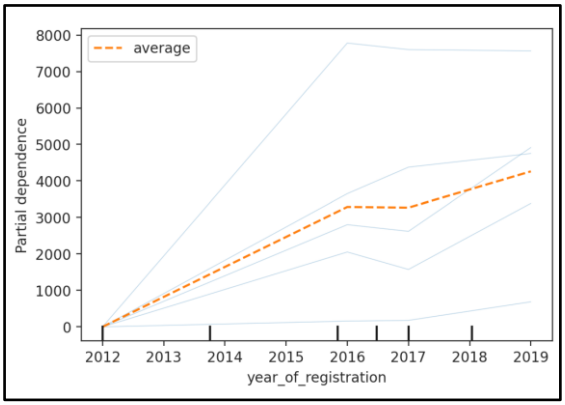
In the following instances, the Partial Dependency Plots (PDPs) have been used to gain deeper insights into the "Best Random Forest" because this is a complex and less interpretable model compared to the "Linear or Polynomial Model".

In the first case, using a small data subset that belongs to the bulk of the dataset: 7,495 £ <= True Data [Price] <= 20,000 £, the plot shows a sharp linear decrease in the average line (from 0 to 10,000 miles) and then a moderate decrease (from 10,000 to 50,000 miles). This means that after a certain mileage (10,000 miles), the effects of the feature "mileage" (independent of the values of other features) on the model's predictions are not so meaningful. Finally, in this plot, it is also evident that, on average, a car with great mileage will be cheaper than a car with none or lower mileage (i.e., there is a strong relationship between the "price" and the feature "mileage").

	mileage	mileage_year	reg_code	standard_make	body_type	year_of_registration	price
2	24487.0	12.140307	17	Peugeot	Hatchback	2017.0	8795
3	52023.0	25.805060	66	Volkswagen	Estate	2016.0	13500
4	29500.0	14.662028	12	Mercedes-Benz	Saloon	2012.0	9495
7	12999.0	6.438336	69	Peugeot	Hatchback	2019.0	9995



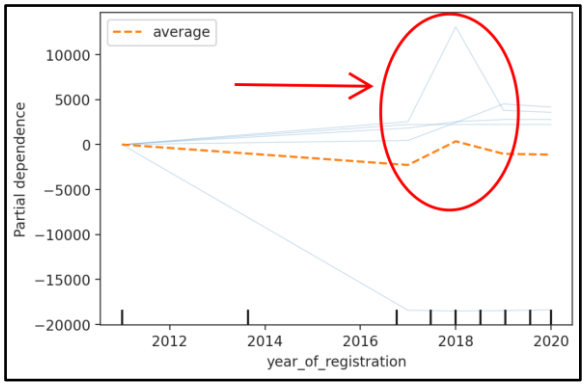
In the second case, using a small data subset that belongs to the bulk of the dataset: 7,495 £ <= True Data [Price] <= 20,000 £, the plot shows a sharp linear increase in the average line (from 2012 to 2016), then flat (from 2016 to 2017) probably due to the Brexit event, and finally increase again (from 2017 to 2019). This means that the newer the car, the higher the price will be for this specific range of prices. As a consequence, the effects of the feature "year_of_registration" (independent of the values of other features) on the model's predictions are relevant. Unambiguously, in this plot, it is also evident that, on average, a car with a later year of registration will be more expensive than a car with an older year of registration (i.e., there is a strong relationship between the "price" and the feature "year of registration").



In the third case, using a small data subset (that belongs to the following range of data: 20,000 £ <= True Data [Price] <= 100,000 £, the plot shows a slightly linear decrease in the average line (from around 2012 to 2016), then a marginally peak (around 2018), and then a moderate decrease (from 2018 to 2020). This could mean that the "best random forest model" for this subset of data in this specific range of prices has learned some contradictory or noisy patterns from the data. In other words, it seems in some way illogical that, on average, the cars with a "year_of_registration" of 2018 were more expensive than the cars with a later "year_of_registration," such as in 2019 and 2020

In addition, this strange behavior in the average line of the Partial Dependency Plot (shown in a red circle in the plot below) could also be a sign that maybe there were other variables exerting influence on the price of the cars that had "year_of_registration" in 2018. In the end, the effects of the feature "year_of_registration" (independent of the values of other features) on the model's predictions are not as relevant for the model's prediction on the range of prices (20,000 £ <= True Data [Price] <= 100,000 £) as it were for the range of prices of (7,495 £ <= True Data [Price] <= 20,000 £).

	mileage	mileage_year	reg_code	standard_make	body_type	year_of_registration	price
1	9500.0	4.705300	19	Land Rover	SUV	2019.0	79995
10	19000.0	9.415263	18	Mercedes-Benz	Saloon	2018.0	32980
13	58000.0	28.841372	61	Bentley	Coupe	2011.0	39990
17	22369.0	11.090233	17	Volkswagen	SUV	2017.0	24000
18	4000.0	1.980198	70	Mercedes-Benz	SUV	2020.0	42980



In the fourth case, using a small data subset (that belongs to the following range of data: 20,000 £ <= True Data [Price] <= 100,000 £, The surface plot shows interesting interactions between the features "year_of_registration" and "mileage." These interactions include the following:

- There is a strong impact on "prices" from the variable "year_of_registration" from 2011 to 2017 and the variable "mileage" from 0 to 20,000 miles (see the yellow arrow in the plot).
- There is a modest impact on "prices" from the feature "mileage", to be specific, from 20,000 to 50,000+ miles (see the green dashed arrow in the plot).
- There is a moderate and progressive impact on "prices" from the variable "year_of_registration" from 2017 to 2020 and the variable "mileage" from 0 to 50,000+ miles.