denis_stashkevich_hr_attrition

Denis Stashkevich

2023-01-02

```
rm(list = ls())
suppressWarnings(suppressPackageStartupMessages(library(tidyverse)))
suppressWarnings(suppressPackageStartupMessages(library(data.table)))
suppressWarnings(suppressPackageStartupMessages(library(rcompanion)))
suppressWarnings(suppressPackageStartupMessages(library(effsize)))
suppressWarnings(suppressPackageStartupMessages(library(roperators)))
suppressWarnings(suppressPackageStartupMessages(library(future.apply)))
suppressWarnings(suppressPackageStartupMessages(library(dqrng)))
suppressWarnings(suppressPackageStartupMessages(library(tidymodels)))
suppressWarnings(suppressPackageStartupMessages(library(themis)))
suppressWarnings(suppressPackageStartupMessages(library(caret)))
suppressWarnings(suppressPackageStartupMessages(library(treemapify)))
suppressWarnings(suppressPackageStartupMessages(library(gridExtra)))
suppressWarnings(suppressPackageStartupMessages(library(pbapply)))
suppressWarnings(suppressPackageStartupMessages(library(FactoMineR)))
```

```
IBM_HR_data_raw <- fread("WA_Fn-UseC_-HR-Employee-Attrition.csv")
```

We drop these columns since they do not contain any information

```
IBM_HR_data_proc <-IBM_HR_data_raw %>%
    dplyr::select(-Over18, -EmployeeNumber, -EmployeeCount, -StandardHours)
```

To select relevant variables, we write a function that would iterate over columns and apply an appropriate statistical test

1. For categorical data we choose chi-squared test as a test and Cohen's w to measure effect size. Other choices may include G-test as a test and Goodman and Kruskal's lambda or Cramers'V to measure effect size

2. For numeric data we choose Wilcoxon rank sum test since we have only two groups. Vargha and Delaney's A would be used to measure effect size. There are many other options for measuring effect size, but VDA is very easy to interpret.

3. We correct p-values for the inflated probability of making type I error by using False Discovery Rate method. There are many other methods to do the correction but FDA is regarded as a sensible approach in general case.

```
wilcox_test <- function(cat_vec, num_vec){
  categories <- cat_vec %>% unique()
```

```r
  stopifnot(length(categories) == 2)

  x <- num_vec[cat_vec == categories[1]]
  y <- num_vec[cat_vec == categories[2]]

  return(wilcox.test(x,y)$p.value)
}

vda <- function(cat_vec, num_vec){
  categories <- cat_vec %>% unique()
  stopifnot(length(categories) == 2)

  x <- num_vec[cat_vec == categories[1]]
  y <- num_vec[cat_vec == categories[2]]

  return(effsize::VD.A(x,y)$estimate)
}


explore_possible_effects <- function(data_df,
                                     column_to_compare,
                                     other_cols,
                                     cat_test_fun = function(x,y) chisq.test(x,y)$p.value,
                                     cat_effect_size = rcompanion::cohenW,
                                     num_test_fun = wilcox_test,
                                     num_effect_size = vda,
                                     p.adjust_method = "fdr",  #"holm" "hochberg" "hommel" "bonferroni"
                                     alpha = 0.001
){

  inter_VDA   <- function(x){
      case_when(
        x %between% c(0.56, 0.64) || x %between% c(0.34, 0.44) ~ "Small effect",
        x %between% c(0.64, 0.71) || x %between% c(0.29, 0.34) ~ "Medium effect",
        x >0.71 || x < 0.29 ~ "Large effect",
        TRUE ~ "No effect or negligible"
      )
    }

  inter_CohenW <- function(x){
      case_when(
        x < 0.1 ~ "No effect or negligible",
        x %between% c(0.1, 0.3) ~ "Small effect",
        x %between% c(0.3, 0.5) ~ "Medium effect",
        x > 0.5 ~ "Large effect"
      )
    }

  compare_two_cols <- function(name_col_a, name_col_b, col_a,col_b){

    stopifnot(class(col_a) %in% c("factor", "character"))

    if(class(col_a) == "factor"){
```

```r
    col_a = col_a %>% as.character()
  }

  if(class(col_b) %in% c("numeric", "integer")){

      p_val = num_test_fun(col_a,col_b)
      ef_size <- num_effect_size(col_a,col_b)

        return(data.frame(
          "Col A" = name_col_a,
          "Col B" = name_col_b,
          "type" = "numeric",
          "test" = "Wilcoxon rank sum test",
          "effect size" = "Vargha and Delaney's A",
          "p-value" = p_val,
          "effect size value" = ef_size,
          "effect size interpretation" = inter_VDA(ef_size)
        ))
  }else{
      col_b = col_b %>% as.character()

      p_val = cat_test_fun(col_a,col_b)
      ef_size <- cat_effect_size(col_a,col_b)

        return(data.frame(
          "Col A" = name_col_a,
          "Col B" = name_col_b,
          "type" = "categorical",
          "test" = "Chi-Squared test",
          "effect size" = "Cohen W",
          "p-value" = p_val,
          "effect size value" = ef_size,
          "effect size interpretation" = inter_CohenW(ef_size)
        ))

  }
}

###########################

res_list <- vector("list", length(other_cols))
iter <- 1
for(col_ in other_cols){
  res_list[[iter]] <- compare_two_cols(name_col_a = column_to_compare,
                                       name_col_b = col_,
                                       col_a = data_df[[column_to_compare]],
                                       col_b = data_df[[col_]])
  iter %+=% 1
}

res_dt <- rbindlist(res_list)
res_dt[["p-value_adj"]] <- p.adjust(res_dt[["p.value"]], method = p.adjust_method)
res_dt[["statistically significant with chosen alpha"]] <- ifelse(res_dt[["p-value_adj"]] > alpha, "N
```

```
  return(res_dt)

}
```

```
explore_possible_effects(data_df = IBM_HR_data_proc,
                         column_to_compare = "Attrition",
                         other_cols = setdiff(colnames(IBM_HR_data_proc), "Attrition"))
```

```
## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect
```

```
##         Col.A                    Col.B        type                   test
##  1: Attrition                      Age     numeric Wilcoxon rank sum test
##  2: Attrition           BusinessTravel categorical       Chi-Squared test
##  3: Attrition                DailyRate     numeric Wilcoxon rank sum test
##  4: Attrition               Department categorical       Chi-Squared test
##  5: Attrition         DistanceFromHome     numeric Wilcoxon rank sum test
##  6: Attrition                Education     numeric Wilcoxon rank sum test
##  7: Attrition           EducationField categorical       Chi-Squared test
##  8: Attrition  EnvironmentSatisfaction     numeric Wilcoxon rank sum test
##  9: Attrition                   Gender categorical       Chi-Squared test
## 10: Attrition               HourlyRate     numeric Wilcoxon rank sum test
## 11: Attrition           JobInvolvement     numeric Wilcoxon rank sum test
## 12: Attrition                 JobLevel     numeric Wilcoxon rank sum test
## 13: Attrition                  JobRole categorical       Chi-Squared test
## 14: Attrition          JobSatisfaction     numeric Wilcoxon rank sum test
## 15: Attrition            MaritalStatus categorical       Chi-Squared test
## 16: Attrition            MonthlyIncome     numeric Wilcoxon rank sum test
## 17: Attrition              MonthlyRate     numeric Wilcoxon rank sum test
## 18: Attrition       NumCompaniesWorked     numeric Wilcoxon rank sum test
## 19: Attrition                 OverTime categorical       Chi-Squared test
## 20: Attrition        PercentSalaryHike     numeric Wilcoxon rank sum test
## 21: Attrition        PerformanceRating     numeric Wilcoxon rank sum test
## 22: Attrition RelationshipSatisfaction     numeric Wilcoxon rank sum test
## 23: Attrition         StockOptionLevel     numeric Wilcoxon rank sum test
## 24: Attrition        TotalWorkingYears     numeric Wilcoxon rank sum test
## 25: Attrition    TrainingTimesLastYear     numeric Wilcoxon rank sum test
## 26: Attrition          WorkLifeBalance     numeric Wilcoxon rank sum test
## 27: Attrition           YearsAtCompany     numeric Wilcoxon rank sum test
## 28: Attrition       YearsInCurrentRole     numeric Wilcoxon rank sum test
## 29: Attrition  YearsSinceLastPromotion     numeric Wilcoxon rank sum test
## 30: Attrition     YearsWithCurrManager     numeric Wilcoxon rank sum test
##         Col.A                    Col.B        type                   test
##               effect.size      p.value effect.size.value
##  1: Vargha and Delaney's A 5.304342e-11         0.3656787
##  2:               Cohen W 5.608614e-06         0.1283000
##  3: Vargha and Delaney's A 2.900458e-02         0.4552787
##  4:               Cohen W 4.525607e-03         0.0857000
##  5: Vargha and Delaney's A 2.387047e-03         0.5619908
##  6: Vargha and Delaney's A 2.448310e-01         0.4772433
##  7:               Cohen W 6.773980e-03         0.1044000
##  8: Vargha and Delaney's A 2.173049e-04         0.4270295
##  9:               Cohen W 2.905724e-01         0.0294500
```

```
## 10: Vargha and Delaney's A 7.976303e-01        0.4947471
## 11: Vargha and Delaney's A 4.651927e-06        0.4173451
## 12: Vargha and Delaney's A 2.956987e-13        0.3583914
## 13:                Cohen W 2.752482e-15        0.2421000
## 14: Vargha and Delaney's A 7.957918e-05        0.4221548
## 15:                Cohen W 9.455511e-11        0.1772000
## 16: Vargha and Delaney's A 2.950831e-14        0.3443301
## 17: Vargha and Delaney's A 5.587481e-01        0.5119772
## 18: Vargha and Delaney's A 2.423651e-01        0.5233351
## 19:                Cohen W 8.158424e-21        0.2461000
## 20: Vargha and Delaney's A 3.655146e-01        0.4815756
## 21: Vargha and Delaney's A 9.119454e-01        0.5014167
## 22: Vargha and Delaney's A 1.020252e-01        0.4677231
## 23: Vargha and Delaney's A 4.013375e-11        0.3750962
## 24: Vargha and Delaney's A 2.399569e-14        0.3441471
## 25: Vargha and Delaney's A 4.729571e-02        0.4612451
## 26: Vargha and Delaney's A 4.647300e-02        0.4644071
## 27: Vargha and Delaney's A 2.916191e-13        0.3510425
## 28: Vargha and Delaney's A 4.429560e-12        0.3600494
## 29: Vargha and Delaney's A 4.117911e-02        0.4598369
## 30: Vargha and Delaney's A 1.806754e-11        0.3639762
##                effect.size      p.value effect.size.value
##     effect.size.interpretation  p-value_adj
##  1:               Small effect 1.591303e-10
##  2:               Small effect 1.294296e-05
##  3:    No effect or negligible 4.579670e-02
##  4:    No effect or negligible 7.986365e-03
##  5:               Small effect 4.475713e-03
##  6:    No effect or negligible 2.937971e-01
##  7:               Small effect 1.128997e-02
##  8:               Small effect 4.346098e-04
##  9:    No effect or negligible 3.352759e-01
## 10:    No effect or negligible 8.251348e-01
## 11:               Small effect 1.162982e-05
## 12:               Small effect 1.478494e-12
## 13:               Small effect 4.128722e-14
## 14:               Small effect 1.705268e-04
## 15:               Small effect 2.578776e-10
## 16:               Small effect 2.213123e-13
## 17:    No effect or negligible 5.986587e-01
## 18:    No effect or negligible 2.937971e-01
## 19:               Small effect 2.447527e-19
## 20:    No effect or negligible 4.061274e-01
## 21:    No effect or negligible 9.119454e-01
## 22:    No effect or negligible 1.330763e-01
## 23:               Small effect 1.337792e-10
## 24:               Small effect 2.213123e-13
## 25:    No effect or negligible 6.449415e-02
## 26:    No effect or negligible 6.449415e-02
## 27:               Small effect 1.478494e-12
## 28:               Small effect 1.898383e-11
## 29:    No effect or negligible 6.176866e-02
## 30:               Small effect 6.775328e-11
##     effect.size.interpretation  p-value_adj
```

```
##     statistically significant with chosen alpha
##  1:                                         Yes
##  2:                                         Yes
##  3:                                          No
##  4:                                          No
##  5:                                          No
##  6:                                          No
##  7:                                          No
##  8:                                         Yes
##  9:                                          No
## 10:                                          No
## 11:                                         Yes
## 12:                                         Yes
## 13:                                         Yes
## 14:                                         Yes
## 15:                                         Yes
## 16:                                         Yes
## 17:                                          No
## 18:                                          No
## 19:                                         Yes
## 20:                                          No
## 21:                                          No
## 22:                                          No
## 23:                                         Yes
## 24:                                         Yes
## 25:                                          No
## 26:                                          No
## 27:                                         Yes
## 28:                                         Yes
## 29:                                          No
## 30:                                         Yes
##     statistically significant with chosen alpha
```

The tests helped us to identify potentially useful variables, namely:

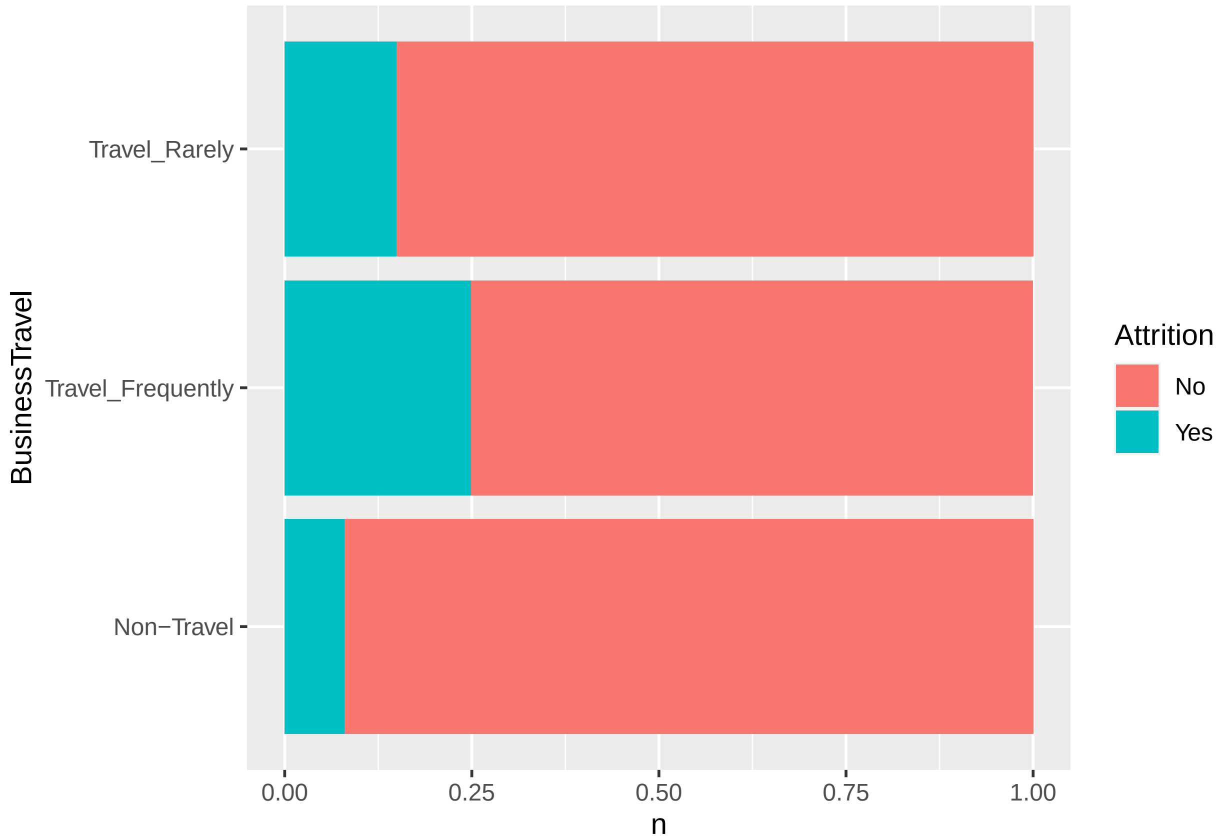
Categorical: BusinessTravel, JobRole, MaritalStatus, OverTime

Numeric: Age, DistanceFromHome, EnvironmentSatisfaction,JobInvolvement, JobLevel, JobSatisfaction, MonthlyIncome, StockOptionLevel, TotalWorkingYears, YearsAtCompany, YearsInCurrentRole, YearsWithCurrManager

We may investigate these variables and their relationship with Attrition more closely

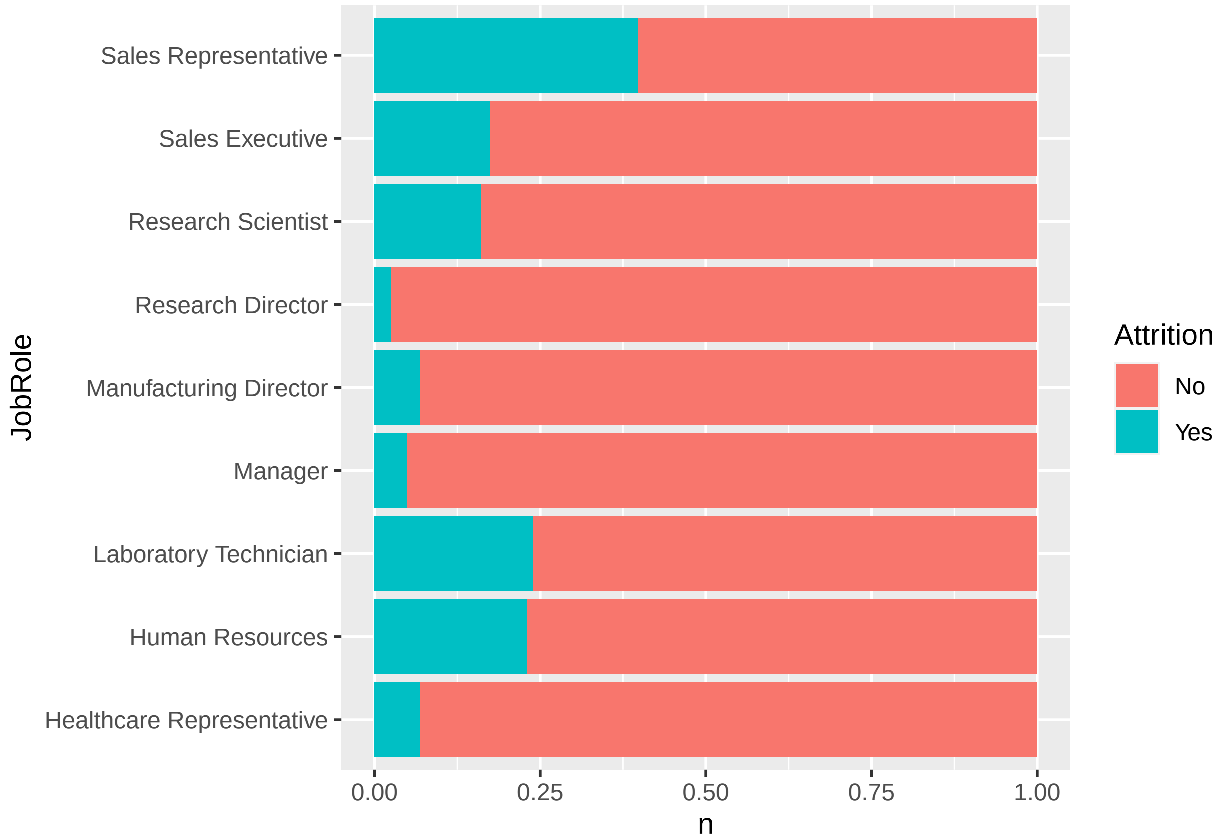
```
make_stacked_barplot <- function(data, x,y){
data %>%
  group_by(!! sym(x),  !! sym(y)) %>%
  count() %>%
  ggplot(aes_string(fill=x, y='n', x=y)) +
  geom_bar(position="fill", stat="identity") +
  coord_flip()
}
```

```
make_stacked_barplot(IBM_HR_data_proc, "Attrition", "BusinessTravel")
```



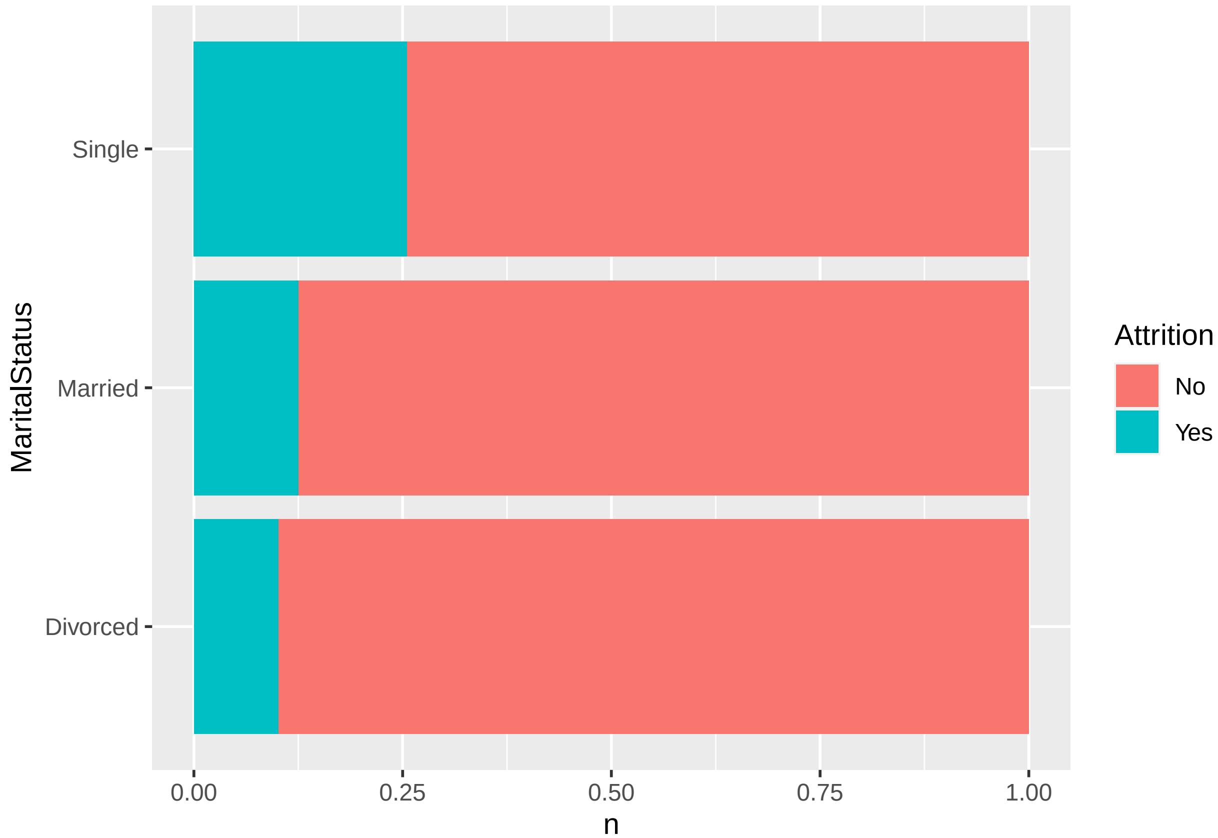
It is evident that there is a clear relationship between the frequency of travelling and Attrition: the more the worker travels, the more likely he/she is to face attrition faster

```
make_stacked_barplot(IBM_HR_data_proc, "Attrition", "JobRole")
```



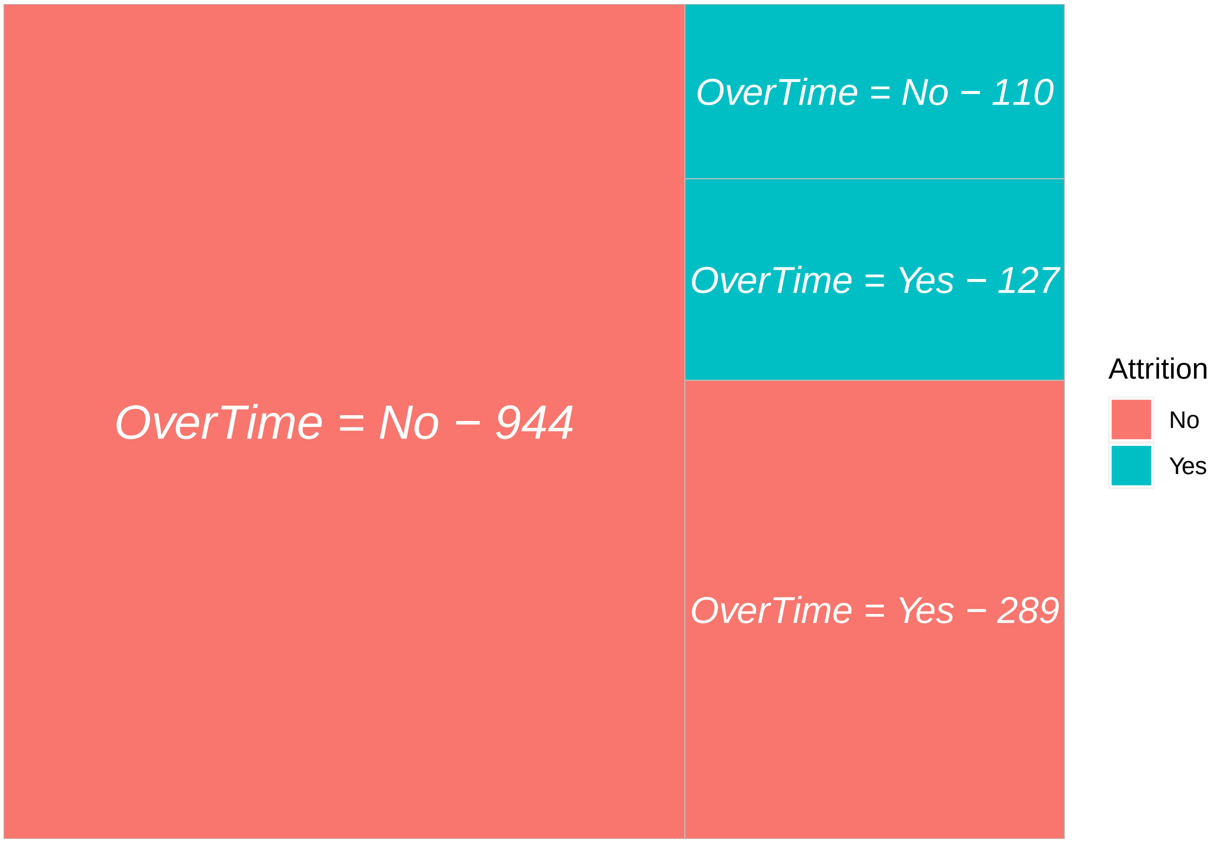
We can observe that different job roles differ in the likelihood of attrition with Sales Representatives being the most likely. However, it is difficult to formulate the exact pattern.

```
make_stacked_barplot(IBM_HR_data_proc, "Attrition", "MaritalStatus")
```



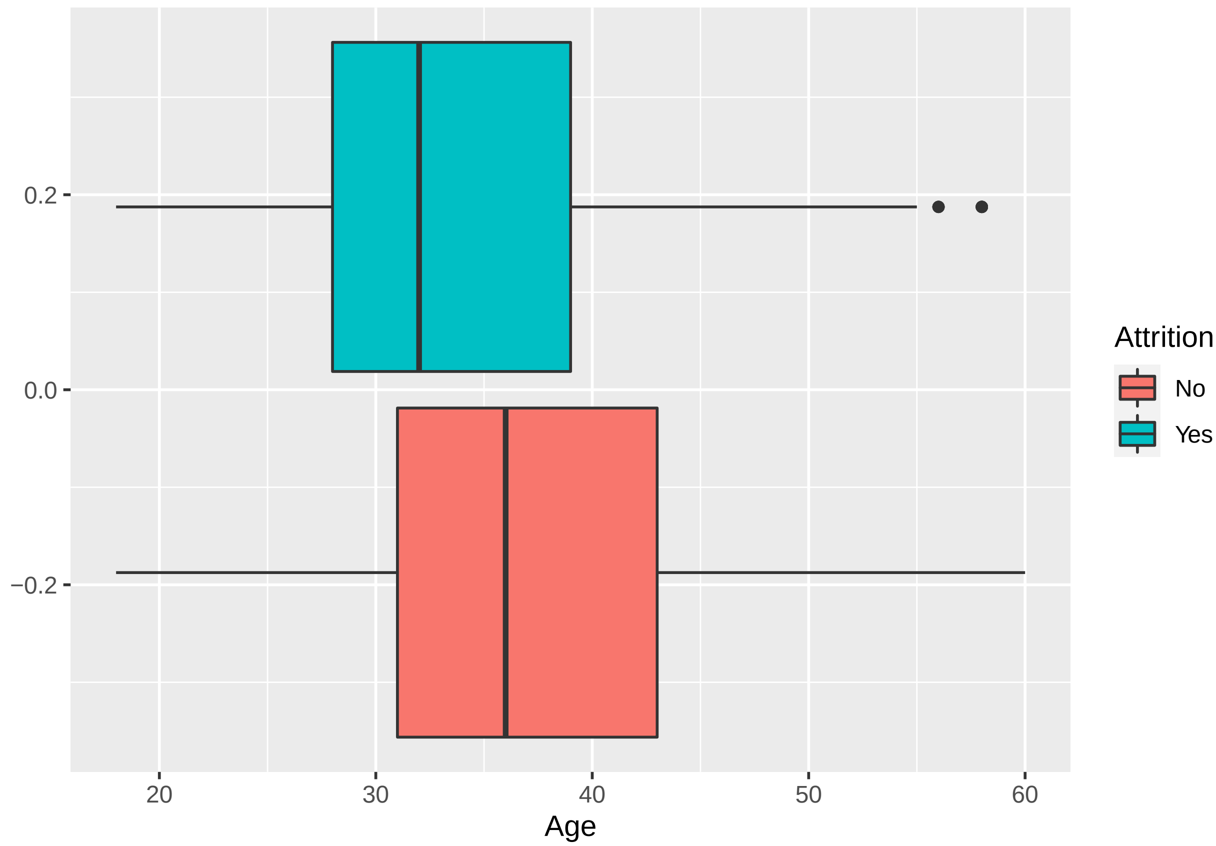
One may conclude that single people are more likely to experience attrition. However, the possible explanation may be that married workers are more dependent on keeping the the job. It is hard to know the exact cause of the observed data.

```
make_a_treemap <- function(data, x,y){
data %>%
  group_by(!! sym(x),  !! sym(y)) %>%
  count() %>%
  mutate(lab = paste(y,'=', !!sym(y), "-", n)) %>%
  ggplot(aes_string(area = 'n', fill = x, label = "lab")) +
  geom_treemap() +
  geom_treemap_text(fontface = "italic", colour = "white", place = "centre")
}

make_a_treemap(IBM_HR_data_proc, "Attrition", "OverTime")
```



From the treemap above it is clear that from all people that experienced attrition - most of them worked overtime. It makes this variable a very good predictor of attrition.

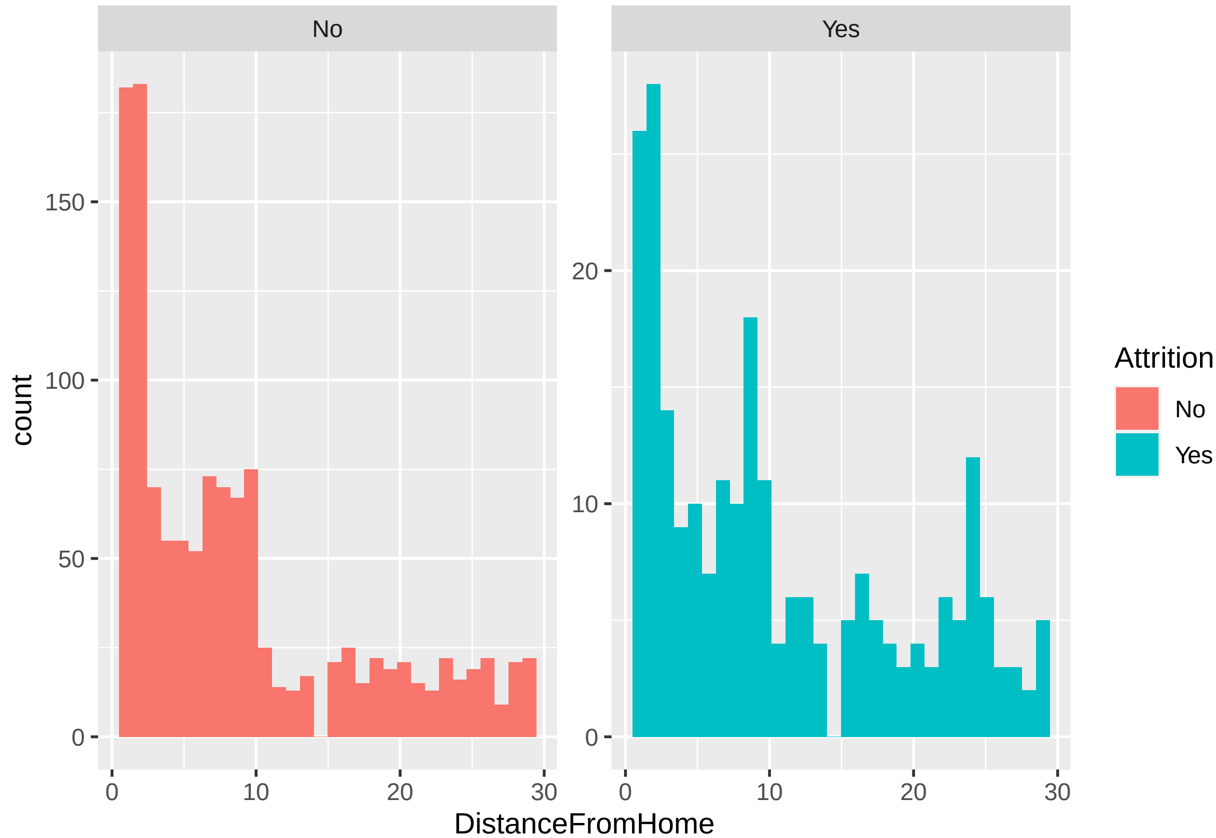
```
IBM_HR_data_proc %>%
  ggplot(aes(x = Age, fill = Attrition)) +
  geom_boxplot()
```



It can be seen that the people that experienced attrition tend to be younger.

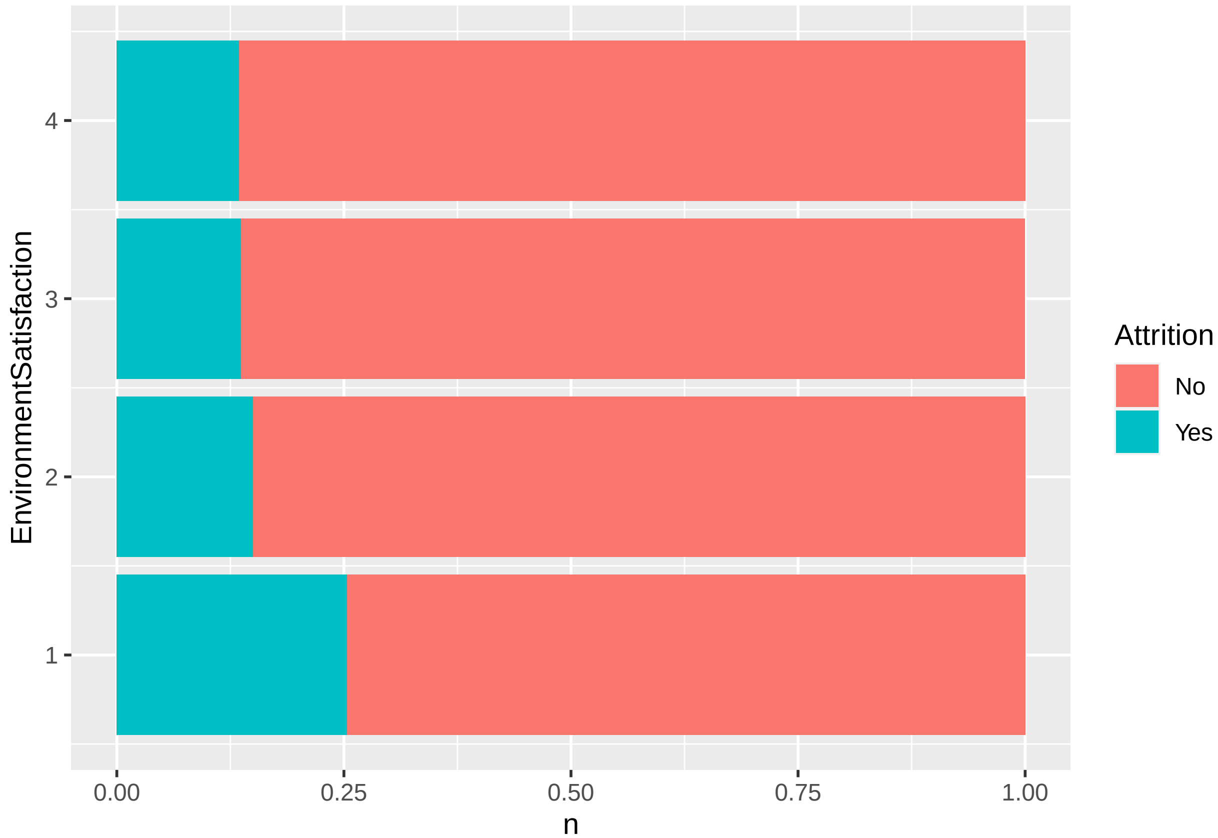
```
IBM_HR_data_proc %>%
  ggplot(aes(x = DistanceFromHome, fill = Attrition)) +
  geom_histogram() +
  facet_wrap(~Attrition, scales = "free_y")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



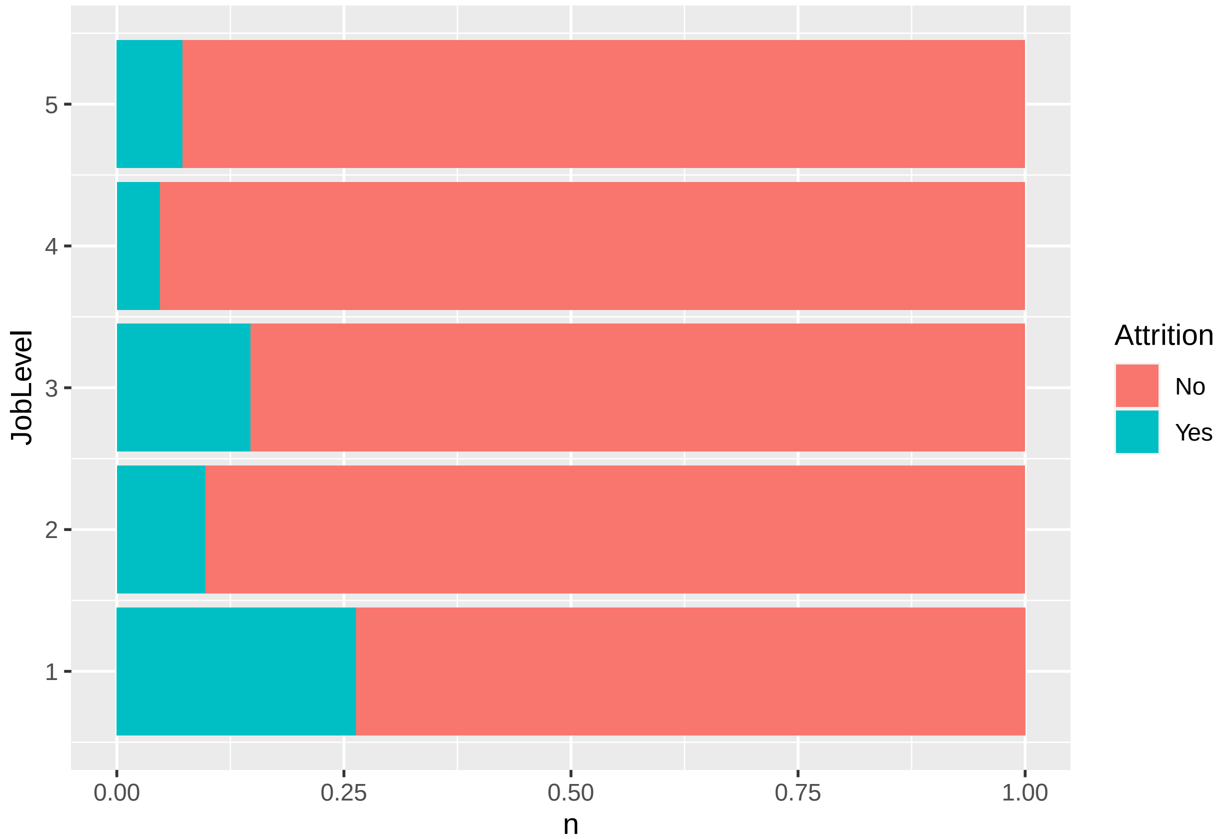
Looking at the histograms above one may observe that the further away a person is from home, the more likely she/he is to experience attrition. The obvious explanation is that people may dislike the lengthy commutes.

```
make_stacked_barplot(IBM_HR_data_proc, "Attrition", "EnvironmentSatisfaction")
```



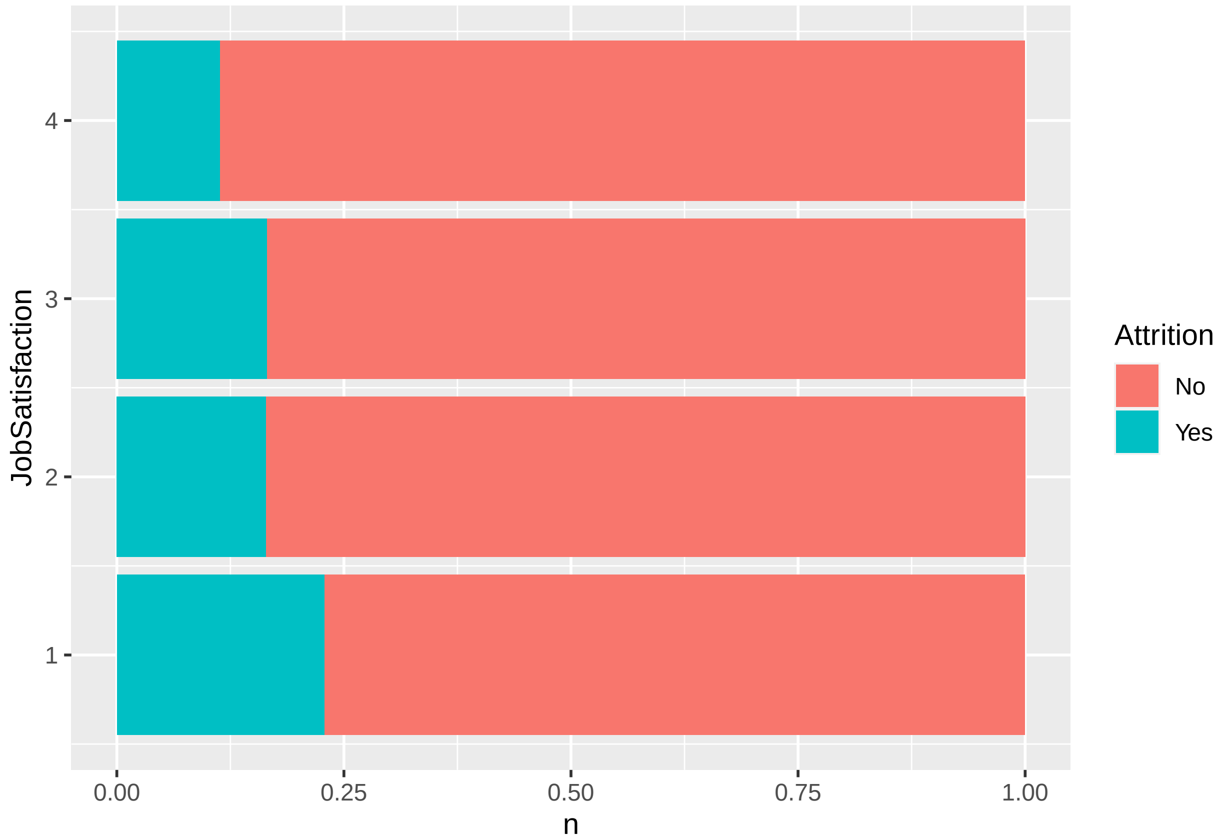
The barplot above suggests an interesting pattern - people that are clearly not satisfied with the environment are more likely to experience attrition. However, the relationship is not linear and for

```
make_stacked_barplot(IBM_HR_data_proc, "Attrition", "JobLevel")
```



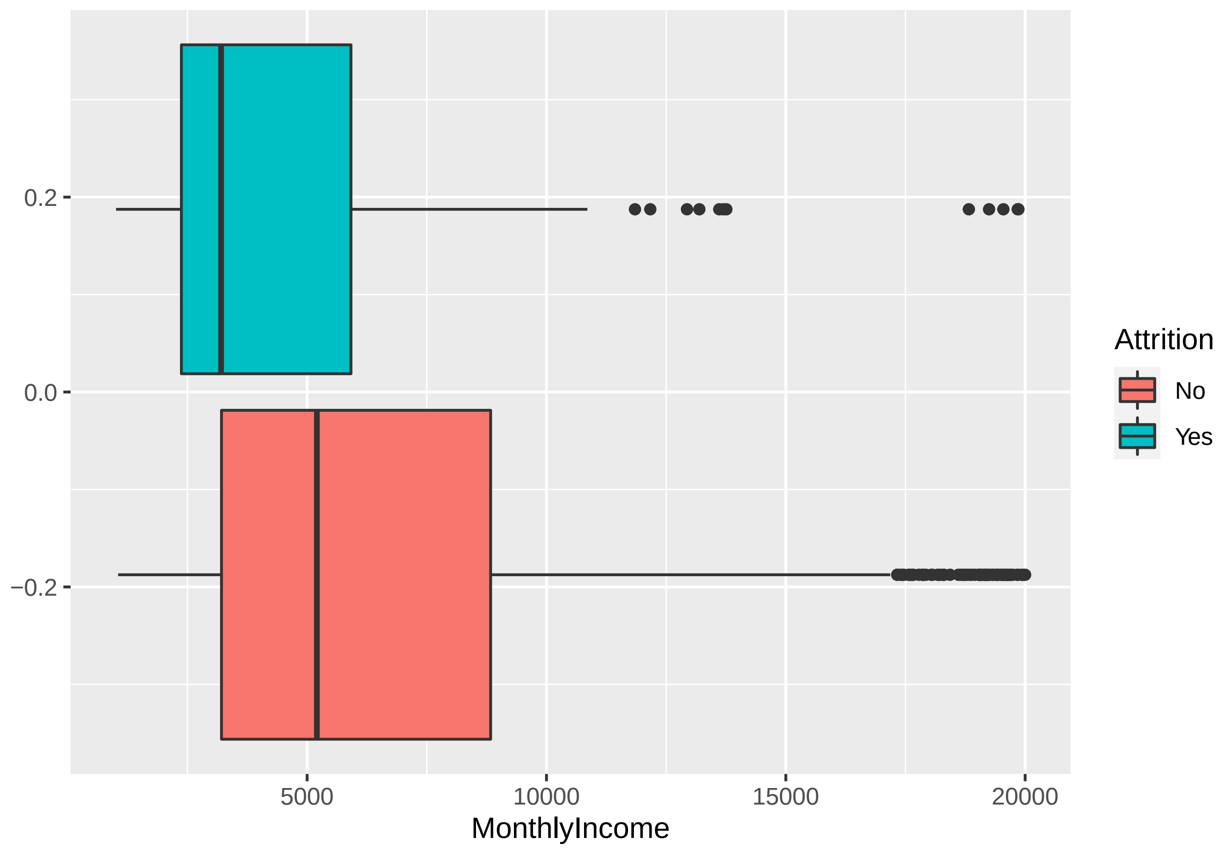
Graph suggests that the likelihood of attrition is dependent on the job level with the lowest in the hierarchy being the most likely to attire. However, the relationship is clearly not linear as can be seen from the proportions for different job levels.

```
make_stacked_barplot(IBM_HR_data_proc, "Attrition", "JobSatisfaction")
```



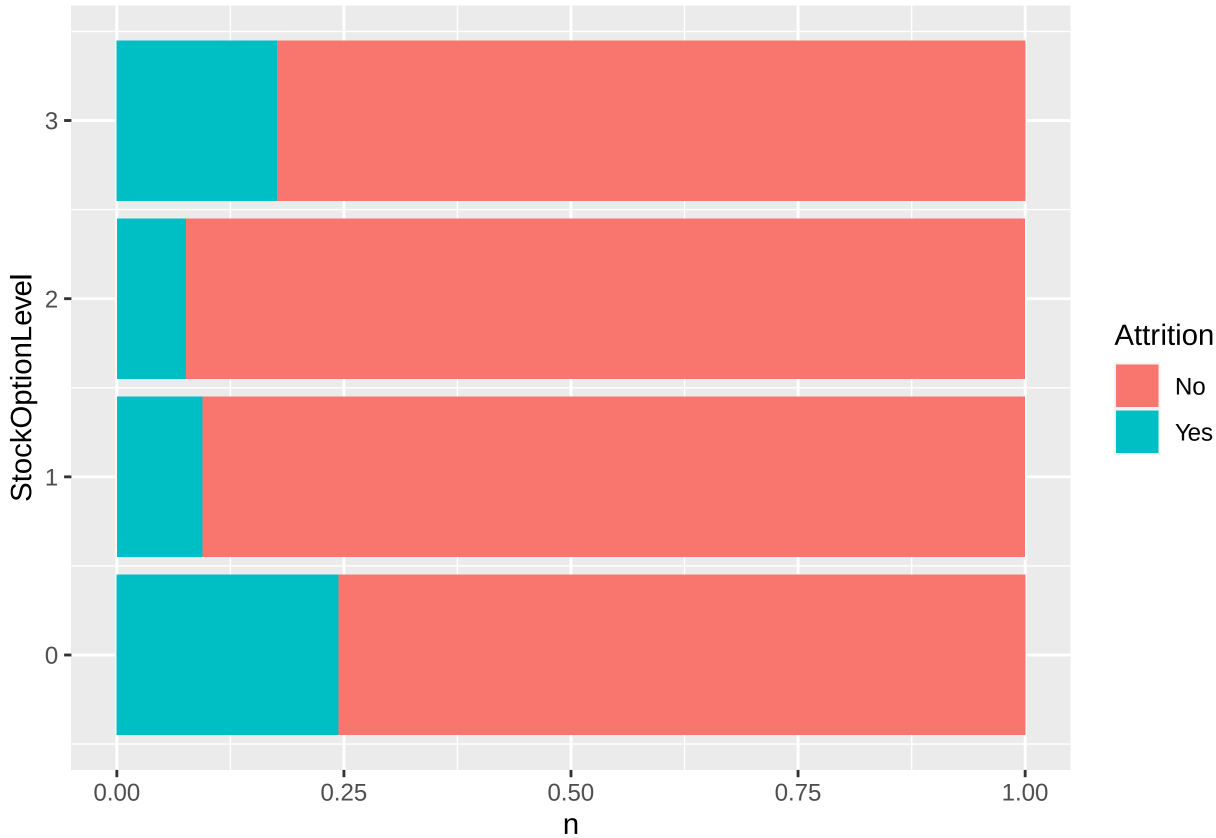
The barplot above simply confirms the common sense - the more a person is satisfied with the job, the less likely he/she is to face attrition.

```
IBM_HR_data_proc %>%
  ggplot(aes(x = MonthlyIncome, fill = Attrition)) +
  geom_boxplot()
```



It seems that money also plays an important role - there is a statistically significant difference between the monthly income of people that did experience attrition and ones that did not.

```
make_stacked_barplot(IBM_HR_data_proc, "Attrition", "StockOptionLevel")
```

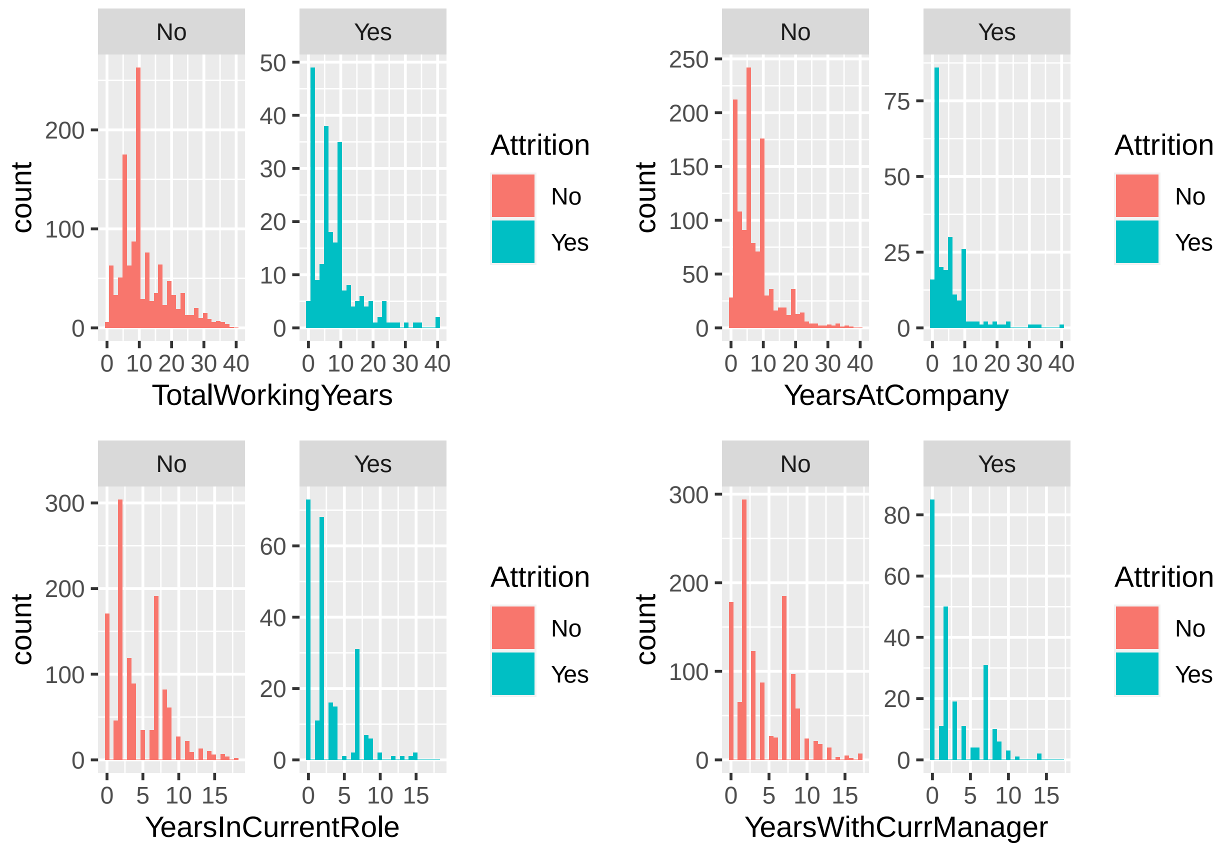


No clear pattern is evident from the graph above other than the fact the the proportions are clearly different.

```
total_w_years <- IBM_HR_data_proc %>%
  ggplot(aes(x = TotalWorkingYears, fill = Attrition)) +
  geom_histogram() +
  facet_wrap(~Attrition, scales = "free_y")

total_years_company <- IBM_HR_data_proc %>%
  ggplot(aes(x = YearsAtCompany, fill = Attrition)) +
  geom_histogram() +
  facet_wrap(~Attrition, scales = "free_y")

total_year_curr_role <- IBM_HR_data_proc %>%
  ggplot(aes(x = YearsInCurrentRole, fill = Attrition)) +
  geom_histogram() +
  facet_wrap(~Attrition, scales = "free_y")

total_year_curr_manager <- IBM_HR_data_proc %>%
  ggplot(aes(x = YearsWithCurrManager, fill = Attrition)) +
  geom_histogram() +
  facet_wrap(~Attrition, scales = "free_y")

grid.arrange(total_w_years, total_years_company, total_year_curr_role, total_year_curr_manager, nrow =
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



All these four histograms show a pretty similar picture - attrition is mostly a problem of relatively new workers in the company.

We also may try some alternative approaches to identifying important variables. One way to do it is through step-wise addition of predictors to model (logistic regression) using AIC to choose models. At this point we do not care about the accuracy of the models, so we do not check assumptions for these models

```
model_null <-       glm(Attrition ~ 1, data = IBM_HR_data_proc %>% mutate(Attrition = Attrition %>% as.
formula_for_scope <-    glm(Attrition ~ ., data = IBM_HR_data_proc %>% mutate(Attrition = Attrition %>%

model_pruned_step_forw   <- stats::step(
    model_null,
    scope = formula_for_scope,
  method = 'forward',
    trace = 0)

formula_pruned <- formula(model_pruned_step_forw)
formula_pruned
```

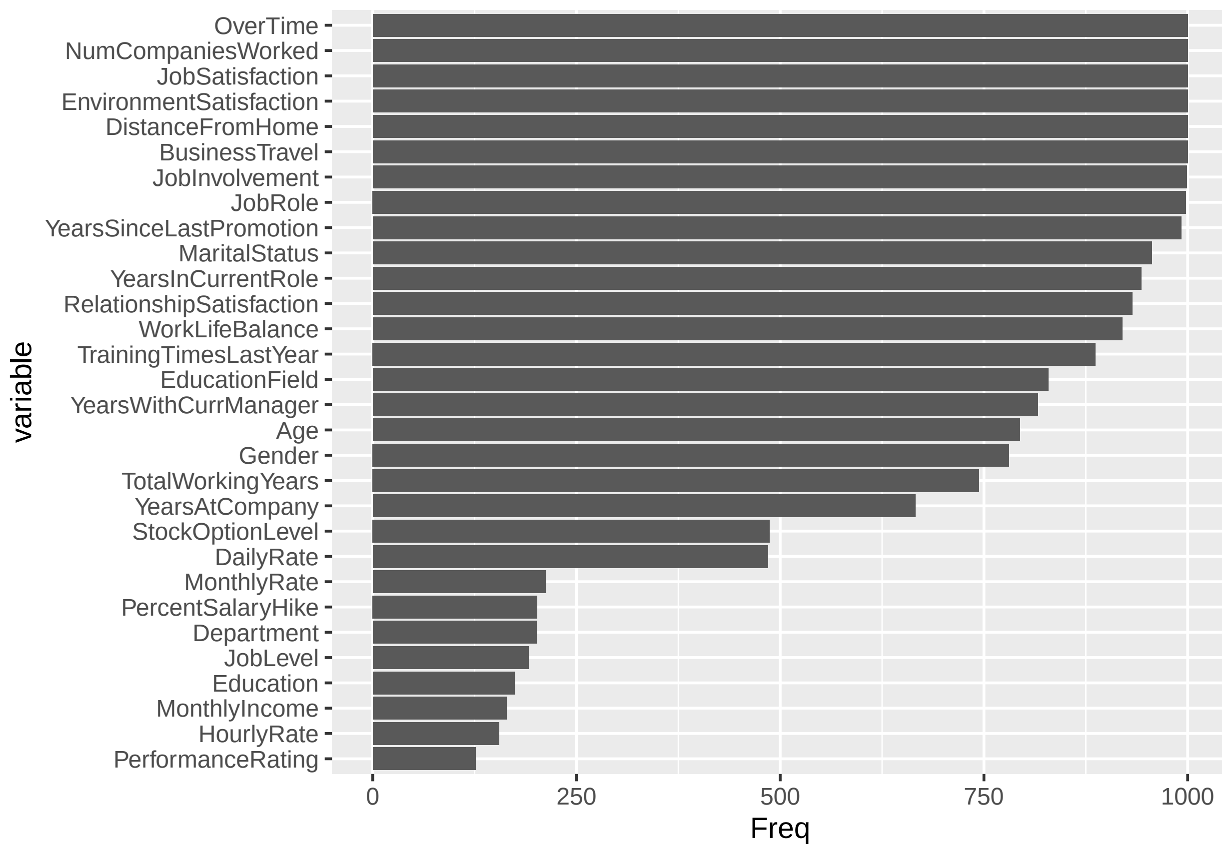
```
## Attrition ~ OverTime + JobRole + MaritalStatus + EnvironmentSatisfaction +
##     JobSatisfaction + JobInvolvement + BusinessTravel + YearsInCurrentRole +
##     YearsSinceLastPromotion + DistanceFromHome + NumCompaniesWorked +
##     Age + WorkLifeBalance + RelationshipSatisfaction + TrainingTimesLastYear +
```

```
##     YearsWithCurrManager + Gender + EducationField + TotalWorkingYears +
##     YearsAtCompany + StockOptionLevel
```

However, variables may end up in the final formula due to chance. To mitigate this possibility, bootstrapping approach can be chosen. We make 1000 bootstrap samples and do the procedure for each one of them. The idea is that important variables will end up in all such samples or the majority and irrelevant variables will not.

```
give_formula_boot <- function(initial_data, formula_for_scope){

  boot_sample <- initial_data[dqrng::dqsample.int(nrow(initial_data), nrow(initial_data), replace = T),]
  model_n <-        glm(Attrition ~ 1, data = boot_sample, family = 'binomial')

  model_pruned_step_forw   <- stats::step(
    model_n,
    scope = formula_for_scope,
    method = 'forward',
    trace = 0)
  return(model_pruned_step_forw %>% formula() %>% as.character() %>% .[3])
}

#plan(multisession)
#vector_of_formulas <- future_replicate(1000, give_formula_boot(IBM_HR_data_proc %>% mutate(Attrition =

vector_of_formulas <- readRDS("vector_of_formulas.rds")
```

```
counts_features <- lapply(vector_of_formulas, function(x) strsplit(x, '\\+')) %>%
    unlist() %>%
    trimws() %>%
    table() %>%
    sort()

counts_features %>%
  as.data.frame() %>%
  rename(variable = 1) %>%
  ggplot(aes(x = variable, y = Freq)) +
  geom_col() +
  coord_flip()
```



We see exactly what we expect - some variable end up in every or almost every bootstrap sample, while others only in the minority of them. A cutoff of 250 can be chosen. It is also clear that most of the variables concur with those identified by the statistical tests.

```
valid_variables <- counts_features[counts_features > 250] %>% names()
```

```
setDF(IBM_HR_data_proc)
IBM_HR_data_proc_selected <- IBM_HR_data_proc[, c(valid_variables, "Attrition")] %>%
  mutate_if(is.character, as.factor)
```

From here we will use tidymodels framework to specify, train and evaluate ML models. Initially we split the data to the training and testing datasets

```
split <- rsample::initial_split(IBM_HR_data_proc_selected, prop = .7, strata = Attrition)
train <- rsample::training(split)
test <- rsample::testing(split)
```

Next, we choose the method of validation. repeated (2 times) 10-fold cross validation method was chosen as a method with extremely reliable results. After that we specify the recipe - sequence of preprocessing steps. Since xgboost does not work with categorical variables as is, we perform one-hot-encoding on them. We also standartize (transforming to the z-scores) the variables that span several orders of magnitude - step_normalize. It is clear that we are facing the problem of class imbalance in our data => the models are going to be skewed towards predicting the majority class. To combat this, several techniques are available. We chose downsampling and SMOTE methods.

```r
folds <- vfold_cv(train, repeats = 2)

rec_xbg <- recipe(Attrition ~ . , data = train) %>%
  step_dummy(OverTime, BusinessTravel, JobRole,MaritalStatus,EducationField, Gender, one_hot = TRUE) %>%
  step_downsample(Attrition) %>% #step_smote was also tried
  step_normalize(DistanceFromHome, DailyRate, Age) %>%
  prep()

#juice(rec_1)
```

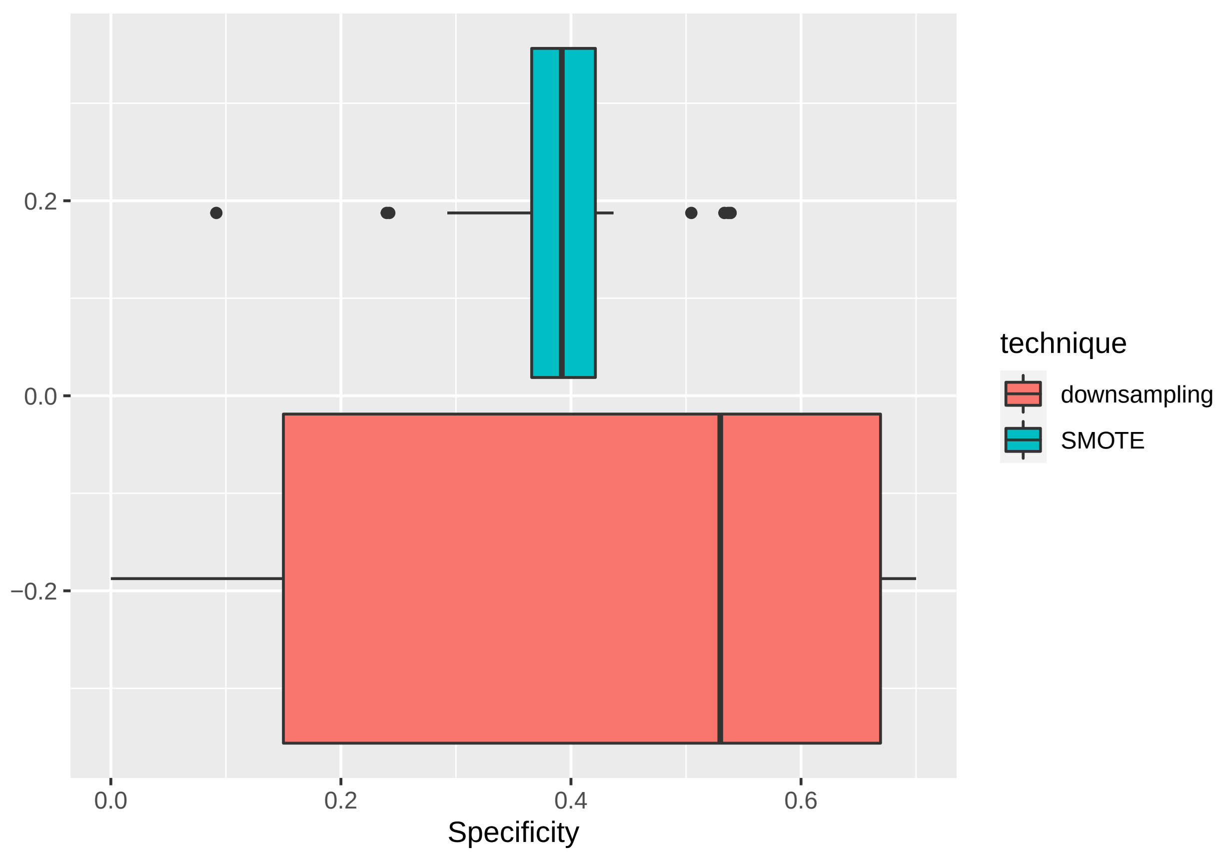
Next, we specify the hyperparameters that are going to be tuned and create the grid with 20 combinations and search for the most optimal one. The metric that is the most important in our case is specificity (NOT sensitivity because "No" is viewed as a positive class!). In other words, we are mostly interested in predicting "Yes" for Attrition. However, if this assumption of the author is not accurate, metrics "accuracy" and "sensitivity" are also present in the final tables.

```r
# xgb_spec <- boost_tree(
#       trees = 1000,
#       tree_depth = tune(),
#       min_n = tune(),
#       loss_reduction = tune(),
#       sample_size = tune(),
#       mtry = tune(),
#       learn_rate = tune()
#   ) %>%
#   set_engine('xgboost')%>%
#   set_mode('classification')
#
# xgb_grid <- grid_latin_hypercube(
#   tree_depth(),
#   min_n(),
#   loss_reduction(),
#   sample_size = sample_prop(),
#   finalize(mtry(),train),
#   learn_rate(),
#   size = 20
# )
#
# xgb_wf <- workflow() %>% add_recipe(rec_xbg) %>% add_model(xgb_spec)
#
# grid_s <- tune_grid(
#   xgb_wf,
#   resamples = folds,
#   grid = xgb_grid,
#   control = control_grid(save_pred = F, verbose= T),
#   metrics = metric_set(accuracy, spec, sens)
# )
#
# collect_metrics(grid_s) %>% dplyr::filter(.metric == 'spec') %>% arrange(desc(mean))
```

```r
downsampled_xgb <- readRDS("downsampled_xgb.rds")
smote_xgb <- readRDS("smote_xgb.rds")
downsampled_xgb %>% collect_metrics()
```

```
## # A tibble: 60 x 12
##     mtry min_n tree_depth learn_rate loss_~1 sampl~2 .metric .esti~3  mean     n
##    <int> <int>      <int>      <dbl>   <dbl>   <dbl> <chr>   <chr>   <dbl> <int>
##  1    19     2          1   2.33e- 8 1.15e-4   0.860 accura~ binary  0.709    20
##  2    19     2          1   2.33e- 8 1.15e-4   0.860 sens    binary  0.738    20
##  3    19     2          1   2.33e- 8 1.15e-4   0.860 spec    binary  0.560    20
##  4     6    33         13   2.42e-10 3.22e-6   0.879 accura~ binary  0.840    20
##  5     6    33         13   2.42e-10 3.22e-6   0.879 sens    binary  1        20
##  6     6    33         13   2.42e-10 3.22e-6   0.879 spec    binary  0        20
##  7     4    37         12   3.31e- 3 1.22e-9   0.617 accura~ binary  0.367    20
##  8     4    37         12   3.31e- 3 1.22e-9   0.617 sens    binary  0.3      20
##  9     4    37         12   3.31e- 3 1.22e-9   0.617 spec    binary  0.7      20
## 10    22    21          7   1.29e- 7 1.49e+0   0.730 accura~ binary  0.667    20
## # ... with 50 more rows, 2 more variables: std_err <dbl>, .config <chr>, and
## #   abbreviated variable names 1: loss_reduction, 2: sample_size, 3: .estimator
```

```r
downsampled_xgb %>% collect_metrics() %>%
  mutate(technique = "downsampling") %>%
  bind_rows(smote_xgb %>% collect_metrics() %>% mutate(technique = "SMOTE")) %>%
  dplyr::filter(.metric == 'spec') %>%
  ggplot(aes(x = mean, fill = technique)) +
  geom_boxplot() +
  xlab("Specificity")
```



Using the boxplots, we compare the performance of two class balancing techniques. It is clear that downsampling is superior in our particular case. Next, we use the model with the best hyperparameters and evaluate it on the testing data

```
classif_res <- readRDS("best_xgb_downsampled.rds")
#classif_res <- readRDS("best_xgb_smote.rds")

#best_params <- try(grid_s %>% tune::select_best(metric = 'spec'))

#classif_res <-
#           xgb_wf %>%
#           tune::finalize_workflow(best_params) %>%
#           parsnip::fit(data = train)

conf_m <- confusionMatrix(predict(classif_res, new_data = test) %>% unlist(), test$Attrition)
conf_m
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  278  20
##        Yes  92  52
##
##                Accuracy : 0.7466
##                  95% CI : (0.7034, 0.7865)
##     No Information Rate : 0.8371
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3376
##
##  Mcnemar's Test P-Value : 1.961e-11
##
##             Sensitivity : 0.7514
##             Specificity : 0.7222
##          Pos Pred Value : 0.9329
##          Neg Pred Value : 0.3611
##              Prevalence : 0.8371
##          Detection Rate : 0.6290
##    Detection Prevalence : 0.6742
##       Balanced Accuracy : 0.7368
##
##        'Positive' Class : No
##
```

Despite the model having lower accuracy than naive classifier (always classifies as No), the specificity is relatively high. Next, we try KNN classifier with downsampling technique.

```
# rec_knn <- recipe(Attrition ~ . , data = train) %>%
#   step_downsample(Attrition) %>%
#   step_normalize(DistanceFromHome, DailyRate, Age) %>%
#   prep()
#
# knn_spec <- nearest_neighbor(
#     neighbors = tune()
```

```r
#   ) %>%
#   set_engine('kknn')%>%
#   set_mode('classification')
#
# knn_grid <- expand.grid(neighbors = 2:10)
#
# knn_wf <- workflow() %>% add_recipe(rec_knn) %>% add_model(knn_spec)
#
# grid_s <- tune_grid(
#   knn_wf,
#   resamples = folds,
#   grid = knn_grid,
#   control = control_grid(save_pred = F, verbose= T),
#   metrics = metric_set(accuracy, spec, sens)
# )
#
# collect_metrics(grid_s) %>% dplyr::filter(.metric == 'spec') %>% arrange(desc(mean))

downsample_knn <- readRDS("downsample_knn.rds")
downsample_knn %>% collect_metrics()
```

```
## # A tibble: 27 x 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         2 accuracy binary     0.634    20 0.0103  Preprocessor1_Model1
##  2         2 sens     binary     0.646    20 0.0117  Preprocessor1_Model1
##  3         2 spec     binary     0.568    20 0.0275  Preprocessor1_Model1
##  4         3 accuracy binary     0.634    20 0.0103  Preprocessor1_Model2
##  5         3 sens     binary     0.646    20 0.0117  Preprocessor1_Model2
##  6         3 spec     binary     0.568    20 0.0275  Preprocessor1_Model2
##  7         4 accuracy binary     0.634    20 0.0103  Preprocessor1_Model3
##  8         4 sens     binary     0.646    20 0.0117  Preprocessor1_Model3
##  9         4 spec     binary     0.568    20 0.0275  Preprocessor1_Model3
## 10         5 accuracy binary     0.678    20 0.00968 Preprocessor1_Model4
## # ... with 17 more rows
```

```r
classif_res <- readRDS("best_knn_downsampled.rds")

#best_params <- try(grid_s %>% tune::select_best(metric = 'spec'))

#classif_res <-
#           knn_wf %>%
#           tune::finalize_workflow(best_params) %>%
#           parsnip::fit(data = train)

conf_m <- confusionMatrix(predict(classif_res, new_data = test) %>% unlist(), test$Attrition)
conf_m
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
```

```
##        No  271  15
##        Yes  99  57
##
##                Accuracy : 0.7421
##                  95% CI : (0.6986, 0.7823)
##     No Information Rate : 0.8371
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3566
##
##  Mcnemar's Test P-Value : 7.625e-15
##
##             Sensitivity : 0.7324
##             Specificity : 0.7917
##          Pos Pred Value : 0.9476
##          Neg Pred Value : 0.3654
##              Prevalence : 0.8371
##          Detection Rate : 0.6131
##    Detection Prevalence : 0.6471
##       Balanced Accuracy : 0.7620
##
##        'Positive' Class : No
##
```

Out next model to try is logistic regression. We will not check the assumptions of the model, since we are not interested in using the explaining power of the model, but only the predicting power. Logistic model doesn't really have hyperparameters to tune, so we simply evaluate it on the training data using repeated-10-foldCV. Since this step is not computationally intensive like the previous ones, we keep it without commenting.

```
rec_glm <- recipe(Attrition ~ . , data = train) %>%
  step_downsample(Attrition) %>%
  step_normalize(DistanceFromHome, DailyRate, Age) %>%
  prep()

glm_spec <- logistic_reg() %>%
  set_engine("glm")

glm_wf_forw <- workflow() %>%
  add_model(glm_spec) %>%
  add_recipe(rec_glm)

glm_res <- fit_resamples(
  glm_wf_forw,
  folds,
  metrics = metric_set(accuracy, sens, spec),
  control = tune::control_resamples(verbose = F,
                                    save_pred = F)
)

#glm_res %>% collect_metrics()

downsample_glm <- readRDS("downsample_glm.rds")
downsample_glm %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.722    20  0.0111 Preprocessor1_Model1
## 2 sens     binary     0.717    20  0.0121 Preprocessor1_Model1
## 3 spec     binary     0.735    20  0.0242 Preprocessor1_Model1
```

```
glm_model <- glm_wf_forw %>% parsnip::fit(., data = train)

predictions <- glm_model %>%  predict(test, type = "class")

conf_m <- confusionMatrix(predictions %>% unlist(), test$Attrition)
conf_m
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  280  18
##        Yes  90  54
##
##                Accuracy : 0.7557
##                  95% CI : (0.7128, 0.795)
##     No Information Rate : 0.8371
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3613
##
##  Mcnemar's Test P-Value : 8.375e-12
##
##             Sensitivity : 0.7568
##             Specificity : 0.7500
##          Pos Pred Value : 0.9396
##          Neg Pred Value : 0.3750
##              Prevalence : 0.8371
##          Detection Rate : 0.6335
##    Detection Prevalence : 0.6742
##       Balanced Accuracy : 0.7534
##
##        'Positive' Class : No
##
```

It has observed by the author that the final result for the specificity varies quite significantly from split to split. It is because every single correct or incorrect classification represents about 1.3% of the total score. That is why it was decided to perform monte-carlo cross validation for the 3 models with the best hyperparameters.

```
MC_cross_validation <- function(data_df, wf, best_params = NULL, B = 200){

     fit_and_test <- function(data_df, wf, best_params){
      split <- rsample::initial_split(data_df, prop = .7, strata = Attrition)
      train <- rsample::training(split)
      test <- rsample::testing(split)
```

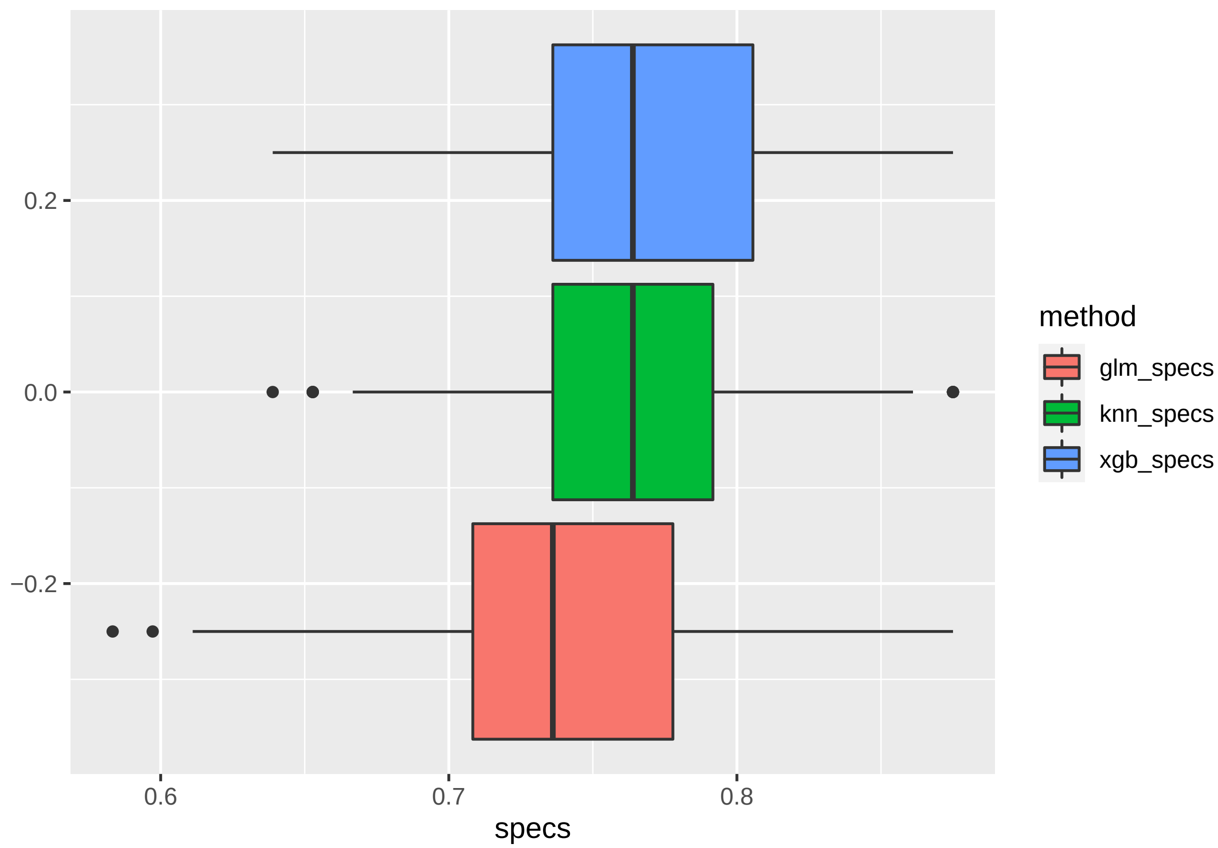
```r
      if(is.null(best_params)){
        model <- wf %>% parsnip::fit(., data = train)
        predictions <- model %>%  predict(test, type = "class")
        conf_m <- confusionMatrix(predictions %>% unlist(), test$Attrition)
      }else{
        model <-    wf %>%  tune::finalize_workflow(best_params) %>%    parsnip::fit(data = train)
        conf_m <- confusionMatrix(predict(classif_res, new_data = test) %>% unlist(), test$Attrition)
      }

      return(conf_m$byClass["Specificity"])
     }


  MC_test_errors <-pbreplicate(B,fit_and_test(data_df, wf, best_params))

}
```

```r
#  rec_glm <- recipe(Attrition ~ . , data = IBM_HR_data_proc_selected) %>%
#    step_downsample(Attrition) %>%
#    step_normalize(DistanceFromHome, DailyRate, Age) %>%
#    prep()
#
#  glm_spec <- logistic_reg() %>%
#    set_engine("glm")
#
#  glm_wf <- workflow() %>%
#    add_model(glm_spec) %>%
#    add_recipe(rec_glm)
#
# ###
# rec_knn <- recipe(Attrition ~ . , data = IBM_HR_data_proc_selected) %>%
#   step_downsample(Attrition) %>%
#   step_normalize(DistanceFromHome, DailyRate, Age) %>%
#   prep()
#
# knn_spec <- nearest_neighbor(
#     neighbors = tune()
#   ) %>%
#   set_engine('kknn')%>%
#   set_mode('classification')
#
# knn_wf <- workflow() %>% add_recipe(rec_knn) %>% add_model(knn_spec)
# ###
#
# rec_xbg <- recipe(Attrition ~ . , data = IBM_HR_data_proc_selected) %>%
#   step_dummy(OverTime, BusinessTravel, JobRole,MaritalStatus,EducationField, Gender, one_hot = TRUE)
#   step_downsample(Attrition) %>% #step_smote was also tried
#   step_normalize(DistanceFromHome, DailyRate, Age) %>%
#   prep()
#
# xgb_spec <- boost_tree(
#      trees = 1000,
#      tree_depth = tune(),
```

```r
#       min_n = tune(),
#       loss_reduction = tune(),
#       sample_size = tune(),
#       mtry = tune(),
#       learn_rate = tune()
#   ) %>%
#   set_engine('xgboost')%>%
#   set_mode('classification')
#
# xgb_wf <- workflow() %>% add_recipe(rec_xbg) %>% add_model(xgb_spec)
# ###
#
# best_params_knn <- downsample_knn %>% tune::select_best(metric = 'spec')
# best_params_xgb <- downsampled_xgb %>% tune::select_best(metric = 'spec')
#
# knn_specs <- MC_cross_validation(IBM_HR_data_proc_selected, knn_wf, best_params_knn, B = 300)
# glm_specs <- MC_cross_validation(IBM_HR_data_proc_selected, glm_wf, NULL, B = 300)
# xgb_specs <- MC_cross_validation(IBM_HR_data_proc_selected, xgb_wf, best_params_xgb, B = 300)

knn_specs <- readRDS("knn_specs.rds")
glm_specs <- readRDS("glm_specs.rds")
xgb_specs <- readRDS("xgb_specs.rds")
```

```r
results_final <- data.frame("knn_specs" = knn_specs, "glm_specs" = glm_specs, "xgb_specs" = xgb_specs) %
  gather(key = "method", value = "specs")
```

```r
results_final %>%
  ggplot(aes(fill = method, x = specs)) +
  geom_boxplot()
```




```
results_final %>%
  group_by(method) %>%
  summarize(mean_spec = mean(specs),
            sd_spec = sd(specs))
```

```
## # A tibble: 3 x 3
##   method    mean_spec sd_spec
##   <chr>         <dbl>   <dbl>
## 1 glm_specs     0.742  0.0534
## 2 knn_specs     0.767  0.0443
## 3 xgb_specs     0.767  0.0437
```

As an alternative approach, We could have performed dimensionality reduction using PCA. In order to do that, we need to manually perform one-hot-encoding and standartization

```
data_for_pca <- fastDummies::dummy_cols(IBM_HR_data_proc %>% dplyr::select(-Attrition), remove_selected_
data_for_pca$DailyRate <- scale(data_for_pca$DailyRate, center = T, scale = T)
data_for_pca$Age <- scale(data_for_pca$Age, center = T, scale = T)
data_for_pca$DistanceFromHome <- scale(data_for_pca$DistanceFromHome, center = T, scale = T)
data_for_pca$MonthlyRate <- scale(data_for_pca$MonthlyRate, center = T, scale = T)
data_for_pca$MonthlyIncome <- scale(data_for_pca$MonthlyIncome, center = T, scale = T)
data_for_pca$HourlyRate <- scale(data_for_pca$HourlyRate, center = T, scale = T)
data_for_pca$PercentSalaryHike <- scale(data_for_pca$PercentSalaryHike, center = T, scale = T)

pca <- FactoMineR::PCA(data_for_pca, scale.unit = F, graph = F, ncp = 25)
```

```
summary(pca)
```

```
##
## Call:
## FactoMineR::PCA(X = data_for_pca, scale.unit = F, ncp = 25, graph = F)
##
##
## Eigenvalues
##                        Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.6   Dim.7
## Variance              97.888  25.456   6.062   5.344   4.132   3.620   1.667
## % of var.             60.860  15.827   3.769   3.322   2.569   2.251   1.036
## Cumulative % of var.  60.860  76.687  80.457  83.779  86.348  88.599  89.635
##                        Dim.8   Dim.9  Dim.10  Dim.11  Dim.12  Dim.13  Dim.14
## Variance               1.252   1.244   1.172   1.092   1.071   1.045   1.014
## % of var.              0.778   0.773   0.728   0.679   0.666   0.650   0.630
## Cumulative % of var.  90.414  91.187  91.915  92.594  93.260  93.910  94.540
##                       Dim.15  Dim.16  Dim.17  Dim.18  Dim.19  Dim.20  Dim.21
## Variance               0.937   0.928   0.890   0.839   0.563   0.512   0.492
## % of var.              0.583   0.577   0.553   0.522   0.350   0.318   0.306
## Cumulative % of var.  95.123  95.700  96.253  96.775  97.125  97.443  97.749
##                       Dim.22  Dim.23  Dim.24  Dim.25  Dim.26  Dim.27  Dim.28
## Variance               0.476   0.460   0.391   0.356   0.340   0.310   0.184
## % of var.              0.296   0.286   0.243   0.221   0.212   0.193   0.114
## Cumulative % of var.  98.045  98.331  98.574  98.795  99.007  99.200  99.314
##                       Dim.29  Dim.30  Dim.31  Dim.32  Dim.33  Dim.34  Dim.35
## Variance               0.141   0.125   0.124   0.112   0.111   0.092   0.083
## % of var.              0.088   0.078   0.077   0.070   0.069   0.057   0.051
## Cumulative % of var.  99.402  99.480  99.557  99.626  99.695  99.752  99.803
##                       Dim.36  Dim.37  Dim.38  Dim.39  Dim.40  Dim.41  Dim.42
## Variance               0.077   0.060   0.058   0.046   0.041   0.013   0.012
## % of var.              0.048   0.037   0.036   0.029   0.025   0.008   0.007
## Cumulative % of var.  99.851  99.888  99.924  99.953  99.978  99.986  99.994
##                       Dim.43  Dim.44  Dim.45  Dim.46  Dim.47  Dim.48  Dim.49
## Variance               0.007   0.004   0.000   0.000   0.000   0.000   0.000
## % of var.              0.004   0.002   0.000   0.000   0.000   0.000   0.000
## Cumulative % of var.  99.998 100.000 100.000 100.000 100.000 100.000 100.000
##                       Dim.50  Dim.51
## Variance               0.000   0.000
## % of var.              0.000   0.000
## Cumulative % of var. 100.000 100.000
##
## Individuals (the 10 first)
##                                      Dist     Dim.1     ctr    cos2     Dim.2
## 1                               |   8.564 |  -3.045   0.006   0.126 |   0.036
## 2                               |   7.212 |   1.958   0.003   0.074 |   4.258
## 3                               |  11.663 |  -9.634   0.064   0.682 |  -4.519
## 4                               |   7.501 |  -2.156   0.003   0.083 |   2.868
## 5                               |  10.969 |  -7.771   0.042   0.502 |  -1.776
## 6                               |   6.613 |  -1.210   0.001   0.033 |   4.672
## 7                               |  10.409 |  -5.396   0.020   0.269 |  -7.051
## 8                               |  14.345 | -13.491   0.126   0.885 |   1.150
## 9                               |   7.508 |   1.759   0.002   0.055 |   4.321
## 10                              |   9.865 |   6.542   0.030   0.440 |  -1.610
```

```
##                                        ctr    cos2     Dim.3     ctr    cos2
## 1                                    0.000   0.000 |   0.540   0.003   0.004 |
## 2                                    0.048   0.349 |  -2.464   0.068   0.117 |
## 3                                    0.055   0.150 |   1.484   0.025   0.016 |
## 4                                    0.022   0.146 |   0.861   0.008   0.013 |
## 5                                    0.008   0.026 |   3.977   0.177   0.131 |
## 6                                    0.058   0.499 |  -0.333   0.001   0.003 |
## 7                                    0.133   0.459 |   0.239   0.001   0.001 |
## 8                                    0.004   0.006 |  -0.388   0.002   0.001 |
## 9                                    0.050   0.331 |  -2.907   0.095   0.150 |
## 10                                   0.007   0.027 |   4.878   0.267   0.245 |
##
## Variables (the 10 first)
##                                       Dim.1    ctr   cos2    Dim.2    ctr   cos2
## Age                                 |  0.556  0.315  0.309 | -0.439  0.756  0.193
## DailyRate                           | -0.007  0.000  0.000 | -0.050  0.010  0.002
## DistanceFromHome                    |  0.010  0.000  0.000 |  0.014  0.001  0.000
## Education                           |  0.130  0.017  0.016 | -0.097  0.037  0.009
## EnvironmentSatisfaction             |  0.002  0.000  0.000 |  0.008  0.000  0.000
## HourlyRate                          | -0.014  0.000  0.000 | -0.027  0.003  0.001
## JobInvolvement                      | -0.006  0.000  0.000 | -0.003  0.000  0.000
## JobLevel                            |  0.814  0.676  0.541 | -0.324  0.412  0.086
## JobSatisfaction                     | -0.019  0.000  0.000 |  0.018  0.001  0.000
## MonthlyIncome                       |  0.717  0.525  0.514 | -0.310  0.379  0.096
##                                       Dim.3    ctr   cos2
## Age                                 |  0.056  0.052  0.003 |
## DailyRate                           | -0.002  0.000  0.000 |
## DistanceFromHome                    | -0.009  0.001  0.000 |
## Education                           |  0.050  0.041  0.002 |
## EnvironmentSatisfaction             |  0.036  0.021  0.001 |
## HourlyRate                          | -0.009  0.001  0.000 |
## JobInvolvement                      | -0.010  0.002  0.000 |
## JobLevel                            | -0.019  0.006  0.000 |
## JobSatisfaction                     | -0.044  0.033  0.002 |
## MonthlyIncome                       | -0.008  0.001  0.000 |
```

We see that first 20 principal components explain more than 99% of the variation in the data. we will keep only them (insted of original 41) and train+evaluate GLM model.

```r
data_df <- cbind(pca$ind$coord) %>% as.data.frame() %>% mutate(Attrition = IBM_HR_data_proc$Attrition %

split <- rsample::initial_split(data_df, prop = .7, strata = Attrition)
train <- rsample::training(split)
test <- rsample::testing(split)

folds <- vfold_cv(train, repeats = 2)

rec_glm <- recipe(Attrition ~ . , data = train) %>%
  step_downsample(Attrition) %>%
  prep()

glm_spec <- logistic_reg() %>%
  set_engine("glm")
```

```r
glm_wf_forw <- workflow() %>%
  add_model(glm_spec) %>%
  add_recipe(rec_glm)

glm_res <- fit_resamples(
  glm_wf_forw,
  folds,
  metrics = metric_set(accuracy, sens, spec),
  control = tune::control_resamples(verbose = F,
                                    save_pred = F)
)

glm_res %>% collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.720    20  0.0118 Preprocessor1_Model1
## 2 sens     binary     0.712    20  0.0137 Preprocessor1_Model1
## 3 spec     binary     0.755    20  0.0210 Preprocessor1_Model1
```

```r
glm_model <- glm_wf_forw %>% parsnip::fit(., data = train)

predictions <- glm_model %>%  predict(test, type = "class")

conf_m <- confusionMatrix(predictions %>% unlist(), as.factor(test$Attrition))
conf_m
```

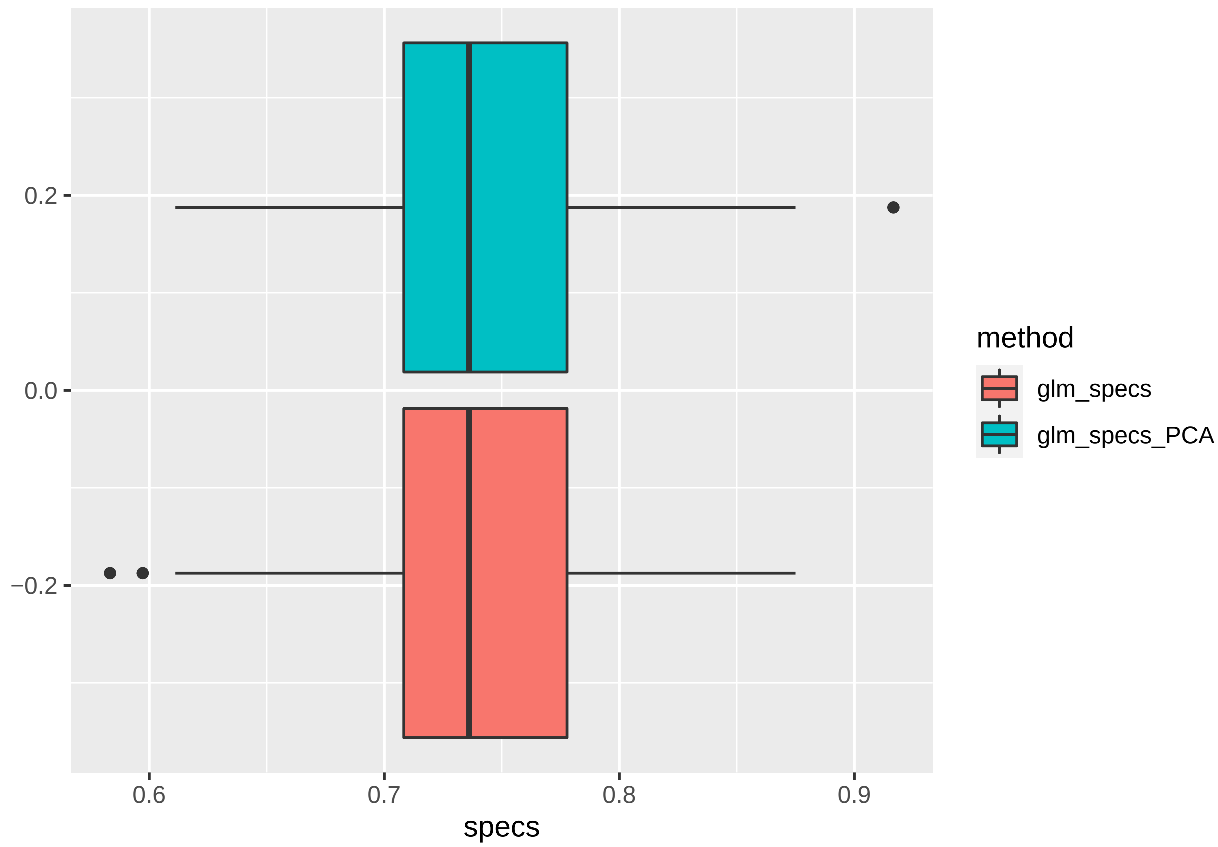
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  285  20
##        Yes  85  52
##
##                Accuracy : 0.7624
##                  95% CI : (0.72, 0.8014)
##     No Information Rate : 0.8371
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3612
##
##  Mcnemar's Test P-Value : 4.217e-10
##
##             Sensitivity : 0.7703
##             Specificity : 0.7222
##          Pos Pred Value : 0.9344
##          Neg Pred Value : 0.3796
##              Prevalence : 0.8371
##          Detection Rate : 0.6448
##    Detection Prevalence : 0.6900
##       Balanced Accuracy : 0.7462
```

```
##
##        'Positive' Class : No
##
```

```r
#glm_specs_PCA <- MC_cross_validation(data_df, glm_wf_forw, NULL, B = 300)
glm_specs_PCA <- readRDS("glm_specs_PCA.rds")
```

It is evident from the results on training and testing data that dimensionality reduction via PC didn't help to improve results but didn't make them worse either.

```r
data.frame("glm_specs_PCA" = glm_specs_PCA, "glm_specs" = glm_specs) %>%
  gather(key = "method", value = "specs") %>%
  ggplot(aes(fill = method, x = specs)) +
  geom_boxplot()
```



To conclude, the following steps were taken:

1. Most important variables were identified via statistical tests
2. Exploratory data analysis was performed
3. Important variables were confirmed via using step function and bootstapping
4. GLM, KNN and Xgboost classifiers were trained and evaluated.
5. KNN and Xgboost show very similar performance
6. Dimensionality reduction via PCA did not help to improve results but did not make them worse either