R Notebook tp1

```r
#-----------importation des librairy---------------------------

library(ggplot2)
library(smotefamily)
library(ROSE)
```

```
## Loaded ROSE 0.0-4
```

```r
library(caret)
```

```
## Le chargement a nécessité le package : lattice
```

```r
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'

## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag

## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attachement du package : 'pROC'

## Les objets suivants sont masqués depuis 'package:stats':
##
##     cov, smooth, var
```

```r
library(caret)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.

##
## Attachement du package : 'randomForest'

## L'objet suivant est masqué depuis 'package:dplyr':
##
##     combine

## L'objet suivant est masqué depuis 'package:ggplot2':
##
##     margin
```

```r
library(nnet)
library(naivebayes)
```

```
## naivebayes 1.0.0 loaded

## For more information please visit:

## https://majkamichal.github.io/naivebayes/
```

```r
library(e1071) # Pour Naive Bayes et SVM
library(class) # Pour k-NN
library(C50)
library(ranger)
```

```
##
## Attachement du package : 'ranger'

## L'objet suivant est masqué depuis 'package:randomForest':
##
##     importance
```

```r
library(xgboost)
```

```
##
## Attachement du package : 'xgboost'

## L'objet suivant est masqué depuis 'package:dplyr':
##
##     slice
```

```r
library(kernlab)
```

```
##
## Attachement du package : 'kernlab'

## L'objet suivant est masqué depuis 'package:ggplot2':
##
##     alpha
```

```r
library(adabag)
```

```
## Le chargement a nécessité le package : rpart

## Le chargement a nécessité le package : foreach

## Le chargement a nécessité le package : doParallel

## Le chargement a nécessité le package : iterators

## Le chargement a nécessité le package : parallel
```

```r
#library(RSNNS)
```

```r
#------------(question 1)---------------------------------------

donnees <- read.csv("C:/Users/ASUS/Documents/Programme/M1_TNSID/s8/data_mining/tp/tp1/Data_TP1.csv", se

# Afficher les premières lignes des données pour avoir un aperçu
head(donnees)
```

```
##      IDSM LABELLOC       DATE PREC TPMAX TPMIN TPMOY VTMOY VTHM VTDIR VTVIT
## 1 GARD_03     MTP1 01/01/2016 0.00    30     9    NA  2.46   NA    10   8.1
## 2 GARD_03     MTP1 02/01/2016 0.00    31    13    NA  2.01   NA   270   6.0
## 3 GARD_03     MTP1 03/01/2016 0.00    28    10    NA  0.67   NA   150  10.1
## 4 GARD_03     MTP1 04/01/2016 0.01    35    21    NA  1.34   NA   270   8.1
## 5 GARD_03     MTP1 05/01/2016 1.61    25    15    NA  2.46   NA   140  10.1
## 6 GARD_03     MTP1 06/01/2016 0.80    24    12    NA  2.91   NA    80  12.1
##   MET1 MET2 MET3
## 1    0    0    1
## 2    0    0    0
## 3    0    0    0
## 4    0    0    0
## 5    1    1    0
## 6    1    0    0
```

```r
# Quantité de données et nombre d'attributs
cat("Nombre d'observations:", nrow(donnees), "\n")
```

```
## Nombre d'observations: 1827
```

```r
cat("Nombre d'attributs:", ncol(donnees), "\n")
```

```
## Nombre d'attributs: 14
```

```r
# Type des attributs
str(donnees)
```

```
## 'data.frame':    1827 obs. of  14 variables:
##  $ IDSM    : chr  "GARD_03" "GARD_03" "GARD_03" "GARD_03" ...
##  $ LABELLOC: chr  "MTP1" "MTP1" "MTP1" "MTP1" ...
##  $ DATE    : chr  "01/01/2016" "02/01/2016" "03/01/2016" "04/01/2016" ...
##  $ PREC    : num  0 0 0 0.01 1.61 0.8 0.3 0 0 0 ...
##  $ TPMAX   : int  30 31 28 35 25 24 25 28 26 28 ...
##  $ TPMIN   : int  9 13 10 21 15 12 11 10 12 17 ...
##  $ TPMOY   : logi  NA NA NA NA NA NA ...
##  $ VTMOY   : num  2.46 2.01 0.67 1.34 2.46 2.91 1.79 2.24 1.12 1.12 ...
##  $ VTHM    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ VTDIR   : int  10 270 150 270 140 80 20 280 90 270 ...
##  $ VTVIT   : num  8.1 6 10.1 8.1 10.1 12.1 10.1 8.1 6 6.9 ...
##  $ MET1    : int  0 0 0 0 1 1 1 0 0 0 ...
##  $ MET2    : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ MET3    : int  1 0 0 0 0 0 0 0 0 0 ...
```

```r
# Statistiques descriptives pour les attributs numériques
summary(donnees)
```

```
##      IDSM             LABELLOC             DATE                PREC        
##  Length:1827        Length:1827        Length:1827        Min.   :0.00000  
##  Class :character   Class :character   Class :character   1st Qu.:0.00000  
##  Mode  :character   Mode  :character   Mode  :character   Median :0.00000  
##                                                           Mean   :0.03572  
##                                                           3rd Qu.:0.00000  
##                                                           Max.   :2.67000  
##                                                                            
##      TPMAX          TPMIN        TPMOY              VTMOY             VTHM     
##  Min.   :18.0   Min.   : 4.00   Mode:logical   Min.   :0.000   Min.   : 103  
##  1st Qu.:36.0   1st Qu.:19.00   NA's:1827      1st Qu.:1.120   1st Qu.:1338  
##  Median :42.0   Median :24.00                  Median :1.570   Median :1442  
##  Mean   :42.5   Mean   :24.12                  Mean   :1.676   Mean   :1405  
##  3rd Qu.:49.0   3rd Qu.:30.00                  3rd Qu.:2.010   3rd Qu.:1545  
##  Max.   :77.0   Max.   :45.00                  Max.   :8.050   Max.   :2314  
##                                                NA's   :4       NA's   :1765  
##      VTDIR            VTVIT             MET1             MET2        
##  Min.   : 10.0   Min.   : 2.900   Min.   :0.0000   Min.   :0.00000  
##  1st Qu.:260.0   1st Qu.: 8.100   1st Qu.:0.0000   1st Qu.:0.00000  
##  Median :270.0   Median : 8.900   Median :0.0000   Median :0.00000  
##  Mean   :254.1   Mean   : 9.062   Mean   :0.3016   Mean   :0.02135  
##  3rd Qu.:270.0   3rd Qu.:10.100   3rd Qu.:1.0000   3rd Qu.:0.00000  
##  Max.   :360.0   Max.   :21.000   Max.   :1.0000   Max.   :1.00000  
##  NA's   :5       NA's   :4                                          
##       MET3       
##  Min.   :0.0000  
##  1st Qu.:0.0000  
##  Median :0.0000  
##  Mean   :0.4713  
##  3rd Qu.:1.0000  
##  Max.   :1.0000  
## 
```

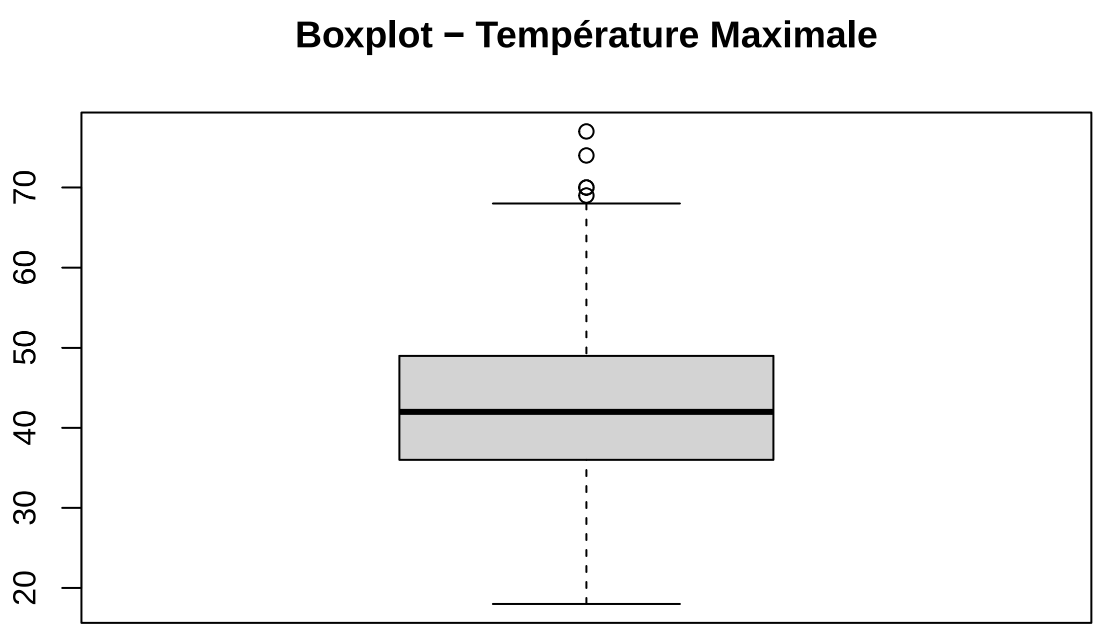
```r
# le nombre de valeurs uniques pour chaque attribut
sapply(donnees, function(x) length(unique(x)))
```

```
##      IDSM LABELLOC     DATE     PREC    TPMAX    TPMIN    TPMOY    VTMOY
##         1        1     1827       77       52       42        1       33
##      VTHM    VTDIR   VTVIT     MET1     MET2     MET3
##        58       30      18        2        2        2
```

```r
# Identifier et compter les valeurs manquantes par attribut
sapply(donnees, function(x) sum(is.na(x)))
```

```
##      IDSM LABELLOC     DATE     PREC    TPMAX    TPMIN    TPMOY    VTMOY
##         0        0        0        0        0        0     1827        4
##      VTHM    VTDIR   VTVIT     MET1     MET2     MET3
##      1765        5       4        0        0        0
```

```r
boxplot(donnees$TPMAX, main="Boxplot - Température Maximale")
```

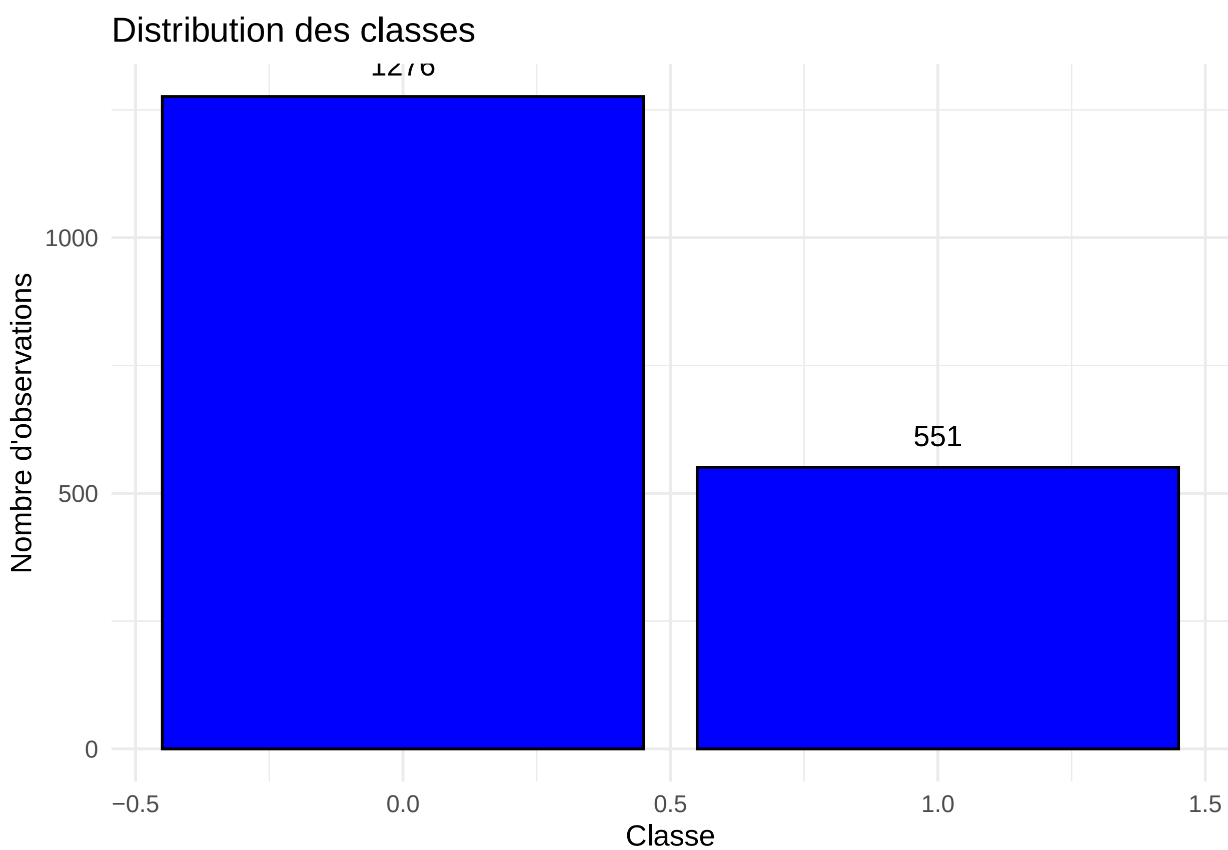



```r
table(donnees$MET1)
```

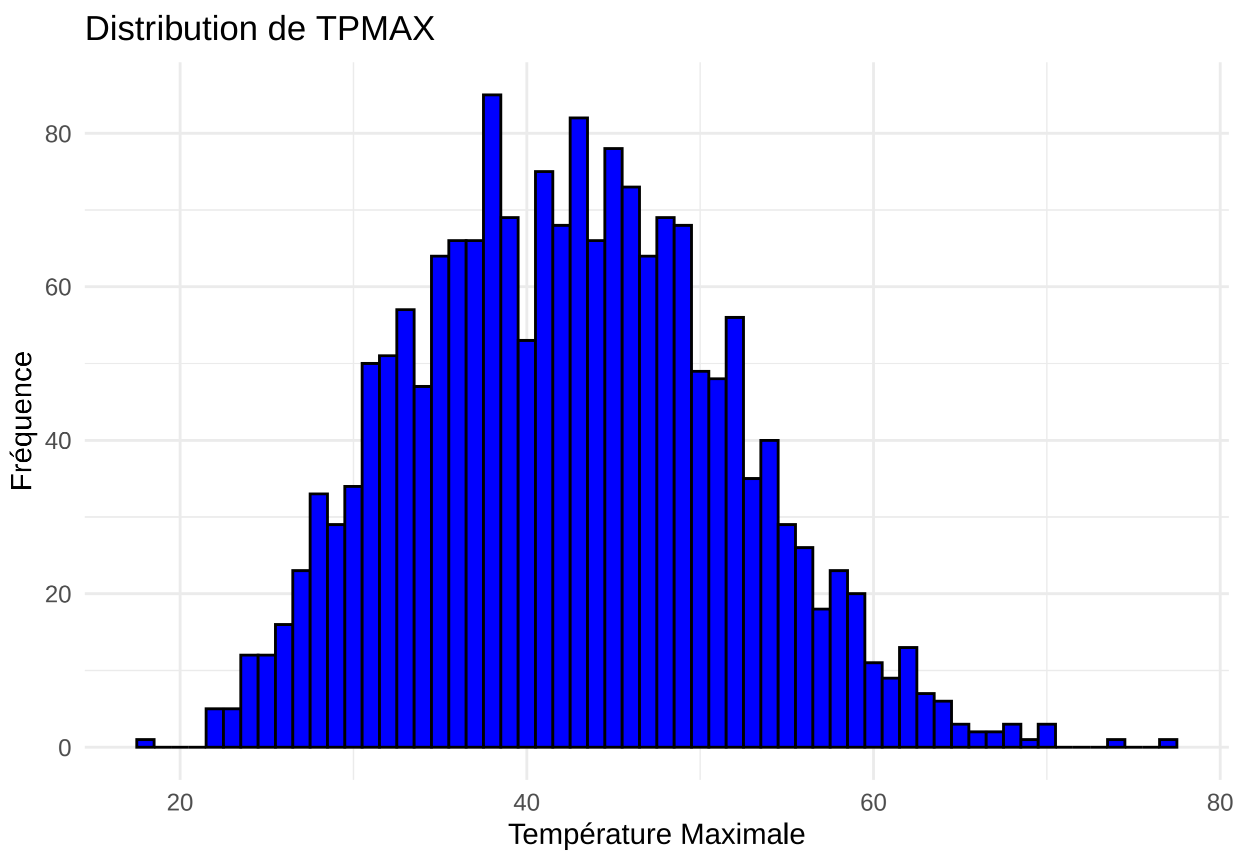
```
##
##    0    1
## 1276  551
```

```r
# **************** data viz + (equilibrage des données)******************
library(ggplot2)
ggplot(donnees, aes(x = MET1)) +
  geom_bar(fill = "blue", color = "black") +
  theme_minimal() +
  labs(title = "Distribution des classes ", x = "Classe", y = "Nombre d'observations") +
  geom_text(stat='count', aes(label=..count..), vjust=-1)
```

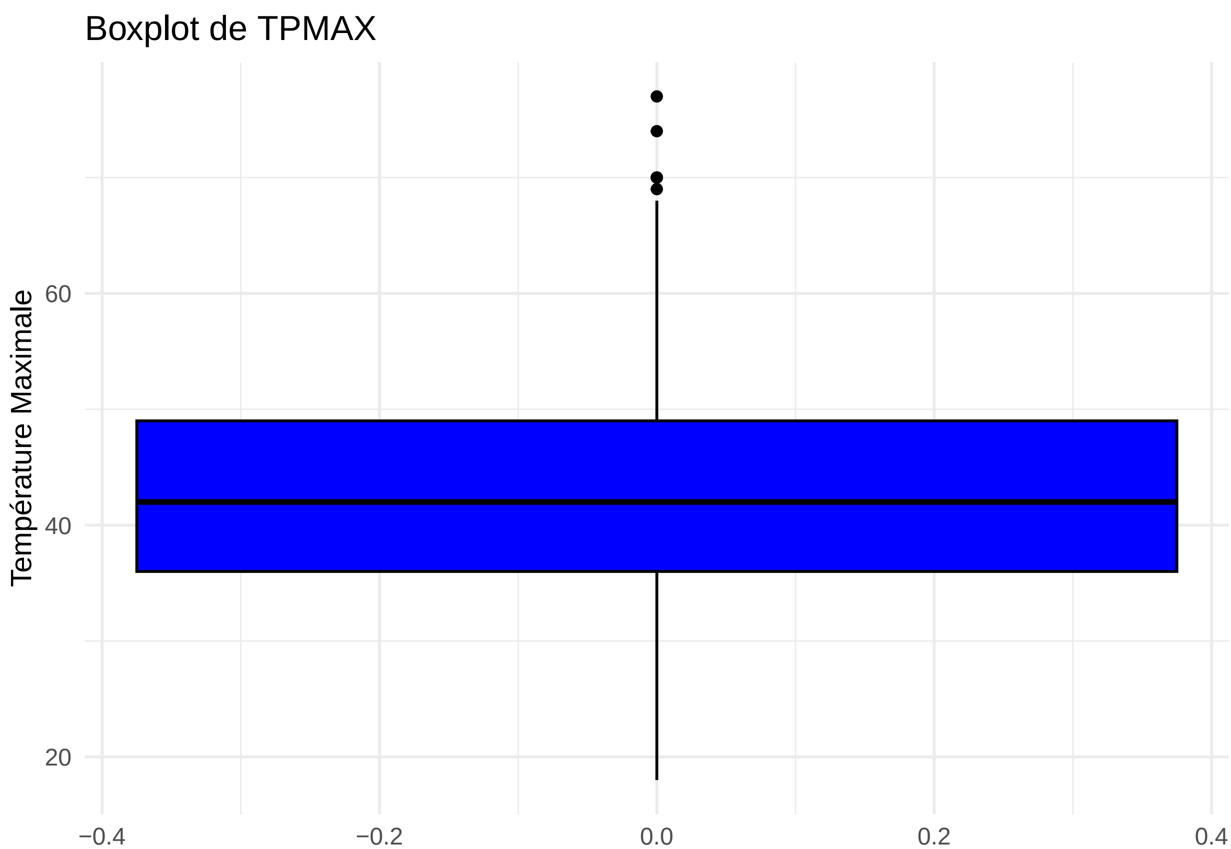
```
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```r
# Histogramme pour TPMAX
ggplot(donnees, aes(x = TPMAX)) +
  geom_histogram(fill = "blue", color = "black", binwidth = 1) +
  theme_minimal() +
  labs(title = "Distribution de TPMAX", x = "Température Maximale", y = "Fréquence")
```




```r
# Boxplot pour TPMAX
ggplot(donnees, aes(y = TPMAX)) +
  geom_boxplot(fill = "blue", color = "black") +
  theme_minimal() +
  labs(title = "Boxplot de TPMAX", y = "Température Maximale")
```




```r
# ****************** (etude correlation) ********************************

numeric_columns <- donnees[sapply(donnees, is.numeric)]

corr_matrix <- cor(numeric_columns)
print(corr_matrix)
```

```
##               PREC       TPMAX       TPMIN VTMOY VTHM VTDIR VTVIT        MET1
## PREC   1.000000000 -0.29233842 -0.14281093    NA   NA    NA    NA  0.25232677
## TPMAX -0.292338424  1.00000000  0.72854404    NA   NA    NA    NA -0.18503424
## TPMIN -0.142810931  0.72854404  1.00000000    NA   NA    NA    NA -0.07073798
## VTMOY           NA          NA          NA     1   NA    NA    NA          NA
## VTHM            NA          NA          NA    NA    1    NA    NA          NA
## VTDIR           NA          NA          NA    NA   NA     1    NA          NA
## VTVIT           NA          NA          NA    NA   NA    NA     1          NA
## MET1   0.252326773 -0.18503424 -0.07073798    NA   NA    NA    NA  1.00000000
## MET2   0.005126083 -0.02133547 -0.07272846    NA   NA    NA    NA  0.22474916
## MET3  -0.112099844  0.15066190  0.17359380    NA   NA    NA    NA  0.35200456
##               MET2       MET3
## PREC   0.005126083 -0.1120998
## TPMAX -0.021335466  0.1506619
## TPMIN -0.072728465  0.1735938
## VTMOY           NA         NA
## VTHM            NA         NA
## VTDIR           NA         NA
```

```
## VTVIT           NA         NA
## MET1   0.224749164  0.3520046
## MET2   1.000000000  0.1260902
## MET3   0.126090216  1.0000000
```

```
highly_correlated <- findCorrelation(corr_matrix, cutoff=0.8)
highly_correlated_vars <- names(numeric_columns)[highly_correlated]

print(highly_correlated_vars)
```

```
## character(0)
```

```
# ------------question 2 (version 2) ----------------------------------
library(dplyr)

# Suppression des colonnes non nécessaires
donnees <- donnees[, !names(donnees) %in% c("IDSM", "DATE", "LABELLOC", "VTHM")]

# taux de valeurs manquantes pour chaque attribut
taux_valeurs_manquantes <- sapply(donnees, function(x) sum(is.na(x)) / nrow(donnees))
donnees <- donnees[, taux_valeurs_manquantes < 0.9]

# Remplacement des valeurs manquantes par la moyenne pour les attributs numériques
donnees[is.na(donnees)] <- sapply(donnees[is.na(donnees)], function(x) mean(x, na.rm = TRUE))

# Pour VTMOY
donnees$VTMOY[is.na(donnees$VTMOY)] <- mean(donnees$VTMOY, na.rm = TRUE)

# Pour VTDIR
donnees$VTDIR[is.na(donnees$VTDIR)] <- mean(donnees$VTDIR, na.rm = TRUE)

# Pour VTVIT
donnees$VTVIT[is.na(donnees$VTVIT)] <- mean(donnees$VTVIT, na.rm = TRUE)

# Suppression des instances avec des données aberrantes pour TPMAX
limite_sup <- mean(donnees$TPMAX, na.rm = TRUE) + 3 * sd(donnees$TPMAX, na.rm = TRUE)
donnees <- donnees[donnees$TPMAX <= limite_sup, ]

# Vérifier si la colonne `MET3` existe avant de la supprimer
if ("MET3" %in% names(donnees)) {
  donnees <- select(donnees, -MET3)
}

# Sélectionner et normaliser les colonnes spécifiques via z-score
colonnes_a_normaliser <- c("TPMAX", "TPMIN", "VTMOY", "VTDIR", "VTVIT", "PREC")
donnees[, colonnes_a_normaliser] <- scale(donnees[, colonnes_a_normaliser])

head(donnees)
```

```
##         PREC      TPMAX      TPMIN      VTMOY      VTDIR      VTVIT MET1 MET2
## 1 -0.1884054 -1.3779254 -2.0973348  0.8629724 -4.4649986 -0.4769549    0    0
## 2 -0.1884054 -1.2669556 -1.5410579  0.3677926  0.2893355 -1.5189661    0    0
## 3 -0.1884054 -1.5998649 -1.9582656 -1.1067428 -1.9049725  0.5154368    0    0
```

```
## 4 -0.1358043 -0.8230765 -0.4285042 -0.3694751  0.2893355 -0.4769549    0    0
## 5  8.2803699 -1.9327742 -1.2629195  0.8629724 -2.0878315  0.5154368    1    1
## 6  4.0196817 -2.0437440 -1.6801272  1.3581522 -3.1849856  1.5078285    1    0
```

```r
library(dplyr)
library(smotefamily)
library(ROSE)
library(caret)
library(pROC)
library(randomForest)
#cat("F1-Score sur l'ensemble de test: ", f1_score, "\n")
#cat("AUC sur l'ensemble de test: ", auc_value, "\n")


# Équilibrage des classes avec suréchantillonnage
class_counts <- table(donnees$MET1)
max_class_count <- max(class_counts)

donnees_balanced <- lapply(split(donnees, donnees$MET1), function(x) {
  sample_frac(x, max_class_count / nrow(x), replace = TRUE)
}) %>% bind_rows()

# Mise à jour des données pour l'entraînement
donnees <- donnees_balanced
```

```r
#----------------(question 3)--------------------------------

#install.packages("ranger")
#install.packages("C50")
#install.packages("naivebayes")
#install.packages("xgboost")
#install.packages("kernlab")
#install.packages("RSNNS")

library(caret)
library(randomForest)
library(nnet)
library(naivebayes)
library(e1071) # Pour Naive Bayes et SVM
library(class) # Pour k-NN
library(C50)
library(ranger)
library(xgboost)
library(kernlab)



# Préparation des données (exemple simplifié)
donnees$MET1 <- as.factor(donnees$MET1) # Assurer que la cible est un facteur

# Division en ensemble d'apprentissage et de test
set.seed(123) # Pour la reproductibilité
index <- createDataPartition(donnees$MET1, p = .8, list = FALSE)
trainData <- donnees[index,]
```

```r
testData <- donnees[-index,]

trainData$MET1 <- as.factor(trainData$MET1)
testData$MET1 <- as.factor(testData$MET1)

# ************Entraîner un modèle C5.0******************
model_c50 <- C5.0(MET1 ~ ., data = trainData)

# Faire des prédictions sur l'ensemble de test
predictions_c50 <- predict(model_c50, testData)

# Calculer et afficher la matrice de confusion
confusionMatrix(predictions_c50, testData$MET1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 205  67
##          1  49 187
##
##                Accuracy : 0.7717
##                  95% CI : (0.7326, 0.8075)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5433
##
##  Mcnemar's Test P-Value : 0.1145
##
##             Sensitivity : 0.8071
##             Specificity : 0.7362
##          Pos Pred Value : 0.7537
##          Neg Pred Value : 0.7924
##              Prevalence : 0.5000
##          Detection Rate : 0.4035
##    Detection Prevalence : 0.5354
##       Balanced Accuracy : 0.7717
##
##        'Positive' Class : 0
##
```

```r
# ******resultat des modeles (base line) ******************************
train_control <- trainControl(method="cv", number=10)

# Modèles
model_rf <- train(MET1 ~ ., data=trainData, method="rf", trControl=train_control)
model_nb <- train(MET1 ~ ., data=trainData, method="naive_bayes", trControl=train_control)
model_knn <- train(MET1 ~ ., data=trainData, method="knn", trControl=train_control)
model_logistic <- train(MET1 ~ ., data=trainData, method="multinom", trControl=train_control)
```

```
## # weights:  9 (8 variable)
```

```
## initial  value 1269.152488
## iter  10 value 1081.618034
## iter  20 value 1065.104446
## iter  30 value 1064.969671
## final  value 1064.962996
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1084.414439
## iter  20 value 1068.602636
## final  value 1068.592491
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1081.620875
## iter  20 value 1065.109705
## iter  30 value 1064.980143
## final  value 1064.976976
## converged
## # weights:  9 (8 variable)
## initial  value 1269.845635
## iter  10 value 1068.687999
## iter  20 value 1055.781988
## iter  30 value 1055.643575
## final  value 1055.624706
## converged
## # weights:  9 (8 variable)
## initial  value 1269.845635
## iter  10 value 1069.411113
## iter  20 value 1059.168687
## final  value 1059.164436
## converged
## # weights:  9 (8 variable)
## initial  value 1269.845635
## iter  10 value 1068.688726
## iter  20 value 1055.787082
## iter  30 value 1055.652087
## final  value 1055.638188
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1078.401150
## iter  20 value 1066.269275
## iter  30 value 1066.119083
## final  value 1066.111474
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1080.897076
## iter  20 value 1069.582212
## final  value 1069.579674
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
```

```
## iter  10 value 1078.403710
## iter  20 value 1066.274284
## iter  30 value 1066.129061
## final  value 1066.125138
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1086.152729
## iter  20 value 1056.794854
## iter  30 value 1056.670654
## final  value 1056.659809
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1090.725061
## iter  20 value 1060.392441
## final  value 1060.373690
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1086.157247
## iter  20 value 1056.800236
## iter  30 value 1056.680361
## final  value 1056.673765
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1105.931641
## iter  20 value 1068.793577
## iter  30 value 1068.635487
## final  value 1068.622936
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1112.114920
## iter  20 value 1072.049050
## final  value 1072.045527
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1105.937676
## iter  20 value 1068.798443
## iter  30 value 1068.644500
## final  value 1068.636384
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1086.864093
## iter  20 value 1068.272280
## iter  30 value 1068.122300
## final  value 1068.114667
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
```

```
## iter  10 value 1090.346108
## iter  20 value 1071.599318
## final  value 1071.596246
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1086.867585
## iter  20 value 1068.277335
## iter  30 value 1068.132340
## final  value 1068.128402
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1075.855731
## iter  20 value 1058.310033
## iter  30 value 1058.220620
## final  value 1058.215668
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1078.550759
## iter  20 value 1061.942782
## final  value 1061.914876
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1075.858463
## iter  20 value 1058.315748
## iter  30 value 1058.231603
## final  value 1058.229699
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1080.690430
## iter  20 value 1059.169337
## iter  30 value 1059.004130
## final  value 1058.995563
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1084.617293
## final  value 1062.566672
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1080.694345
## iter  20 value 1059.174414
## iter  30 value 1059.014165
## final  value 1059.009466
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1096.053047
## iter  20 value 1068.536321
```

```
## iter  30 value 1068.373623
## final  value 1068.366252
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1098.198143
## final  value 1071.745977
## converged
## # weights:  9 (8 variable)
## initial  value 1268.459340
## iter  10 value 1096.054323
## iter  20 value 1068.541137
## iter  30 value 1068.383488
## final  value 1068.379743
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1072.817704
## iter  20 value 1060.591668
## iter  30 value 1060.445902
## final  value 1060.435406
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1074.969231
## iter  20 value 1064.030327
## final  value 1064.020546
## converged
## # weights:  9 (8 variable)
## initial  value 1269.152488
## iter  10 value 1072.819923
## iter  20 value 1060.596775
## iter  30 value 1060.455523
## final  value 1060.449195
## converged
## # weights:  9 (8 variable)
## initial  value 1409.861365
## iter  10 value 1199.028012
## iter  20 value 1184.766055
## final  value 1184.764052
## converged
```

```
model_mlp <- train(MET1 ~ ., data=trainData, method="mlp", trControl=train_control, preProcess = "scale
model_c50 <- train(MET1 ~ ., data=trainData, method="C5.0", trControl=train_control)
```

```
## Warning: 'trials' should be <= 9 for this object. Predictions generated using 9
## trials

## Warning: 'trials' should be <= 9 for this object. Predictions generated using 9
## trials

## Warning: 'trials' should be <= 8 for this object. Predictions generated using 8
## trials
```

```
## Warning: 'trials' should be <= 8 for this object. Predictions generated using 8
## trials

## Warning: 'trials' should be <= 7 for this object. Predictions generated using 7
## trials

## Warning: 'trials' should be <= 7 for this object. Predictions generated using 7
## trials

## Warning: 'trials' should be <= 8 for this object. Predictions generated using 8
## trials

## Warning: 'trials' should be <= 8 for this object. Predictions generated using 8
## trials

## Warning: 'trials' should be <= 7 for this object. Predictions generated using 7
## trials

## Warning: 'trials' should be <= 7 for this object. Predictions generated using 7
## trials

## Warning: 'trials' should be <= 9 for this object. Predictions generated using 9
## trials

## Warning: 'trials' should be <= 9 for this object. Predictions generated using 9
## trials
```

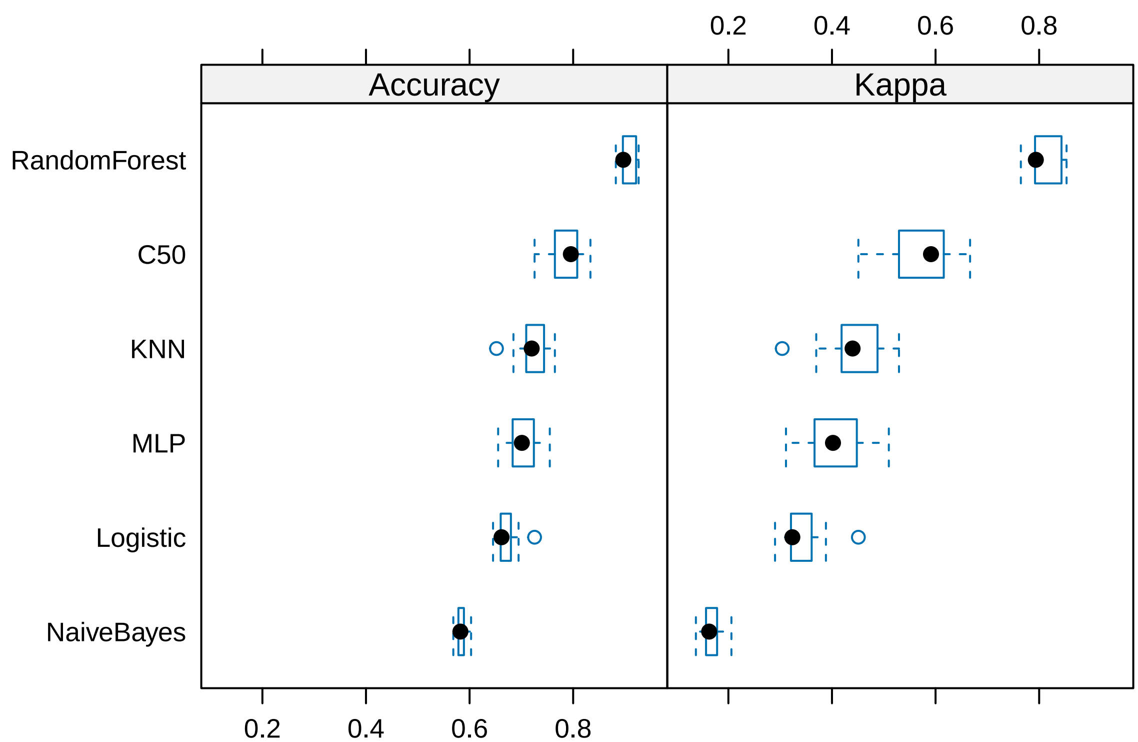
```r
# Comparaison des résultats
results <- resamples(list(RandomForest=model_rf, NaiveBayes=model_nb, KNN=model_knn,
                          Logistic=model_logistic, MLP=model_mlp, C50=model_c50))

# Afficher les résultats de la comparaison
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: RandomForest, NaiveBayes, KNN, Logistic, MLP, C50
## Number of resamples: 10
##
## Accuracy
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RandomForest 0.8823529 0.8961676 0.8968053 0.9031164 0.9178922 0.9264706    0
## NaiveBayes   0.5686275 0.5786255 0.5823071 0.5845560 0.5888905 0.6029412    0
## KNN          0.6519608 0.7097158 0.7199242 0.7197805 0.7435253 0.7647059    0
## Logistic     0.6453202 0.6605151 0.6617647 0.6710719 0.6765189 0.7254902    0
## MLP          0.6551724 0.6847900 0.7009804 0.7035095 0.7241379 0.7549020    0
## C50          0.7254902 0.7693422 0.7955665 0.7876485 0.8078818 0.8333333    0
##
## Kappa
```

```
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RandomForest 0.7647059 0.7923064 0.7936432 0.8062278 0.8357843 0.8529412    0
## NaiveBayes   0.1372549 0.1572510 0.1628694 0.1691132 0.1777810 0.2058824    0
## KNN          0.3039216 0.4193152 0.4398286 0.4395869 0.4872650 0.5294118    0
## Logistic     0.2900719 0.3214875 0.3235294 0.3421828 0.3539335 0.4509804    0
## MLP          0.3111973 0.3695989 0.4019608 0.4070234 0.4479141 0.5098039    0
## C50          0.4509804 0.5385660 0.5911740 0.5752663 0.6158381 0.6666667    0
```

```r
bwplot(results)
```




```r
# ****************gridsearch cv (random forest)*************************
# Nombre de variables explicatives
numVars <- ncol(trainData) - 1

# Définition de la grille de recherche pour Random Forest
rfGrid <- expand.grid(
  mtry = seq(2, numVars, by = 2),
  splitrule = c("gini", "extratrees"),
  min.node.size = c(1, 3, 5)
)

# Configuration de la validation croisée
train_control <- trainControl(method = "cv", number = 10, search = "grid")

# Entraînement du modèle Random Forest avec recherche de grille
```

```r
set.seed(123)
model_rf <- train(
  MET1 ~ .,
  data = trainData,
  method = "ranger",
  trControl = train_control,
  tuneGrid = rfGrid,
  metric = "Accuracy"
)

# Affichage des meilleurs paramètres
print(model_rf$bestTune)
```

```
##    mtry splitrule min.node.size
## 13    6      gini             1
```

```r
# Prédiction et évaluation sur l'ensemble de test
predictions_rf <- predict(model_rf, testData)
confusionMatrix(predictions_rf, testData$MET1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 230  25
##          1  24 229
##
##                Accuracy : 0.9035
##                  95% CI : (0.8745, 0.9278)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8071
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9055
##             Specificity : 0.9016
##          Pos Pred Value : 0.9020
##          Neg Pred Value : 0.9051
##              Prevalence : 0.5000
##          Detection Rate : 0.4528
##    Detection Prevalence : 0.5020
##       Balanced Accuracy : 0.9035
##
##        'Positive' Class : 0
##
```

```r
# ****************************gridsearch cv (MLP )**************************

mlpGrid <- expand.grid(
  size = c(5,8,10, 15,20),  # Nombre de neurones dans la couche cachée
```

```r
  decay = c(0.4,0.3,0.1, 0.001, 0.0001)  # Paramètre de régularisation
)

train_control <- trainControl(method="cv", number=10)

set.seed(123)
model_mlp <- train(
  MET1 ~ .,
  data=trainData,
  method="nnet",
  trControl=train_control,
  tuneGrid=mlpGrid,
  metric="Accuracy",
  linout=FALSE,  # FALSE pour la classification, TRUE pour la régression
  trace=FALSE  # Désactive l'affichage de la progression de l'entraînement
)

predictions_mlp <- predict(model_mlp, testData)
confusionMatrix(predictions_mlp, testData$MET1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 199  63
##          1  55 191
##
##                Accuracy : 0.7677
##                  95% CI : (0.7285, 0.8038)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5354
##
##  Mcnemar's Test P-Value : 0.5193
##
##             Sensitivity : 0.7835
##             Specificity : 0.7520
##          Pos Pred Value : 0.7595
##          Neg Pred Value : 0.7764
##              Prevalence : 0.5000
##          Detection Rate : 0.3917
##    Detection Prevalence : 0.5157
##       Balanced Accuracy : 0.7677
##
##        'Positive' Class : 0
##
```

```r
# ****************************gridsearch cv (C5.0)*************************

c50Grid <- expand.grid(
  .trials = c(1,7,9),  # Nombre de subdivisions pour le boosting
  .model = c("tree", "rules"),  # Type de modèle: arbre ou ensemble de règles
```

```r
  .winnow = c(TRUE, FALSE)  # Activation de la sélection d'attributs
)

train_control <- trainControl(method="cv", number=10)


set.seed(123)
model_c50 <- train(
  MET1 ~ .,
  data=trainData,
  method="C5.0",
  trControl=train_control,
  tuneGrid=c50Grid,
  metric="Accuracy"
)
```

```
## Warning: 'trials' should be <= 5 for this object. Predictions generated using 5
## trials

## Warning: 'trials' should be <= 5 for this object. Predictions generated using 5
## trials

## Warning: 'trials' should be <= 5 for this object. Predictions generated using 5
## trials

## Warning: 'trials' should be <= 5 for this object. Predictions generated using 5
## trials

## Warning: 'trials' should be <= 5 for this object. Predictions generated using 5
## trials

## Warning: 'trials' should be <= 5 for this object. Predictions generated using 5
## trials
```

```r
print(model_c50$bestTune)
```

```
##    trials model winnow
## 12      9 rules   TRUE
```

```r
predictions_c50 <- predict(model_c50, testData)
confusionMatrix(predictions_c50, testData$MET1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 217  70
##          1  37 184
##
##                Accuracy : 0.7894
##                  95% CI : (0.7513, 0.824)
##     No Information Rate : 0.5
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5787
##
##  Mcnemar's Test P-Value : 0.001978
##
##             Sensitivity : 0.8543
##             Specificity : 0.7244
##          Pos Pred Value : 0.7561
##          Neg Pred Value : 0.8326
##              Prevalence : 0.5000
##          Detection Rate : 0.4272
##    Detection Prevalence : 0.5650
##       Balanced Accuracy : 0.7894
##
##        'Positive' Class : 0
##
```

```r
# ****************************gridsearch cv (adaboost)*************************
install.packages("adabag")
```

```
## Warning: le package 'adabag' est en cours d'utilisation et ne sera pas installé
```

```r
library(adabag)
library(caret)
library(pROC)

# Préparation des données
trainData$MET1 <- as.factor(trainData$MET1)
testData$MET1 <- as.factor(testData$MET1)

# Définir les valeurs pour mfinal pour la recherche sur grille
mfinal_values <- c(10, 50, 100)  # Nombre de répétitions pour AdaBoost
coeflearn_values <- c(1, 0.5, 0.1) # Taux d'apprentissage

# Initialiser un vecteur pour stocker les résultats
results <- data.frame(mfinal = mfinal_values,coeflearn = coeflearn_values, Accuracy = numeric(length(mf

# Boucle sur la grille de paramètres
for(i in seq_along(mfinal_values)) {
  set.seed(123)  # Pour la reproductibilité
  # Entraînement du modèle AdaBoost
  model <- boosting(MET1 ~ ., data = trainData, mfinal = mfinal_values[i], coeflearn = "Freund")

  # Prédiction sur l'ensemble de test
  predictions <- predict(model, newdata = testData, type = "class")

  predicted_classes <- predictions$class

  predicted_classes <- factor(predicted_classes, levels = levels(testData$MET1))

  # Calcul de l'Accuracy
  cm <- confusionMatrix(predicted_classes, testData$MET1)
```

```r
  accuracy <- cm$overall['Accuracy']

  # Stockage des résultats
  results$Accuracy[i] <- accuracy
}

print(results)
```

```
##   mfinal coeflearn  Accuracy
## 1     10       1.0 0.7500000
## 2     50       0.5 0.7755906
## 3    100       0.1 0.8011811
```

```r
# ***************************gridsearch cv (xgboost)*************************

xgbGrid <- expand.grid(
  nrounds = 50,  # Réduire le nombre de rondes
  eta = c(0.1, 0.3),  # Moins de valeurs pour le taux d'apprentissage
  max_depth = c(3, 6),  # Moins de profondeur
  gamma = c(0, 0.1),  # Moins de valeurs pour gamma
  colsample_bytree = c(0.8),  # Moins de valeurs pour colsample_bytree
  min_child_weight = c(1, 10),  # Éventail plus large pour aider à prévenir le surapprentissage
  subsample = c(0.75)  # Moins de valeurs pour subsample
)

train_control <- trainControl(
  method = "cv",
  number = 5,  # Réduire le nombre de plis pour la CV
  allowParallel = TRUE,
  verboseIter = FALSE  # Réduire la verbosité pour accélérer
)

set.seed(123)
model_xgb <- train(
  MET1 ~ .,
  data = trainData,
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = xgbGrid,
  metric = "Accuracy"
)

print(model_xgb$bestTune)
```

```
##    nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 15      50         6 0.3   0.1              0.8                1      0.75
```

```r
predictions_xgb <- predict(model_xgb, testData)
confusionMatrix(predictions_xgb, testData$MET1)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   0   1
##          0 216  54
##          1  38 200
##
##                Accuracy : 0.8189
##                  95% CI : (0.7826, 0.8514)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6378
##
##  Mcnemar's Test P-Value : 0.1179
##
##             Sensitivity : 0.8504
##             Specificity : 0.7874
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.8403
##              Prevalence : 0.5000
##          Detection Rate : 0.4252
##    Detection Prevalence : 0.5315
##       Balanced Accuracy : 0.8189
##
##        'Positive' Class : 0
##
```

```r
# ***************gridsearch cv (svm (avec noyeau))************************
svmGrid <- expand.grid(
  sigma = 2^(-15:-3),  # Équivalent à l'inverse du paramètre gamma
  C = 2^(2:10)  # Paramètre de coût
)

train_control <- trainControl(method="cv", number=5, search="grid")

set.seed(123)
model_svm <- train(
  MET1 ~ .,
  data = trainData,
  method = "svmRadial",
  trControl = train_control,
  tuneGrid = svmGrid,
  metric = "Accuracy",
  preProcess = c("center", "scale")  # Il est recommandé de normaliser les données pour SVM
)

print(model_svm$bestTune)
```

```
##     sigma    C
## 117 0.125 1024
```

```r
predictions_svm <- predict(model_svm, testData)
confusionMatrix(predictions_svm, testData$MET1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 197  67
##          1  57 187
##
##                Accuracy : 0.7559
##                  95% CI : (0.7161, 0.7927)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5118
##
##  Mcnemar's Test P-Value : 0.419
##
##             Sensitivity : 0.7756
##             Specificity : 0.7362
##          Pos Pred Value : 0.7462
##          Neg Pred Value : 0.7664
##              Prevalence : 0.5000
##          Detection Rate : 0.3878
##    Detection Prevalence : 0.5197
##       Balanced Accuracy : 0.7559
##
##        'Positive' Class : 0
##
```

```
# **********model final : voting classifier *************************

predictions_rf <- predict(model_rf, testData, type = "raw")
predictions_svm <- predict(model_svm, testData, type = "raw")
predictions_xgb <- predict(model_xgb, testData, type = "raw")
predictions_mlp <- predict(model_mlp, testData, type = "raw")
predictions_c50 <- predict(model_c50, testData, type = "raw")


combined_predictions <- data.frame(predictions_rf, predictions_svm, predictions_xgb,predictions_mlp,pred

# Convertir en facteurs si ce ne sont pas déjà des facteurs
combined_predictions <- data.frame(lapply(combined_predictions, factor, levels = levels(testData$MET1)))

# Calculer le vote majoritaire
library(dplyr)
majority_vote <- combined_predictions %>%
  rowwise() %>%
  mutate(majority = names(sort(table(c(predictions_rf, predictions_svm, predictions_xgb)), decreasing =
  ungroup() %>%
  select(majority)

# Convertir en facteur
majority_vote <- factor(majority_vote$majority, levels = levels(testData$MET1))

confusionMatrix(majority_vote, testData$MET1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 221  45
##          1  33 209
##
##                Accuracy : 0.8465
##                  95% CI : (0.8121, 0.8767)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6929
##
##  Mcnemar's Test P-Value : 0.2129
##
##             Sensitivity : 0.8701
##             Specificity : 0.8228
##          Pos Pred Value : 0.8308
##          Neg Pred Value : 0.8636
##              Prevalence : 0.5000
##          Detection Rate : 0.4350
##    Detection Prevalence : 0.5236
##       Balanced Accuracy : 0.8465
##
##        'Positive' Class : 0
##
```

```r
# *********************** resultat final ******************************
library(dplyr)

# Prédictions déjà faites
# predictions_rf, predictions_svm, predictions_xgb, predictions_mlp, predictions_c50

# Créer un data.frame pour comparer les résultats
results_comparison <- data.frame(
  True = testData$MET1,
  RF = predictions_rf,
  SVM = predictions_svm,
  XGB = predictions_xgb,
  MLP = predictions_mlp,
  C50 = predictions_c50
)

# Fonction pour calculer le nombre d'instances correctement classées
calc_correct_predictions <- function(pred_col, true_col) {
  sum(pred_col == true_col)
}

# Appliquer la fonction à chaque modèle et stocker les résultats
num_correct <- sapply(results_comparison[-1], calc_correct_predictions, true_col = results_comparison$T

# Comparer le nombre d'instances correctement classées
num_correct <- tibble(Model = names(num_correct), CorrectlyClassified = num_correct)
```

```r
# Afficher le résultat
print(num_correct)
```

```
## # A tibble: 5 x 2
##   Model CorrectlyClassified
##   <chr>               <int>
## 1 RF                    459
## 2 SVM                   384
## 3 XGB                   416
## 4 MLP                   390
## 5 C50                   401
```