EST0133 - INTRODUÇÃO À MODELAGEM DE BIG DATA

Projeto II

Jaylhane Nunes

24/02/2022

```
library(tidyverse)
library(tidymodels)
library(GGally)
library(gridExtra)
library(grid)
library(doParallel)
library(onehot)
library(vip)
```

```
theme_set(theme_minimal()+
            theme(
              axis.title.y = element_text(size = 10),
              axis.title.x = element_text(size = 10),
              plot.title = element_text(size=12),
              plot.subtitle = element_text(size=10),
              panel.grid.minor.y = element_blank(),
              panel.grid.minor.x = element_blank()
            )
)
```

Parte I - Classificação

O arquivo `ataques_cardiacos.csv` traz informações a respeito de 299 pacientes que sofreram ataque cardíaco em algum momento de suas vidas. Eles foram acompanhados durante algum tempo e As colunas presentes são

- `idade`: idade do paciente (anos)
- `anemia`: se o paciente está anêmico ou não
- `cpk`: nível da enzima CPK no sangue (µg/L)
- `diabetes`: se o paciente possui diabetes
- `fracao_ejecao`: percentual de sangue saindo do coração a cada batida
- `pressao_alta`: se o paciente é hipertenso
- `plaquetas`: quantidade de plaquetas no sangue (em milhares/mL)
- `creatinina_sangue`: nível de creatinina no sangue (em mg/dL)
- `sodio`: nível de sódio no sangue (em mEq/L)
- `genero`: gênero do paciente
- `fumante`: se o paciente é fumante
- `morte`: evento de morte do paciente, isto é, se ele faleceu durante o acompanhamento médico

Queremos criar um modelo preditivo para o evento de morte do paciente, baseando-nos nas outras variáveis do conjunto de dados.

Questão 1

(*05 pontos*) *O primeiro passo será preparar o conjunto de dados para análise. Para isso, crie um objeto chamado* `coracao` *com o conteúdo do arquivo* `ataques_cardiacos.csv`*. Transforme a coluna* `morte` *de modo que* `sim` *seja o nível de referência.*

```
coracao <- read.csv("G:/Meu Drive/Graduacao Estatistica/2021.2/Intro a BigData/BigData/Projeto_II/dados/
  mutate(morte=ifelse(morte=="sim",1,0)) %>%
  mutate_if(is.character,as.factor)
```

Questão 2

(*05 pontos*) *Utilize a semente 1201 para criar os conjuntos de treino e teste. O conjunto de treino deve ser criado com 78% das observações.*

```
set.seed(1201, kind= "Mersenne-Twister", normal.kind = "Inversion")

(coracao_split <- initial_split( coracao, prop = .78))
```

```
## <Analysis/Assess/Total>
## <233/66/299>
```

```
coracao_treino <- training(coracao_split)
nrow(coracao_treino)/nrow(coracao)
```

```
## [1] 0.78
```

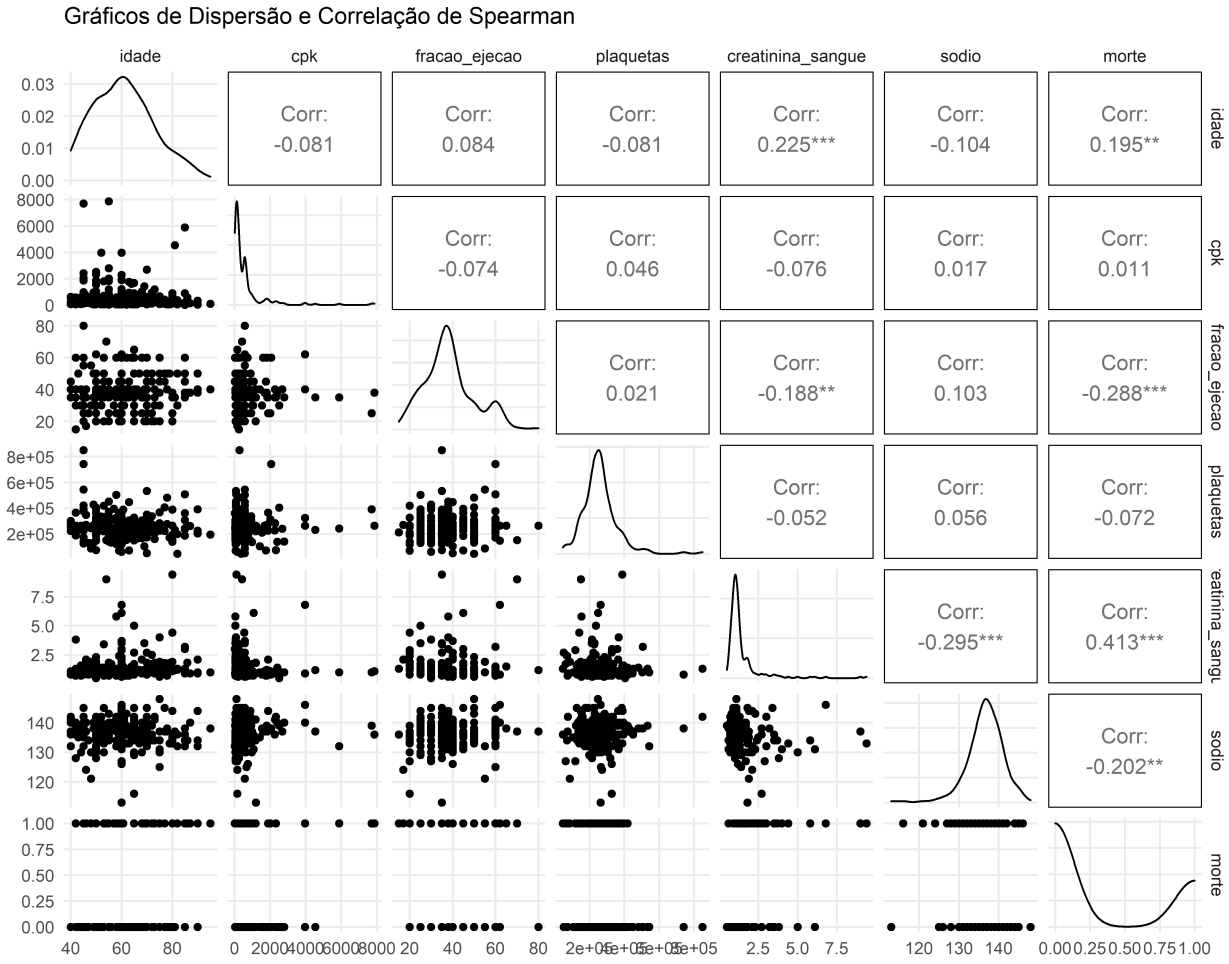
```
coracao_teste <- testing(coracao_split)
nrow(coracao_teste)/nrow(coracao)
```

```
## [1] 0.22
```

Questão 3

(05 pontos) Crie gráficos de dispersão em duas dimensões entre todas as variáveis quantitativas do conjunto de dados de treino. Informe também o valor da correlação de Spearman entre estas variáveis. Existe alguma suspeita de multicolinearidade entre estas variáveis? Justifique.

```
coracao_treino %>%
  select_if(is.numeric)%>%
  ggpairs(title = "Gráficos de Dispersão e Correlação de Spearman",
    upper = list(continuous=wrap("cor",method="spearman")))
```



Levando em consideração que:

- H_0 : A correlação entre a X_1 e X_2 é zero;
- H_1 : A correlação é diferente de zero;

Dado que o teste de correlação de spearman rejeitou a hipótese nula a um nível de significância de 5%, temos alguns candidatos a multicolinearidade, sendo eles:

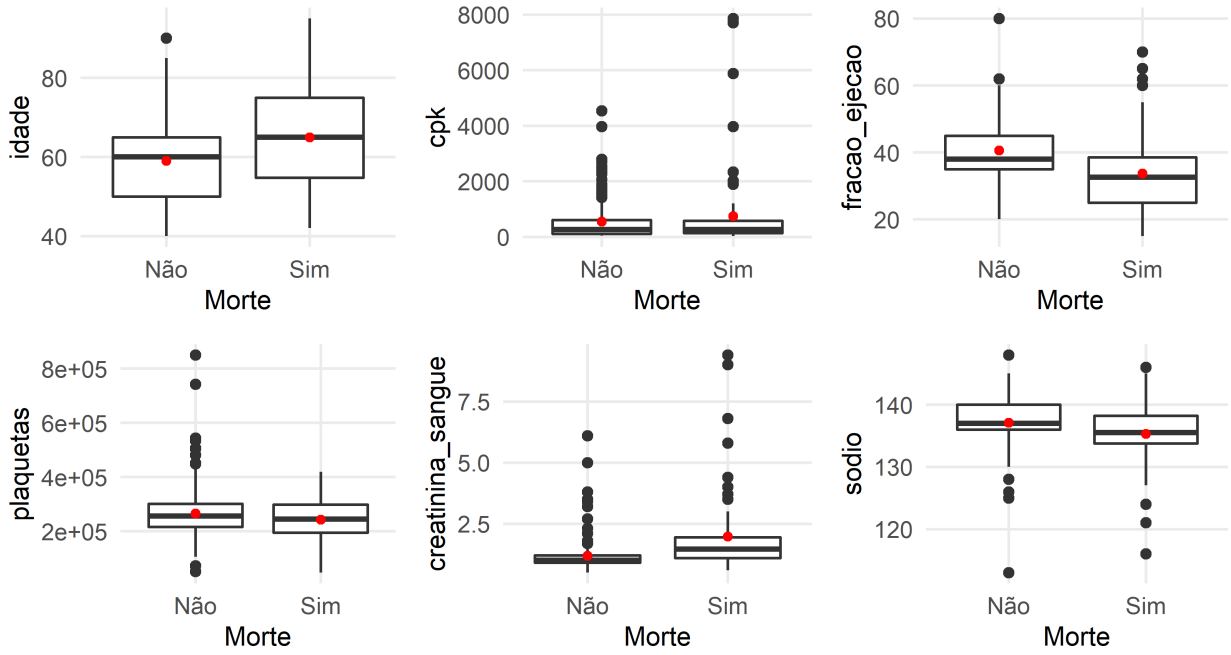
- `creatinina_sangue` com `idade`,
- `creatinina_sangue` com `fracao_ejecao`,
- `cretainina_sangue` com `sodio`.

No entanto, ao observar o valor da correlação percebemos que os valores são baixos (menores do que 0,3), o que diminui a preocupação de multicolinearidade, mas é necessário atenção nesses pares, uma vez que não é interessante remover a `creatinina_sangue` do modelo, uma vez que a mesma também está correlacionada com a morte.

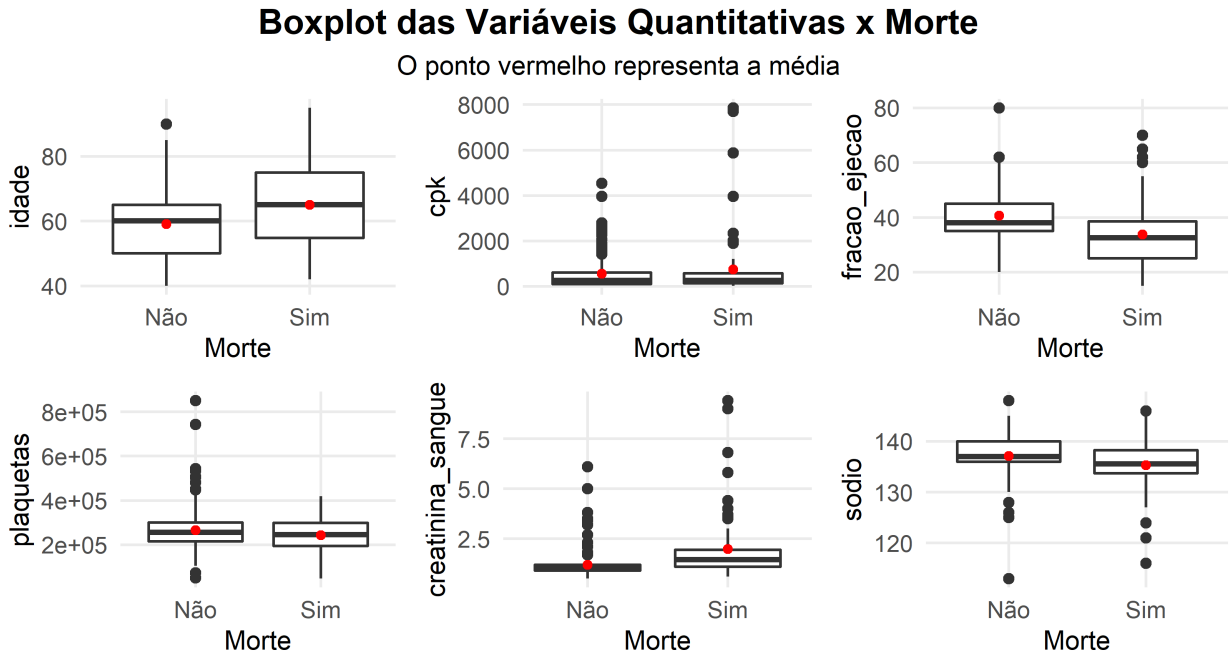
Questão 4

(*05 pontos*) *Crie boxplots comparando os valores das variáveis preditoras quantitativas entre os níveis de* `morte`*. Alguma (ou mais de uma) variável quantitativa poderia ser considerada como uma boa preditora para discriminar entre os níveis de* `morte`*? Qual (ou quais) e por quê?*

```
grafico_boxplot <- function(variavel){
coracao_treino %>%
  select_if(is.numeric) %>%
    mutate(morte=as.factor(morte)) %>%
    ggplot(aes_string(x="morte", y=variavel))+
    geom_boxplot()+
    stat_summary(fun=mean, geom="point", shape=20, size=2, color="red", fill="red")+
    labs(x="Morte")+
    scale_x_discrete(breaks = c(0,1),
                     labels = c("Não","Sim"))
}

nomes_vars_quantitativas <-
  coracao_treino %>%
  select_if(is.numeric) %>%
  names()

graficos <- list()

for (i in 1:(length(nomes_vars_quantitativas)-1)) {
  graficos[[i]] <- grafico_boxplot(nomes_vars_quantitativas[i])
}


library(grid)

tg <- textGrob("Boxplot das Variáveis Quantitativas x Morte",
                     gp = gpar(fontsize = 13, fontface = 'bold'))
sg <- textGrob("O ponto vermelho representa a média",
                     gp = gpar(fontsize = 10))
margin <- unit(0.5, "line")
grided <- grid.arrange(grobs = graficos, nrow = 2)
```




```
grid.arrange(tg, sg, grided,
                        heights = unit.c(grobHeight(tg) + 1.2*margin,
                                         grobHeight(sg) + margin,
                                         unit(1,"null")))
```



As variáveis que apresentam em seus gráficos diferença na variação e média no nível "Sim" são as melhores candidatas a variável preditora, pois se elas apresentam diferença nessas quantidades possivelmente estão relacionadas com a causa da morte

Dessa forma, entre as variáveis quantitativas, teremos como variáveis preditora:

- `idade`;
- `fracao_ejecao`;
- `creatinina_sangue`;
- `sodio`.

Questão 5

(*05 pontos*) *Pré-processe os dados com apenas 3 transformações:*

i) *Balanceie o número de observações para cada classe da variável resposta;*

ii) *Deixe a média das variáveis preditoras igual a zero;*

iii) *Faça com que a variância das variáveis preditoras seja igual a um.*

Não é necessário realizar nenhum outro tipo de pré-processamento para essa análise. Aplique as transformações nos conjuntos de treino e teste.

Pensando nas variáveis preditoras, para gerar os conjuntos de treino e teste, não irei incluir as variáveis de `plaquetas` e `cpk`, pois elas não deram significativas na correlação e também apresentam a mesma média na análise visual do boxplot. No entanto, irei incluir as demais variáveis categóricas, pois elas podem ser importantes no modelo final de predição.

```
coracao_treino <- coracao_treino %>%
  mutate(morte=as.factor(morte)) %>%
  select(-plaquetas,
         -cpk)

coracao_teste <- coracao_teste %>%
  mutate(morte=as.factor(morte))%>%
  select(-plaquetas,
         -cpk)

coracao_rec <-
  recipe(morte~ .,
         data = coracao_treino) %>%
  themis::step_downsample(morte) %>%
  step_center(where(is.numeric)) %>%
  step_scale(where(is.numeric)) %>%
  prep()

coracao_treino_t <- juice(coracao_rec)

coracao_teste_t <- bake(coracao_rec,
                        new_data = coracao_teste)
```

Questão 6

(*05 pontos*) *Defina a validação cruzada com 6 grupos para avaliar o desempenho dos algoritmos que aplicaremos a esses dados. Utilize a semente 2022 para isso.*

```r
set.seed(2022, kind = "Mersenne-Twister", normal.kind = "Inversion")

coracao_treino_cv <- vfold_cv(coracao_treino, v=6)
```

Questão 7

(05 pontos) Crie grids de procura para os hiperparâmetros dos métodos CART e Random Forest. Encontre o melhor valor de ***cost_complexity*** *para o CART entre os valores* 10^{-5} *e* 10^{-1}*,* ***tree_depth*** *entre 1 e 5 e* ***min_n*** *entre 10 e 100. Utilize 5, 5 e 10 valores diferentes, respectivamente, para cada um destes hiperparâmetros (ou seja, ajuste 250 modelos diferentes). Para o random forest, encontre o melhor valor de* ***mtry*** *1 e o máximo permitido,* ***trees*** *entre 500 e 1000 e* ***min_n*** *entre 10 e 100. Utilize 4, 2 e 10 valores diferentes, respectivamente, para cada um destes hiperparâmetros (ou seja, ajuste 80 modelos diferentes).*

- CART

```r
## grid de procura
coracao_rpart_grid <- grid_regular(
  cost_complexity(range(-5,-1)),
  tree_depth(range(1,5)),
  min_n(range(10,100)),
  levels = c(5,5,10)
)

head(coracao_rpart_grid)
```

```
## # A tibble: 6 x 3
##   cost_complexity tree_depth min_n
##             <dbl>      <int> <int>
## 1         0.00001          1    10
## 2         0.0001           1    10
## 3         0.001            1    10
## 4         0.01             1    10
## 5         0.1              1    10
## 6         0.00001          2    10
```

```r
## definição do tuning

coracao_rpart_tune <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune(),
    min_n = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("classification")

## workflow

coracao_rpart_tune_wflow <-
  workflow() %>%
  add_model(coracao_rpart_tune) %>%
```

```
  add_formula(morte ~ .)

## parallel para melhora computacional
all_cores <- parallel::detectCores(logical = FALSE)

cl <- makePSOCKcluster(all_cores)
registerDoParallel(cl)

# avaliação do modelo

coracao_rpart_fit_tune <-
  coracao_rpart_tune_wflow %>%
  tune_grid(
    resamples = coracao_treino_cv,
    grid = coracao_rpart_grid
  )

parallel::stopCluster(cl)

## melhores modelos
coracao_rpart_fit_tune %>%
 show_best("roc_auc")
```

```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err
##             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl>
## 1         0.00001          5    20 roc_auc binary     0.768     6  0.0472
## 2         0.0001           5    20 roc_auc binary     0.768     6  0.0472
## 3         0.001            5    20 roc_auc binary     0.768     6  0.0472
## 4         0.00001          4    20 roc_auc binary     0.715     6  0.0506
## 5         0.0001           4    20 roc_auc binary     0.715     6  0.0506
## # ... with 1 more variable: .config <chr>
```

```
coracao_rpart_fit_tune %>%
 show_best("accuracy")
```

```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric  .estimator  mean     n std_err
##             <dbl>      <int> <int> <chr>    <chr>      <dbl> <int>   <dbl>
## 1         0.00001          5    20 accuracy binary     0.772     6  0.0336
## 2         0.0001           5    20 accuracy binary     0.772     6  0.0336
## 3         0.001            5    20 accuracy binary     0.772     6  0.0336
## 4         0.01             5    20 accuracy binary     0.742     6  0.0380
## 5         0.00001          4    20 accuracy binary     0.738     6  0.0384
## # ... with 1 more variable: .config <chr>
```

```
## melhor modelo

coracao_rpart_best <-
  coracao_rpart_fit_tune %>%
  select_best("accuracy")
```

Para a seleção dos melhores hiperparamêtros do método CART estou considerando a acurácia pois apresentou erro padrão ligeiramente menor do que na curva ROC. Dessa forma, os melhores hiperparametros para o método CART são:

i. `cost_complexity` : 10^{-5}
ii. `tree_depth` : 5
iii. `min_n` : 20

- Random Forest

```
## grid de procura

coracao_rf_grid <- grid_regular(mtry(range(1,9)),
                                trees(range(500,1000)),
                                min_n(range(10,100)),
                                levels = c(4,2,10))
## definição do tuning

coracao_rf_tune <- rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()
) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

# workflow

coracao_rf_tune_wflow <-
  workflow() %>%
  add_model(coracao_rf_tune) %>%
  add_formula(morte ~ .)

# avaliacao do modelo

## parallel para melhora computacional
all_cores <- parallel::detectCores(logical = FALSE)

cl <- makePSOCKcluster(all_cores)
registerDoParallel(cl)

coracao_rf_fit_tune <-  coracao_rf_tune_wflow %>%
  tune_grid(
    resamples = coracao_treino_cv,
    grid = coracao_rf_grid
  )

parallel::stopCluster(cl)

## melhores modelos
coracao_rf_fit_tune %>%
 show_best("roc_auc")
```

```
## # A tibble: 5 x 9
```

```
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     3   500   100 roc_auc binary     0.817     6  0.0329 Preprocessor1_Model74
## 2     3  1000    60 roc_auc binary     0.816     6  0.0352 Preprocessor1_Model46
## 3     3   500    40 roc_auc binary     0.815     6  0.0336 Preprocessor1_Model26
## 4     3  1000   100 roc_auc binary     0.815     6  0.0352 Preprocessor1_Model78
## 5     3   500    80 roc_auc binary     0.815     6  0.0355 Preprocessor1_Model58
```

```
coracao_rf_fit_tune %>%
 show_best("accuracy")
```

```
## # A tibble: 5 x 9
##    mtry trees min_n .metric  .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     9   500    30 accuracy binary     0.781     6  0.0255 Preprocessor1_Model~
## 2     6   500    30 accuracy binary     0.777     6  0.0316 Preprocessor1_Model~
## 3     9  1000    30 accuracy binary     0.777     6  0.0271 Preprocessor1_Model~
## 4     9  1000    40 accuracy binary     0.777     6  0.0287 Preprocessor1_Model~
## 5     6  1000    50 accuracy binary     0.768     6  0.0220 Preprocessor1_Model~
```

```
## melhor modelo

coracao_rf_best <-
  coracao_rf_fit_tune %>%
  select_best("roc_auc")
```

Para a seleção dos melhores hiperparametros de RF, optei pelo método de curva roc (`roc_auc`), pois, o erro padrão entre os dois métodos é bem similar, mas a curva roc apresentou maior média para a seleção das variáveis que irão compor o modelo, dessa forma os melhores hiperparametros são:

i. `mtry` : 3
ii. `trees` : 500
iii. `min_n` : 100

Questão 8

(*05 pontos*) *Rode o ajuste dos modelos definidos anteriormente. A seguir, utilize os meios necessários para determinar se a acurácia e a área sob a curva dos ajustes com os algoritmos utilizados foram maximizadas em algum momento.*

- CART

```
## melhor modelo

coracao_rpart_final <-
  coracao_rpart_tune_wflow %>%
  finalize_workflow(coracao_rpart_best)

coracao_rpart_final <- fit(coracao_rpart_final,
  coracao_treino_t)
```

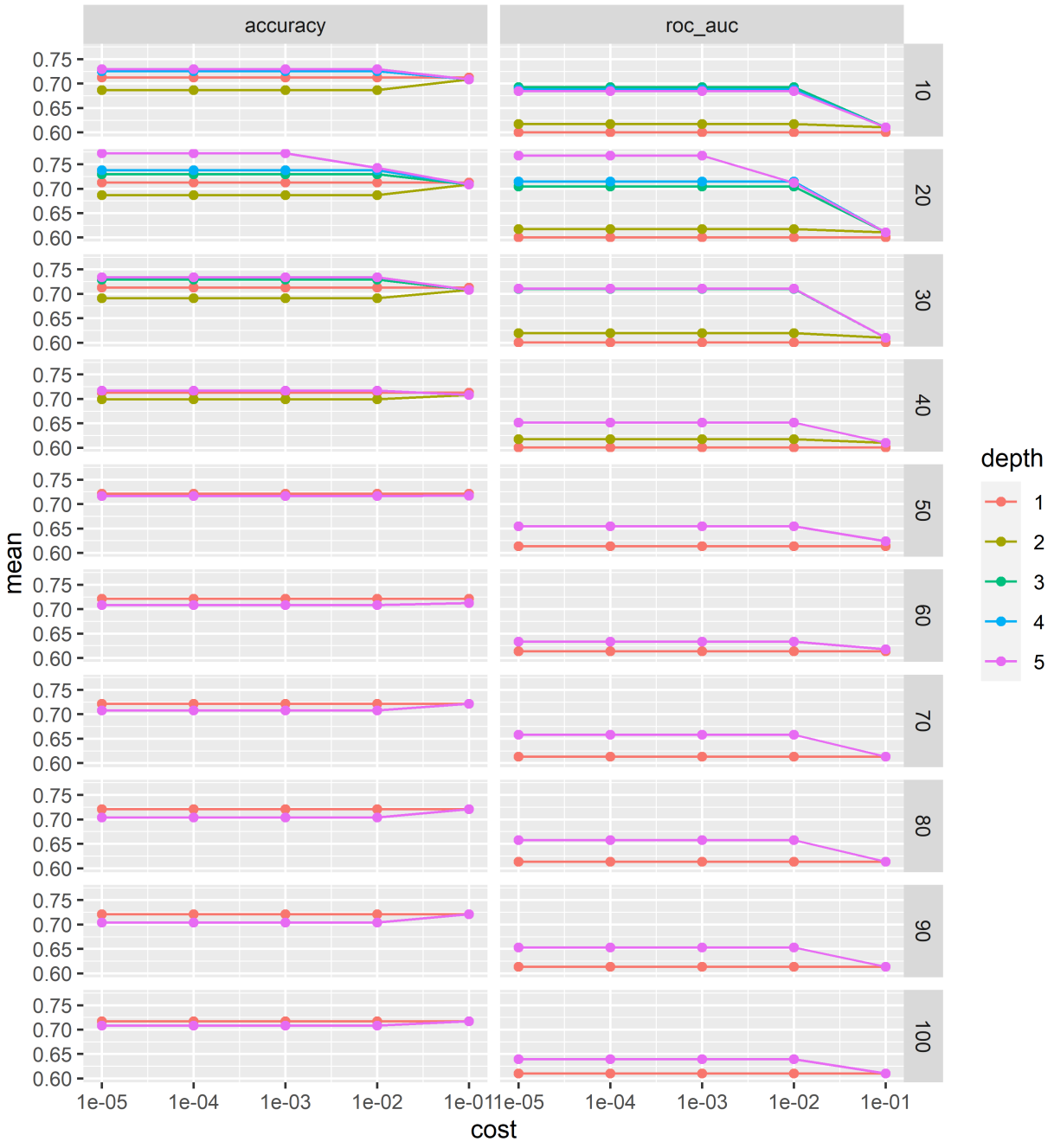
```
## resultados no conjunto de teste

resultado_rpart <- coracao_teste_t %>%
  bind_cols(predict(coracao_rpart_final, coracao_teste_t) %>%
              rename(predicao_rpart = .pred_class))

metrics(resultado_rpart,
        truth = morte,
        estimate = predicao_rpart,
        options = "accuracy")
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.652
## 2 kap      binary         0.240
```

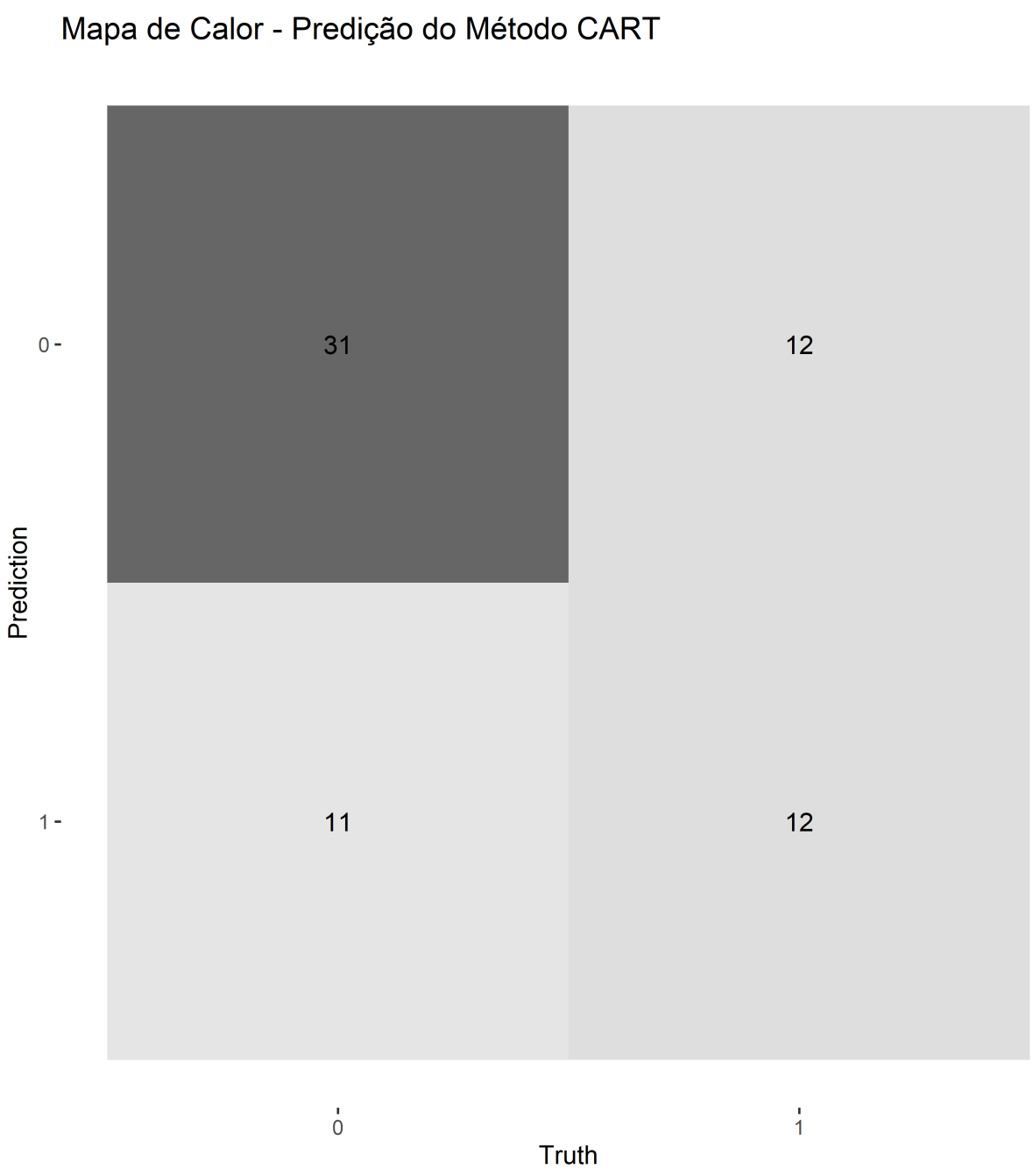
```
## resultados

collect_metrics(coracao_rpart_fit_tune)
```

```
## # A tibble: 500 x 9
##    cost_complexity tree_depth min_n .metric  .estimator  mean     n std_err
##              <dbl>      <int> <int> <chr>    <chr>      <dbl> <int>   <dbl>
##  1         0.00001          1    10 accuracy binary     0.713     6  0.0304
##  2         0.00001          1    10 roc_auc  binary     0.600     6  0.0271
##  3         0.0001           1    10 accuracy binary     0.713     6  0.0304
##  4         0.0001           1    10 roc_auc  binary     0.600     6  0.0271
##  5         0.001            1    10 accuracy binary     0.713     6  0.0304
##  6         0.001            1    10 roc_auc  binary     0.600     6  0.0271
##  7         0.01             1    10 accuracy binary     0.713     6  0.0304
##  8         0.01             1    10 roc_auc  binary     0.600     6  0.0271
##  9         0.1              1    10 accuracy binary     0.713     6  0.0304
## 10         0.1              1    10 roc_auc  binary     0.600     6  0.0271
## # ... with 490 more rows, and 1 more variable: .config <chr>
```

```
coracao_rpart_fit_tune %>%
  collect_metrics() %>%
  mutate(cost = cost_complexity,
    depth = factor(tree_depth)) %>%
  ggplot(., aes(x = cost, y = mean, colour = depth, group = depth)) +
    geom_line() +
    geom_point() +
    facet_grid(min_n ~ .metric) +
    scale_x_continuous(trans = "log10")
```




```
conf_mat(resultado_rpart,
         truth = morte,
         estimate = predicao_rpart) %>%
  autoplot(type = "heatmap")+
  ggtitle("Mapa de Calor - Predição do Método CART")
```



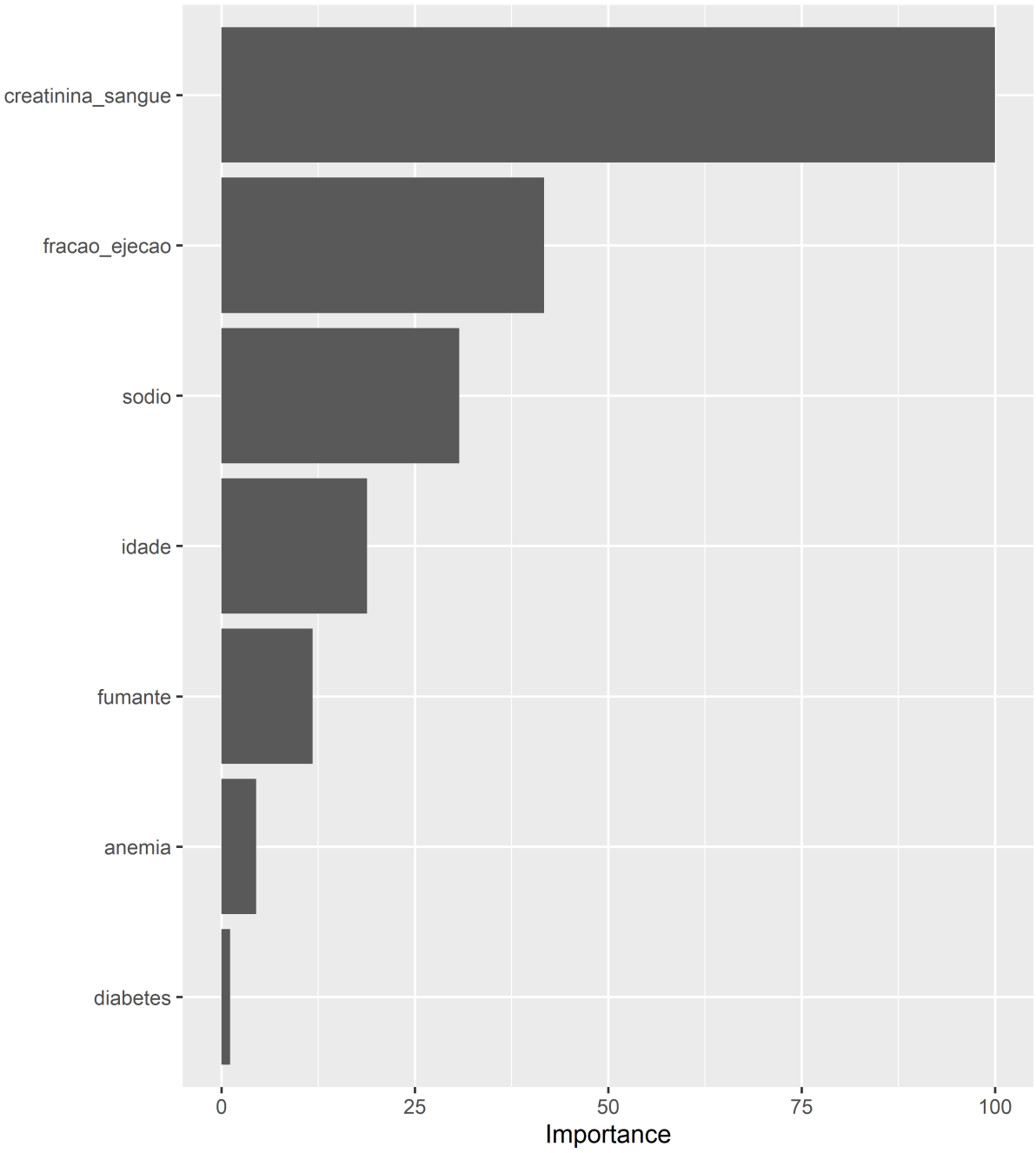

```
(sens_cart <- sens(resultado_rpart,
     truth = morte,
     estimate = predicao_rpart))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.738
```

```
(spec_cart <- spec(resultado_rpart,
     truth = morte,
     estimate = predicao_rpart))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary           0.5
```

```
# importancia das variaveis

coracao_rpart_final %>%
  pull_workflow_fit() %>%
  vip(scale=TRUE)
```



- Random Forest

```
# melhor modelo

coracao_rf_final <- coracao_rf_tune_wflow %>%
  finalize_workflow(coracao_rf_best)

coracao_rf_final <- fit(coracao_rf_final,
                        coracao_treino_t)
```

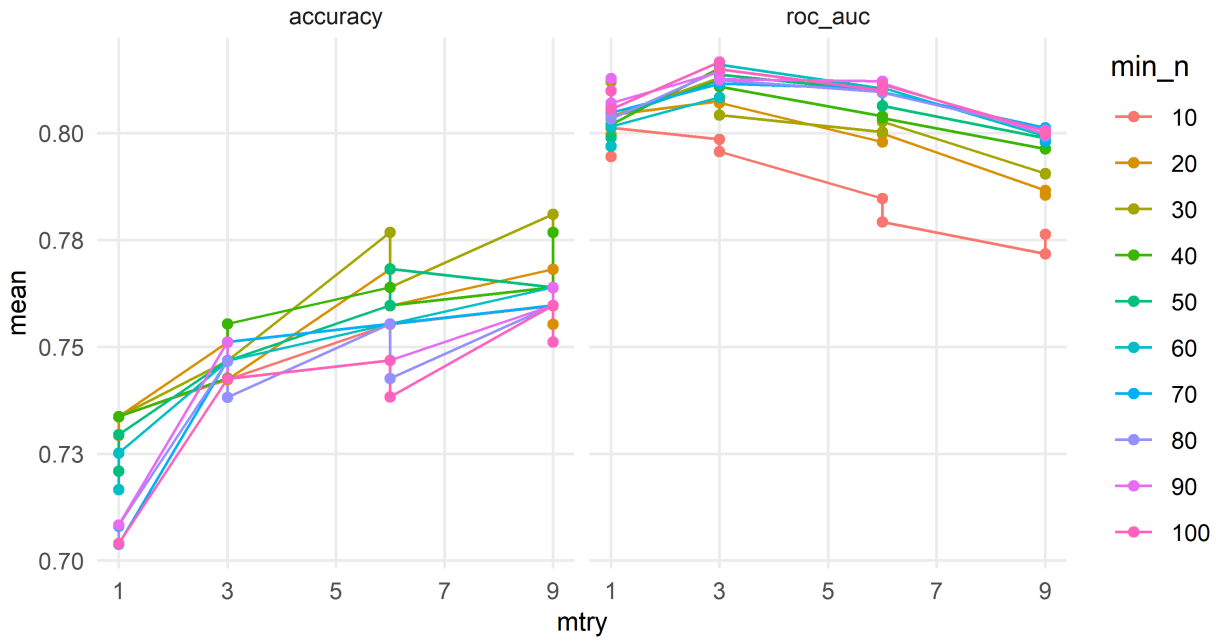
```r
# resultado no conjunto de teste

resultado_rf <- coracao_teste_t %>%
  bind_cols(predict(coracao_rf_final,coracao_teste_t) %>%
              rename(predicao_rf = .pred_class))

metrics(resultado_rf,
        truth = morte,
        estimate = predicao_rf,
        options = "roc")
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.667
## 2 kap      binary         0.292
```

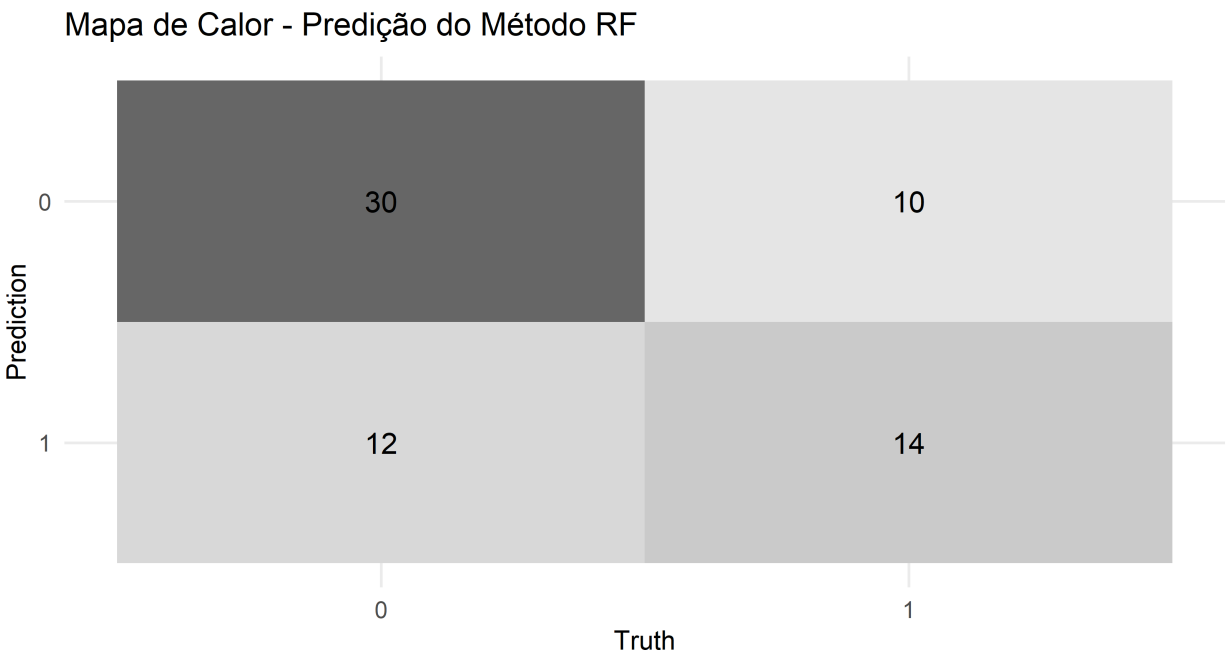
```r
## resultados

collect_metrics(coracao_rf_fit_tune)
```

```
## # A tibble: 160 x 9
##     mtry trees min_n .metric  .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1     1   500    10 accuracy binary     0.734     6  0.0348 Preprocessor1_Mode~
##  2     1   500    10 roc_auc  binary     0.795     6  0.0308 Preprocessor1_Mode~
##  3     3   500    10 accuracy binary     0.742     6  0.0313 Preprocessor1_Mode~
##  4     3   500    10 roc_auc  binary     0.799     6  0.0407 Preprocessor1_Mode~
##  5     6   500    10 accuracy binary     0.755     6  0.0310 Preprocessor1_Mode~
##  6     6   500    10 roc_auc  binary     0.785     6  0.0463 Preprocessor1_Mode~
##  7     9   500    10 accuracy binary     0.760     6  0.0295 Preprocessor1_Mode~
##  8     9   500    10 roc_auc  binary     0.772     6  0.0476 Preprocessor1_Mode~
##  9     1  1000    10 accuracy binary     0.734     6  0.0346 Preprocessor1_Mode~
## 10     1  1000    10 roc_auc  binary     0.801     6  0.0315 Preprocessor1_Mode~
## # ... with 150 more rows
```

```r
coracao_rf_fit_tune %>%
 collect_metrics() %>%
 mutate(min_n = factor(min_n)) %>%
 ggplot(., aes(x = mtry, y = mean, colour = min_n, group = min_n)) +
 geom_line() +
 geom_point() +
 facet_grid(~ .metric) +
 scale_x_continuous(breaks = seq(1, 9, 2))
```




```
conf_mat(resultado_rf,
         truth = morte,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap")+
  ggtitle("Mapa de Calor - Predição do Método RF")
```



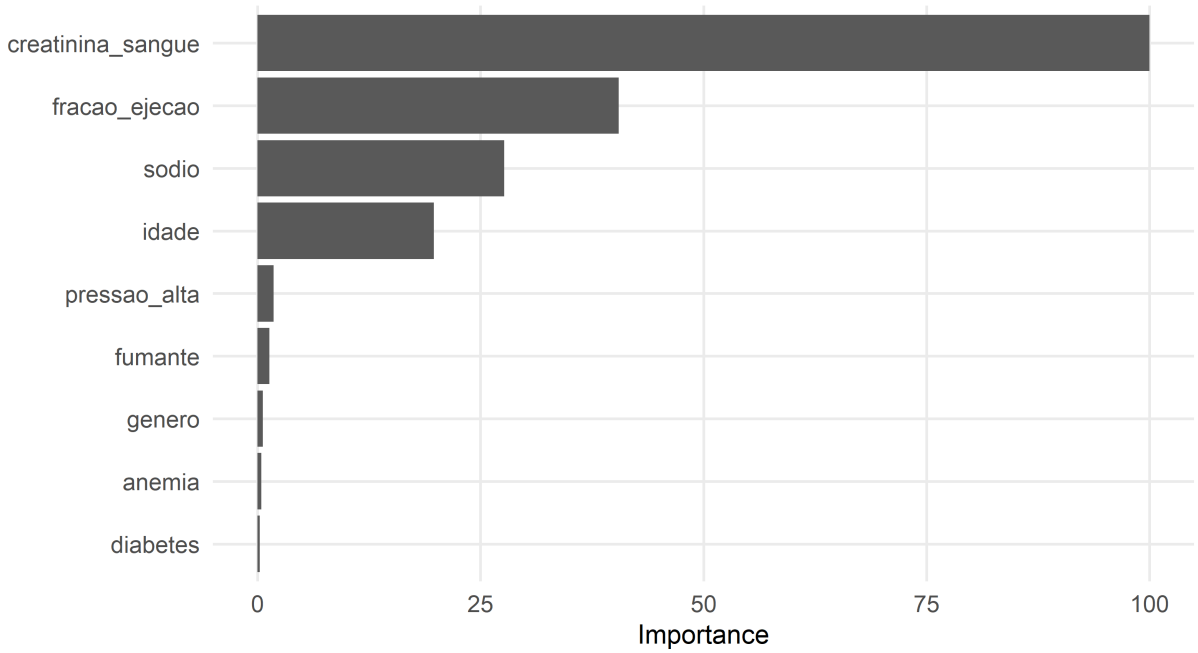

```
(sens_rf <- sens(resultado_rf,
     truth = morte,
     estimate = predicao_rf))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.714
```

```
(spec_rf <- spec(resultado_rf,
     truth = morte,
     estimate = predicao_rf))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary         0.583
```

```
# importancia das variaveis

coracao_rf_final %>%
  pull_workflow_fit() %>%
  vip::vip(scale=TRUE)
```



Questão 9

(*05 pontos*) *Qual é a sua opção de algoritmo para modelar estes dados? Justifique a sua escolha.*

Dado que estamos buscando um modelo capaz de predizer a morte para os dados e que os valores de sensibilidade e especificidade identificadas na questão 8 para RF são , respectivamente, 0.71 e 0.58, e são em média maiores do que de CART, com 0.74 e 0.5, portanto, eu escolheria o algoritmo de RF (Random Forest).

Acrescento que faria essa escolha também pois não há a informação se a morte e as causas do infarto estão relacionados, uma vez que os dados do estudo são provenientes de pacientes que já tiveram algum

infarto durante a vida e que morreram durante o acompanhamento, mas não necessariamente do infarto, o que considero ser um indício de que a condição de predição da morte mantem-se naturalmente imprevisível, de formo que considero mais relevante ter uma boa medida de especificidade.

Questão 10

(*05 pontos*) *Considerando métricas adequadas aplicadas nos conjuntos de treino e teste, o resultado obtido com a modelagem definitiva é bom o suficiente, na sua opinião? Cite alguma sugestão a ser aplicada nos dados ou na modelagem, que talvez pudesse melhorar o resultado obtido. Não é necessário implementar a sugestão, apenas comentá-la e justificá-la.*

Dado as taxas de sensibilidade e especificidade identificadas na questão 8, não considero que o modelo escolhido seja bom o suficiente, dado que a verdadeira proporção de mortes e não-mortes identificadas é de sens, binary, 0.738095238095238 e ‘spec, binary, 0.5, respectivamente, o que apresenta muita margem para predições equivocadas.

Acredito que dois procedimentos poderiam ser adotados para melhorar esses resultados:

- Uma análise de sobrevivência, levando em consideração que o paciente morrer durante o acompanhamento não necessariamente significa que ele morreu em decorrência de diabetes, infarto, cancêr de pulmão, velhice, ou outras possíveis causas relacionadas ao dados;
- Informar nos dados a causa da morte, pois talvez os modelos consigam predizer melhor a possibilidade de morte nos dados de acordo com a causa.

Parte II - Regressão

O twitch é um serviço de *streaming* de vídeos ao vivo. É bastante identificado com a comunidade de *esports*, embora possua canais especializados em diversas outras áreas de entretenimento. O arquivo `twitch.csv` possui informações sobre os 1000 canais mais populares em 2020, a saber:

- `channel`: nome do canal
- `watch_time_minutes`: somatório da quantidade total de minutos que o canal foi assistindo, considerando todos os usuários da plataforma
- `stream_time_minutes`: quantidade de minutos que o canal ficou ao vivo durante o ano
- `peak_viewers`: número máximo de espectadores simultâneos do canal
- `average_viewers`: quantidade média de espectadores simultâneos do canal
- `followers`: quantidade de seguidores do canal no final do ano
- `followers_gained`: diferença entre a quantidade de seguidores do canal no final e no começo do ano
- `views_gained`: visualizações ganhas pelo canal durante o ano
- `mature`: variável indicando se o conteúdo do canal é para adultos
- `language`: idioma principal do canal

O objetivo desta tarefa é modelar a variável `followers_gained`, a fim de explicar que fatores são capazes de determinar o número de seguidores que um canal pode arregimentar em um ano.

Questão 11

(*05 pontos*) *Importe para o R o conjunto de dados do problema. Retire a coluna com o nome do canal e recodifique a coluna* `language`*, mantendo apenas o nível* `English` *original e juntando todas as demais em* `Other`*.*

```
twitch <- read.csv("G:/Meu Drive/Graduacao Estatistica/2021.2/Intro a BigData/BigData/Projeto_II/dados/t
  select(-channel) %>%
  mutate(language=ifelse(language=="English","English","Other"))  %>%
  mutate_if(is.character,factor)

head(twitch)
```

```
##   watch_time_minutes stream_time_minutes peak_viewers average_viewers followers
## 1            6.2e+09              215250       222720           27716   3246298
## 2            6.1e+09              211845       310998           25610   5310163
## 3            5.6e+09              515280       387315           10976   1767635
## 4            4.0e+09              517740       300575            7714   3944850
## 5            3.7e+09              123660       285644           29602   8938903
## 6            3.7e+09               82260       263720           42414   1563438
##   followers_gained views_gained mature language
## 1          1734810     93036735     no  English
## 2          1370184     89705964     no  English
## 3          1023779    102611607    yes    Other
## 4           703986    106546942     no  English
## 5          2068424     78998587     no  English
## 6           554201     61715781     no  English
```

Questão 12

(*05 pontos*) *Utilize a semente 2109 para criar os conjuntos de treino e teste. O conjunto de treino deve ser criado com 70% das observações.*

```r
set.seed(2109, kind= "Mersenne-Twister", normal.kind = "Inversion")

(twitch_split <- initial_split(twitch, prop = .70))
```

```
## <Analysis/Assess/Total>
## <700/300/1000>
```

```r
twitch_treino <- training(twitch_split)
nrow(twitch_treino)/nrow(twitch)
```

```
## [1] 0.7
```

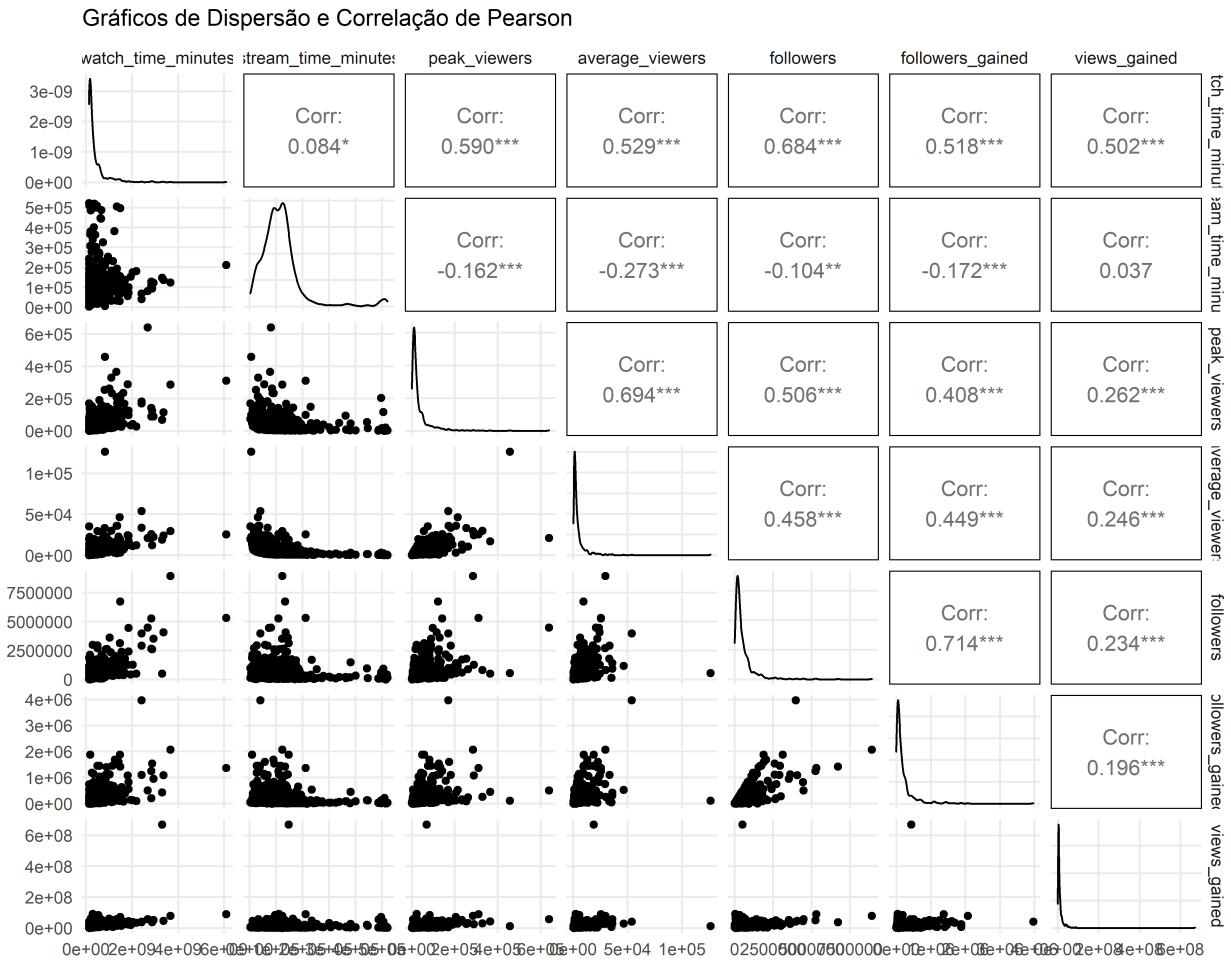
```r
twitch_teste <- testing(twitch_split)
nrow(twitch_teste)/nrow(twitch)
```

```
## [1] 0.3
```

Questão 13

Crie gráficos de dispersão em duas dimensões entre todas as variáveis quantitativas do conjunto de dados de treino. Informe também o valor da correlação linear entre estas variáveis. Alguma correlação entre as variáveis preditoras e a variável resposta se destaca? Existem indícios de multicolinearidade? Justifique.

```r
twitch_treino %>%
  select_if(is.numeric)%>%
  ggpairs(title = "Gráficos de Dispersão e Correlação de Pearson")
```



Há uma correlação significativa entre `followers` e `watch_time_minutes`, acima de 0,6, o que indica uma relação linear, no entanto, tal relação é esperada e intuitiva dado que é esperado que quanto mais seguidores o canal tenha, mais tempo assistindo a plataforma ele terá.

Da mesma forma entre `followers` e `followers_gained`, dado que há uma correlação com um valor acima de 0,7.

Além disso, como `followers` também dá significativo com as demais variáveis quantitativas do conjunto, ainda que com valores mais baixos, reforça indícios de multicolinearidade dessa variável com as demais no modelo, dado que todas as variáveis se baseiam ou tem alguma relação com a quantidade de usuários, no entanto, no momento, não iremos remover essa variável para ajuste do modelo.

Questão 14

(*05 pontos*) *Pré-processe os dados com apenas 4 transformações:*

i) *Transforme as variáveis quantitativas (exceto a resposta) utilizando logaritmo;*

ii) *Crie versões* dummy *das variáveis qualitativas usando a função* `step_dummy`

iii) *Deixe a média das variáveis preditoras igual a zero;*

iv) *Faça com que a variância das variáveis preditoras seja igual a um.*

Não é necessário realizar nenhum outro tipo de pré-processamento para essa análise. Aplique as transformações nos conjuntos de treino e teste.

```
twitch_rec <- recipe(followers_gained ~ . , data = twitch_treino) %>%
  step_dummy(all_nominal(), keep_original_cols = TRUE) %>%
  step_log(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes()) %>%
  prep(training= twitch_treino, retain = TRUE)

twitch_treino_t <- juice(twitch_rec)

head(twitch_treino_t)
```

```
## # A tibble: 6 x 11
##   watch_time_minutes stream_time_minutes peak_viewers average_viewers followers
##                <dbl>               <dbl>        <dbl>           <dbl>     <dbl>
## 1            -0.877              0.438      -0.292          -0.791      -0.381
## 2            -1.06              -0.0303     -1.11           -0.812      -0.669
## 3            -0.873             -0.835      -0.0243         -0.0677     -0.209
## 4             0.0158             0.143      -0.0383          0.00133    -0.214
## 5            -0.366              0.270      -0.662          -0.490      -0.624
## 6             0.823             -0.162       0.00161         0.841       0.293
## # ... with 6 more variables: views_gained <dbl>, mature <fct>, language <fct>,
## #   followers_gained <int>, mature_yes <dbl>, language_Other <dbl>
```

```
twitch_teste_t <- bake(twitch_rec, new_data = twitch_teste)

head(twitch_teste_t)
```

```
## # A tibble: 6 x 11
##   watch_time_minutes stream_time_minutes peak_viewers average_viewers followers
##                <dbl>               <dbl>        <dbl>           <dbl>     <dbl>
## 1               4.12                1.16         2.34            2.49      2.30
## 2               3.99                2.43         2.87            1.48      1.69
## 3               3.52                2.44         2.63            1.09      2.50
## 4               3.41               -0.240        2.50            2.96      1.57
## 5               2.94               -0.742        2.41            2.97      2.88
## 6               2.80                0.251        1.65            2.12      2.15
## # ... with 6 more variables: views_gained <dbl>, mature <fct>, language <fct>,
## #   followers_gained <int>, mature_yes <dbl>, language_Other <dbl>
```

Obs.: Por algum motivo que não conseguiu identificar ou encontrar justificativas, a função `step_dummy` *não funcionou, tentei mudar vários argumentos na função, mas não deu certo, dessa forma, mantive as colunas originais para substitui-las posteriormente*

```
twitch_treino_t <- twitch_treino_t %>%
  mutate(mature_yes=ifelse(mature=="yes",1,0),
         language_English=ifelse(language=="English",1,0)) %>%
  select(-mature,-language,-language_Other)

twitch_teste_t <- twitch_teste_t %>%
  mutate(mature_yes=ifelse(mature=="yes",1,0),
         language_English=ifelse(language=="English",1,0)) %>%
  select(-mature,-language,-language_Other)
```

Questão 15

(05 pontos) Defina a validação cruzada com 5 grupos para avaliar o desempenho dos algoritmos que aplicaremos a esses dados. Utilize a semente 2220 para isso.

```
set.seed(2220, kind = "Mersenne-Twister", normal.kind = "Inversion")

twitch_treino_cv <- vfold_cv(twitch_treino_t, v=5)
```

Questão 16

Utilize funções do pacote ***tidymodels*** *para ajustar um modelo de regressão linear múltipla aos dados que estamos analisando. Não é preciso realizar o tunning deste modelo.*

```
(glmn_fit <- linear_reg(penalty = .001, mixture = .5) %>%
  set_engine("glmnet") %>%
  fit(followers_gained ~., data=twitch_treino_t))
```

```
## parsnip model object
##
## Fit time:  80ms
##
## Call:  glmnet::glmnet(x = maybe_matrix(x), y = y, family = "gaussian",      alpha = ~0.5)
##
##    Df %Dev Lambda
## 1   0  0.0 374000
## 2   1  4.0 341000
## 3   1  7.8 310000
## 4   1 11.1 283000
## 5   2 14.7 258000
## 6   2 18.1 235000
## 7   2 21.1 214000
## 8   3 23.8 195000
## 9   4 26.2 178000
## 10  4 28.3 162000
## 11  4 30.1 147000
## 12  4 31.6 134000
## 13  4 33.0 122000
## 14  4 34.1 112000
## 15  4 35.0 102000
## 16  4 35.8  92600
## 17  4 36.5  84400
## 18  4 37.0  76900
## 19  4 37.5  70000
## 20  4 37.9  63800
## 21  4 38.2  58100
## 22  4 38.5  53000
## 23  4 38.8  48300
## 24  4 39.0  44000
## 25  5 39.2  40100
## 26  5 39.5  36500
## 27  5 39.7  33300
```

```
## 28  5 39.9  30300
## 29  5 40.0  27600
## 30  5 40.1  25200
## 31  5 40.2  22900
## 32  5 40.3  20900
## 33  5 40.4  19000
## 34  5 40.5  17300
## 35  5 40.5  15800
## 36  5 40.6  14400
## 37  5 40.6  13100
## 38  5 40.6  12000
## 39  5 40.6  10900
## 40  5 40.7   9930
## 41  5 40.7   9050
## 42  5 40.7   8240
## 43  5 40.7   7510
## 44  6 40.7   6840
## 45  6 40.8   6240
## 46  7 40.8   5680
## 47  8 40.8   5180
## 48  8 40.9   4720
## 49  8 40.9   4300
## 50  8 40.9   3920
## 51  8 40.9   3570
## 52  8 40.9   3250
## 53  8 41.0   2960
## 54  8 41.0   2700
## 55  8 41.0   2460
## 56  8 41.0   2240
## 57  8 41.0   2040
## 58  7 41.0   1860
## 59  7 41.0   1700
## 60  7 41.0   1540
## 61  7 41.0   1410
## 62  7 41.0   1280
## 63  7 41.0   1170
## 64  7 41.0   1060
## 65  7 41.0    970
## 66  7 41.0    884
## 67  7 41.0    805
## 68  7 41.0    734
## 69  7 41.0    669
## 70  7 41.0    609
## 71  8 41.0    555
## 72  8 41.0    506
## 73  8 41.0    461
## 74  8 41.0    420
## 75  8 41.0    383
## 76  8 41.0    349
## 77  8 41.1    318
## 78  8 41.1    289
```

Questão 17

(05 pontos) Utilize o random forest para ajustar um modelo a estes dados. Encontre o melhor valor de ***mtry*** *1 e o máximo permitido,* ***trees*** *entre 500 e 1000 e* ***min_n*** *entre 10 e 50. Utilize todos os valores possíveis, 2 e 5 valores diferentes, respectivamente, para cada um destes hiperparâmetros.*

```
# tuning

twitch_rf_tune <- rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()
) %>%
  set_mode("regression") %>%
  set_engine("ranger", importance = "impurity")

# grid de procura

twitch_rf_grid <- grid_regular(mtry(range(1,8)),
                               trees(range(500,1000)),
                               min_n(range(10,50)),
                               levels = c(8,2,5))

# workflow

twitch_rf_tune_wflow <- workflow() %>%
  add_model(twitch_rf_tune) %>%
  add_formula(followers_gained ~ .)

# avaliacao do modelo

all_cores <- parallel::detectCores(logical = FALSE)
cl <- makePSOCKcluster(all_cores)
registerDoParallel(cl)

twitch_rf_fit_tune <- twitch_rf_tune_wflow %>%
  tune_grid(
    resamples = twitch_treino_cv,
    grid = twitch_rf_grid
  )

parallel::stopCluster(cl)

# melhor modelo

twitch_rf_best <- twitch_rf_fit_tune %>%
  select_best("rmse")
```

De acordo com o melhor modelo selecionado visando o menor `rmse`, os hiperparametros do modelo são:

i. `mtry` : 4
ii. `trees` : 500
iii. `min_n` : 40

Questão 18

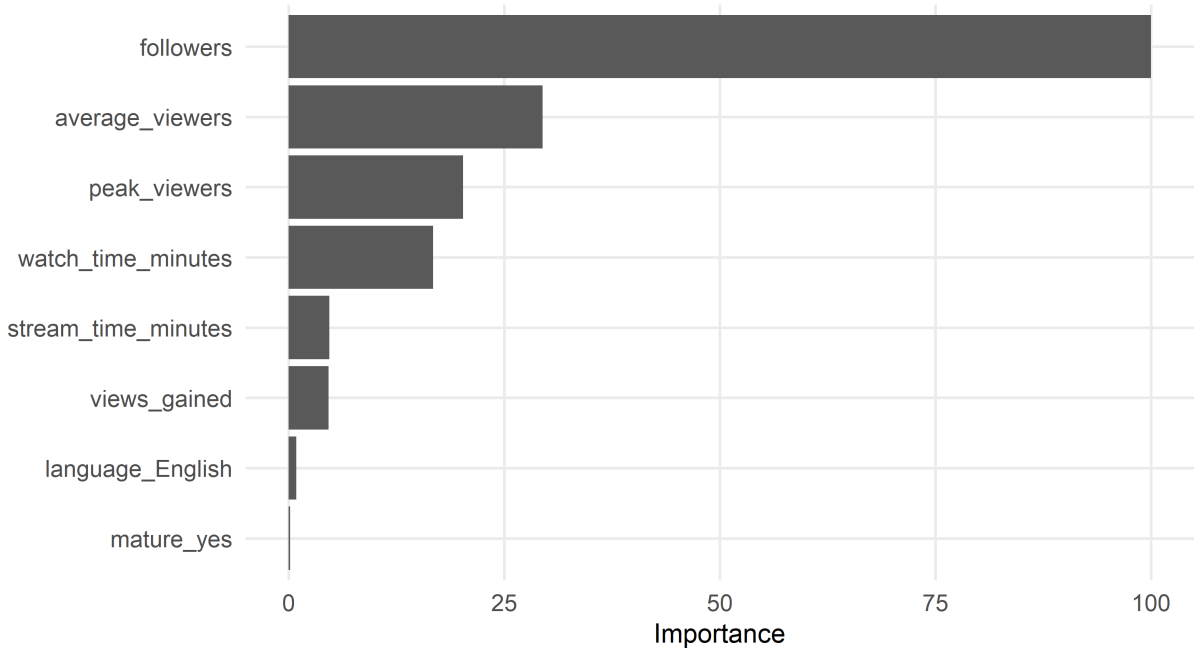
(05 pontos) Compare os resultados obtidos (no conjunto de treino) entre a regressão linear e o modelo final obtido com random forest utilizando a raiz do erro quadrático médio como critério. Qual é a sua opção de modelagem para estes dados e por quê?

Obs.: na questão 16 onde tem `data=twitch_teste_t`*, deveria ser* `data = bake(twitch_rec, new_data = NULL)` *justamente para que fosse possível comparar os modelos, conforme orientação obtida no guia: Regression models two ways , no entanto, como a minha "receita" não funcionou, não consegui realizar a questão 18, pois ao tentar fazer manualmente os erros persistiram.*

Questão 19

(05 pontos) Segundo o random forest, qual é a variável mais importante para o modelo ajustado? Intuitivamente, esse resultado faz sentido? Justifique.

```
#melhor modelo

twitch_rf_final <- twitch_rf_tune_wflow %>%
  finalize_workflow(twitch_rf_best)


twitch_rf_final <- fit(twitch_rf_final,
                       twitch_treino_t)

# importancia das variaveis

twitch_rf_final %>%
  pull_workflow_fit() %>%
  vip(scale=TRUE)
```



De acordo com o gráfico acima, a variável mais importante no modelo é a variável `followers`.

Conforme mencionado na questão 13, já era esperado que `followers` fosse ter essa importância no modelo, bem como era intuitivo esse comportamento, dado que as demais variáveis são definidas com base nessa variável .

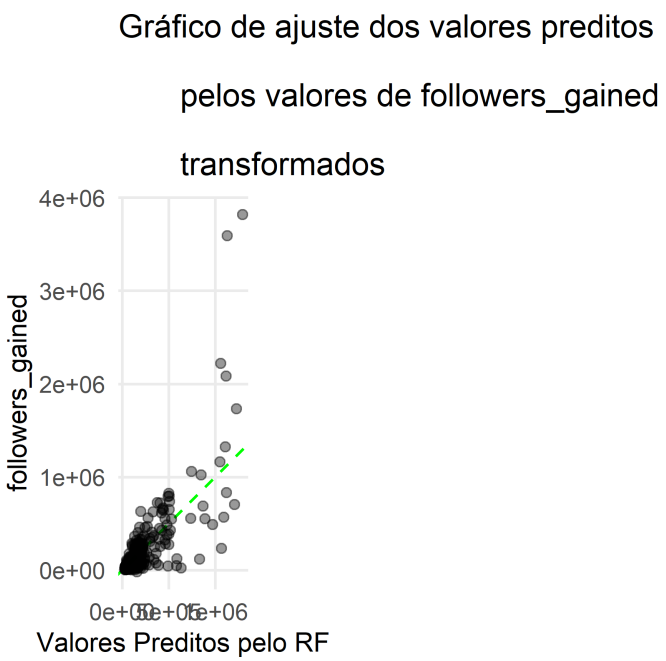
Questão 20

(*05 pontos*) *Considerando o conjunto de teste, o resultado obtido com a melhor modelagem é bom o suficiente? Utilize argumentos numéricos e gráficos para justificar a sua resposta.*

```
#resultados no conjunto de teste

resultado_rf <-
  twitch_teste_t %>%
  bind_cols(predict(twitch_rf_final, twitch_teste_t)%>%
              rename(predicao_rf = .pred))

(metricas <- metrics(resultado_rf,
        truth = followers_gained,
        estimate = predicao_rf,
        options = "rmse"))
```

```
## # A tibble: 3 x 3
##   .metric .estimator  .estimate
##   <chr>   <chr>           <dbl>
## 1 rmse    standard   268437.
## 2 rsq     standard        0.575
## 3 mae     standard   117176.
```

```
resultado_rf %>%
  ggplot(aes(x=predicao_rf, y=followers_gained))+
  geom_abline(col="green", lty=2)+
  geom_point(alpha=.4)+
  coord_fixed()+
  labs(title = "Gráfico de ajuste dos valores preditos\n
       pelos valores de followers_gained\n
       transformados",
       x = "Valores Preditos pelo RF")
```



De acordo com o valor de R^2 encontrado, 0.58, pode-se dizer que o modelo encontrado consegue explicar aproximadamente 57.51 % da variabilidade da variável resposta.

Apesar desse valor ser satisfatório, dado que ele capta a tendência dos dados, ao verificarmos graficamente o comportamento desse ajuste, percebemos uma heterocedasticidade, pois há uma concentração dos dados próximo a origem e um espalhamento conforme os pontos preditos aumentam, sendo possível observar inclusive a presença de possíveis outliers.