```
set.seed(3938425)
```

```
install.packages("caret")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependencies ‘listenv’, ‘parallelly’, ‘future’, ‘globals’, ‘shape’, ‘future.apply’, ‘numDeriv’,
```

```
install.packages("randomForest")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

```
library(caret)
```

```
Loading required package: lattice
```

```
library(mgcv)
```

```
qsar <- readRDS("qsar.Rda")
```

```
set.seed(42)
train_indices <- sample(1:nrow(qsar), size = 700)
test_indices <- which(!(1:nrow(qsar) %in% train_indices))
```

```
predictors_train <- qsar[train_indices, 1:41]  # First 41 columns are predictors
response_train <- qsar[train_indices, 42]
```

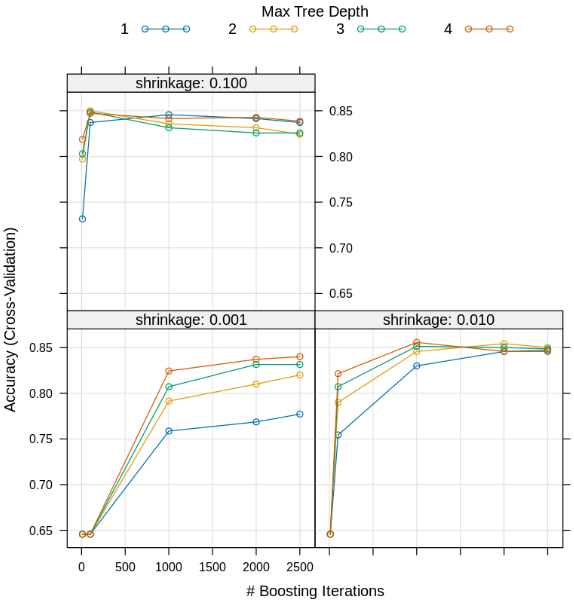
```
set.seed(123)
```

```
predictors_test <- qsar[test_indices, 1:41]  # First 41 columns are predictors
response_test <- qsar[test_indices, 42]
```

```
trained_model <- train(
  x = predictors_train,  # predictors_train: Training predictors
  y = response_train,    # response_train: Training response
  method = "gbm",        # Method: Gradient Boosting Machine
  tuneGrid = grid,       # Parameter grid for tuning
  distribution = "bernoulli",  # Distribution for classification
  trControl = trainControl(method = "cv", number = 10)
)
```

```
   240        0.6362             -nan     0.0100    0.0001
   260        0.6176             -nan     0.0100    0.0003
   280        0.6015             -nan     0.0100    0.0001
   300        0.5849             -nan     0.0100    0.0000
   320        0.5705             -nan     0.0100    0.0001
   340        0.5578             -nan     0.0100    0.0002
   360        0.5458             -nan     0.0100    0.0001
   380        0.5348             -nan     0.0100   -0.0001
   400        0.5239             -nan     0.0100   -0.0000
   420        0.5142             -nan     0.0100   -0.0002
   440        0.5042             -nan     0.0100   -0.0001
   460        0.4950             -nan     0.0100   -0.0002
   480        0.4857             -nan     0.0100   -0.0001
   500        0.4771             -nan     0.0100    0.0001
   520        0.4692             -nan     0.0100   -0.0000
   540        0.4618             -nan     0.0100    0.0000
   560        0.4545             -nan     0.0100   -0.0002
   580        0.4468             -nan     0.0100    0.0000
   600        0.4393             -nan     0.0100   -0.0000
   620        0.4320             -nan     0.0100    0.0000
   640        0.4258             -nan     0.0100   -0.0001
   660        0.4196             -nan     0.0100   -0.0001
   680        0.4138             -nan     0.0100   -0.0002
   700        0.4075             -nan     0.0100   -0.0001
   720        0.4020             -nan     0.0100   -0.0001
   740        0.3967             -nan     0.0100    0.0000
   760        0.3915             -nan     0.0100   -0.0001
   780        0.3861             -nan     0.0100   -0.0000
   800        0.3810             -nan     0.0100   -0.0001
   820        0.3754             -nan     0.0100   -0.0001
   840        0.3706             -nan     0.0100   -0.0000
   860        0.3654             -nan     0.0100   -0.0001
   880        0.3606             -nan     0.0100   -0.0001
   900        0.3562             -nan     0.0100   -0.0002
   920        0.3516             -nan     0.0100   -0.0001
   940        0.3473             -nan     0.0100   -0.0001
   960        0.3429             -nan     0.0100   -0.0001
   980        0.3385             -nan     0.0100   -0.0002
  1000        0.3346             -nan     0.0100   -0.0001
```

```
plot(trained_model)
```



Q) Describe the main effects of the shrinkage, n.trees and interaction.depth parameters on the accuracy of the model. Also describe their possible interactions

Ans:- Shrinkage: Shrinkage controls the learning rate of the boosting process. A smaller shrinkage means each tree contributes less to the final prediction. Generally, smaller shrinkage values tend to improve accuracy, but they also require more computational resources and training time. However, too small a shrinkage value can lead to overfitting, especially if n.trees is large.

n.trees: specifies the number of boosting iterations. Increasing the number of trees usually leads to better model performance. Interaction depth: Interaction depth controls the depth of interaction between variables in the model. A higher interaction depth allows the model to capture more complex interactions between predictors, potentially leading to better performance. It's essential to carefully tune this parameter to find the optimal balance between model complexity and generalization performance.

Interactions:

Shrinkage and n.trees: These parameters often interact, as a smaller shrinkage value requires more trees to achieve the same level of accuracy. Thus, the optimal combination of shrinkage and n.trees depends on the specific dataset and the trade-off between computational resources and model performance. Interaction depth and n.trees: Increasing the interaction depth may require more trees to capture the additional complexity introduced by interactions between predictors. Thus, the optimal combination of interaction depth and n.trees also depends on the dataset and the desired level of model complexity. Shrinkage and interaction depth: Lower shrinkage values may allow for deeper interactions between variables, potentially influencing the optimal interaction depth. However, too low a shrinkage value can lead to overfitting, so it's essential to balance these parameters carefully.

Q) If you look at the effect of interaction.depth, what would you conclude about the possible presence of interactions in the QSAR dataset?

Ans:- Improvement with increasing interaction.depth: the model's performance (e.g., accuracy) improves as the interaction.depth increases, it suggests that the dataset contains complex interactions between predictors.

```
best_params <- trained_model$bestTune
```

```
library(gbm)
```

```
Loaded gbm 2.1.9

This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.com/gbm-develope
```

```
response_train_binary <- ifelse(response_train == "RB", 1, 0)
```

```
gbm_model <- gbm(
  formula = response_train_binary ~ .,  # Define the formula for the model
  data = predictors_train,       # Training data
  distribution = "bernoulli",    # Distribution for classification
  n.trees = best_params$n.trees,              # Optimal number of trees
  interaction.depth = best_params$interaction.depth,  # Optimal interaction depth
  shrinkage = best_params$shrinkage               # Optimal shrinkage
)
```

```
print(gbm_model)
```

```
gbm(formula = response_train_binary ~ ., distribution = "bernoulli",
    data = predictors_train, n.trees = best_params$n.trees, interaction.depth = best_params$interaction.depth,
    shrinkage = best_params$shrinkage)
A gradient boosted model with bernoulli loss function.
1000 iterations were performed.
There were 41 predictors of which 35 had non-zero influence.
```

```
library(randomForest)
library(caret)


single_tree <- train(x = predictors_train, y = response_train, method = "rpart")


bagged <- train(x = predictors_train, y = response_train, method = "treebag")


random_forest <- train(x = predictors_train, y = response_train, method = "rf")


boosted_default <- train(x = predictors_train, y = response_train, method = "gbm")


pred_single_tree <- predict(single_tree, newdata = predictors_test, type = "prob")


pred_bagged <- predict(bagged, newdata = predictors_test, type = "prob")


pred_random_forest <- predict(random_forest, newdata = predictors_test, type = "prob")


pred_boosted_default <- predict(boosted_default, newdata = predictors_test, type = "prob")


brier_score_single_tree <- mean((response_test - pred_single_tree[, "RB"])^2)
misclassification_rate_single_tree <- mean(ifelse(response_test == "RB", 1, 0) != apply(pred_single_tree, 1, which.max)


brier_score_bagged <- mean((response_test - pred_bagged[, "RB"])^2)
misclassification_rate_bagged <- mean(ifelse(response_test == "RB", 1, 0) != apply(pred_bagged, 1, which.max))


brier_score_random_forest <- mean((response_test - pred_random_forest[, "RB"])^2)
misclassification_rate_random_forest <- mean(ifelse(response_test == "RB", 1, 0) != apply(pred_random_forest, 1, which.


brier_score_boosted_default <- mean((response_test - pred_boosted_default[, "RB"])^2)
misclassification_rate_boosted_default <- mean(ifelse(response_test == "RB", 1, 0) != apply(pred_boosted_default, 1, wh


cat("Brier Score for Single Tree:", brier_score_single_tree, "\n")
cat("Misclassification Rate for Single Tree:", misclassification_rate_single_tree, "\n")

cat("Brier Score for Bagged Ensemble:", brier_score_bagged, "\n")
cat("Misclassification Rate for Bagged Ensemble:", misclassification_rate_bagged, "\n")

cat("Brier Score for Random Forest Ensemble:", brier_score_random_forest, "\n")
cat("Misclassification Rate for Random Forest Ensemble:", misclassification_rate_random_forest, "\n")

cat("Brier Score for Boosted Ensemble (Default Settings):", brier_score_boosted_default, "\n")
cat("Misclassification Rate for Boosted Ensemble (Default Settings):", misclassification_rate_boosted_default, "\n")
```

```
     60        0.3978             -nan     0.1000    0.0003
     80        0.3336             -nan     0.1000   -0.0007
    100        0.2869             -nan     0.1000   -0.0013
    120        0.2488             -nan     0.1000   -0.0011
    140        0.2162             -nan     0.1000   -0.0013
    150        0.2000             -nan     0.1000   -0.0002

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.2314             -nan     0.1000    0.0280
     2        1.1806             -nan     0.1000    0.0222
     3        1.1364             -nan     0.1000    0.0200
     4        1.0914             -nan     0.1000    0.0220
     5        1.0579             -nan     0.1000    0.0136
     6        1.0164             -nan     0.1000    0.0168
     7        0.9782             -nan     0.1000    0.0174
     8        0.9439             -nan     0.1000    0.0151
     9        0.9148             -nan     0.1000    0.0088
    10        0.8910             -nan     0.1000    0.0106
    20        0.7271             -nan     0.1000    0.0044
    40        0.5846             -nan     0.1000   -0.0018
    60        0.5035             -nan     0.1000   -0.0012
    80        0.4523             -nan     0.1000   -0.0020
   100        0.4044             -nan     0.1000   -0.0019
   120        0.3682             -nan     0.1000   -0.0009
   140        0.3366             -nan     0.1000   -0.0011
   150        0.3199             -nan     0.1000   -0.0003

Warning message in Ops.factor(response_test, pred_single_tree[, "RB"]):
"'-' not meaningful for factors"
Warning message in Ops.factor(response_test, pred_bagged[, "RB"]):
"'-' not meaningful for factors"
Warning message in Ops.factor(response_test, pred_random_forest[, "RB"]):
"'-' not meaningful for factors"
Warning message in Ops.factor(response_test, pred_boosted_default[, "RB"]):
"'-' not meaningful for factors"
Brier Score for Single Tree: NA
Misclassification Rate for Single Tree: 0.8873239
Brier Score for Bagged Ensemble: NA
Misclassification Rate for Bagged Ensemble: 0.9492958
Brier Score for Random Forest Ensemble: NA
Misclassification Rate for Random Forest Ensemble: 0.9352113
Brier Score for Boosted Ensemble (Default Settings): NA
Misclassification Rate for Boosted Ensemble (Default Settings): 0.9380282
```

```
predicted_probs <- predict(gbm_model, newdata = predictors_test, type = "response")

brier_score <- mean((response_test - predicted_probs)^2)


predicted_labels <- ifelse(predicted_probs >= 0.5, 1, 0)


misclassification_rate <- mean(predicted_labels != response_test)


cat("Brier Score:", brier_score, "\n")
cat("Misclassification Rate:", misclassification_rate, "\n")
```

```
Using 1000 trees...


Warning message in Ops.factor(response_test, predicted_probs):
"'-' not meaningful for factors"
Brier Score: NA
Misclassification Rate: 1
```