Introduction au Machine Learning

Fabien Haury

2022-10-24

1 Librairies

```
library(dplyr, quietly = TRUE)
library(tidyverse, quietly = TRUE)
library(tidymodels, quietly = TRUE)
tidymodels_prefer(quiet = TRUE)
library(baguette, quietly = TRUE)
library(palmerpenguins, quietly = TRUE)
library(explore, quietly = TRUE)
library(tree, quietly = TRUE)
library(parttree, quietly = TRUE)
library(rattle, quietly = TRUE)
library(gt, quietly = TRUE)

library(partykit, quietly = TRUE)

library(rpart, quietly = TRUE)
library(rpart.plot, quietly = TRUE)

cores <- parallel::detectCores()
```

2 Spécification police et thème

```
library(extrafont, quietly = TRUE)
loadfonts(device = "win")

theme_set(theme_bw(base_family = "serif"))
theme_update(
  axis.title = element_text(size = 15),
  axis.text.x = element_text(size = 12),
  axis.text.y = element_text(size = 12)
)
```

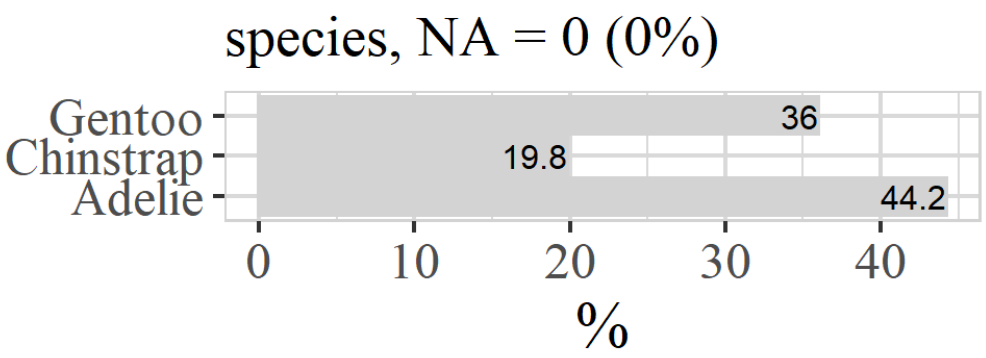
3 Projet 1

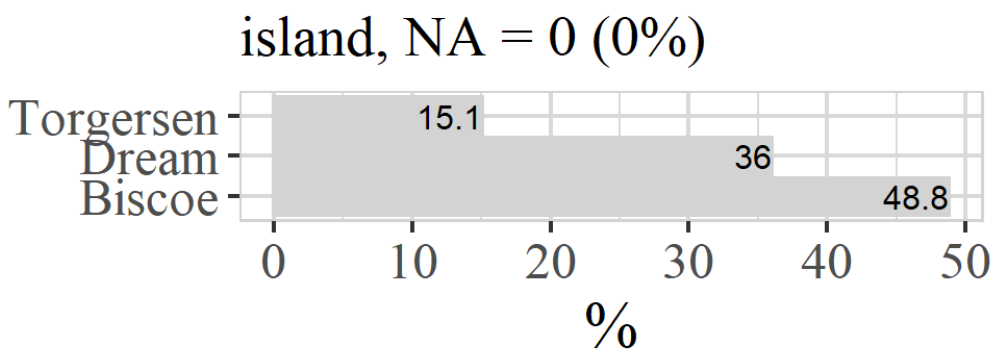
3.1 Import/préparation données

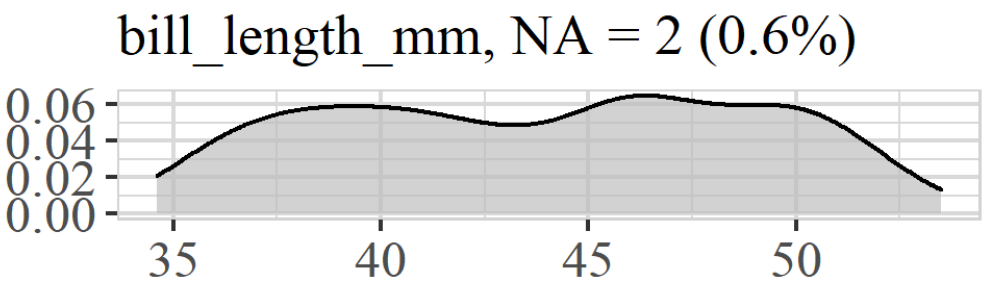
```
data(penguins)

penguin <- penguins

penguin <- penguin %>% na.omit()
penguin <- penguin %>% select(species, bill_length_mm, bill_depth_mm)
```

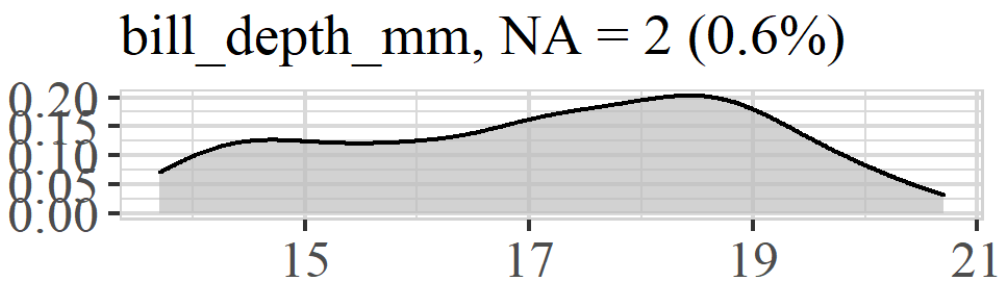
3.2 Rapide exploration

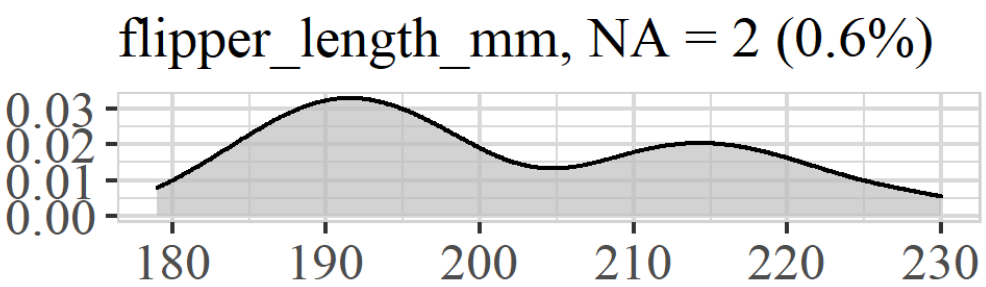
```
explore_all(penguins)
```

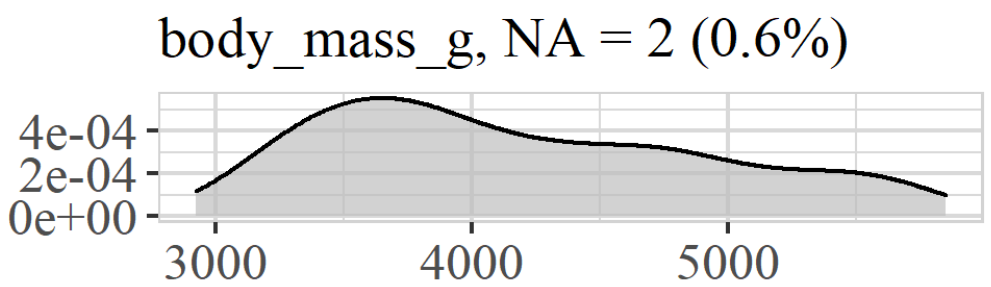

species, NA = 0 (0%)
Gentoo
Chinstrap
Adelie
36
19.8
44.2
0
10
20
30
40
%

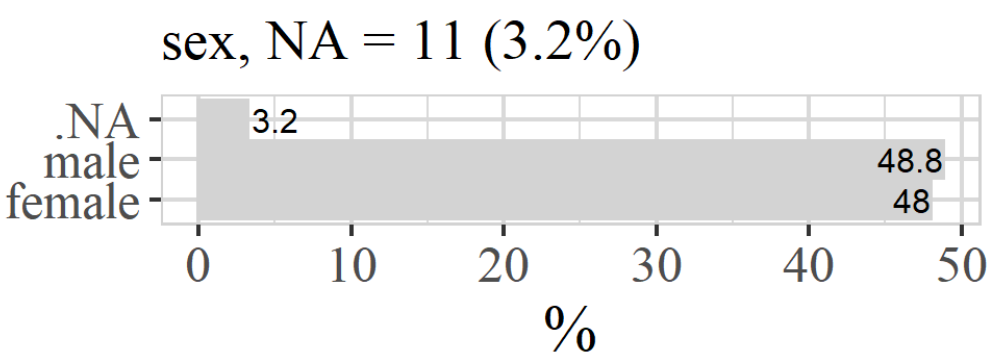

island, NA = 0 (0%)
Torgersen
Dream
Biscoe
15.1
36
48.8
0
10
20
30
40
50
%

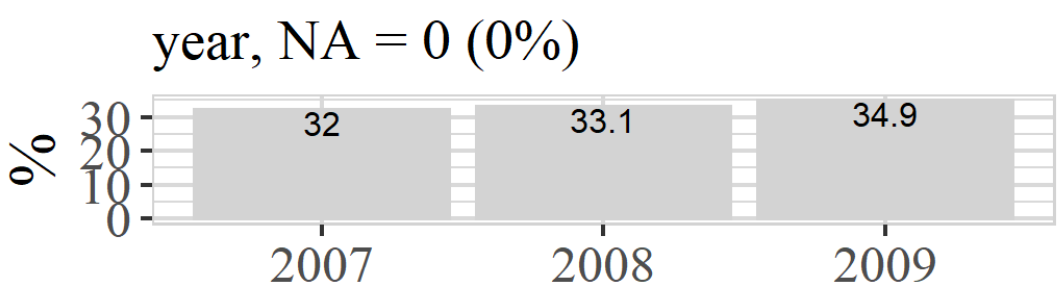

bill_length_mm, NA = 2 (0.6%)
0.06
0.04
0.02
0.00
35
40
45
50


bill_depth_mm, NA = 2 (0.6%)
0.20
0.15
0.10
0.05
0.00
15
17
19
21

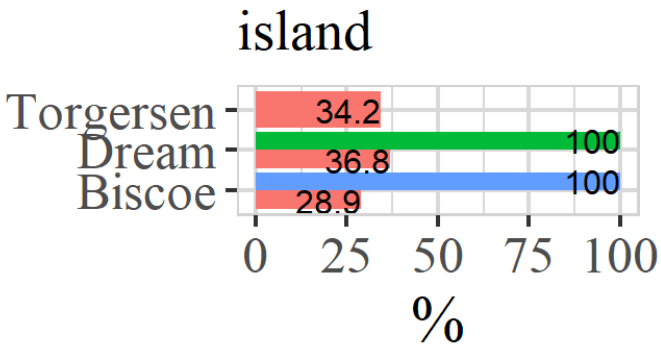

flipper_length_mm, NA = 2 (0.6%)
0.03
0.02
0.01
0.00
180
190
200
210
220
230


body_mass_g, NA = 2 (0.6%)
4e-04
2e-04
0e+00
3000
4000
5000

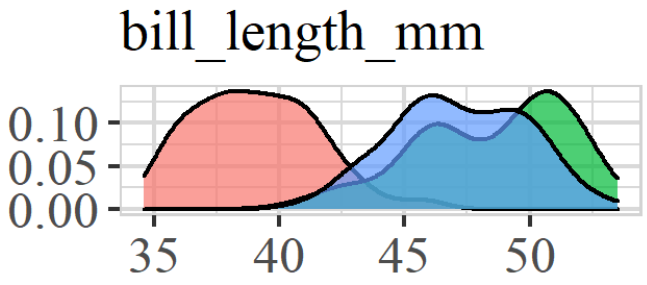

sex, NA = 11 (3.2%)
.NA
male
female
3.2
48.8
48
0
10
20
30
40
50
%

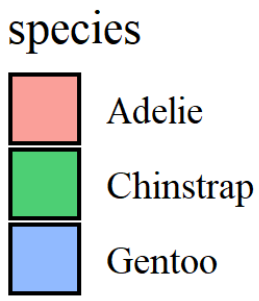

year, NA = 0 (0%)
%
30
20
10
0
32
33.1
34.9
2007
2008
2009

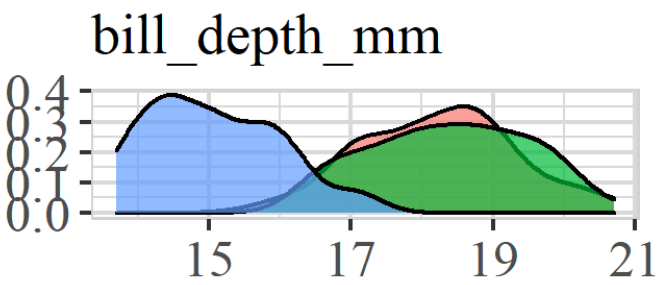
```
explore_all(penguins, target = species)
```

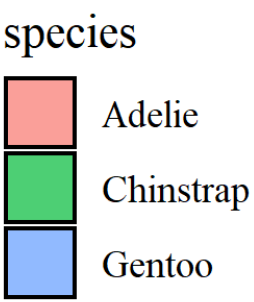

island
Torgersen
Dream
Biscoe
34.2
100
36.8
100
28.9
0
25
50
75
100
%

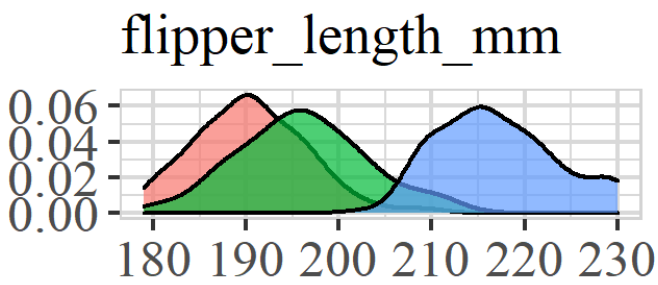

species
Adelie
Chinstrap
Gentoo

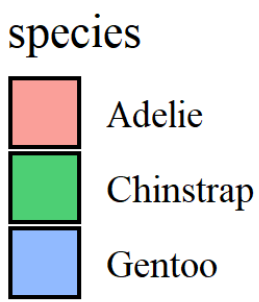

bill_length_mm
0.10
0.05
0.00
35
40
45
50

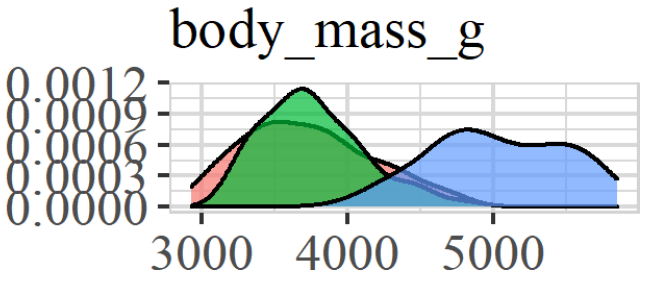

species
Adelie
Chinstrap
Gentoo

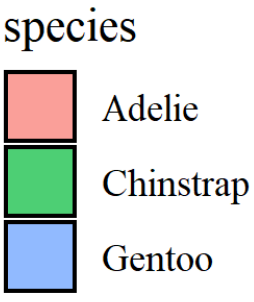

bill_depth_mm
0.4
0.3
0.2
0.1
0.0
15
17
19
21

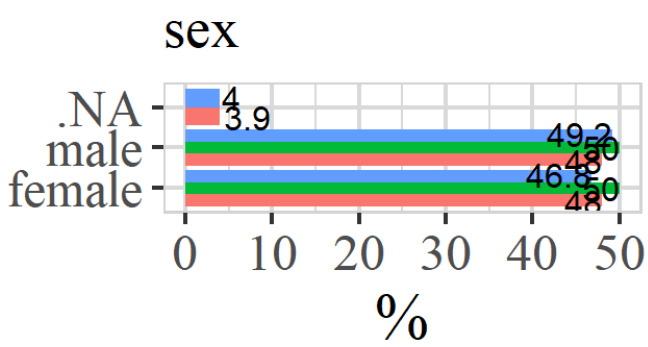

species
Adelie
Chinstrap
Gentoo


flipper_length_mm
0.06
0.04
0.02
0.00
180
190
200
210
220
230

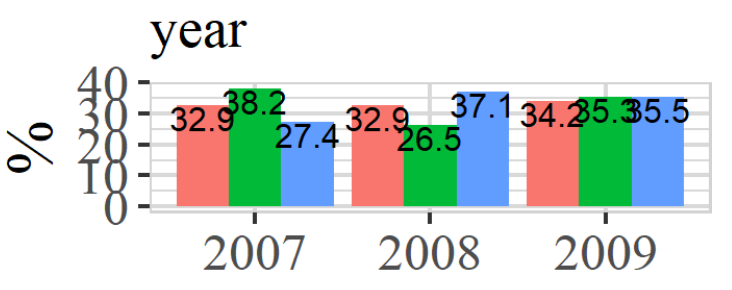

species
Adelie
Chinstrap
Gentoo


body_mass_g
0.0012
0.0009
0.0006
0.0003
0.0000
3000
4000
5000


species
Adelie
Chinstrap
Gentoo


sex
.NA
male
female
4
3.9
49.2
46.8
0
10
20
30
40
50
%


species
Adelie
Chinstrap
Gentoo


year
%
40
30
20
10
0
32.9
38.2
27.4
32.9
26.5
37.1
34.2
35.3
35.5
2007
2008
2009

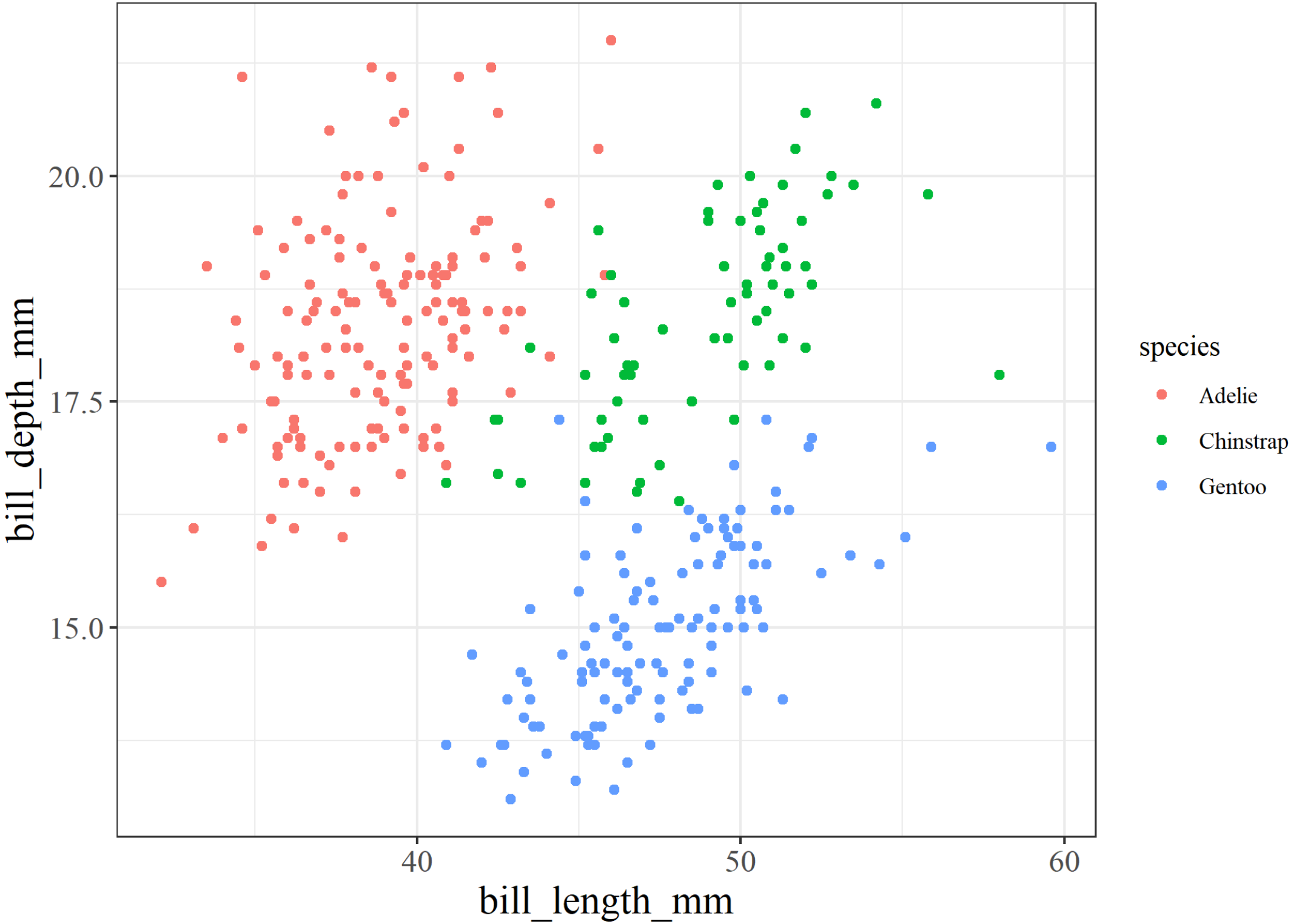

species
Adelie
Chinstrap
Gentoo

3.3 Question 1

```
# Fit
dtc_pg_mod_1 <- decision_tree(tree_depth = 1) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("classification") %>%
  fit(species ~ ., penguin)

# Extract fit features
dtc_pg_mod_efit_1 <- extract_fit_engine(dtc_pg_mod_1)
```

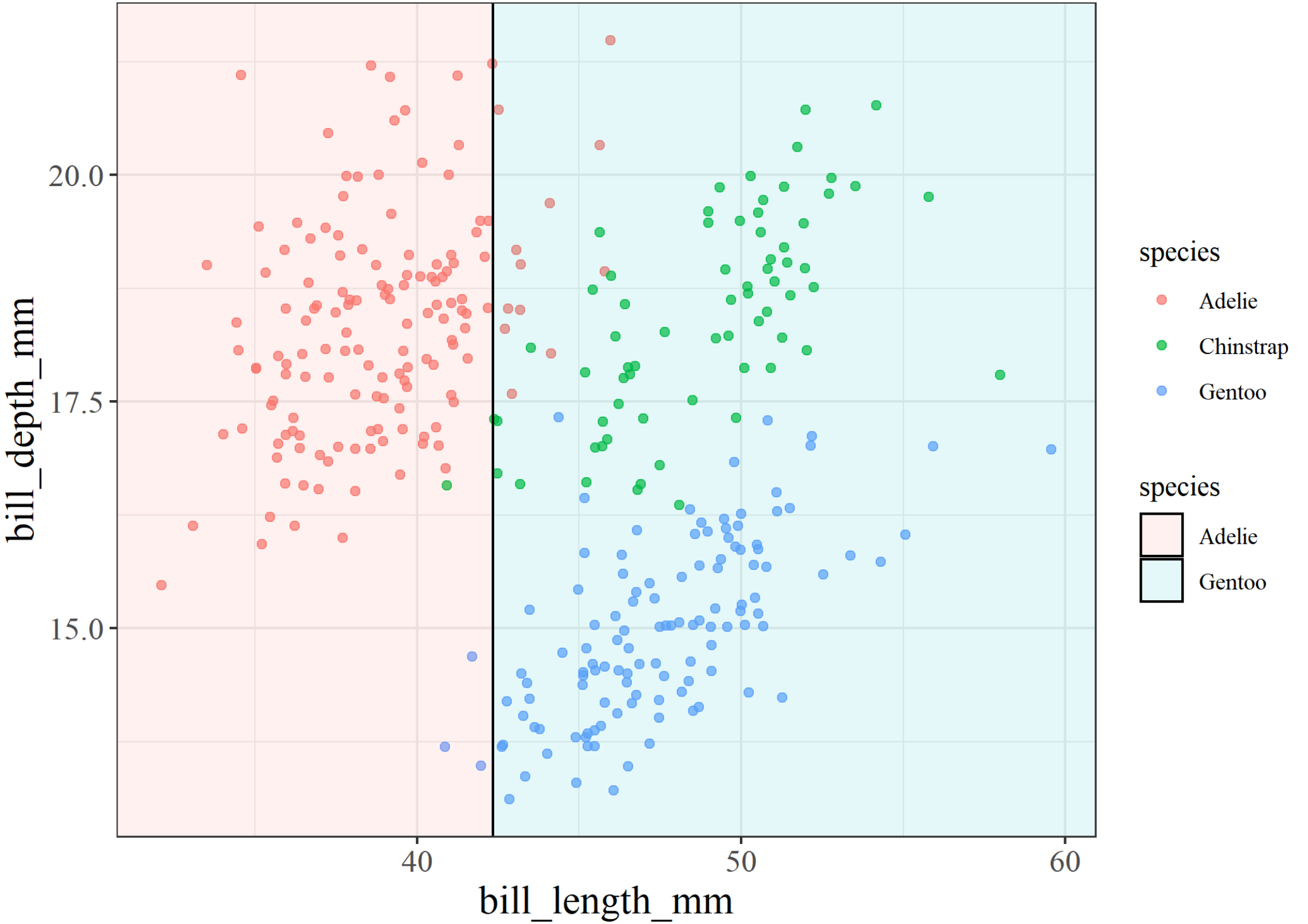
3.4 Question 2

```
penguin %>%
  ggplot(aes(bill_length_mm, bill_depth_mm, color = species)) +
  geom_point()
```


bill_depth_mm
20.0
17.5
15.0
40
50
60
bill_length_mm
species
Adelie
Chinstrap
Gentoo

3.5 Question 3

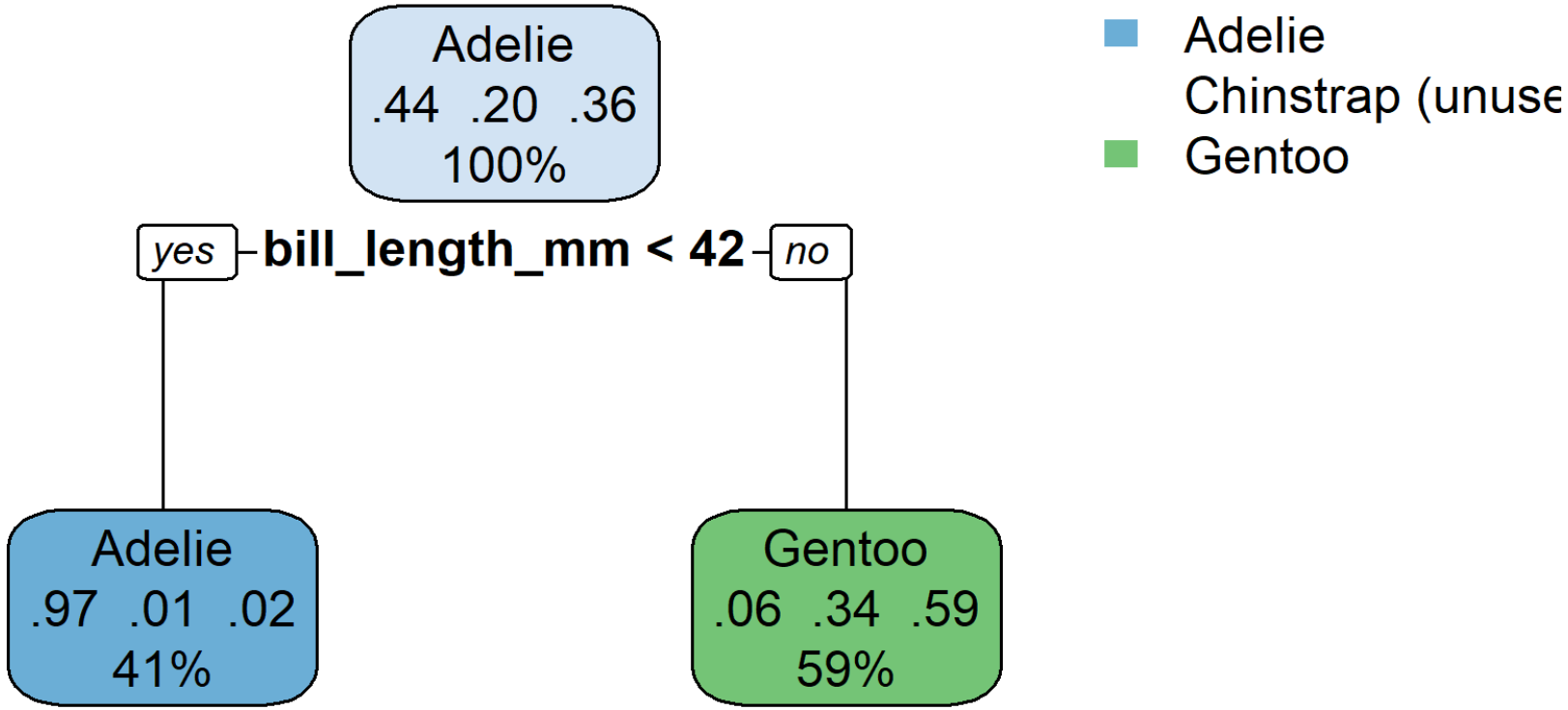
```
penguin %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm)) +
  geom_jitter(aes(col = species), alpha = 0.7) +
  geom_parttree(data = dtc_pg_mod_1, aes(fill = species), alpha = 0.1)
```


bill_depth_mm
20.0
17.5
15.0
40
50
60
bill_length_mm
species
Adelie
Chinstrap
Gentoo
species
Adelie
Gentoo

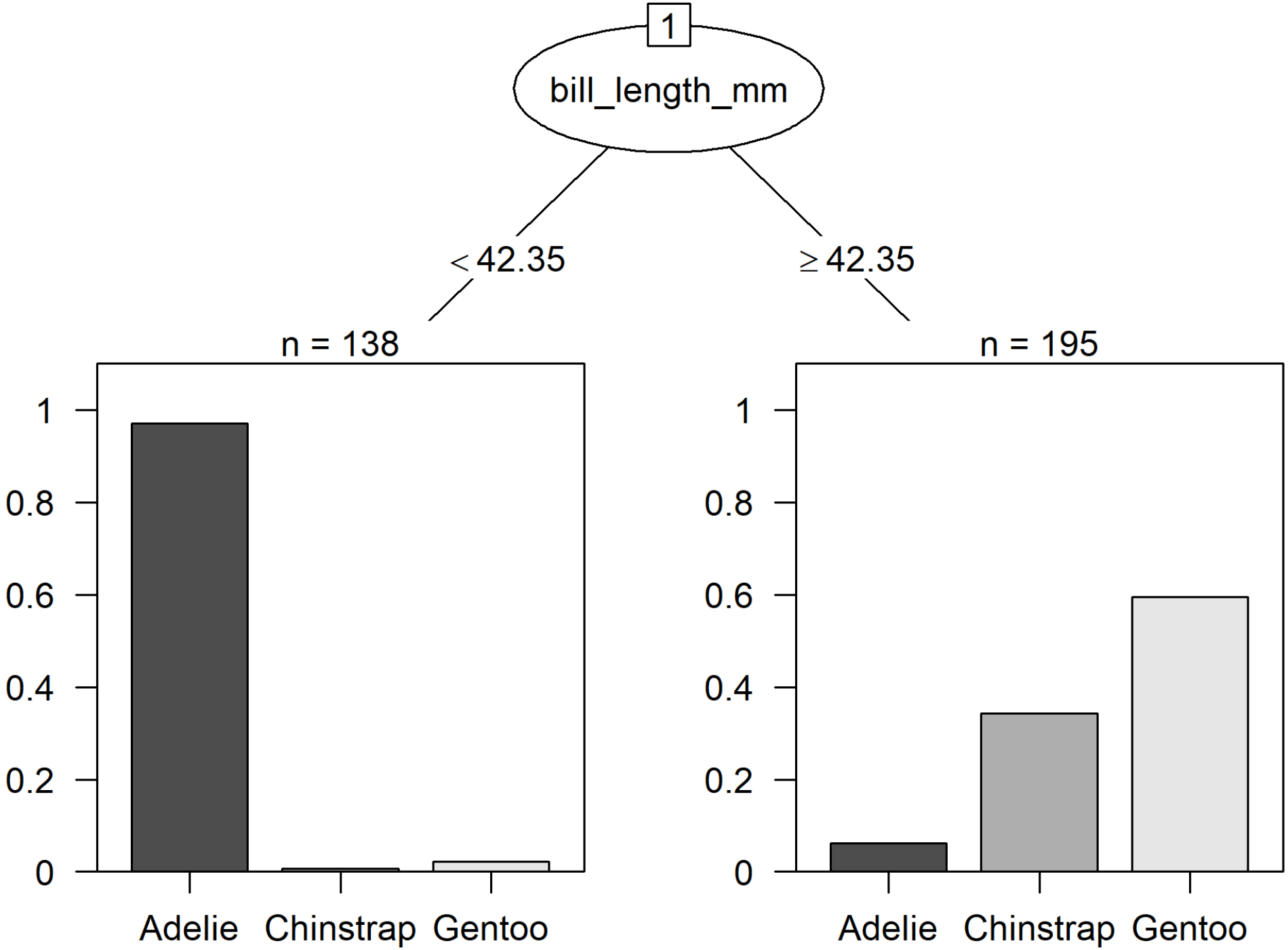
L’algorithme d’arbre de classification a sélectionné la longueur du bec pour réaliser la première partition.

3.6 Question 4

```
rpart.plot(dtc_pg_mod_efit_1, extra = "auto")
```


Adelie
.44 .20 .36
100%
yes
bill_length_mm < 42
no
Adelie
.97 .01 .02
41%
Gentoo
.06 .34 .59
59%
Adelie
Chinstrap (unuse
Gentoo

```
plot(as.party(dtc_pg_mod_efit_1), tp_args = list(id = FALSE))
```

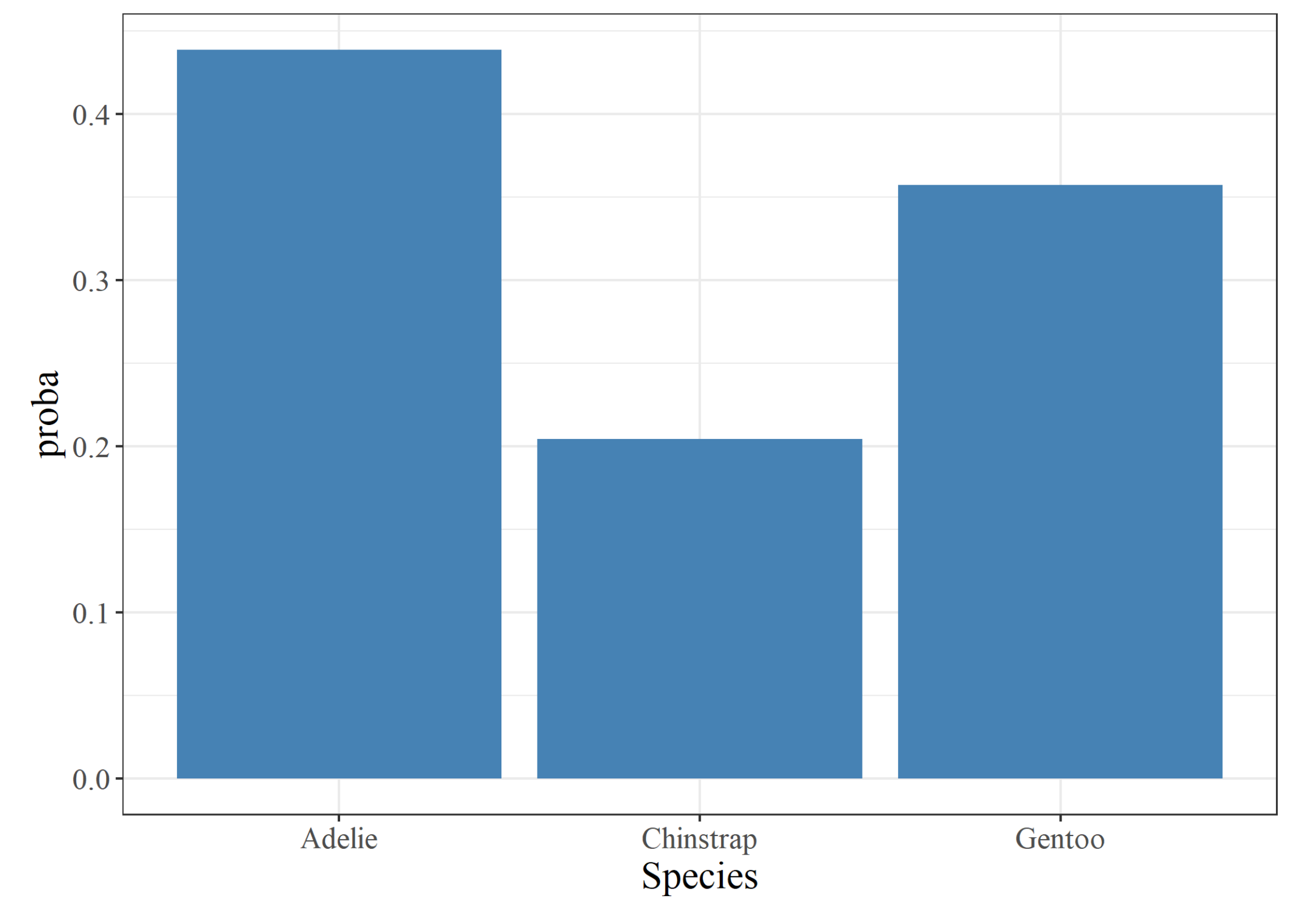

1
bill_length_mm
< 42.35
≥ 42.35
n = 138
n = 195
1
0.8
0.6
0.4
0.2
0
Adelie
Chinstrap
Gentoo

Pour la racine, l'espèce Adelie est choisie, car elle représente 44 % du total jeu de données. La feuille numéro 2 est labellisée comme étant Adelie avec 97 % du total de cette feuille. En regardant le second graphique de la partie Rapide exploration groupé par espèces, nous pouvons voir que l'espèce Adelie représente la quasi-totalité des pingouins ayant une longueur de bec inférieure à 42 mm. La feuille numéro 3 est labellisée comme étant Gentoo avec 59 % du total de cette feuille.

Si le modèle suivait une logique de prédiction, il ne prédirait que des pingouins d'espèce Adelie ou Gentoo. Ce modèle ferait la séparation entre ces deux espèces uniquement basée sur la longueur du bec.

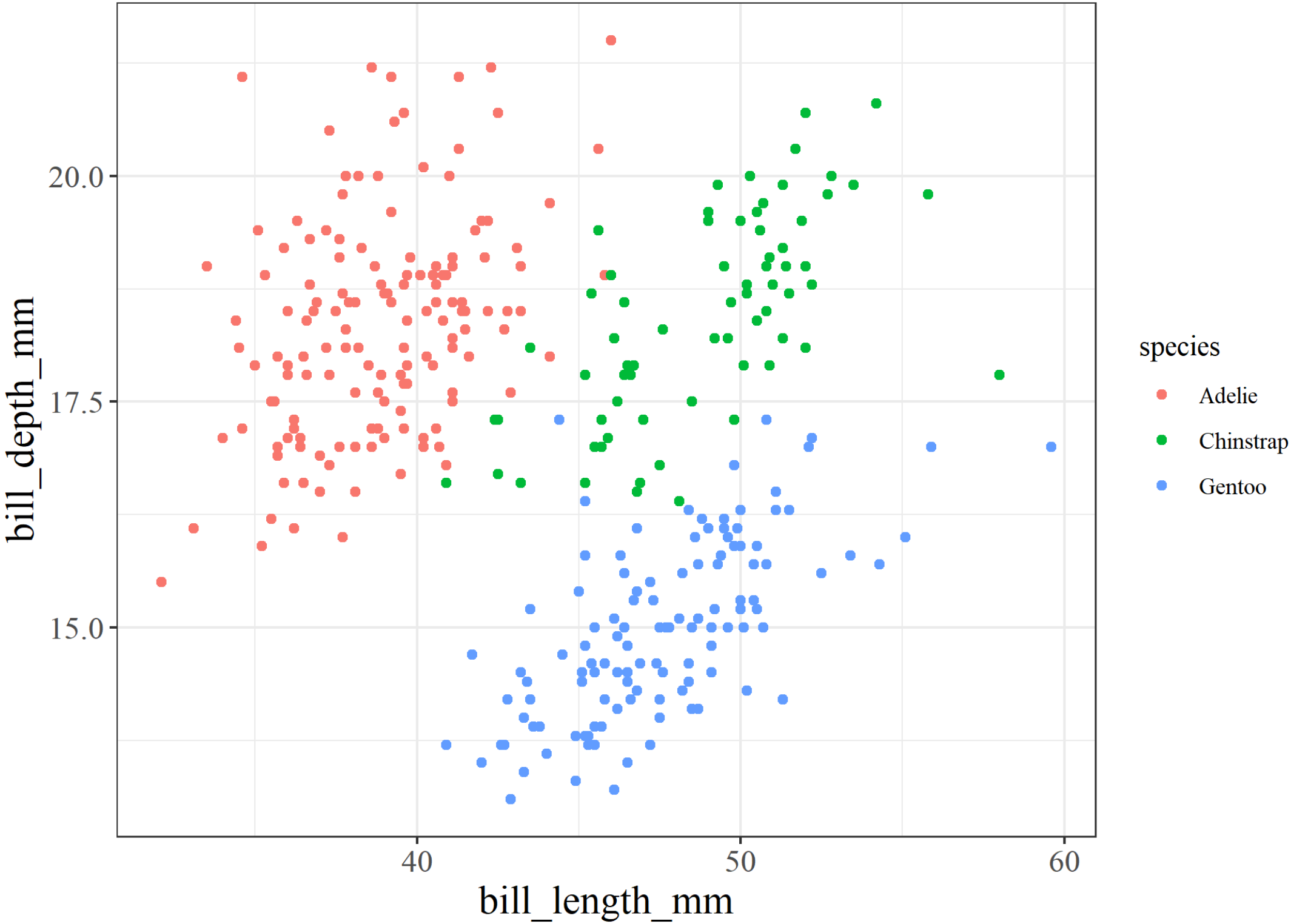
3.7 Question 5

```
# Création jeu de données imaginaire.
penguin_imaginaire <- tibble(
  species = factor(sample(rep(c("Adelie","Gentoo", "Chinstrap"), times = c(36, 20, 44)))),
  bill_length_mm = 35,
  bill_depth_mm = 17
)

# Fit imaginaire
dtc_pg_imaginaire_mod_1 <- decision_tree(tree_depth = 1) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("classification") %>%
  fit(species ~ ., penguin_imaginaire)

# Predictions
dtc_pg_imaginaire_1_pred <- bind_cols(
  select(penguin, species),
  predict(dtc_pg_imaginaire_mod_1, penguin)
)

dtc_pg_imaginaire_1_pred %>%
  count(species, .pred_class) %>%
  mutate(proba = n / sum(n)) %>%
  ggplot(aes(species, proba)) +
  geom_col(fill = "steelblue") +
  xlab("Species")
```

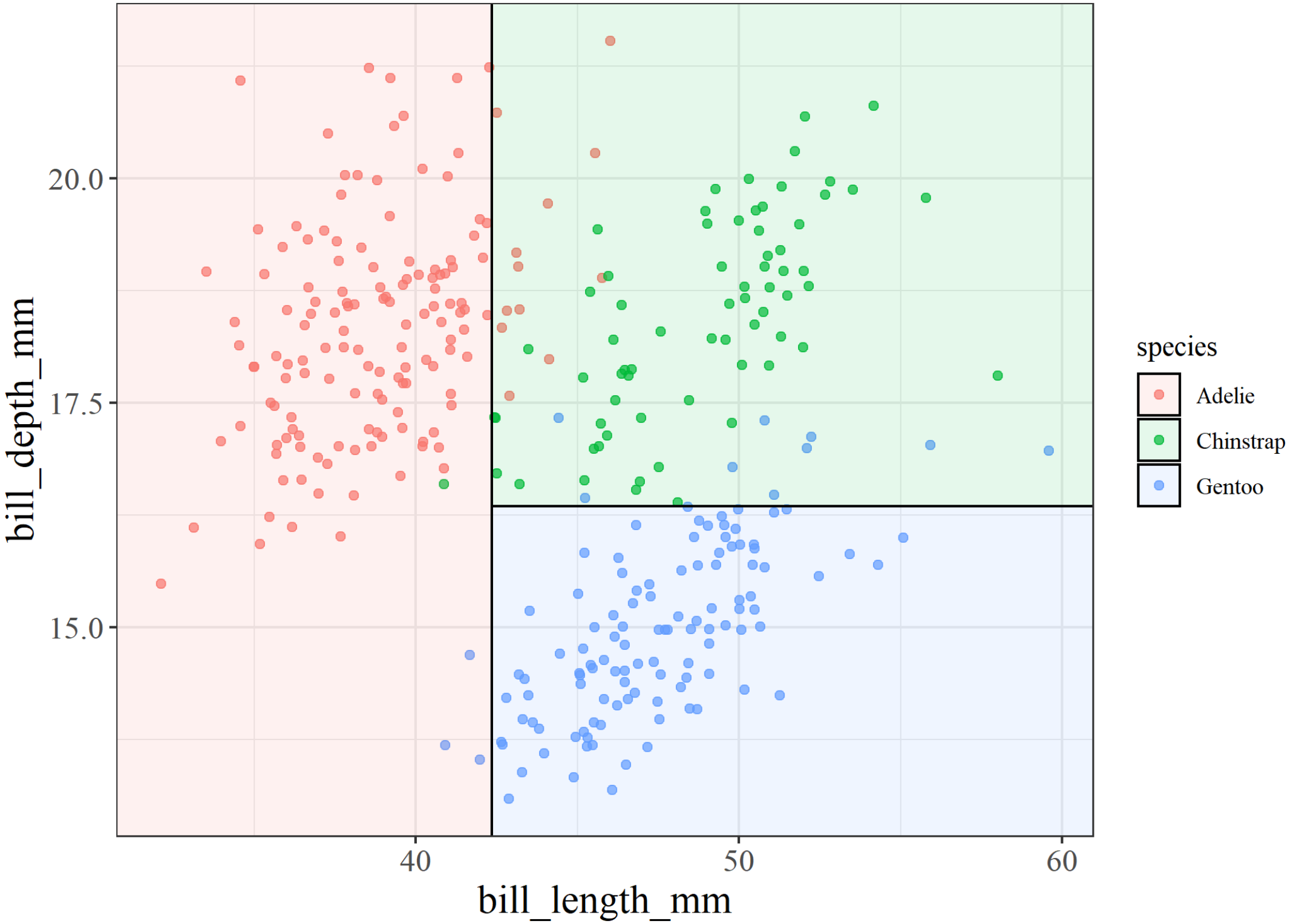

0.4
0.3
0.2
0.1
0.0
proba
Adelie
Chinstrap
Gentoo
Species

3.8 Question 6

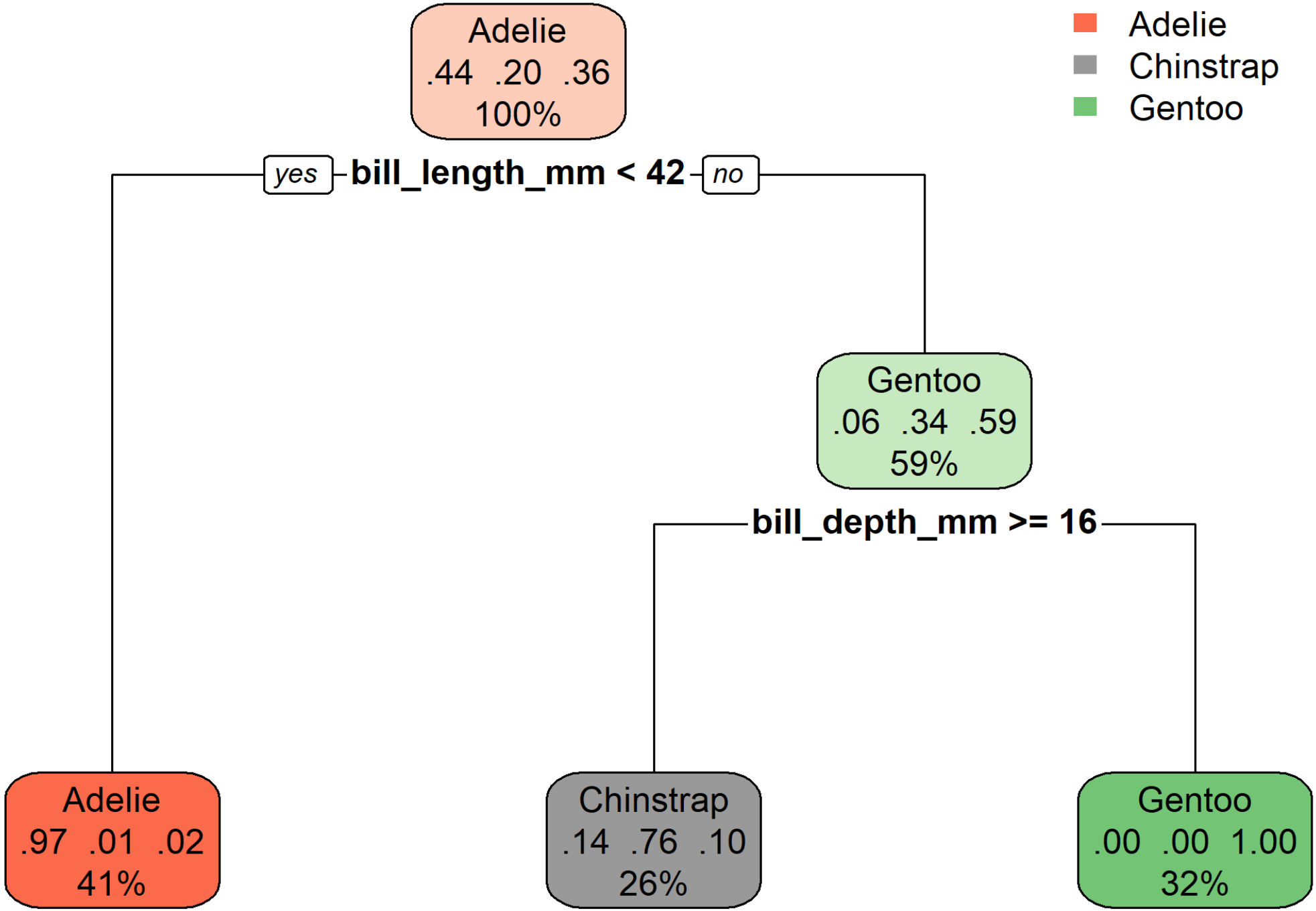
```
# Fit
dtc_pg_mod_2 <- decision_tree(tree_depth = 2) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("classification") %>%
  fit(species ~ ., penguin)

# Extract fit features
dtc_pg_mod_efit_2 <- extract_fit_engine(dtc_pg_mod_2)

penguin %>%
  ggplot(aes(bill_length_mm, bill_depth_mm, color = species)) +
  geom_point()
```


bill_depth_mm
20.0
17.5
15.0
40
50
60
bill_length_mm
species
Adelie
Chinstrap
Gentoo

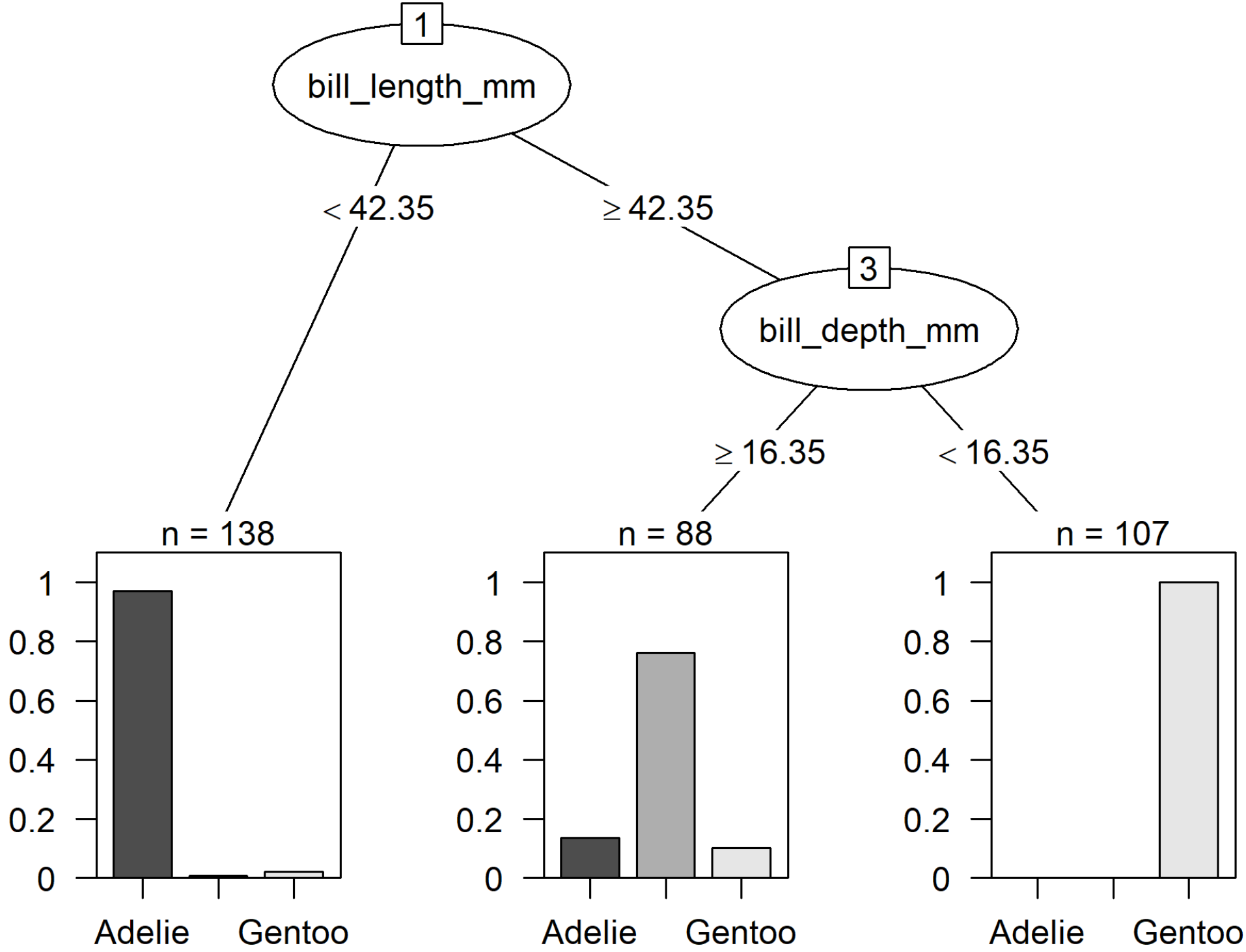
```r
penguin %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm)) +
  geom_jitter(aes(col = species), alpha = 0.7) +
  geom_parttree(data = dtc_pg_mod_2, aes(fill = species), alpha = 0.1)
```


bill_depth_mm
20.0
17.5
15.0
40
50
60
bill_length_mm
species
Adelie
Chinstrap
Gentoo

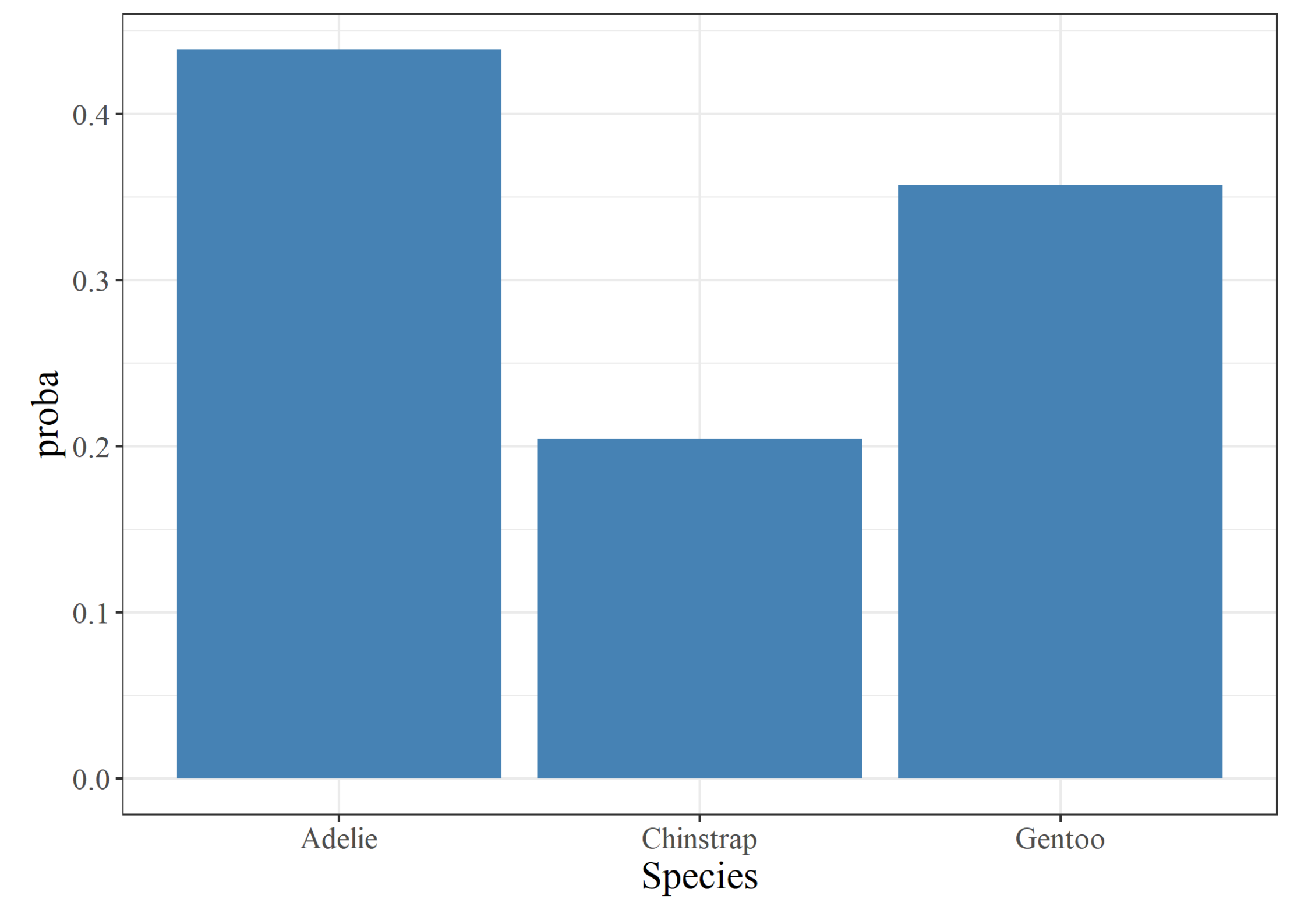
```
rpart.plot(dtc_pg_mod_efit_2, extra = "auto")
```


Adelie
Chinstrap
Gentoo
Adelie
.44 .20 .36
100%
yes
bill_length_mm < 42
no
Gentoo
.06 .34 .59
59%
bill_depth_mm >= 16
Adelie
.97 .01 .02
41%
Chinstrap
.14 .76 .10
26%
Gentoo
.00 .00 1.00
32%

```
plot(as.party(dtc_pg_mod_efit_2), tp_args = list(id = FALSE))
```

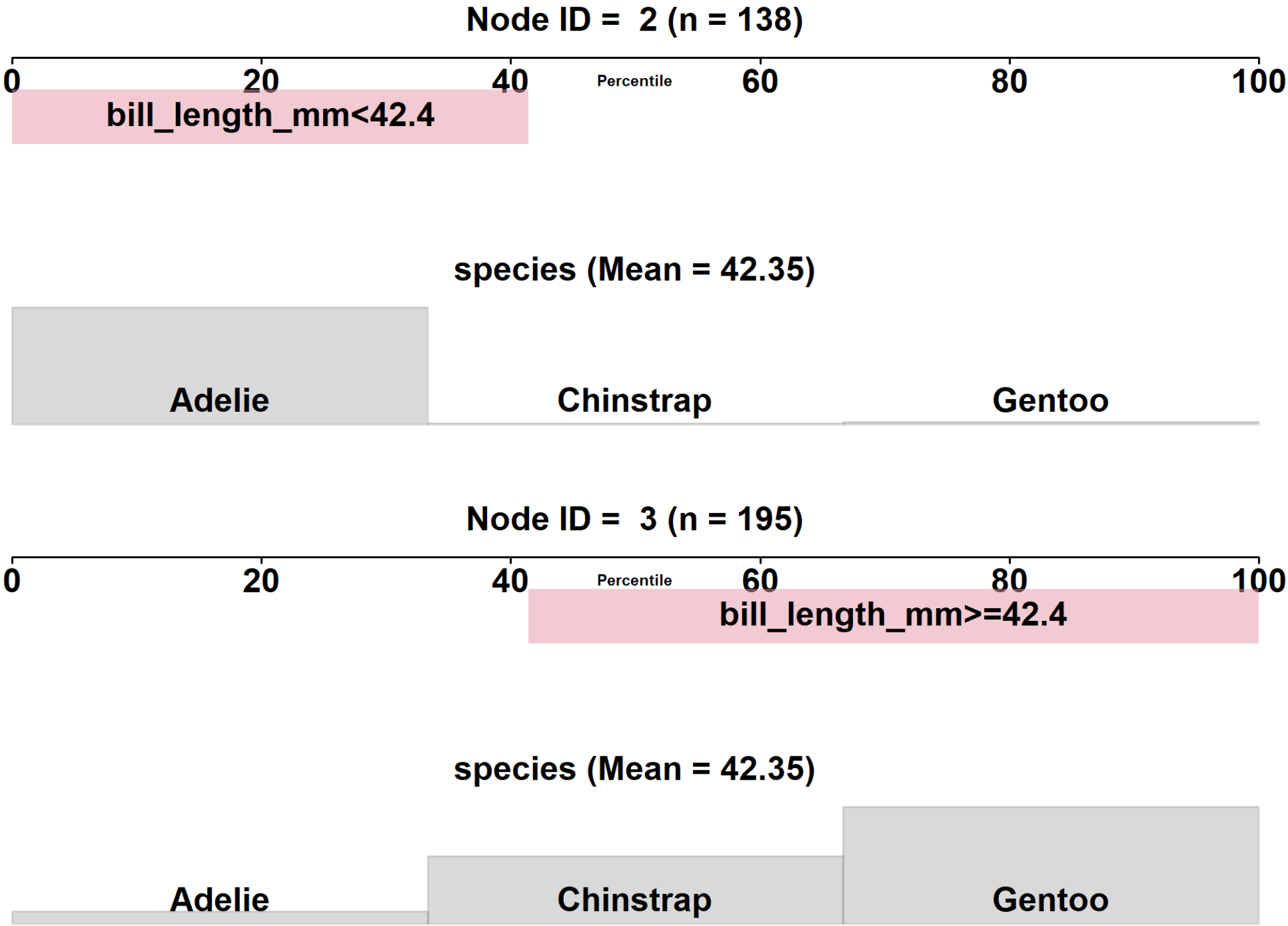

1
bill_length_mm
< 42.35
≥ 42.35
3
bill_depth_mm
≥ 16.35
< 16.35
n = 138
n = 88
n = 107
1
0.8
0.6
0.4
0.2
0
Adelie
Gentoo

```r
# Fit imaginaire
dtc_pg_imaginaire_mod_2 <- decision_tree(tree_depth = 2) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("classification") %>%
  fit(species ~ ., penguin_imaginaire)

# predictions
dtc_pg_imaginaire_2_pred <- bind_cols(
  select(penguin, species),
  predict(dtc_pg_imaginaire_mod_2, penguin))

dtc_pg_imaginaire_2_pred %>%
  count(species, .pred_class) %>%
  mutate(proba = n / sum(n)) %>%
  ggplot(aes(species, proba)) +
  geom_col(fill = "steelblue") +
  xlab("Species")
```

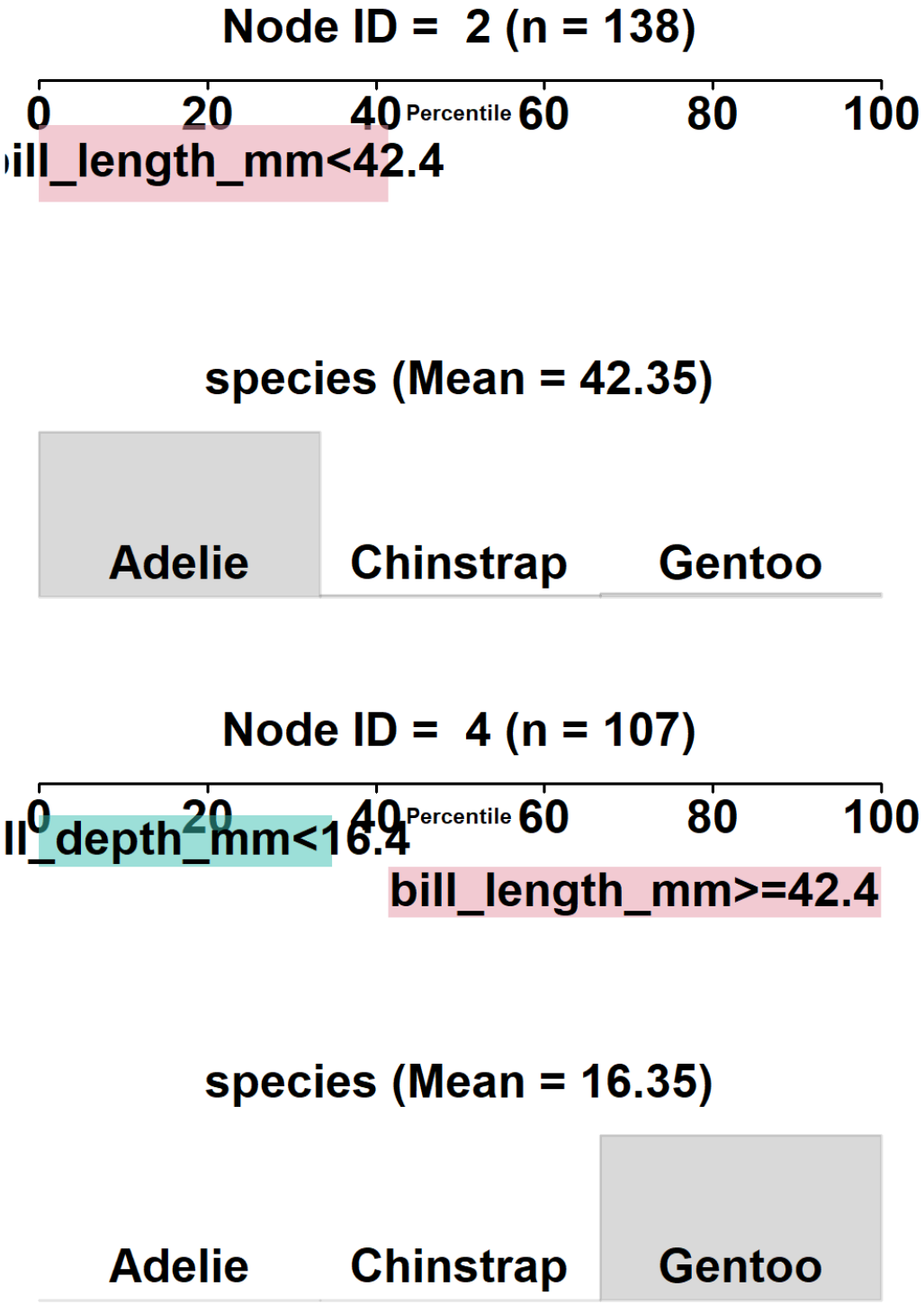

0.4
0.3
0.2
0.1
0.0
proba
Adelie
Chinstrap
Gentoo
Species

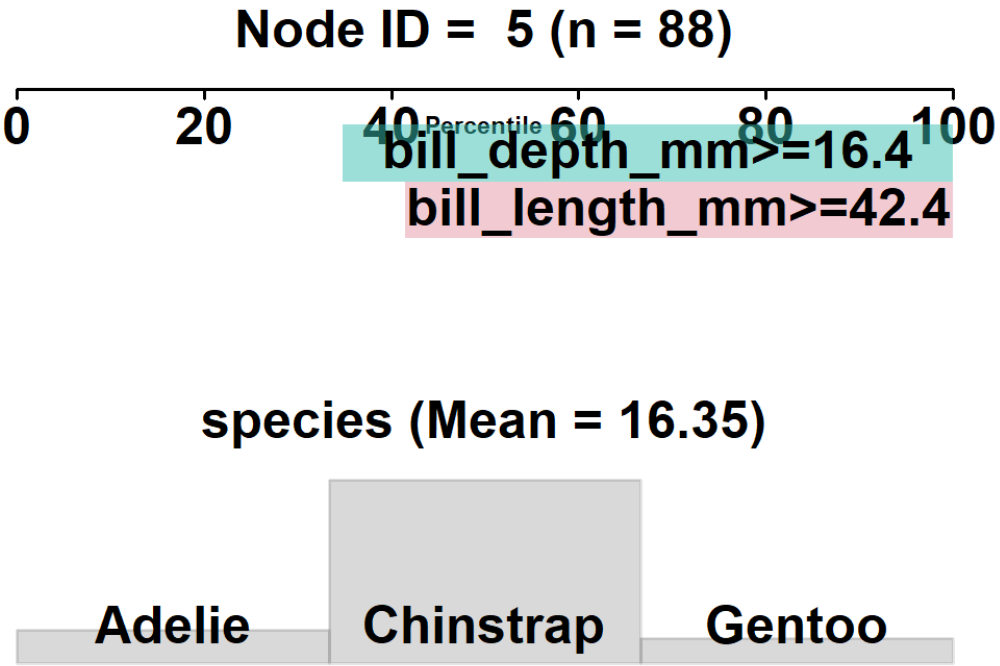
3.9 Extra

```
# Autre façon de visualiser un arbre
# model 1
visTree::visTree(dtc_pg_mod_efit_1)
```


Node ID = 2 (n = 138)
0
20
40
Percentile
60
80
100
bill_length_mm<42.4
species (Mean = 42.35)
Adelie
Chinstrap
Gentoo
Node ID = 3 (n = 195)
0
20
40
Percentile
60
80
100
bill_length_mm>=42.4
species (Mean = 42.35)
Adelie
Chinstrap
Gentoo

```r
# model 2
visTree::visTree(dtc_pg_mod_efit_2)
```


Node ID = 2 (n = 138)
0
20
40
Percentile
60
80
100
ill_length_mm<42.4
species (Mean = 42.35)
Adelie
Chinstrap
Gentoo
Node ID = 4 (n = 107)
0
20
40
Percentile
60
80
100
ll_depth_mm<16.4
bill_length_mm>=42.4
species (Mean = 16.35)
Adelie
Chinstrap
Gentoo


Node ID = 5 (n = 88)
0
20
40
Percentile
60
80
100
bill_depth_mm>=16.4
bill_length_mm>=42.4
species (Mean = 16.35)
Adelie
Chinstrap
Gentoo

4 Projet 2

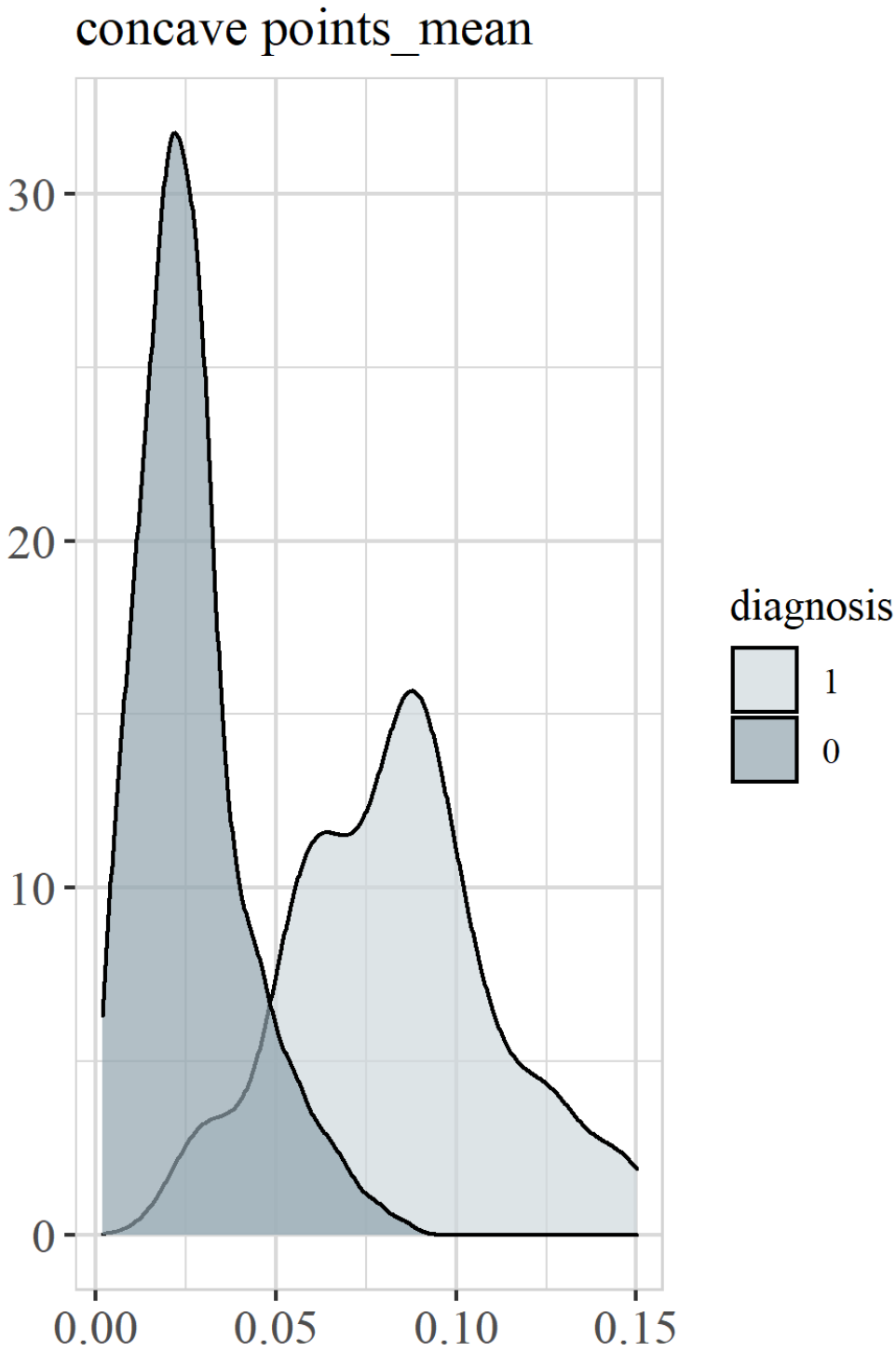
4.1 Exercice 1 : Entrainer un arbre de classification

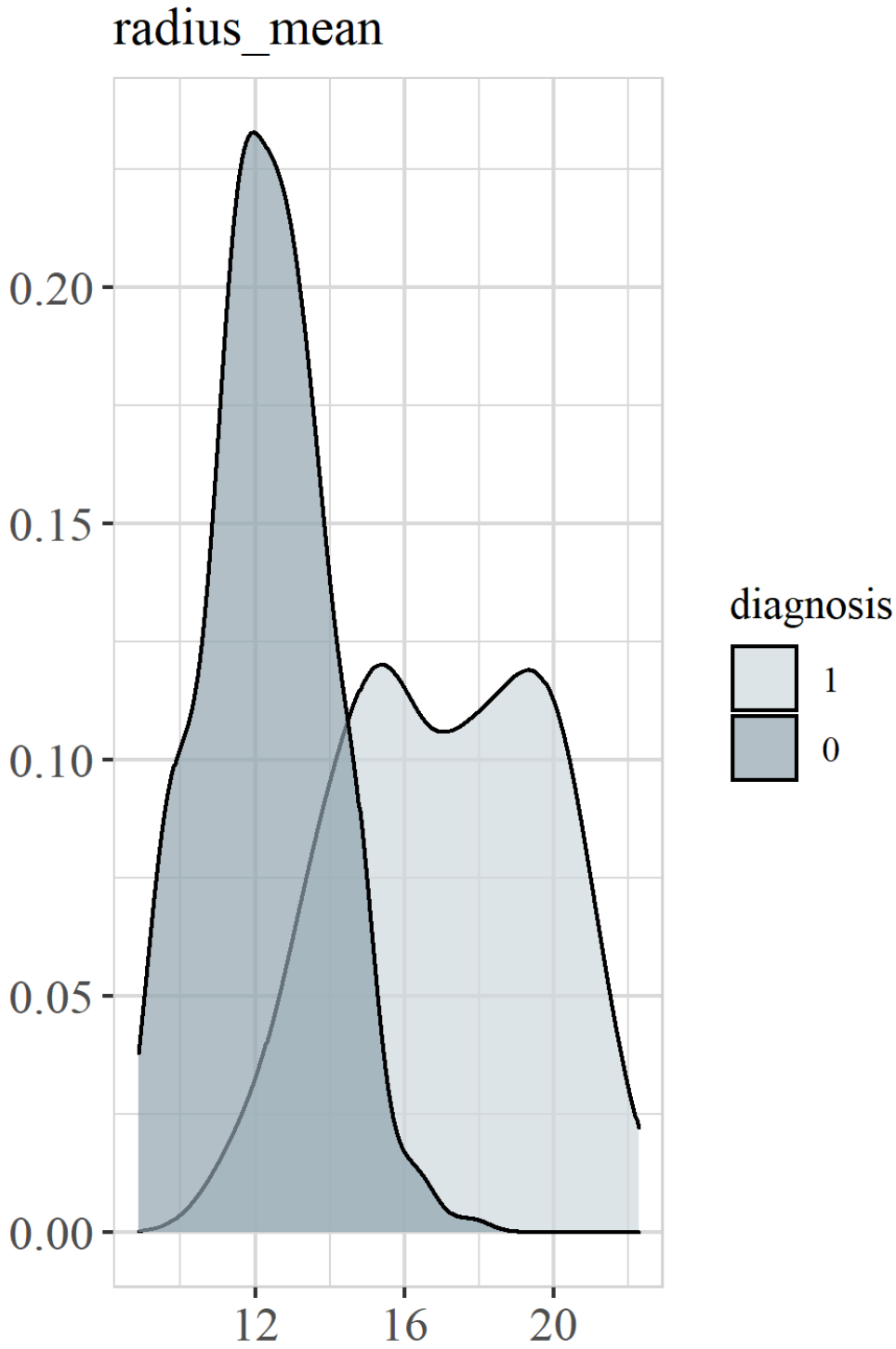
4.1.1 Import/préparation données

```
breast_cancer <- read_csv("dataset/breast_cancer.csv")

breast_cancer <- breast_cancer %>% select(diagnosis, `concave points_mean`, radius_mean)

breast_cancer$diagnosis <- breast_cancer$diagnosis %>%
  fct_relevel(
    "M", "B"
  )

breast_cancer$diagnosis <- factor(breast_cancer$diagnosis,
                                  levels = c("M", "B"),
                                  labels = c(1, 0)
)
```

4.1.2 Rapide exploration

```
explore_all(breast_cancer, target = diagnosis)
```


concave points_mean
30
20
10
0
0.00
0.05
0.10
0.15
diagnosis
1
0


radius_mean
0.20
0.15
0.10
0.05
0.00
12
16
20
diagnosis
1
0

4.1.3 Question 1

```
set.seed(1)

bc_split <- initial_split(breast_cancer, 0.8, strata = diagnosis)

bc_train <- training(bc_split)
bc_test <- testing(bc_split)
```

4.1.4 Question 2

```
# Model
dtc_bc_mod <- decision_tree(tree_depth = 6) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("classification")

# Workflow
dtc_bc_wflow <- workflow() %>%
  add_model(dtc_bc_mod) %>%
  add_variables(outcomes = diagnosis, predictors = everything())
```

4.1.5 Question 3

```
# Fit
dtc_bc_fit <- fit(dtc_bc_wflow, bc_train)
```

4.1.6 Question 4

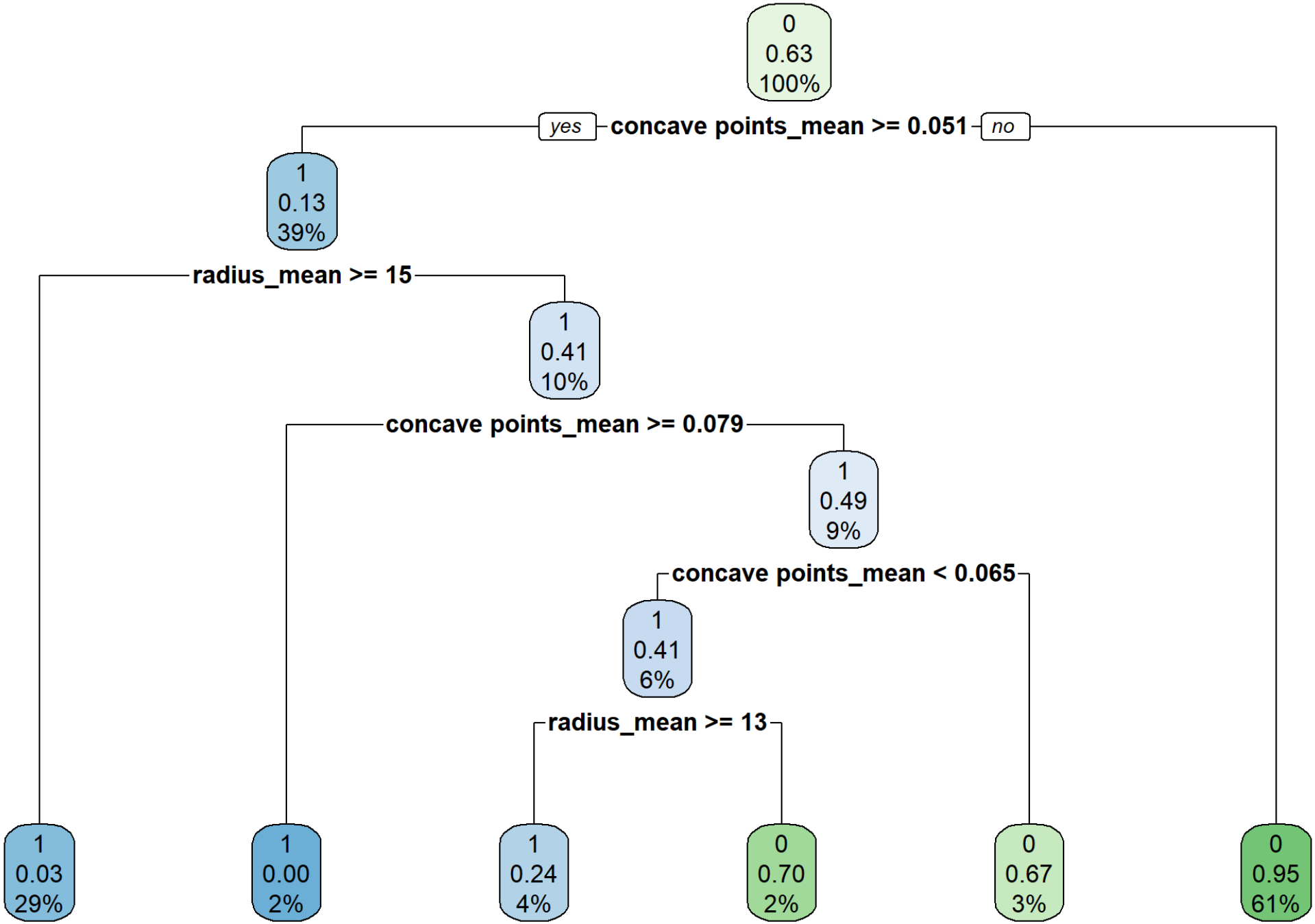
```
# prediction
dtc_bc_predictions <- bind_cols(
  select(bc_test, diagnosis),
  predict(dtc_bc_fit, bc_test)
```

```
)

head(dtc_bc_predictions, 5) %>% gt()
```

diagnosis	.pred_class
1	1
1	0
1	1
1	1
1	1

4.1.7 Question 5

```
extract_fit_engine(dtc_bc_fit) %>% rpart.plot(extra = "auto")
```


0
0.63
100%
yes
concave points_mean >= 0.051
no
1
0.13
39%
radius_mean >= 15
1
0.41
10%
concave points_mean >= 0.079
1
0.49
9%
concave points_mean < 0.065
1
0.41
6%
radius_mean >= 13
1
0.03
29%
1
0.00
2%
1
0.24
4%
0
0.70
2%
0
0.67
3%
0
0.95
61%

4.2 Exercice 2 : Evaluation d'un arbre, choix du critère d'information

4.2.1 Question 1

```
metrics(dtc_bc_predictions, truth = diagnosis, estimate = .pred_class) %>%
  gt() %>% fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate
accuracy	binary	0.90
kap	binary	0.77

4.2.2 Question 2

```
# Model
dtc_bc_mod_entropy <- decision_tree() %>%
  set_mode("classification") %>%
  set_engine("rpart", model = TRUE, parms = list(split = "information"))

# Workflow
dtc_bc_wflow_entropy <- workflow() %>%
  add_model(dtc_bc_mod_entropy) %>%
  add_variables(outcomes = diagnosis, predictors = everything())

# Fit
dtc_bc_fit_entropy <- fit(dtc_bc_wflow_entropy, bc_train)

# Predictions
dtc_bc_predictions_entropy <- bind_cols(
  select(bc_test, diagnosis),
  predict(dtc_bc_fit_entropy, bc_test)
)
```

```
head(dtc_bc_predictions_entropy) %>% gt()
```

diagnosis	.pred_class
1	1
1	1
1	1
1	1
1	1
1	1

```
# Model
dtc_bc_mod_gini <- decision_tree() %>%
  set_mode("classification") %>%
  set_engine("rpart", model = TRUE, parms = list(split = "gini"))

# Workflow
dtc_bc_wflow_gini <- workflow() %>%
  add_model(dtc_bc_mod_gini) %>%
  add_variables(outcomes = diagnosis, predictors = everything())

# Fit
dtc_bc_fit_gini <- fit(dtc_bc_wflow_gini, bc_train)

# Predictions
dtc_bc_predictions_gini <- bind_cols(
  select(bc_test, diagnosis),
  predict(dtc_bc_fit_gini, bc_test)
)

head(dtc_bc_predictions_gini) %>% gt()
```

diagnosis	.pred_class
1	1
1	0
1	1
1	1
1	1
1	1

4.2.3 Question 3

L'indice de Gini est la probabilité qu'une variable ne soit pas classée correctement si elle était choisie au hasard.

La formule de l'indice de Gini est la suivante :

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

L'entropie est une mesure de l'information (ou plutôt de son absence). On calcule le gain d'information en faisant une division. Ce qui correspond à la différence d'entropie. Cela mesure comment vous réduisez l'incertitude sur l'étiquette.

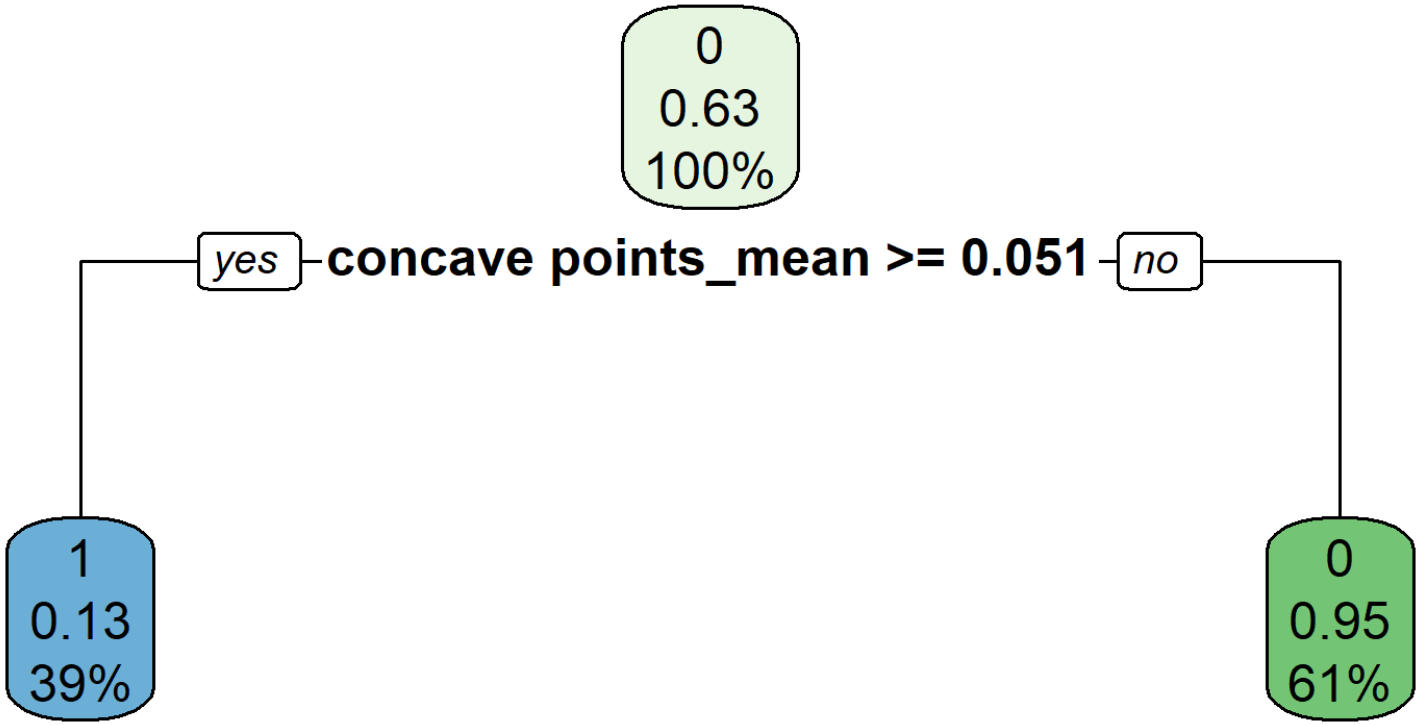
La formule de l'entropie est la suivante :

$$H(X) = -\sum_{i=1}^{k} P_i log_2 P_i$$

4.2.4 Question 4

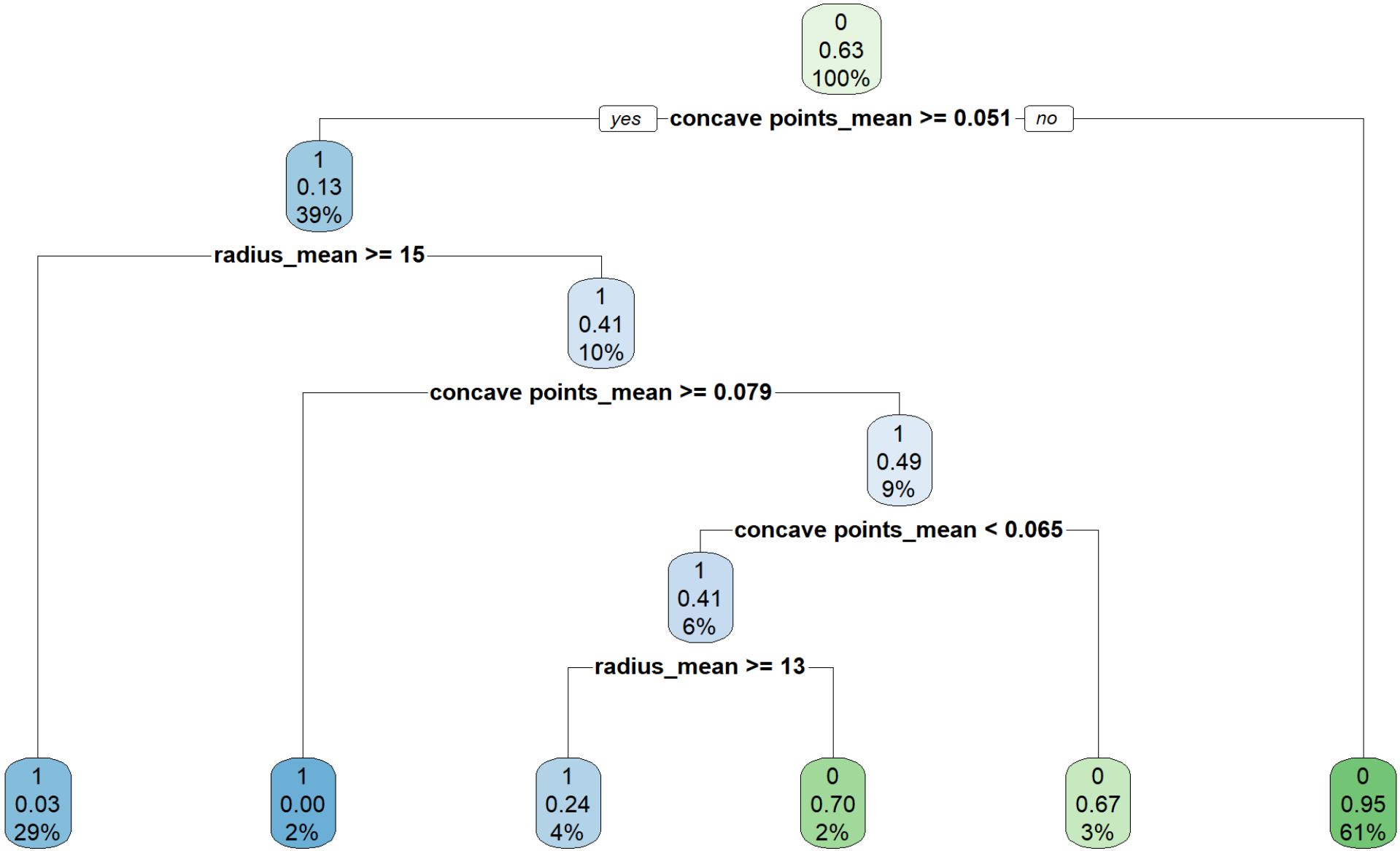
```
# Entropy model
extract_fit_engine(dtc_bc_fit_entropy) %>% rpart.plot(extra = "auto",
                                                      main = "Entropy model")
```

Entropy model



```r
# Gini model
extract_fit_engine(dtc_bc_fit_gini) %>% rpart.plot(extra = "auto",
                                                   main = "Gini model")
```

Gini model


0
0.63
100%
yes
concave points_mean >= 0.051
no
1
0.13
39%
radius_mean >= 15
1
0.41
10%
concave points_mean >= 0.079
1
0.49
9%
concave points_mean < 0.065
1
0.41
6%
radius_mean >= 13
1
0.03
29%
1
0.00
2%
1
0.24
4%
0
0.70
2%
0
0.67
3%
0
0.95
61%

4.2.5 Question 5

```
# Entropy model
metrics(dtc_bc_predictions_entropy, truth = diagnosis, estimate = .pred_class) %>% gt() %>% fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate
accuracy	binary	0.90
kap	binary	0.79

```
# Gini model
metrics(dtc_bc_predictions_gini, truth = diagnosis, estimate = .pred_class) %>% gt() %>% fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate
accuracy	binary	0.90
kap	binary	0.77

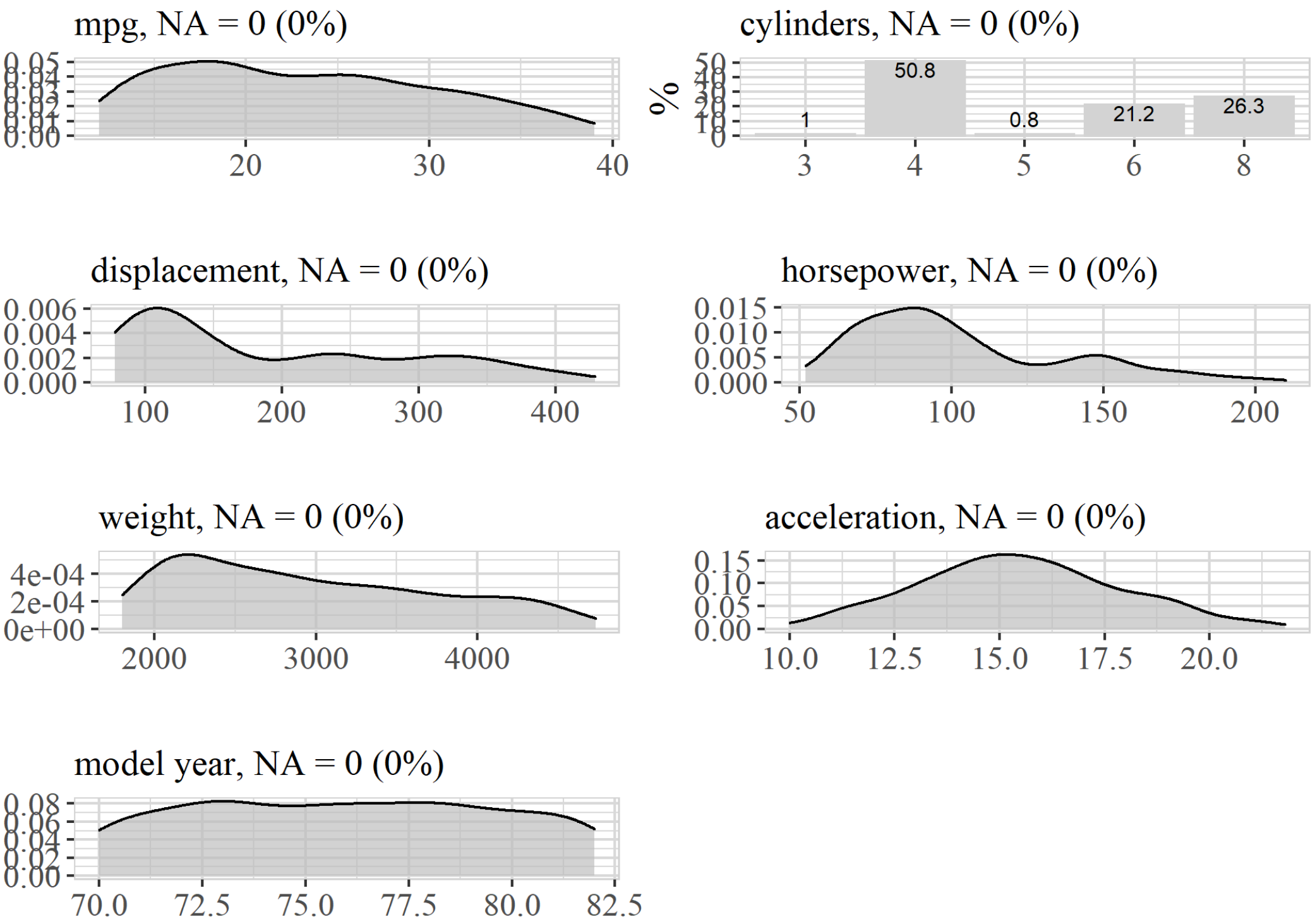
4.3 Exercice 3 : Arbre de régression

4.3.1 Import/préparation données

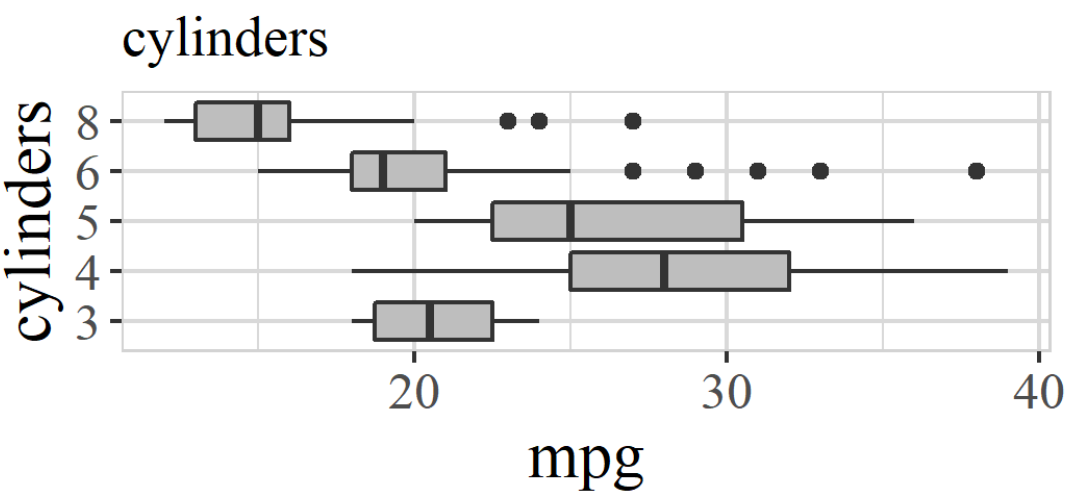
```
cars <- read_csv("dataset/auto-mpg.csv")

cars <- cars %>%
  select(mpg, cylinders, displacement, horsepower, weight, acceleration, `model year`)
cars$mpg <- round(cars$mpg, 0)
cars$horsepower <- as.numeric(cars$horsepower)
cars <- cars %>% na.omit()
```

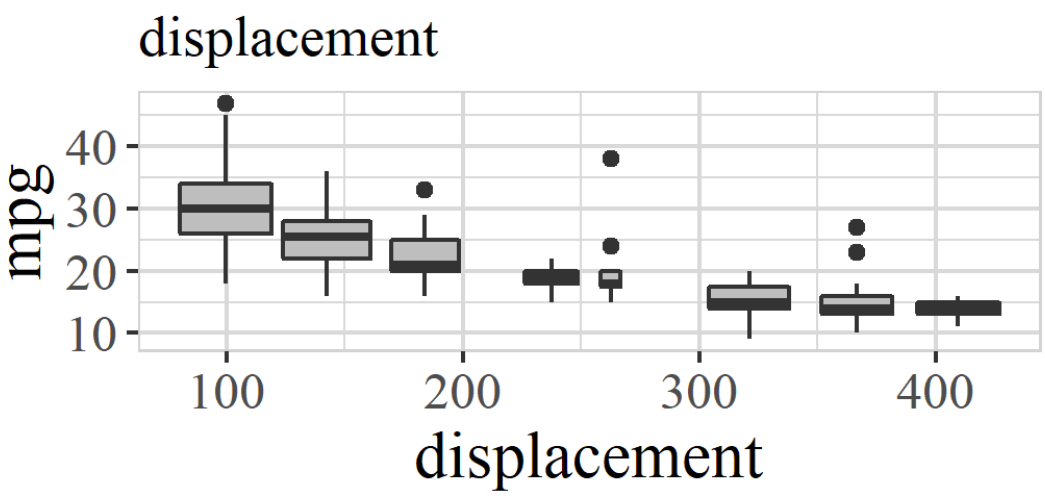
4.3.2 Rapide exploration

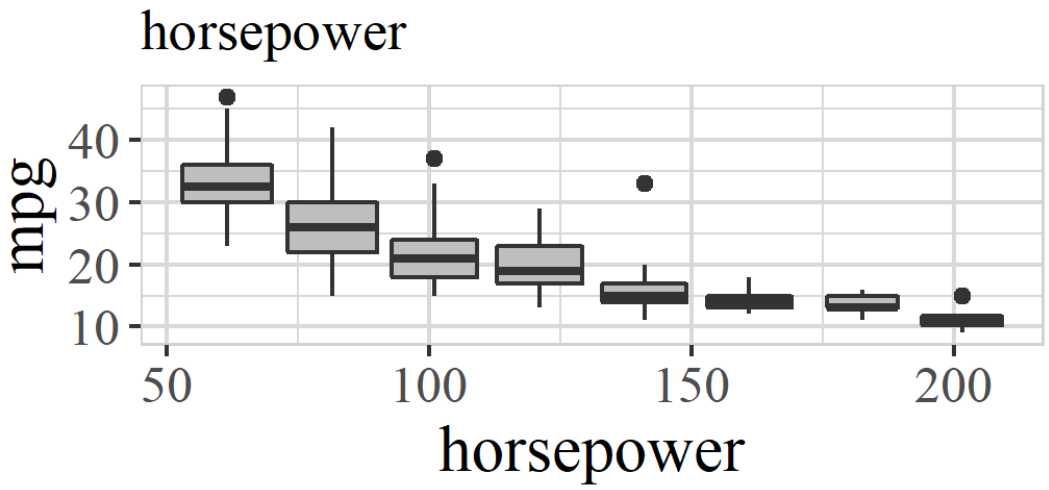
```
explore_all(cars)
```

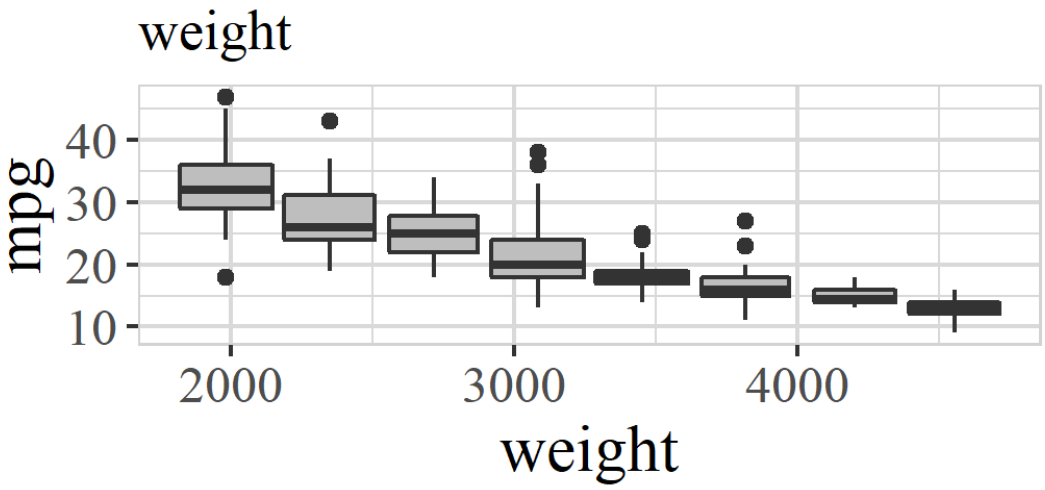

mpg, NA = 0 (0%)
0.05
0.04
0.03
0.02
0.01
0.00
20
30
40
cylinders, NA = 0 (0%)
%
50
40
30
20
10
0
50.8
1
0.8
21.2
26.3
3
4
5
6
8
displacement, NA = 0 (0%)
0.006
0.004
0.002
0.000
100
200
300
400
horsepower, NA = 0 (0%)
0.015
0.010
0.005
0.000
50
100
150
200
weight, NA = 0 (0%)
4e-04
2e-04
0e+00
2000
3000
4000
acceleration, NA = 0 (0%)
0.15
0.10
0.05
0.00
10.0
12.5
15.0
17.5
20.0
model year, NA = 0 (0%)
0.08
0.06
0.04
0.02
0.00
70.0
72.5
75.0
77.5
80.0
82.5

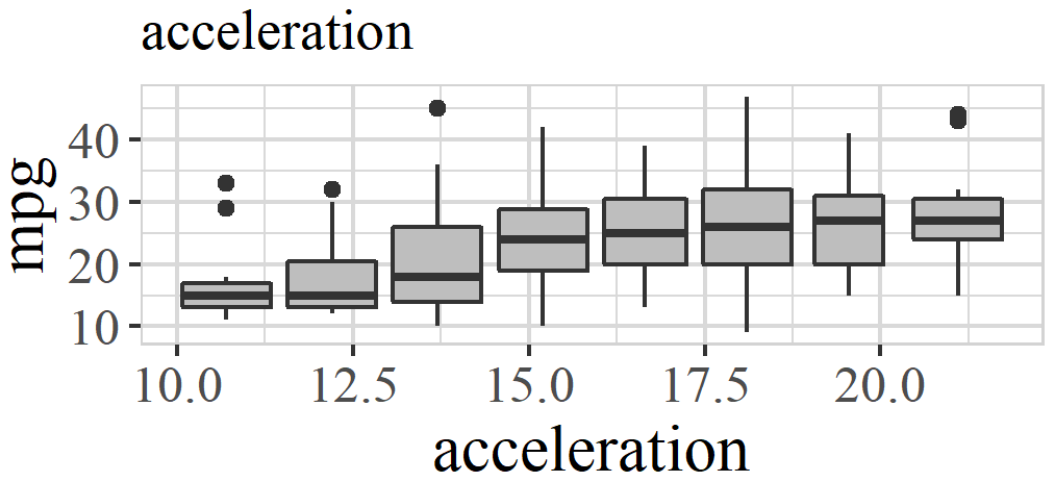
```
explore_all(cars, target = mpg)
```

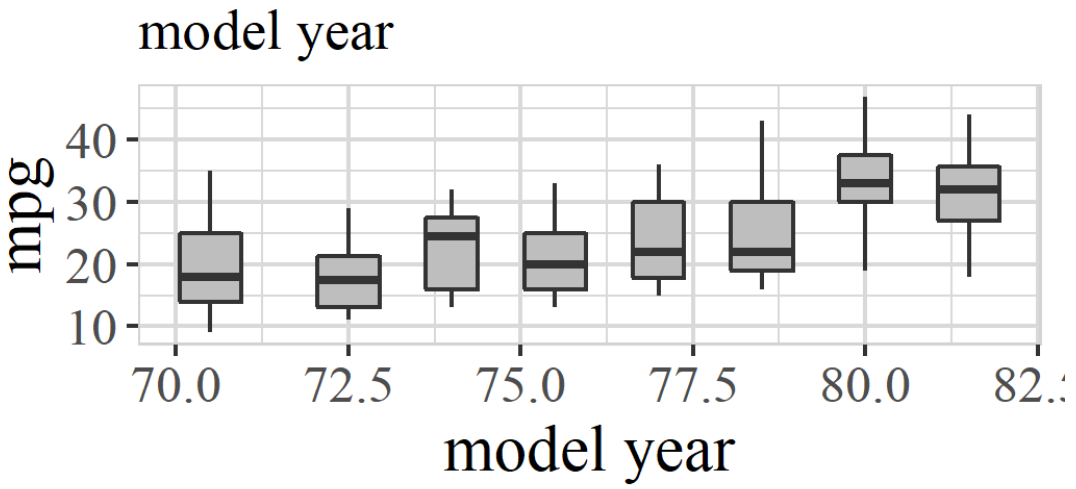

cylinders
cylinders
8
6
5
4
3
20
30
40
mpg


displacement
mpg
40
30
20
10
100
200
300
400
displacement


horsepower
mpg
40
30
20
10
50
100
150
200
horsepower


weight
mpg
40
30
20
10
2000
3000
4000
weight


acceleration
mpg
40
30
20
10
10.0
12.5
15.0
17.5
20.0
acceleration

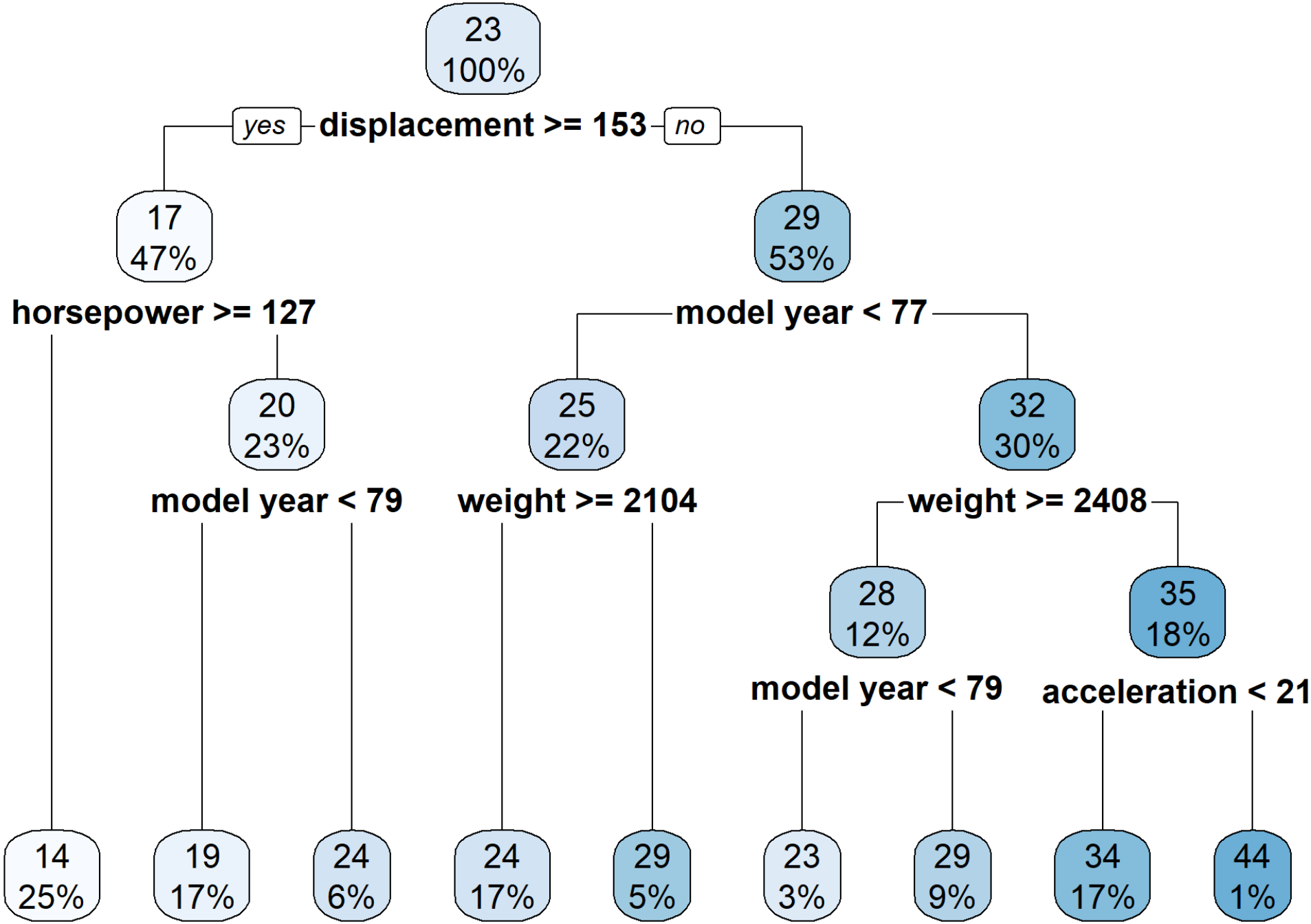

model year
mpg
40
30
20
10
70.0
72.5
75.0
77.5
80.0
model year

4.3.3 Question 1

```
# Split
cars_split <- initial_split(cars, 0.8, strata = mpg)

cars_train <- training(cars_split)
cars_test <- testing(cars_split)

# Model
dtr_cars_mod <- decision_tree(tree_depth = 8, min_n = 13) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("regression")

# Workflow
dtr_cars_wflow <- workflow() %>%
  add_model(dtr_cars_mod) %>%
  add_variables(outcomes = mpg, predictors = everything())

# Fit
dtr_cars_fit <- fit(dtr_cars_wflow, cars_train)
```

4.3.4 Question 2

```
extract_fit_engine(dtr_cars_fit) %>% rpart.plot(extra = "auto")
```


23
100%
yes
displacement >= 153
no
17
47%
29
53%
horsepower >= 127
model year < 77
20
23%
25
22%
32
30%
model year < 79
weight >= 2104
weight >= 2408
28
12%
35
18%
model year < 79
acceleration < 21
14
25%
19
17%
24
6%
24
17%
29
5%
23
3%
29
9%
34
17%
44
1%

4.3.5 Question 3

```
# Predictions
dtr_cars_predictions <- bind_cols(
  select(cars_test, mpg),
  predict(dtr_cars_fit, cars_test)
)
metrics(dtr_cars_predictions, truth = mpg, estimate = .pred) %>% gt() %>% fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate
rmse	standard	3.86
rsq	standard	0.75
mae	standard	2.63

Le MSE est fortement biaisé pour les valeurs plus élevées.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}[(\hat{y}_i - y_i)^2$$

Le RMSE est plus performant lorsqu'il s'agit de grandes valeurs d'erreur.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[(\hat{y}_i - y_i)^2}$$

4.3.6 Question 4

```
cars_lm <- lm(mpg ~ ., cars_train)
cars_pred_lm <- predict(cars_lm, cars_test)
round(modelr::rmse(cars_lm, cars_test), 2)
```

```
## [1] 3.36
```

4.4 Exercice 4 : Biais, variance, erreur de généralisation

4.4.1 Import/préparation données

```
cars <- read_csv("dataset/auto-mpg.csv")

cars <- cars %>%
  select(mpg, cylinders, displacement, horsepower, weight, acceleration, `model year`)
cars$mpg <- round(cars$mpg, 0)
cars$horsepower <- as.numeric(cars$horsepower)
cars <- cars %>% na.omit()
```

4.4.2 Question 1

Le biais est connu comme la différence entre la prédiction des valeurs par le modèle ML et la valeur correcte. Un biais élevé donne une grande erreur dans la formation ainsi que dans les données de test.

La variabilité de la prédiction du modèle pour un point de données donné qui nous indique la propagation de nos données est appelée la variance du modèle.

Le principe de validation croisée est une procédure de ré-échantillonage permettant d'évaluer un modèle même avec des données limitées.

L'erreur se calcule sur le jeu de test.

4.4.3 Question 2

```r
set.seed(1)

# Split
cars_split <- initial_split(cars, 0.7, strata = mpg)

cars_train <- training(cars_split)
cars_test <- testing(cars_split)

# V-fold
cars_train_folds <- vfold_cv(cars_train, v = 10, strata = mpg)
keep_pred <- control_resamples(save_pred = TRUE, save_workflow = TRUE)
multi_metric <- metric_set(rmse, rsq)
```

4.4.4 Question 3

```r
# Model
dtr_cars_mod <- decision_tree(tree_depth = 4, min_n = 26) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("regression")

# Workflow
dtr_cars_wflow <- workflow() %>%
  add_model(dtr_cars_mod) %>%
  add_variables(outcomes = mpg, predictors = everything())

# Fit normal model
dtr_cars_fit <- fit(dtr_cars_wflow, cars_train)

# Fit v-folds model
dtr_cars_fit_folds <- fit_resamples(dtr_cars_wflow, cars_train_folds,
                                    control = keep_pred, metrics = multi_metric
)
```

4.4.5 Question 4

```
collect_metrics(dtr_cars_fit_folds) %>%
  gt() %>% fmt_number(c(mean, std_err), decimals = 2)
```

.metric	.estimator	mean	n	std_err	.config
rmse	standard	3.44	10	0.17	Preprocessor1_Model1
rsq	standard	0.80	10	0.02	Preprocessor1_Model1

4.4.6 Question 5

```
# Normal model
dtr_cars_predictions <- bind_cols(
  select(cars_test, mpg),
  predict(dtr_cars_fit, cars_test)
)

metrics(dtr_cars_predictions, truth = mpg, estimate = .pred) %>% gt() %>%
  fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate
rmse	standard	3.77
rsq	standard	0.79
mae	standard	2.73

```
# V-fold model
dtr_cars_fit_folds_predictions <- collect_predictions(dtr_cars_fit_folds)

metrics(dtr_cars_fit_folds_predictions, truth = mpg, estimate = .pred) %>% gt() %>% fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate
rmse	standard	3.48
rsq	standard	0.79
mae	standard	2.61

4.4.7 Question 6

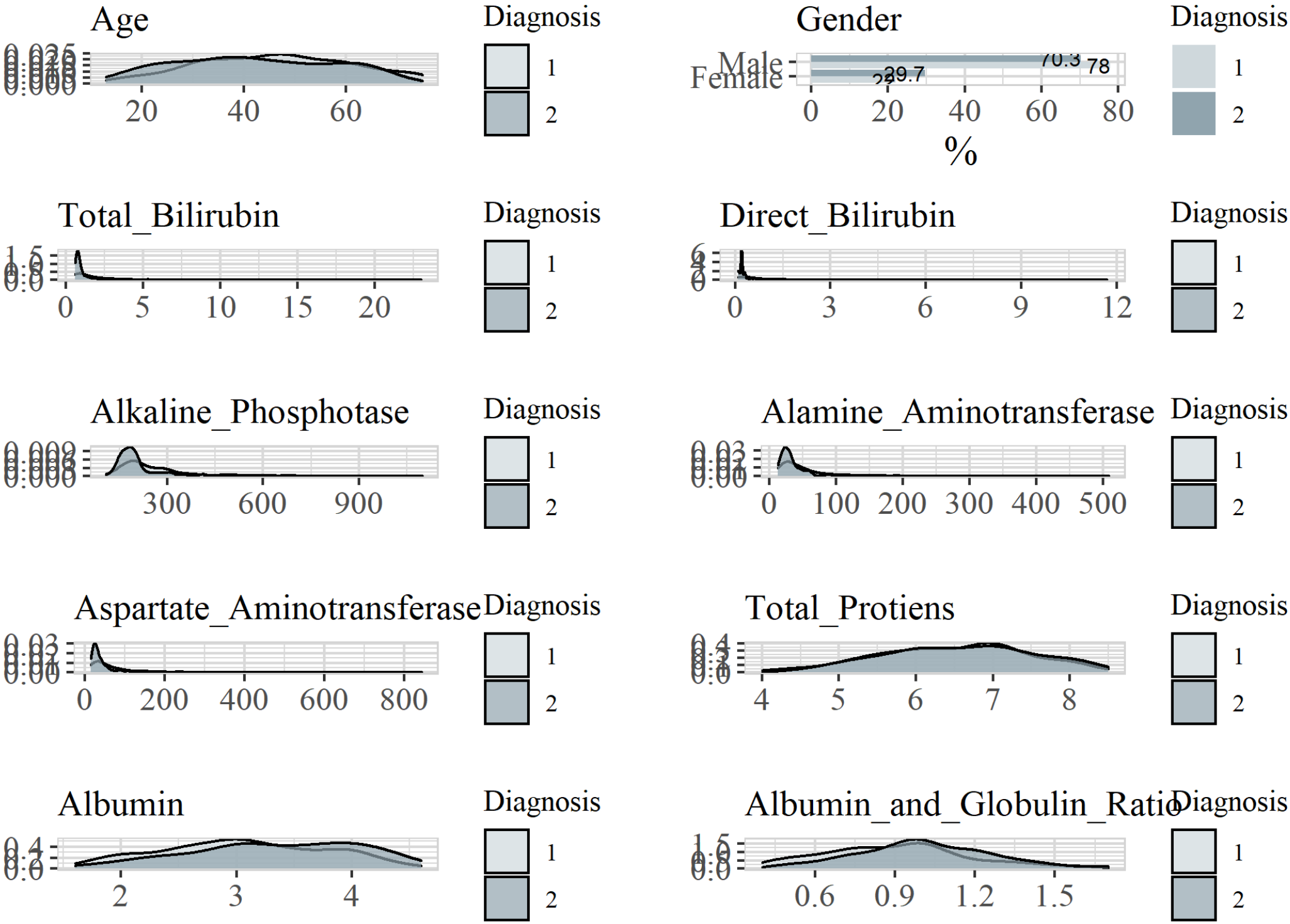
4.5 Exercice 5 : Bagging

4.5.1 Import/préparation données

```
liver <- read_csv("dataset/indian_liver_patient.csv")
liver$Gender <- as_factor(liver$Gender)
liver$Dataset <- as_factor(liver$Dataset)
liver <- liver %>% na.omit()
liver <- liver %>% rename(Diagnosis = Dataset)
```

4.5.2 Rapide exploration

```
explore_all(liver, target = Diagnosis)
```


Age
Diagnosis
1
2
20
40
60
Gender
Male
Female
70.3
78
29.7
0
20
40
60
80
%
Diagnosis
1
2
Total_Bilirubin
Diagnosis
1
2
0
5
10
15
20
Direct_Bilirubin
Diagnosis
1
2
0
3
6
9
12
Alkaline_Phosphotase
Diagnosis
1
2
300
600
900
Alamine_Aminotransferase
Diagnosis
1
2
0
100
200
300
400
500
Aspartate_Aminotransferase
Diagnosis
1
2
0
200
400
600
800
Total_Protiens
Diagnosis
1
2
4
5
6
7
8
Albumin
Diagnosis
1
2
2
3
4
Albumin_and_Globulin_Ratio
Diagnosis
1
2
0.6
0.9
1.2
1.5

4.5.3 Question 1

$$Bagging = Bootstrap + aggregating$$

Le bootstrap est une technique de ré-échantillonnage qui consiste à tirer de façon répétée des échantillons des données sources avec remplacement. Par remplacement, nous entendons que le même point de données peut être inclus plusieurs fois dans notre ensemble de données ré-échantillonné.

Un des atouts de cette méthode est de pouvoir **réduire la variance**, en effet même si les modèles ne sont pas entraînés sur le même jeu de donnée, les échantillons de bootstrap partagent des tuples en commun, ce qui produit du **biais**. Ce biais produit fait réduire la variance.

Un deuxième atout est de corriger les erreurs de prédictions. Chaque modèle fait une erreur de prédictions de classification, mais chaque erreur est corrigée par le système de vote.

4.5.4 Question 2

```
liver_split <- initial_split(liver, 0.70, strata = Diagnosis)

liver_train <- training(liver_split)
liver_test <- testing(liver_split)

dtc_liver_folds <- vfold_cv(liver_train)
```

4.5.5 Question 3

```
# Modele simple
rfc_liver_mod <- rand_forest(trees = 1000) %>%
  set_engine("ranger", importance = "impurity", num.threads = cores) %>%
  set_mode("classification")
```

```r
# Worflow
rfc_liver_wf <- workflow() %>%
  add_model(rfc_liver_mod) %>%
  add_variables(outcomes = Diagnosis, predictors = everything())


# Fit
rfc_liver_fit <- fit(rfc_liver_wf, liver_train)


# Pred/metrics
rfc_liver_pred <- bind_cols(
  select(liver_test, Diagnosis),
  predict(rfc_liver_fit, liver_test)
)

my_metrics <- metric_set(accuracy)

rf_acc <- my_metrics(rfc_liver_pred, truth = Diagnosis, estimate = .pred_class)

rfc_liver_efit <- extract_fit_engine(rfc_liver_fit)

# Bagging
liver_bag <- ipred::bagging(Diagnosis ~ ., liver_train, coob = T, nbagg = 100)

liver_bag_pred <- predict(liver_bag, liver_test)

liver_bag_pred <- data.frame(original = liver_test$Diagnosis, predicted = liver_bag_pred)

bag_acc <- accuracy(liver_bag_pred, truth = liver_test$Diagnosis, estimate = predicted)
```

4.5.6 Question 4

blabla oob accuracy

```r
# Modele simple
rfc_liver_efit$prediction.error
```

```
## [1] 0.172018
```

```
# Bagging
liver_bag$err
```

```
## [1] 0.279703
```

4.5.7 Question 5

```
tab_score <- tibble(
  .model = c("RandomForest", "Bagging", "RandomForest", "Bagging"),
  .metric = c("Accuracy", "Accuracy", "OOB", "OOB"),
  .score = c(rf_acc$.estimate, bag_acc$.estimate, rfc_liver_efit$prediction.error, liver_bag$err)
)

tab_score %>% gt() %>% fmt_number(.score, decimals = 2)
```

.model	.metric	.score
RandomForest	Accuracy	0.68
Bagging	Accuracy	0.71
RandomForest	OOB	0.17
Bagging	OOB	0.28

4.6 Exercice 6 : Forêt aléatoire

4.6.1 Import/préparation données

```r
bike_train <- read_csv("dataset/train_bikeshare.csv")

bike_train <- bike_train %>% select(-c(casual, registered))

bike_train$datetime <- lubridate::as_datetime(bike_train$datetime)
bike_train$day_week <- lubridate::wday(bike_train$datetime, label = TRUE)
bike_train$month <- lubridate::month(bike_train$datetime, label = TRUE)
bike_train$hour <- lubridate::hour(bike_train$datetime)
```

4.6.2 Question 1

Les forêts aléatoires ne souffrent pas de sur-apprentissage, ont de meilleure performance que les arbres de décision. La parallélisation possible.

Les forêts aléatoires consistent à faire tourner en parallèle un grand nombre (plusieurs centaines) d'arbres de décisions construits aléatoirement, avant de les moyenner.

En termes statistiques, si les arbres sont décorrélés, cela permet de réduire la variance des prévisions.

4.6.3 Question 2

```r
bike_split <- initial_split(bike_train, 0.8, strata = count)
bike_train <- training(bike_split)
bike_test <- testing(bike_split)

bike_folds <- vfold_cv(bike_train, v = 10, strata = count)

get_model <- function(x) {
  x %>%
    extract_fit_parsnip() %>%
    vip::vi()
}

ctrl_rsp <- control_resamples(save_pred = TRUE, save_workflow = TRUE, extract = get_model)
```

```
my_metrics <- metric_set(rmse)


# Recipe
bike_rec <- recipe(count ~ ., bike_train)

# Engine/mode
rfr_bike_model <- rand_forest(trees = 1000) %>%
  set_engine("ranger", importance = "impurity", seed = 1, num.threads = cores) %>%
  set_mode("regression")

# workflow
rfr_bike_wflow <- workflow() %>%
  add_model(rfr_bike_model) %>%
  add_recipe(bike_rec)

# fit
rfr_bike_fit <- fit_resamples(rfr_bike_wflow,
                          bike_folds,
                          control = ctrl_rsp,
                          metrics = my_metrics)

rfr_bike_metric_sum_on <- rfr_bike_fit %>%
  collect_metrics()
```

4.6.4 Question 3

```
# Engine/mode
lr_bike_model <- linear_reg() %>%
  set_engine("glm")

# workflow
lr_bike_wflow <- workflow() %>%
  add_model(lr_bike_model) %>%
  add_recipe(bike_rec)

# fit
```

```
lr_bike_fit <- fit_resamples(lr_bike_wflow,
                         bike_folds,
                         control = ctrl_rsp,
                         metrics = my_metrics)

lr_bike_metric_sum_on <- collect_metrics(lr_bike_fit)
```

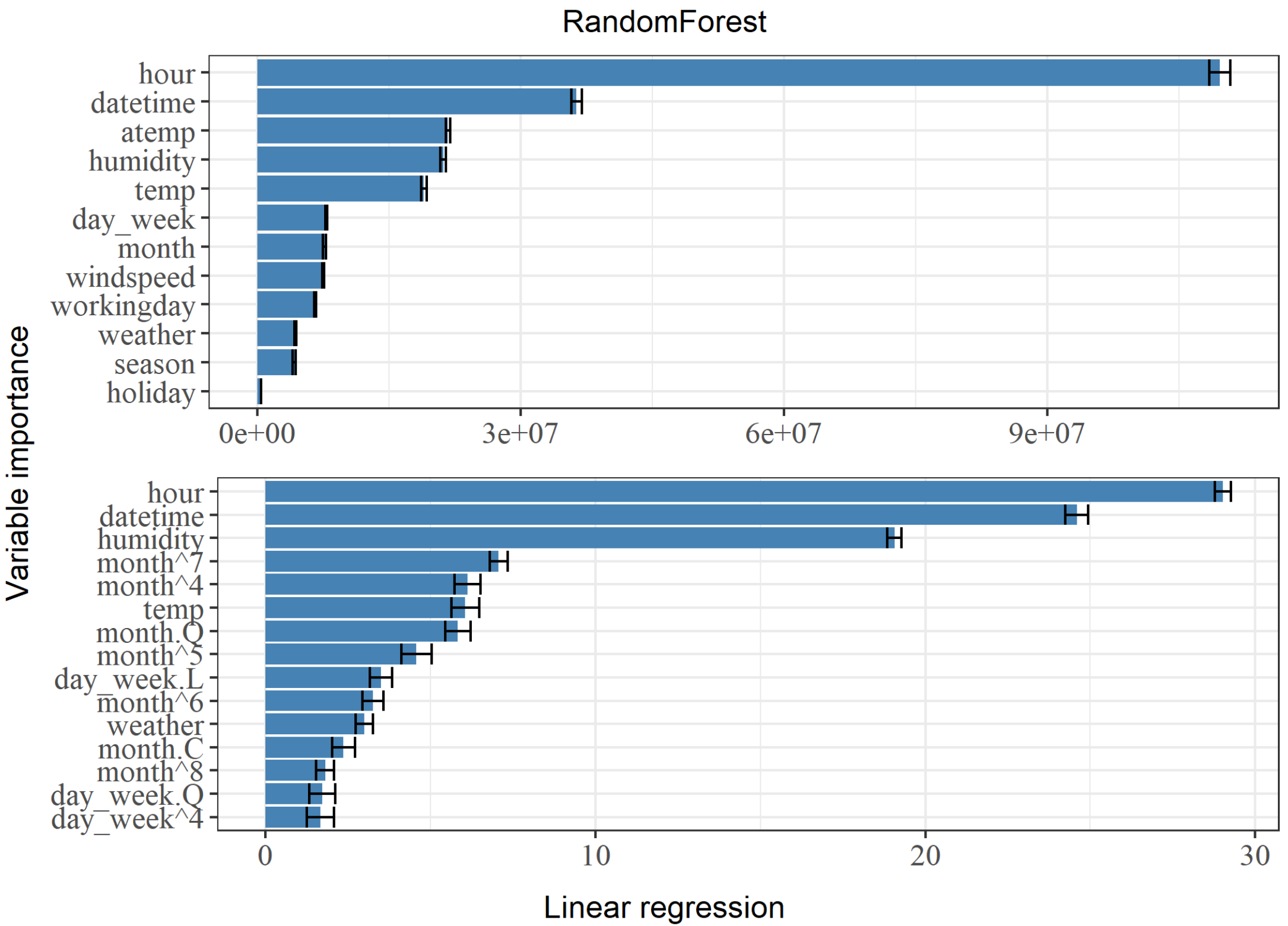
```
tibble(
  model = c("RandomForest", "Linear regression"),
  metric = "rmse",
  score = c(rfr_bike_metric_sum_on$mean, lr_bike_metric_sum_on$mean)
) %>% gt() %>% fmt_number(score, decimals = 2)
```

model	metric	score
RandomForest	rmse	64.68
Linear regression	rmse	139.95

4.6.5 Question 4

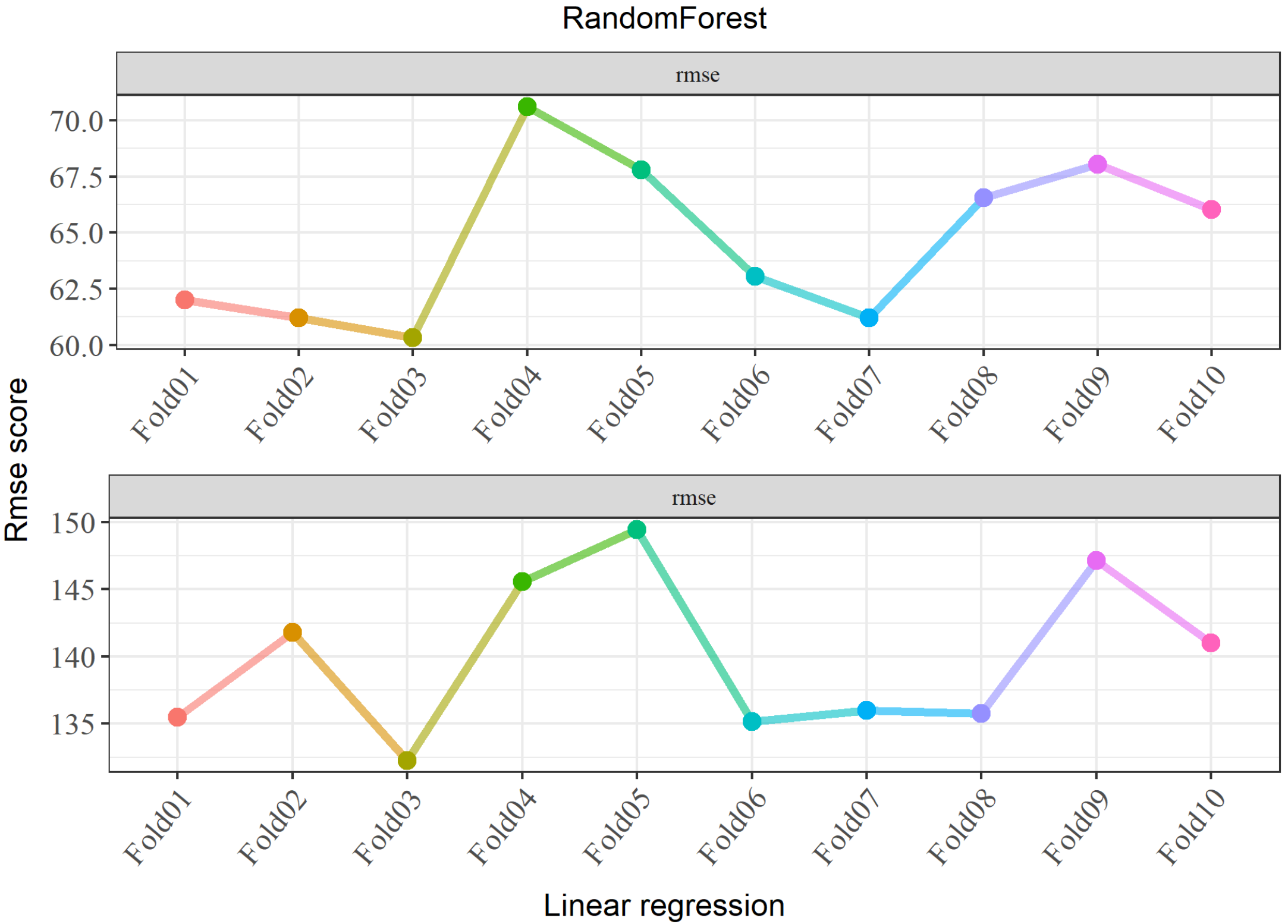
```
# Random forest model
rfr_bike_varimp <- rfr_bike_fit %>%
  select(id, .extracts) %>%
  unnest(.extracts) %>%
  unnest(.extracts) %>%
  group_by(Variable) %>%
  summarise(Mean = mean(Importance),
            Variance = sd(Importance)) %>%
  slice_max(Mean, n = 15) %>%
  ggplot(aes(Mean, reorder(Variable, Mean))) +
  geom_col(fill = "steelblue") +
  geom_errorbar(aes(xmin = Mean - Variance, xmax = Mean + Variance)) +
  labs(y = NULL,
       x = NULL)
```

```r
# Linear regression model
lr_bike_varimp <- lr_bike_fit %>%
  select(id, .extracts) %>%
  unnest(.extracts) %>%
  unnest(.extracts) %>%
  group_by(Variable) %>%
  summarise(Mean = mean(Importance),
            Variance = sd(Importance)) %>%
  slice_max(Mean, n = 15) %>%
  ggplot(aes(Mean, reorder(Variable, Mean))) +
  geom_col(fill = "steelblue") +
  geom_errorbar(aes(xmin = Mean - Variance, xmax = Mean + Variance)) +
  labs(y = NULL,
       x = NULL)

figure_varimp <- ggpubr::ggarrange(rfr_bike_varimp, lr_bike_varimp,
          ncol = 1, nrow = 2)
ggpubr::annotate_figure(figure_varimp,
                top = ggpubr::text_grob("RandomForest"),
                bottom = ggpubr::text_grob("Linear regression"),
                left = ggpubr::text_grob("Variable importance", rot = 90)
)
```


RandomForest
hour
datetime
atemp
humidity
temp
day_week
month
windspeed
workingday
weather
season
holiday
0e+00
3e+07
6e+07
9e+07
hour
datetime
humidity
month^7
month^4
temp
month.Q
month^5
day_week.L
month^6
weather
month.C
month^8
day_week.Q
day_week^4
0
10
20
30
Variable importance
Linear regression

4.6.6 Question 5

```
# Random forest model
rfr_bike_lc <- rfr_bike_fit %>%
  collect_metrics(summarize = FALSE) %>%
  ggplot(aes(id, .estimate, color = id, group = 1)) +
  geom_line(size = 1.5, alpha = 0.6) +
  geom_point(size = 3) +
  facet_wrap(~.metric, scales = "free") +
  labs(y = NULL,
       x = NULL) +
  theme(axis.text.x = element_text(angle = 50, vjust = 1, hjust = 1),
        legend.position = "none")

# Linear regression model
lr_bike_lc <- lr_bike_fit %>%
  collect_metrics(summarize = FALSE) %>%
  ggplot(aes(id, .estimate, color = id, group = 1)) +
  geom_line(size = 1.5, alpha = 0.6) +
  geom_point(size = 3) +
  facet_wrap(~.metric, scales = "free") +
  labs(y = NULL,
       x = NULL) +
  theme(axis.text.x = element_text(angle = 50, vjust = 1, hjust = 1),
        legend.position = "none")


figure_lc <- ggpubr::ggarrange(rfr_bike_lc, lr_bike_lc,
          ncol = 1, nrow = 2)
ggpubr::annotate_figure(figure_lc,
                top = ggpubr::text_grob("RandomForest"),
                bottom = ggpubr::text_grob("Linear regression"),
                left = ggpubr::text_grob("Rmse score", rot = 90)
)
```


RandomForest
rmse
70.0
67.5
65.0
62.5
60.0
Fold01
Fold02
Fold03
Fold04
Fold05
Fold06
Fold07
Fold08
Fold09
Fold10
rmse
150
145
140
135
Fold01
Fold02
Fold03
Fold04
Fold05
Fold06
Fold07
Fold08
Fold09
Fold10
Rmse score
Linear regression

4.7 Exercice 7 : Hyperparamètres et grid search, une introduction

4.7.1 Import/préparation données

```
liver <- read_csv("dataset/indian_liver_patient.csv")

liver$Gender <- as_factor(liver$Gender)
liver$Dataset <- as_factor(liver$Dataset)

liver <- liver %>% na.omit()
liver <- liver %>% rename(Diagnosis = Dataset)



# Ssplit
set.seed(1)
liver_split <- initial_split(liver, 0.70, strata = Diagnosis)

liver_train <- training(liver_split)
liver_test <- testing(liver_split)

set.seed(1)
liver_folds <- vfold_cv(liver_train, v = 5, strata = Diagnosis)
```

4.7.2 Question 1

```
# Recipe
liver_rec <- recipe(Diagnosis ~ ., liver_train)

# Engine/mode
liver_model <- decision_tree(
  min_n = tune(),
  tree_depth = tune()
) %>%
  set_engine("rpart", model = TRUE) %>%
  set_mode("classification")
```

```r
# Workflow
liver_wflow <- workflow() %>%
  add_recipe(liver_rec) %>%
  add_model(liver_model)
```

4.7.3 Question 2

```r
# Tree grid
liver_grid <- grid_regular(
  range_set(tree_depth(), c(2, 4)),
  range_set(min_n(), c(12, 18)),
  levels = 5
)
```

4.7.4 Question 3

AUC signifie “Area under the ROC Curve”” (aire sous la courbe ROC). C’est-à-dire que l’AUC mesure toute l’aire à deux dimensions sous l’intégralité de la courbe ROC.

Un modèle ayant un AUC égale à 0.50 est considérer comme inutile, au contraire un modèle ayant un AUC au voisinage de 1 est considéré comme extrèment performant.

La sensibilité est une mesure de la capacité d’un modèle d’apprentissage automatique à détecter des instances positives. Elle est également connue sous le nom de taux de vrais positifs (True Positive Rate, TPR) .

$$Sensibilité = VP/(VP + FN)$$

La sensibilité est utilisée pour évaluer les performances du modèle, elle est souvent comparée à la spécificité. La spécificité mesure la proportion de vrais négatifs qui sont correctement identifiés par le modèle.

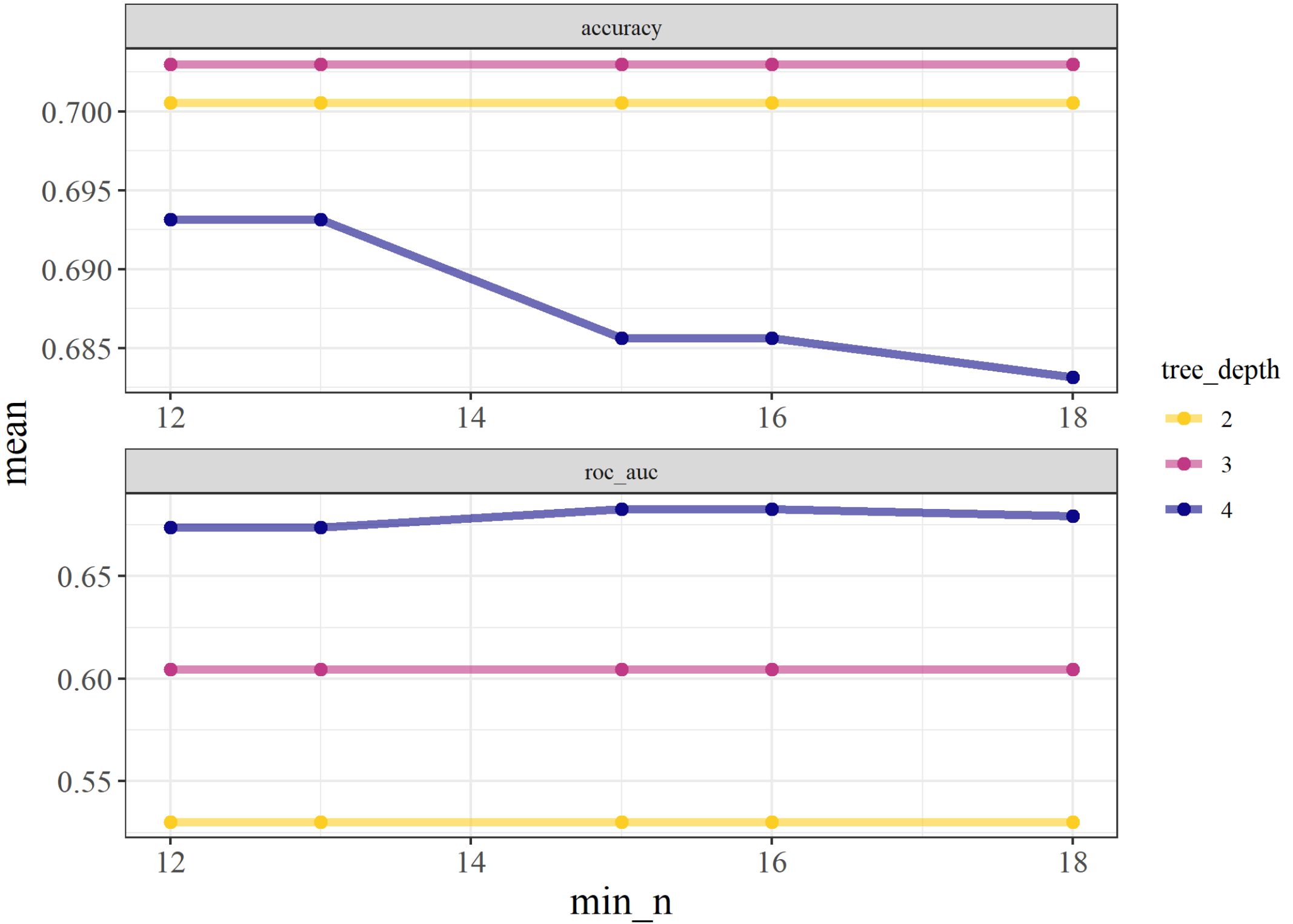
$$Specificity = (VN)/(FP + VN)$$

4.7.5 Question 4

```
# Tune grid
set.seed(1)
liver_res <-
  liver_wflow %>%
  tune_grid(
    resamples = liver_folds,
    grid = liver_grid
  )
```

4.7.6 Question 5

```
# Graph metrics
liver_res %>%
  collect_metrics() %>%
  mutate(tree_depth = factor(tree_depth)) %>%
  ggplot(aes(min_n, mean, color = tree_depth)) +
  geom_line(size = 1.5, alpha = 0.6) +
  geom_point(size = 2) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_color_viridis_d(option = "plasma", begin = .9, end = 0)
```


accuracy
roc_auc
mean
min_n
0.700
0.695
0.690
0.685
0.65
0.60
0.55
12
14
16
18
tree_depth
2
3
4

```r
# Best model
liver_best <- liver_res %>%
  select_best("roc_auc")
```

4.7.7 Question 6

4.7.8 Question 7

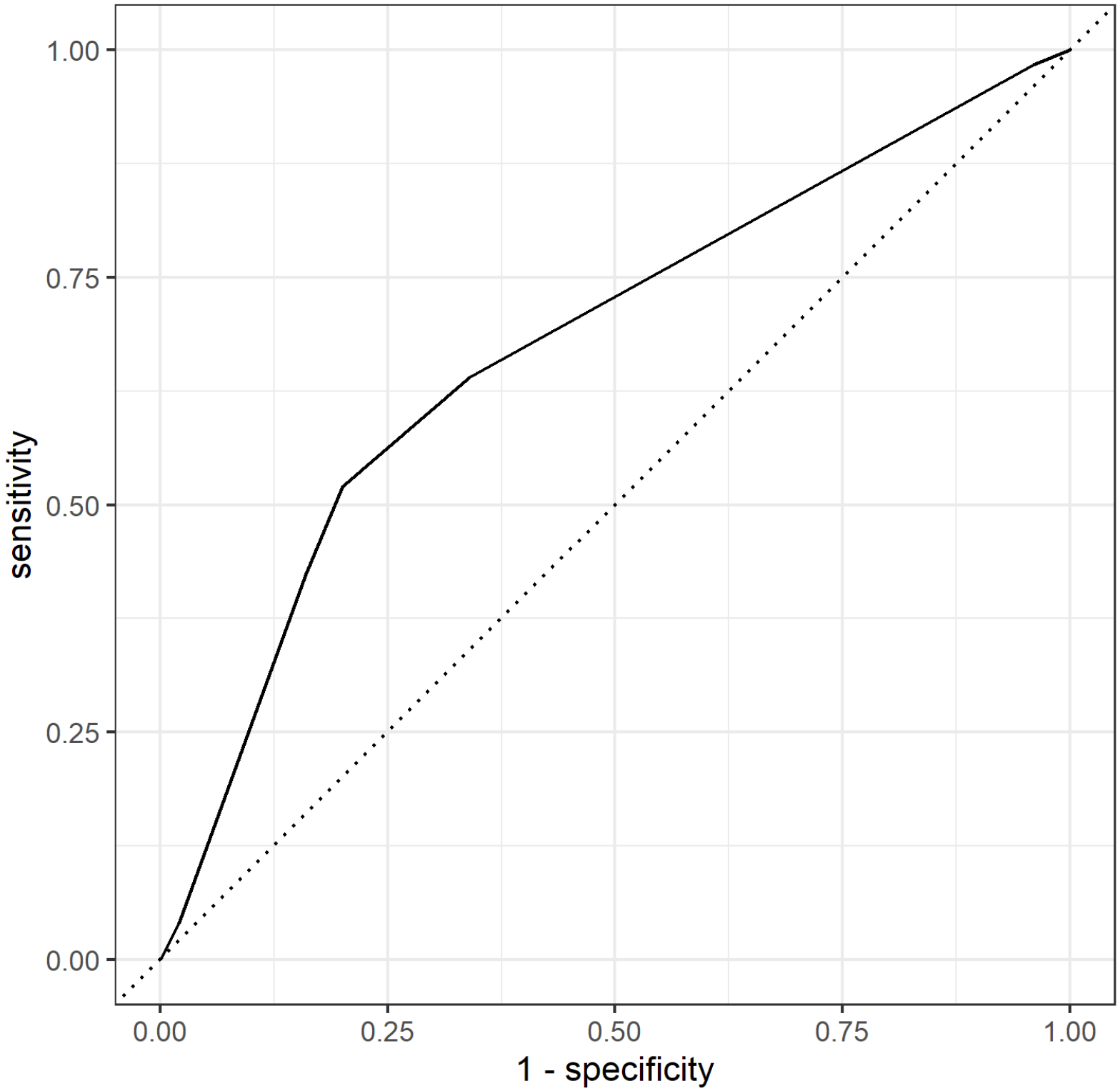
4.7.9 Question 8

```r
# Custom metrics
my_metrics <- metric_set(accuracy, precision, sens, recall, roc_auc)

# Finalize workflow
final_liver_wflow <-
  liver_wflow %>%
  finalize_workflow(liver_best)

## Last fit
liver_final_fit <-
  final_liver_wflow %>%
  last_fit(liver_split,
           metrics = my_metrics)

# Collect metrics
liver_final_fit %>%
  collect_metrics() %>%
  gt() %>% fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate	.config
accuracy	binary	0.65	Preprocessor1_Model1
precision	binary	0.82	Preprocessor1_Model1

sens	binary	0.64	Preprocessor1_Model1
recall	binary	0.64	Preprocessor1_Model1
roc_auc	binary	0.68	Preprocessor1_Model1

```
# ROC curve
liver_final_fit %>%
  collect_predictions() %>%
  roc_curve(Diagnosis, .pred_1) %>%
  autoplot()
```


1.00
0.75
0.50
0.25
0.00
sensitivity
0.00
0.25
0.50
0.75
1.00
1 - specificity

```r
# Predictions
liver_final_fit_pred <-  liver_final_fit %>% collect_predictions()

# Confusion matrix
caret::confusionMatrix(liver_final_fit_pred$.pred_class, liver_test$Diagnosis, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 80 17
##          2 45 33
##
##                Accuracy : 0.6457
##                  95% CI : (0.57, 0.7164)
##     No Information Rate : 0.7143
##     P-Value [Acc > NIR] : 0.9801540
##
##                   Kappa : 0.2568
##
##  Mcnemar's Test P-Value : 0.0006058
##
##             Sensitivity : 0.6400
##             Specificity : 0.6600
##          Pos Pred Value : 0.8247
##          Neg Pred Value : 0.4231
##               Precision : 0.8247
##                  Recall : 0.6400
##                      F1 : 0.7207
##              Prevalence : 0.7143
##          Detection Rate : 0.4571
##    Detection Prevalence : 0.5543
##       Balanced Accuracy : 0.6500
##
##        'Positive' Class : 1
##
```

4.7.10 Question 9

Taux de pourcentage des positifs prédits sur le total des positifs. C'est la même chose que le TPR (taux de vrais positifs).

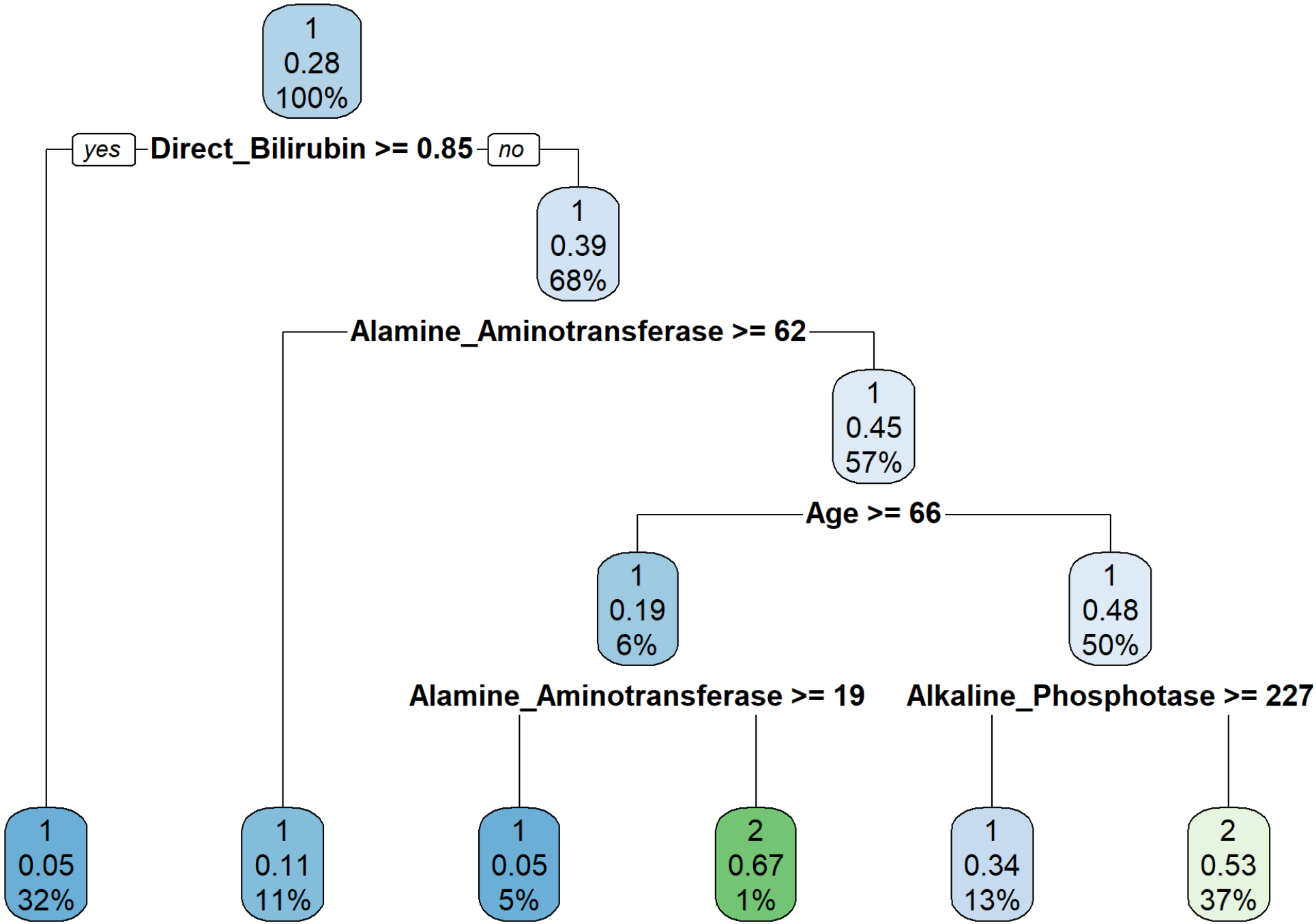
$$Rappel = VP/(VP + FN)$$

Pourcentage réellement positif sur l'ensemble des prédictions positives. La valeur de précision est comprise entre 0 et 1.

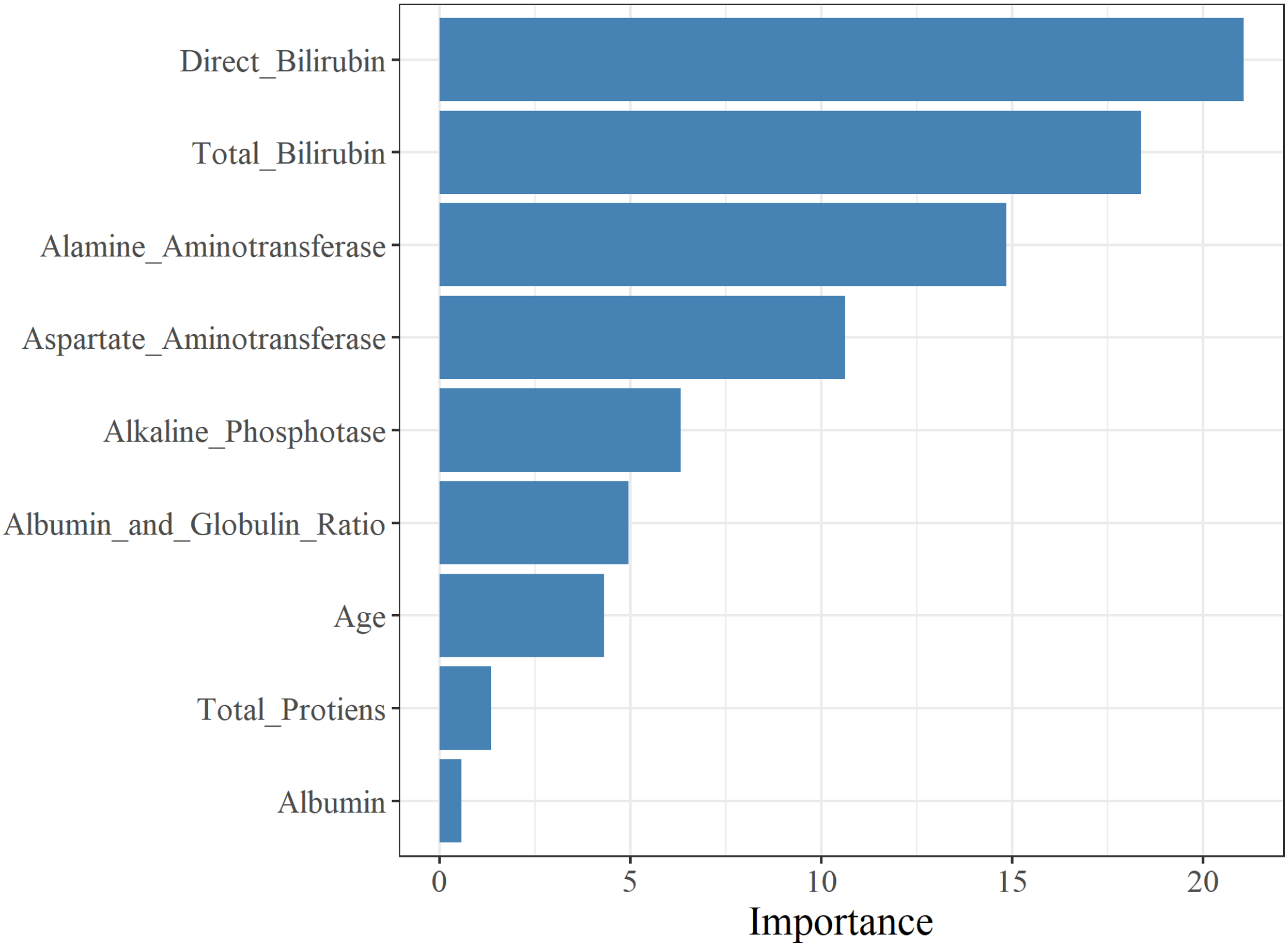
$$Precision = VP/(VP + FP)$$

4.7.11 Extra

```
### tree
liver_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint = FALSE)
```


1
0.28
100%
yes
Direct_Bilirubin >= 0.85
no
1
0.39
68%
Alamine_Aminotransferase >= 62
1
0.45
57%
Age >= 66
1
0.19
6%
1
0.48
50%
Alamine_Aminotransferase >= 19
Alkaline_Phosphotase >= 227
1
0.05
32%
1
0.11
11%
1
0.05
5%
2
0.67
1%
1
0.34
13%
2
0.53
37%

```r
### Most important feature
liver_final_fit %>%
  extract_fit_parsnip() %>%
  vip::vip(aesthetics = list(fill = "steelblue", size = 0.8))
```


Direct_Bilirubin
Total_Bilirubin
Alamine_Aminotransferase
Aspartate_Aminotransferase
Alkaline_Phosphotase
Albumin_and_Globulin_Ratio
Age
Total_Protiens
Albumin
0
5
10
15
20
Importance

4.8 Exercice 8 : Grid search en autonomie

4.8.1 Import/préparation données

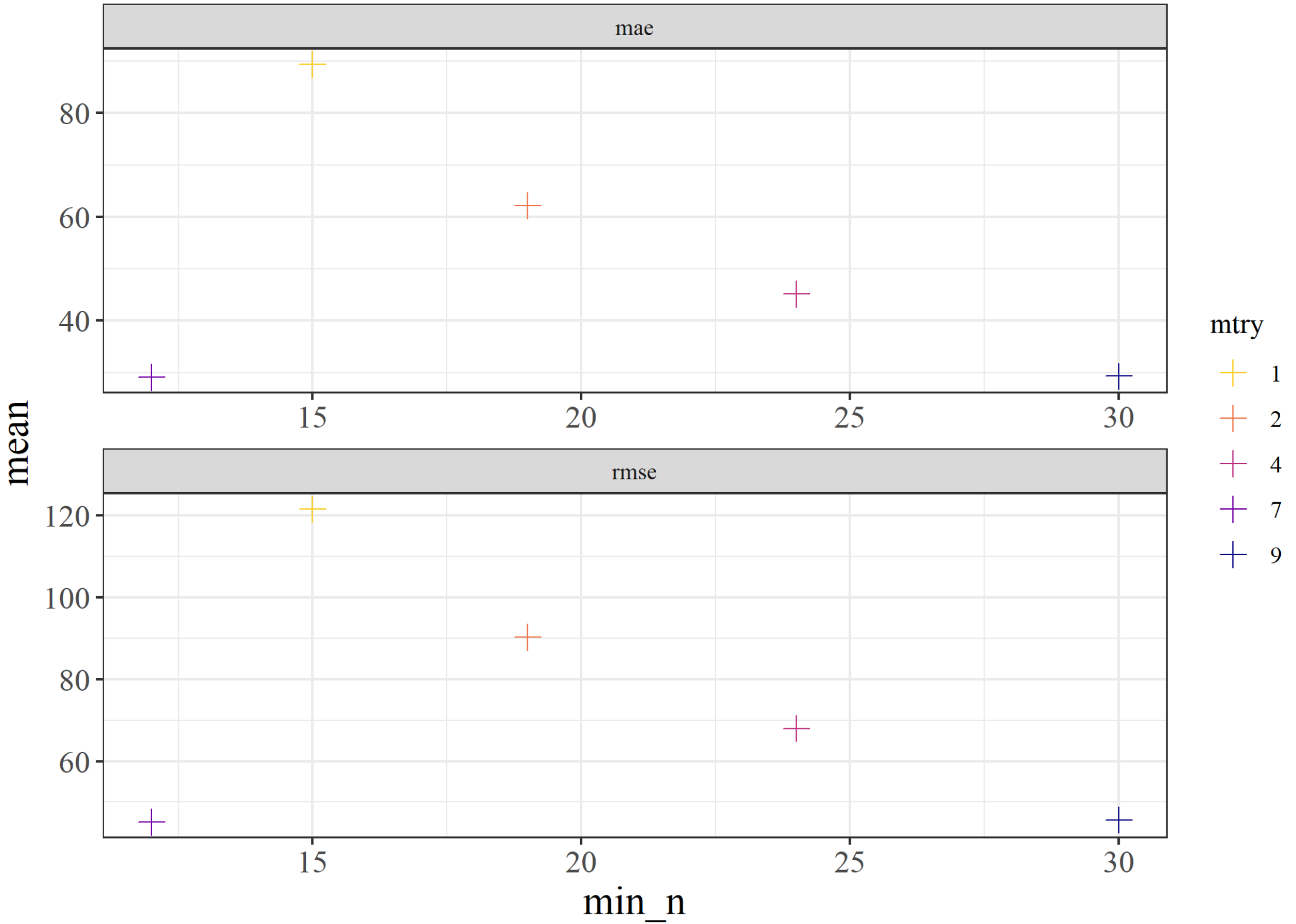
```
bike_train <- read_csv("dataset/train_bikeshare.csv")

bike_train <- bike_train %>% select(-c(casual, registered))

bike_train$datetime <- lubridate::as_datetime(bike_train$datetime)
bike_train$day_week <- lubridate::wday(bike_train$datetime, label = TRUE)
bike_train$month <- lubridate::month(bike_train$datetime, label = TRUE)
bike_train$hour <- lubridate::hour(bike_train$datetime)

# Split
set.seed(1)
bike_split <- initial_split(bike_train, 0.8, strata = count)
bike_train <- training(bike_split)
bike_test <- testing(bike_split)

# V-folds
set.seed(1)
bike_folds <- vfold_cv(bike_train, v = 3, strata = count)

# Fonction pour vi
get_model <- function(x) {
  x %>%
    extract_fit_parsnip() %>%
    vip::vi()
}

# Option pour controle resamples
ctrl_rsp <- control_resamples(save_pred = TRUE, save_workflow = TRUE, extract = get_model)

# Metric de choix
my_metrics <- metric_set(rmse, mae)
```

```r
# Recipe
bike_rec <-
  recipe(formula = count ~ ., data = bike_train)

# Model
bike_mod <-
  rand_forest(mtry = tune(), min_n = tune(), trees = tune()) %>%
  set_mode("regression") %>%
  set_engine("ranger", num.threads = cores, importance = "impurity")

# Workflow
bike_wflow <-
  workflow() %>%
  add_recipe(bike_rec) %>%
  add_model(bike_mod)
```

```r
# Création grid
set.seed(1)
bike_grid <- grid_random(
  range_set(mtry(), c(1, 12)),
  range_set(min_n(), c(2, 30)),
  range_set(trees(), c(500, 1000)),
  size = 5
)

# Grid tuning
set.seed(1)
bike_tune <-
  tune_grid(bike_wflow,
            resamples = bike_folds,
            grid = bike_grid,
            metrics = my_metrics)
```

```r
bike_tune %>%
  collect_metrics() %>%
  mutate(mtry = factor(mtry)) %>%
```

```
ggplot(aes(min_n, mean, color = mtry)) +
geom_point(size = 3, shape = 3) +
facet_wrap(~.metric, scales = "free", nrow = 2) +
scale_color_viridis_d(option = "plasma", begin = .9, end = 0)
```

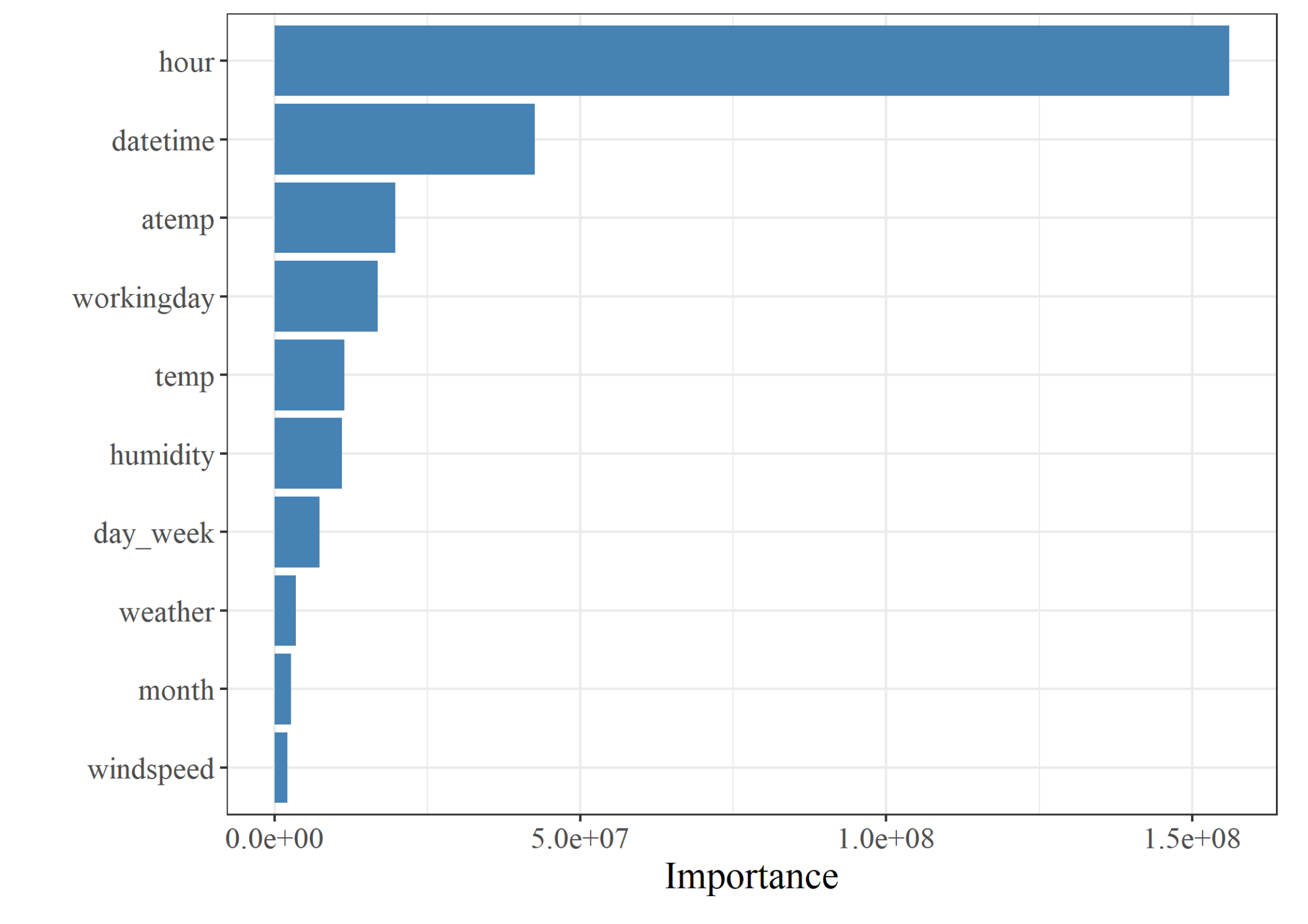

mae
rmse
mean
min_n
mtry
1
2
4
7
9
80
60
40
120
100
80
60
15
20
25
30

```r
# Best model
bike_best <- bike_tune %>%
  select_best("rmse")

# Finalize workflow
final_bike_wflow <-
  bike_wflow %>%
  finalize_workflow(bike_best)

## Last fit
bike_final_fit <-
  final_bike_wflow %>%
  last_fit(bike_split,
           metrics = my_metrics)
```

```r
bike_final_fit %>%
  collect_metrics() %>%
  gt() %>% fmt_number(.estimate, decimals = 2)
```

.metric	.estimator	.estimate	.config
rmse	standard	43.37	Preprocessor1_Model1
mae	standard	27.36	Preprocessor1_Model1

```r
bike_final_fit %>%
  extract_fit_parsnip() %>%
  vip::vip(aesthetics = list(fill = "steelblue", size = 0.8))
```


hour
datetime
atemp
workingday
temp
humidity
day_week
weather
month
windspeed
0.0e+00
5.0e+07
1.0e+08
1.5e+08
Importance

5 Extra : Screening many models

5.0.1 Import/préparation données

```
library(kknn)

data(penguins)
penguin <- penguins
penguin <- penguin %>%
  na.omit() %>%
  select(-year)

# Split
set.seed(1)
penguin_split <- initial_split(penguin, strata = species)
penguin_train <- training(penguin_split)
penguin_test <- testing(penguin_split)

# V-folds
set.seed(1)
penguin_folds <- vfold_cv(penguin_train, strata = species, repeats = 5)

# Normalizing data for model who needs it.
pg_normalized_rec <- recipe(species ~ ., penguin_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

```
bag_tree_rpart_spec <-
  bag_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")

decision_tree_rpart_spec <-
  decision_tree(tree_depth = tune(), min_n = tune(), cost_complexity = tune()) %>%
```

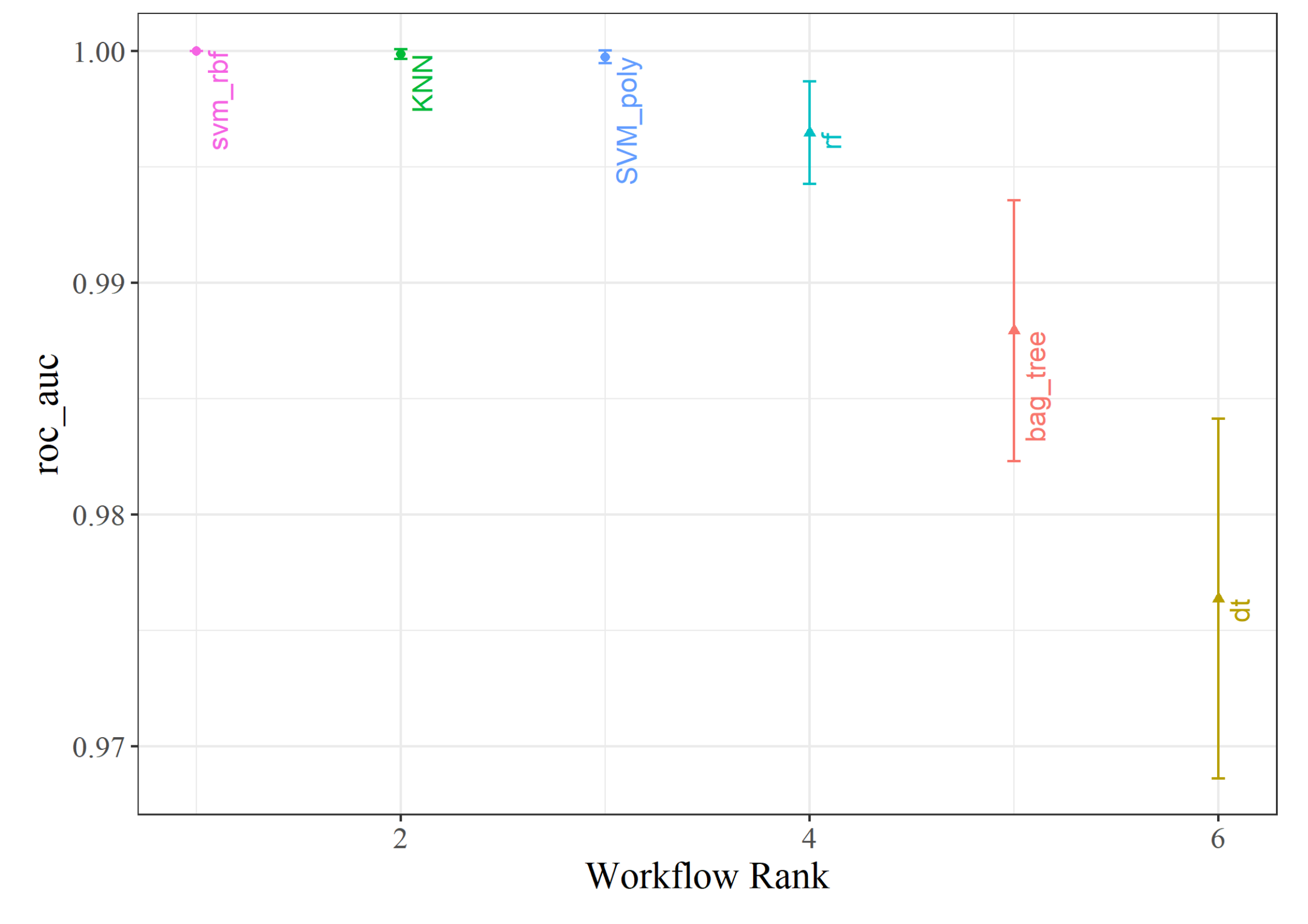
```
  set_engine("rpart") %>%
  set_mode("classification")

nearest_neighbor_kknn_spec <-
  nearest_neighbor(neighbors = tune(), weight_func = tune(), dist_power = tune()) %>%
  set_engine("kknn") %>%
  set_mode("classification")

rand_forest_ranger_spec <-
  rand_forest(mtry = tune(), min_n = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")

svm_poly_kernlab_spec <-
  svm_poly(cost = tune(), degree = tune(), scale_factor = tune(), margin = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

svm_rbf_kernlab_spec <-
  svm_rbf(cost = tune(), rbf_sigma = tune(), margin = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")
```

```
# model variables
model_vars <-
  workflow_variables(
    outcomes = species,
    predictors = everything()
  )

# Workflow for normalized models
normalized <-
  workflow_set(
    preproc = list(normalized = pg_normalized_rec),
    models = list(
      SVM_poly = svm_poly_kernlab_spec, KNN = nearest_neighbor_kknn_spec,
      svm_rbf = svm_rbf_kernlab_spec
    )
```

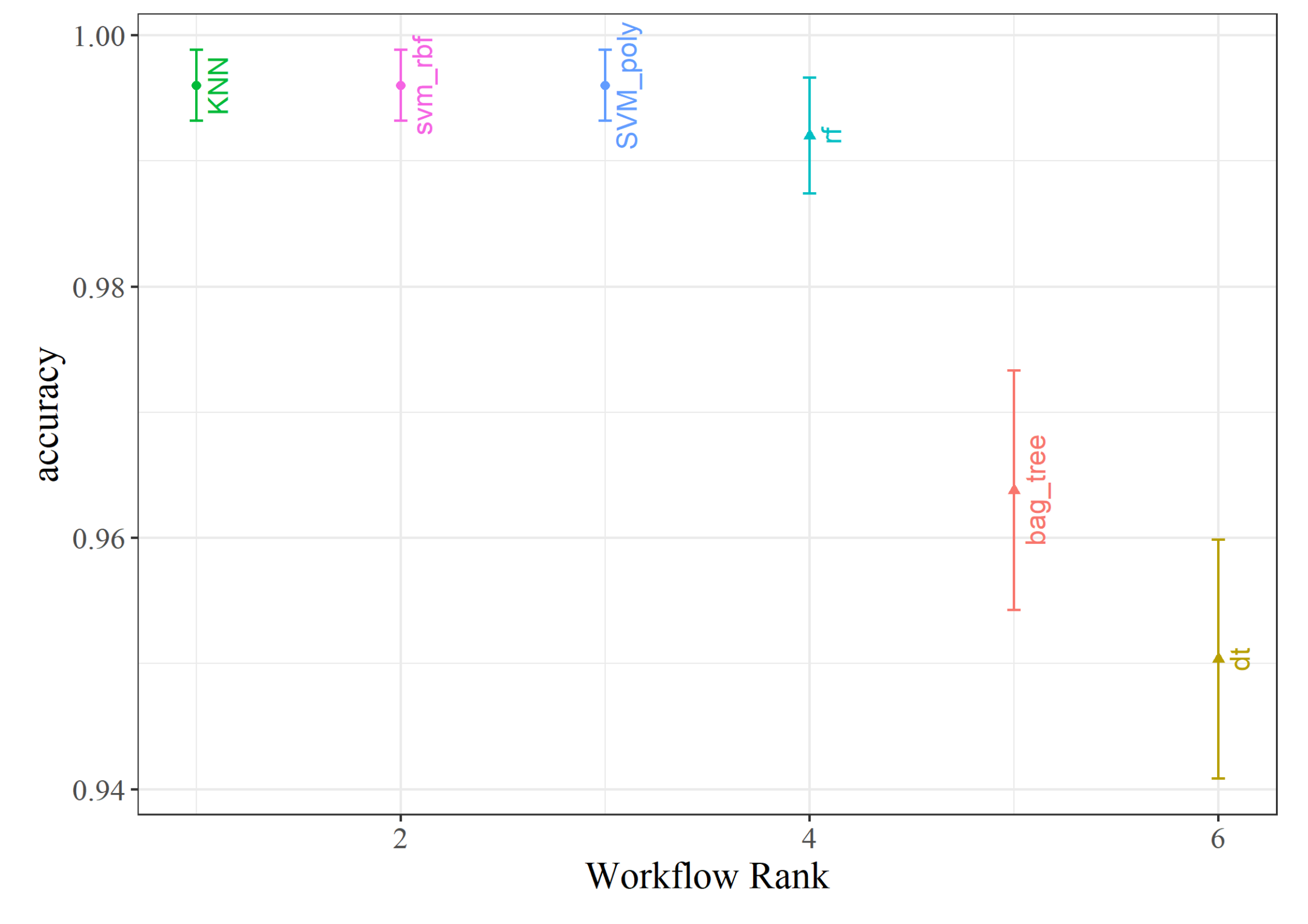
```
  )


# Workflow for not normalized models
no_pre_proc <-
  workflow_set(
    preproc = list(simple = model_vars),
    models = list(
      bag_tree = bag_tree_rpart_spec, dt = decision_tree_rpart_spec, rf = rand_forest_ranger_spec
    )
  )


# Merge all workflows
all_workflows <-
  bind_rows(no_pre_proc, normalized) %>%
  mutate(wflow_id = gsub("(simple_)|(normalized_)", "", wflow_id))
```

```
# Grid control parameters
grid_ctrl <-
  control_grid(
    save_pred = TRUE,
    parallel_over = "everything",
    save_workflow = TRUE
  )

# Grid tuning
grid_results <-
  all_workflows %>%
  workflow_map(
    seed = 1,
    resamples = penguin_folds,
    grid = 10,
    control = grid_ctrl,
    verbose = TRUE
  )
```

```r
# Ranking by roc_auc
autoplot(
  grid_results,
  rank_metric = "roc_auc",
  metric = "roc_auc",
  select_best = TRUE
) +
  geom_text(aes(label = wflow_id), angle = 90, hjust = 1, vjust = 1.5) +
  theme(legend.position = "none")
```


1.00
0.99
0.98
0.97
roc_auc
svm_rbf
KNN
SVM_poly
rf
bag_tree
dt
2
4
6
Workflow Rank

```r
# Ranking by accuracy
autoplot(
  grid_results,
  rank_metric = "accuracy",
  metric = "accuracy",
  select_best = TRUE
) +
  geom_text(aes(label = wflow_id), angle = 90, vjust = 1.5) +
  theme(legend.position = "none")
```


1.00
0.98
0.96
0.94
accuracy
KNN
svm_rbf
SVM_poly
rf
bag_tree
dt
2
4
6
Workflow Rank

```r
# best roc_auc
knn_best_results <- grid_results %>%
  extract_workflow_set_result("KNN") %>%
  select_best(metric = "roc_auc")

# Last fit
knn_test_results <-
  grid_results %>%
  extract_workflow("KNN") %>%
  finalize_workflow(knn_best_results) %>%
  last_fit(split = penguin_split)
```

```r
knn_results_pred <- knn_test_results %>%
  collect_predictions()

knn_results_pred %>% conf_mat(truth = species, estimate = .pred_class)
```

```
##            Truth
## Prediction  Adelie Chinstrap Gentoo
##   Adelie        37         0      0
##   Chinstrap      0        17      0
##   Gentoo         0         0     30
```