

Final project

2024-04-24

##Question As the main means of transportation, cars important for almost every family, therefore there exists a large market for car trading, especially the used car trading market. Because families with a small budget usually choose to buy a used car to find value for money. My research objective in this project is to analyze the importance of factors that influence the price of used cars. The dataset I chose is from Kaggle (<https://www.kaggle.com/datasets/adityadesai13/used-car-dataset-ford-and-mercedes/data>) It is a data of used cars in the UK and I chose the Toyota brand as the the target brand for the project. The data set contains information of price, transmission, mileage, fuel type, road tax, miles per gallon (mpg), and engine size.

##Method/approach In this project, I aim to use gradient boosting to analyze the factors that influence the price of used Toyota cars in the UK, specifically using the XGBoost algorithm as it is efficient and effective in processing structured data. My approach included an in-depth study of the feature import values generated by the gradient boosting model, as I recognized that tree-based methods, while useful, do not always provide a completely reliable interpretation of feature importance. To address possible discrepancies in feature importance, I will employ LIME as an additional method for validating and comparing the results obtained from XGBoost. This dual approach allows for a more reliable validation of the importance of each feature such as mileage, fuel type, and engine size. Additionally, to enhance my understanding of model robustness and interpretability, I will perform a sensitivity analysis on the hyperparameters. After establishing a baseline of optimized parameters through cross-validation, I will intentionally vary these parameters slightly to observe their impact on model performance and feature import stability. This approach will not only help to identify the most influential factors affecting used car prices, but also test the reliability and interpretability of the model under different configurations, leading to a deeper understanding of the dynamics of model behavior and the relevance of the features in predicting used car prices.

##Result

```
#code categorical variable & normalize continuous variables  
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 4.3.1
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.1
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.1
```

```
data <- read.csv("toyota.csv")
```

```
#data cleaning
```

```
data$model <- NULL
```

```
data$transmission <- as.numeric(factor(data$transmission))
```

```
data$fuelType <- as.numeric(factor(data$fuelType))
```

```
features <- data[, !(names(data) %in% 'price')]
```

```
target <- data$price
```

```
#split dataset
```

```
set.seed(5293)
```

```
indexes <- createDataPartition(target, p = 0.8, list = FALSE)
```

```
train_features <- features[indexes, ]
```

```
train_target <- target[indexes]
```

```
test_features <- features[-indexes, ]
```

```
test_target <- target[-indexes]
```

```
#Training the XGBoost Model and Feature Importance Analysis
```

```
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.3.1
```

```
dtrain <- xgb.DMatrix(data = as.matrix(train_features), label = train_target)
```

```
dtest <- xgb.DMatrix(data = as.matrix(test_features), label = test_target)
```

```
params <- list(
```

```
  booster = "gbtree",
```

```
  objective = "reg:squarederror",
```

```
  eta = 0.1,
```

```
  gamma = 0,
```

```
  max_depth = 6,
```

```
  min_child_weight = 1,
```

```
  subsample = 0.5,
```

```
  colsample_bytree = 0.5
```

```
)
```

```
set.seed(5293)
```

```
xgb_model <- xgb.train(
```

```
  params = params,
```

```
  data = dtrain,
```

```
  nrounds = 100,
```

```
  watchlist = list(train = dtrain, test = dtest),
```

```
  print.every.n = 10,
```

```
  early_stopping_rounds = 10,
```

```
  maximize = FALSE
```

```
)
```

```
## Warning: 'print.every.n' is deprecated.
```

```
## Use 'print_every_n' instead.
```

```
## See help("Deprecated") and help("xgboost-deprecated").
```

```
## [20:14:15] WARNING: src/learner.cc:767:
## Parameters: { "print_every_n" } are not used.
##
## [1] train-rmse:12763.120500 test-rmse:12561.047439
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [11] train-rmse:5021.398983 test-rmse:4892.118061
## [21] train-rmse:2353.994606 test-rmse:2335.157947
## [31] train-rmse:1526.126857 test-rmse:1573.913511
## [41] train-rmse:1273.074778 test-rmse:1378.798766
## [51] train-rmse:1183.682227 test-rmse:1320.317450
## [61] train-rmse:1129.460684 test-rmse:1279.039346
## [71] train-rmse:1086.120047 test-rmse:1251.158799
## [81] train-rmse:1055.988263 test-rmse:1223.606863
## [91] train-rmse:1029.069033 test-rmse:1204.715096
## [100] train-rmse:1009.526730 test-rmse:1190.305512

importance_matrix <- xgb.importance(feature_names = colnames(train_features), model = xgb_model)
print(importance_matrix)
```

	Feature	Gain	Cover	Frequency
	<char>	<num>	<num>	<num>
## 1:	engineSize	0.33069023	0.16362493	0.09339159
## 2:	mpg	0.21749025	0.30924923	0.25204806
## 3:	fuelType	0.13269964	0.06783332	0.06389951
## 4:	year	0.10936157	0.10693739	0.12261060
## 5:	mileage	0.10131542	0.20997884	0.28590934
## 6:	tax	0.07596451	0.09597737	0.12097215
## 7:	transmission	0.03247840	0.04639892	0.06116876

#explain: Gain: This measures the average gain of the feature when it is used in trees. A higher value indicates the feature is more important for generating a split. Cover: This measures the relative quantity of observations concerned by a feature. Frequency: This is the percentage representing the relative number of times a particular feature occurs in the trees of the model. In my output, I found : EngineSize has the highest gain, which suggests that it contributes most to the model's predictions per split, meaning changes in engine size heavily influence the predicted price. mpg (miles per gallon) is second in gain, but has the highest cover, implying that it affects a large number of observations and is also quite influential. fuelType, year, and mileage have relatively lower gain scores, suggesting they have a lesser impact on the model's prediction per split compared to engineSize and mpg. However, mileage has a relatively high frequency, indicating it's used often in the trees. tax and transmission seem to be the least important in terms of gain but still have a decent cover and frequency, suggesting they affect a reasonable number of observations and are used quite regularly in the model.

#Validate Feature Importance with LIME

```
library(lime)

predict_model <- function(model, newdata) {
  if (!is.matrix(newdata)) {
    newdata <- as.matrix(newdata)
  }
  preds <- predict(model, xgb.DMatrix(newdata))
}
```

```

    return(data.frame(Price = preds))
  }

explainer <- lime::lime(
  x = as.data.frame(as.matrix(train_features)),
  model = xgb_model,
  bin_continuous = FALSE,
  preprocess = NULL
)

explanations <- lime::explain(
  as.data.frame(as.matrix(test_features[1:10,])),
  explainer = explainer,
  n_features = 5,
  fun = predict_model
)

print(explanations)

```

```

## # A tibble: 50 x 11
##   model_type case  model_r2 model_intercept model_prediction feature
##   <chr>      <chr>    <dbl>         <dbl>          <dbl> <chr>
## 1 regression 15      0.663      -1047214.      16243. engineSize
## 2 regression 15      0.663      -1047214.      16243. fuelType
## 3 regression 15      0.663      -1047214.      16243. year
## 4 regression 15      0.663      -1047214.      16243. mileage
## 5 regression 15      0.663      -1047214.      16243. mpg
## 6 regression 22      0.663      -1126921.      18652. engineSize
## 7 regression 22      0.663      -1126921.      18652. year
## 8 regression 22      0.663      -1126921.      18652. fuelType
## 9 regression 22      0.663      -1126921.      18652. mileage
## 10 regression 22      0.663      -1126921.      18652. mpg
## # i 40 more rows
## # i 5 more variables: feature_value <dbl>, feature_weight <dbl>,
## #   feature_desc <chr>, data <list>, prediction <dbl>

```

#explain My XGBoost model finds engineSize and mpg to be very important in determining the price of a car, and LIME suggests similar importance at the individual prediction level. This indicates a strong influence of these features on the model across both global (model-level) and local (individual prediction) scales.

#Sensitivity Analysis on Hyperparameters

```

library(caret)

tune_grid <- expand_grid(
  nrounds = c(50, 100, 150),
  eta = c(0.05, 0.1),
  max_depth = c(4, 6, 8),
  gamma = c(0, 0.1),
  colsample_bytree = c(0.5, 0.7),
  min_child_weight = c(1, 2),
  subsample = c(0.5, 0.7)
)

```

```

)

train_control <- trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  returnData = FALSE
)

tune_results <- train(
  x = as.matrix(train_features),
  y = train_target,
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = tune_grid
)

## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=1, subsample=0.5, n
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=1, subsample=0.5, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=1, subsample=0.7, n
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=1, subsample=0.7, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, n
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=2, subsample=0.7, n
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.5, min_child_weight=2, subsample=0.7, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=1, subsample=0.5, n
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:16] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=1, subsample=0.5, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=1, subsample=0.7, n
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=1, subsample=0.7, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=2, subsample=0.5, n
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=2, subsample=0.5, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=2, subsample=0.7, n
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.0, colsample_bytree=0.7, min_child_weight=2, subsample=0.7, n
## + Fold1: eta=0.05, max_depth=4, gamma=0.1, colsample_bytree=0.5, min_child_weight=1, subsample=0.5, n
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:14:17] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.05, max_depth=4, gamma=0.1, colsample_bytree=0.5, min_child_weight=1, subsample=0.5, n

```



```
##      nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 66      150          6 0.05      0              0.7              1      0.7
```

#explain I use Sensitivity Analysis on Hyperparameters to find the ones that give the best performance according to some evaluation metric.

#output: nrounds: The number of boosting rounds or trees to build. A higher number of rounds increases the potential for learning complex patterns but also increases the risk of overfitting.

max_depth: This controls the maximum depth of the trees. Deep trees can model more complex relationships by creating more specific rules, but too much depth can lead to overfitting. A depth of 150 is extremely high for typical XGBoost applications and might suggest overfitting unless the dataset is very complex.

eta: Also known as the learning rate, this determines the step size at each iteration while moving toward a minimum of a loss function. A smaller value makes the model more robust to overfitting but requires more rounds (nrounds).

gamma: This is the minimum loss reduction required to make a further partition on a leaf node. It acts as a regularization parameter. A higher value leads to more conservative models.

colsample_bytree: This parameter sets the fraction of features to randomly sample from before creating each tree. Subsampling occurs once every boosting iteration.

min_child_weight: It determines the minimum sum of instance weight (hessian) needed in a child. It is used to control overfitting; higher values prevent the model from learning relations which might be highly specific to the particular sample selected for a tree.

subsample: This parameter controls the fraction of the total number of instances trained on during each iteration. A lower value can prevent overfitting; it's like row sampling in random forest.

To explain the results: nrounds: 150 - This indicates the number of gradient boosting rounds. It's the number of trees that the model will use. max_depth: 6 - This is a much more typical value for the maximum depth of a tree. It allows the model to learn complex patterns but is not so deep that it should automatically lead to overfitting. eta: 0.05 - A common and conservative learning rate that helps prevent overfitting by making the boosting process more gradual. gamma: 0 - Specifies the minimum loss reduction required to make a split on a tree's leaf. A value of 0 means that no regularization is applied via this method. colsample_bytree: 0.7 - Each tree uses 70% of the features, which provides a good balance between making the individual trees robust and allowing them to benefit from all the features. min_child_weight: 1 - This is the minimum sum of instance weight (hessian) needed in a child node and is the default value. It is conservative in that it allows for children that cover very few instances. subsample: 0.7 - Each tree is trained on 70% of the data instances, reducing the risk of overfitting by injecting randomness into the dataset each tree sees.

#refit the XGBoost Model by using suggested parameter

```
# Update parameters with the tuned values
tuned_params <- list(
  booster = "gbtree",
  objective = "reg:squarederror",
  eta = 0.05,                # Same as before
  gamma = 0,                 # Tuned value
  max_depth = 6,
  min_child_weight = 1,      # Same as before
  subsample = 0.7,          # Tuned value
  colsample_bytree = 0.7    # Tuned value
)

# Train the model with the tuned parameters
set.seed(5293)
```

```
xgb_model_tuned <- xgb.train(
  params = tuned_params,
  data = dtrain,
  nrounds = 150,           # Tuned value
  watchlist = list(train = dtrain, test = dtest),
  print.every.n = 10,
  early_stopping_rounds = 10,
  maximize = FALSE
)
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```

```
## [20:15:38] WARNING: src/learner.cc:767:
## Parameters: { "print_every_n" } are not used.
##
## [1]  train-rmse:13405.723484 test-rmse:13207.732649
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [11] train-rmse:8284.097027  test-rmse:8152.507786
## [21] train-rmse:5189.972661  test-rmse:5105.799799
## [31] train-rmse:3358.396341  test-rmse:3309.205321
## [41] train-rmse:2298.655324  test-rmse:2282.793861
## [51] train-rmse:1715.849213  test-rmse:1738.676481
## [61] train-rmse:1406.263393  test-rmse:1447.948774
## [71] train-rmse:1242.105167  test-rmse:1315.370718
## [81] train-rmse:1151.550067  test-rmse:1246.337831
## [91] train-rmse:1096.492653  test-rmse:1211.451567
## [101]  train-rmse:1065.598554  test-rmse:1188.362167
## [111]  train-rmse:1039.805725  test-rmse:1171.526769
## [121]  train-rmse:1023.236153  test-rmse:1162.827688
## [131]  train-rmse:1004.832855  test-rmse:1152.294976
## [141]  train-rmse:990.607979   test-rmse:1145.619577
## [150]  train-rmse:976.613424   test-rmse:1141.911438
```

```
importance_matrix_tuned <- xgb.importance(feature_names = colnames(train_features), model = xgb_model_tuned)
print(importance_matrix_tuned)
```

```
##      Feature      Gain      Cover  Frequency
##      <char>      <num>      <num>      <num>
## 1:  engineSize 0.40502983 0.22830099 0.10640732
## 2:      year 0.17358795 0.13946897 0.15331808
## 3:      mpg 0.15897042 0.27347796 0.22703498
## 4:  mileage 0.08765874 0.17929624 0.30630925
## 5:  fuelType 0.08429237 0.04764603 0.04887218
## 6:      tax 0.04660256 0.07893543 0.09921543
## 7: transmission 0.04385813 0.05287438 0.05884276
```

```
summary(xgb_model_tuned)
```

##	Length	Class	Mode
## handle	1	xgb.Booster.handle	externalptr
## raw	523816	-none-	raw
## best_iteration	1	-none-	numeric
## best_ntreelimit	1	-none-	numeric
## best_score	1	-none-	numeric
## best_msg	1	-none-	character
## niter	1	-none-	numeric
## evaluation_log	3	data.table	list
## call	8	-none-	call
## params	10	-none-	list
## callbacks	3	-none-	list
## feature_names	7	-none-	character
## nfeatures	1	-none-	numeric

#Analysis and Comparison: *#engineSize:* The tuned model shows a slight increase in gain, suggesting a greater impact on the model's predictions. The cover also increases, indicating this feature is affecting more observations after tuning. *#mpg:* There's a noticeable decrease in gain, meaning its influence on the model's predictions has diminished slightly after tuning. The cover for mpg decreased as well, suggesting it's now affecting fewer observations in the model. *#year:* Year has a higher gain after tuning, indicating an increase in its importance for the model's predictions. Year's frequency also increased, showing it's being used more often in the trees. *#mileage:* Mileage's gain has decreased slightly in the tuned model, but it remains significant. However, mileage's frequency increased substantially, making it the most frequent splitter in the tuned model. *#fuelType:* FuelType's gain increased in the tuned model, suggesting it has become a more significant predictor. Its cover and frequency have decreased, though, which implies it's being used in fewer splits. *#tax and transmission:* Both features show a small increase in gain in the tuned model, indicating a slightly higher impact on model predictions.

Overall Interpretation: The tuning process has altered the contribution of the various features to the model. Notably, engineSize and year have become more influential, while mpg and mileage have seen a decrease in their relative gain. However, mileage has become the most frequent feature used for splitting, which suggests it's still a very relevant predictor in the tuned model.

#compare the accuracy

```
# Predictions on the test set
preds <- predict(xgb_model, dtest)

# Calculate MSE
mse <- mean((test_target - preds)^2)
print(paste("MSE:", mse))
```

```
## [1] "MSE: 1416827.21112521"
```

```
# Calculate RMSE
rmse <- sqrt(mse)
print(paste("RMSE:", rmse))
```

```
## [1] "RMSE: 1190.30551167556"
```

```
# Calculate MAE
mae <- mean(abs(test_target - preds))
print(paste("MAE:", mae))
```

```
## [1] "MAE: 828.065655689892"
```

```
# Calculate R-squared
rss <- sum((test_target - preds)^2)
tss <- sum((test_target - mean(test_target))^2)
r_squared <- 1 - rss/tss
print(paste("R-squared:", r_squared))
```

```
## [1] "R-squared: 0.962104874815247"
```

```
# Predictions on the test set
preds <- predict(xgb_model_tuned, dtest)
```

```
# Calculate MSE
mse <- mean((test_target - preds)^2)
print(paste("MSE:", mse))
```

```
## [1] "MSE: 1303961.73470204"
```

```
# Calculate RMSE
rmse <- sqrt(mse)
print(paste("RMSE:", rmse))
```

```
## [1] "RMSE: 1141.91143908013"
```

```
# Calculate MAE
mae <- mean(abs(test_target - preds))
print(paste("MAE:", mae))
```

```
## [1] "MAE: 792.129516329491"
```

```
# Calculate R-squared
rss <- sum((test_target - preds)^2)
tss <- sum((test_target - mean(test_target))^2)
r_squared <- 1 - rss/tss
print(paste("R-squared:", r_squared))
```

```
## [1] "R-squared: 0.965123627789857"
```

#Analysis and Comparison: Overall, all four metrics indicate that the tuned model performs better than the original model. The errors (MSE, RMSE, MAE) are lower, and the explanatory power (R-squared) is higher. This suggests that the tuning process has led to a more accurate model.

#Sensitivity Analysis on Hyperparameters again with different parameter choice in this time, I try larger range of parameters than before

```

library(caret)

tune_grid2 <- expand.grid(
  nrounds = c(25,100,200),
  eta = c(0.025, 0.2),
  max_depth = c(3, 9, 15),
  gamma = c(0.5, 1),
  colsample_bytree = c(0.2, 0.9),
  min_child_weight = c(0, 3),
  subsample = c(0.2, 0.9)
)

train_control2 <- trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  returnData = FALSE
)

tune_results2 <- train(
  x = as.matrix(train_features),
  y = train_target,
  method = "xgbTree",
  trControl = train_control2,
  tuneGrid = tune_grid2
)

```

```

## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=0, subsample=0.2
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=0, subsample=0.2
## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=0, subsample=0.9
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=0, subsample=0.9
## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=3, subsample=0.2
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=3, subsample=0.2
## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=3, subsample=0.9
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=3, subsample=0.9
## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=3, subsample=0.9
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.2, min_child_weight=3, subsample=0.9
## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.9, min_child_weight=0, subsample=0.2
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.9, min_child_weight=0, subsample=0.2
## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.9, min_child_weight=0, subsample=0.9
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:15:38] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.9, min_child_weight=0, subsample=0.9
## + Fold1: eta=0.025, max_depth= 3, gamma=0.5, colsample_bytree=0.9, min_child_weight=3, subsample=0.2
## [20:15:39] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead

```



```
## + Fold5: eta=0.200, max_depth=15, gamma=1.0, colsample_bytree=0.9, min_child_weight=3, subsample=0.9
## [20:17:34] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## [20:17:34] WARNING: src/c_api/c_api.cc:935: 'ntree_limit' is deprecated, use 'iteration_range' instead
## - Fold5: eta=0.200, max_depth=15, gamma=1.0, colsample_bytree=0.9, min_child_weight=3, subsample=0.9
## Aggregating results
## Selecting tuning parameters
## Fitting nrounds = 200, max_depth = 9, eta = 0.025, gamma = 0.5, colsample_bytree = 0.9, min_child_weight = 3
```

```
print(tune_results2$bestTune)
```

```
##      nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 66         200         9 0.025  0.5              0.9              0         0.9
```

#refit the XGBoost Model by using suggested parameter

```
tuned_params <- list(
  booster = "gbtree",
  objective = "reg:squarederror",
  eta = 0.025,
  gamma = 1,
  max_depth = 9,
  min_child_weight = 0,
  subsample = 0.9,
  colsample_bytree = 0.9
)

# Train the model with the tuned parameters
set.seed(5293)
xgb_model_tuned <- xgb.train(
  params = tuned_params,
  data = dtrain,
  nrounds = 200,
  watchlist = list(train = dtrain, test = dtest),
  print.every.n = 10,
  early_stopping_rounds = 10,
  maximize = FALSE
)
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```

```
## [20:17:35] WARNING: src/learner.cc:767:
## Parameters: { "print_every_n" } are not used.
##
## [1]  train-rmse:13736.490196 test-rmse:13530.106057
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [11] train-rmse:10755.647208 test-rmse:10585.964802
## [21] train-rmse:8434.612703 test-rmse:8295.072725
## [31] train-rmse:6633.832024 test-rmse:6522.836029
```

```
## [41] train-rmse:5238.821640 test-rmse:5152.134988
## [51] train-rmse:4160.769740 test-rmse:4099.473023
## [61] train-rmse:3327.200932 test-rmse:3287.226344
## [71] train-rmse:2689.629098 test-rmse:2673.861318
## [81] train-rmse:2199.043701 test-rmse:2210.494261
## [91] train-rmse:1829.001552 test-rmse:1870.846616
## [101] train-rmse:1550.783730 test-rmse:1625.420026
## [111] train-rmse:1342.661530 test-rmse:1453.439201
## [121] train-rmse:1186.043637 test-rmse:1334.476924
## [131] train-rmse:1069.759324 test-rmse:1252.167578
## [141] train-rmse:984.794197 test-rmse:1198.147672
## [151] train-rmse:921.796706 test-rmse:1163.718847
## [161] train-rmse:875.183826 test-rmse:1140.308017
## [171] train-rmse:837.632609 test-rmse:1123.084389
## [181] train-rmse:810.190799 test-rmse:1112.981542
## [191] train-rmse:788.488389 test-rmse:1106.829255
## [200] train-rmse:772.180518 test-rmse:1101.485722
```

```
importance_matrix_tuned <- xgb.importance(feature_names = colnames(train_features), model = xgb_model_tuned)
print(importance_matrix_tuned)
```

```
##      Feature      Gain      Cover Frequency
##      <char>      <num>      <num>      <num>
## 1: engineSize 0.526767728 0.24228887 0.06059300
## 2:      year 0.223722983 0.15354252 0.15673919
## 3:      mpg 0.094029966 0.20993136 0.17959008
## 4:      mileage 0.076731945 0.26417778 0.44657071
## 5:      fuelType 0.057545873 0.03018180 0.02659857
## 6:      tax 0.011424514 0.03341176 0.08016052
## 7: transmission 0.009776991 0.06646591 0.04974794
```

```
summary(xgb_model_tuned)
```

```
##      Length Class      Mode
## handle      1 xgb.Booster.handle externalptr
## raw      2192798 -none-      raw
## best_iteration      1 -none-      numeric
## best_ntreelimit      1 -none-      numeric
## best_score      1 -none-      numeric
## best_msg      1 -none-      character
## niter      1 -none-      numeric
## evaluation_log      3 data.table      list
## call      8 -none-      call
## params      10 -none-      list
## callbacks      3 -none-      list
## feature_names      7 -none-      character
## nfeatures      1 -none-      numeric
```

```
preds <- predict(xgb_model_tuned, dtest)
```

```
mse <- mean((test_target - preds)^2)
print(paste("MSE:", mse))
```

```
## [1] "MSE: 1213270.79745579"
```

```
rmse <- sqrt(mse)
print(paste("RMSE:", rmse))
```

```
## [1] "RMSE: 1101.4857227653"
```

```
mae <- mean(abs(test_target - preds))
print(paste("MAE:", mae))
```

```
## [1] "MAE: 751.357663476272"
```

```
rss <- sum((test_target - preds)^2)
tss <- sum((test_target - mean(test_target))^2)
r_squared <- 1 - rss/tss
print(paste("R-squared:", r_squared))
```

```
## [1] "R-squared: 0.967549290138154"
```

#compared with the original one: Feature Importance: There is a significant increase in the importance of engineSize and year in the tuned model, indicating that the tuning process might have highlighted the relevance of these features to the model's predictions. mpg, fuelType, mileage, tax, and transmission all see a decrease in importance in the tuned model. This suggests that the tuning process could have deprioritized these features or the model found better splits with engineSize and year.

Model Performance: The tuned model has a lower MSE and RMSE, indicating better performance in terms of prediction error. The MAE is also lower, which means that on average, the tuned model has a smaller prediction error per data point. The R-squared value is higher for the tuned model, which means it explains a higher proportion of the variance in the target variable. Interpretation: The tuning process has led to a model that is more precise in its predictions. The model has become more confident about the impact of engineSize and year on the target variable after tuning. Lower error metrics and a higher R-squared value suggest that the tuned model is likely to generalize better to unseen data compared to the original model. Conclusion The tuning has improved the model's predictive accuracy and has shifted the importance towards engineSize and year, indicating these features are possibly more predictive for the target variable in the context of the model's task (which, based on the features, seems to be related to vehicle price prediction).

#compared with the 1st tuned one: Feature Importance: Engine Size remains the most important feature in both models, but its importance is slightly reduced in the second tuned model. Year also sees a decrease in importance in the second tuned model compared to the first tuned model. MPG has become more important in the second tuned model, indicating that the second tuned model may be capturing more nuanced relationships between MPG and the target variable. Mileage and Fuel Type have both seen increases in their importance scores in the second tuned model. Tax and Transmission show an increase in importance in the second tuned model, suggesting that the tuning altered the model's sensitivity to these features. Model Performance: The MSE and RMSE have increased from the first to the second tuned model, which means that the second model, on average, makes larger errors in its predictions. The MAE is also higher in the second model, indicating that average errors are larger. The R-squared value, which represents the proportion of the variance for the dependent variable that's explained by the independent variables in the model, has decreased slightly in the second tuned model. This suggests that the first tuned model fits the data slightly better.

#conclusion: Why different parameter cause different result?

Learning Rate (eta) First Model: 0.05 Second Model: 0.025 The learning rate, (eta), controls the step size at each iteration while moving toward a minimum of a loss function. A smaller learning rate makes the

model more robust to overfitting but also requires more rounds to train. This can explain why second model might need more rounds to converge, which might also lead to different feature importance distributions as the model has more iterations to fine-tune the importance of each feature.

Gamma First Model: 0 Second Model: 1 Gamma specifies the minimum loss reduction required to make a further partition on a leaf node of the tree. A higher gamma value leads to more conservative models. The second model's higher gamma value may result in fewer splits, thereby focusing the model's attention on the features with the most significant splits. This can make certain features appear more important.

Max Depth First Model: 6 Second Model: 9 The maximum depth of a tree controls how deep the tree can grow during any boosting round. A deeper tree can model more complex relationships by creating more specific rules, but it also risks overfitting. The increased depth in the second model allows it to capture more nuanced patterns which can change feature importances.

Minimum Child Weight First Model: 1 Second Model: 0 Minimum child weight is the minimum sum of instance weight needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than `min_child_weight`, then the building process will give up further partitioning. A lower value is more complex and can overfit, so setting this to 0 in the second model could allow the model to learn more specific leaf nodes, thus affecting the importance scores.

Subsample & Colsample_bytree Both Models: Decreased from the first to the second model (subsample: 0.7 to 0.9, colsample_bytree: 0.7 to 0.9) Subsample and colsample_bytree control the fraction of the sample and features that are randomly sampled to grow trees. Increasing these values makes the algorithm use more of the data, which can capture more feature interactions but can also lead to overfitting.

Number of Rounds (nrounds) First Model: 150 Second Model: 200 More rounds give more opportunity for the model to adjust weights on features, which changes the importance rankings.

Seed Both models use the same random seed, which ensures that the randomness in the training process is consistent between the two models.

#Summary The second model has been tuned to be more conservative (higher gamma) and complex (higher `max_depth` and lower `min_child_weight`), with more data utilized in each tree (higher subsample and colsample_bytree) and more iterations to learn from (higher nrounds). These changes can lead to a model that captures the data's nuances differently, hence the change in feature importance and performance metrics.

#overall summary: Through different model selection and factor selection, I get that for the UK Toyota used car market, price is most affected by engine size, followed by vehicle year, MPG and mileage are also important predictors. Fuel type, tax rate and transmission are less influential. In the study of this project, I also learned deeply about the changes in modeling results brought about by adjusting parameters. I think it is necessary to conduct Sensitivity Analysis on Hyperparameters because this process can help us to choose the best model, for example, I have shown that the both tuned model generally performs better than the original model, with less error and more accuracy. I also realized that when there is a deep understanding of what each parameter stands for, we can go ahead and set different metrics according to the research objectives, for example after I changed the range of parameter selection, the best parametric model was different and gave different results (refer to the results of tuned model 1 and tuned model 2). Even though the accuracy of Tuning Model 2 is not as high as that of Tuning Model 1, I cannot say that Tuning Model 2 is inferior to Tuning Model 1. When faced with a choice between these two models, if the primary goal is predictive accuracy, then the first model is preferable. However, if the primary goal is to understand the impact of different characteristics, the second model provides a different perspective, especially by giving more weight to MPG, mileage, and fuel type. So we need to carefully consider the model application context when making the final decision on which model to use.