Projet Big Data Analytics :

Analyse de la Clientèle d'un Concessionnaire Automobile pour la Recommandation de Modèles de Véhicules

Data Mining, Machine Learning et Deep Learning

2023/2024

M2 MIAGE MBDS de l'l'Université Côte d'Azur(UCA)

Réalisépar :

BERRIRI Yassine
BOUCHEFFA Badis
KRIMI Ibrahim
LARBI Marwane
ZOUBID Dounia

Encadré par :

PASQUIER Nicolas

Data Analysis for Automobile Dealership

2023-12-31

Introduction : Ce projet d'analyse de données au sein d'une concession automobile vise à prédire la catégorie de véhicules la plus adaptée aux clients. La première étape consiste à identifier les catégories en regroupant les clients ayant des caractéristiques similaires. Les résultats sont associés à des catégories de véhicules, intégrant une nouvelle variable "Catégorie". Ensuite, un modèle de prédiction est construit en utilisant divers classifieurs et en évaluant leur performance sur un ensemble de test. Le modèle sélectionné est ensuite appliqué aux données marketing pour recommander des catégories de véhicules aux nouveaux clients, améliorant ainsi la pertinence des suggestions dans le processus de vente. Ce projet combine clustering, classification et prédiction pour optimiser les recommandations de véhicules et mieux répondre aux besoins des clients.

Étude et exploration de données:

Avant d'aborder l'exploration et l'analyse des données, nous souhaitons d'abord vous fournir des détails sur la manière dont nous avons récupéré ces données. Comme vous le savez, notre architecture repose sur l'utilisation d'un data lake dans le Hive, où les données sont initialement stockées dans des tables externes. En traitant ces tables, nous avons créé des tables internes au sein de ce data lake. Afin de récupérer et traiter ces données, une connexion au data lake est nécessaire. Nous allons maintenant vous expliquer comment cette connexion a été établie.

Démarrons en établissant une connexion avec une base de données Hive distante en utilisant le pilote JDBC pour Hive. Procédons ensuite à l'installation et au chargement des packages requis :

- **dplyr** : *Manipulation et analyse des données avec des fonctions pour sélectionner, filtrer, grouper et réorganiser.*
- **caret** : *Outils pour la création de modèles prédictifs incluant la préparation des données, la sélection des caractéristiques et le réglage des modèles.*
- **rpart** : *Analyse de régression et classification via des arbres de décision pour des modèles simples et interprétables.*
- **C50** : *Construction de modèles de classification avec les algorithmes C4.5 et C5.0, adaptés aux grandes bases de données.*
- **tree** : *Classification et régression avec des arbres de décision, offrant une approche graphique pour les modèles prédictifs.*
- **e1071** : *Fonctions pour la statistique et l'apprentissage machine, y compris les SVM, distributions de probabilité, et plus.*

- **RJDBC** : *Connexion aux bases de données via Java Database Connectivity, pour une interface entre R et diverses bases de données.*
- **rJava** : *Interface entre R et Java, permettant l'exécution de code Java et l'utilisation de classes Java dans R.*
- **DBI** : *Abstraction pour l'accès aux bases de données avec une interface commune pour divers systèmes de gestion de base de données.*
- **randomForest** : *Méthode des forêts aléatoires pour la classification et la régression, offrant un outil de modélisation prédictive puissant.*

```
knitr::opts_chunk$set(echo = TRUE)
library(dplyr)

## Warning: package 'dplyr' was built under R version 4.3.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(caret)

## Warning: package 'caret' was built under R version 4.3.2

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.3.2

## Loading required package: lattice

library(rpart)

## Warning: package 'rpart' was built under R version 4.3.2

library(C50)

## Warning: package 'C50' was built under R version 4.3.2

library(tree)

## Warning: package 'tree' was built under R version 4.3.2

library(e1071)

## Warning: package 'e1071' was built under R version 4.3.2
```

```
library(RJDBC)

## Warning: package 'RJDBC' was built under R version 4.3.2

## Loading required package: DBI

## Warning: package 'DBI' was built under R version 4.3.2

## Loading required package: rJava

## Warning: package 'rJava' was built under R version 4.3.2

library(rJava)
library(DBI)
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.3.2

## randomForest 4.7-1.1

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

on initialise d'abord un pilote JDBC pour se connecter à une base de données Hive, en spécifiant le chemin vers le fichier JAR du pilote Hive. Ensuite, avec la fonction dbConnect, on établit une connexion à la base de données Hive située sur localhost au port 10000, en utilisant les identifiants vagrant. Cette méthode permet une interaction fluide avec Hive depuis R, facilitant les opérations de gestion des données.

```
drv = JDBC("org.apache.hive.jdbc.HiveDriver" ,
"C:\\Users\\marwa\\Documents\\WorkSpace\\NoSql\\vagrant-
projects\\OracleDatabase\\21.3.0\\projetTpa\\normalisation\\HiveJdbcDriver\\h
ive-jdbc-4.0.0-alpha-1-standalone.jar")
conn = dbConnect(drv
 , "jdbc:hive2://localhost:10000/default" , "vagrant","vagrant")
```

nous avons utilisé la fonction dbGetQuery pour exécuter la commande "show tables" sur notre connexion Hive. Cela nous a permis de récupérer un dataframe nommé show_tables, contenant la liste des tables disponibles dans la base de données. Nous avons ensuite affiché ce dataframe, révélant huit tables distinctes, chacune étant listée sous la colonne tab_name. Cette approche nous a facilité la visualisation et l'accès aux tables dans Hive, directement depuis notre environnement de travail en R.

```
show_tables <- dbGetQuery(conn, "show tables ")

print(show_tables)
```

```
##                               tab_name
## 1                 catalogue_hdfs_ext
## 2                clients12_mongo_ext
## 3        clients12_mongo_ext_updated
## 4                  clients19_ons_ext
## 5                       co2_hdfs_ext
## 6         immatriculations_mongo_ext
## 7 immatriculations_mongo_ext_updated
## 8                  marketing_ons_ext
```

Data Loading

```
clients <- dbGetQuery.csv("select * from clients12_mongo_ext_updated ")
```

```
immatriculations <- dbGetQuery.csv("select * from
immatriculations_mongo_ext_updated ")
```

Fusionner les données sur le numéro d'immatriculation

nous avons effectué une fusion de deux ensembles de données, clients et immatriculations, en utilisant la fonction merge. La clé de fusion était le champ immatriculation, commun aux deux dataframes. Cette opération a permis de combiner les lignes des deux tables là où les valeurs du champ immatriculation coïncident, résultant en un nouveau dataframe donnees_fusionnees

```
donnees_fusionnees <- merge(clients, immatriculations, by =
"immatriculation")
```

après avoir fusionné les ensembles de données clients et immatriculations, nous avons utilisé la fonction head sur le dataframe résultant donnees_fusionnees.

```
head (donnees_fusionnees)
```

```
##   immatriculation age sexe taux situationFamiliale nbEnfantsAcharge
## 1         0 AS 74  27    M  413        Celibataire                0
## 2         0 FP 65  33    M  421            Seul(e)                3
## 3         0 MD 67  44    M 1388          En Couple                0
## 4         0 ME 78  24    F 1359           Marie(e)                0
## 5         0 WV 66  77    M 1256          En Couple                0
## 6         0 YQ 91  24    F  218        Celibataire                0
##   X2eme.voiture  marque      nom puissance    longueur nbPlaces nbPortes
## 1             0     BMW     120i       150     moyenne        5        5
## 2             0   VOLVO   S80 T6       272 tres longue        5        5
## 3             0     BMW       M5       507 tres longue        5        5
```

```
## 4             1    AUDI   A2 1.4        75      courte        5        5
## 5             0    SAAB 9.3 1.8T       150      longue        5        5
## 6             0 PEUGEOT 1007 1.4        75      courte        5        5
##   couleur occasion  prix
## 1    bleu        1 25060
## 2   rouge        0 50500
## 3   blanc        0 94800
## 4    gris        0 18310
## 5   rouge        0 38600
## 6    noir        1  9625
```

Sélectionner les colonnes pertinentes pour le clustering

Dans notre processus d'analyse, nous avons spécifiquement préparé les données pour le clustering. Pour cela, nous avons sélectionné un ensemble de colonnes pertinentes à partir du dataframe donnees_fusionnees en utilisant la fonction select. Les colonnes choisies incluent des variables personnelles et des caractéristiques de véhicules telles que age, sexe, taux, situationFamiliale, nbEnfantsAcharge, X2eme.voiture, marque, puissance, longueur, nbPlaces, nbPortes, couleur, occasion, et prix. Ce nouveau dataframe, donnees_clustering, contient désormais les informations essentielles pour procéder à un clustering efficace, permettant de dégager des groupes distincts et pertinents au sein de nos données.

```
donnees_clustering <- select(donnees_fusionnees, age, sexe, taux,
situationFamiliale, nbEnfantsAcharge, `X2eme.voiture`, marque, puissance,
longueur, nbPlaces, nbPortes, couleur, occasion, prix)

head(donnees_clustering)
```

```
##   age sexe taux situationFamiliale nbEnfantsAcharge X2eme.voiture  marque
## 1  27    M  413        Celibataire                0             0     BMW
## 2  33    M  421            Seul(e)                3             0   VOLVO
## 3  44    M 1388          En Couple                0             0     BMW
## 4  24    F 1359           Marie(e)                0             1    AUDI
## 5  77    M 1256          En Couple                0             0    SAAB
## 6  24    F  218        Celibataire                0             0 PEUGEOT
##   puissance    longueur nbPlaces nbPortes couleur occasion  prix
## 1       150     moyenne        5        5    bleu        1 25060
## 2       272 tres longue        5        5   rouge        0 50500
## 3       507 tres longue        5        5   blanc        0 94800
## 4        75      courte        5        5    gris        0 18310
## 5       150      longue        5        5   rouge        0 38600
## 6        75      courte        5        5    noir        1  9625
```

Traitement des valeurs manquantes

nous avons abordé le traitement des valeurs manquantes. Pour ce faire, nous avons d'abord défini une fonction personnalisée getmode en R. Cette fonction calcule le mode, c'est-à-dire la valeur la plus fréquemment observée dans un vecteur v. Elle fonctionne en identifiant les

valeurs uniques dans v avec unique(v), puis en utilisant tabulate(match(v, uniqv)) pour compter le nombre de fois que chaque valeur unique apparaît. Finalement, getmode retourne la valeur qui apparaît le plus souvent. Cette fonction est particulièrement utile pour remplacer les valeurs manquantes dans les données catégorielles par le mode, assurant ainsi une approche statistiquement raisonnable pour gérer les lacunes dans nos données.

```
# Fonction pour calculer le mode
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}
```

Normaliser les données

nous avons abordé la normalisation des données pour le clustering en traitant les valeurs manquantes dans donnees_clustering avec dplyr. Pour les colonnes numériques, les valeurs manquantes sont remplacées par la médiane de chaque colonne. Pour les colonnes catégorielles, les valeurs manquantes sont remplacées par le mode, calculé par notre fonction personnalisée getmode. Cette méthode assure une cohérence dans nos données, les rendant ainsi plus robustes et adaptées à des analyses de clustering efficaces.

```
donnees_clustering <- donnees_clustering %>%
  mutate_if(is.numeric, ~ifelse(is.na(.), median(., na.rm = TRUE), .)) %>%
  mutate_if(is.factor, ~ifelse(is.na(.), as.character(getmode(.)), .))
```

Trouver le nombre optimal de clusters

Dans notre analyse de clustering, nous avons commencé par transformer certaines variables catégorielles en variables numériques dans donnees_clustering. Cette transformation est essentielle pour permettre leur utilisation dans des algorithmes de clustering qui nécessitent des données numériques. Les colonnes sexe, situationFamiliale, marque, longueur, et couleur ont été converties en facteurs, puis en nombres

```
donnees_clustering$sexe <- as.numeric(factor(donnees_clustering$sexe))
donnees_clustering$situationFamiliale <-
as.numeric(factor(donnees_clustering$situationFamiliale))
donnees_clustering$marque <- as.numeric(factor(donnees_clustering$marque))
donnees_clustering$longueur <-
as.numeric(factor(donnees_clustering$longueur))
donnees_clustering$couleur <- as.numeric(factor(donnees_clustering$couleur))
```

Dans notre processus de clustering, nous avons d'abord éliminé les colonnes sans variation de `donnees_clustering` en ne conservant que celles avec un écart type non nul. Ensuite, nous avons normalisé les colonnes numériques, une étape essentielle pour garantir que chaque variable contribue équitablement au processus de clustering. Finalement, toutes les valeurs anormales (NA, NaN, infinies) dans le dataframe normalisé ont été remplacées par

zéro, assurant ainsi que nos données sont propres et prêtes pour une analyse de clustering précise et efficace.

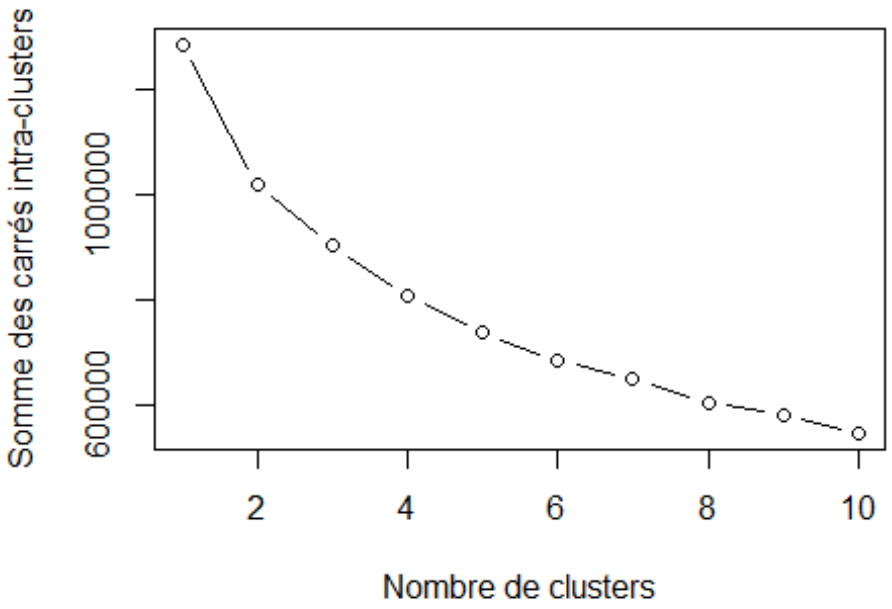
```
donnees_clustering <- donnees_clustering %>% select_if(function(x) sd(x,
na.rm = TRUE) != 0)
donnees_clustering_normalisees <- scale(donnees_clustering %>%
select_if(is.numeric))
donnees_clustering_normalisees[is.na(donnees_clustering_normalisees) |
is.nan(donnees_clustering_normalisees) |
is.infinite(donnees_clustering_normalisees)] <- 0
```

Dans notre analyse visant à déterminer le nombre optimal de clusters, nous avons utilisé la méthode des k-moyennes (`kmeans`). Pour chaque nombre de clusters, allant de 1 à 10, nous avons calculé la somme des carrés des écarts à l'intérieur des clusters (`tot.withinss`). Ceci a été réalisé en utilisant `sapply` pour appliquer la fonction `kmeans` à notre dataframe normalisé `donnees_clustering_normalisees`, avec différents nombres de centres (clusters). Le paramètre `nstart = 10` indique que l'algorithme de k-moyennes sera exécuté 10 fois avec des points de départ aléatoires pour chaque nombre de clusters, afin d'optimiser la solution. Cette approche nous permet de tracer ensuite une courbe pour identifier le point où l'ajout de nouveaux clusters n'amène plus d'amélioration significative, connu sous le nom de méthode du coude (Elbow Method), pour choisir le nombre optimal de clusters.

```
wss <- sapply(1:10, function(k){kmeans(donnees_clustering_normalisees,
centers = k, nstart = 10)$tot.withinss})
```

Pour visualiser comment la somme des carrés des écarts à l'intérieur des clusters (tot.withinss) varie avec le nombre de clusters, nous avons créé un graphique en points reliés (`type = "b"`) à l'aide de la fonction `plot`. L'axe des abscisses (`xlab`) représente le nombre de clusters, variant de 1 à 10, tandis que l'axe des ordonnées (`ylab`) indique la somme des carrés intra-clusters. Ce graphique est essentiel pour l'application de la méthode du coude, permettant de déterminer visuellement le nombre de clusters optimal pour notre analyse. L'idée est de trouver un équilibre, où augmenter le nombre de clusters ne réduit pas significativement la somme des carrés intra-clusters, indiquant ainsi le nombre approprié de clusters à utiliser dans notre modèle de clustering.

```
plot(1:10, wss, type = "b", xlab = "Nombre de clusters", ylab = "Somme des
carrés intra-clusters")
```



Afin de garantir la reproductibilité de notre analyse de clustering, nous avons d'abord fixé une graine aléatoire en utilisant `set.seed(123)`. Cela assure que les résultats du clustering seront les mêmes à chaque exécution du code. Ensuite, nous avons appliqué l'algorithme de k-moyennes (`kmeans`) sur notre dataframe normalisé `donnees_clustering_normalisees`, en choisissant de former 5 clusters (`centers = 5`). Le paramètre `nstart = 10` indique que l'algorithme sera exécuté 10 fois avec des points de départ différents, pour assurer une optimisation robuste. Cette étape est cruciale pour identifier des groupes distincts dans nos données, permettant ainsi une analyse approfondie des caractéristiques et tendances au sein de chaque cluster.

```
set.seed(123) # Pour la reproductibilité
km <- kmeans(donnees_clustering_normalisees, centers = 5, nstart = 10)
```

Pour ajouter une dimension interprétative à nos résultats de clustering, nous avons défini la fonction `associerCategorie`. Cette fonction attribue une catégorie spécifique à chaque cluster identifié par l'algorithme k-moyennes. Les clusters sont catégorisés comme "Familiale", "Sportive", "Citadine", "Luxueuse", ou "Autre" selon leur numéro. Ensuite, nous avons utilisé `sapply` pour appliquer cette fonction à chaque élément dans `km$cluster`, qui contient les numéros de cluster assignés à chaque observation dans `donnees_clustering_normalisees`. Le résultat, un vecteur de catégories, est ensuite ajouté au dataframe `donnees_fusionnees` sous la nouvelle colonne `Categorie`. Cette étape enrichit nos données fusionnées avec des informations catégorielles significatives, offrant une perspective claire sur la segmentation réalisée par le clustering.

```
associerCategorie <- function(cluster) {
  if (cluster == 1) {
```

```
    return("Familiale")
  } else if (cluster == 2) {
    return("Sportive")
  } else if (cluster == 3) {
    return("Citadine")
  } else if (cluster == 4) {
    return("Luxueuse")
  } else {
    return("Autre")
  }}

  donnees_fusionnees$Categorie <- sapply(km$cluster, associerCategorie)

head(donnees_fusionnees)
```

```
##   immatriculation age sexe taux situationFamiliale nbEnfantsAcharge
## 1         0 AS 74  27    M  413        Celibataire                0
## 2         0 FP 65  33    M  421            Seul(e)                3
## 3         0 MD 67  44    M 1388          En Couple                0
## 4         0 ME 78  24    F 1359           Marie(e)                0
## 5         0 WV 66  77    M 1256          En Couple                0
## 6         0 YQ 91  24    F  218        Celibataire                0
##   X2eme.voiture  marque      nom puissance    longueur nbPlaces nbPortes
## 1             0     BMW     120i       150     moyenne        5        5
## 2             0   VOLVO   S80 T6       272 tres longue        5        5
## 3             0     BMW       M5       507 tres longue        5        5
## 4             1    AUDI   A2 1.4        75      courte        5        5
## 5             0    SAAB 9.3 1.8T       150      longue        5        5
## 6             0 PEUGEOT 1007 1.4        75      courte        5        5
##   couleur occasion  prix Categorie
## 1    bleu        1 25060     Autre
## 2   rouge        0 50500  Citadine
## 3   blanc        0 94800  Sportive
## 4    gris        0 18310 Familiale
## 5   rouge        0 38600 Familiale
## 6    noir        1  9625     Autre
```

On utilise la fonction match() pour trouver les correspondances entre les immatriculations dans le dataframe clients et celles dans donnees_fusionnees. Ensuite, on extrait les catégories correspondantes depuis donnees_fusionnees et on les assigne aux clients dans le dataframe clients.

```
clients$Categorie <-
donnees_fusionnees$Categorie[match(clients$immatriculation,
donnees_fusionnees$immatriculation)]
```

L'opération réalisée ici est la suppression de la colonne immatriculation du dataframe.

```
clients <- select(clients, -immatriculation)

head(clients)
```

```
##   age sexe taux situationFamiliale nbEnfantsAcharge X2eme.voiture
Categorie
## 1  58    M  921          En Couple                4             0
Sportive
## 2  57    M  462        Celibataire                0             0
Autre
## 3  58    F  525          En Couple                3             1
Citadine
## 4  18    M  728        Celibataire                0             0
Familiale
## 5  33    M 1113        Celibataire                0             0
Familiale
## 6  84    M 1256          En Couple                0             0
Familiale
```

on convertit plusieurs colonnes du dataframe clients en facteurs pour faciliter leur utilisation dans des analyses statistiques. La conversion en facteurs est réalisée pour les colonnes Categorie, sexe, situationFamiliale, X2eme.voiture, et nbEnfantsAcharge

```
clients$Categorie <- as.factor(clients$Categorie)
clients$sexe <- as.factor(clients$sexe)
clients$situationFamiliale <- as.factor(clients$situationFamiliale)
clients$X2eme.voiture <- as.factor(clients$X2eme.voiture)
clients$nbEnfantsAcharge <- as.factor(clients$nbEnfantsAcharge)
```

Nous avons commencé par fixer une graine aléatoire avec set.seed(123) pour assurer la reproductibilité des résultats. Ensuite, nous avons utilisé la fonction createDataPartition() pour diviser le dataframe clients en deux sous-ensembles. Cette fonction sélectionne aléatoirement 80% des données (spécifié par p = 0.8) pour le trainingSet en se basant sur la distribution de la variable Categorie. Le reste des données constitue le testingSet. Cette méthode permet d'assurer que les deux ensembles sont représentatifs de la distribution globale de la variable Categorie dans le dataset origina

```
set.seed(123)
partition <- createDataPartition(clients$Categorie, p = 0.8, list = FALSE)

trainingSet <- clients[partition, ]
testingSet <- clients[-partition, ]

head(trainingSet)
```

```
##   age sexe taux situationFamiliale nbEnfantsAcharge X2eme.voiture
Categorie
## 3  58    F  525          En Couple                3             1
Citadine
## 5  33    M 1113        Celibataire                0             0
Familiale
## 6  84    M 1256          En Couple                0             0
Familiale
## 7  19    M  515          En Couple                4             1
```

```
Citadine
## 8  65    M  495          En Couple                3             0
Citadine
## 9  18    M  493          En Couple                1             1
Luxueuse
```

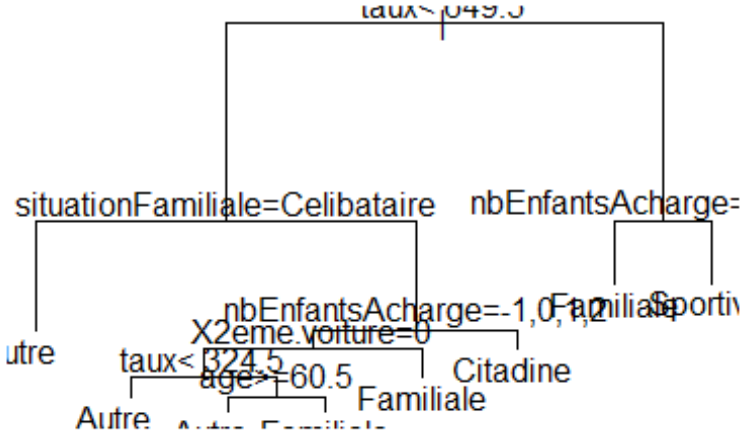
```
head(testingSet)
```

```
##    age sexe taux situationFamiliale nbEnfantsAcharge X2eme.voiture
Categorie
## 1   58    M  921          En Couple                4             0
Sportive
## 2   57    M  462        Celibataire                0             0
Autre
## 4   18    M  728        Celibataire                0             0
Familiale
## 12  81    F  447          En Couple                1             1
Luxueuse
## 22  55    M  165          En Couple                4             0
Citadine
## 31  42    M  198          En Couple                2             0
Autre
```

Modèle avec rpart

Nous avons construit un modèle de classification en utilisant l'algorithme rpart, une méthode réputée pour le partitionnement récursif, souvent employée dans la construction d'arbres de décision. En exécutant fitRpart <- rpart(Categorie ~ ., data = trainingSet), nous avons créé un arbre de décision pour prédire la variable Categorie en utilisant toutes les autres variables disponibles dans trainingSet. Cette approche permet de comprendre comment les différentes variables contribuent à la prédiction de la catégorie des clients.

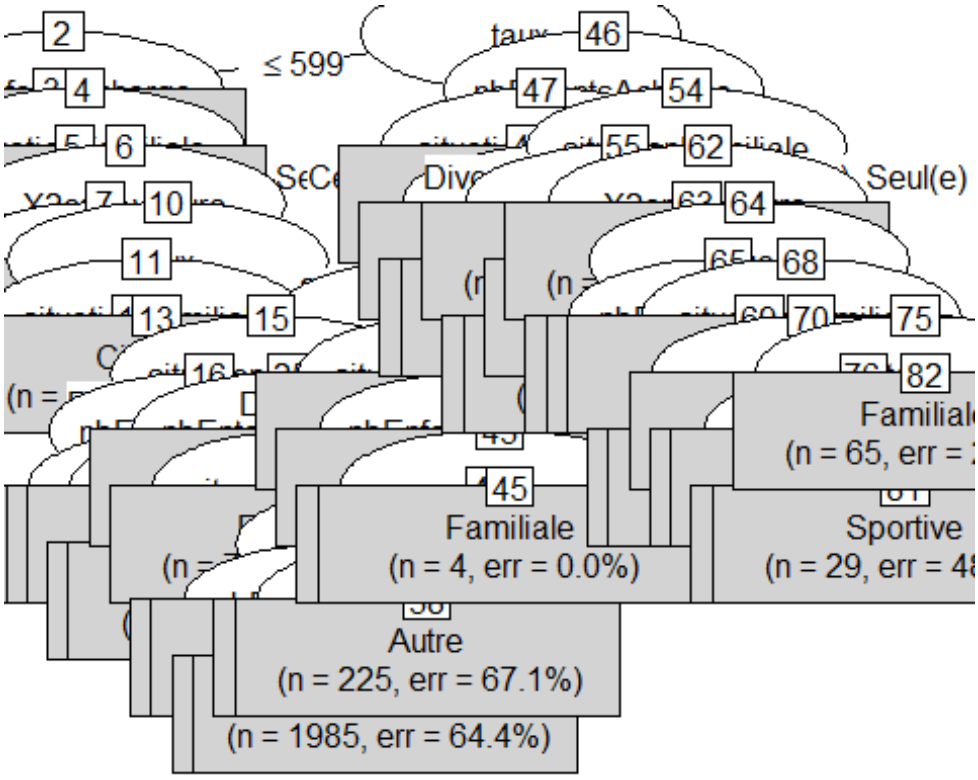
```
fitRpart <- rpart(Categorie ~ ., trainingSet)

plot(fitRpart)
text(fitRpart, pretty = 0)
```



Modèle avec C50

Nous avons également mis en place un modèle de classification en utilisant l'algorithme C5.0, une évolution de l'algorithme C4.5 réputé pour la construction d'arbres de décision et de règles de classification. Avec le code fitC50 <- C5.0(Categorie ~ ., data = trainingSet), nous avons formé un modèle C5.0 pour prédire la variable Categorie à partir de toutes les autres variables dans trainingSet.

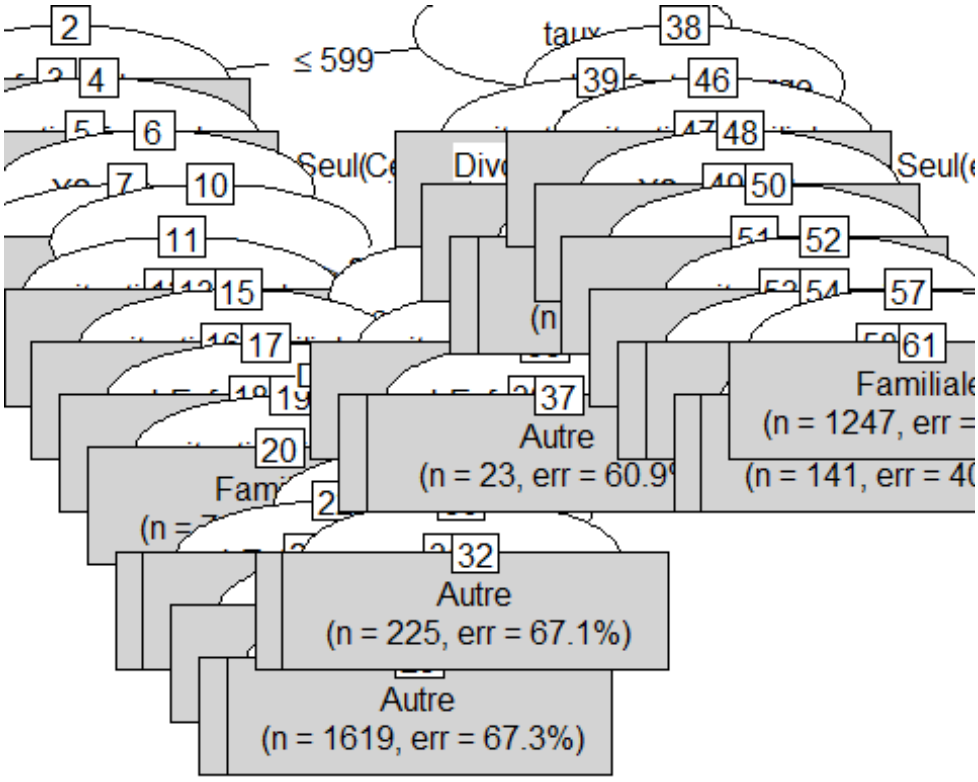
```
fitC50 <- C5.0(Categorie ~ ., trainingSet)

plot(fitC50, type="simple")
```



Modèle avec C50 (minCases)

Dans cette partie de notre rapport, nous avons amélioré notre modèle de classification utilisant l'algorithme C5.0 en ajustant certains paramètres pour affiner les résultats. Nous avons créé un nouveau modèle, fitC50minCases, en spécifiant des contrôles supplémentaires via C5.0Control.

```
fitC50minCases <- C5.0(Categorie ~ ., data = trainingSet,
                       control = C5.0Control(minCases = 10,
                                             noGlobalPruning = FALSE,
                                             winnow = TRUE)) # Use Information Gain
plot(fitC50minCases, type="simple")
```

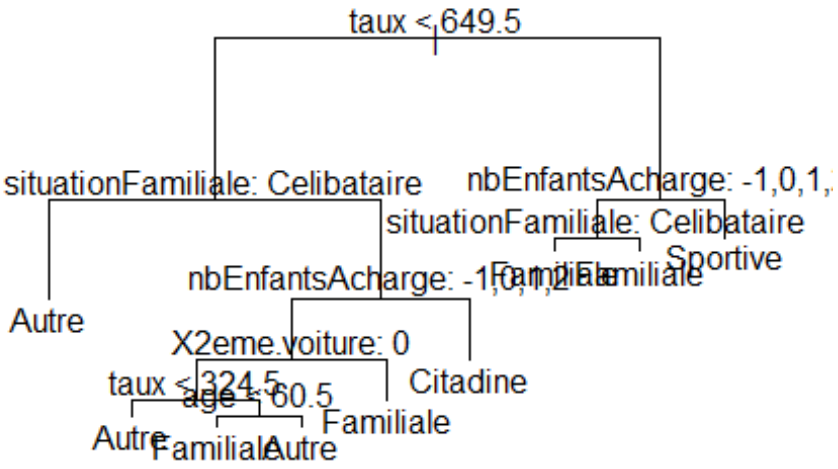


Modèle avec tree

nous avons également exploré un modèle de classification en utilisant l'algorithme tree. Cet algorithme est une autre méthode populaire pour construire des arbres de décision.

Nous avons d'abord formé le modèle avec le code fitTree <- tree(Categorie ~., data = trainingSet). Ici, tree() est utilisé pour construire un arbre de décision qui prédit la variable Categorie en utilisant toutes les autres variables disponibles dans trainingSet.

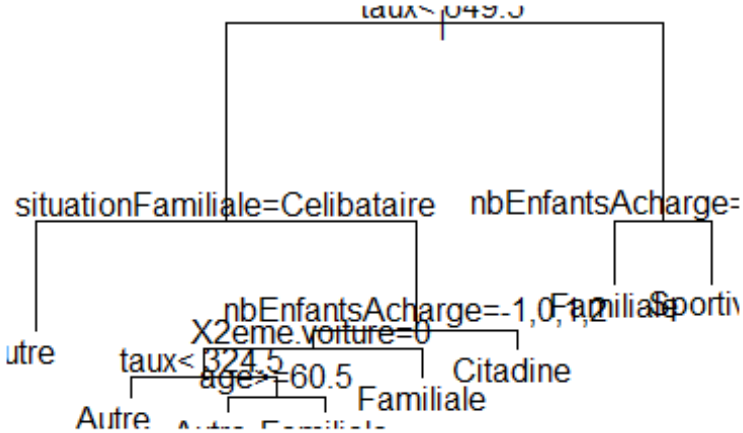
```
fitTree <- tree(Categorie ~., data = trainingSet)
plot(fitTree)
text(fitTree, pretty=0)
```



Modèle avec tree (gini)

nous avons approfondi notre analyse en utilisant l'algorithme rpart avec une spécification différente, en optant pour l'indice de Gini comme critère de division. Cela représente une variation de notre approche précédente avec l'algorithme rpart.

```
fitRpartgini <- rpart(Categorie ~ ., data = trainingSet,
                  method="class",
                  parms=list(split="gini"), # Using Gini Index for splits
                  control=rpart.control(minbucket=10))

plot(fitRpartgini)
text(fitRpartgini, pretty = 0)
```




```
models <- list(rpart = fitRpart, C50 = fitC50, tree = fitTree)
print(models)
```

```
## $rpart
## n= 78805
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 78805 56308 Familiale (0.28 0.22 0.29 0.065 0.15)
##    2) taux< 649.5 54649 32622 Autre (0.4 0.31 0.13 0.094 0.065)
##      4) situationFamiliale=Celibataire 16404  3192 Autre (0.81 6.1e-05
0.00012 0.19 0.00018) *
##      5) situationFamiliale=Divorcee,En Couple,Marie(e),Seul(e) 38245 21228
Citadine (0.23 0.44 0.18 0.051 0.093)
##       10) nbEnfantsAcharge=-1,0,1,2 26552 17740 Autre (0.33 0.28 0.26
0.073 0.052)
##         20) X2eme.voiture=0 21583 13212 Autre (0.39 0.35 0.2 0.0013 0.064)
##           40) taux< 324.5 6103  2540 Autre (0.58 0.41 0.0018 0.0016
0.00066) *
##           41) taux>=324.5 15480 10494 Citadine (0.31 0.32 0.28 0.0012
0.089)
##             82) age>=60.5 2983  1157 Autre (0.61 0.38 0.0027 0.002 0) *
##             83) age< 60.5 12497  8212 Familiale (0.24 0.31 0.34 0.001
0.11) *
##         21) X2eme.voiture=1 4969  2352 Familiale (0.089 0.0008 0.53 0.38
0) *
```

```
##       11) nbEnfantsAcharge=3,4 11693  2181 Citadine (0.00026 0.81 0.00051
0 0.19) *
##    3) taux>=649.5 24156  8588 Familiale (0.011 0.011 0.64 8.3e-05 0.33)
##      6) nbEnfantsAcharge=-1,0,1,2 18948  3562 Familiale (0.014 0.008 0.81
0 0.17) *
##      7) nbEnfantsAcharge=3,4 5208   290 Sportive (0 0.02 0.035 0.00038
0.94) *
##
## $C50
##
## Call:
## C5.0.formula(formula = Categorie ~ ., data = trainingSet)
##
## Classification Tree
## Number of samples: 78805
## Number of predictors: 6
##
## Tree size: 43
##
## Non-standard options: attempt to group attributes
##
##
## $tree
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 78805 237600 Familiale ( 2.830e-01 2.192e-01 2.855e-01 6.502e-02
1.473e-01 )
##    2) taux < 649.5 54649 152000 Autre ( 4.031e-01 3.114e-01 1.268e-01
9.373e-02 6.501e-02 )
##      4) situationFamiliale: Celibataire 16404  16270 Autre ( 8.054e-01
6.096e-05 1.219e-04 1.942e-01 1.829e-04 ) *
##      5) situationFamiliale: Divorcee,En Couple,Marie(e),Seul(e) 38245
105500 Citadine ( 2.305e-01 4.449e-01 1.811e-01 5.062e-02 9.282e-02 )
##       10) nbEnfantsAcharge: -1,0,1,2 26552  75310 Autre ( 3.319e-01
2.827e-01 2.607e-01 7.291e-02 5.190e-02 )
##         20) X2eme.voiture: 0 21583  53560 Autre ( 3.879e-01 3.475e-01
1.994e-01 1.344e-03 6.385e-02 )
##           40) taux < 324.5 6103   8620 Autre ( 5.838e-01 4.121e-01 1.802e-
03 1.639e-03 6.554e-04 ) *
##           41) taux > 324.5 15480  40460 Citadine ( 3.106e-01 3.221e-01
2.773e-01 1.227e-03 8.876e-02 )
##             82) age < 60.5 12497  33030 Familiale ( 2.386e-01 3.075e-01
3.429e-01 1.040e-03 1.099e-01 ) *
##             83) age > 60.5 2983   4155 Autre ( 6.121e-01 3.832e-01 2.682e-
03 2.011e-03 0.000e+00 ) *
##         21) X2eme.voiture: 1 4969   9202 Familiale ( 8.875e-02 8.050e-04
5.267e-01 3.838e-01 0.000e+00 ) *
##       11) nbEnfantsAcharge: 3,4 11693  11380 Citadine ( 2.566e-04 8.135e-
01 5.131e-04 0.000e+00 1.858e-01 ) *
```

```
##    3) taux > 649.5 24156  36200 Familiale ( 1.130e-02 1.068e-02 6.445e-01
8.280e-05 3.335e-01 )
##      6) nbEnfantsAcharge: -1,0,1,2 18948  21470 Familiale ( 1.441e-02
8.022e-03 8.120e-01 0.000e+00 1.656e-01 )
##       12) situationFamiliale: Celibataire 7120   2301 Familiale ( 3.722e-
02 0.000e+00 9.625e-01 0.000e+00 2.809e-04 ) *
##       13) situationFamiliale: Divorcee,En Couple,Marie(e),Seul(e) 11828
15340 Familiale ( 6.764e-04 1.285e-02 7.214e-01 0.000e+00 2.650e-01 ) *
##      7) nbEnfantsAcharge: 3,4 5208   2641 Sportive ( 0.000e+00 2.035e-02
3.495e-02 3.840e-04 9.443e-01 ) *
```

Making predictions on the test set using the trained models

nous effectuons des prédictions sur l'ensemble de test en utilisant trois modèles d'apprentissage automatique différents : "fitRpart", "fitC50" et "fitTree". Ces modèles ont été précédemment entraînés sur l'ensemble d'apprentissage et nous les utilisons maintenant pour prédire les catégories des observations dans l'ensemble de test.

```
predictions_rpart <- predict(fitRpart, newdata = testingSet, type = "class")
predictions_C50 <- predict(fitC50, newdata = testingSet)
predictions_tree <- predict(fitTree, newdata = testingSet, type = "class")
```

Evaluate the accuracy of the models

```
# Calculate the success rate (accuracy) of each model
success_rate_rpart <- sum(predictions_rpart == testingSet$Categorie) /
nrow(testingSet)
success_rate_C50 <- sum(predictions_C50 == testingSet$Categorie) /
nrow(testingSet)
success_rate_tree <- sum(predictions_tree == testingSet$Categorie) /
nrow(testingSet)
```

Les résultats de précision pour ces modèles sont les suivants :

Précision de "rpart" : environ 69,93%. Précision de "C50" : environ 72,04%. Précision de "Tree" : environ 69,93%. Ces résultats indiquent que le modèle "C50" a la meilleure précision parmi les trois modèles testés, avec une précision d'environ 72,04%.

```
# Convert to percentage
success_rate_rpart_percent <- success_rate_rpart * 100
success_rate_C50_percent <- success_rate_C50 * 100
success_rate_tree_percent <- success_rate_tree * 100
# Print out the accuracy for each model
print(paste("Success Rate of Rpart:", success_rate_rpart_percent, "%"))

## [1] "Success Rate of Rpart: 69.9309574576099 %"

print(paste("Success Rate of C50:", success_rate_C50_percent, "%"))

## [1] "Success Rate of C50: 72.0428469895421 %"
```

```r
print(paste("Success Rate of Tree:", success_rate_tree_percent, "%"))
```

```
## [1] "Success Rate of Tree: 69.9309574576099 %"
```

Making predictions on the test set using the trained models

fitRpart: C'est probablement un modèle d'arbre de décision que nous avons créé en utilisant la bibliothèque rpart en R. L'objet fitRpart représente le modèle entraîné. Avec la fonction predict, nous appliquons ce modèle à notre testingSet (l'ensemble de données de test) pour obtenir des prédictions. L'argument type = "class" indique que nous voulons des prédictions de classe (plutôt que des probabilités, par exemple).

fitC50: Ici nous utilisons un modèle entraîné avec une implémentation de l'algorithme C5.0. La syntaxe ne spécifie pas un argument type, ce qui signifie que nous récupérons le type par défaut de prédictions produites par ce modèle lorsque nous l'appliquons à testingSet.

fitTree: Ce modèle est probablement un modèle d'arbre simple créé avec une autre bibliothèque ou méthode, et nous faisons à nouveau des prédictions en spécifiant que nous voulons le type = "class". Cela pourrait être un modèle générique d'arbre de décision dans R.

```r
test_tree_rpart <- predict(fitRpart, newdata = testingSet, type = "class")
test_tree_C5 <- predict(fitC50, newdata = testingSet)
test_tree_tree <- predict(fitTree, newdata = testingSet, type = "class")
```

nous procédons à la création de matrices de confusion pour évaluer les performances de nos modèles d'arbres de décision. Chaque matrice de confusion met en correspondance les prédictions de chaque modèle avec les véritables valeurs issues de l'ensemble de test.

Chacune de ces matrices de confusion nous aidera à voir où nos modèles ont correctement prédit la catégorie (Categorie) et où ils ont fait des erreurs. Pour nous, ces matrices sont des outils diagnostics cruciaux qui informent sur la précision de nos modèles et sur les éventuels ajustements qui pourraient être nécessaires pour améliorer leur performance.

```r
# Creating confusion matrices
mc_tree_rpart <- table(testingSet$Categorie, test_tree_rpart)
mc_tree_C5 <- table(testingSet$Categorie, test_tree_C5)
mc_tree_tree <- table(testingSet$Categorie, test_tree_tree)

print("Confusion Matrix for Rpart Model:")
```

```
## [1] "Confusion Matrix for Rpart Model:"
```

```r
print(mc_tree_rpart)
```

```
##            test_tree_rpart
##             Autre Citadine Familiale Luxueuse Sportive
##   Autre      4654        2       918        0        0
##   Citadine    943     2362       975        0       39
##   Familiale     7        1      5569        0       47
```

```
##   Luxueuse    810        0       469        0        1
##   Sportive      1      550      1160        0     1190
```

En analysant cette matrice, nous pouvons voir comment se comporte le modèle concernant les prédictions des différentes catégories de véhicules. Voici comment nous pourrions interpréter les résultats :

La colonne Autre montre que 4654 véhicules ont été correctement prédits comme Autre, mais il y a 943 véhicules qui étaient en réalité de la catégorie Citadine et qui ont été incorrectement classifiés comme Autre. De même, la colonne Citadine indique que 2362 véhicules Citadine ont été correctement identifiés, cependant, 550 véhicules de la catégorie Sportive ont été faussement identifiés comme Citadine. Pour la catégorie Familiale, notre modèle a très bien performé en identifiant correctement 5569 véhicules, avec relativement peu d'erreurs par rapport aux autres catégories. Concernant les Luxueuses, aucun n'a été correctement identifié, indiquant une faiblesse potentielle dans notre modèle ou la possibilité que les données d'entraînement pour cette catégorie ne soient pas suffisamment représentatives ou volumineuses. Enfin, pour les véhicules Sportive, 1190 ont été correctement prédits mais de nombreux autres ont été classés à tort comme Familiale ou Citadine.

```
print("Confusion Matrix for C50 Model:")
```

```
## [1] "Confusion Matrix for C50 Model:"
```

```
print(mc_tree_C5)
```

```
##            test_tree_C5
##             Autre Citadine Familiale Luxueuse Sportive
##   Autre      4992      205       300       77        0
##   Citadine    982     3129       206        0        2
##   Familiale   385      415      4736       76       12
##   Luxueuse    805        6       285      183        1
##   Sportive    111      722       917        0     1151
```

Dans la catégorie Autre, notre modèle C50 a correctement identifié un grand nombre de véhicules (4992 véhicules). Cependant, nous observons qu'il y a eu quelques confusions substantielles, en particulier 205 véhicules qui ont été classés à tort comme Citadine, et 300 comme Familiale. En ce qui concerne la catégorie Citadine, il semble y avoir une amélioration notable par rapport au modèle Rpart précédent; 3129 Citadines ont été correctement classifiées, même si près de 982 ont été incorrectement prédits comme étant de la catégorie Autre. Pour les véhicules Familiale, la majorité ont été identifiés avec succès (4736), mais des erreurs persistantes nous indiquent qu'il y a encore des optimisations à envisager, notamment avec les 385 Familiales prédites comme Autre et les 415 comme Citadine. La catégorie Luxueuse montre une amélioration par rapport à Rpart, avec 183 prédictions correctes, suggérant une meilleure capacité de discrimination de ce modèle. Néanmoins, les erreurs avec d'autres catégories, notamment 805 Luxueuses prédites comme Autre, nous indiquent où nous pourrions focaliser nos actions d'amélioration. Enfin, pour la catégorie Sportive, la situation demeure complexe avec une répartition des erreurs

sur plusieurs autres catégories, même si 1151 véhicules Sportive ont été identifiés correctement.

```
print("Confusion Matrix for Tree Model:")
```

```
## [1] "Confusion Matrix for Tree Model:"
```

```
print(mc_tree_tree)
```

```
##            test_tree_tree
##             Autre Citadine Familiale Luxueuse Sportive
##   Autre      4654        2       918        0        0
##   Citadine    943     2362       975        0       39
##   Familiale     7        1      5569        0       47
##   Luxueuse    810        0       469        0        1
##   Sportive      1      550      1160        0     1190
```

Pour la catégorie des voitures Autre, notre modèle a correctement identifié 4654 unités. Il semble néanmoins confondre assez fréquemment cette catégorie avec Familiale, avec 918 erreurs dans ce sens. En passant à la catégorie Citadine, nous pouvons affirmer que 2362 prédictions sont correctes. Cependant, il y a un nombre significatif de confusion, surtout avec la catégorie Familiale où 975 Citadines ont été incorrectement cataloguées. S'agissant des véhicules Familiale, le modèle Tree a affiché une excellente capacité de reconnaissance en identifiant correctement 5569 véhicules. Toutefois, un petit nombre de Familiale a été confondu avec la catégorie Sportive. Pour la catégorie Luxueuse, il semble que notre modèle ne parvienne pas à identifier correctement ces véhicules puisqu'aucune prédiction correcte n'est enregistrée. Les erreurs sont en grande partie avec les catégories Autre et Familiale. Enfin, pour les voitures Sportive, notre modèle a correctement catégorisé 1190 d'entre elles mais a tendance à les confondre avec les catégories Familiale et dans une mesure plus faible Citadine.

```
# Class-specific Accuracy
accuracy_Familiale <- mc_tree_rpart["Familiale", "Familiale"] /
sum(mc_tree_rpart["Familiale", ])
accuracy_Sportive <- mc_tree_rpart["Sportive", "Sportive"] /
sum(mc_tree_rpart["Sportive", ])
accuracy_Citadine <- mc_tree_rpart["Citadine", "Citadine"] /
sum(mc_tree_rpart["Citadine", ])
accuracy_Luxueuse <- mc_tree_rpart["Luxueuse", "Luxueuse"] /
sum(mc_tree_rpart["Luxueuse", ])
accuracy_Autre <- mc_tree_rpart["Autre", "Autre"] /
sum(mc_tree_rpart["Autre", ])
```

L'exactitude pour la catégorie Familiale est de 99.02%, ce qui est extrêmement élevé et suggère que le modèle est très précis pour cette classe.

La classe Sportive, cependant, présente une exactitude bien plus basse de 41.02%, montrant que le modèle a des difficultés à identifier correctement ces véhicules.

Pour les véhicules de classe Citadine, l'exactitude est de 54.69%, ce qui est meilleur que les véhicules Sportive mais illustre toujours un potentiel important d'amélioration.

Quant à la classe Autre, l'exactitude est de 83.49%, une valeur assez haute, indiquant une précision satisfaisante pour la classification de cette catégorie diverse.

```
# Printing the results
print(paste("Accuracy for Familiale:", accuracy_Familiale))

## [1] "Accuracy for Familiale: 0.990220483641536"

print(paste("Accuracy for Sportive:", accuracy_Sportive))

## [1] "Accuracy for Sportive: 0.410203378145467"

print(paste("Accuracy for Citadine:", accuracy_Citadine))

## [1] "Accuracy for Citadine: 0.546885853206761"

print(paste("Accuracy for Autre:", accuracy_Autre))

## [1] "Accuracy for Autre: 0.834947972730535"

# Class-specific Precision
precision_Familiale <- mc_tree_rpart["Familiale", "Familiale"] /
sum(mc_tree_rpart[, "Familiale"])
precision_Sportive <- mc_tree_rpart["Sportive", "Sportive"] /
sum(mc_tree_rpart[, "Sportive"])
precision_Citadine <- mc_tree_rpart["Citadine", "Citadine"] /
sum(mc_tree_rpart[, "Citadine"])
precision_Autre <- ifelse(sum(mc_tree_rpart[, "Autre"]) == 0, NaN,
mc_tree_rpart["Autre", "Autre"] / sum(mc_tree_rpart[, "Autre"]))
```

La précision pour la catégorie Familiale est de 61.26%. Cela signifie que du nombre total de prédictions pour Familiale, environ 61.26% étaient correctes.

La précision pour la catégorie Sportive est de 93.19%, ce qui est très élevé. Cela indique que la plupart des prédictions de la catégorie Sportive étaient correctes.

La classe Citadine a une précision de 81.03%, une bonne précision suggérant que la majorité des prédictions Citadine étaient également correctes.

Pour la classe Autre, la précision est de 72.55%. Malgré la diversité potentielle au sein de cette catégorie, la proportion des prédictions correctes est assez élevée.

```
print(paste("Precision for Familiale:", precision_Familiale))

## [1] "Precision for Familiale: 0.612583874161258"

print(paste("Precision for Sportive:", precision_Sportive))

## [1] "Precision for Sportive: 0.931871574001566"

print(paste("Precision for Citadine:", precision_Citadine))
```

```
## [1] "Precision for Citadine: 0.810291595197256"

print(paste("Precision for Autre:", precision_Autre))

## [1] "Precision for Autre: 0.725487139516758"
```

#Training the SVM Model entraîner un modèle de Machine à Vecteurs de Support (SVM pour Support Vector Machine) à l'aide de la fonction svm() du package {e1071}, qui est un package populaire pour les techniques de machine learning dans R. Le modèle est entraîné pour classer des instances en fonction de la variable cible Categorie, en utilisant toutes les autres variables disponibles comme prédictives (indiqué par ~ .)

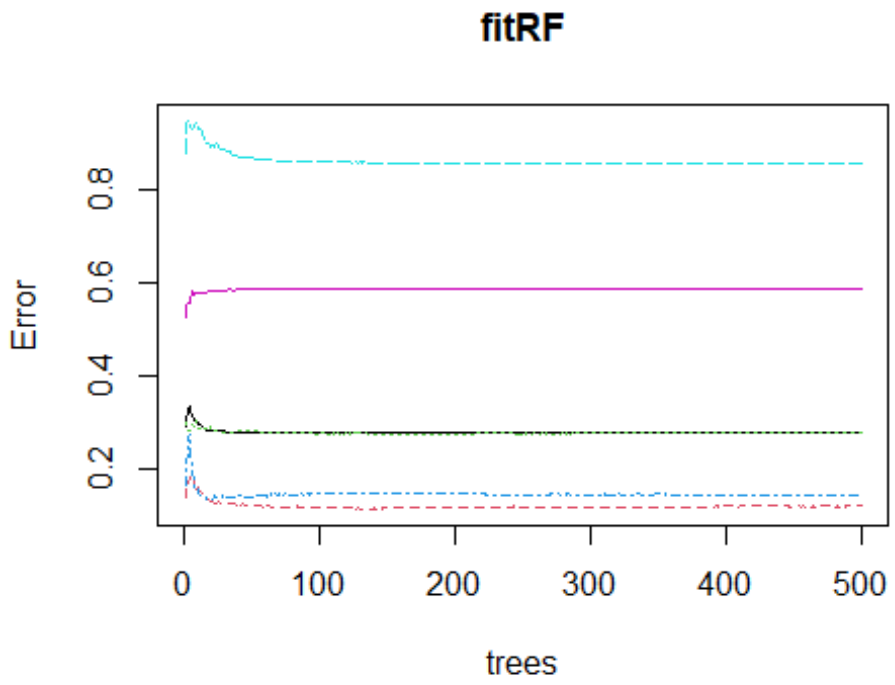
```
set.seed(123) # For reproducibility
fitSVM <- svm(Categorie ~ ., data = trainingSet, method = "C-classification",
kernel = "radial")

# Making predictions on the test set using the trained models
predictions_SVM <- predict(fitSVM, newdata = testingSet)

# Creating confusion matrices
mc_tree_SVM <- table(testingSet$Categorie, predictions_SVM)

# Evaluate the accuracy of the models
accuracy_SVM <- sum(predictions_SVM == testingSet$Categorie) /
nrow(testingSet)
```

Cette valeur indique que le modèle a correctement prédit 71.74% des étiquettes de catégorie pour les instances dans cet ensemble de données

```
# Print out the accuracy for each model
print(paste("Accuracy of SVM:", accuracy_SVM))

## [1] "Accuracy of SVM: 0.717382475378211"
```

#Training the Random Forest Model Le code ci-dessus initialise l'entraînement d'un modèle Random Forest en utilisant la fonction randomForest et en fixant la graine aléatoire à 123 pour garantir que les résultats sont reproductibles. Le modèle est formé pour prédire la variable Categorie et utilise toutes les autres variables du jeu de données trainingSet comme prédicteurs. Le nombre d'arbres (ntree) spécifié pour la forêt est de 500, ce qui devrait permettre au modèle de capturer les tendances complexes dans les données tout en évitant le surajustement.

```
set.seed(123) # For reproducibility
fitRF <- randomForest(Categorie ~ ., data = trainingSet, ntree = 500)

plot(fitRF)
```



Nous avons utilisé notre modèle Random Forest entraîné pour faire des prédictions sur notre ensemble de test. En créant une matrice de confusion (confusion matrix), nous avons pu visualiser la performance de notre modèle de manière plus détaillée. En calculant l'exactitude de notre modèle (accuracy), qui s'est révélée être de 72.12%, nous avons obtenu une mesure quantitative qui reflète la proportion des prédictions justes par rapport au total des prédictions faites. Cette métrique démontre une performance relativement bonne et confirme l'efficacité de notre modèle Random Forest sur les données testées.

```
# Making predictions on the test set using the trained models
predictions_RF <- predict(fitRF, newdata = testingSet)

# Creating confusion matrices
mc_tree_RF <- table(testingSet$Categorie, predictions_RF)

# Evaluate the accuracy of the models
accuracy_RF <- sum(predictions_RF == testingSet$Categorie) / nrow(testingSet)

# Print out the accuracy for each model
print(paste("Accuracy of Random Forest:", accuracy_RF))

## [1] "Accuracy of Random Forest: 0.72124073510001"
```

Adding the Models to Models List

```
models <- list(rpart = fitRpart, C50 = fitC50, tree = fitTree, SVM = fitSVM,
RandomForest = fitRF)

print(models)
```

```
## $rpart
## n= 78805
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 78805 56308 Familiale (0.28 0.22 0.29 0.065 0.15)
##    2) taux< 649.5 54649 32622 Autre (0.4 0.31 0.13 0.094 0.065)
##      4) situationFamiliale=Celibataire 16404  3192 Autre (0.81 6.1e-05
0.00012 0.19 0.00018) *
##      5) situationFamiliale=Divorcee,En Couple,Marie(e),Seul(e) 38245 21228
Citadine (0.23 0.44 0.18 0.051 0.093)
##       10) nbEnfantsAcharge=-1,0,1,2 26552 17740 Autre (0.33 0.28 0.26
0.073 0.052)
##         20) X2eme.voiture=0 21583 13212 Autre (0.39 0.35 0.2 0.0013 0.064)
##           40) taux< 324.5 6103  2540 Autre (0.58 0.41 0.0018 0.0016
0.00066) *
##           41) taux>=324.5 15480 10494 Citadine (0.31 0.32 0.28 0.0012
0.089)
##             82) age>=60.5 2983  1157 Autre (0.61 0.38 0.0027 0.002 0) *
##             83) age< 60.5 12497  8212 Familiale (0.24 0.31 0.34 0.001
0.11) *
##         21) X2eme.voiture=1 4969  2352 Familiale (0.089 0.0008 0.53 0.38
0) *
##       11) nbEnfantsAcharge=3,4 11693  2181 Citadine (0.00026 0.81 0.00051
0 0.19) *
##    3) taux>=649.5 24156  8588 Familiale (0.011 0.011 0.64 8.3e-05 0.33)
##      6) nbEnfantsAcharge=-1,0,1,2 18948  3562 Familiale (0.014 0.008 0.81
0 0.17) *
##      7) nbEnfantsAcharge=3,4 5208   290 Sportive (0 0.02 0.035 0.00038
0.94) *
##
## $C50
##
## Call:
## C5.0.formula(formula = Categorie ~ ., data = trainingSet)
##
## Classification Tree
## Number of samples: 78805
## Number of predictors: 6
##
## Tree size: 43
##
## Non-standard options: attempt to group attributes
##
##
## $tree
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
```

```
##  1) root 78805 237600 Familiale ( 2.830e-01 2.192e-01 2.855e-01 6.502e-02
1.473e-01 )
##    2) taux < 649.5 54649 152000 Autre ( 4.031e-01 3.114e-01 1.268e-01
9.373e-02 6.501e-02 )
##      4) situationFamiliale: Celibataire 16404  16270 Autre ( 8.054e-01
6.096e-05 1.219e-04 1.942e-01 1.829e-04 ) *
##      5) situationFamiliale: Divorcee,En Couple,Marie(e),Seul(e) 38245
105500 Citadine ( 2.305e-01 4.449e-01 1.811e-01 5.062e-02 9.282e-02 )
##       10) nbEnfantsAcharge: -1,0,1,2 26552  75310 Autre ( 3.319e-01
2.827e-01 2.607e-01 7.291e-02 5.190e-02 )
##         20) X2eme.voiture: 0 21583  53560 Autre ( 3.879e-01 3.475e-01
1.994e-01 1.344e-03 6.385e-02 )
##           40) taux < 324.5 6103   8620 Autre ( 5.838e-01 4.121e-01 1.802e-
03 1.639e-03 6.554e-04 ) *
##           41) taux > 324.5 15480  40460 Citadine ( 3.106e-01 3.221e-01
2.773e-01 1.227e-03 8.876e-02 )
##             82) age < 60.5 12497  33030 Familiale ( 2.386e-01 3.075e-01
3.429e-01 1.040e-03 1.099e-01 ) *
##             83) age > 60.5 2983   4155 Autre ( 6.121e-01 3.832e-01 2.682e-
03 2.011e-03 0.000e+00 ) *
##         21) X2eme.voiture: 1 4969   9202 Familiale ( 8.875e-02 8.050e-04
5.267e-01 3.838e-01 0.000e+00 ) *
##       11) nbEnfantsAcharge: 3,4 11693  11380 Citadine ( 2.566e-04 8.135e-
01 5.131e-04 0.000e+00 1.858e-01 ) *
##    3) taux > 649.5 24156  36200 Familiale ( 1.130e-02 1.068e-02 6.445e-01
8.280e-05 3.335e-01 )
##      6) nbEnfantsAcharge: -1,0,1,2 18948  21470 Familiale ( 1.441e-02
8.022e-03 8.120e-01 0.000e+00 1.656e-01 )
##       12) situationFamiliale: Celibataire 7120   2301 Familiale ( 3.722e-
02 0.000e+00 9.625e-01 0.000e+00 2.809e-04 ) *
##       13) situationFamiliale: Divorcee,En Couple,Marie(e),Seul(e) 11828
15340 Familiale ( 6.764e-04 1.285e-02 7.214e-01 0.000e+00 2.650e-01 ) *
##      7) nbEnfantsAcharge: 3,4 5208   2641 Sportive ( 0.000e+00 2.035e-02
3.495e-02 3.840e-04 9.443e-01 ) *
##
## $SVM
##
## Call:
## svm(formula = Categorie ~ ., data = trainingSet, method = "C-
classification",
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  43699
##
```

```
##
## $RandomForest
##
## Call:
##  randomForest(formula = Categorie ~ ., data = trainingSet, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 27.64%
## Confusion matrix:
##           Autre Citadine Familiale Luxueuse Sportive class.error
## Autre     19667      750      1540      343        0   0.1180717
## Citadine   3722    12524      1019        1       10   0.2750637
## Familiale  1172     1646     19302      354       23   0.1420189
## Luxueuse   3194       19      1182      728        1   0.8579235
## Sportive    376     2809      3617        0     4806   0.5859752
```

```
conf_matrix_rpart <- confusionMatrix(predictions_rpart, testingSet$Categorie)
print(conf_matrix_rpart)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Autre Citadine Familiale Luxueuse Sportive
##   Autre      4654      943         7      810        1
##   Citadine      2     2362         1        0      550
##   Familiale   918      975      5569      469     1160
##   Luxueuse      0        0         0        0        0
##   Sportive      0       39        47        1     1190
##
## Overall Statistics
##
##                Accuracy : 0.6993
##                  95% CI : (0.6929, 0.7057)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5904
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Autre Class: Citadine Class: Familiale
## Sensitivity                0.8349          0.5469           0.9902
## Specificity                0.8753          0.9640           0.7498
## Pos Pred Value             0.7255          0.8103           0.6126
## Neg Pred Value             0.9307          0.8834           0.9948
## Prevalence                 0.2830          0.2193           0.2855
```

```
## Detection Rate              0.2363          0.1199           0.2827
## Detection Prevalence        0.3257          0.1480           0.4615
## Balanced Accuracy           0.8551          0.7555           0.8700
##                       Class: Luxueuse Class: Sportive
## Sensitivity                   0.00000         0.41020
## Specificity                   1.00000         0.99482
## Pos Pred Value                    NaN         0.93187
## Neg Pred Value                0.93502         0.90712
## Prevalence                    0.06498         0.14727
## Detection Rate                0.00000         0.06041
## Detection Prevalence          0.00000         0.06483
## Balanced Accuracy             0.50000         0.70251
```

une précision globale de 69.93%, avec des spécificités élevées pour chaque catégorie, indiquant une bonne performance dans l'identification des négatifs réels. La catégorie Familiale a la plus haute sensibilité (99.02%), signifiant que le modèle est efficace pour détecter cette classe. Pourtant, la précision n'est pas aussi forte pour la catégorie Luxueuse, car toutes les prédictions sont erronées pour cette classe.

```
conf_matrix_C50 <- confusionMatrix(as.factor(predictions_C50),
testingSet$Categorie)
print(conf_matrix_C50)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Autre Citadine Familiale Luxueuse Sportive
##   Autre      4992      982       385      805      111
##   Citadine    205     3129       415        6      722
##   Familiale   300      206      4736      285      917
##   Luxueuse     77        0        76      183        0
##   Sportive      0        2        12        1     1151
##
## Overall Statistics
##
##                Accuracy : 0.7204
##                  95% CI : (0.7141, 0.7267)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6234
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: Autre Class: Citadine Class: Familiale
## Sensitivity                0.8956          0.7245           0.8421
## Specificity                0.8384          0.9123           0.8786
## Pos Pred Value             0.6862          0.6989           0.7349
```

```
## Neg Pred Value              0.9532          0.9218           0.9330
## Prevalence                  0.2830          0.2193           0.2855
## Detection Rate              0.2534          0.1588           0.2404
## Detection Prevalence        0.3693          0.2273           0.3271
## Balanced Accuracy           0.8670          0.8184           0.8604
##                       Class: Luxueuse Class: Sportive
## Sensitivity                   0.14297         0.39676
## Specificity                   0.99169         0.99911
## Pos Pred Value                0.54464         0.98714
## Neg Pred Value                0.94334         0.90557
## Prevalence                    0.06498         0.14727
## Detection Rate                0.00929         0.05843
## Detection Prevalence          0.01706         0.05919
## Balanced Accuracy             0.56733         0.69793
```

La deuxième matrice améliore légèrement la précision globale à 72.04%, avec une meilleure sensibilité pour la catégorie Autre comparée à la première matrice. La classe Luxueuse montre une faible sensibilité, suggérant que le modèle continue de lutter pour identifier correctement cette catégorie.

```
conf_matrix_tree <- confusionMatrix(predictions_tree, testingSet$Categorie)
print(conf_matrix_tree)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Autre Citadine Familiale Luxueuse Sportive
##   Autre      4654      943         7      810        1
##   Citadine      2     2362         1        0      550
##   Familiale   918      975      5569      469     1160
##   Luxueuse      0        0         0        0        0
##   Sportive      0       39        47        1     1190
##
## Overall Statistics
##
##                Accuracy : 0.6993
##                  95% CI : (0.6929, 0.7057)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5904
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Autre Class: Citadine Class: Familiale
## Sensitivity                0.8349          0.5469           0.9902
## Specificity                0.8753          0.9640           0.7498
## Pos Pred Value             0.7255          0.8103           0.6126
```

```
## Neg Pred Value              0.9307          0.8834           0.9948
## Prevalence                  0.2830          0.2193           0.2855
## Detection Rate              0.2363          0.1199           0.2827
## Detection Prevalence        0.3257          0.1480           0.4615
## Balanced Accuracy           0.8551          0.7555           0.8700
##                      Class: Luxueuse Class: Sportive
## Sensitivity                  0.00000         0.41020
## Specificity                  1.00000         0.99482
## Pos Pred Value                   NaN         0.93187
## Neg Pred Value               0.93502         0.90712
## Prevalence                   0.06498         0.14727
## Detection Rate               0.00000         0.06041
## Detection Prevalence         0.00000         0.06483
## Balanced Accuracy            0.50000         0.70251
```

```
conf_matrix_SVM <- confusionMatrix(predictions_SVM, testingSet$Categorie)
print(conf_matrix_SVM)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Autre Citadine Familiale Luxueuse Sportive
##   Autre      4773      843       339      804      115
##   Citadine    375     3199       379        7      695
##   Familiale   349      258      4820      284      936
##   Luxueuse     77        0        77      184        0
##   Sportive      0       19         9        1     1155
##
## Overall Statistics
##
##                Accuracy : 0.7174
##                  95% CI : (0.711, 0.7237)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6197
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: Autre Class: Citadine Class: Familiale
## Sensitivity                0.8563          0.7407           0.8570
## Specificity                0.8512          0.9053           0.8702
## Pos Pred Value             0.6944          0.6872           0.7251
## Neg Pred Value             0.9375          0.9255           0.9384
## Prevalence                 0.2830          0.2193           0.2855
## Detection Rate             0.2423          0.1624           0.2447
## Detection Prevalence       0.3490          0.2363           0.3374
## Balanced Accuracy          0.8538          0.8230           0.8636
```

```
##                      Class: Luxueuse Class: Sportive
## Sensitivity                 0.143750         0.39814
## Specificity                 0.991639         0.99827
## Pos Pred Value              0.544379         0.97551
## Neg Pred Value              0.943388         0.90569
## Prevalence                  0.064981         0.14727
## Detection Rate              0.009341         0.05864
## Detection Prevalence        0.017159         0.06011
## Balanced Accuracy           0.567694         0.69821
```

Avec 71.74% de précision globale, cette matrice montre que la sensibilité pour la catégorie Autre est très élevée (85.63%), mais les catégories Sportive et Luxueuse présentent toujours des faiblesses en termes de sensibilité.

```
conf_matrix_RF <- confusionMatrix(predictions_RF, testingSet$Categorie)
print(conf_matrix_RF)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Autre Citadine Familiale Luxueuse Sportive
##   Autre      4922      921       316      806       86
##   Citadine    221     3147       415        9      717
##   Familiale   357      249      4808      284      948
##   Luxueuse     74        0        79      180        0
##   Sportive      0        2         6        1     1150
##
## Overall Statistics
##
##                Accuracy : 0.7212
##                  95% CI : (0.7149, 0.7275)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6245
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: Autre Class: Citadine Class: Familiale
## Sensitivity                0.8830          0.7286           0.8549
## Specificity                0.8493          0.9114           0.8694
## Pos Pred Value             0.6981          0.6979           0.7234
## Neg Pred Value             0.9484          0.9228           0.9375
## Prevalence                 0.2830          0.2193           0.2855
## Detection Rate             0.2499          0.1598           0.2441
## Detection Prevalence       0.3580          0.2289           0.3374
## Balanced Accuracy          0.8661          0.8200           0.8622
##                      Class: Luxueuse Class: Sportive
```

```
## Sensitivity                 0.140625         0.39642
## Specificity                 0.991693         0.99946
## Pos Pred Value              0.540541         0.99223
## Neg Pred Value              0.943196         0.90555
## Prevalence                  0.064981         0.14727
## Detection Rate              0.009138         0.05838
## Detection Prevalence        0.016905         0.05884
## Balanced Accuracy           0.566159         0.69794
```

matrice présente une précision globale similaire à la quatrième, à 72.12%. La sensibilité pour Autre et Familiale reste solide, tandis que Luxueuse et Sportive ont toujours des performances en dessous de l'idéal.

```
# Précision pour le modèle Rpart
accuracy_rpart <- sum(predictions_rpart == testingSet$Categorie) /
nrow(testingSet)
print(paste("Taux de succès de Rpart:", accuracy_rpart * 100, "%"))

## [1] "Taux de succès de Rpart: 69.9309574576099 %"

# Précision pour le modèle C50
accuracy_C50 <- sum(as.factor(predictions_C50) == testingSet$Categorie) /
nrow(testingSet)
print(paste("Taux de succès de C50:", accuracy_C50 * 100, "%"))

## [1] "Taux de succès de C50: 72.0428469895421 %"

# Précision pour le modèle Tree
accuracy_tree <- sum(predictions_tree == testingSet$Categorie) /
nrow(testingSet)
print(paste("Taux de succès de Tree:", accuracy_tree * 100, "%"))

## [1] "Taux de succès de Tree: 69.9309574576099 %"

# Précision pour le modèle SVM
accuracy_SVM <- sum(predictions_SVM == testingSet$Categorie) /
nrow(testingSet)
print(paste("Taux de succès de SVM:", accuracy_SVM * 100, "%"))

## [1] "Taux de succès de SVM: 71.7382475378211 %"

# Précision pour le modèle Random Forest
accuracy_RF <- sum(predictions_RF == testingSet$Categorie) / nrow(testingSet)
print(paste("Taux de succès de Random Forest:", accuracy_RF * 100, "%"))

## [1] "Taux de succès de Random Forest: 72.124073510001 %"
```

#interpretation

Rpart (Arbre de décision récursif partitionné): Le modèle Rpart présente un taux de succès de 69.93%, ce qui est en général acceptable mais indique que des améliorations sont

nécessaires, surtout si l'on compare aux autres modèles ou au besoin de précisions plus élevées pour des applications spécifiques.

C50 (Un type d'arbre de décision): Avec un taux de succès de 72.04%, le modèle C50 est légèrement plus performant que Rpart. Cette amélioration peut être due à des heuristiques ou des modes de traitement des données différenciés propres à l'algorithme C5.0.

Tree (un autre modèle d'arbre de décision): Ce modèle a le même taux de succès que Rpart, probablement parce qu'il est basé sur une méthode semblable de classification via des arbres de décision. Le choix entre les deux se fera en fonction des caractéristiques spécifiques des données ou des préférences de modélisation.

SVM (Machine à vecteurs de support): Avec un taux de 71.74%, le SVM offre une certaine amélioration par rapport aux arbres de décision. Cela peut être dû à sa capacité à traiter efficacement des jeux de données complexes et à trouver une frontière de décision optimale dans un espace à plus grandes dimensions.

Random Forest: Le modèle Random Forest présente le meilleur taux de succès avec 72.12%. Cela reflète l'efficacité de mixer plusieurs arbres de décision pour obtenir un modèle général plus robuste et moins sujet au surajustement.