final2

Groupe F

`r Sys.Date

Resumer

on va se concentrer sur ces 10 variables: Price,Sales,Revenue,BSR,FBA.Fees,Active.Sellers.. ,Review.velocity,Ratings,Review.Count,Images nous allons faire notre analyse en fonction de la variable Revenue pour les regression et Price pour les classifications. nous allons analyser toutes les autres variables pour comprendre leur influence sur les Revenue. ## Package

```
#install.packages("dplyr")
#install.packages("outliers")
#install.packages("ggplot2")
#install.packages("EnvStats")
#install.packages("MASS")
#install.packages("car")
#install.packages("gbm")
#install.packages("ipred")
#install.packages("rpart")
#install.packages("caret")
#install.packages("glmnet")
```

Librairie

```
library(dplyr)
library(outliers)
library(ggplot2)
library(EnvStats)
library(MASS)
library(car)
library(gbm)
library(ipred)
library(rpart)
library(caret)
library(glmnet)
```

Chargement des données

```
#on charge les données
df<-read.csv("dataset.csv")

#créons un data contenant uniquement les variable numerique
data<-select_if(df, is.numeric)
```

```r
# Supprimons la variable "Weight" du dataset
data <- subset(data, select = -Weight)

# Afficher les premières lignes du nouveau dataframe
head(data)
```

```
##   Price Sales Revenue BSR FBA.Fees Active.Sellers.. Ratings Review.Count
Images Review.velocity
## 1  3.50 13466   47131   3     2.62               30       5         44069
12             868
## 2 18.28 13338  243819   2     8.38               30       5         40397
5              830
## 3  9.99 11194  111828   4     3.93               22       5          3827
6              235
## 4 14.87 13492  200626   2     5.19               11       5         28800
5              466
## 5 45.00 13377  601965   6    11.03                1       5         27494
9              223
## 6  8.00 11983   95864   5     6.16               21       5         11120
6              701
```

Maintenant qu'On a chargé nos données on va fais une étude de statistiques descriptives pour comprendre nos données,

Statistiques descriptives

```r
#dim(data)
```

Nous avons pours les 10 colonnes 6341 lignes.

Nous pouvons observé un résumé des données

```r
#resumé de données
#summary(data)
```

Nous observons beaucoup de données manquantes commme dans la variable FBA.Fees qui represente les frais qu'Amazon facture.

Variances

```r
# Calculer la variance de chaque variable en excluant les données manquantes
variance <- sapply(data, function(x) var(x, na.rm = TRUE))
# Définir les options pour afficher les valeurs sans notation scientifique
options(scipen = 999)

# Afficher la sortie de la variance avec les valeurs en notation scientifique
print(variance)
```

```
##              Price              Sales            Revenue
BSR          FBA.Fees   Active.Sellers..
```

```
##         158.173017     5905612.125601  1377545870.500346
38854825482.263039          12.402860          53.152643
##            Ratings       Review.Count             Images
Review.velocity
##           0.365398     9087884.080438           8.486415
62124.465325
```

Les données montrent une variance significative dans les ventes, le revenu, le classement des meilleures ventes et le nombre d'avis, indiquant une grande variabilité dans ces mesures, tandis que la cohérence est observée dans les prix et les frais FBA, suggérant une stabilité relative dans ces domaines.

Ecart type

```
# Calculer l'ecart type de chaque variable en excluant les données manquantes
ecart <- sapply(data, function(x) sd(x, na.rm = TRUE))
# Définir les options pour afficher les valeurs sans notation scientifique
options(scipen = 999)

# Afficher la sortie de l'ecart type avec les valeurs en notation
scientifique
print(ecart)
```

```
##            Price            Sales          Revenue              BSR
FBA.Fees Active.Sellers..
##       12.5766855     2430.1465235    37115.3050708   197116.2740168
3.5217694        7.2905859
##          Ratings     Review.Count           Images  Review.velocity
##        0.6044816     3014.6117628        2.9131452      249.2477990
```

Les évaluations présentent une faible dispersion, avec des valeurs fluctuant en moyenne de 0.60 unité autour de la moyenne, tandis que le classement des meilleurs vendeurs affiche une variabilité importante, avec des valeurs s'éloignant en moyenne de 197116.27 unités de la moyenne.

Coefficient de variation

```
#coefficient de variation
CV_func <- function(x){ # x est une variable d'entr?e
  CV <- ecart/variance
  return(CV)
  }

#sapply(data, CV_func)
```

Les coefficients de variation pour Price, Sales, Revenue, BSR, Review Count et Review Velocity sont tous très faibles, suggérant une dispersion relative relativement faible par rapport à leur moyenne et une stabilité relativement élevée. Cependant, le coefficient de variation pour Ratings est très élevé, indiquant une dispersion extrêmement élevée par rapport à sa moyenne, ce qui peut nécessiter une attention particulière lors de l'analyse en raison de données très variées ou d'une distribution asymétrique.

```
donnees<-c(1264,1802,1511,453,2636,51,828,492,50)
median(donnees)
```

```
## [1] 828
```

```
mean(donnees)
```

```
## [1] 1009.667
```

pour une moyenne de 1000 donnés manquantes sur les 6000.

En se basant sur ces interpretations nous pouvons observer une dispersion des données autour de la moyenne, donc nous allons procéder a ume imputation des données

Nettoyages des données

imputation des données

```
#on va se creer une fonction qui va le faire pour toutes les variables

impute <- function(d, col_name) {
  col <- d[[col_name]]
  nbre_missing <- sum(is.na(col))
  mu <- mean(col, na.rm = TRUE)
  sd <- sd(col, na.rm = TRUE)
  col[is.na(col)] <- rnorm(nbre_missing, mu, sd)
  return(col)
}
#on recupere la liste de nom des variables
liste_colonnes <- names(data)
data_copy<-data
#boucle
for (col_name in liste_colonnes) {
  data_copy[[col_name]] <- impute(data_copy, col_name)
}
#on verifie qu'il n'ya plus de données manquantes

sum(is.na(data_copy))
```

```
## [1] 0
```

effectivement, nous avons plus de données manquantes

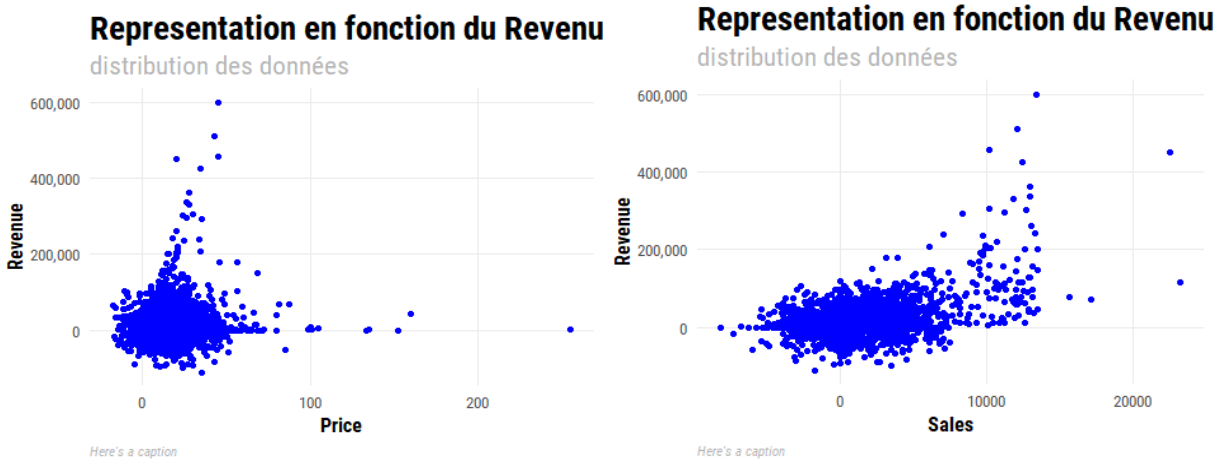
nous pouvons passer a la visualisation pour avoir un appercu des données

Visualisation

visualisation de certaines variables.

```
windowsFonts(`Roboto Condensed` = windowsFont("Roboto Condensed"))

visualisation <- function(df, x_var, y_var) {
  ggplot(df, aes_string(x = x_var, y = y_var)) +
```

```
    geom_point(color="blue")+
    scale_y_continuous(labels = scales::comma)   +  # Correction de la
parenthèse ici
    labs(x = x_var, y = y_var, color = y_var,
         title = "Representation en fonction du Revenue",
         subtitle = "distribution des données",
         caption = "Here's a caption")+
 theme_minimal(base_family = "Roboto Condensed", base_size = 12) +
 theme(panel.grid.minor = element_blank(),
 # Titre en gras et plus gros
 plot.title = element_text(face = "bold", size = rel(1.7)),
 # Sous-titre simple, légèrement plus grand et gris
 plot.subtitle = element_text(face = "plain", size = rel(1.3), color =
"grey70"),
 # Légende en italique, plus petite, grise et alignée à gauche
 plot.caption = element_text(face = "italic", size = rel(0.7),
 color = "grey70", hjust = 0),
 # Titres de légende en gras
 legend.title = element_text(face = "bold"),
 # Titres de facettes en gras, légèrement plus grands, alignés à gauche pour
des raisons de répétition
 strip.text = element_text(face = "bold", size = rel(1.1), hjust = 0),
 # Titres des axes en gras
 axis.title = element_text(face = "bold"),
 # Ajoutez un peu d'espace au-dessus du titre de l'axe des x et alignez-le à
gauche
 )
}

liste <- c("Price", "Sales")

# Utilisation de la fonction avec les variables spécifiées

 for (x_var in liste) {
  print(visualisation(data_copy, x_var, "Revenue"))
}
```

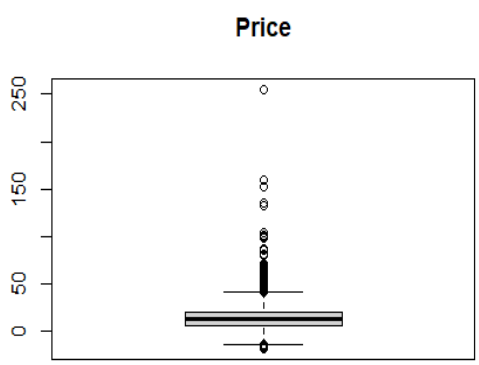


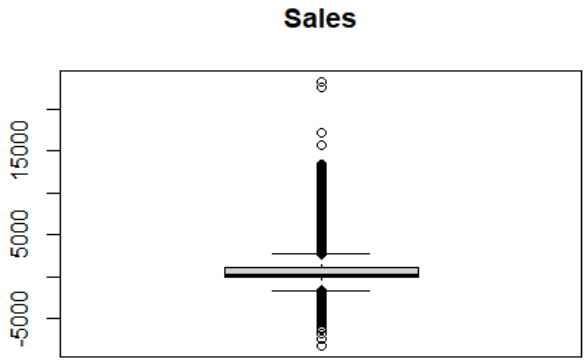
Nous observons de fortes concentrations des données vers un niveaux et une certaine dispersion des données.

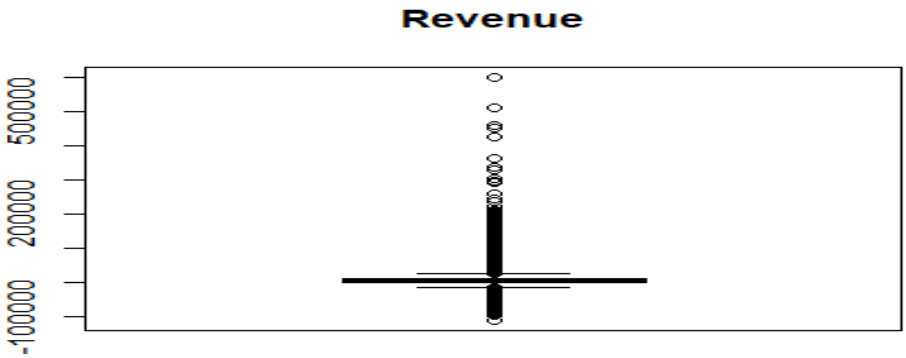
Outliers

Nous allons faire des graphes de boxplot pour visualiser sil ya des outliers ou pas

```
# Définir la liste des colonnes à afficher
listes <- c("Price", "Sales","Revenue")

# Modifier la fonction data_boxplots pour afficher uniquement les boxplots des variables dans liste_colonnes
data_boxplots <- function(e, columns) {
  for (col in names(e)) {
    if (col %in% columns) {
      boxplot(e[[col]], main = col)
    }
  }
}

# Appeler la fonction data_boxplots avec les données et la liste des colonnes spécifiée
data_boxplots(data_copy, listes)
```







Nous observons plusieurs outliers que nous allons essayer de gerer

on va observé avec une analyse univarié

```
#fonction qui renvoie les resultats du test de grab pour chacune des
variables
grubbs_test <- function(df) {
  results <- list()
  for (col in names(df)) {
    grubbs_result <- grubbs.test(df[[col]])
    results[[col]] <- grubbs_result
  }
  return(results)
}

#grubbs_test(data_copy)
```

tous les p value < 0.05 on rejette H0 donc les valeurs H1 sont des outliers

```
#pour l'option opposite maintenant
grubbs_test_opposite <- function(df) {
```

```
  results <- list()
  for (col in names(df)) {
    grubbs_result <- grubbs.test(df[[col]], opposite = TRUE)
    results[[col]] <- grubbs_result
  }
  return(results)
}

#grubbs_test_opposite(data_copy)
```

on remarque aussi des outliers dans les variables BSR et Review.count

on va faire le test de rosner

```
# calcul de la distance
distances <- mahalanobis(x = data_copy , center = colMeans(data_copy) , cov =
cov(data_copy))

# Cutoff value for distances from Chi-Sqaure Dist.
cutoff <- sqrt(qchisq(p = 0.95 , df = ncol(data_copy))) # ncol(air) est le
nombre de variable

## Display observation whose distance greater than cutoff value
#data_copy[distances > cutoff ,]

data_num_copy <- data_copy[distances < cutoff ,]
```

Nous observons des données contenant des signes - dans le cas des ventes il se peut que ca soit une vente annulé. dans le cas des revenues il se peut que comme la vente à été annulé on peut considerer

chercons les correlations entre les variables.

Correlation

```
#cor(data_num_copy)
```

il y a une forte corrélation entre Price et FBA.Fees; Sales et Revenue; il y a une corrélation modéré entre Active.Sellers.. et BSR il y a par contre beaucoup de faible corrélation entre les autres variables;

REGRESSION

on va partager nos données en données de test et d'entrainement.

```
# Division des données en ensembles d'entraînement et de test (70% pour
l'entraînement, 30% pour le test)
set.seed(1) # Définition de la graine aléatoire pour la reproductibilité
train_index <- sample(1:nrow(data_num_copy), 0.8 * nrow(data_num_copy))
train_data <- data_num_copy[train_index, ]
test_data <- data_num_copy[-train_index, ]
```

linéaire

```
model <- lm(formula = Revenue ~ . , data = train_data)
#model
#summary(model)
```

pour ameliorer le modele

```
#step(model, trace = TRUE)
```

le modele a choisi ces variables pour augmenter sa performance

```
model1<-lm(formula = Revenue ~ Price + Sales + BSR + FBA.Fees + Review.Count
+
    Images + Review.velocity, data = train_data)
#summary(model1)
```

ce qui suggère que ces variables ont un effet significatif sur la variable Revenue.

```
#test du model de regression linéaire de la prédiction de Revenue sur le data
test
#on va predire revenue
predictions1 <- predict(model1, newdata = test_data,
        interval = "prediction")

# Extraction des prédictions et des vraies valeurs
predicted_values <- predictions1[, 1]
lower_confidence <- predictions1[, 2]
upper_confidence <- predictions1[, 3]

# Calcul du coefficient de détermination (R²)
r_squared <- summary(model1)$r.squared
cat("Coefficient de détermination (R²) :", r_squared, "\n")

## Coefficient de détermination (R²) : 0.4187712

# Calcul de l'erreur quadratique moyenne (RMSE)
rmse <- sqrt(mean((test_data$Revenue - predicted_values)^2))
cat("Erreur quadratique moyenne (RMSE) :", rmse, "\n")

## Erreur quadratique moyenne (RMSE) : 7439.152
```

Ridge

```
# Convertir les données en format de matrice modèle pour l'ensemble
d'entraînement et de test
x_train <- model.matrix(Revenue ~ Price + Sales + BSR + FBA.Fees +
Review.Count + Images + Review.velocity, data = train_data)
x_test <- model.matrix(Revenue ~ Price + Sales + BSR + FBA.Fees +
Review.Count + Images + Review.velocity, data = test_data)

y_train <- train_data$Revenue
y_test <- test_data$Revenue
```

```r
# Définir la grille de valeurs de lambda
grid <- 10^seq(10, -2, length = 100)

# Ajuster le modèle de régression Ridge sur l'ensemble d'entraînement
ridge_mod <- glmnet(x_train, y_train, alpha = 0, lambda = grid)

# Sélectionner la valeur optimale de lambda en utilisant la validation
croisée
cv.out <- cv.glmnet(x_train, y_train, alpha = 0)
best_lambda <- cv.out$lambda.min

# Utiliser la valeur optimale de lambda pour faire des prédictions sur
l'ensemble de test
ridge_pred <- predict(ridge_mod, s = best_lambda, newx = x_test)
# Calculer le coefficient de détermination (R²)
ridge_r_squared <- cor(y_test, ridge_pred)^2
print(paste("Coefficient de détermination (R²) pour la régression Ridge :",
ridge_r_squared))
```

```
## [1] "Coefficient de détermination (R²) pour la régression Ridge :
0.438591983364453"
```

```r
# Calculer l'erreur quadratique moyenne sur l'ensemble de test
mse <- mean((ridge_pred - y_test)^2)
print(paste("Erreur quadratique moyenne sur l'ensemble de test :", mse))
```

```
## [1] "Erreur quadratique moyenne sur l'ensemble de test : 55345318.2251451"
```

lasso

```r
# Convertir les données en format de matrice modèle pour l'ensemble
d'entraînement et de test
x_train <- model.matrix(Revenue ~ Price + Sales + BSR + FBA.Fees +
Review.Count + Images + Review.velocity, data = train_data)
x_test <- model.matrix(Revenue ~ Price + Sales + BSR + FBA.Fees +
Review.Count + Images + Review.velocity, data = test_data)

y_train <- train_data$Revenue
y_test <- test_data$Revenue

# Définir la grille de valeurs de lambda
grid <- 10^seq(10, -2, length = 100)

# Ajuster le modèle de régression Lasso sur l'ensemble d'entraînement
lasso_mod <- glmnet(x_train, y_train, alpha = 1, lambda = grid)

# Sélectionner la valeur optimale de lambda en utilisant la validation
croisée
cv.out <- cv.glmnet(x_train, y_train, alpha = 1)
best_lambda <- cv.out$lambda.min
```

```
# Utiliser la valeur optimale de lambda pour faire des prédictions sur
l'ensemble de test
lasso_pred <- predict(lasso_mod, s = best_lambda, newx = x_test)
# Calculer le coefficient de détermination (R²)
lasso_r_squared <- cor(y_test, lasso_pred)^2
print(paste("Coefficient de détermination (R²) pour la régression Lasso :",
lasso_r_squared))
```

```
## [1] "Coefficient de détermination (R²) pour la régression Lasso :
0.437581539382135"
```

```
# Calculer l'erreur quadratique moyenne sur l'ensemble de test
mse <- mean((lasso_pred - y_test)^2)
print(paste("Erreur quadratique moyenne sur l'ensemble de test :", mse))
```

```
## [1] "Erreur quadratique moyenne sur l'ensemble de test : 55309706.9889148"
```

Analyse comparative Regression linéaire, ridge, lasso

Dans ce cas particulier, la régression linéaire semble être le meilleur modèle en termes de précision des prédictions sur l'ensemble de test, suivie de près par la régression Ridge et la régression Lasso. Cependant, les différences de performance entre les trois modèles sont relativement faibles.

CLASSIFICATION

on va se baser sur la variable Price

on va creer une nouvelle variable binaire pour savoir si le prix est élévé ou pas.

```
data_new<-data_num_copy
# Calculer la moyenne des prix
moyenne_prix <- mean(data_new$Price)

# Créer une variable binaire pour le prix
data_new$Prix_categorie <- ifelse(data_new$Price > moyenne_prix, "Prix
élevé", "Prix bas")

# Afficher les premières lignes du jeu de données avec la nouvelle variable
#head(data_new)

data_new$Prix_binaire <- ifelse(data_new$Prix_categorie == "Prix élevé", 1,
0)

# Supprimer la variable "Prix_categorie" du jeu de données
data_new <- subset(data_new, select = -Prix_categorie)
data_new <- subset(data_new, select = -Price)
```

```r
# Division des données en ensembles d'entraînement et de test (70% pour
l'entraînement, 30% pour le test)
set.seed(1) # Définition de la graine aléatoire pour la reproductibilité
train_index1 <- sample(1:nrow(data_new), 0.8 * nrow(data_new))
train_data1 <- data_new[train_index, ]
test_data1 <- data_new[-train_index, ]
```

Regression logistique

```r
# Installer le package caret si ce n'est pas déjà fait
# install.packages("caret")

# Charger le package caret

# Création du modèle de régression logistique
modele_logistique <- glm(Prix_binaire ~ ., data = train_data1, family =
binomial)
predict2<-predict(modele_logistique, newdata =test_data1 ,type = "response")


# Créer un objet de type confusionMatrix pour calculer les métriques
d'évaluation
confusion_matrix <- confusionMatrix(data = as.factor(ifelse(predict2 > 0.5,
1, 0)),
                                    reference =
as.factor(test_data1$Prix_binaire))

# Afficher les métriques d'évaluation
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 305 115
##          1  52 185
##
##                Accuracy : 0.7458
##                  95% CI : (0.7107, 0.7787)
##     No Information Rate : 0.5434
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.479
##
##  Mcnemar's Test P-Value : 0.000001605
##
##             Sensitivity : 0.8543
##             Specificity : 0.6167
##          Pos Pred Value : 0.7262
##          Neg Pred Value : 0.7806
##              Prevalence : 0.5434
```

```
##          Detection Rate : 0.4642
##    Detection Prevalence : 0.6393
##       Balanced Accuracy : 0.7355
##
##        'Positive' Class : 0
##
```

DecisionTreeClassifier

```
# Installer et charger le package "rpart" pour les arbres de décision
# install.packages("rpart")

# Créer le modèle d'arbre de décision
decision_tree_model <- rpart(Prix_binaire ~ ., data = train_data1, method =
"class")

# Faire des prédictions sur l'ensemble de test
predictions_tree <- predict(decision_tree_model, newdata = test_data1, type =
"class")

# Calculer les métriques d'évaluation
confusion_matrix_tree <- confusionMatrix(data = as.factor(predictions_tree),
                                         reference =
as.factor(test_data1$Prix_binaire))
print(confusion_matrix_tree)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 288  46
##          1  69 254
##
##                Accuracy : 0.825
##                  95% CI : (0.7937, 0.8533)
##     No Information Rate : 0.5434
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.6494
##
##  Mcnemar's Test P-Value : 0.04022
##
##             Sensitivity : 0.8067
##             Specificity : 0.8467
##          Pos Pred Value : 0.8623
##          Neg Pred Value : 0.7864
##              Prevalence : 0.5434
##          Detection Rate : 0.4384
##    Detection Prevalence : 0.5084
##       Balanced Accuracy : 0.8267
##
```

```
##        'Positive' Class : 0
##
```

RandomForest

```
# Installer et charger le package "randomForest"
# install.packages("randomForest")
library(randomForest)

random_forest_model <- randomForest(Prix_binaire ~ ., data = train_data1)

## Warning in randomForest.default(m, y, ...): The response has five or fewer
unique values.  Are you sure you want to
## do regression?

# Faire des prédictions sur l'ensemble de test avec des probabilités
predictions_forest_probs <- predict(random_forest_model, newdata =
test_data1, type = "response")

# Convertir les probabilités en classes prédites
predictions_forest <- ifelse(predictions_forest_probs > 0.5, 1, 0)

# Calculer les métriques d'évaluation
confusion_matrix_forest <- confusionMatrix(data =
as.factor(predictions_forest),
                                           reference =
as.factor(test_data1$Prix_binaire))
print(confusion_matrix_forest)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 314  32
##          1  43 268
##
##                Accuracy : 0.8858
##                  95% CI : (0.859, 0.9091)
##     No Information Rate : 0.5434
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.7706
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.8796
##             Specificity : 0.8933
##          Pos Pred Value : 0.9075
##          Neg Pred Value : 0.8617
##              Prevalence : 0.5434
##          Detection Rate : 0.4779
```

```
##    Detection Prevalence : 0.5266
##       Balanced Accuracy : 0.8864
##
##        'Positive' Class : 0
##
```

Bagging

```
# Installer et charger le package "ipred"
# install.packages("ipred")
# Créer le modèle de bagging
bagging_model <- bagging(Prix_binaire ~ ., data = train_data1)
# Faire des prédictions sur l'ensemble de test avec des probabilités
predictions_bagging_probs <- predict(bagging_model, newdata = test_data1,
type = "prob")

# Convertir les probabilités en classes prédites
predictions_bagging <- ifelse(predictions_bagging_probs > 0.5, 1, 0)

# Calculer les métriques d'évaluation
confusion_matrix_bagging <- confusionMatrix(data =
as.factor(predictions_bagging),
                                            reference =
as.factor(test_data1$Prix_binaire))
print(confusion_matrix_bagging)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 286  49
##          1  71 251
##
##                Accuracy : 0.8174
##                  95% CI : (0.7856, 0.8462)
##     No Information Rate : 0.5434
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.6341
##
##  Mcnemar's Test P-Value : 0.05523
##
##             Sensitivity : 0.8011
##             Specificity : 0.8367
##          Pos Pred Value : 0.8537
##          Neg Pred Value : 0.7795
##              Prevalence : 0.5434
##          Detection Rate : 0.4353
##    Detection Prevalence : 0.5099
##       Balanced Accuracy : 0.8189
##
```

```
##        'Positive' Class : 0
##
```

Boosting

```
# Installer et charger le package "gbm"
# install.packages("gbm")

# Créer le modèle de boosting
boosting_model <- gbm(Prix_binaire ~ ., data = train_data1, distribution =
"bernoulli", n.trees = 100)

# Faire des prédictions sur l'ensemble de test
predictions_boosting <- predict(boosting_model, newdata = test_data1, type =
"response", n.trees = 100)

# Convertir les probabilités en classes prédites
predictions_boosting_class <- ifelse(predictions_boosting > 0.5, 1, 0)

# Calculer les métriques d'évaluation
confusion_matrix_boosting <- confusionMatrix(data =
as.factor(predictions_boosting_class),
                                             reference =
as.factor(test_data1$Prix_binaire))
print(confusion_matrix_boosting)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 280  46
##          1  77 254
##
##                Accuracy : 0.8128
##                  95% CI : (0.7808, 0.8419)
##     No Information Rate : 0.5434
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.6258
##
##  Mcnemar's Test P-Value : 0.00683
##
##             Sensitivity : 0.7843
##             Specificity : 0.8467
##          Pos Pred Value : 0.8589
##          Neg Pred Value : 0.7674
##              Prevalence : 0.5434
##          Detection Rate : 0.4262
##    Detection Prevalence : 0.4962
##       Balanced Accuracy : 0.8155
##
```

```
##        'Positive' Class : 0
##
```

Le modèle Random Forest se distingue par sa plus haute précision, sensibilité et spécificité parmi les modèles évalués, ainsi qu'un coefficient Kappa élevé, démontrant sa fiabilité dans la prédiction des deux classes. Ses valeurs prédictives positives et négatives sont également solides, confirmant sa capacité à fournir des prédictions précises. En résumé, le modèle Random Forest s'avère être le choix optimal pour notre problème de prédiction.