A.Aronikov - Fairness in ML Project - Final 2

Aleksandr Aronikov

11/10/2022

Note: Feel free to use the outline button on the right to navigate through this notebook in R-Studio

1.1 Introduction

Participatory Budgeting can be used as an attempt to ensure a bottom-up, inclusive and direct democratic practice. It has had many real-life demonstrated use cases on a state level, for instance in Brazil. It often consists of these phases:

1. The separation of the region under question in sub-regions
2. Budgeting proposals are generated and discussed by the public
3. The public votes on the proposals
4. The budget is allocated according to the vote

This practice can be facilitated with technology in all four phases. Also, e-participatory budgeting through voting online could increase voter participation in the population, which is one of the largest issues of all democratic practices.

Another issue of voting is the danger of underrepresented protected groups. This could be because, for instance, they sometimes differ from the majority populations, but have little means of having their voice heard. As a result, they are often discouraged to participate in the vote in the first place.

The question then can become if e-participatory budgeting can support in tackling that issue. Firstly, it can help in increasing voter participation through being online. Also, individuals might be more interested to vote for projects directly instead of voting for representatives. They also have the chance to directly suggest the budget items to vote on.

By simulating the vote and training an ML model on it, we can achieve several things. First, we can see what a simplified potential election in a developing country could look like, provided the vote is normally distributed. There, we can see how the distribution and population together form a vote per district.

What can the ML model be used for? Once trained, the model could be used to simulate the vote and outcome in a population comparable to the one in the training set. For this, we have to assume similarities between the populations. The vote does not have to be counted and decided and one could already project the chance of a person's preferred project will

be realized. Further, we can analyze what subsections of the population have which chances to have their vote realized. Such a model, properly trained and used, has strong potential use cases in political or demographic studies. It could project future vote results and how for example targeting a specific voting group can change the outcome.

Then, by examining the ML model, we can investigate, firstly, how and to what extent the Logistic Regression and the Random Forest Models are able to predict if the desired project is realized or not. We can then identify which variables are most useful for predicting the outcome on a local and global level. This helps us further improve the model and understand the populations at the same time.

When looking at the fairness of these models, we can determine a protected group and see if there is any significant bias along any of the fairness metrics. Then, we can see if any bias mitigation techniques are helpful.

1.2 The Case

Eqatorial-Kundu (EK) is a fictional developing country. I have decided to select Cuba as a loose inspiration for the demographic. This is because it is not a fully developed country, which holds elections (albeit one-party elections) and had suitable data available. EK consists of several provinces of various populations,

For the purpose of this project, EK is holding a nationwide election in the spirit of participatory budgeting. Each province contains from 40,000 to over 300,000 people who voted in the election. Further, the provinces are characterized by their urbanization, either urban or rural. Further, the sex of each voter is known. The majority of people in the country identify as white, while there is a large non-white minority.

There is always the choice between two projects for each province (0 or 1). Each project is realized on a Province level, hence Project 0 can be realized for Province A and B, and Project 1 can be realized for Province C and vice versa. Each person has only one vote, to vote for either project 0 or project 1. The project with more votes in each state is realized (winner takes all).

We make it so the vote for the project is only influenced by race and is normally distributed. We further implement substantial differences between the two groups:

1. People who identify as white, on average, prefer project 0 over project 1. Yet, this group of people acts very heterogeneously and not as a monolith - their choice has a large standard deviation
2. People who identify as non-white, on average, very strongly prefer project 1 over 0. This group acts more homogeneously and more monolithic, with their choice having a small standard deviation.

After the vote, the respective projects are realized. We want to examine if the voting data together with the knowledge if the preferred project was realized can be leveraged to train

an ML model to predict/understand future elections on other budget items in this country or, potentially, other countries.

1.3 Creating Data Set

```
# Generating Data

library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2
──
## ✓ ggplot2 3.4.0      ✓ purrr   0.3.5
## ✓ tibble  3.1.8      ✓ dplyr   1.0.10
## ✓ tidyr   1.2.1      ✓ stringr 1.4.1
## ✓ readr   2.1.3      ✓ forcats 0.5.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts()
──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()

library(tidymodels)

## ── Attaching packages ─────────────────────────────────────── tidymodels 1.0.0
──
## ✓ broom        1.0.1     ✓ rsample      1.1.0
## ✓ dials        1.0.0     ✓ tune         1.0.1
## ✓ infer        1.0.3     ✓ workflows    1.1.0
## ✓ modeldata    1.0.1     ✓ workflowsets 1.0.0
## ✓ parsnip      1.0.2     ✓ yardstick    1.1.0
## ✓ recipes      1.0.2
## ── Conflicts ────────────────────────────────────────── tidymodels_conflicts()
──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
```

```
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Search for functions across packages at https://www.tidymodels.org/find/

# set your working directory to the current file directory
tryCatch({
  library(rstudioapi)
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
}, error=function(cond){message(paste("cannot change working directory"))
})

### Load Data from similar country: Using Cuba as insipration

# Loading and inspect Cuba data

Demographics <- read_csv('UNdata_Export_20221213_175019341.csv')

## Rows: 36 Columns: 10
## ── Column specification
──────────────────────────────────────────────────────────────────────

## Delimiter: ","
## chr (6): Country or Area, Area, Sex, National and/or ethnic group, Record Ty...
## dbl (3): Year, Source Year, Value
## lgl (1): Value Footnotes
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.

Provinces <- read.csv2('Cuba Voter Turnout Provinces.csv')

head(Provinces)

##           Province..Registered
## 1 voters",% of Registred Voters
## 2                          ,,
## 3 Pinar del Río,"452,557",5.24%
```

```
## 4      Artemisa,"393,143",4.55%
## 5  La Habana,"1,685,651",19.51%
## 6     Mayabeque,"296,126",3.43%
```

```
head(Demographics)
```

```
## # A tibble: 6 × 10
##   Country or …¹  Year Area  Sex   Natio…² Recor…³ Relia…⁴ Sourc…⁵  Value Value…⁶
##   <chr>         <dbl> <chr> <chr> <chr>   <chr>   <chr>     <dbl>  <dbl> <lgl>
## 1 Cuba           2002 Total Both… Total   Census… Final …    2007 1.12e7 NA
## 2 Cuba           2002 Total Both… Black   Census… Final …    2007 1.13e6 NA
## 3 Cuba           2002 Total Both… Mulatto Census… Final …    2007 2.78e6 NA
## 4 Cuba           2002 Total Both… White   Census… Final …    2007 7.27e6 NA
## 5 Cuba           2002 Total Male  Total   Census… Final …    2007 5.60e6 NA
## 6 Cuba           2002 Total Male  Black   Census… Final …    2007 5.94e5 NA
## # … with abbreviated variable names ¹`Country or Area`,
## #   ²`National and/or ethnic group`, ³`Record Type`, ⁴Reliability,
## #   ⁵`Source Year`, ⁶`Value Footnotes`
```

##1 Create Population with similar demographics per province

#####Total Pop needs to be reduced from Cuba because the Random Forest breaks Rstudio with higher number. We reduce the voters from the turnout data by the factor 10 and ony include a few provinces.

```
total_pop = 1177743
id <- (1:total_pop) #from total population in demographics
head(Provinces)
```

```
##           Province..Registered
## 1 voters",% of Registred Voters
## 2                         ,,
## 3 Pinar del Río,"452,557",5.24%
## 4      Artemisa,"393,143",4.55%
## 5  La Habana,"1,685,651",19.51%
## 6     Mayabeque,"296,126",3.43%
```

```
province_pop = round(total_pop * Provinces$X..of.Registred.Voters)


Pinar_pop = data.frame("ID"= 1:50000,"Province"= "Pinar", "Urbanisation" = "Rural", "Race" =
"White")
Pinar_pop2 = data.frame("ID"= 1:8548,"Province"= "Pinar", "Urbanisation" = "Rural", "Race" =
"Non-White")
Pinar = rbind(Pinar_pop,Pinar_pop2)

Atermisa_pop = data.frame("ID"= 1:40861,"Province"= "Atermisa", "Urbanisation" =
"Rural","Race" = "White")
Atermisa_pop2 = data.frame("ID"= 1:1000,"Province"= "Atermisa", "Urbanisation" = "Rural",
"Race" = "Non-White")
Atermisa = rbind(Atermisa_pop,Atermisa_pop2)

Mayabeque_pop = data.frame("ID"= 1:30000,"Province"= "Mayabeque", "Urbanisation" =
"Rural","Race" = "White")
Mayabeque_pop2 = data.frame("ID"= 1:8310,"Province"= "Mayabeque", "Urbanisation" =
"Rural", "Race" = "Non-White")
Mayabeque = rbind(Mayabeque_pop,Mayabeque_pop2)

Matanzas_pop = data.frame("ID"= 1:70000,"Province"= "Matanzas", "Urbanisation" =
"Rural","Race" = "White")
Matanzas_pop2 = data.frame("ID"= 1:2601,"Province"= "Matanzas", "Urbanisation" = "Rural",
"Race" = "Non-White")
Matanzas = rbind(Matanzas_pop,Matanzas_pop2)

La_Habana_pop = data.frame("ID"= 1:200000,"Province"= "La Habana", "Urbanisation" =
"Urban","Race" = "White")
La_Habana_pop2 = data.frame("ID"= 1:118076,"Province"= "La Habana", "Urbanisation" =
"Urban", "Race" = "Non-White")
La_Habana = rbind(La_Habana_pop,La_Habana_pop2)

Villa_pop = data.frame("ID"= 1:50000,"Province"= "Villa", "Urbanisation" = "Urban","Race"
="White")
```

```
Villa_pop2 = data.frame("ID"= 1:2700,"Province"= "Villa", "Urbanisation" = "Urban", "Race" =
"Non-White")
Villa  = rbind(Villa_pop, Villa_pop2)

# 2 Integrate into one Data Set
df = rbind(Pinar, Atermisa, Mayabeque, Matanzas, La_Habana, Villa)

# 3 Randomly Generate Sex
sex = c("Male","Female")

library(dplyr)



df = df %>%
  mutate(Sex = sample(sex, 582096, replace=TRUE))


#4 Seperate Data Frame by Race

grp_race <- df %>% group_by(Race)
racelist = group_split(grp_race)
race_non_white <- as.data.frame(racelist[[1]])
race_white <- as.data.frame(racelist[[2]])

count_race_non_white = as.numeric(count(race_non_white))
print(count_race_non_white)

## [1] 141235

count_race_white = as.numeric(count(race_white))
print(count_race_white)

## [1] 440861

#5 Simulate Vote with normal distribution based on race

library(truncnorm)
```

```r
non_white_vote = round(rtruncnorm(n=count_race_non_white, a=0, b=1, mean=0.8, sd=0.2))
race_non_white = race_non_white %>%
  mutate(Vote = non_white_vote)

white_vote = round(rtruncnorm(n=count_race_white, a=0, b=1, mean=0.35, sd=0.5))
race_white = race_white %>%
  mutate(Vote = white_vote)

#6 Integrate into one Data Frame again

df = rbind(race_non_white, race_white)

# Calculate Result of Vote and add it to Data Frame

library(data.table)

##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose

sum(df$Vote == 1 & df$Province == "Pinar") - sum(df$Vote == 0 & df$Province == "Pinar")

## [1] 60

# Result is Yes
setDT(df)[df$Province == "Pinar", Project := "1"]

sum(df$Vote == 1 & df$Province == "Atermisa") - sum(df$Vote == 0 & df$Province ==
"Atermisa") # Result is No
```

```
## [1] -4875

#Result is No
setDT(df)[df$Province == "Atermisa", Project := "0"]

sum(df$Vote == 1 & df$Province == "Mayabeque") - sum(df$Vote == 0 & df$Province ==
"Mayabeque")

## [1] 3062

# Result is Yes
setDT(df)[df$Province == "Mayabeque", Project := "1"]

sum(df$Vote == 1 & df$Province == "Matanzas") - sum(df$Vote == 0 & df$Province ==
"Matanzas")

## [1] -7485

# Result is No
setDT(df)[df$Province == "Matanzas", Project := "0"]

sum(df$Vote == 1 & df$Province == "La Habana") - sum(df$Vote == 0 & df$Province == "La
Habana")

## [1] 71932

# Result is Yes
setDT(df)[df$Province == "La Habana", Project := "1"]

sum(df$Vote == 1 & df$Province == "Villa") - sum(df$Vote == 0 & df$Province == "Villa")

## [1] -4718

# Result is No
setDT(df)[df$Province == "Villa", Project := "0"]

# Add vote_realized column, based on if the project voted for by the individual is realized

add_column(df, "vote_realized")
```

```
##          ID Province Urbanisation      Race    Sex Vote Project
##      1:     1    Pinar       Rural Non-White Female    1       1
##      2:     2    Pinar       Rural Non-White   Male    1       1
##      3:     3    Pinar       Rural Non-White   Male    1       1
##      4:     4    Pinar       Rural Non-White Female    1       1
##      5:     5    Pinar       Rural Non-White   Male    1       1
##     ---
## 582092: 49996    Villa       Urban     White Female    0       0
## 582093: 49997    Villa       Urban     White Female    0       0
## 582094: 49998    Villa       Urban     White Female    0       0
## 582095: 49999    Villa       Urban     White   Male    0       0
## 582096: 50000    Villa       Urban     White   Male    0       0
##         "vote_realized"
##      1:   vote_realized
##      2:   vote_realized
##      3:   vote_realized
##      4:   vote_realized
##      5:   vote_realized
##     ---
## 582092:   vote_realized
## 582093:   vote_realized
## 582094:   vote_realized
## 582095:   vote_realized
## 582096:   vote_realized
```

```
library(data.table)
setDT(df)[df$Vote == df$Project, vote_realized := 1]
setDT(df)[df$Vote != df$Project, vote_realized := 0]
```

2 Building the First Models

2.1 Building the Logistic Regression

```
library(tidyverse)
library(tidymodels)
```

```
dataset = df #for ease of use of previous formulas

# Mutating to factors to be able to work with the variables
head(dataset)
```

```
##    ID Province Urbanisation      Race    Sex Vote Project vote_realized
## 1:  1    Pinar       Rural Non-White Female    1       1            1
## 2:  2    Pinar       Rural Non-White   Male    1       1            1
## 3:  3    Pinar       Rural Non-White   Male    1       1            1
## 4:  4    Pinar       Rural Non-White Female    1       1            1
## 5:  5    Pinar       Rural Non-White   Male    1       1            1
## 6:  6    Pinar       Rural Non-White   Male    1       1            1
```

```
dataset <- dataset %>% mutate_if(is.character, as.factor)
dataset <- dataset %>% mutate_if(is.integer, as.factor)
dataset <- dataset %>% mutate_if(is.numeric, as.factor)

head(dataset)
```

```
##    ID Province Urbanisation      Race    Sex Vote Project vote_realized
## 1:  1    Pinar       Rural Non-White Female    1       1            1
## 2:  2    Pinar       Rural Non-White   Male    1       1            1
## 3:  3    Pinar       Rural Non-White   Male    1       1            1
## 4:  4    Pinar       Rural Non-White Female    1       1            1
## 5:  5    Pinar       Rural Non-White   Male    1       1            1
## 6:  6    Pinar       Rural Non-White   Male    1       1            1
```

```
# we create a data partition (training-test) by specifying our target variable
data_split <- initial_split(data = dataset,
                            prop = 0.8,
                            strata = vote_realized) # create a 80-20 split and use vote_Realized to stratify

data_train <- training(data_split) # this will be our training data
data_test <- testing(data_split) # this will be our test data

# step A: specify which variables will be used as predictors, and which one as target variable
```

```
target_var <- 'vote_realized'
model_form <- vote_realized ~ Race + Sex  + Vote + Project + Urbanisation


#### It is important not to include Province as a predictor variable. Including it leads to the glm.fit: algorithm did not converge error. This is probably because it perfectly separates the response variable.

# step B: specify any additional data processing
model_recipe <- recipe(formula = model_form, data = data_train) %>%
  step_dummy(all_nominal_predictors()) %>%  # this creates dummy variables for all categorical predictors, but not the target variable
  step_zv(all_predictors()) # this removes potential zero variance predictors


### Workflow Step 3: Specify the ML Model Type and Workflow ###


logreg_model <-
  # specify that the model is a logistic regression
  logistic_reg() %>%
  # note: the lm package/ the logistic regression model does not have hyperparameters, hence we don't need to specify extra arguments
  # select the engine/package that underlies the model
  set_engine("glm") %>%
  # select the binary classification mode
  set_mode("classification")


# then, let's put everything into a workflow
logreg_workflow <- workflow() %>%
    # add the recipe (data pre-processing)
    add_recipe(model_recipe) %>%
    # add the ML model
    add_model(logreg_model)
```

```r
classification_metrics <- metric_set(yardstick::accuracy,
                                     yardstick::kap,
                                     yardstick::roc_auc,
                                     yardstick::mn_log_loss,
                                     yardstick::sens,
                                     yardstick::spec,
                                     yardstick::f_meas,
                                     yardstick::precision,
                                     yardstick::recall)

# train the model
# we only need to change the workflow fit() function to fit_resamples()
# we can additionally specify control parameters for the resampling procedure
# e.g., here we specify that the second outcome level ("yes") is the level of interest
set.seed(99)



control <- control_resamples(save_pred = TRUE,
                             event_level = "second")

folds <- vfold_cv(data_train, v = 3) # REDUCED folds for quicker computation

logreg_fit_cv <-
  logreg_workflow %>%
  fit_resamples(folds,
                control = control,
                metrics  = classification_metrics)

metric_to_use <- "roc_auc"

best_params <- logreg_fit_cv %>% select_best(metric = metric_to_use)

logreg_fitted_workflow <- logreg_workflow %>%
  finalize_workflow(parameters = best_params) %>%
```

```
 fit(data_train)

##this takes some time

# investigate the result
logreg_fit_cv

## # Resampling results
## # 3-fold cross-validation
## # A tibble: 3 × 5
##   splits               id    .metrics         .notes           .predictions
##   <list>               <chr> <list>           <list>           <list>
## 1 <split [310450/155226]> Fold1 <tibble [9 × 4]> <tibble [0 × 3]> <tibble>
## 2 <split [310451/155225]> Fold2 <tibble [9 × 4]> <tibble [0 × 3]> <tibble>
## 3 <split [310451/155225]> Fold3 <tibble [9 × 4]> <tibble [0 × 3]> <tibble>

# get the evaluation metrics averaged across the different folds
logreg_fit_cv %>% collect_metrics()

## # A tibble: 9 × 6
##   .metric     .estimator  mean     n  std_err .config
##   <chr>       <chr>      <dbl> <int>    <dbl> <chr>
## 1 accuracy    binary     0.695     3 0.000277 Preprocessor1_Model1
## 2 f_meas      binary     0.734     3 0.000362 Preprocessor1_Model1
## 3 kap         binary     0.376     3 0.000581 Preprocessor1_Model1
## 4 mn_log_loss binary     0.563     3 0.000219 Preprocessor1_Model1
## 5 precision   binary     0.741     3 0.00116  Preprocessor1_Model1
## 6 recall      binary     0.728     3 0.000798 Preprocessor1_Model1
## 7 roc_auc     binary     0.771     3 0.000479 Preprocessor1_Model1
## 8 sens        binary     0.728     3 0.000798 Preprocessor1_Model1
## 9 spec        binary     0.650     3 0.00152  Preprocessor1_Model1
```

2.1.1 Discussing the Logistic Regression

As mentioned before: It is important not to include Province as a predictor variable. Including it leads to the glm.fit: algorithm did not converge error. This is probably because it perfectly separates the response variable.

The model has a roc_auc of 0.77 which can be classified as decent and is not far away from being a good model. It is better than guessing, meaning its prediction is fair. Since the ROC is displaying both sensitivity and specificity, it is worthwhile to investigate them as well. The sensitivity is high at 0,73, the specificity is slightly lower at 0.65. That means that we predict slightly more correct positives than correctly predicting negatives. This practice is probably not particularly dangerous for this use case, as there is no harmful impact of either (unlike with medical tests, for instance).

When it comes to precision vs recall, we can see that the recall is almost as high (0.73) as precision (0.74). We can argue that most relevant instances were retrieved and there is a somewhat high fraction of relevant instances among the retrieved instances.

The accuracy is about 0.7, which is fair and better than guessing. Overall, it seems the model is pretty balanced. Both trade-offs of metrics (sensitivity vs specificity) and (precision vs recall) are very balanced, so one of them is not disproportionately off.

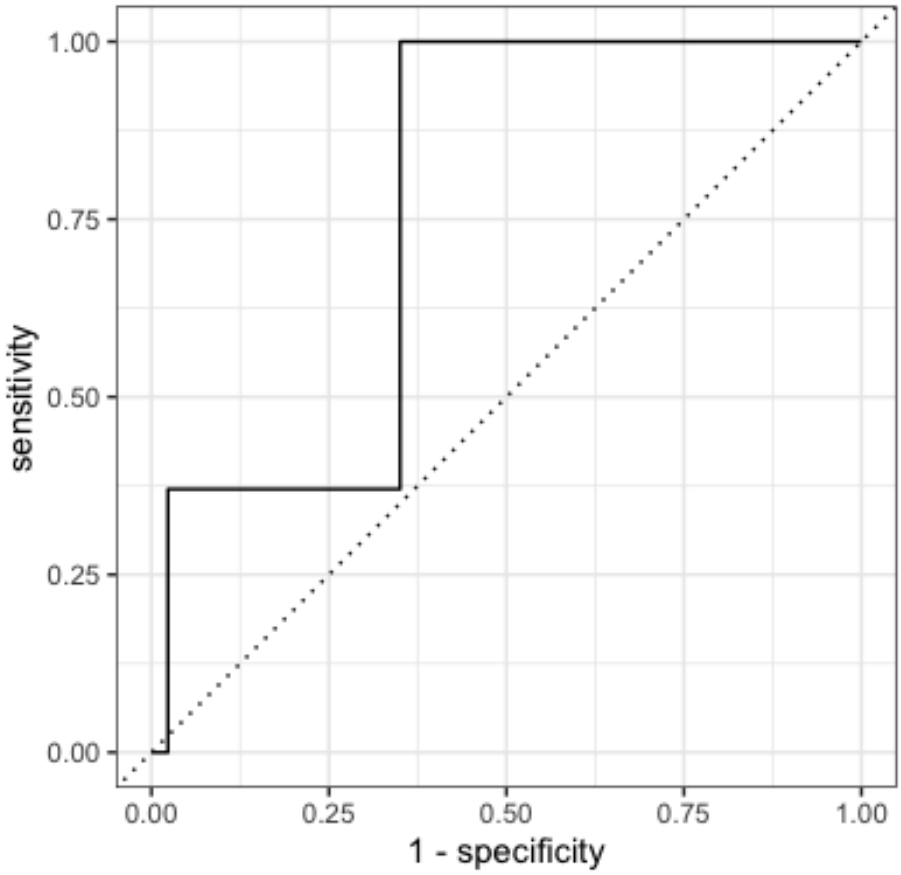
2.1.2 Discussing the ROC curve of the Logistical Regression

```
# first, we select the best hyperparameters based on the roc_auc metric (comment: could use a different metric as well)
metric_to_use <- "roc_auc"
best_params <- logreg_fit_cv %>% select_best(metric = metric_to_use)
best_params

## # A tibble: 1 × 1
##   .config
##   <chr>
## 1 Preprocessor1_Model1

# second, we update/finalize our model workflow using these parameters, and then re-fit on the entire training data
logreg_workflow_final <- logreg_workflow %>%
  finalize_workflow(parameters = best_params)

logreg_fit_final <- logreg_workflow_final %>%
  fit(data_train)

# investigate the statistical significance of the logistic regression model
logreg_model_fitted <- logreg_fit_final %>% extract_fit_engine()
summary(logreg_model_fitted)
```

```
##
## Call:
## stats::glm(formula = ..y ~ ., family = stats::binomial, data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3097  -0.8515   0.4597   0.9418   1.3692
##
## Coefficients:
##                    Estimate Std. Error  z value Pr(>|z|)
## (Intercept)         1.164057   0.013867   83.947   <2e-16 ***
## Race_White         -1.604484   0.010787 -148.748   <2e-16 ***
## Sex_Male            0.006839   0.006708    1.019   0.3080
## Vote_X1             1.411125   0.006979  202.188   <2e-16 ***
## Project_X1         -0.401120   0.007999  -50.147   <2e-16 ***
## Urbanisation_Urban  0.013471   0.007581    1.777   0.0756 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 633849  on 465675  degrees of freedom
## Residual deviance: 524183  on 465670  degrees of freedom
## AIC: 524195
##
## Number of Fisher Scoring iterations: 4

# note that we only use the training data for now
logreg_pred_train <-
  predict(logreg_fit_final, data_train) %>%
  bind_cols(predict(logreg_fit_final, data_train, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_train %>% select(all_of(target_var)))
```

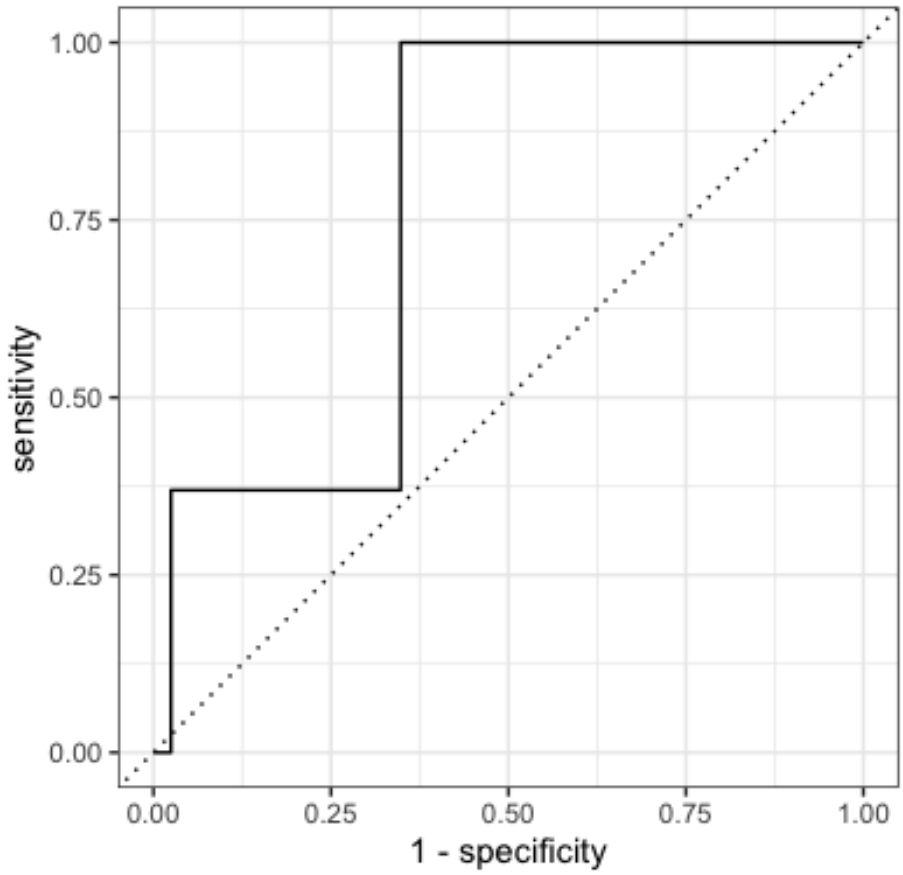
```
# note: you need to select the truth and estimate variables based on the column names of the logreg_pred_train object
classification_metrics(data = logreg_pred_train,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1, # use the second outcome (1) as the level of interest
                       event_level = 'second') # note: the "second" indicates that we use the second class (vote_realized = 1) as the level of interest

## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.695
## 2 kap         binary         0.376
## 3 sens        binary         0.728
## 4 spec        binary         0.650
## 5 f_meas      binary         0.734
## 6 precision   binary         0.741
## 7 recall      binary         0.728
## 8 roc_auc     binary         0.771
## 9 mn_log_loss binary         0.563

conf_mat(data = logreg_pred_train,
        truth = vote_realized,
        estimate = .pred_class)

##            Truth
## Prediction      0      1
##          0 127315  73338
##          1  68670 196353

two_class_curve_train <- roc_curve(data = logreg_pred_train,
                                   truth = vote_realized,
                                   estimate = .pred_1,
```

```
                          event_level = 'second')
autoplot(two_class_curve_train)
```



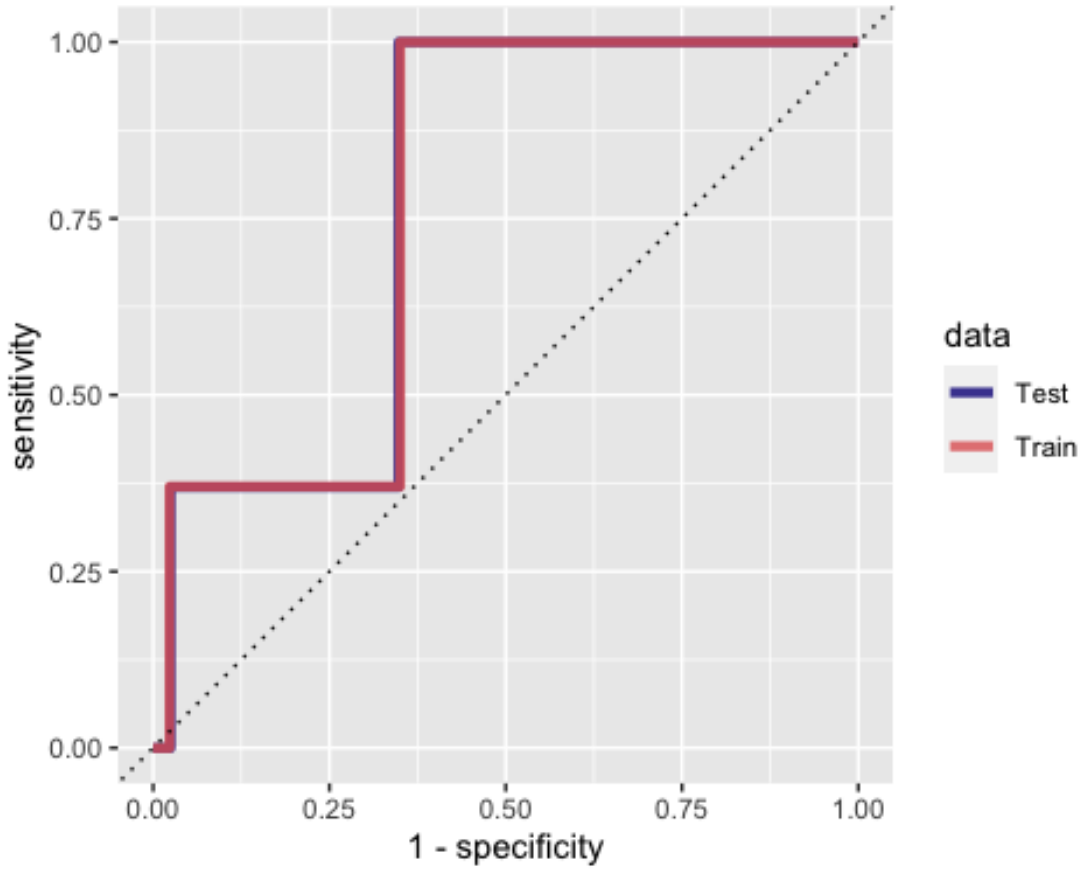

```
# Step 6 (5)
# note that we use the test data here!
logreg_pred_test <-
  predict(logreg_fit_final, data_test) %>%
  bind_cols(predict(logreg_fit_final, data_test, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_test %>% select(all_of(target_var)))


# you can manually change this, e.g., using the "probably" package


classification_metrics(data = logreg_pred_test,
                       truth = vote_realized,
```

```
                  estimate = .pred_class,
                  .pred_1,
                  event_level = 'second') # note: the "second" indicates that we use the second
class (vote_realized = 1) as the level of interest

## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.697
## 2 kap         binary         0.380
## 3 sens        binary         0.729
## 4 spec        binary         0.652
## 5 f_meas      binary         0.736
## 6 precision   binary         0.742
## 7 recall      binary         0.729
## 8 roc_auc     binary         0.771
## 9 mn_log_loss binary         0.563

conf_mat(data = logreg_pred_test,
        truth = vote_realized,
        estimate = .pred_class)

##           Truth
## Prediction     0     1
##          0 31937 18243
##          1 17060 49180

two_class_curve_test <- roc_curve(data = logreg_pred_test,
                                  truth = vote_realized,
                                  estimate = .pred_1,
                                  event_level = 'second')
autoplot(two_class_curve_test)
```




```
# side note: we can also plot them together!
# for this, we combine the results, add a column with the model ID, and color the lines according to the model ID

twoclass_train_test <- bind_rows(logreg_pred_train %>% mutate(data="Train"),
                                 logreg_pred_test %>% mutate(data="Test"))

twoclass_train_test <- bind_rows(two_class_curve_train %>% mutate(data="Train"),
                                 two_class_curve_test %>% mutate(data="Test"))

ggplot(data = twoclass_train_test,
       aes(x = 1 - specificity, y = sensitivity, color = data)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
```



The ROC curve shows that it is better than guessing (more than 0.5, the standard baseline) . Yet, there is still much room for improvement, as one can see in the empty space in the top left corner.

2.2. Building the Random Forest Model

```
library(randomForest)
library(ranger)
library(sparklyr)
library(bonsai)

library(tidymodels)
tidymodels_prefer()

forest_model <- rand_forest(
  mode="classification",
```

```
  engine="ranger",
  mtry = as.integer(7),
  trees = as.integer(1500),
  min_n = as.integer(1))
#define workflow
forest_workflow <- workflow() %>%
  add_recipe(model_recipe) %>%
  add_model(forest_model)

forest_fit_cv <-
  forest_workflow %>%
  fit_resamples(folds,
                control = control,
                metrics  = classification_metrics)
#Investigating
forest_fit_cv
forest_fit_cv %>% collect_metrics()

plot(forest_fit_cv)
metric_to_use <- "roc_auc"

rf_best_params <- forest_fit_cv %>% select_best(metric_to_use)

# finally, re-fit the logistic regression on the entire training data
rf_fitted_workflow <- rf_workflow %>%
  finalize_workflow(rf_best_params) %>%
  fit(data_train)
```

2.2.1 Discussing the Random Forest Model

```
# not rendered because of errors
## Using best parameters for the final model of the random forest :

# first, we select the best hyperparameters based on the roc_auc metric
metric_to_use <- "roc_auc"
randfor_best_params <- forest_fit_cv %>% select_best(metric = metric_to_use)
```

```
randfor_best_params

# second, we update/finalize our model workflow using these parameters, and then re-fit on the
entire training data
forest_workflow_final <- forest_workflow %>%
  finalize_workflow(parameters = randfor_best_params)

randfor_fit_final <- forest_workflow %>%
  fit(data_train)

# investigate the statistical significance of the random forest model
randfor_model_fitted <- randfor_fit_final %>% extract_fit_engine()
summary(randfor_model_fitted)

# note that we only use the training data for now
randfor_pred_train <-
  predict(randfor_fit_final, data_train) %>%
  bind_cols(predict(randfor_fit_final, data_train, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_train %>% select(all_of(target_var)))

# note: you need to select the truth and estimate variables based on the column names of the
logreg_pred_train object
classification_metrics(data = randfor_pred_train,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1, # use the second outcome (1) as the level of interest
                       event_level = 'second')

conf_mat(data = randfor_pred_train,
         truth = vote_realized,
         estimate = .pred_class)

two_class_curve_train <- roc_curve(data = randfor_pred_train,
                                   truth = vote_realized,
```

```
                        estimate = .pred_1,
                        event_level = 'second')
autoplot(two_class_curve_train)
```

2.2.2 Discussing the ROC curve of the Random Forest

```
# not rendered because of errors

##Computing test predictions for the random forest model:

randfor_pred_test <-
  predict(randfor_fit_final, data_test) %>%
  bind_cols(predict(randfor_fit_final, data_test, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_test %>% select(all_of(target_var)))


classification_metrics(data = randfor_pred_test,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1,
                       event_level = 'second')
#######SECOND CLASS

conf_mat(data = randfor_pred_test,
         truth = vote_realized,
         estimate = .pred_class)

two_class_curve_test <- roc_curve(data = randfor_pred_test,
                        truth = vote_realized,
                        estimate = .pred_1,
                        event_level = 'second')
autoplot(two_class_curve_test)



twoclass_train_test <- bind_rows(randfor_pred_train %>% mutate(data="Train"),
```

```
                        randfor_pred_test %>% mutate(data="Test"))

twoclass_train_test <- bind_rows(two_class_curve_train %>% mutate(data="Train"),
                        two_class_curve_test %>% mutate(data="Test"))

ggplot(data = twoclass_train_test,
       aes(x = 1 - specificity, y = sensitivity, color = data)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```

With the confusion matrix, we can see that the Random Forest Model has found a way to perfectly predict the outcome. This is confirmed in the metrics, where accuracy, sensitivity, specificity, recall, precision, and roc_auc are all 1.

This is, of course, due to the simplicity of the simulated data more than due to the excellence of the random forest model. Interestingly, the data was not simple enough for the logistic regression to yield perfect (or even good) results, but it does so for the random forest.

I assume that the reason lies in the different functions of the two models. While the logistic regression model attempts to approximate using a combination of the predictor variables. The random forest, on the other hand, uses a large number of decision trees. Here, the decision trees could have "found out" that if the project is realized and the vote is = 1, then the vote is realized. I now attempt to negate this by removing the project variable and retrying.

3. Building the Improved Models

3.1 Logistic Regression without Project Variable

```
library(tidyverse)
library(tidymodels)

dataset = df #for ease of use of previous formulas

# Mutating to factors to be able to work with the variables
head(dataset)
```

```
##    ID Province Urbanisation      Race    Sex Vote Project vote_realized
## 1:  1    Pinar       Rural Non-White Female    1       1            1
## 2:  2    Pinar       Rural Non-White   Male    1       1            1
## 3:  3    Pinar       Rural Non-White   Male    1       1            1
## 4:  4    Pinar       Rural Non-White Female    1       1            1
## 5:  5    Pinar       Rural Non-White   Male    1       1            1
## 6:  6    Pinar       Rural Non-White   Male    1       1            1
```

```
dataset <- dataset %>% mutate_if(is.character, as.factor)
dataset <- dataset %>% mutate_if(is.integer, as.factor)
dataset <- dataset %>% mutate_if(is.numeric, as.factor)

head(dataset)
```

```
##    ID Province Urbanisation      Race    Sex Vote Project vote_realized
## 1:  1    Pinar       Rural Non-White Female    1       1            1
## 2:  2    Pinar       Rural Non-White   Male    1       1            1
## 3:  3    Pinar       Rural Non-White   Male    1       1            1
## 4:  4    Pinar       Rural Non-White Female    1       1            1
## 5:  5    Pinar       Rural Non-White   Male    1       1            1
## 6:  6    Pinar       Rural Non-White   Male    1       1            1
```

```
# we create a data partition (training-test) by specifying our target variable
data_split <- initial_split(data = dataset,
                            prop = 0.8,
                            strata = vote_realized) # create a 80-20 split and use vote_realized to stratify
the samples

data_train <- training(data_split) # this will be our training data
data_test <- testing(data_split) # this will be our test data

# step A: specify which variables will be used as predictors, and which one as target variable
target_var <- 'vote_realized'
model_form <- vote_realized ~ Race + Sex  + Vote + Urbanisation
```

```
##It is important not to include Province as a predictor variable. Including it leads to the glm.fit: algorithm did not converge error. This is probably because it perfectly separates the response variable.

# step B: specify any additional data processing
model_recipe <- recipe(formula = model_form, data = data_train) %>%
  step_dummy(all_nominal_predictors()) %>%  # this creates dummy variables for all categorical predictors, but not the target variable
  step_zv(all_predictors()) # this removes potential zero variance predictors


### Workflow Step 3: Specify the ML Model Type and Workflow ###


logreg_model <-
  # specify that the model is a logistic regression
  logistic_reg() %>%
  # note: the lm package/ the logistic regression model does not have hyperparameters, hence we don't need to specify extra arguments
  # select the engine/package that underlies the model
  set_engine("glm") %>%
  # select the binary classification mode
  set_mode("classification")


# then, let's put everything into a workflow
logreg_workflow <- workflow() %>%
    # add the recipe (data pre-processing)
    add_recipe(model_recipe) %>%
    # add the ML model
    add_model(logreg_model)

classification_metrics <- metric_set(yardstick::accuracy,
                                     yardstick::kap,
                                     yardstick::roc_auc,
```

```
                          yardstick::mn_log_loss,
                          yardstick::sens,
                          yardstick::spec,
                          yardstick::f_meas,
                          yardstick::precision,
                          yardstick::recall)

# train the model
# we only need to change the workflow fit() function to fit_resamples()
# we can additionally specify control parameters for the resampling procedure
# e.g., here we specify that the second outcome level ("yes") is the level of interest
set.seed(99)



control <- control_resamples(save_pred = TRUE,
                             event_level = "second")

folds <- vfold_cv(data_train, v = 3) # REDUCED folds for quicker computation

logreg_fit_cv <-
  logreg_workflow %>%
  fit_resamples(folds,
                control = control,
                metrics  = classification_metrics)

metric_to_use <- "roc_auc"

best_params <- logreg_fit_cv %>% select_best(metric = metric_to_use)

logreg_fitted_workflow <- logreg_workflow %>%
  finalize_workflow(parameters = best_params) %>%
  fit(data_train)

##this takes some time
```

```
# investigate the result
logreg_fit_cv
```

```
## # Resampling results
## # 3-fold cross-validation
## # A tibble: 3 × 5
##   splits                id    .metrics         .notes           .predictions
##   <list>                <chr> <list>           <list>           <list>
## 1 <split [310450/155226]> Fold1 <tibble [9 × 4]> <tibble [0 × 3]> <tibble>
## 2 <split [310451/155225]> Fold2 <tibble [9 × 4]> <tibble [0 × 3]> <tibble>
## 3 <split [310451/155225]> Fold3 <tibble [9 × 4]> <tibble [0 × 3]> <tibble>
```

```
# get the evaluation metrics averaged across the different folds
logreg_fit_cv %>% collect_metrics()
```

```
## # A tibble: 9 × 6
##   .metric     .estimator  mean     n   std_err .config
##   <chr>       <chr>      <dbl> <int>     <dbl> <chr>
## 1 accuracy    binary     0.696     3 0.000368  Preprocessor1_Model1
## 2 f_meas      binary     0.735     3 0.000426  Preprocessor1_Model1
## 3 kap         binary     0.377     3 0.000761  Preprocessor1_Model1
## 4 mn_log_loss binary     0.565     3 0.0000177 Preprocessor1_Model1
## 5 precision   binary     0.741     3 0.00124   Preprocessor1_Model1
## 6 recall      binary     0.729     3 0.000755  Preprocessor1_Model1
## 7 roc_auc     binary     0.754     3 0.00110   Preprocessor1_Model1
## 8 sens        binary     0.729     3 0.000755  Preprocessor1_Model1
## 9 spec        binary     0.650     3 0.00163   Preprocessor1_Model1
```

```
# first, we select the best hyperparameters based on the roc_auc metric (comment: could use a different metric as well)
metric_to_use <- "roc_auc"
best_params <- logreg_fit_cv %>% select_best(metric = metric_to_use)
best_params
```

```
## # A tibble: 1 × 1
##   .config
```

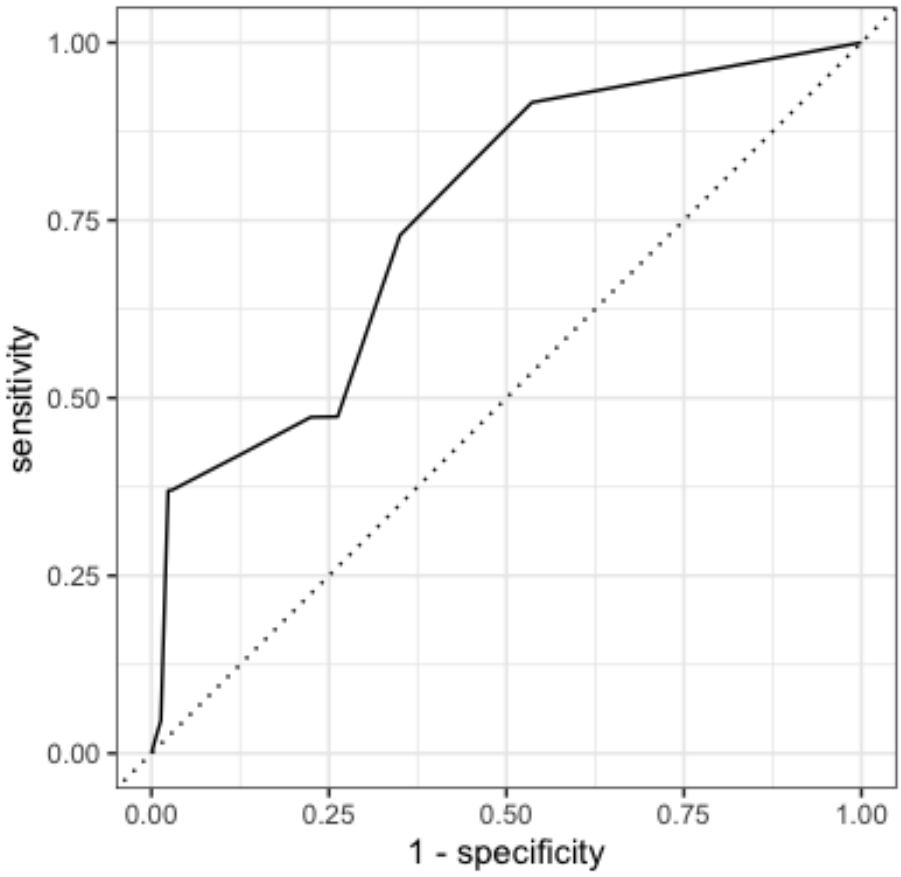
```
##   <chr>
## 1 Preprocessor1_Model1

# second, we update/finalize our model workflow using these parameters, and then re-fit on the entire training data
logreg_workflow_final <- logreg_workflow %>%
  finalize_workflow(parameters = best_params)

logreg_fit_final <- logreg_workflow_final %>%
  fit(data_train)

# investigate the statistical significance of the logistic regression model
logreg_model_fitted <- logreg_fit_final %>% extract_fit_engine()
summary(logreg_model_fitted)

##
## Call:
## stats::glm(formula = ..y ~ ., family = stats::binomial, data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1971  -0.8843   0.4613   0.9107   1.5039
##
## Coefficients:
##                    Estimate Std. Error  z value Pr(>|z|)
## (Intercept)        0.912975   0.012983   70.319   <2e-16 ***
## Race_White        -1.521330   0.010662 -142.692   <2e-16 ***
## Sex_Male           0.003890   0.006691    0.581    0.561
## Vote_X1            1.402996   0.006951  201.845   <2e-16 ***
## Urbanisation_Urban -0.132843   0.006985  -19.017   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 633849  on 465675  degrees of freedom
## Residual deviance: 526357  on 465671  degrees of freedom
## AIC: 526367
##
## Number of Fisher Scoring iterations: 4

# note that we only use the training data for now
logreg_pred_train <-
  predict(logreg_fit_final, data_train) %>%
  bind_cols(predict(logreg_fit_final, data_train, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_train %>% select(all_of(target_var)))

# note: you need to select the truth and estimate variables based on the column names of the
logreg_pred_train object
classification_metrics(data = logreg_pred_train,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1, # use the second outcome (1) as the level of interest
                       event_level = 'second') # note: the "second" indicates that we use the second
class (vote_realized = 1) as the level of interest

## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.696
## 2 kap         binary         0.377
## 3 sens        binary         0.729
## 4 spec        binary         0.650
## 5 f_meas      binary         0.735
## 6 precision   binary         0.741
## 7 recall      binary         0.729
## 8 roc_auc     binary         0.755
## 9 mn_log_loss binary         0.565
```

```
conf_mat(data = logreg_pred_train,
         truth = vote_realized,
         estimate = .pred_class)
```

```
##            Truth
## Prediction      0      1
##          0 127364  73136
##          1  68621 196555
```

```
two_class_curve_train <- roc_curve(data = logreg_pred_train,
                                   truth = vote_realized,
                                   estimate = .pred_1,
                                   event_level = 'second')
autoplot(two_class_curve_train)
```

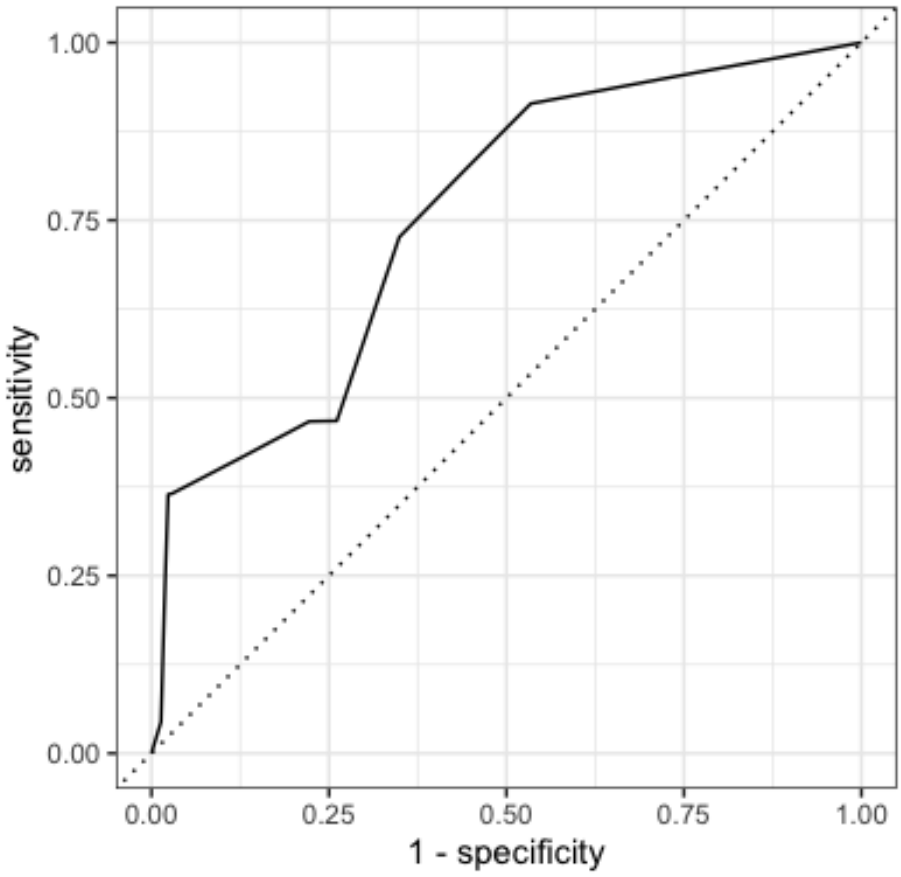



```
# Step 6 (5)
# note that we use the test data here!
```

```
logreg_pred_test <-
  predict(logreg_fit_final, data_test) %>%
  bind_cols(predict(logreg_fit_final, data_test, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_test %>% select(all_of(target_var)))


# you can manually change this, e.g., using the "probably" package


classification_metrics(data = logreg_pred_test,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1,
                       event_level = 'second') # note: the "second" indicates that we use the second
class (vote_realized = 1) as the level of interest

## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.695
## 2 kap         binary         0.376
## 3 sens        binary         0.726
## 4 spec        binary         0.651
## 5 f_meas      binary         0.734
## 6 precision   binary         0.741
## 7 recall      binary         0.726
## 8 roc_auc     binary         0.753
## 9 mn_log_loss binary         0.567

conf_mat(data = logreg_pred_test,
         truth = vote_realized,
         estimate = .pred_class)

##          Truth
## Prediction    0    1
```

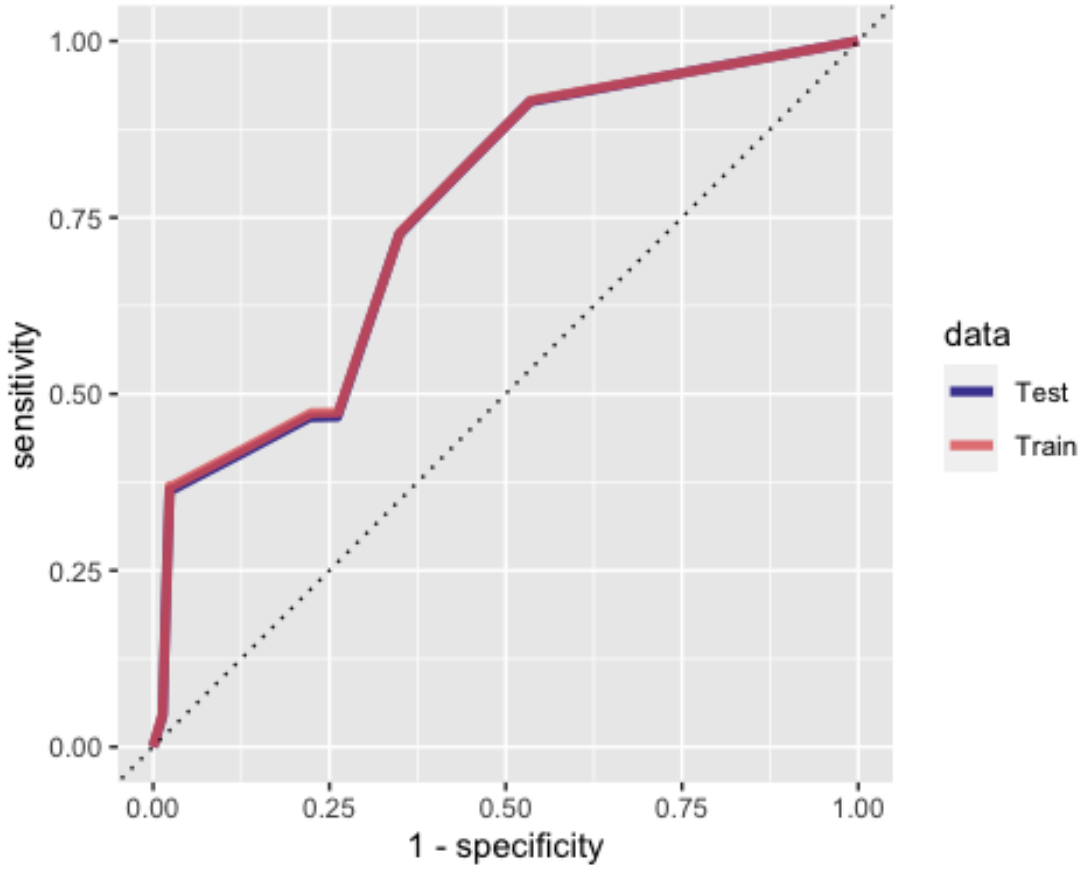
```
##          0 31888 18445
##          1 17109 48978
```

```
two_class_curve_test <- roc_curve(data = logreg_pred_test,
                                  truth = vote_realized,
                                  estimate = .pred_1,
                                  event_level = 'second')
autoplot(two_class_curve_test)
```




```
# side note: we can also plot them together!
# for this, we combine the results, add a column with the model ID, and color the lines according to the model ID

twoclass_train_test <- bind_rows(logreg_pred_train %>% mutate(data="Train"),
                                 logreg_pred_test %>% mutate(data="Test"))

twoclass_train_test <- bind_rows(two_class_curve_train %>% mutate(data="Train"),
```

```
                two_class_curve_test %>% mutate(data="Test"))

ggplot(data = twoclass_train_test,
       aes(x = 1 - specificity, y = sensitivity, color = data)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```



We see that the metrics of the logistic regression are almost unaffected by the removal or the "Project" variable. Only roc_auc loses about 0.015, which is understandable, as it benefits from all variables even slightly contributing to the explanation of the dependent variable. This confirms the model did not make almost any use of the variable.

3.2 Random Forest without Project Variable

```
library(randomForest)

## randomForest 4.7-1.1
```

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

##
## Attaching package: 'randomForest'

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

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

library(ranger)

##
## Attaching package: 'ranger'

## The following object is masked from 'package:randomForest':
##
##     importance

library(sparklyr)

##
## Attaching package: 'sparklyr'

## The following object is masked from 'package:purrr':
##
##     invoke

## The following object is masked from 'package:stats':
##
##     filter

library(bonsai)

library(tidymodels)
tidymodels_prefer()
```

```
forest_model <- rand_forest(
  mode="classification",
  engine="ranger",
  mtry = as.integer(7),
  trees = as.integer(1500),
  min_n = as.integer(1))
#define workflow
forest_workflow <- workflow() %>%
  add_recipe(model_recipe) %>%
  add_model(forest_model)

forest_fit_cv <-
  forest_workflow %>%
  fit_resamples(folds,
                control = control,
                metrics  = classification_metrics)
```

```
## ! Fold1: preprocessor 1/1, model 1/1: 7 columns were requested but there were 4 predictors in the data. 4 will...
```

```
## ! Fold2: preprocessor 1/1, model 1/1: 7 columns were requested but there were 4 predictors in the data. 4 will...
```

```
## ! Fold3: preprocessor 1/1, model 1/1: 7 columns were requested but there were 4 predictors in the data. 4 will...
```

```
#Investigating
forest_fit_cv
```

```
## # Resampling results
## # 3-fold cross-validation
## # A tibble: 3 × 5
##   splits               id    .metrics         .notes           .predictions
##   <list>               <chr> <list>           <list>           <list>
## 1 <split [310450/155226]> Fold1 <tibble [9 × 4]> <tibble [1 × 3]> <tibble>
```

```
## 2 <split [310451/155225]> Fold2 <tibble [9 × 4]> <tibble [1 × 3]> <tibble>
## 3 <split [310451/155225]> Fold3 <tibble [9 × 4]> <tibble [1 × 3]> <tibble>
##
## There were issues with some computations:
##
##   - Warning(s) x3: 7 columns were requested but there were 4 predictors in the data....
##
## Run `show_notes(.Last.tune.result)` for more information.
```

```
forest_fit_cv %>% collect_metrics()
```

```
## # A tibble: 9 × 6
##   .metric     .estimator  mean     n  std_err .config
##   <chr>       <chr>      <dbl> <int>    <dbl> <chr>
## 1 accuracy    binary     0.766     3 0.000341 Preprocessor1_Model1
## 2 f_meas      binary     0.801     3 0.000400 Preprocessor1_Model1
## 3 kap         binary     0.517     3 0.000719 Preprocessor1_Model1
## 4 mn_log_loss binary     0.476     3 0.000696 Preprocessor1_Model1
## 5 precision   binary     0.790     3 0.000474 Preprocessor1_Model1
## 6 recall      binary     0.812     3 0.000338 Preprocessor1_Model1
## 7 roc_auc     binary     0.839     3 0.000601 Preprocessor1_Model1
## 8 sens        binary     0.812     3 0.000338 Preprocessor1_Model1
## 9 spec        binary     0.703     3 0.000527 Preprocessor1_Model1
```

```
#plot(forest_fit_cv)
metric_to_use <- "roc_auc"

rf_best_params <- forest_fit_cv %>% select_best(metric_to_use)

# finally, re-fit the logistic regression on the entire training data
rf_fitted_workflow <- forest_workflow %>%
  finalize_workflow(rf_best_params) %>%
  fit(data_train)
```

```
## Warning: 7 columns were requested but there were 4 predictors in the data. 4
## will be used.
```

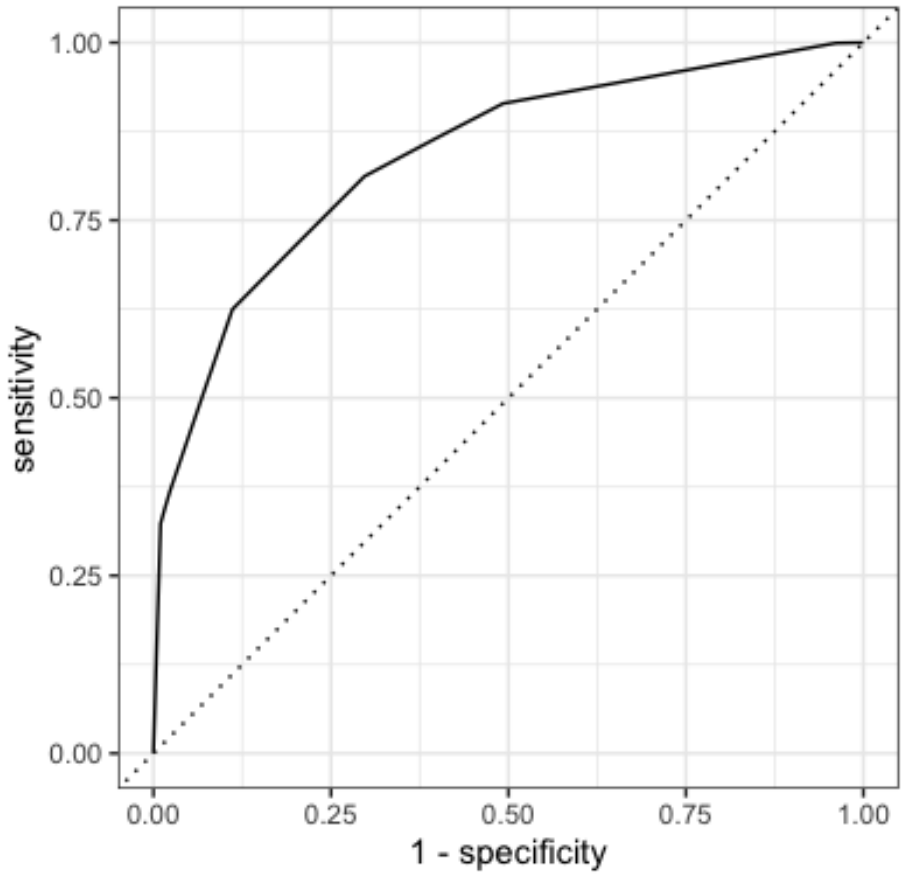
```
## Using best parameters for the final model of the random forest :

# first, we select the best hyperparameters based on the roc_auc metric
metric_to_use <- "roc_auc"
randfor_best_params <- forest_fit_cv %>% select_best(metric = metric_to_use)
randfor_best_params

## # A tibble: 1 × 1
##   .config
##   <chr>
## 1 Preprocessor1_Model1

# second, we update/finalize our model workflow using these parameters, and then re-fit on the
entire training data
forest_workflow_final <- forest_workflow %>%
  finalize_workflow(parameters = randfor_best_params)

randfor_fit_final <- forest_workflow %>%
  fit(data_train)

## Warning: 7 columns were requested but there were 4 predictors in the data. 4
## will be used.

# investigate the statistical significance of the random forest model
randfor_model_fitted <- randfor_fit_final %>% extract_fit_engine()
summary(randfor_model_fitted)

##                         Length Class         Mode
## predictions             931352 -none-        numeric
## num.trees                    1 -none-        numeric
## num.independent.variables    1 -none-        numeric
## mtry                         1 -none-        numeric
## min.node.size                1 -none-        numeric
## prediction.error             1 -none-        numeric
## forest                      10 ranger.forest list
## splitrule                    1 -none-        character
```

```
## treetype                   1 -none-      character
## call                      10 -none-      call
## importance.mode            1 -none-      character
## num.samples                1 -none-      numeric
## replace                    1 -none-      logical

# note that we only use the training data for now
randfor_pred_train <-
  predict(randfor_fit_final, data_train) %>%
  bind_cols(predict(randfor_fit_final, data_train, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_train %>% select(all_of(target_var)))

# note: you need to select the truth and estimate variables based on the column names of the
logreg_pred_train object
classification_metrics(data = randfor_pred_train,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1, # use the second outcome (1) as the level of interest
                       event_level = 'second')

## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.766
## 2 kap         binary         0.517
## 3 sens        binary         0.812
## 4 spec        binary         0.703
## 5 f_meas      binary         0.801
## 6 precision   binary         0.790
## 7 recall      binary         0.812
## 8 roc_auc     binary         0.839
## 9 mn_log_loss binary         0.476
```

```
conf_mat(data = randfor_pred_train,
         truth = vote_realized,
         estimate = .pred_class)
```

```
##           Truth
## Prediction      0      1
##          0 137740  50751
##          1  58245 218940
```

```
two_class_curve_train <- roc_curve(data = randfor_pred_train,
                                   truth = vote_realized,
                                   estimate = .pred_1,
                                   event_level = 'second')
autoplot(two_class_curve_train)
```



##Computing test predictions for the random forest model:

```
randfor_pred_test <-
  predict(randfor_fit_final, data_test) %>%
  bind_cols(predict(randfor_fit_final, data_test, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_test %>% select(all_of(target_var)))


classification_metrics(data = randfor_pred_test,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1,
                       event_level = 'second')
```

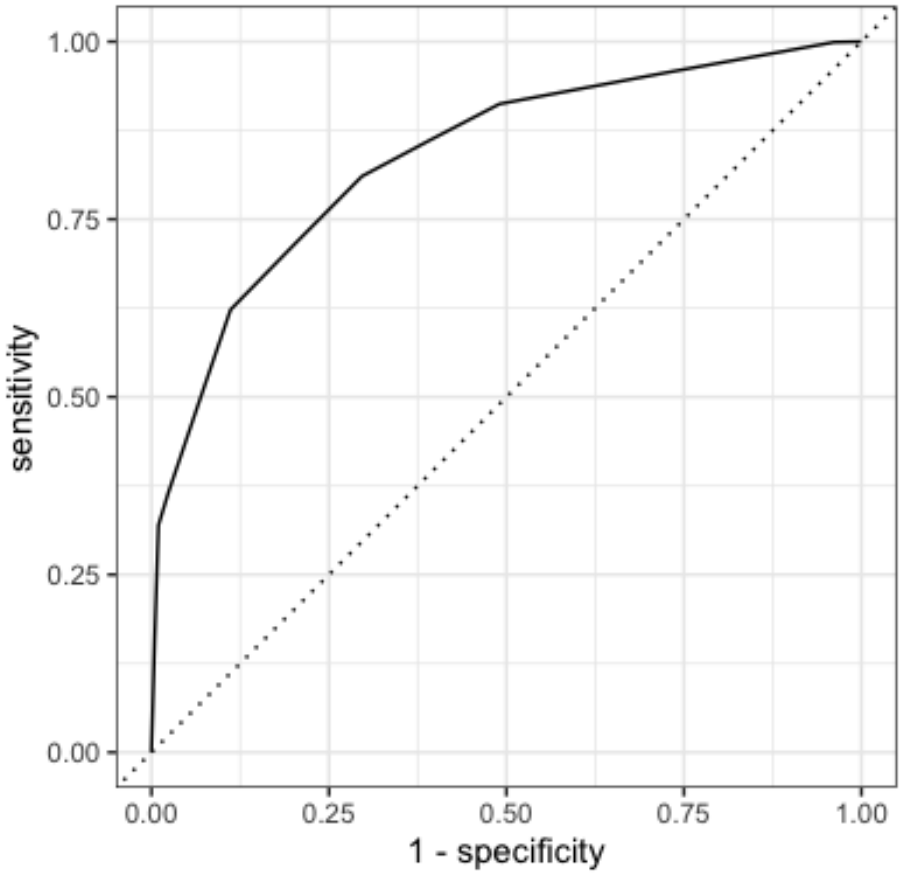
```
## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.765
## 2 kap         binary         0.516
## 3 sens        binary         0.811
## 4 spec        binary         0.703
## 5 f_meas      binary         0.800
## 6 precision   binary         0.790
## 7 recall      binary         0.811
## 8 roc_auc     binary         0.838
## 9 mn_log_loss binary         0.478
```

#######SECOND CLASS

```
conf_mat(data = randfor_pred_test,
         truth = vote_realized,
         estimate = .pred_class)
```

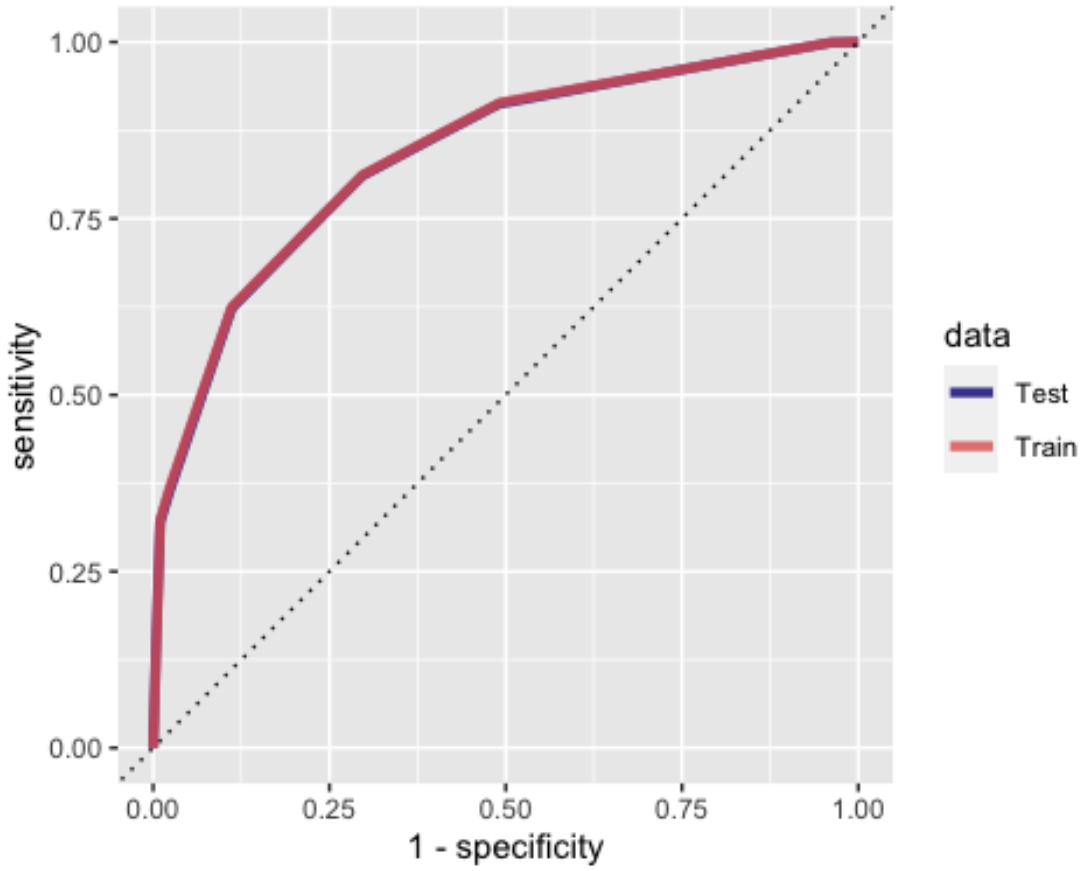
```
##            Truth
## Prediction     0     1
##          0 34465 12773
##          1 14532 54650
```

```
two_class_curve_test <- roc_curve(data = randfor_pred_test,
                                  truth = vote_realized,
                                  estimate = .pred_1,
                                  event_level = 'second')
autoplot(two_class_curve_test)
```




```
twoclass_train_test <- bind_rows(randfor_pred_train %>% mutate(data="Train"),
                                 randfor_pred_test %>% mutate(data="Test"))

twoclass_train_test <- bind_rows(two_class_curve_train %>% mutate(data="Train"),
                                 two_class_curve_test %>% mutate(data="Test"))

ggplot(data = twoclass_train_test,
       aes(x = 1 - specificity, y = sensitivity, color = data)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
```

```
coord_equal() +
scale_color_viridis_d(option = "plasma", end = .6)
```



3.3 Discussion

The change from sections 2 to 3 in removing the "Project" is showcasing the difference between the logistic regression and the random forest model. While the linear regression almost is not affected by the change, the random forest model changes from being a perfect prediction model to only a good prediction model. This is because, as previously assumed, the random forest was using a combination of variables including "project" to find a way to perfectly predict the result. This is presumably the combination of "Vote" and "Project".

Its new metrics for the Random Forest include a roc_auc score of 0.84 and an accuracy of 0.77. This makes it a good prediction model. Precision and Recall are 0.79 and 0.81, respectively. Here, specificity (0.70) is smaller than sensitivity (0.812)

The logistic regression is therefore still poorer with a roc_auc score of 0.7 and roc_auc of 0.76. Precision and Recall are both smaller (0.74 and 0.73), as are Sensitivity (0.73) and Specificity (0.65). As with the Random Forest, Specificity is smaller.

We shall be using the models from section 3 from now on, as they are more comparable. Also, a model without the "Project" variable is potentially more usable, as it can be deployed before the all the votes have been counted.

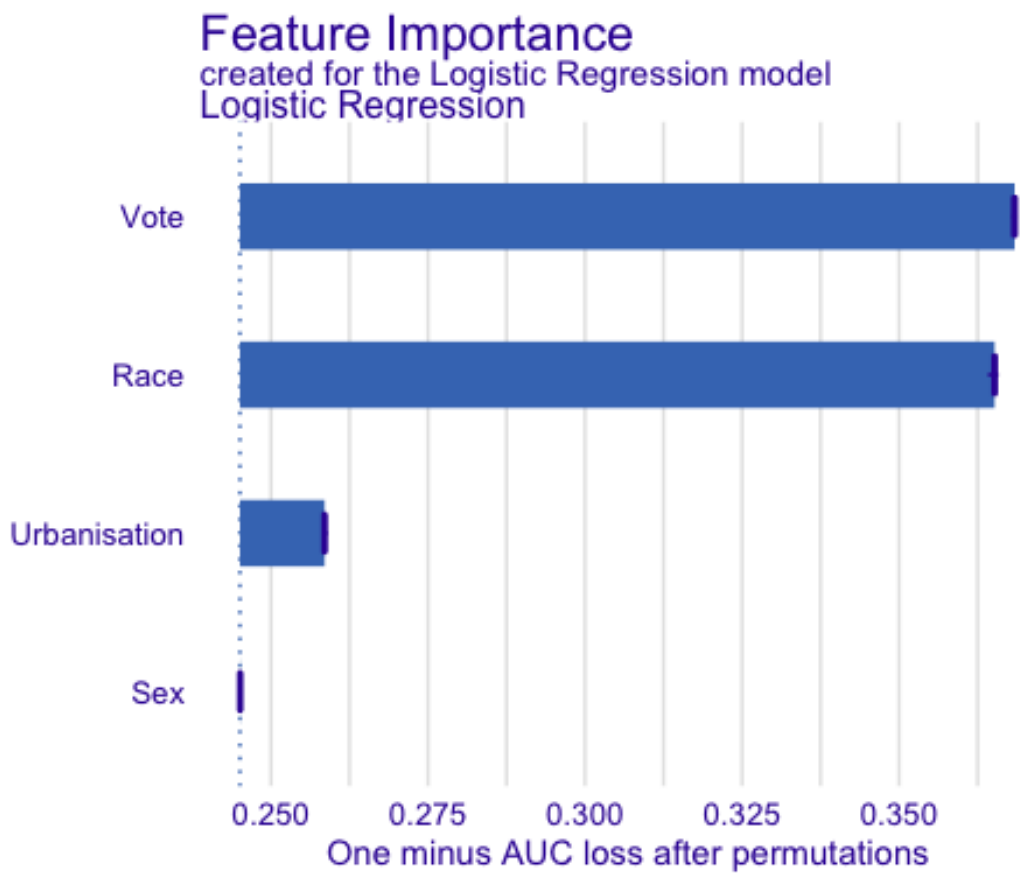
4. Explainability

4.1 Global

4.1.1 Logistic Regression Variable Importance

```
library(DALEX) # the general XAI package

## Welcome to DALEX (version: 2.4.2).
## Find examples and detailed introduction at: http://ema.drwhy.ai/

library(DALEXtra) # extensions of the XAI package to work with tidymodels

## Anaconda not found on your computer. Conda related functionality such as create_env.R
and condaenv and yml parameters from explain_scikitlearn will not be available

install_dependencies()

##
## Installation of the  ingredients  initiated.
##
## Installation of the  iBreakDown  initiated.
##
## Installation of the  ggpubr  initiated.

# note that the "form_pred()" function returns the names of the predictor variables used in the
model formula.
# this is a convenient way of subsetting the data based on the predictors used in the model
logreg_explainer <- explain_tidymodels(
  logreg_fitted_workflow,
  data = data_train %>% dplyr::select(form_pred(model_form)),
  y = ifelse(data_train$vote_realized == 0, 0, 1),
  # predict_function_target_column = 2,
  type = "classification",
```

```
  label = "Logistic Regression"
)

## Preparation of a new explainer is initiated
##   -> model label       :  Logistic Regression
##   -> data              :  465676  rows  4  cols
##   -> target variable   :  465676  values
##   -> predict function  :  yhat.workflow  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package tidymodels , ver. 1.0.0 , task classification (  default  )
##   -> model_info        :  type set to  classification
##   -> predicted values  :  numerical, min =  0.3227422 , mean =  0.5791387 , max =
0.9105085
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.9105085 , mean =  1.0068e-10 , max =  0.6772578
##   A new explainer has been created!

# the N=NULL means that all observations are used. the default is 1000
# Note that the XAI calculations are computationally very expensive, so for a larger dataset you
might need to restrict N to the default value


vip_logreg <- model_parts(logreg_explainer,
                          type = "variable_importance",
                          N=NULL)
plot(vip_logreg)
```



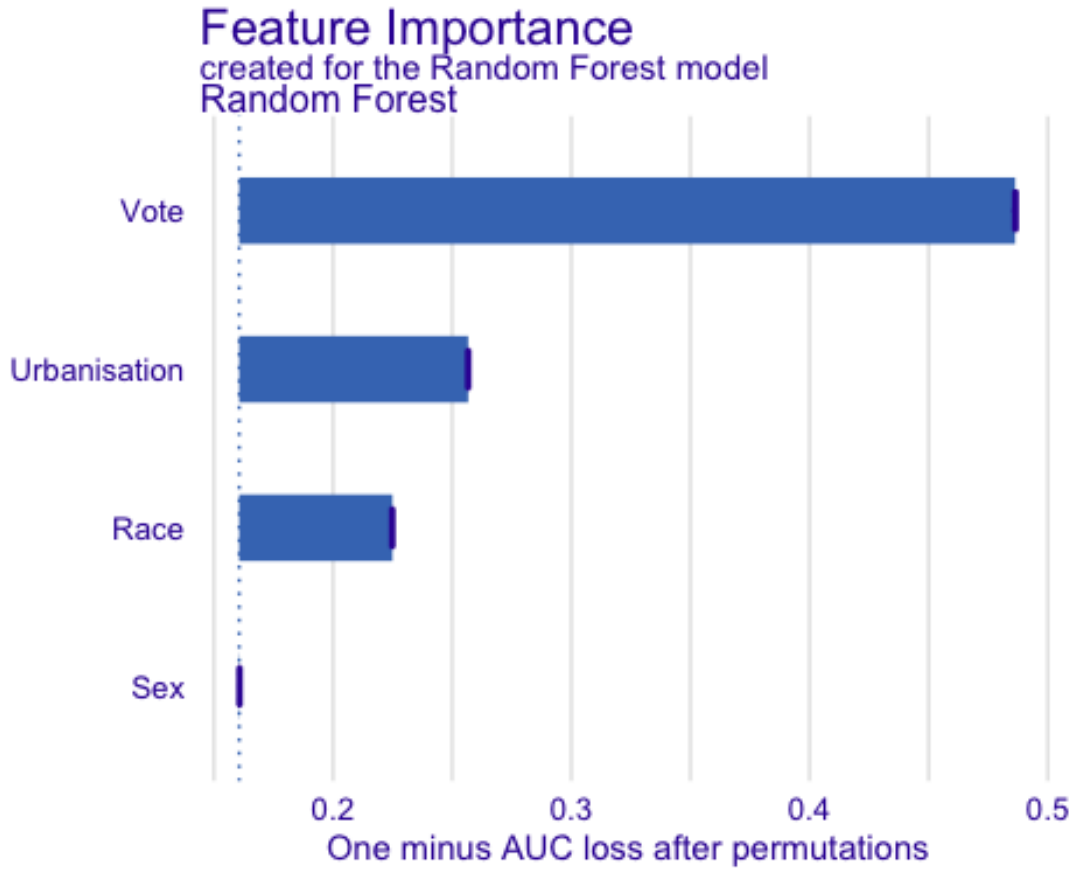
Discussion:

The logistic regression feature importance has two strongly important features: race and vote with over 0.35 each. Urbanization plays a smaller part with slightly over 0.25. We can assume that model is mostly relying on these three variables to predict the dependent variable. Sex seems to have little importance.

4.1.2 Random Forest Variable Importance

```
## Preparation of a new explainer is initiated
##   -> model label       :  Random Forest
##   -> data              :  465676  rows  4  cols
##   -> target variable   :  465676  values
##   -> predict function  :  yhat.workflow  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package tidymodels , ver. 1.0.0 , task classification (  default  )
##   -> model_info        :  type set to  classification
##   -> predicted values  :  numerical, min =  0.02125303 , mean =  0.5791198 , max =
0.9779035
```

```
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.9779035 , mean =  1.892998e-05 , max =
0.978747
##   A new explainer has been created!
```



Interpreting the global feature importance for Random Forest:

It seems that the most important feature when it comes to one's preferred project to be realized is the actual individual vote, which makes sense. Together with Sex having no effect (because it was randomly generated and had no effect on the vote or the outcome), this makes for a good sanity check for the feature importance - it seems to make sense from the data and logic of the population.

The other two variables, urbanization, and race, seem to have mediocre importance of about 0.25, which is a considerable amount. Urbanization is important because it directly relates to Province, which in turn correlates with the realized project. Race was coded to have an impact on the vote, hence it is significant as well. We can thus validate the correctness of the feature important based on the subject matter knowledge. Further, we can demonstrate that one's race is an important factor when it comes to the realization of one's project, which is important when we look at fairness.

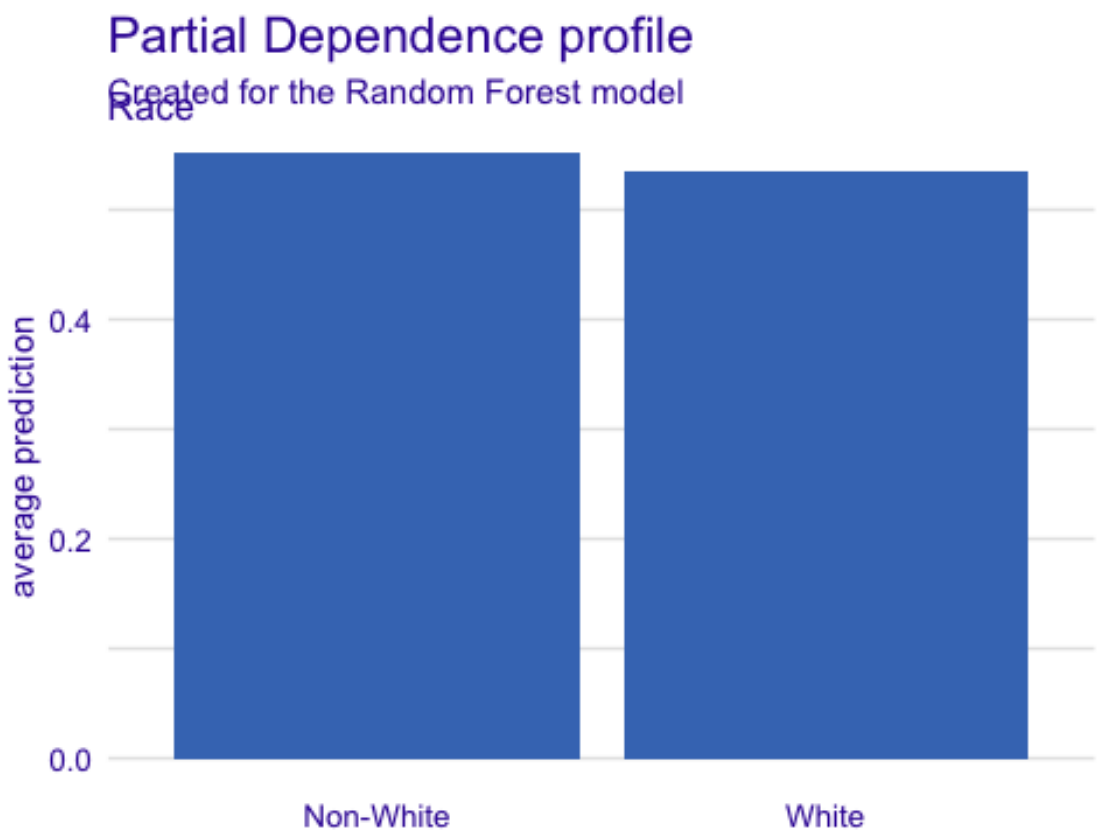
Comparing the feature importance:

Comparing the results here to the results of the feature importance of the logistic regression, we can see that urbanization is much more important. My guess is that the inherent qualities of the random forest allow the model to make the connection between urbanization and province, and ultimately, project.

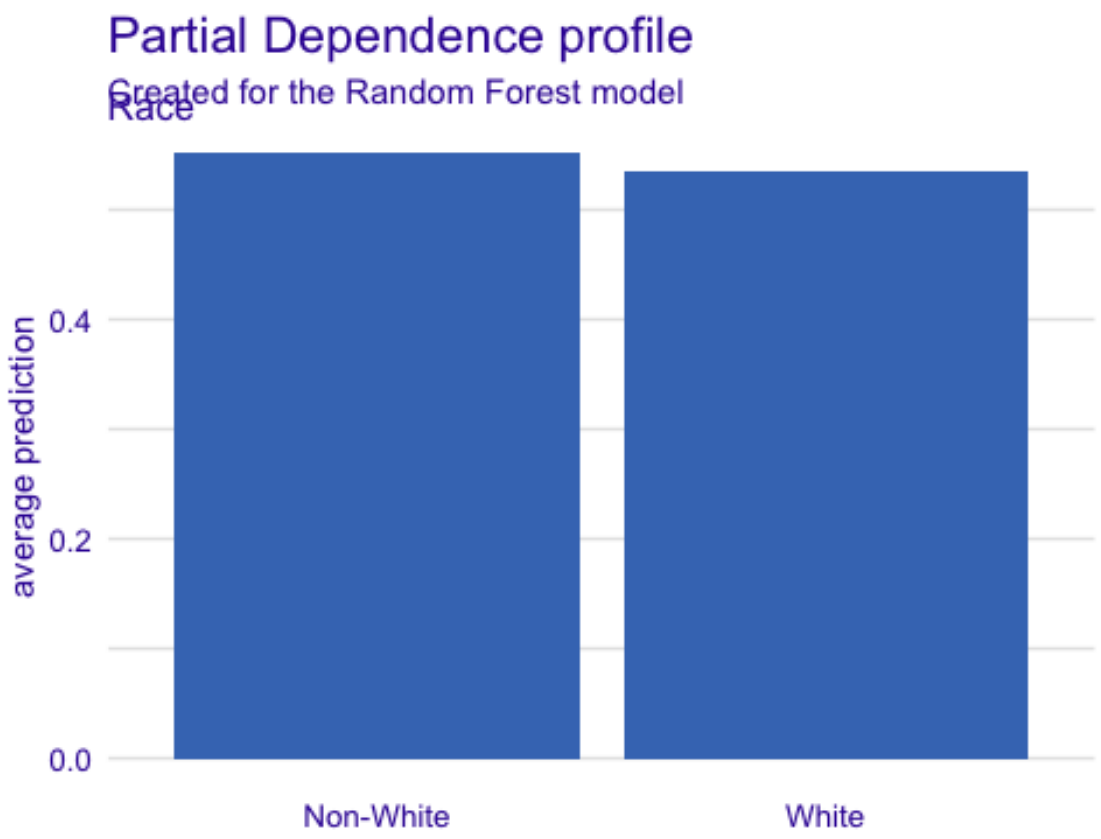
Further, we can see that race has a much smaller impact than in the logistic regression. Again, I assume that the random forest was able to see the connection between race and vote and actually attribute much more to the vote than the logistic regression. Sex has little significance in both models, which again makes sense when examining how the data points were generated.

4.1.3 Partial Dependence Plots (Global Interpretability)

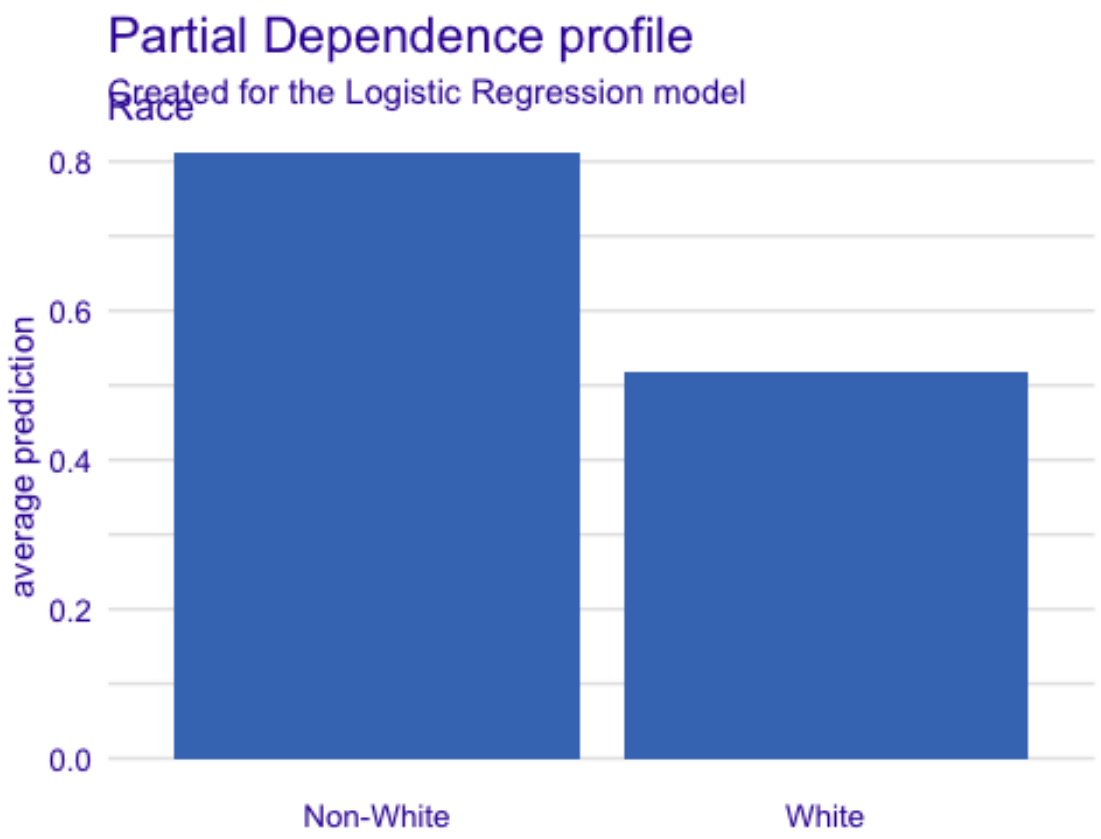
```
pdp_rf <- model_profile(rf_explainer,
                        variables = c("Race"),
                        N=NULL,
                        type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(pdp_rf)
```



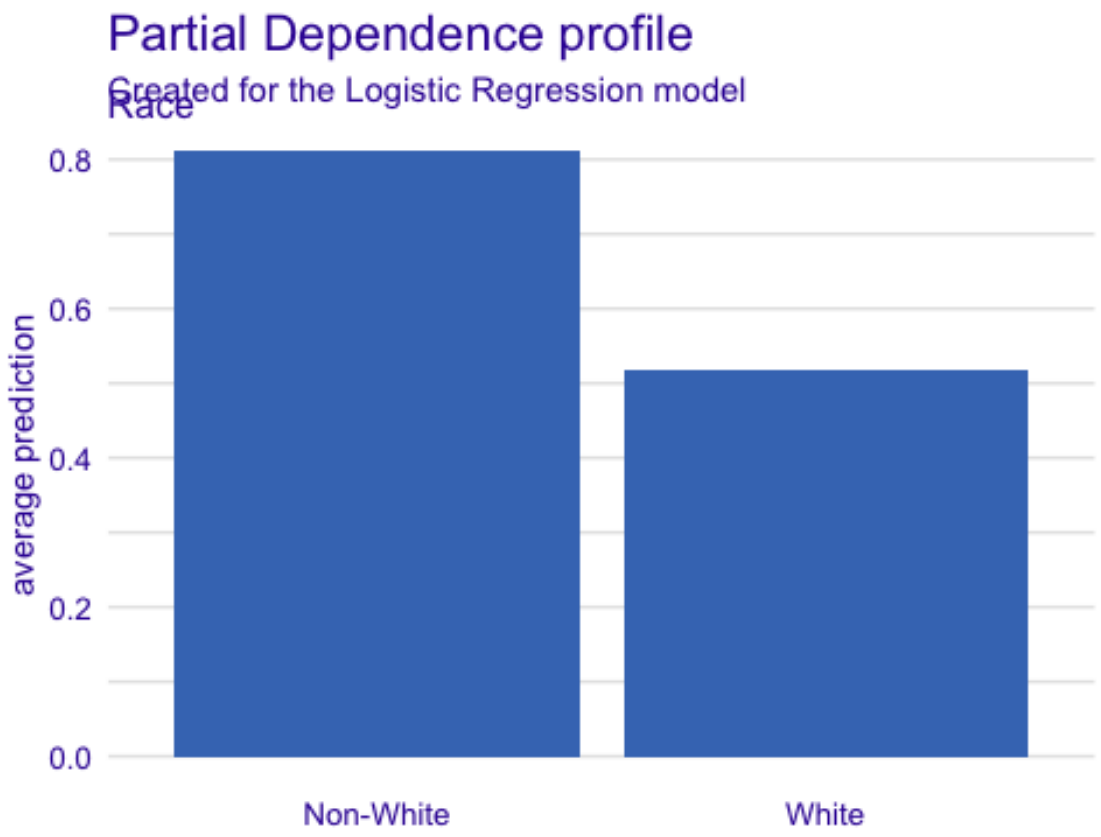

```
var_rf <- variable_profile(rf_explainer,
                           variables = c("Race"),
                           N=NULL,
                           type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(var_rf)
```




```
pdp_rf <- model_profile(logreg_explainer,
                        variables = c("Race"),
                        N=NULL,
                        type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(pdp_rf)
```



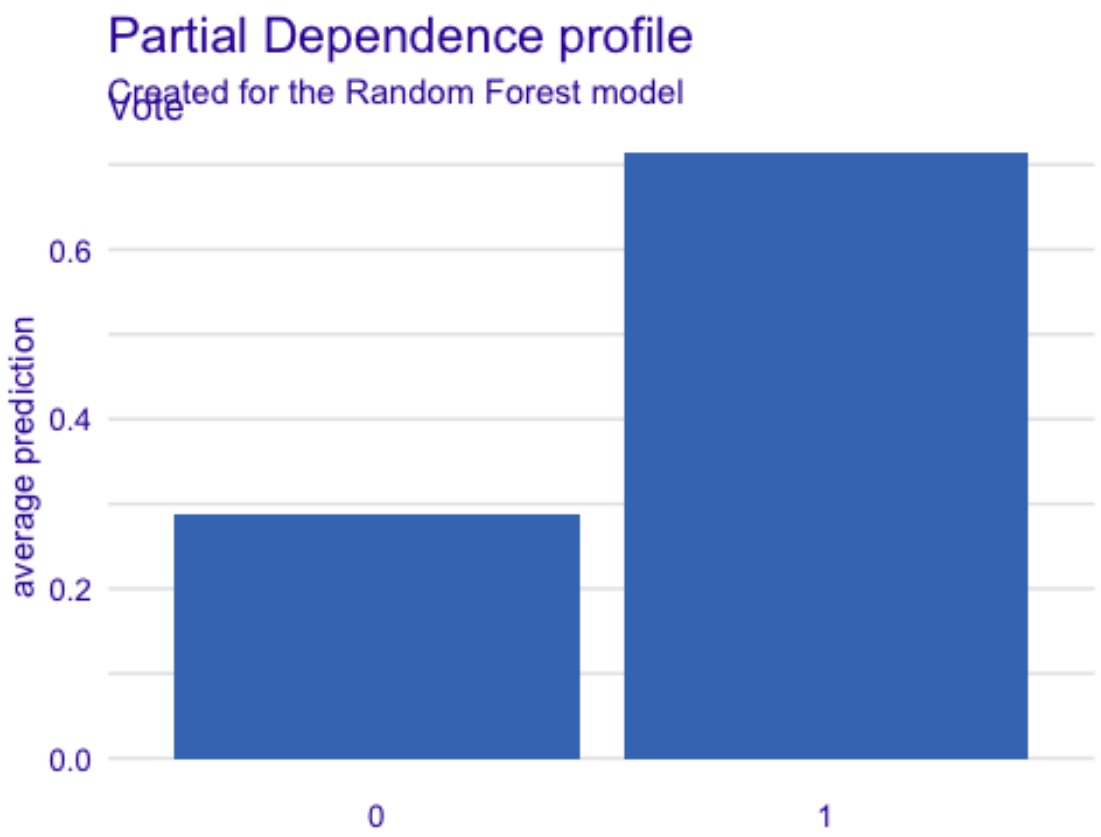

```
var_rf <- variable_profile(logreg_explainer,
                           variables = c("Race"),
                           N=NULL,
                           type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(var_rf)
```



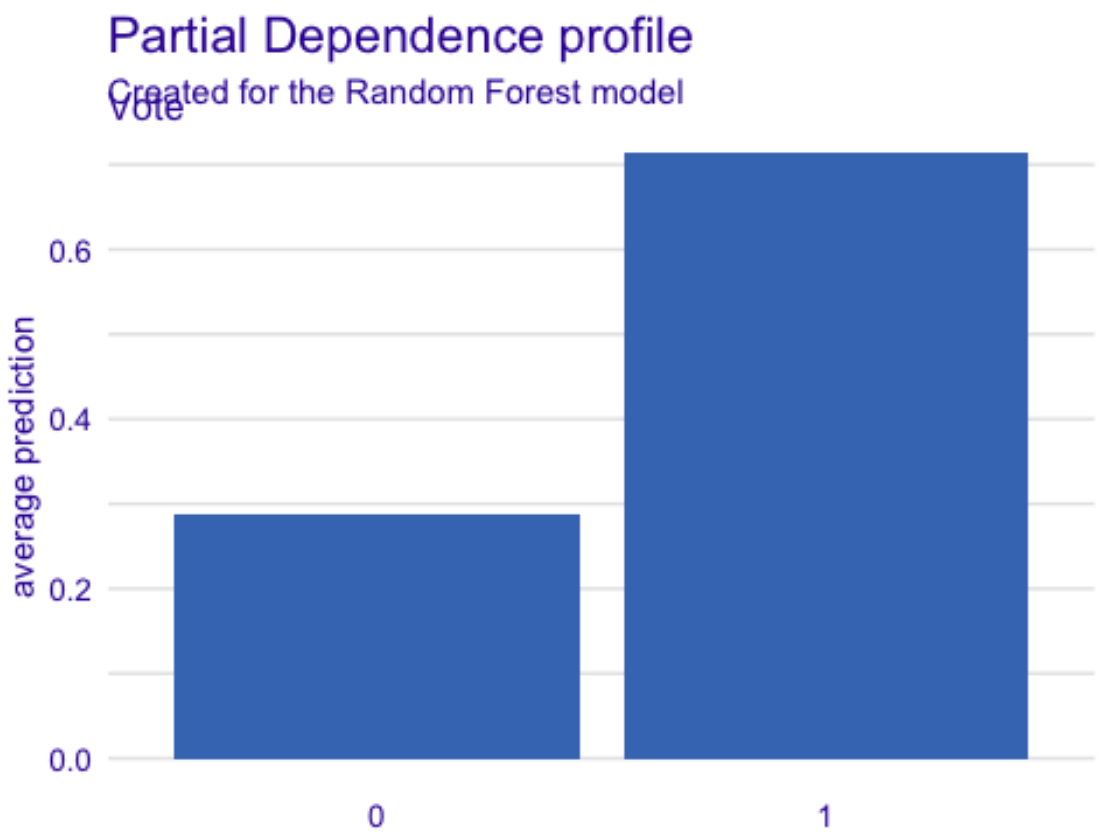
For the logistic regression, non-white has a larger chance of having vote_realized = 1 (0.8 vs 0.5).

For the random forest, We see that non-white has a very slightly larger average prediction, but it is about the same.

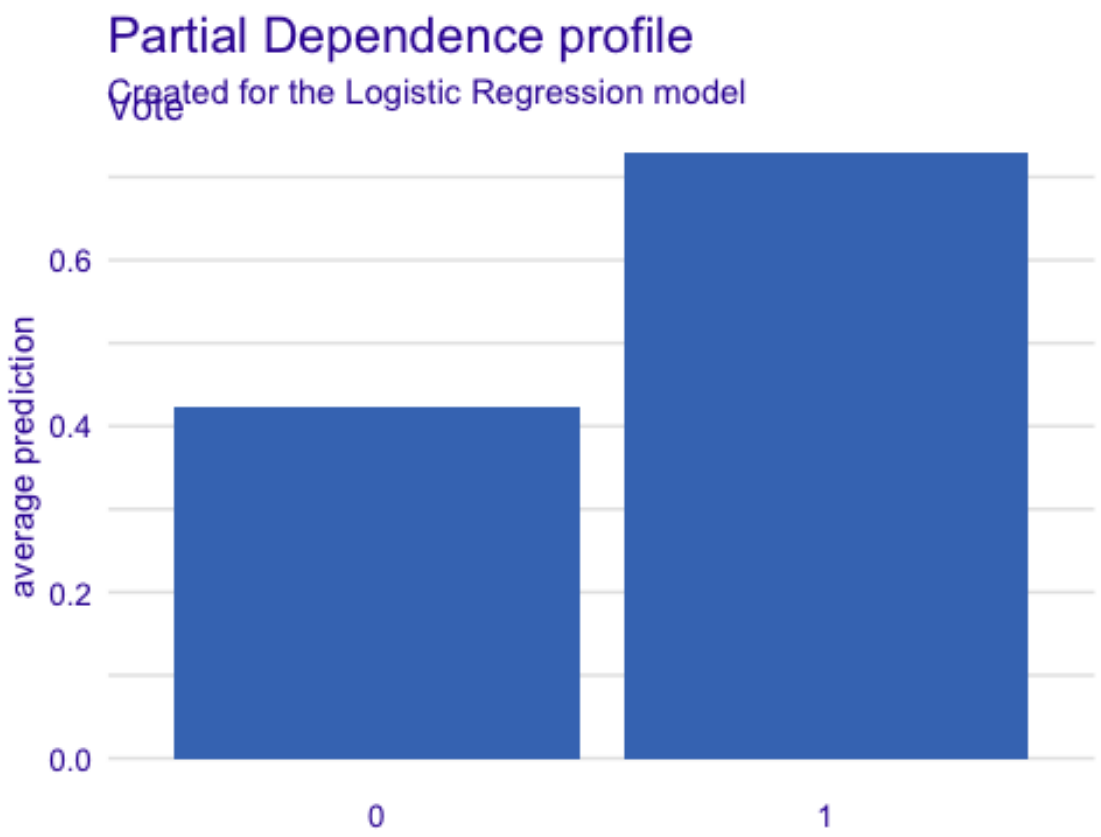
```
pdp_rf <- model_profile(rf_explainer,
                        variables = c("Vote"),
                        N=NULL,
                        type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(pdp_rf)
```



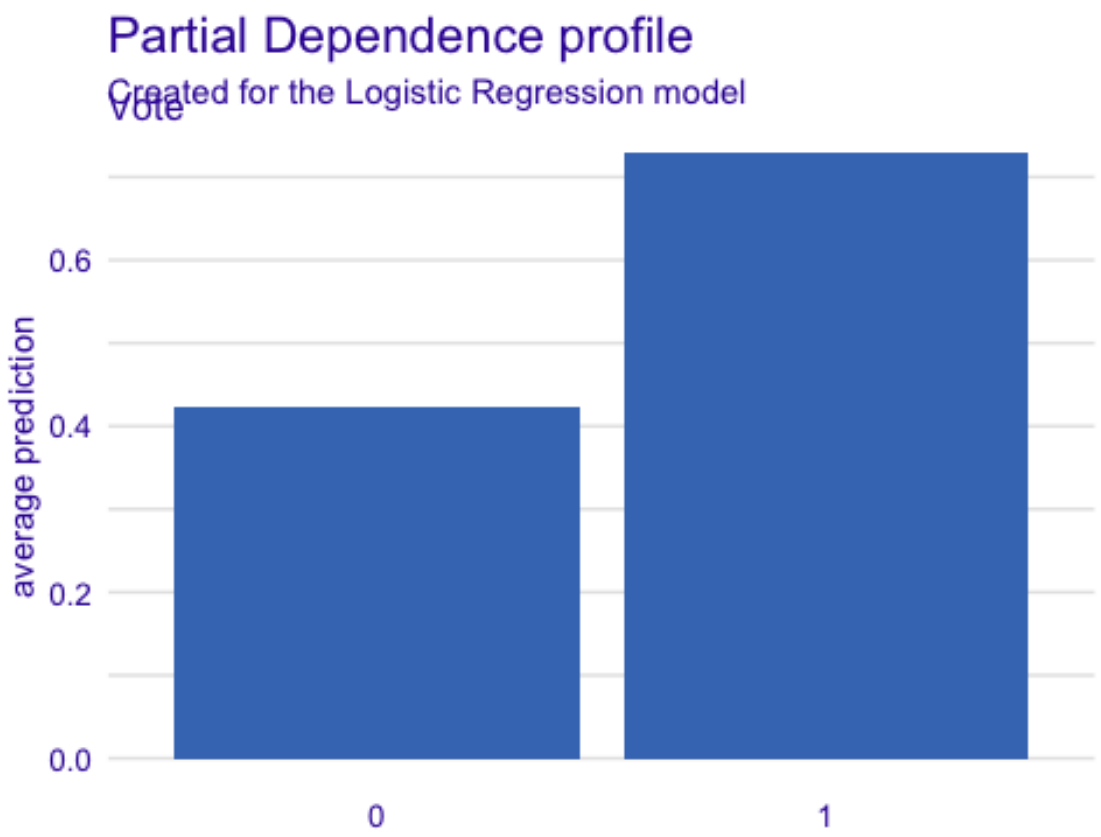

```
var_rf <- variable_profile(rf_explainer,
                           variables = c("Vote"),
                           N=NULL,
                           type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(var_rf)
```




```
pdp_rf <- model_profile(logreg_explainer,
                        variables = c("Vote"),
                        N=NULL,
                        type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(pdp_rf)
```




```
var_rf <- variable_profile(logreg_explainer,
                           variables = c("Vote"),
                           N=NULL,
                           type="partial")

## 'variable_type' changed to 'categorical' due to lack of numerical variables.

plot(var_rf)
```



We see that voting 1 has a higher average change of getting one's project realized, which makes sense because more people live in Provinces which have a 1 for project. This goes for both models.

```
pdp_rf <- model_profile(rf_explainer,
                        variables = c("Urbanisation"),
                        N=NULL,
                        type="partial")
plot(pdp_rf)

var_rf <- variable_profile(rf_explainer,
                        variables = c("Urbanisation

                                      "),
                        N=NULL,
                        type="partial")
plot(var_rf)
```

```
pdp_rf <- model_profile(logreg_explainer,
                        variables = c("Urbanisation"),
                        N=NULL,
                        type="partial")
plot(pdp_rf)

var_rf <- variable_profile(logreg_explainer,
                        variables = c("Urbanisation"),
                        N=NULL,
                        type="partial")
plot(var_rf)
```

We see that switching from rural or urban has little impact on the result for both models.

4.1.4 Ale Plots

Ale plots were attempted, but never compute, code below

```
#ale_rf <- model_profile(logreg_explainer,
#                        variables = c("Race"),
#                        N=NULL,
#                        type="accumulated")
#plot(ale_rf)
#
#{r}
#ale_rf <- model_profile(rf_explainer,
#                        variables = c("Race"),
#                        N=NULL,
#                        type="accumulated")
#plot(ale_rf)
```

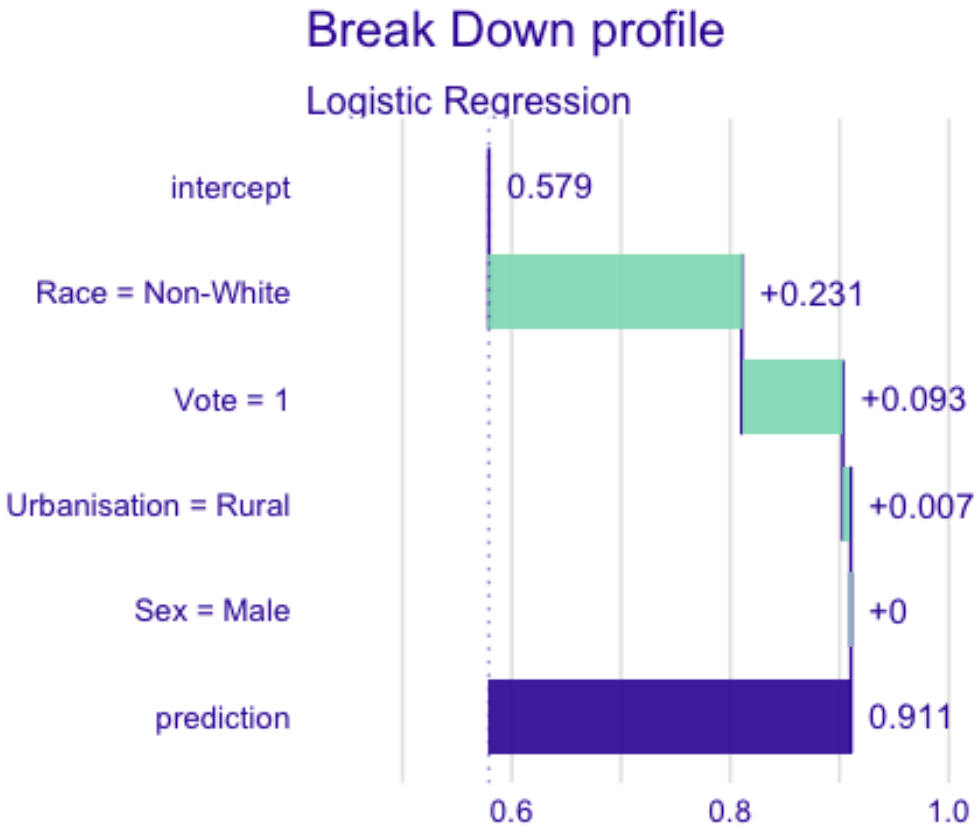
4.2 Local

4.2.1 Break Down Plots

Logistic Regression

```
# here, we use a separate observation
new_observation <- as.data.frame(data_test[1,])
```

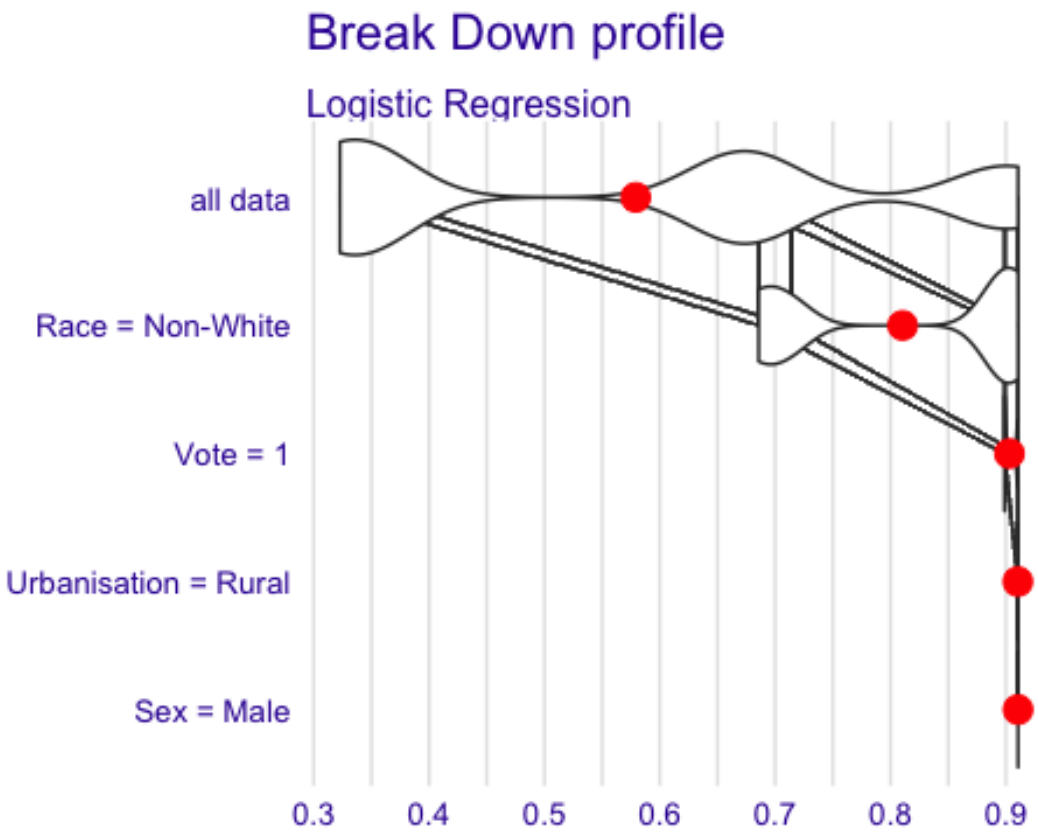
```
# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(logreg_explainer,
                                  new_observation = new_observation,
                                  type= "break_down",
                                  keep_distributions = TRUE)
plot(breakdown)
```



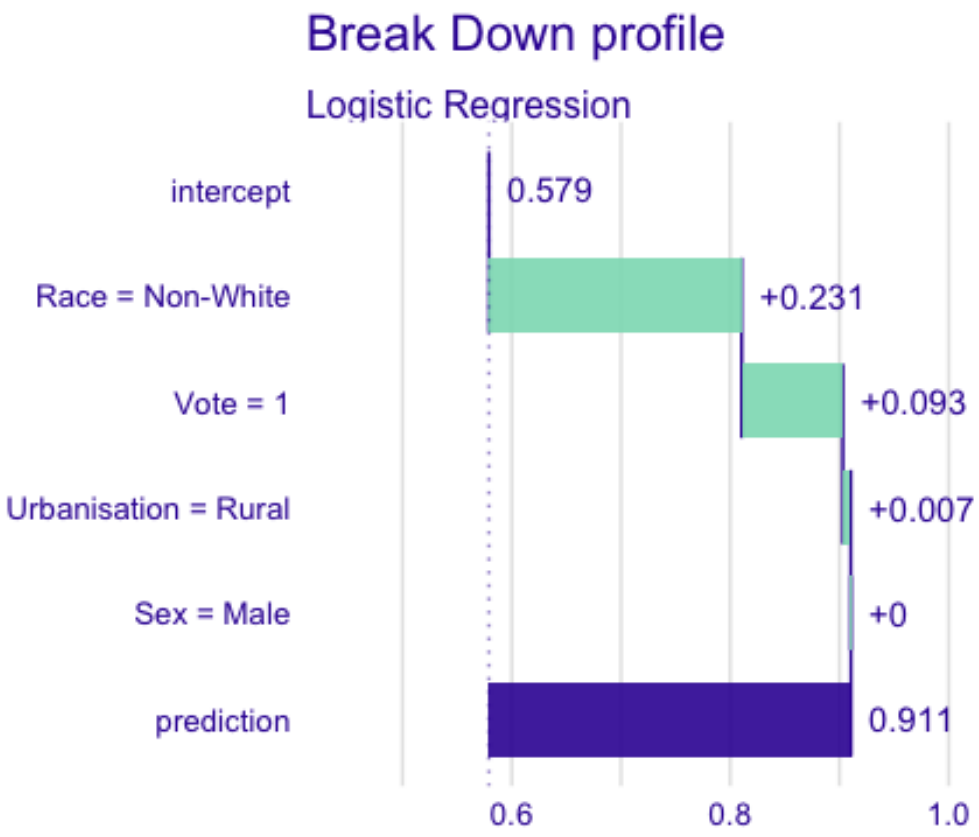

```
plot(breakdown, plot_distributions = TRUE) #we can also plot the general distribution of all variables in the background
```

```
## Warning: The `fun.y` argument of `stat_summary()` is deprecated as of ggplot2 3.3.0.
## ℹ Please use the `fun` argument instead.
## ℹ The deprecated feature was likely used in the iBreakDown package.
##   Please report the issue at
##
```

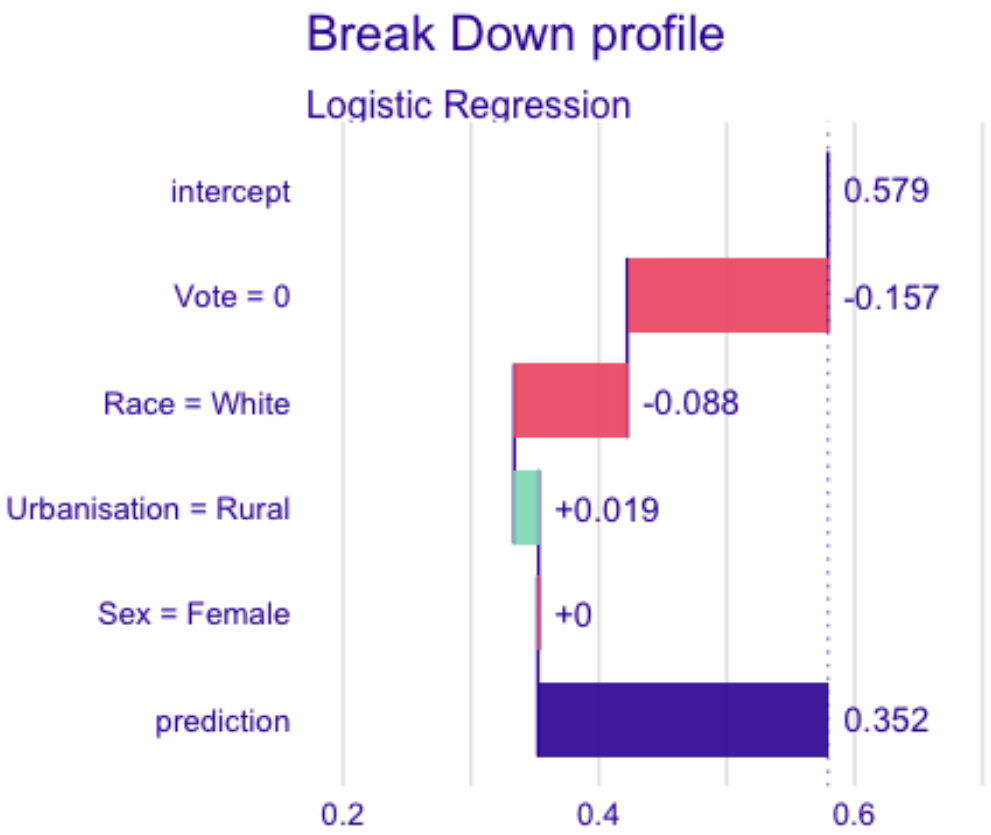
<]8;;https://github.com/ModelOriented/iBreakDown/issueshttps://github.com/ModelOriented/iBreakDown/issues]8;;>.



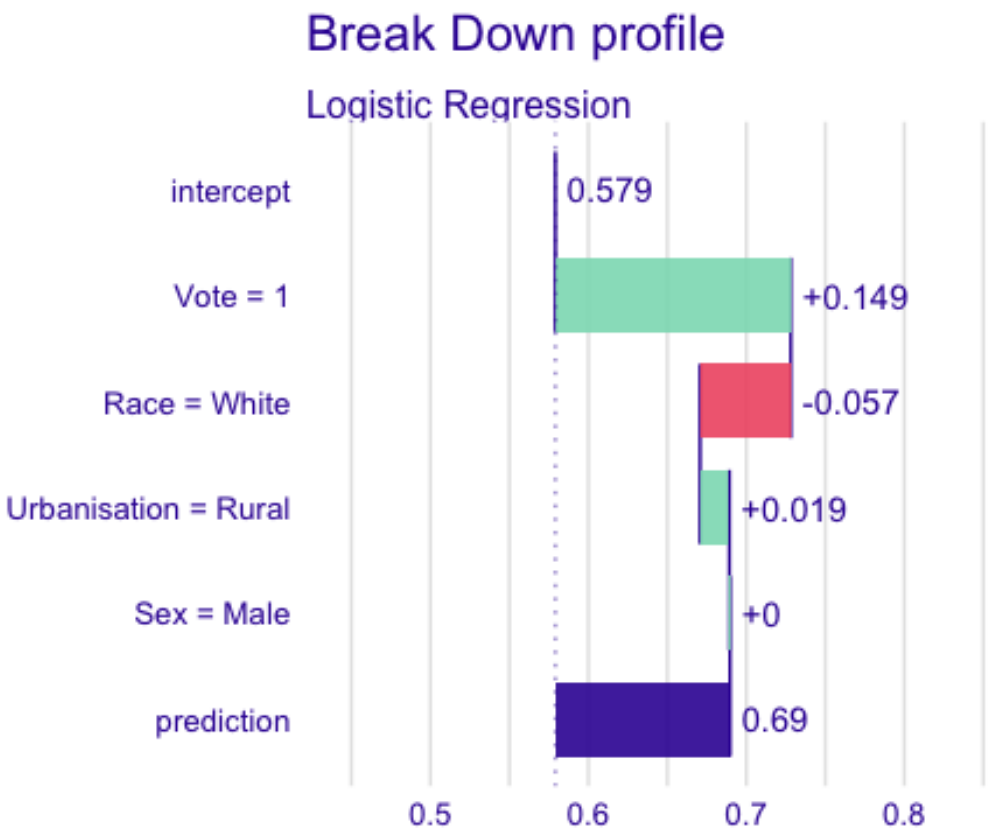

```
# again for another observation
new_observation <- as.data.frame(data_test[3,])

# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(logreg_explainer,
                          new_observation = new_observation,
                          type= "break_down")
plot(breakdown)
```




```
# again for another observation
new_observation <- as.data.frame(data_test[48326,])

# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(logreg_explainer,
                                  new_observation = new_observation,
                                  type= "break_down")
plot(breakdown)
```




```
# again for another observation
new_observation <- as.data.frame(data_test[48327,])

# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(logreg_explainer,
                                  new_observation = new_observation,
                                  type= "break_down")
plot(breakdown)
```



The intercept is at 58% which is the start-off point. We can see that for the first instance, Race = Non-White plays a large role with + 23% for the project to be realized. Vote = 1 provides an additional 9%. Urbanization and Sex play almost no role. The person ends up with a 91% change for project_realized.

This is confirmed in the second observation, where the person voted 0, thus decreasing the chance for project_realized. by 12%. The rest stayed about the same, so the chance is still 71%.

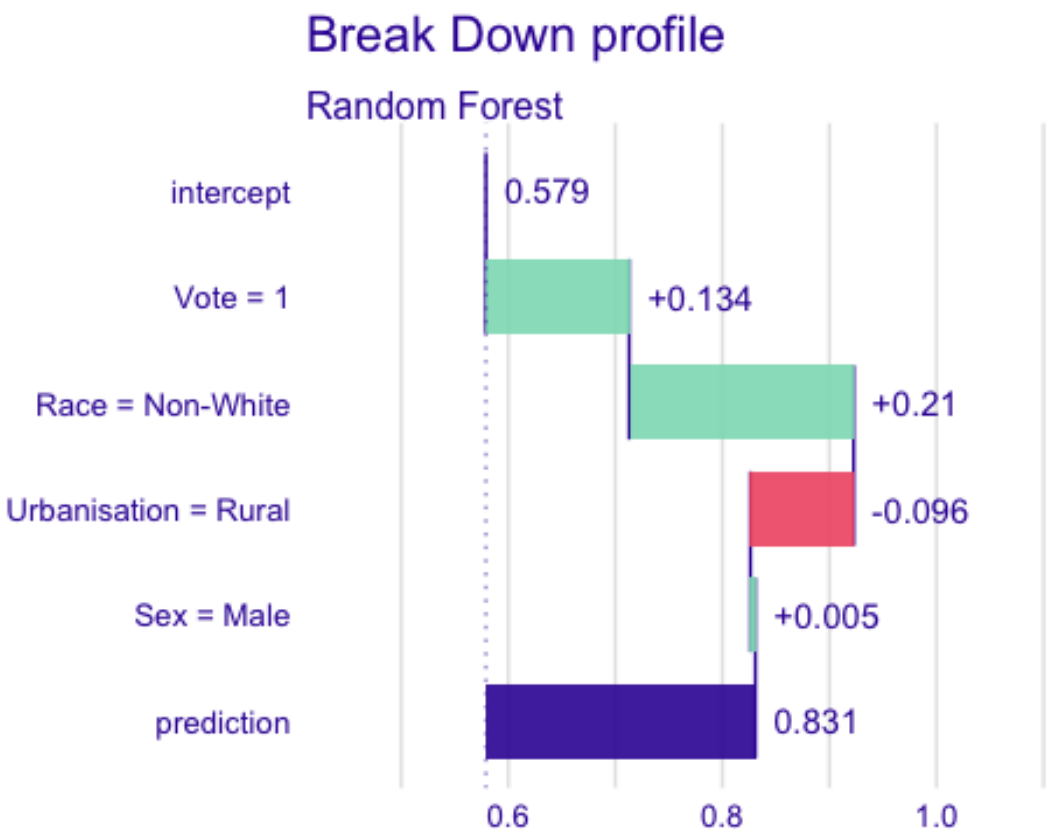
With the third example, we can see that Vote = 1 plays the largest role here with +15%. Race plays a much smaller role here with only -6% This is much smaller than before. The total prediction is 69%.

In the last example, we can again see the large malus of voting 0 (-16%). Race = White also means -9% here, leaving this person only with a 36% prediction, the lowest one by far from the four examined. This is because both negative factors were combined.

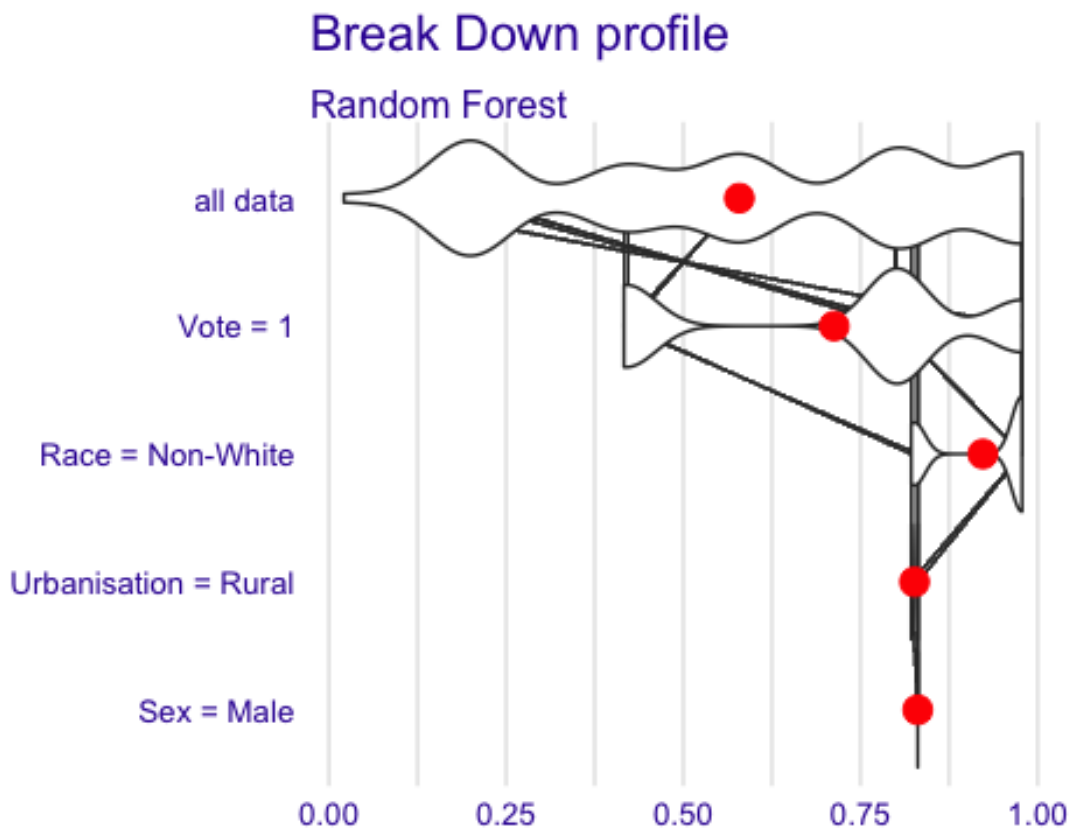
Random Forest

```
# here, we use a separate observation
new_observation <- as.data.frame(data_test[1,])
```

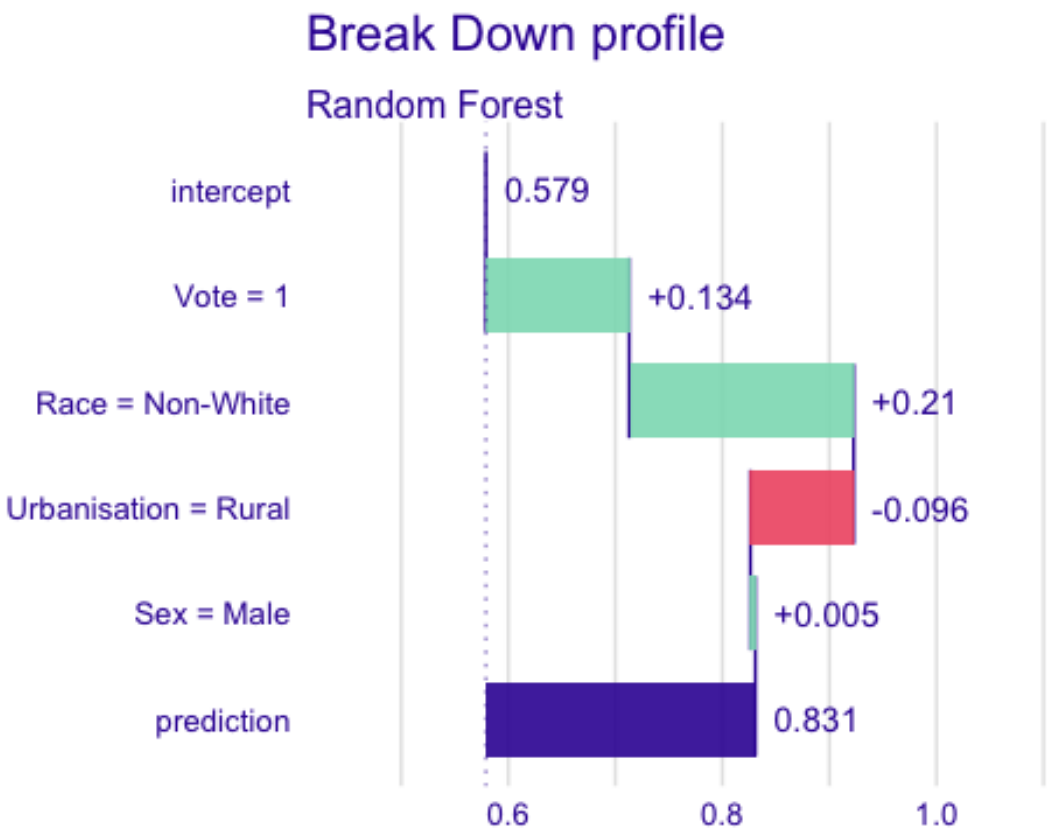
```
# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(rf_explainer,
                                  new_observation = new_observation,
                                  type= "break_down",
                                  keep_distributions = TRUE)
plot(breakdown)
```



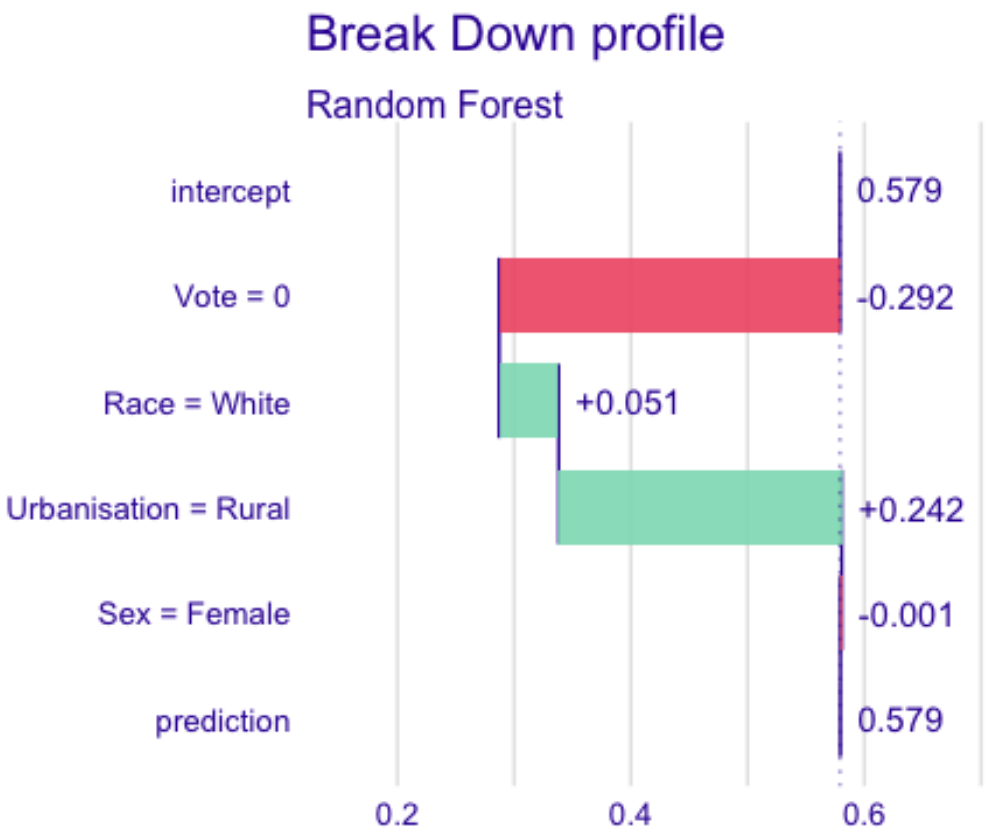

```
plot(breakdown, plot_distributions = TRUE) #we can also plot the general distribution of all variables in the background
```



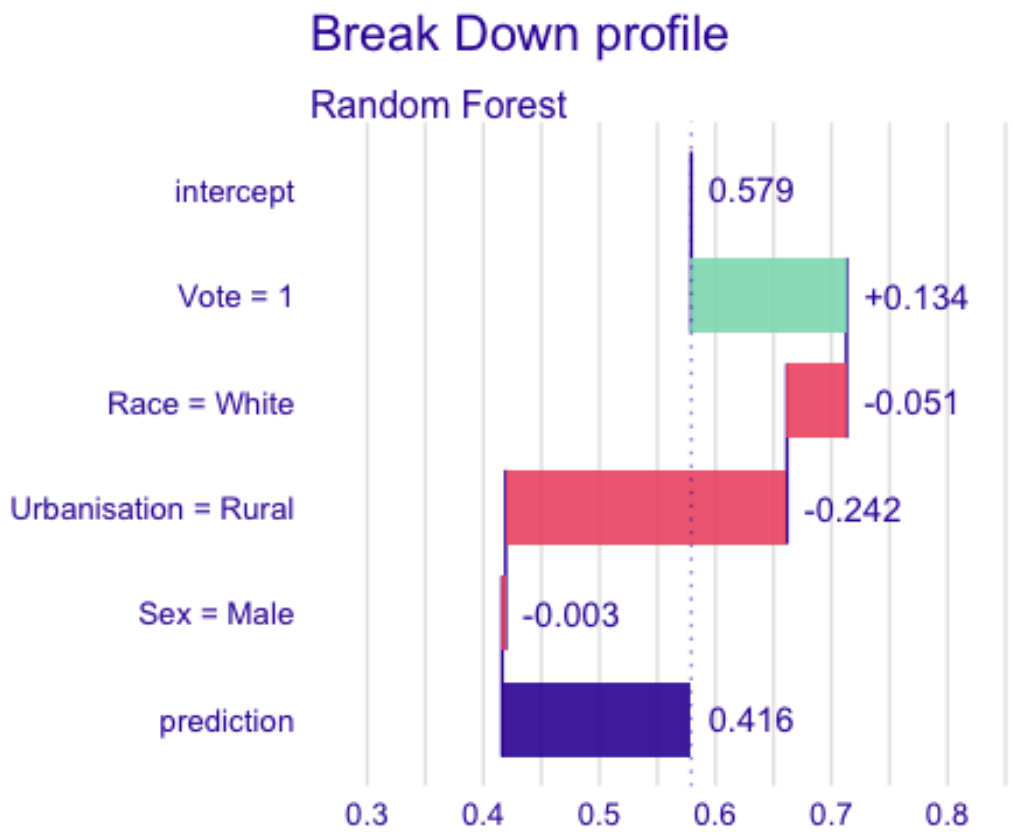

```
# again for another observation
new_observation <- as.data.frame(data_test[3,])

# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(rf_explainer,
                                  new_observation = new_observation,
                                  type= "break_down")
plot(breakdown)
```




```
# again for another observation
new_observation <- as.data.frame(data_test[48326,])

# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(rf_explainer,
                                  new_observation = new_observation,
                                  type= "break_down")
plot(breakdown)
```




```
# again for another observation
new_observation <- as.data.frame(data_test[48327,])

# first, calculate the breakdown plots for a specific observation
breakdown <- DALEX::predict_parts(rf_explainer,
                                  new_observation = new_observation,
                                  type= "break_down")
plot(breakdown)
```



We can see in the first example that vote = 1 and race = non-white still contribute a large positive towards the prediction. Urbanization = Rural plays a much bigger part with almost -10% here. Again, we can assume that the random forest has understood the importance of that metric more. Further, the prediction is only 82%, which is almost 10 percentage points lower than the one from the logistic regression.

Interestingly, in the second example, we see Race = Non_White being counted as a negative for the first time, with quite a large amount (-22%). Vote = 0 remains a strong negative with almost -30%. Urbanization plays a bigger role here again with 8%. In total, the prediction is only 15%, which is a sharp contrast for the same observation with the logistic regression (71%).

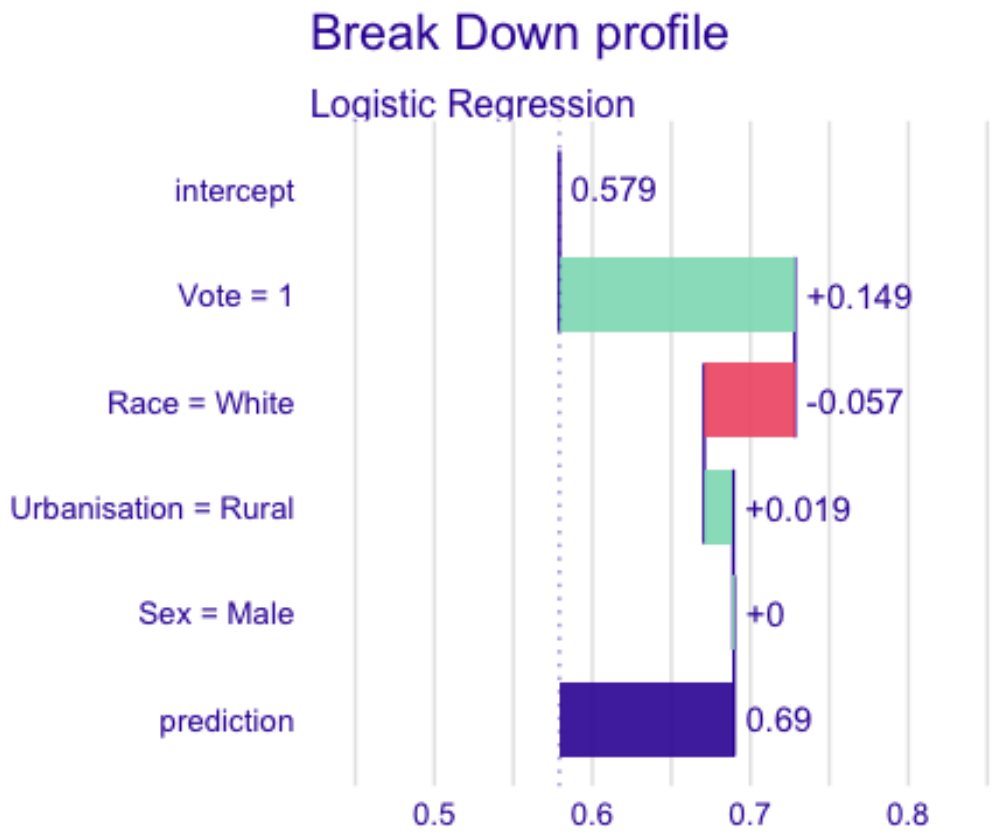
The third example is also surprising, because Urbanization = rural is weighed with -24%. Race = White, by contrast, only impacts by -5%. The prediction is 42%, which is 27 percentage points less than during the logistic regression.

In the last example, we can see a very large negative of vote = 0 again with 29%. Race plays a small role with only 5%. Interestingly, Race = White has a positive impact here. As a sharp contrast to the third example above, Urbanisation = Rural is +24% here. The total prediction is 58%, the only time the prediction of the random forest is higher than for the logistic regression.

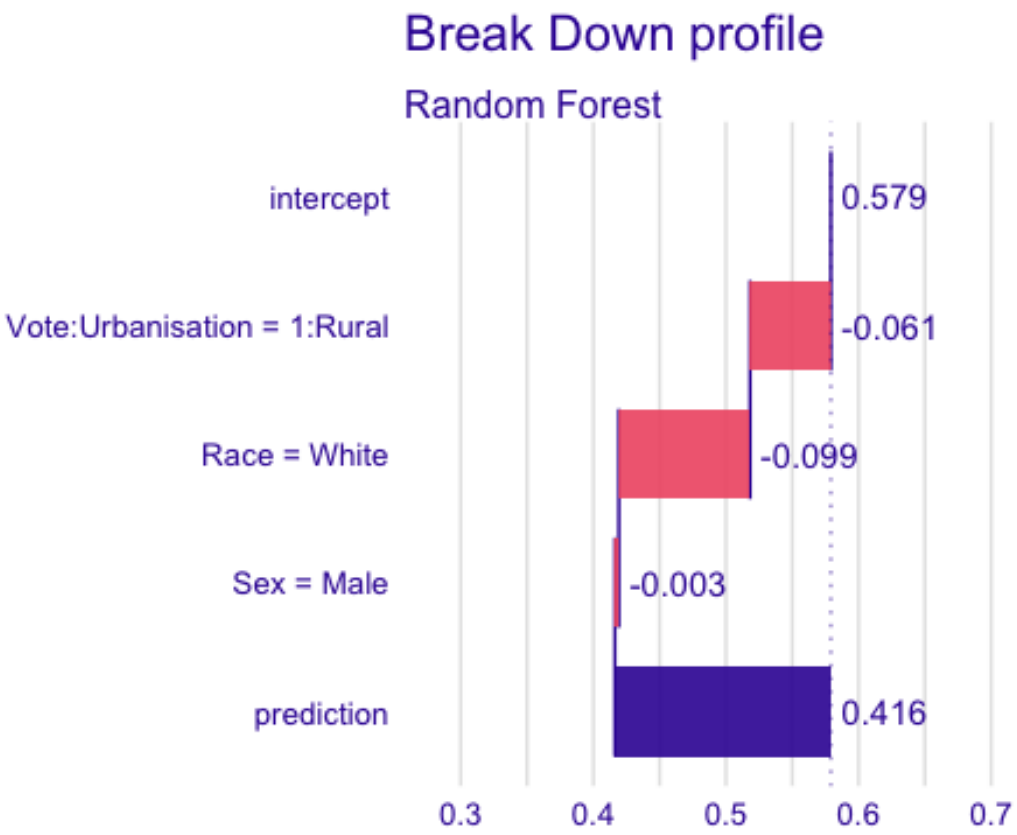
Overall, we can see that the random forest model puts more emphasis on the urbanization variable. It further is more flexible on the impact of the race variable and judges it depending on the other variables. The random forest predictions are less extreme compared to the logistic regression.

4.2.2 Interaction Plots

```
# first, calculate the breakdown plots for a specific observation
breakdown_interactions <- DALEX::predict_parts_break_down_interactions(logreg_explainer,
                          new_observation = new_observation,
                          type= "break_down_interactions")
plot(breakdown_interactions)
```




```
# first, calculate the breakdown plots for a specific observation
breakdown_interactions <- DALEX::predict_parts_break_down_interactions(rf_explainer,
                          new_observation = new_observation,
                          type= "break_down_interactions")
plot(breakdown_interactions)
```

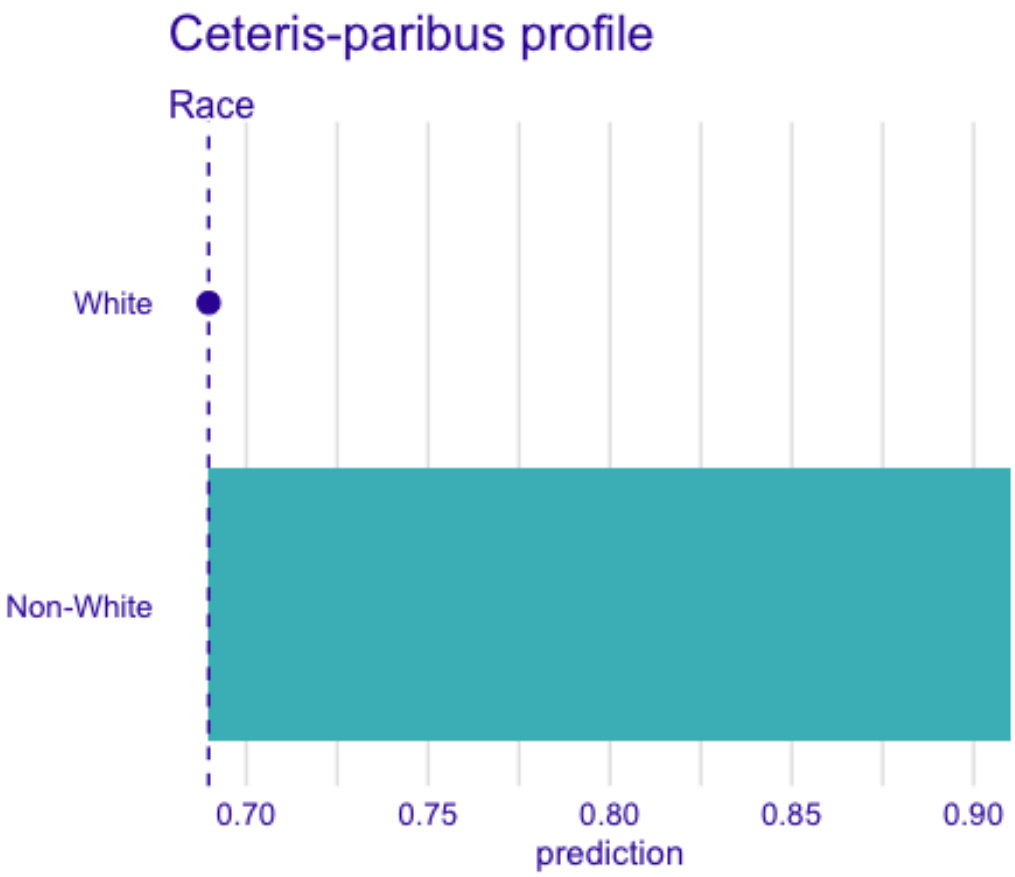


4.2.3 Ceteris-Paribus Plots

```
cp_rf <- predict_profile(explainer = logreg_explainer,
                         new_observation = new_observation)

cp_rf

## Top profiles    :
##      Urbanisation      Race     Sex Vote    _yhat_      _vname_ _ids_
## 1           Rural     White    Male    1 0.6896600 Urbanisation     1
## 1.1         Urban     White    Male    1 0.6605368 Urbanisation     1
## 11          Rural Non-White    Male    1 0.9105085         Race     1
## 1.11        Rural     White    Male    1 0.6896600         Race     1
## 12          Rural     White Female    1 0.6888268          Sex     1
## 1.12        Rural     White    Male    1 0.6896600          Sex     1
##                 _label_
## 1    Logistic Regression
```

```
## 1.1  Logistic Regression
## 11   Logistic Regression
## 1.11 Logistic Regression
## 12   Logistic Regression
## 1.12 Logistic Regression
##
##
## Top observations:
##   Urbanisation  Race  Sex Vote  _yhat_             _label_ _ids_
## 1        Rural White Male    1 0.68966 Logistic Regression     1
```

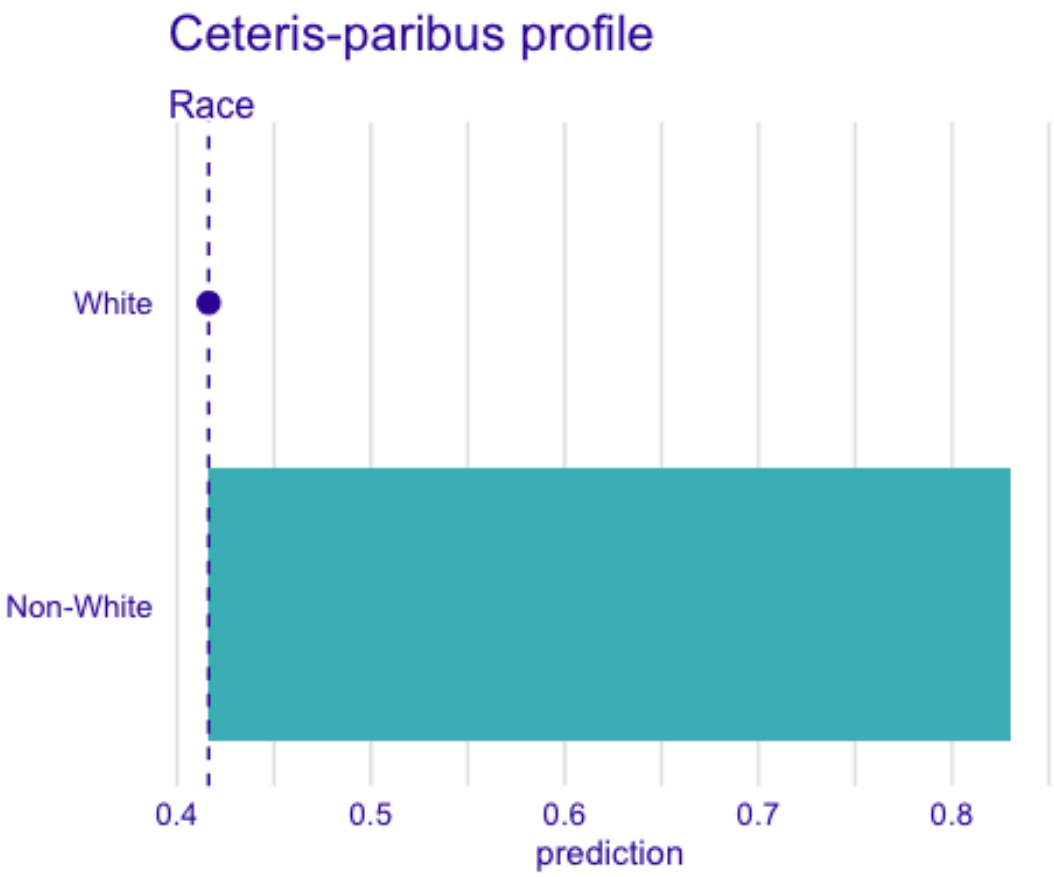
```
library("ggplot2")
# for numerical variables
#plot(cp_rf, variables = c("Race")) +
 # ggtitle("Ceteris-paribus profile", "")

### for categorical variables
plot(cp_rf,
    variables = c("Race"),
    variable_type="categorical",
    categorical_type = "bars") +
 ggtitle("Ceteris-paribus profile", "")
```




```
cp_rf <- predict_profile(explainer = rf_explainer,
                         new_observation = new_observation)

cp_rf

## Top profiles    :
##      Urbanisation      Race     Sex Vote     _yhat_      _vname_ _ids_
## 1           Rural     White    Male    1 0.4162758 Urbanisation     1
## 1.1         Urban     White    Male    1 0.8009311 Urbanisation     1
## 11          Rural Non-White    Male    1 0.8308034         Race     1
## 1.11        Rural     White    Male    1 0.4162758         Race     1
## 12          Rural     White Female    1 0.4222242          Sex     1
## 1.12        Rural     White    Male    1 0.4162758          Sex     1
##           _label_
## 1    Random Forest
## 1.1  Random Forest
```

```
## 11   Random Forest
## 1.11 Random Forest
## 12   Random Forest
## 1.12 Random Forest
##
##
## Top observations:
##   Urbanisation  Race  Sex Vote    _yhat_       _label_ _ids_
## 1        Rural White Male    1 0.4162758 Random Forest     1
```

```
library("ggplot2")
# for numerical variables
#plot(cp_rf, variables = c("MonthlyCharges")) +
 # ggtitle("Ceteris-paribus profile", "")

# for categorical variables
plot(cp_rf,
     variables = c("Race"),
     variable_type="categorical",
     categorical_type = "bars") +
  ggtitle("Ceteris-paribus profile", "")
```

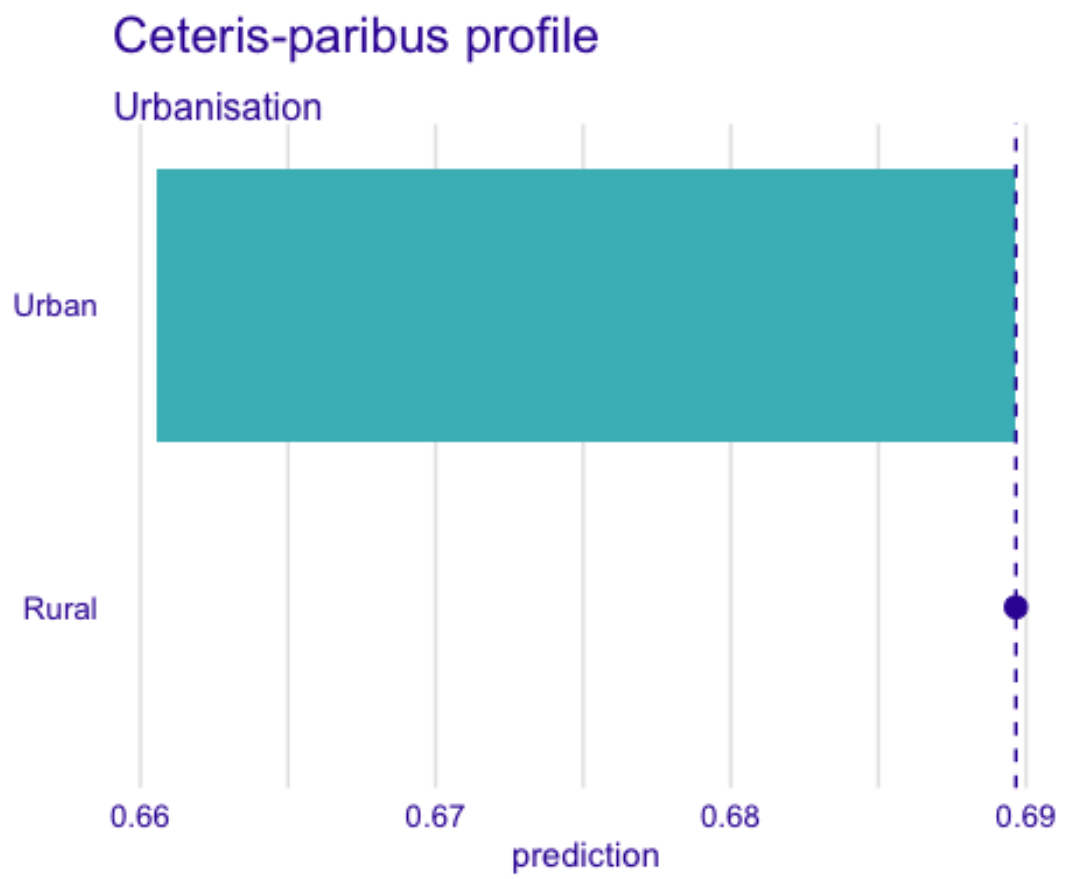


This again describes that changing from Race = White to Non-White increases the prediction, slightly more for the logistic regression than the random forest.

```
cp_rf <- predict_profile(explainer = logreg_explainer,
                         new_observation = new_observation)

cp_rf

## Top profiles    :
##      Urbanisation      Race    Sex Vote    _yhat_      _vname_ _ids_
## 1           Rural     White   Male    1 0.6896600 Urbanisation     1
## 1.1         Urban     White   Male    1 0.6605368 Urbanisation     1
## 11          Rural Non-White   Male    1 0.9105085         Race     1
## 1.11        Rural     White   Male    1 0.6896600         Race     1
## 12          Rural     White Female    1 0.6888268          Sex     1
## 1.12        Rural     White   Male    1 0.6896600          Sex     1
##                _label_
```

```
## 1    Logistic Regression
## 1.1  Logistic Regression
## 11   Logistic Regression
## 1.11 Logistic Regression
## 12   Logistic Regression
## 1.12 Logistic Regression
##
##
## Top observations:
##   Urbanisation  Race  Sex Vote  _yhat_             _label_ _ids_
## 1        Rural White Male    1 0.68966 Logistic Regression     1
```

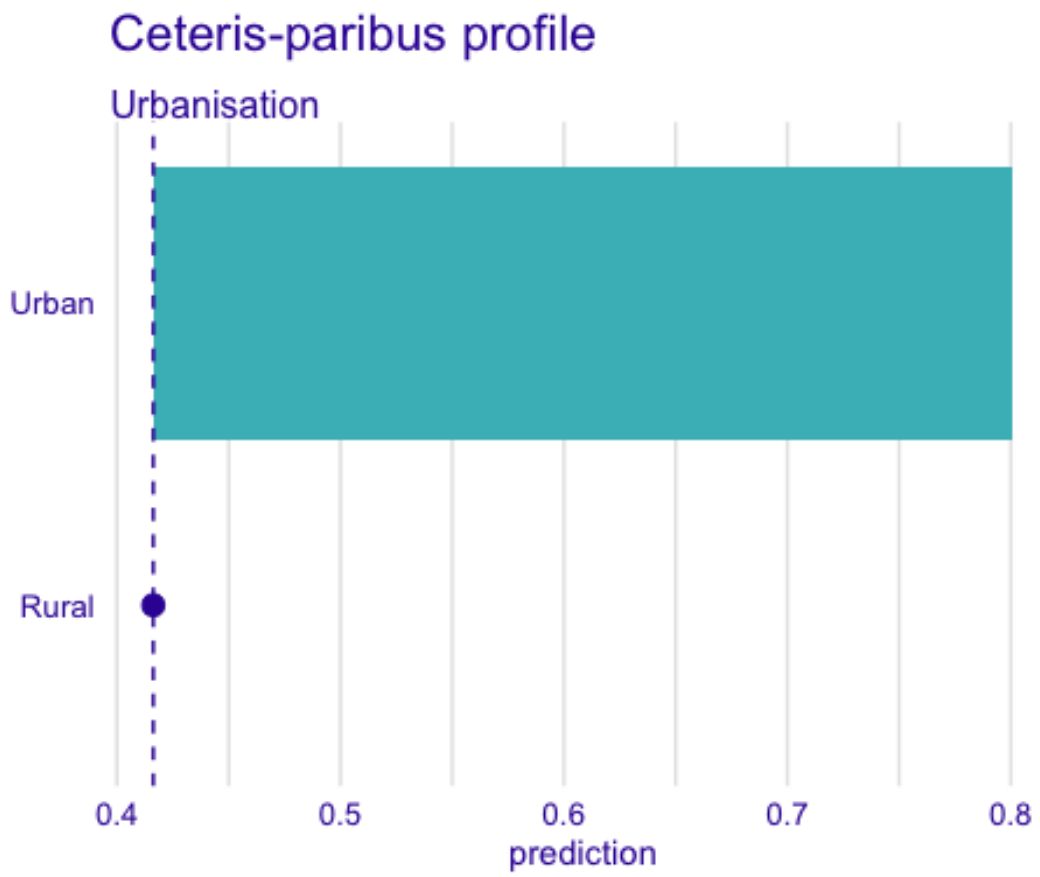
```
library("ggplot2")
# for numerical variables
#plot(cp_rf, variables = c("Race")) +
 # ggtitle("Ceteris-paribus profile", "")

### for categorical variables
plot(cp_rf,
    variables = c("Urbanisation"),
    variable_type="categorical",
    categorical_type = "bars") +
 ggtitle("Ceteris-paribus profile", "")
```




```
cp_rf <- predict_profile(explainer = rf_explainer,
                         new_observation = new_observation)

cp_rf

## Top profiles    :
##      Urbanisation      Race    Sex Vote    _yhat_      _vname_ _ids_
## 1           Rural     White   Male    1 0.4162758 Urbanisation     1
## 1.1         Urban     White   Male    1 0.8009311 Urbanisation     1
## 11          Rural Non-White   Male    1 0.8308034         Race     1
## 1.11        Rural     White   Male    1 0.4162758         Race     1
## 12          Rural     White Female    1 0.4222242          Sex     1
## 1.12        Rural     White   Male    1 0.4162758          Sex     1
##           _label_
## 1    Random Forest
## 1.1  Random Forest
```

```
## 11   Random Forest
## 1.11 Random Forest
## 12   Random Forest
## 1.12 Random Forest
##
##
## Top observations:
##   Urbanisation  Race  Sex Vote    _yhat_       _label_ _ids_
## 1        Rural White Male    1 0.4162758 Random Forest     1
```

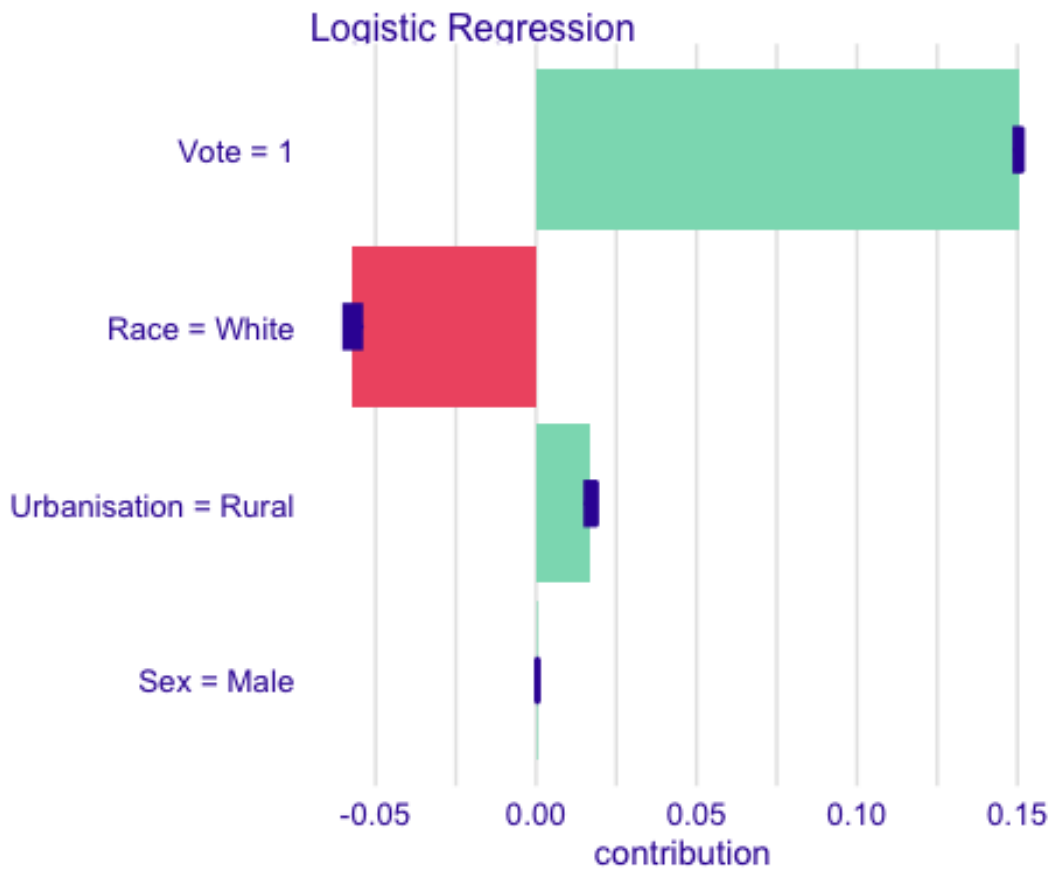
```
library("ggplot2")
# for numerical variables
#plot(cp_rf, variables = c("MonthlyCharges")) +
 # ggtitle("Ceteris-paribus profile", "")

# for categorical variables
plot(cp_rf,
    variables = c("Urbanisation"),
    variable_type="categorical",
    categorical_type = "bars") +
 ggtitle("Ceteris-paribus profile", "")
```



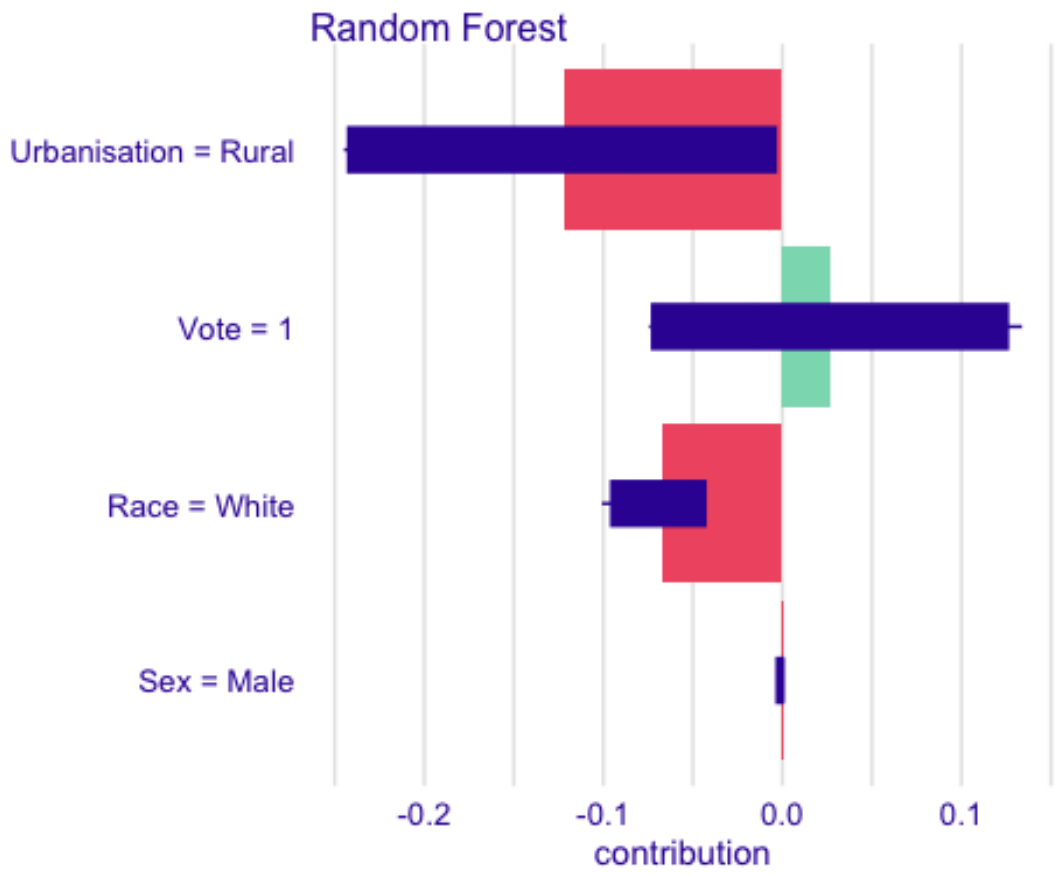
This again describes that Urbanization = Urban has a stronger impact on the prediction in the random forest, all else equal.

4.2.4 SHAP Value

```
# calculate shap values
pred_shap <- DALEX::predict_parts(logreg_explainer,
                                  new_observation = new_observation,
                                  type= "shap")
plot(pred_shap)
```




```
# note that red predictors had a decreasing impact on the specific predicted probability (here: less likely to churn), whereas green predictors had an increasing impact (here: more likely to churn).

#this takes very long
# calculate shap values
pred_shap <- DALEX::predict_parts(rf_explainer,
                                  new_observation = new_observation,
                                  type= "shap")
plot(pred_shap)
```




```
# note that red predictors had a decreasing impact on the specific predicted probability (here: less likely to churn), whereas green predictors had an increasing impact (here: more likely to churn).
```

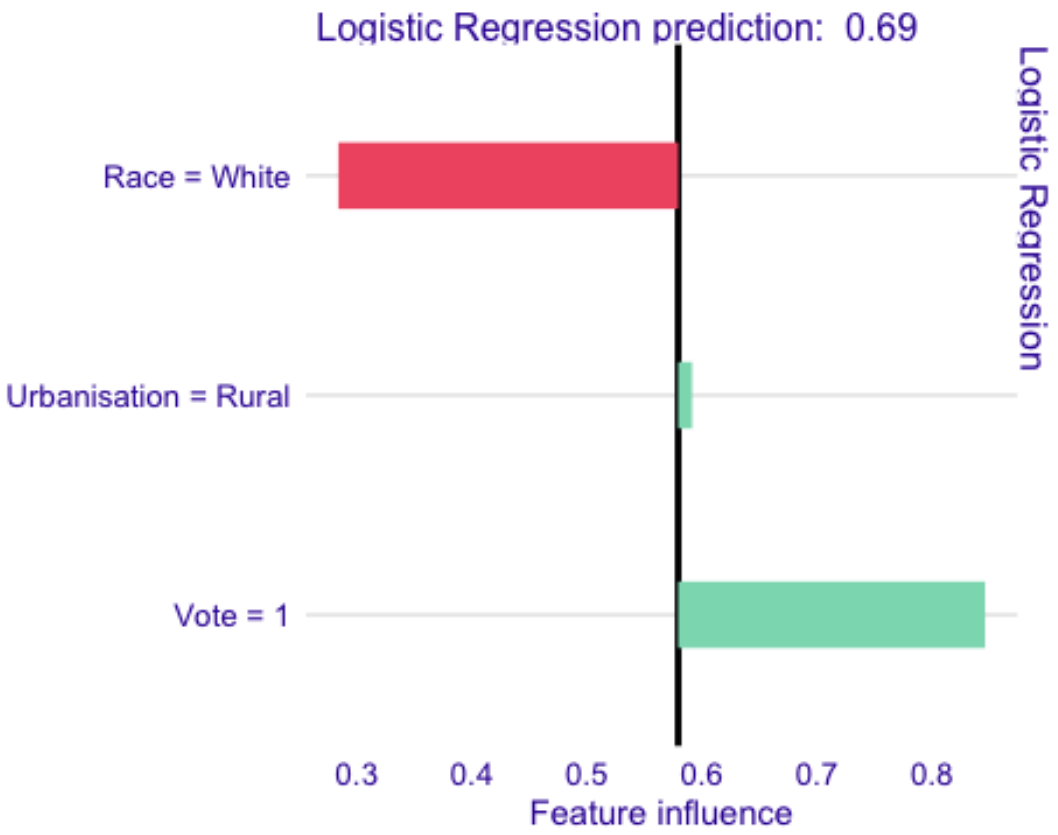
This is generally in line with our other local observations before.

4.2.5 LIME

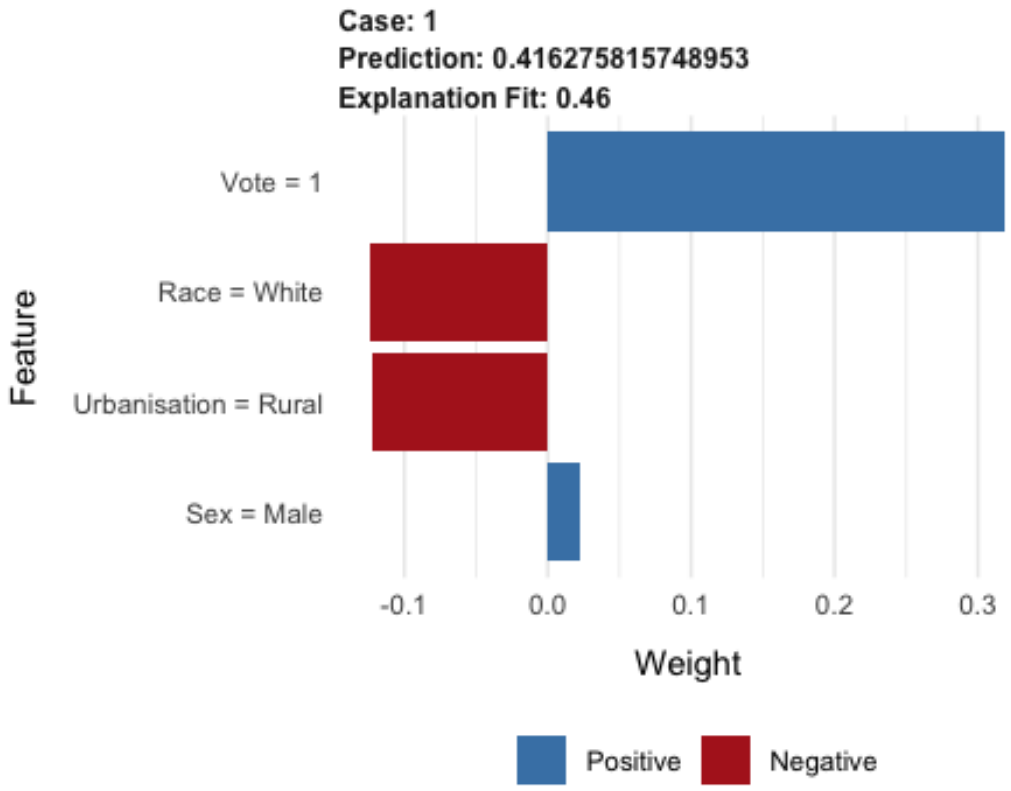
```
set.seed(1)
library("lime")

model_type.dalex_explainer <- DALEXtra::model_type.dalex_explainer
predict_model.dalex_explainer <- DALEXtra::predict_model.dalex_explainer

lime_rf <- predict_surrogate(explainer = logreg_explainer,
                new_observation = new_observation,
                n_features = 10,
                n_permutations = 1000,
```

```
            type = "lime")

plot(lime_rf)
```

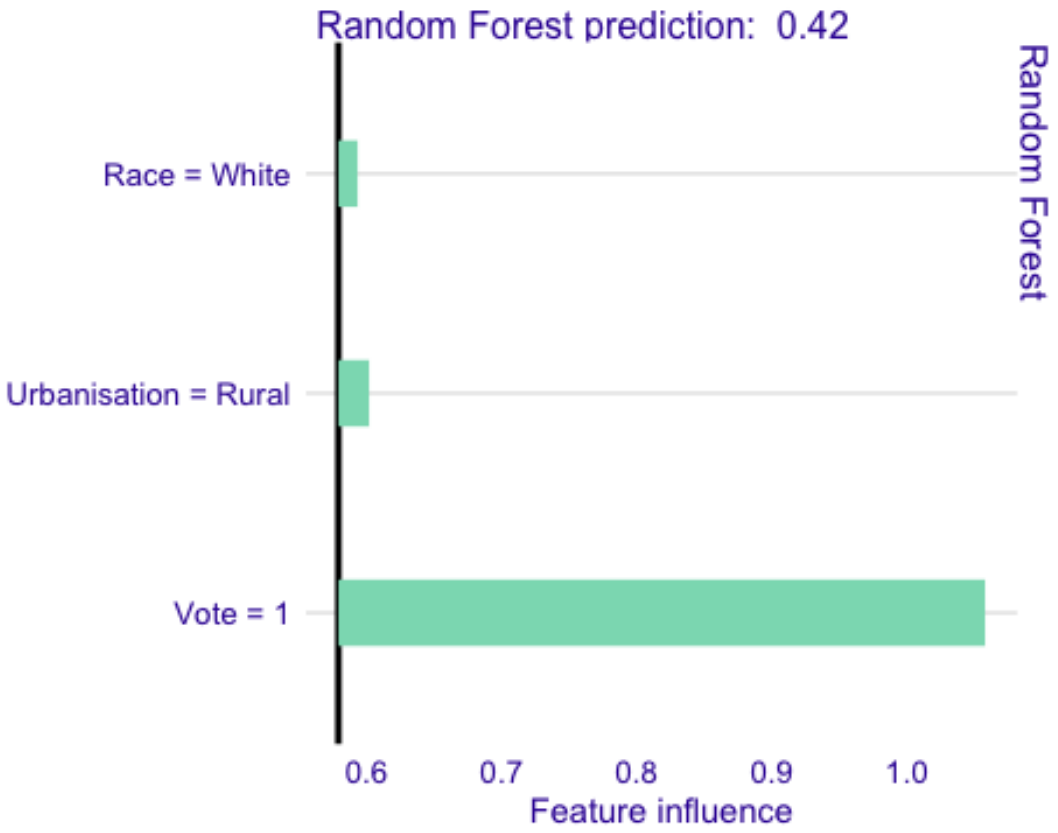

```
library("localModel")
localmodel_rf <- predict_surrogate(explainer = logreg_explainer,
            new_observation = new_observation,
            size = 1000,
            seed = 1,
            type = "localModel")
plot(localmodel_rf)
```




```
set.seed(1)
library("lime")

model_type.dalex_explainer <- DALEXtra::model_type.dalex_explainer
predict_model.dalex_explainer <- DALEXtra::predict_model.dalex_explainer

lime_rf <- predict_surrogate(explainer = rf_explainer,
                new_observation = new_observation,
                n_features = 10,
                n_permutations = 1000,
                type = "lime")

plot(lime_rf)
```




```
library("localModel")
localmodel_rf <- predict_surrogate(explainer = rf_explainer,
                new_observation = new_observation,
                size = 1000,
                seed = 1,
                type = "localModel")
plot(localmodel_rf)
```



This again confirms most of the previous observations in explainablity. Vote = 0 plays a large, negative role in both cases. Race = White is a large negative for the logistic regression and a small positive for the random forest.

5. Fairness

5.1 Investigating- Race as a protected class

```
library(fairmodels)
library(DALEX)
library(DALEXtra)

#dataset <- dataset %>% mutate_at(c("Vote"), as.factor)
#to rerun for previous code
# # the functions require a numeric representation of the target variable

# Hence, we convert the Yes/No values to 0/1, while keeping the classification setting
```

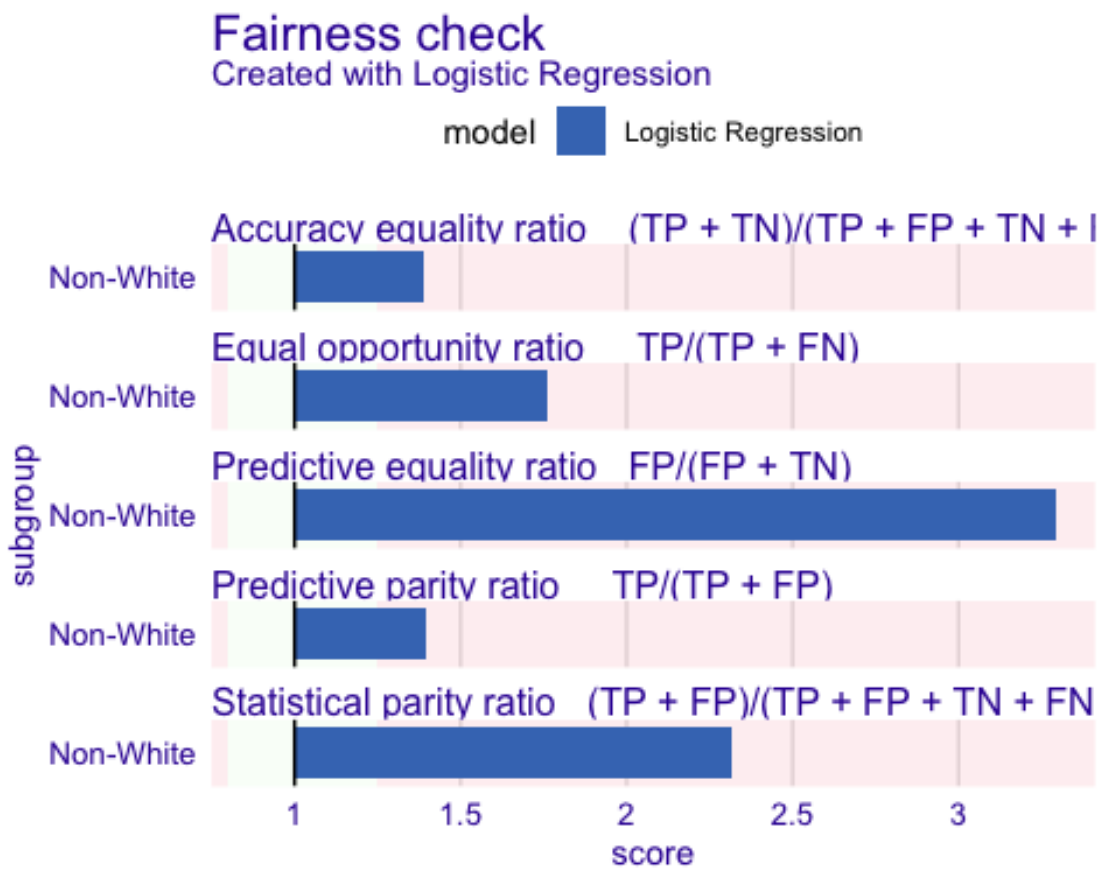
```
protected <- data_train$Race

# also, we only want to calculate the following aspects for the predictors we actually used in our model. otherwise, the calculations would consider all columns in the data set
logreg_explainer <- explain_tidymodels(
  logreg_fitted_workflow,
  data = data_train %>% dplyr::select(form_pred(model_form)),
  y = ifelse(data_train$vote_realized == 0, 0, 1),
  # predict_function_target_column = 2,
  type = "classification",
  label = "Logistic Regression"
)

## Preparation of a new explainer is initiated
##   -> model label       :  Logistic Regression
##   -> data              :  465676  rows  4  cols
##   -> target variable   :  465676  values
##   -> predict function  :  yhat.workflow  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package tidymodels , ver. 1.0.0 , task classification (  default  )
##   -> model_info        :  type set to  classification
##   -> predicted values  :  numerical, min =  0.3227422 , mean =  0.5791387 , max =
0.9105085
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.9105085 , mean =  1.0068e-10 , max =  0.6772578
##   A new explainer has been created!

# get an overview of the model performance
model_performance(logreg_explainer)

## Measures for:  classification
## recall     : 0.7288156
## precision  : 0.7412247
## f1         : 0.7349678
## accuracy   : 0.6955888
```

```
## auc       : 0.7549432
##
## Residuals:
##         0%        10%        20%        30%        40%        50%        60%
## -0.9105085 -0.6865463 -0.3524345 -0.3235931 -0.3227422  0.1009224  0.1012759
##        70%        80%        90%       100%
##  0.3394632  0.3403359  0.6475655  0.6772578
```

```
# side note: these are the same values that we get when we calculate the training performance manually
logreg_pred_train <-
  predict(logreg_fitted_workflow, data_train) %>%
  bind_cols(predict(logreg_fitted_workflow, data_train, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_train %>% select(all_of(target_var)))

# note: you need to select the truth and estimate variables based on the column names of the logreg_pred_train object
classification_metrics(data = logreg_pred_train,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1, # use the second outcome (Yes) as the level of interest
                       event_level = 'second')
```

```
## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.696
## 2 kap         binary         0.377
## 3 sens        binary         0.729
## 4 spec        binary         0.650
## 5 f_meas      binary         0.735
## 6 precision   binary         0.741
## 7 recall      binary         0.729
```

```
## 8 roc_auc     binary         0.755
## 9 mn_log_loss binary         0.565

# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
fobject <- fairness_check(logreg_explainer,
                          protected = protected,
                          privileged = "White")

## Creating fairness classification object
## -> Privileged subgroup       : character ([32m Ok [39m )
## -> Protected variable        : factor ([32m Ok [39m )
## -> Cutoff values for explainers  : 0.5 ( for all subgroups )
## -> Fairness objects      : 0 objects
## -> Checking explainers       : 1 in total ( [32m compatible [39m )
## -> Metric calculation        : 8/13 metrics calculated for all models ( [33m5 NA created[39m )
## [32m Fairness object created succesfully [39m

# get some metrics
metric_scores(fobject)

##
## Metric scores calculated for:  Logistic Regression
## First rows from data:
##        score  subgroup metric              model
## 1  1.0000000 Non-White    TPR Logistic Regression
## 2  0.5687405     White    TPR Logistic Regression
## 5  0.8839596 Non-White    PPV Logistic Regression
## 6  0.6348342     White    PPV Logistic Regression
## 11 1.0000000 Non-White    FPR Logistic Regression
## 12 0.3034281     White    FPR Logistic Regression

# print some of the fairness metrics
plot(fobject)
```

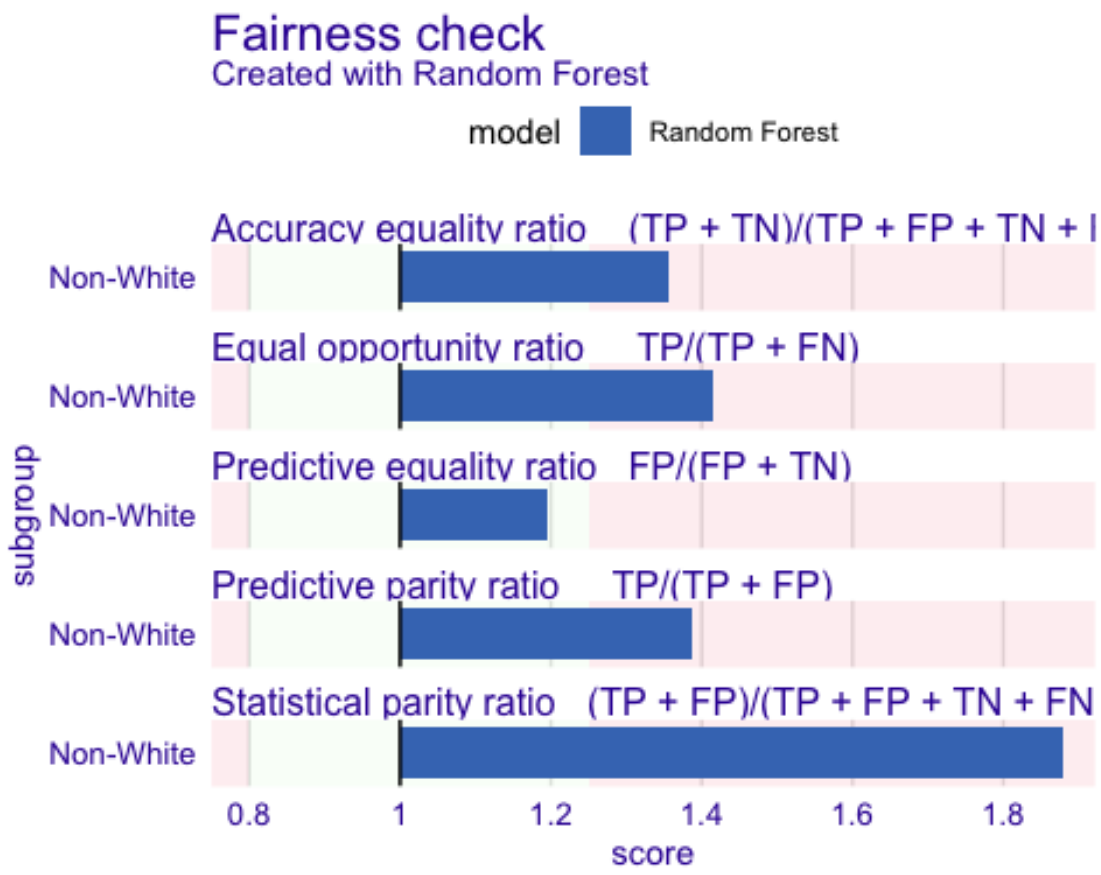



```
#can use  ?fairness_check for explenations

library(DALEX)
library(DALEXtra)
library(fairmodels)



rf_explainer <- explain_tidymodels(
  rf_fitted_workflow,
  data = data_train %>% dplyr::select(form_pred(model_form)),
  y = ifelse(data_train$vote_realized== 0, 0, 1),
  # predict_function_target_column = 2,
  type = "classification",
  label = "Random Forest"
)
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Random Forest
##   -> data              :  465676  rows  4  cols
##   -> target variable   :  465676  values
##   -> predict function  :  yhat.workflow  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package tidymodels , ver. 1.0.0 , task classification (  default  )
##   -> model_info        :  type set to  classification
##   -> predicted values  :  numerical, min =  0.02125303 , mean =  0.5791198 , max =
0.9779035
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.9779035 , mean =  1.892998e-05 , max =
0.978747
##   A new explainer has been created!

# get an overview of the model performance
model_performance(rf_explainer)

## Measures for:  classification
## recall     : 0.811818
## precision  : 0.7898696
## f1         : 0.8006934
## accuracy   : 0.7659403
## auc        : 0.8394109
##
## Residuals:
##         0%         10%         20%         30%         40%         50%
## -0.97790350 -0.57949415 -0.41627582 -0.19986847 -0.19957837  0.02209650
##         60%         70%         80%         90%        100%
##  0.02247174  0.19906891  0.41825753  0.57777583  0.97874697

# side note: these are the same values that we get when we calculate the training performance
manually
#####This is optional and yileds the same result
```

```
randomforest_pred_train <-
  predict(rf_fitted_workflow, data_train) %>%
  bind_cols(predict(rf_fitted_workflow, data_train, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(data_train %>% select(all_of(target_var)))

# note: you need to select the truth and estimate variables based on the column names of the
logreg_pred_train object
classification_metrics(data = randomforest_pred_train,
                       truth = vote_realized,
                       estimate = .pred_class,
                       .pred_1, # use the second outcome (Good) as the level of interest
                       event_level = 'second')

## # A tibble: 9 × 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.766
## 2 kap         binary         0.517
## 3 sens        binary         0.812
## 4 spec        binary         0.703
## 5 f_meas      binary         0.801
## 6 precision   binary         0.790
## 7 recall      binary         0.812
## 8 roc_auc     binary         0.839
## 9 mn_log_loss binary         0.476

# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
fobject_randomforest <- fairness_check(rf_explainer,
                       protected = protected,
                       privileged = "White")

## Creating fairness classification object
## -> Privileged subgroup       : character ([32m Ok [39m )
```

```
## -> Protected variable        : factor ([32m Ok [39m )
## -> Cutoff values for explainers  : 0.5 ( for all subgroups )
## -> Fairness objects      : 0 objects
## -> Checking explainers       : 1 in total ( [32m compatible [39m )
## -> Metric calculation        : 13/13 metrics calculated for all models
## [32m Fairness object created succesfully [39m

# get some metrics
metric_scores(fobject_randomforest)

##
## Metric scores calculated for:  Random Forest
## First rows from data:
##        score  subgroup metric         model
## 1  0.9958643 Non-White    TPR Random Forest
## 2  0.7031789     White    TPR Random Forest
## 5  0.9557822 Non-White    PPV Random Forest
## 6  0.6897729     White    PPV Random Forest
## 11 0.3509626 Non-White    FPR Random Forest
## 12 0.2933266     White    FPR Random Forest

# print some of the fairness metrics
plot(fobject_randomforest)
```



For the logistic regression, we see that all metrics are indicating strong discrimination again the non-white subgroup. All reach beyond the 0.8 cut-off point as known in the US. The largest ratio has a score of over three (Predictive Equality Ratio), while there are two over 1.5 (Equal Opportunity Ratio and Statistical Parity Ratio) and another two close to 1.5 (Accuracy equality ratio and Statistical Parity Ratio).

The unfairness is less strong in the Random Forest, yet it is still extensive in its bias against the non-white group. Only one value is below the cut-off point (Predictive equality ratio). Three are close to 1.4 (Accuracy equality ratio, Equal opportunity ratio, and predictive parity ratio), Statistical parity ratio is close to 2.

5.2 Bias Mitigation

5.2.1 Pre-Processing

5.2.1.1 Disprate Impact Remover

Necessary for the disprate impact remover: transforming the variable to numeric. It needs numeric columns in the data.

Logistic Regression:

```
library(fairmodels)

# not rendered for the knitting to work
# first, let's start with the disparate impact remover


dataset <- dataset %>% mutate_at(c("Vote"), as.numeric)

data_split <- initial_split(data = dataset,
                            prop = 0.8,
                            strata = vote_realized)

data_train <- training(data_split) # this will be our training data
data_test <- testing(data_split) # this will be our test data

data_train_dir <- disparate_impact_remover(data = as.data.frame(data_train),
                                protected = protected,
                                features_to_transform = "Vote")

data_train_dir <- disparate_impact_remover(data = as.data.frame(data_train),
                                protected = protected,
                                features_to_transform = "Vote")


# step B: specify any additional data processing
# note that we specify the recipe to only use the training data
model_recipe_dir <- recipe(formula = model_form, data = data_train_dir) %>%
 step_novel(all_nominal_predictors()) %>% # this adds a novel factor level if the test data comes with new levels of a factor
 step_unknown(all_nominal_predictors()) %>% # this assigns a missing value in a factor variable to 'unknown'
 step_dummy(all_nominal_predictors()) %>%  # this creates dummy variables for all predictors, but not the target variable
 step_zv(all_predictors()) # this removes potential zero variance predictors

### Workflow Step 3: Specify the ML Model Type and Workflow ###
```

```r
# then, let's put everything into a workflow
logreg_workflow_dir <- workflow() %>%
   # add the recipe (data pre-processing)
   add_recipe(model_recipe_dir) %>%
   # add the ML model
   add_model(logreg_model)


# train the model
# we only need to change the workflow fit() function to fit_resamples()
# we can additionally specify control parameters for the resampling procedure
# e.g., here we specify that the second outcome level ("yes") is the level of interest
set.seed(1)
control <- control_resamples(save_pred = TRUE,
                             event_level = "second")

folds <- vfold_cv(data_train_dir, v = 3)

logreg_fit_cv_dir <-
  logreg_workflow_dir %>%
  fit_resamples(folds,
                control = control,
                metrics  = classification_metrics)

metric_to_use <- "roc_auc"

logreg_best_params_dir <- logreg_fit_cv_dir %>% select_best(metric_to_use)

# finally, re-fit the logistic regression on the entire training data
logreg_fitted_workflow_dir <- logreg_workflow_dir %>%
  finalize_workflow(logreg_best_params_dir) %>%
  fit(data_train_dir)


# then, let's look at the fairness performance of the resulting model again
```

```
logreg_explainer_dir <- explain_tidymodels(
  logreg_fitted_workflow_dir,
  data = dplyr::select(data_train_dir, -vote_realized),
  y = ifelse(data_train_dir$vote_realized == 0, 0, 1),
  # predict_function_target_column = 2,
  type = "classification",
  label = "Logistic Regression DIR"
)

# get an overview of the model performance
model_performance(logreg_explainer_dir)


# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
fobject_dir <- fairness_check(logreg_explainer, logreg_explainer_dir,
                      protected = protected,
                      privileged = "White")

# get some metrics
metric_scores(fobject_dir)
# print some of the fairness metrics
plot(fobject_dir)
```

We see that the DIR mostly only worsens the fairness metrics, which end up being even more disadvantageous for the non-white group. The only exception is the Predictive Parity Ratio, which is reduced by a considerable amount, to just below 1.5.

Random Forest:

```
# not rendered for the knitting to work
# first, let's start with the disparate impact remover

data_train_dir <- disparate_impact_remover(data = as.data.frame(data_train),
                                   protected = protected,
                                   features_to_transform = "Vote")
```

```
# step B: specify any additional data processing
# note that we specify the recipe to only use the training data
model_recipe_dir <- recipe(formula = model_form, data = data_train_dir) %>%
 step_novel(all_nominal_predictors()) %>% # this adds a novel factor level if the test data comes with new levels of a factor
 step_unknown(all_nominal_predictors()) %>% # this assigns a missing value in a factor variable to 'unknown'
 step_dummy(all_nominal_predictors()) %>%  # this creates dummy variables for all predictors, but not the target variable
 step_zv(all_predictors()) # this removes potential zero variance predictors

### Workflow Step 3: Specify the ML Model Type and Workflow ###

# then, let's put everything into a workflow
randomforest_workflow_dir <- workflow() %>%
   # add the recipe (data pre-processing)
   add_recipe(model_recipe_dir) %>%
   # add the ML model
   add_model(forest_model)


# train the model
# we only need to change the workflow fit() function to fit_resamples()
# we can additionally specify control parameters for the resampling procedure
# e.g., here we specify that the second outcome level ("yes") is the level of interest
set.seed(99)
control <- control_resamples(save_pred = TRUE,
                             event_level = "second")

folds <- vfold_cv(data_train_dir, v = 3)

randomforest_fit_cv_dir <-
 logreg_workflow_dir %>%
 fit_resamples(folds,
               control = control,
```

```
            metrics  = classification_metrics)

metric_to_use <- "roc_auc"

randomforest_best_params_dir <- randomforest_fit_cv_dir %>% select_best(metric_to_use)

# finally, re-fit the logistic regression on the entire training data
randomforest_fitted_workflow_dir <- randomforest_workflow_dir %>%
 finalize_workflow(randomforest_best_params_dir) %>%
 fit(data_train_dir)


# then, let's look at the fairness performance of the resulting model again
randomforest_explainer_dir <- explain_tidymodels(
 randomforest_fitted_workflow_dir,
 data = dplyr::select(data_train_dir, -vote_realized),
 y = ifelse(data_train_dir$vote_realized == 0, 0, 1),
 # predict_function_target_column = 2,
 type = "classification",
 label = "Random Forest DIR"
)

# get an overview of the model performance
model_performance(randomforest_explainer_dir)


# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
fobject_randomforest_dir <- fairness_check(rf_explainer, randomforest_explainer_dir,
                      protected = protected,
                      privileged = "White")

# get some metrics
metric_scores(fobject_randomforest_dir)
# print some of the fairness metrics
plot(fobject_randomforest_dir)
```

The DIR is not very useful here either, increasing unfairness in 4 out of the 5 metrics. Interestingly, for the predictive equality ratio, the bias switches against the white group.

5.2.1.2 Reweighing

Logistic Regression

```
# first, we calculate the weights on the training data

reweight_fixed <- function (protected, y)

{

 if (!is.factor(protected)) {

   cat("\nchanging protected to factor \n")

   protected <- as.factor(protected)

 }

 stopifnot(is.factor(protected))

 stopifnot(is.numeric(y))

 stopifnot(length(y) == length(protected))

 if (!(all(unique(y) == c(1, 0)) | all(unique(y) == c(0,

   1))))

   stop("y must be numeric vector with values 0 and 1")

 protected_levels <- levels(protected)

n <- length(y)

weight_vector <- rep(NA, n)

for (subgroup in protected_levels) {
```

```
 for (c in unique(y)) {
   Xs <- as.numeric(sum(protected == subgroup))
   Xc <- as.numeric(sum(y == c))
   Xsc <- as.numeric(sum(protected == subgroup & y == c))
   Wsc <- (Xs * Xc)/(n * Xsc)
   weight_vector[protected == subgroup & y == c] <- Wsc
 }
}

 return(weight_vector)
}

weights <- reweight_fixed(protected =protected,
                  y = ifelse(data_train$vote_realized == 0, 0, 1))

# then, we add them to the data set
data_train$case_wts <- importance_weights(weights)

# we also add them to the list of predictors
model_form_with_weights <- model_form <- vote_realized ~ Race + Sex  + Vote + Urbanisation
+ case_wts

model_recipe <- recipe(formula = model_form_with_weights, data = data_train) %>%
```

```
 step_novel(all_nominal_predictors()) %>% # this adds a novel factor level if the test data comes with new levels of a factor
 step_unknown(all_nominal_predictors()) %>% # this assigns a missing value in a factor variable to 'unknown'
 step_dummy(all_nominal_predictors()) %>%  # this creates dummy variables for all predictors, but not the target variable
 step_zv(all_predictors()) # this removes potential zero variance predictors

# then, let's put everything into a workflow
logreg_workflow_reweighing <- workflow() %>%
   add_case_weights(case_wts) %>%
   # add the recipe (data pre-processing)
   add_recipe(model_recipe) %>%
   # add the ML model
   add_model(logreg_model)

# train the model
set.seed(1)

folds <- vfold_cv(data_train, v = 3)

logreg_fit_cv_reweighing <-
 logreg_workflow_reweighing %>%
 fit_resamples(folds,
               control = control,
               metrics  = classification_metrics)

## ! Fold1: preprocessor 1/1, model 1/1: non-integer #successes in a binomial glm!

## ! Fold2: preprocessor 1/1, model 1/1: non-integer #successes in a binomial glm!

## ! Fold3: preprocessor 1/1, model 1/1: non-integer #successes in a binomial glm!

# and then, select the best performing model according to a specified metric again
metric_to_use <- "roc_auc"

logreg_best_params_weights <- logreg_fit_cv_reweighing %>% select_best(metric_to_use)
```

```r
# finally, re-fit the logistic regression on the entire training data
logreg_fitted_workflow_reweighing <- logreg_workflow_reweighing %>%
  finalize_workflow(logreg_best_params_weights) %>%
  fit(data_train)
```

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

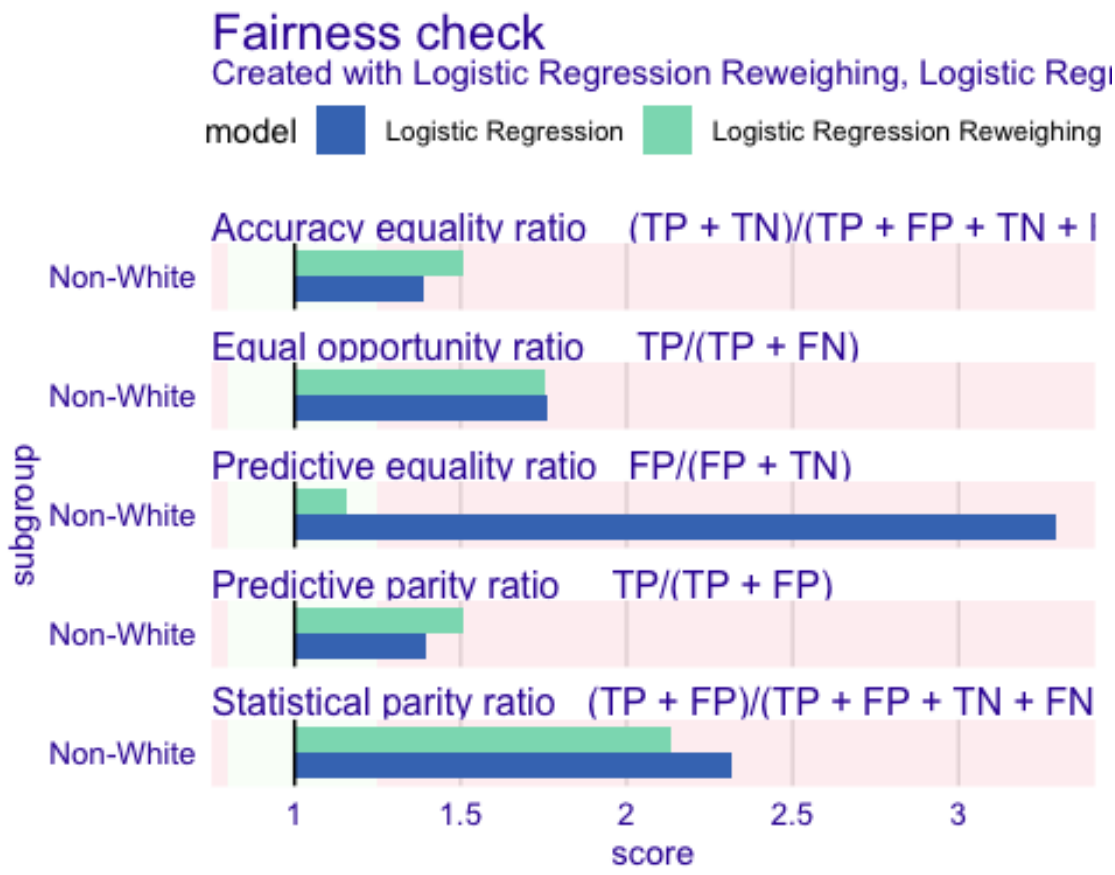
```r
logreg_explainer_reweighing <- explain_tidymodels(
  logreg_fitted_workflow_reweighing,
  data = dplyr::select(data_train, -vote_realized),
  y = ifelse(data_train$vote_realized == 0, 0, 1),
  # predict_function_target_column = 2,
  type = "classification",
  label = "Logistic Regression Reweighing"
)
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Logistic Regression Reweighing
##   -> data              :  465676  rows  8  cols
##   -> target variable   :  465676  values
##   -> predict function  :  yhat.workflow  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package tidymodels , ver. 1.0.0 , task classification (  default  )
##   -> model_info        :  type set to  classification
##   -> predicted values  :  numerical, min =  0.2982316 , mean =  0.5898687 , max =
0.7861256
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.7861256 , mean =  -0.01072997 , max =
0.7017684
##   A new explainer has been created!
```

```r
# get an overview of the model performance
model_performance(logreg_explainer_reweighing)
```

```
## Measures for:  classification
## recall     : 0.7272805
## precision  : 0.7654791
## f1         : 0.745891
## accuracy   : 0.7130151
## auc        : 0.6809006
##
## Residuals:
##         0%        10%        20%        30%        40%        50%        60%
## -0.7861256 -0.7726571 -0.4213067 -0.4043548 -0.4023307  0.2273429  0.2288237
##        70%        80%        90%       100%
##  0.3179072  0.3197338  0.5786933  0.7017684

# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
# exclcuded for knitting
#fobject_reweighing <- fairness_check(logreg_explainer_reweighing, logreg_explainer_dir,
logreg_explainer,
 #                    protected = protected,
  #                   privileged = "White")

fobject_reweighing <- fairness_check(logreg_explainer_reweighing, logreg_explainer,
                     protected = protected,
                     privileged = "White")

## Creating fairness classification object
## -> Privileged subgroup       : character ([32m Ok [39m )
## -> Protected variable        : factor ([32m Ok [39m )
## -> Cutoff values for explainers  : 0.5 ( for all subgroups )
## -> Fairness objects      : 0 objects
## -> Checking explainers       : 2 in total ( [32m compatible [39m )
## -> Metric calculation        : 8/13 metrics calculated for all models ( [33m5 NA created[39m )
## [32m Fairness object created succesfully [39m
```

```
# get some metrics
metric_scores(fobject_reweighing)

##
## Metric scores calculated for:  Logistic Regression Reweighing, Logistic Regression
## First rows from data:
##        score  subgroup metric                        model
## 1  0.9958643 Non-White    TPR Logistic Regression Reweighing
## 2  0.5687405     White    TPR Logistic Regression Reweighing
## 5  0.9557822 Non-White    PPV Logistic Regression Reweighing
## 6  0.6348342     White    PPV Logistic Regression Reweighing
## 11 0.3509626 Non-White    FPR Logistic Regression Reweighing
## 12 0.3034281     White    FPR Logistic Regression Reweighing

# print some of the fairness metrics
plot(fobject_reweighing)
```



```
reweight_fixed <- function (protected, y)
{
  if (!is.factor(protected)) {
    cat("\nchanging protected to factor \n")
    protected <- as.factor(protected)
  }
  stopifnot(is.factor(protected))
  stopifnot(is.numeric(y))
  stopifnot(length(y) == length(protected))
  if (!(all(unique(y) == c(1, 0)) | all(unique(y) == c(0,
    1))))
    stop("y must be numeric vector with values 0 and 1")
  protected_levels <- levels(protected)
n <- length(y)
weight_vector <- rep(NA, n)
for (subgroup in protected_levels) {
  for (c in unique(y)) {
    Xs <- as.numeric(sum(protected == subgroup))
    Xc <- as.numeric(sum(y == c))
```

```
    Xsc <- as.numeric(sum(protected == subgroup & y == c))

    Wsc <- (Xs * Xc)/(n * Xsc)

    weight_vector[protected == subgroup & y == c] <- Wsc

  }

}



  return(weight_vector)

}



weights <- reweight_fixed(protected =protected,

                  y = ifelse(data_train$vote_realized == 0, 0, 1))


# then, we add them to the data set
data_train$case_wts <- importance_weights(weights)

# we also add them to the list of predictors
model_form_with_weights <- model_form <- vote_realized ~ Race + Sex  + Vote + Urbanisation
+ case_wts

model_recipe <- recipe(formula = model_form_with_weights, data = data_train) %>%
  step_novel(all_nominal_predictors()) %>% # this adds a novel factor level if the test data
comes with new levels of a factor
  step_unknown(all_nominal_predictors()) %>% # this assigns a missing value in a factor
variable to 'unknown'
  step_dummy(all_nominal_predictors()) %>%  # this creates dummy variables for all predictors,
but not the target variable
  step_zv(all_predictors()) # this removes potential zero variance predictors
```

```r
# then, let's put everything into a workflow
rf_workflow_reweighing <- workflow() %>%
   add_case_weights(case_wts) %>%
   # add the recipe (data pre-processing)
   add_recipe(model_recipe) %>%
   # add the ML model
   add_model(forest_model)

# train the model
set.seed(99)

folds <- vfold_cv(data_train, v = 3)

randomforest_fit_cv_reweighing <-
 rf_workflow_reweighing %>%
 fit_resamples(folds,
               control = control,
               metrics  = classification_metrics)
```

```
## ! Fold1: preprocessor 1/1, model 1/1: 7 columns were requested but there were 4 predictors in the data. 4 will...

## ! Fold2: preprocessor 1/1, model 1/1: 7 columns were requested but there were 4 predictors in the data. 4 will...

## ! Fold3: preprocessor 1/1, model 1/1: 7 columns were requested but there were 4 predictors in the data. 4 will...
```

```r
# and then, select the best performing model according to a specified metric again
metric_to_use <- "roc_auc"

randomforest_best_params_weights <- randomforest_fit_cv_reweighing %>%
select_best(metric_to_use)

# finally, re-fit the logistic regression on the entire training data
randomforest_fitted_workflow_reweighing <- rf_workflow_reweighing %>%
```

```
 finalize_workflow(randomforest_best_params_weights) %>%
 fit(data_train)
```

```
## Warning: 7 columns were requested but there were 4 predictors in the data. 4
## will be used.
```

```
randomforest_explainer_reweighing <- explain_tidymodels(
 randomforest_fitted_workflow_reweighing,
 data = dplyr::select(data_train, -vote_realized),
 y = ifelse(data_train$vote_realized == 0, 0, 1),
 # predict_function_target_column = 2,
 type = "classification",
 label = "Randfom Forest Reweighing"
)
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Randfom Forest Reweighing
##   -> data              :  465676  rows  8  cols
##   -> target variable   :  465676  values
##   -> predict function  :  yhat.workflow  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package tidymodels , ver. 1.0.0 , task classification (  default  )
##   -> model_info        :  type set to  classification
##   -> predicted values  :  numerical, min =  0.003941527 , mean =  0.6079549 , max =
0.8889887
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.8889887 , mean =  -0.02881614 , max =
0.9960585
##   A new explainer has been created!
```

```
# get an overview of the model performance
model_performance(randomforest_explainer_reweighing)
```

```
## Measures for:  classification
## recall     : 0.8680193
## precision  : 0.7137278
```

```
## f1          : 0.7833484
## accuracy    : 0.7219333
## auc         : 0.8249931
##
## Residuals:
##         0%        10%        20%        30%        40%        50%        60%
## -0.8889887 -0.6715005 -0.5143031 -0.2704363 -0.2701049  0.1110113  0.1130024
##        70%        80%        90%       100%
##  0.1448903  0.3264079  0.4856969  0.9960585
```

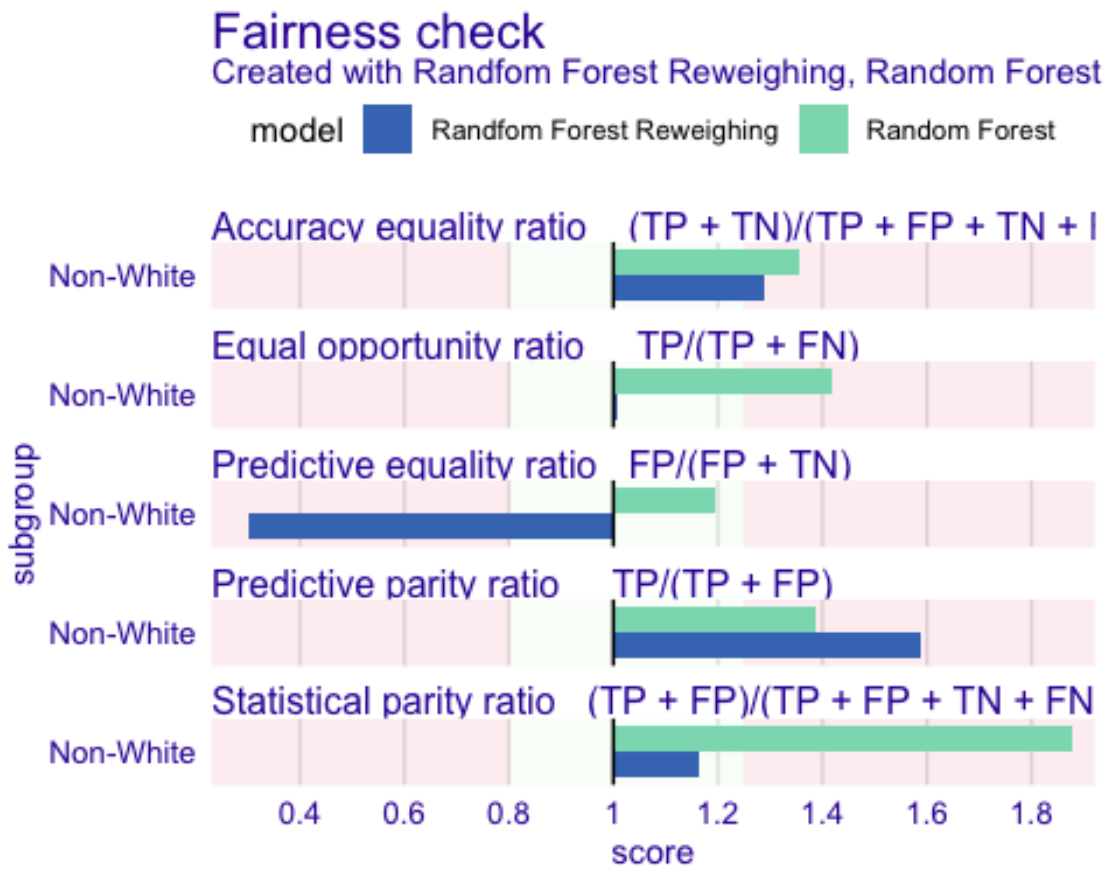
```
# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
# exclcuded for knitting
#fobject_randomforest_reweighing <- fairness_check(randomforest_explainer_reweighing,
randomforest_explainer_dir, rf_explainer,
#                       protected = protected,
 #                      privileged = "White")

fobject_randomforest_reweighing <- fairness_check(randomforest_explainer_reweighing,
rf_explainer,
                      protected = protected,
                      privileged = "White")
```

```
## Creating fairness classification object
## -> Privileged subgroup       : character ([32m Ok [39m )
## -> Protected variable        : factor ([32m Ok [39m )
## -> Cutoff values for explainers  : 0.5 ( for all subgroups )
## -> Fairness objects      : 0 objects
## -> Checking explainers       : 2 in total ( [32m compatible [39m )
## -> Metric calculation        : 13/13 metrics calculated for all models
## [32m Fairness object created succesfully [39m
```

```
# get some metrics
metric_scores(fobject_randomforest_reweighing)
```

```
##
## Metric scores calculated for:  Randfom Forest Reweighing, Random Forest
## First rows from data:
##        score  subgroup metric                  model
## 1  0.8711240 Non-White    TPR Randfom Forest Reweighing
## 2  0.8661867     White    TPR Randfom Forest Reweighing
## 5  0.9777217 Non-White    PPV Randfom Forest Reweighing
## 6  0.6151289     White    PPV Randfom Forest Reweighing
## 11 0.1512061 Non-White    FPR Randfom Forest Reweighing
## 12 0.5026580     White    FPR Randfom Forest Reweighing
```

```
# print some of the fairness metrics
plot(fobject_randomforest_reweighing)
```



Discussion:

Contrasting the Logistic Regression with DIR and Reweighing, we can see that the latter excels in improving fairness according to one specific ratio: the Predictive Equality Ratio. It

shrinks significantly, towards the range of less than 1.2, which is generally regarded as more acceptable. The performance is only slightly better than the Logistic Regression in the Equal Opportunity ratio and the Statistical Parity ratio. Yet, with those two metrics, it still remains high. When it comes to the other two ratios, it is performing better than the DIR, but slightly worse than the normal Logistic Regression. It is therefore not strictly better than the Logistic Regression but offers upsides when it comes to a set of ratios. It is strictly better than the DIR. The metrics show an accuracy of 71% and an auc of 68%, especially since the auc has suffered compared to the normal logistic regression, highlighting this trade-offs.

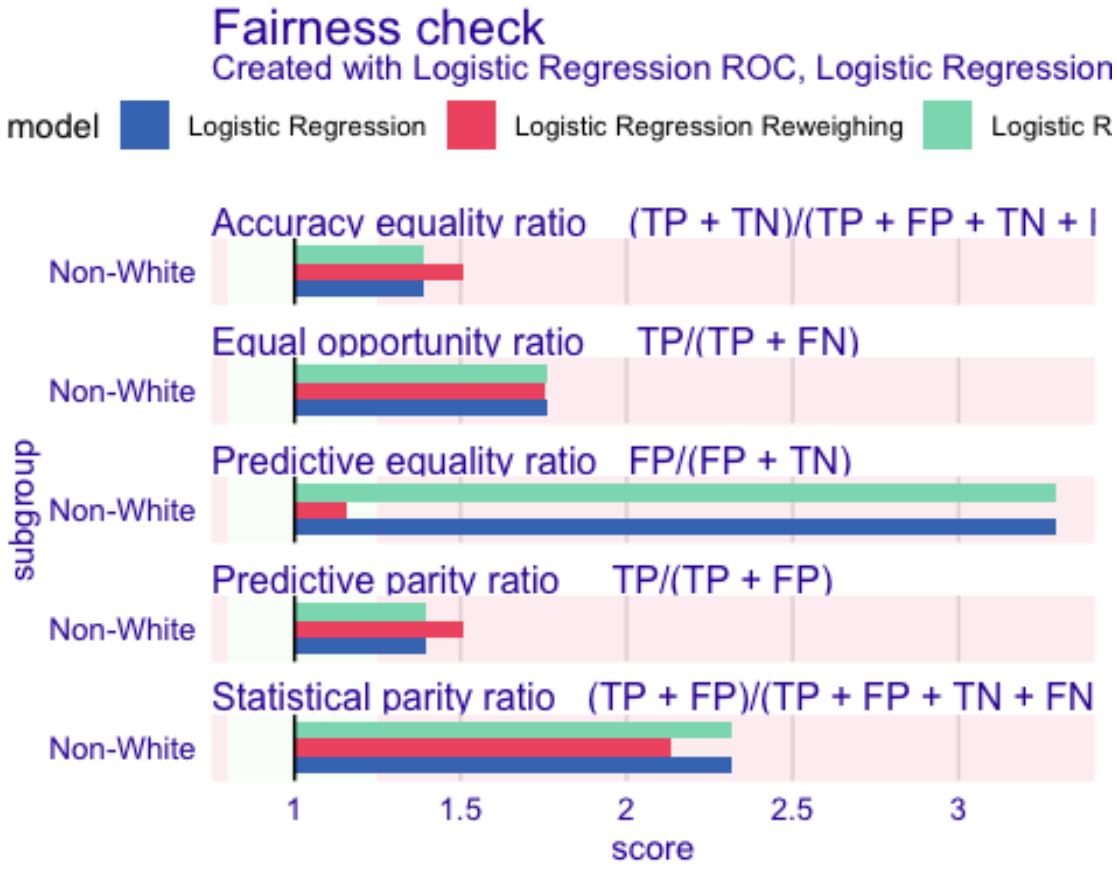
The random forest reweighing, overall, leads to good results with two exemptions. For the three metrics, it shows the best performance of out the models compared. The first is the Accuracy equality ratio, which is close it 1.3. It does tremendously well in the equal opportunity ratio, which is almost entirely removed. The statistical parity ratio also strongly improves, dropping below 1.2. For the predictive parity ratio, the reweighing is better than the DIR, but still worse than the normal random forest. Surprisingly, the predictive equality ratio is strongly negatively impacted. It declines to less than 0.4, switching the discrimination from white to non-white. Overall, the metrics show an auc of 83% and an accuracy of 72%. The accuracy loses about 5 percentage points, for which we receive mostly much better fairness metrics.

5.2.2 Post-Processing

5.2.2.1 ROC

```
# using the existing explainer object, we can directly run the ROC method on the already
created predictions
logreg_explainer_roc <- roc_pivot(logreg_explainer,
                          protected = protected,
                          privileged = "White",
                          cutoff = 0.5,
                          theta = 0.1)

# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
fobject_roc <- fairness_check(fobject_reweighing,
                        logreg_explainer_roc,
                        label="Logistic Regression ROC",
                        protected = protected,
                        privileged = "White")
```

```
## Creating fairness classification object
## -> Privileged subgroup       : character ([32m Ok [39m )
## -> Protected variable        : factor ([32m Ok [39m )
## -> Cutoff values for explainers  : 0.5 ( for all subgroups )
## -> Fairness objects      : 1 object ( [32m compatible [39m )
## -> Checking explainers       : 3 in total ( [32m compatible [39m )
## -> Metric calculation        : 8/13 metrics calculated for all models ( [33m5 NA created[39m )
## [32m Fairness object created succesfully [39m

# print some of the fairness metrics
plot(fobject_roc)
```



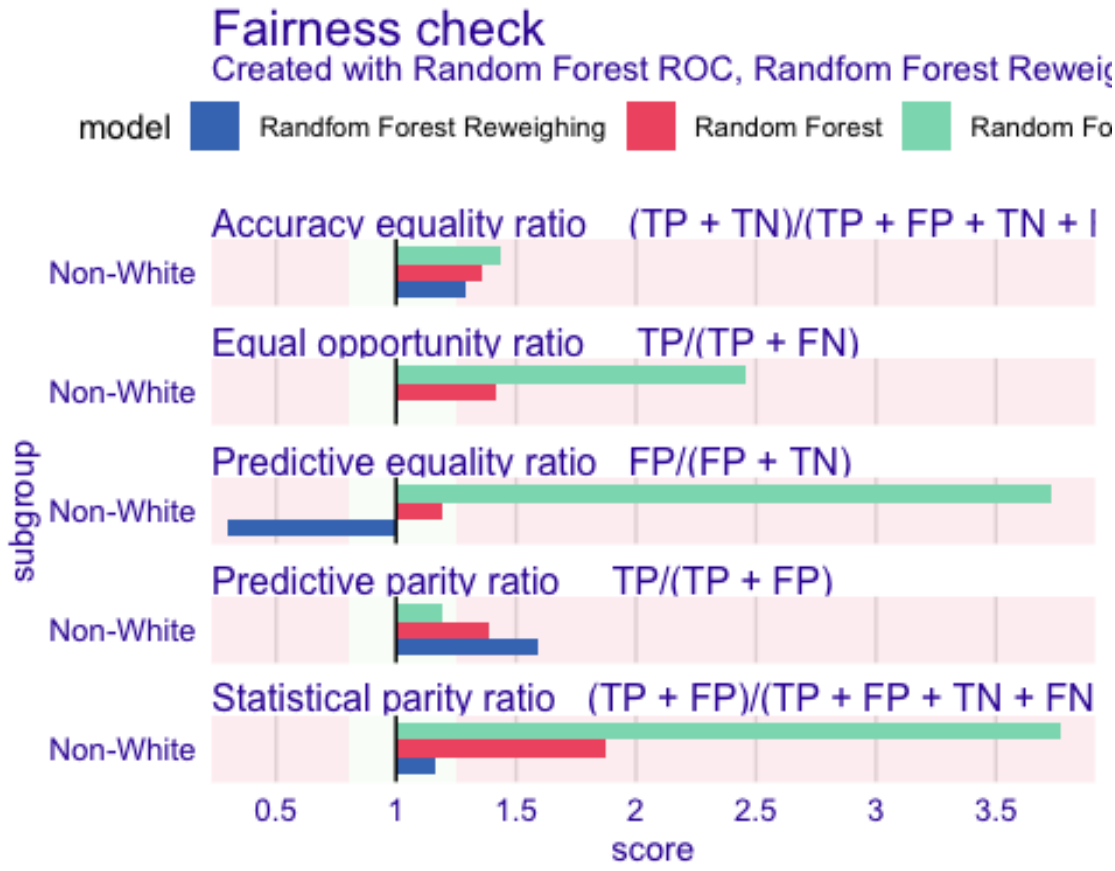

```
# we can print how many fairness checks each mitigation approach passes

print(fobject_roc)
```

```
##
## Fairness check for models: Logistic Regression ROC, Logistic Regression Reweighing,
Logistic Regression
##
## [31mLogistic Regression ROC passes 0/5 metrics
## [39mTotal loss :  5.157979
##
## [31mLogistic Regression Reweighing passes 1/5 metrics
## [39mTotal loss :  3.054479
##
## [31mLogistic Regression passes 0/5 metrics
## [39mTotal loss :  5.157979
```

```
# using the existing explainer object, we can directly run the ROC method on the already
created predictions
randomforest_explainer_roc <- roc_pivot(rf_explainer,
                          protected = protected,
                          privileged = "White",
                          cutoff = 0.5,
                          theta = 0.1)

# then, we can create the fairness object that calculates various fairness metrics
# here, we use the Gender as sensitive attribute
fobject_randomforest_roc <- fairness_check(fobject_randomforest_reweighing,
                         randomforest_explainer_roc,
                         label="Random Forest ROC",
                         protected = protected,
                         privileged = "White")
```

```
## Creating fairness classification object
## -> Privileged subgroup       : character ([32m Ok [39m )
## -> Protected variable        : factor ([32m Ok [39m )
## -> Cutoff values for explainers  : 0.5 ( for all subgroups )
## -> Fairness objects      : 1 object ( [32m compatible [39m )
## -> Checking explainers       : 3 in total ( [32m compatible [39m )
```

```
## -> Metric calculation       : 13/13 metrics calculated for all models
## [32m Fairness object created succesfully [39m

# print some of the fairness metrics
plot(fobject_randomforest_roc)
```




```
# we can print how many fairness checks each mitigation approach passes

print(fobject_randomforest_roc)

##
## Fairness check for models: Random Forest ROC, Randfom Forest Reweighing, Random
Forest
##
## [31mRandom Forest ROC passes 1/5 metrics
## [39mTotal loss :  7.589487
##
```

```
## [31mRandfom Forest Reweighing passes 2/5 metrics
## [39mTotal loss :  1.74373
##
## [31mRandom Forest passes 1/5 metrics
## [39mTotal loss :  2.23144
```

Discussion:

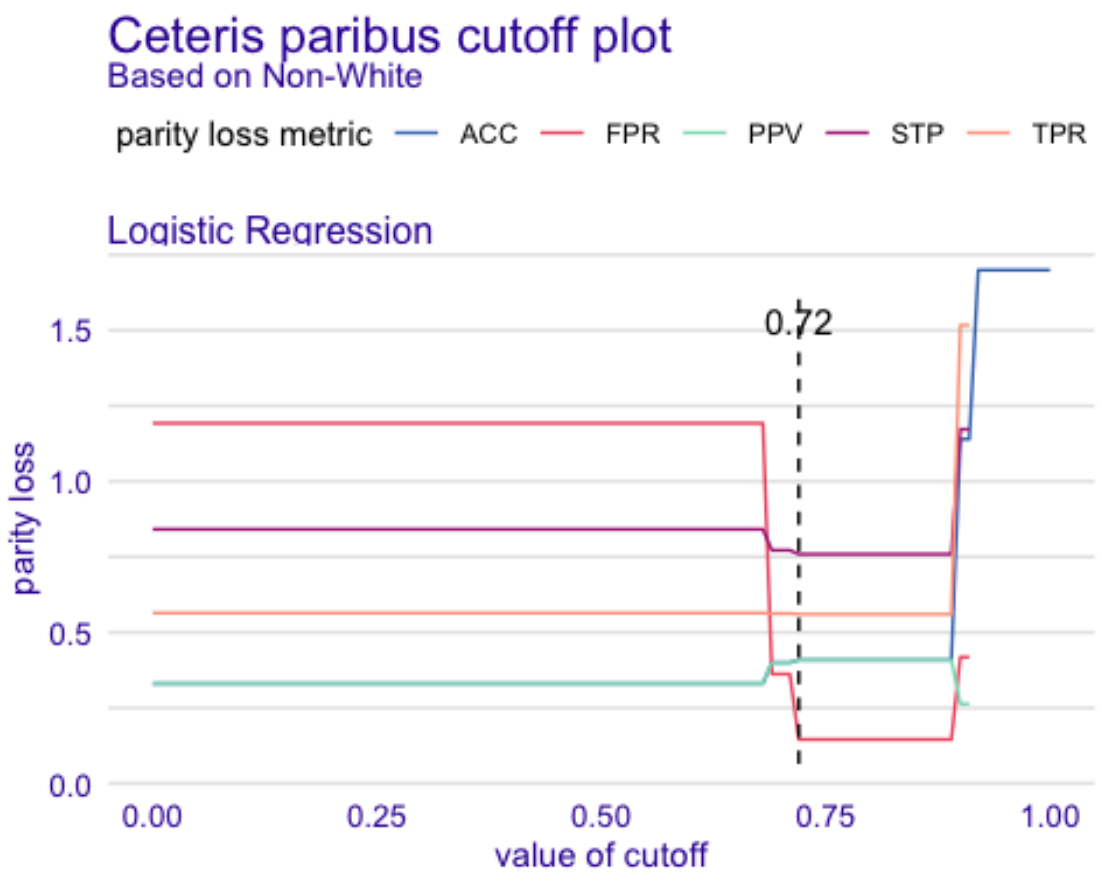
For the logistic regression, we can see that the performance of the ROC post-processing is mixed. We see some minor improvements in three areas: Accuracy Equality Ratio, Equal Opportunity Ratio, and Predictive Parity Ratio. The improvements are very small though and the metrics are almost on par with at least one of the other models. In the case of the statistical parity ratio, it performs worse than the normal logistic regression. Finally, it does not offer improvements of the reweighing in the Predictive Equality Ratio, where it performs much worse. Overall, there is little reason to prefer it over the reweighing.

The ROC works similarly for the random forest. In one area, the Predictive Parity Ratio, it is a large improvement, bringing the metric below 1.2. The other ratios look worse for the ROC, where it is mostly outperformed by two if not all three of the other models. There is again little reason to prefer it over the reweighting, for example.

5.2.2.2 Cut-off

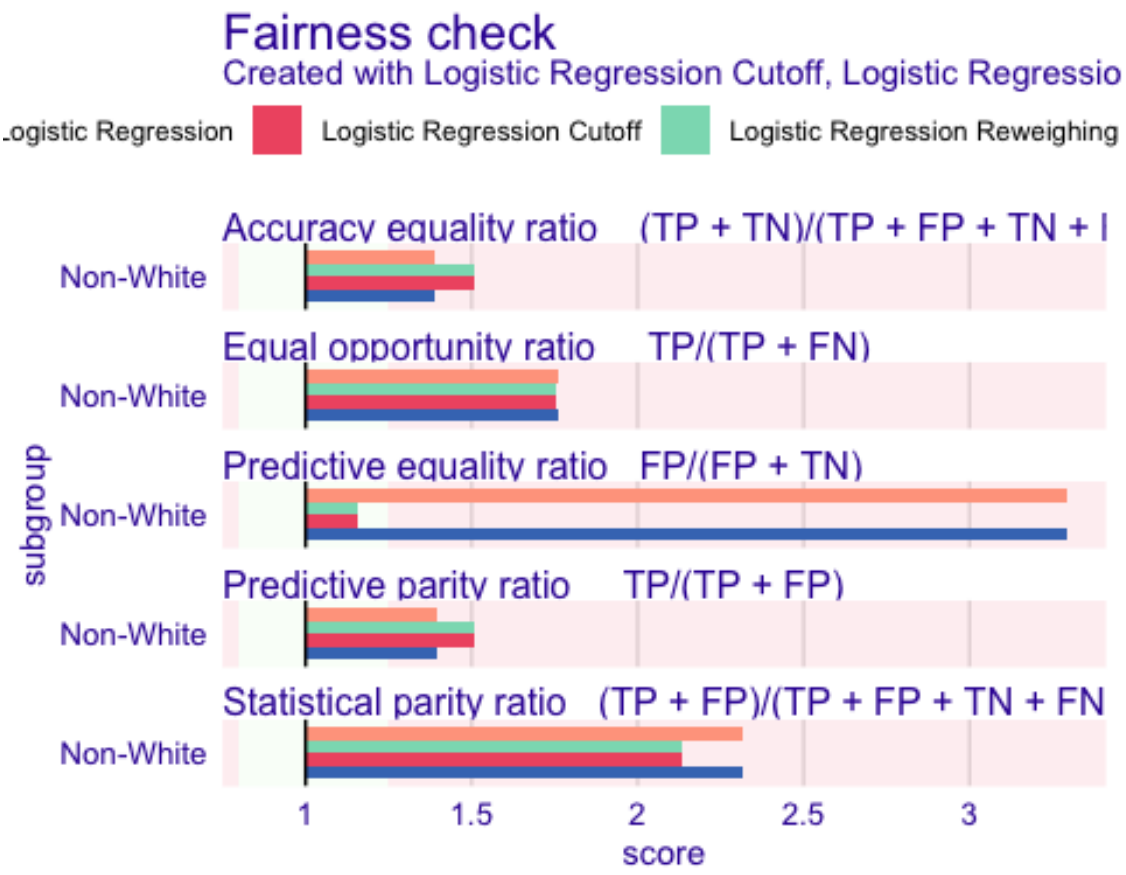
Logistic-Regression

```
#install.packages("ggrepel")
library(ggrepel)
# for this, we can "learn" the best cutoffs based on the existing fairness object
cutoff_analysis <- ceteris_paribus_cutoff(fobject,
                                          subgroup = "Non-White")

plot(cutoff_analysis)

## Warning: Removed 36 rows containing missing values (`geom_line()`).
```




```
# we can get the "optimal" cutoff and plot the resulting fairness metrics again
# note: here, the optimal cutoff can be directly accessed through
"cutoff_analysis$min_data$mins"
fairness_check_cutoff <- fairness_check(logreg_explainer,
                                        fobject_roc,
                                        label="Logistic Regression Cutoff",
                                        cutoff = list("Non-White" = cutoff_analysis$min_data$mins))

## Creating fairness classification object
## -> Privileged subgroup       : character ([33m from first fairness object [39m )
## -> Protected variable        : factor ([33m from first fairness object [39m )
## -> Cutoff values for explainers  : Non-White: 0.72, White: 0.5
## -> Fairness objects      : 1 object ( [32m compatible [39m )
## -> Checking explainers       : 4 in total ( [32m compatible [39m )
## -> Metric calculation        : 13/13 metrics calculated for all models
## [32m Fairness object created succesfully [39m
```

```
plot(fairness_check_cutoff)
```




```
print(fairness_check_cutoff)

##
## Fairness check for models: Logistic Regression Cutoff, Logistic Regression ROC, Logistic Regression Reweighing, Logistic Regression
##
## [31mLogistic Regression Cutoff passes 1/5 metrics
## [39mTotal loss :  3.054479
##
## [31mLogistic Regression ROC passes 0/5 metrics
## [39mTotal loss :  5.157979
##
## [31mLogistic Regression Reweighing passes 1/5 metrics
## [39mTotal loss :  3.054479
```

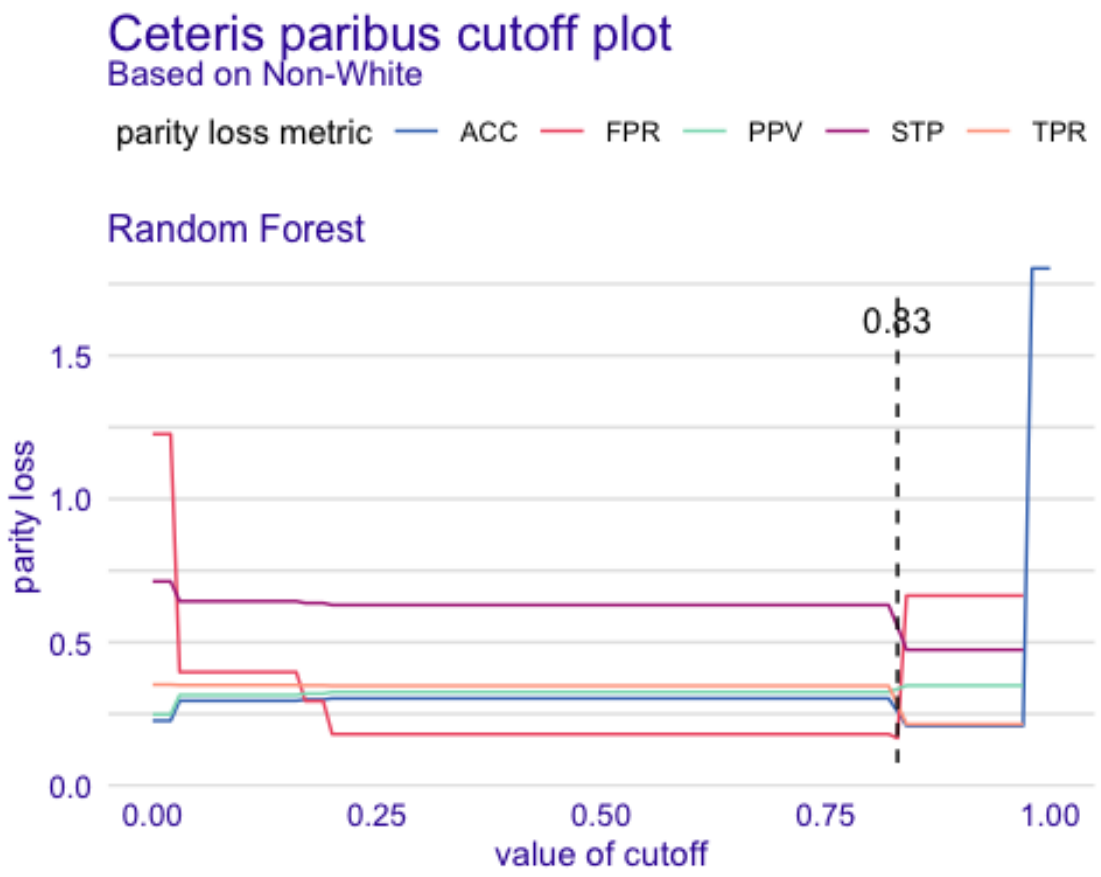
```
##
## [31mLogistic Regression passes 0/5 metrics
## [39mTotal loss :  5.157979
```

The logistic regression cut-off performs analogously to the reweighing.

Random Forest

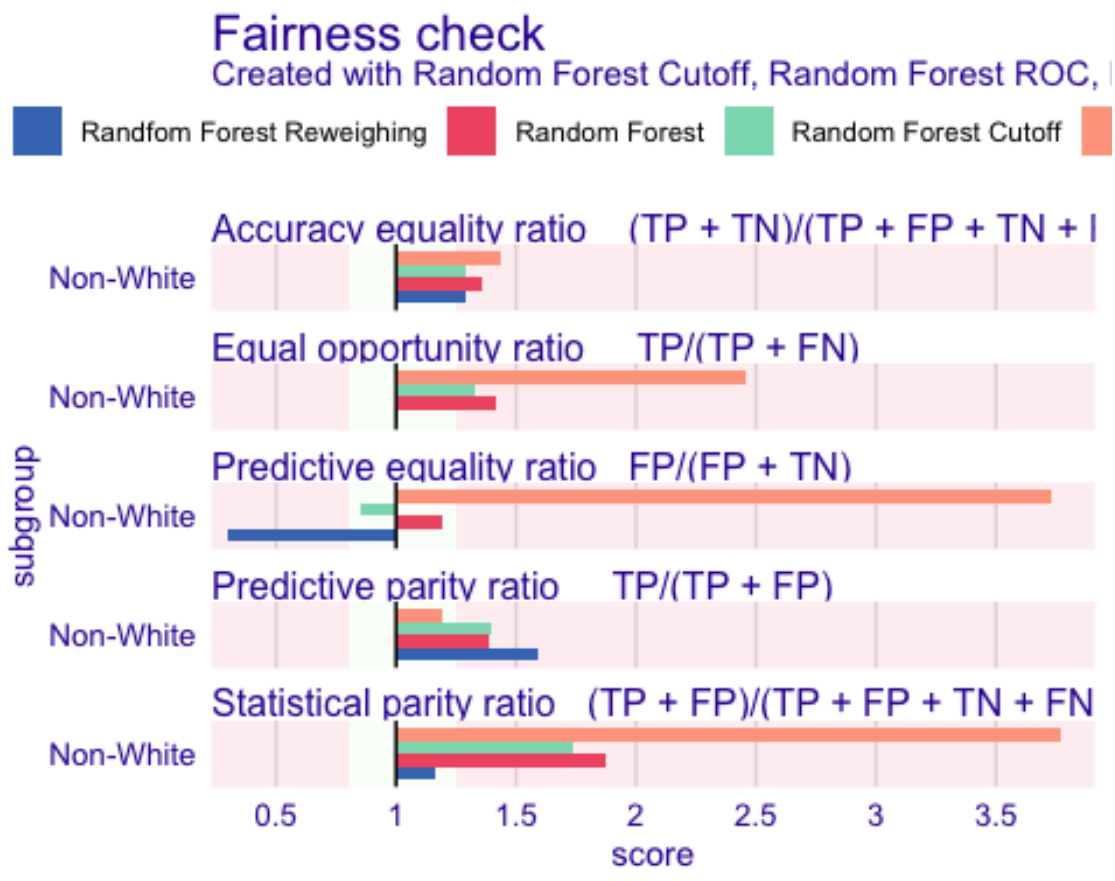
```
# for this, we can "learn" the best cutoffs based on the existing fairness object
cutoff_analysis <- ceteris_paribus_cutoff(fobject_randomforest,
                                          subgroup = "Non-White")

plot(cutoff_analysis)
```

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```




```
# we can get the "optimal" cutoff and plot the resulting fairness metrics again
# note: here, the optimal cutoff can be directly accessed through
"cutoff_analysis$min_data$mins"
```

```
fairness_check_cutoff <- fairness_check(rf_explainer,
                                        fobject_randomforest_roc,
                                        label="Random Forest Cutoff",
                                        cutoff = list("Non-White" = cutoff_analysis$min_data$mins))

## Creating fairness classification object
## -> Privileged subgroup       : character ([33m from first fairness object [39m )
## -> Protected variable        : factor ([33m from first fairness object [39m )
## -> Cutoff values for explainers  : Non-White: 0.83, White: 0.5
## -> Fairness objects      : 1 object ( [32m compatible [39m )
## -> Checking explainers       : 4 in total ( [32m compatible [39m )
## -> Metric calculation        : 13/13 metrics calculated for all models
## [32m Fairness object created succesfully [39m

plot(fairness_check_cutoff)
```




```
print(fairness_check_cutoff)
```

```
##
## Fairness check for models: Random Forest Cutoff, Random Forest ROC, Randfom Forest
Reweighing, Random Forest
##
## [31mRandom Forest Cutoff passes 1/5 metrics
## [39mTotal loss :  1.918016
##
## [31mRandom Forest ROC passes 1/5 metrics
## [39mTotal loss :  7.589487
##
## [31mRandfom Forest Reweighing passes 2/5 metrics
## [39mTotal loss :  1.74373
##
## [31mRandom Forest passes 1/5 metrics
## [39mTotal loss :  2.23144
```

By contrast, the cutoff offers a viable alternative when it comes to fairness for the random forest. While it never offers the best ratio, its fairness always is moderate, with little extreme outliers. It is an option when looking for more consistently moderate fairness metrics.

5.3 Discussion

When investigating fairness with race as a protected class, we see that all fairness metrics are indicating that there are issues with regard to bias. This is more the case of the logistic regression than for the random forest, still it is prevalent in the latter. Looking at pre-processing, we can see that DIR is not useful for either of the models. By contrast, the Reweighing is useful and improves many fairness metrics for both models. However, these improvements come with trade-offs, as few metrics get significantly worse and general model performance worsens.

We see mixed results with post-processing models. While we see some improvements, we also see some metrics worsen significantly with ROC. For both models, we see little reason to prefer this method. This looks differently when using the cut-off method. While it performs analogously for the logistic regression, it provides a more balanced combination of fairness metrics for the random forest. Overall, I suggest using the reweighing strategy for both models, since it improves most of the metrics by the most significant amount, but it is not an easy decision.

6. Conclusion

6.1 Summary

This extensive case demonstrates several things. It first shows how to simulate a simple participatory budgeting vote example based on normally distributed demographic choices. It was very interesting to me how by changing seemingly small things, like demographics or the standard deviation of the vote, completely changes the outcome. The result which seemed seemingly safe by thousands of votes is really fickle and can flip with only a few little changes to the demographics.

This provides many practical insights into the democratic process:

- That groups voting more homogeneously has a large impact on the preferences of the group being elected, even and especially if the group is a minority. In the example, the non-white population managed to elect project 1 three times (which is often their preferred project), besides being a minority.
- Promoting voting, especially for the more homogeneous group is most effective
- From one party's point of view, making the opposite group more heterogeneous in their voting could be as effective as making the own group more homogeneous
- The above also goes for encouraging/discouraging voter participation
- The impact of the allocation of voters to voting districts (see gerrymandering)

When it comes to ML, we can look closer at sections 2 and 3. The example provides valuable insights into the difference between the logistic regression and the random forest model. In section 2, the random forest was using a combination of variables including "project" to find a way to perfectly predict the result, presumably the combination of "Vote" and "Project". The linear regression made very little use of the "project" variable and had a fair prediction quality. We can assume this is due to the decision trees in the random forest, while the linear regression only estimates the impact of individual variables summed together.

In section 3, we removed the "Project" variable. While the linear regression was not affected by the change, the random forest model changes from being a perfect prediction model to only a good prediction model. This confirms the hypothesis from section 2. Overall, we can see that the random forest model is generally outperforming the logistic regression in the prediction quality. It is therefore recommended to use the random forest model.

in section 4, we discuss explainability. We can see that race has a larger impact in the random forest than in the logistic regression. We hypothesize that the random forest was able to see the connection between race and vote and actually attribute much more to the vote than the logistic regression. Sex has little significance in both models, confirming its validity as it was randomly generated. Vote is equally important in the two models. We can

see that the urbanization is more important for the random forest, we can assume because it has made a connection between the urbanization and the project.

Local explainability, in section 4, generally confirms the findings above. Overall, the random forest model puts more emphasis on the urbanization variable. Further, it is more flexible on the race variable and judges it depending on the other variables. Generally, the random forest predictions are less extreme compared to the logistic regression.

In section 5, we first investigate fairness. We find that all metrics are showing strong discrimination again the non-white subgroup, slightly less so for the random forest. The DIR was found not to be effective for either of the model, the reweighing, however, was promising. It was generally chosen as the best way to improve fairness with the model. Both post-processing methods were little effective. Reweighing was suggested as the go-to method for improving fairness in both models, however, trade-offs between several metrics and the general prediction quality of the models exist, naturally.

6.2 Outlook

We propose two models with fair to good prediction quality according to a series of metrics. They could be used to predict the realization of one's preference without knowing the result, assuming the demographics are similar. It further could be used to simulate a change of demographics and vote results to calculate a new vote_realized. Finally, looking at explainability can become beneficial, as one could investigate the impact of different variables on vote_realized and use that information, for example, to lobby different voter groups.

Fairness is an issue in the models and there is little impact of bias mitigation techniques. The best way to improve fairness was reweighing, for both models. Yet, it still leaves some of the metrics in a critically high area. This can become an issue if the model is used for prescriptive purposes and not for explanatory and descriptive purposes, as it has been designed to do. Such a model could be beneficial for voter groups, politicians, policy-makers, political scientists, and polling organizations alike.