final_project_HarvardX

Nancy Chalhoub

1/7/2021

Summary

This is the report relative to "choose your own project" in the Capstone course of HarvardX's Data Science Professional Certificate program.

Introduction

In this project, we will use the "Adult census Income" dataset which was extracted from 1994 Census Bureau database. Our objective is to predict, using the different available information like gender, education level, native country ... (a full list of parameters is detailed in the report), if a certain person's annual salary exceeds 50k.

Methods and Analysis

In this section, we will describe how the dataset was prepared for the analysis. We start by downloading and preparing the dataset that we will use.

Data preparation

```
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
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
```

```
library(stringr)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

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

```
suppressMessages(library(caret))

incomes_ds <- read.csv('adult.csv', stringsAsFactors = F, na.strings=c("","NA"," ","?"))
```

First, we can see that `incomes_ds` dataset has 32561 rows and 15 columns.

```
nrow(incomes_ds)
```

```
## [1] 32561
```

```
ncol(incomes_ds)
```

```
## [1] 15
```

We can take a look at the first 6 entries of the dataset:

```
head(incomes_ds)
```

```
##   age workclass fnlwgt    education education.num marital.status
## 1  90      <NA>  77053      HS-grad             9        Widowed
## 2  82   Private 132870      HS-grad             9        Widowed
## 3  66      <NA> 186061 Some-college            10        Widowed
## 4  54   Private 140359      7th-8th             4       Divorced
## 5  41   Private 264663 Some-college            10      Separated
## 6  34   Private 216864      HS-grad             9       Divorced
##          occupation  relationship  race    sex capital.gain capital.loss
## 1              <NA> Not-in-family White Female            0         4356
## 2   Exec-managerial Not-in-family White Female            0         4356
## 3              <NA>     Unmarried Black Female            0         4356
```

```
## 4 Machine-op-inspct     Unmarried White Female            0         3900
## 5    Prof-specialty     Own-child White Female            0         3900
## 6     Other-service     Unmarried White Female            0         3770
##   hours.per.week native.country income
## 1             40  United-States  <=50K
## 2             18  United-States  <=50K
## 3             40  United-States  <=50K
## 4             40  United-States  <=50K
## 5             40  United-States  <=50K
## 6             45  United-States  <=50K
```

We can see that this is a tidy dataset with one observation per row.

The `incomes_ds` dataset has the following columns:

```
colnames(incomes_ds)
```

```
##  [1] "age"            "workclass"      "fnlwgt"         "education"
##  [5] "education.num"  "marital.status" "occupation"     "relationship"
##  [9] "race"           "sex"            "capital.gain"   "capital.loss"
## [13] "hours.per.week" "native.country" "income"
```

We will now check the structure of `incomes_ds`

```
str(incomes_ds)
```

```
## 'data.frame':    32561 obs. of  15 variables:
##  $ age           : int  90 82 66 54 41 34 38 74 68 41 ...
##  $ workclass     : chr  NA "Private" NA "Private" ...
##  $ fnlwgt        : int  77053 132870 186061 140359 264663 216864 150601 88638 422013 70037 ...
##  $ education     : chr  "HS-grad" "HS-grad" "Some-college" "7th-8th" ...
##  $ education.num : int  9 9 10 4 10 9 6 16 9 10 ...
##  $ marital.status: chr  "Widowed" "Widowed" "Widowed" "Divorced" ...
##  $ occupation    : chr  NA "Exec-managerial" NA "Machine-op-inspct" ...
##  $ relationship  : chr  "Not-in-family" "Not-in-family" "Unmarried" "Unmarried" ...
##  $ race          : chr  "White" "White" "Black" "White" ...
##  $ sex           : chr  "Female" "Female" "Female" "Female" ...
##  $ capital.gain  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ capital.loss  : int  4356 4356 4356 3900 3900 3770 3770 3683 3683 3004 ...
##  $ hours.per.week: int  40 18 40 40 40 45 40 20 40 60 ...
##  $ native.country: chr  "United-States" "United-States" "United-States" "United-States" ...
##  $ income        : chr  "<=50K" "<=50K" "<=50K" "<=50K" ...
```

We can notice that a lot of columns with categorical factors are there, however many of these columns have too many factors than required. In this data cleaning section we'll try to reduce the number of factors by cleaning the columns. We start with the `workclass` column. Using the table() function, we check the frequency of each factor

```
table(incomes_ds$workclass)
```

```
##
##      Federal-gov        Local-gov     Never-worked          Private
##              960             2093                7            22696
##     Self-emp-inc Self-emp-not-inc        State-gov      Without-pay
##             1116             2541             1298               14
```

We can combine "Never-worked" and "Without-pay" into a single group called "Unemployed", "Self-emp-inc" and "Self-emp-not-inc" into a single group called "Self-emp", and "Federal-gov", "State-gov" and "Local-gov" into a group called "Gov":

```
incomes_ds$workclass <- ifelse(incomes_ds$workclass %in% c("Never-worked","Without-pay"),"Unemployed",in
incomes_ds$workclass <- ifelse(incomes_ds$workclass %in% c("Local-gov","Federal-gov","State-gov"),"Gov"
incomes_ds$workclass <- ifelse(incomes_ds$workclass %in% c("Self-emp-inc","Self-emp-not-inc"),"Self-emp

table(incomes_ds$workclass)
```

```
##
##        Gov    Private   Self-emp Unemployed
##       4351      22696       3657         21
```

Next, we will do the same analysis on the marital status column into 3 groups: "Married", "Not-Married" and "Never-married".

```
incomes_ds$marital.status <- ifelse (incomes_ds$marital.status %in% c("Separated","Divorced","Widowed")
incomes_ds$marital.status <- ifelse (incomes_ds$marital.status %in% c("Married-AF-spouse","Married-civ-s
table(incomes_ds$marital.status)
```

```
##
##       Married Never-married   Not-Married
##         15417         10683          6461
```

We can also group the countries into continents as follows:

```
Asia <- c("China","Hong","India","Iran","Cambodia","Japan", "Laos","Philippines" ,"Vietnam" ,"Taiwan", "

N.A <- c("Canada","United-States","Puerto-Rico")

Europe <- c("England","France","Germany" ,"Greece","Holand-Netherlands","Hungary","Ireland","Italy","Pol
            ,"Yugoslavia")

S.A <- c("Columbia","Cuba","Dominican-Republic","Ecuador","El-Salvador","Guatemala","Haiti","Honduras",
                    ,"Outlying-US","Peru","Jamaica","Trinadad&Tobago")
Others <- c("South")
grp_cntry <- function(cntry){
    if (cntry %in% Asia){
        return("Asia")
    }else if (cntry %in% N.A){
        return("North America")
    }else if (cntry %in% Europe){
        return("Europe")
    }else if (cntry %in% S.A){
        return("South America")
    }else{
        return("Others")
    }
}

incomes_ds$native.country <- sapply(incomes_ds$native.country,grp_cntry)
table(incomes_ds$native.country)
```

```
##
##          Asia        Europe North America        Others South America
##           671           521         29405           677          1287
```

To simplify our study, we will convert the chr columns into factors. Moreover, since we are interested in predicting if the income is higher or lower than 50k, we will convert the last column into a factor column with 2 levels: 0 if the income is lower than 50k and 1 otherwise.

```
incomes_ds$workclass <- as.factor(incomes_ds$workclass)
incomes_ds$education <- as.factor(incomes_ds$education)
incomes_ds$education.num <- as.factor(incomes_ds$education.num)
incomes_ds$marital.status <- as.factor(incomes_ds$marital.status)
incomes_ds$occupation <- as.factor(incomes_ds$occupation)
incomes_ds$relationship <- as.factor(incomes_ds$relationship)
incomes_ds$race <- as.factor(incomes_ds$race)
incomes_ds$sex <- as.factor(incomes_ds$sex)
incomes_ds$native.country <- as.factor(incomes_ds$native.country)
incomes_ds$income <- ifelse(incomes_ds$income=='>50K',1,0)
incomes_ds$income <- as.factor(incomes_ds$income)
```

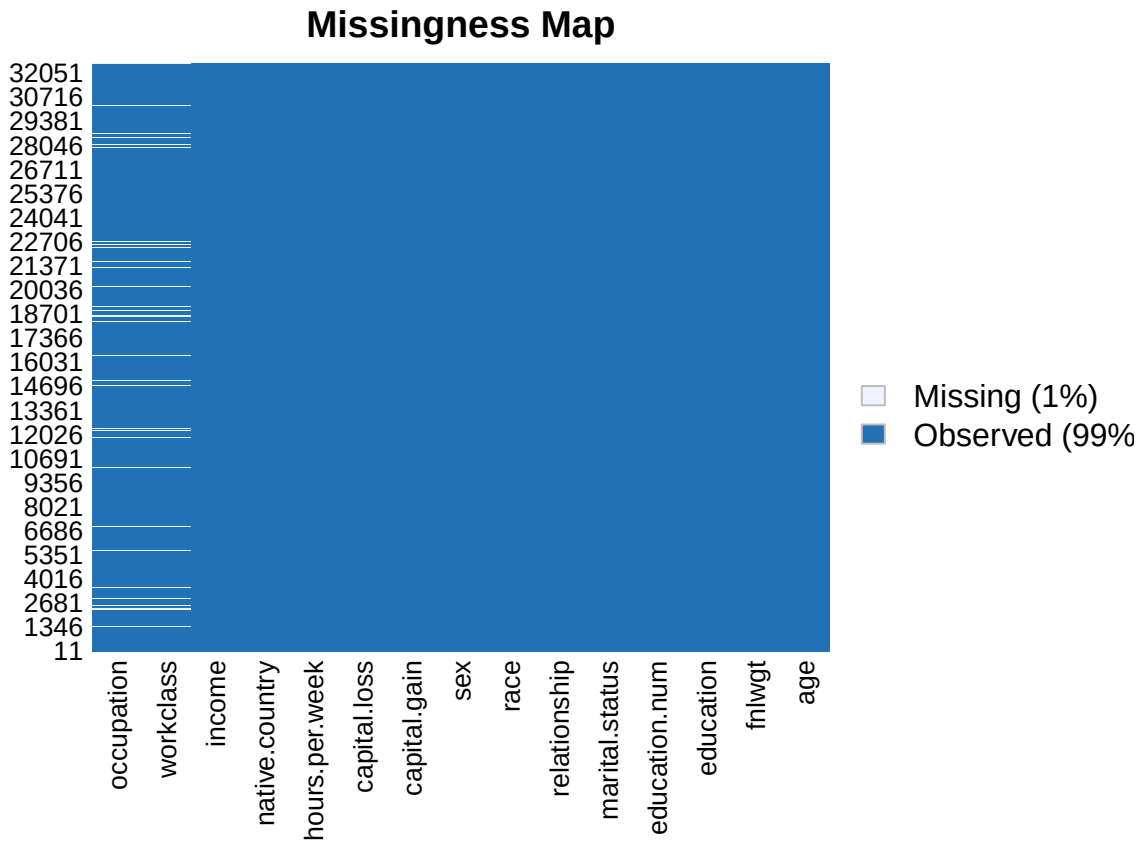
We will know check for missing data.

```
library(Amelia)
```

```
## Loading required package: Rcpp
```

```
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.6, built: 2019-11-24)
## ## Copyright (C) 2005-2021 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```
missmap(incomes_ds)
```



We can see that many observations lack some entries. For example, we can see that we have 1843 entries

where the `occupation` is unknown, and since this is essential for our predictions, we will remove those entries from our dataset.

```
sum(is.na(incomes_ds$occupation))
```

```
## [1] 1843
```

```
incomes_ds <- incomes_ds[!is.na(incomes_ds$occupation),]
```

We can check for remaining missing values in the whole dataset as follows:

```
sapply(1:ncol(incomes_ds), function (n) {sum(is.na(incomes_ds[,n]))})
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

We still have 30718 entry with all entries.

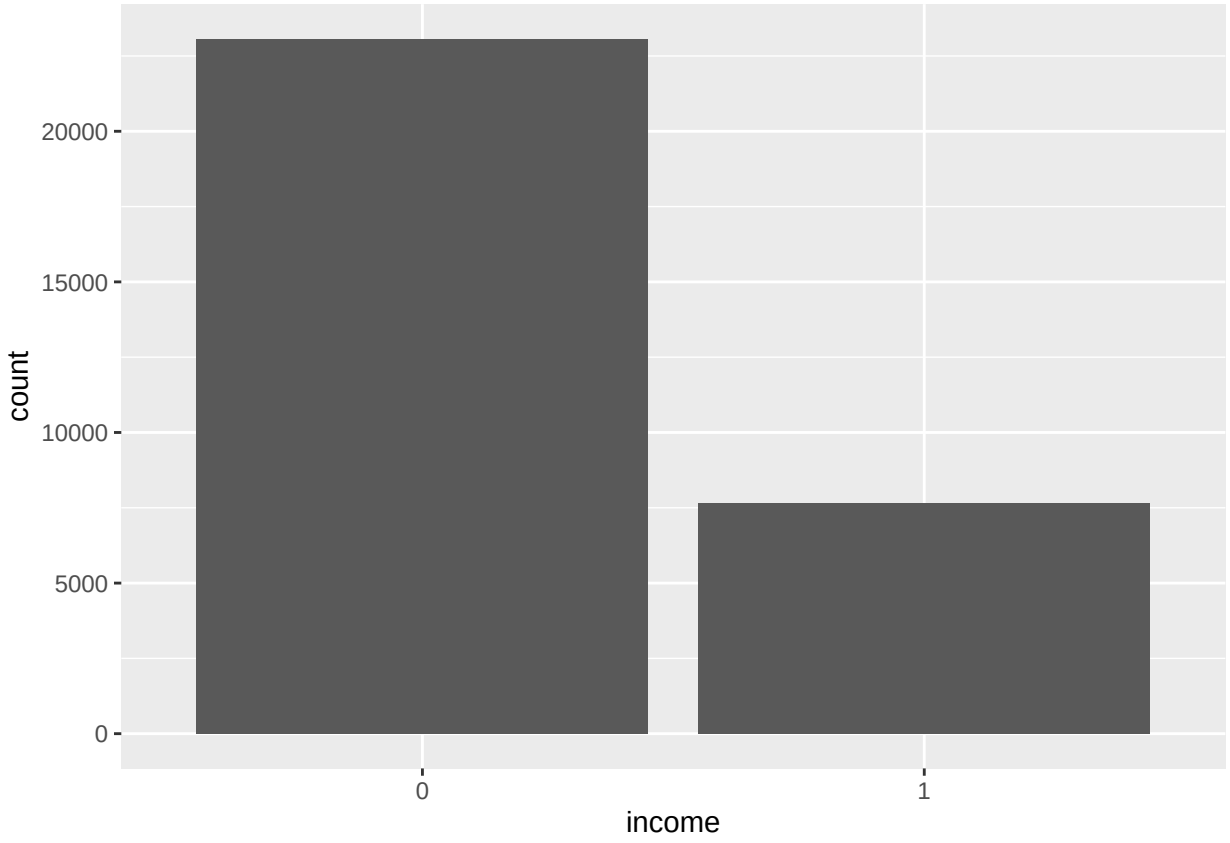
```
nrow(incomes_ds)
```

```
## [1] 30718
```

Data exploration

We will begin now to make some studies on our dataset to better understand it.

```
incomes_ds %>% ggplot(aes(x=income))+ geom_bar()
```




```
incomes_ds %>% group_by(income)%>% summarise(avg=n()/nrow(incomes_ds))
```

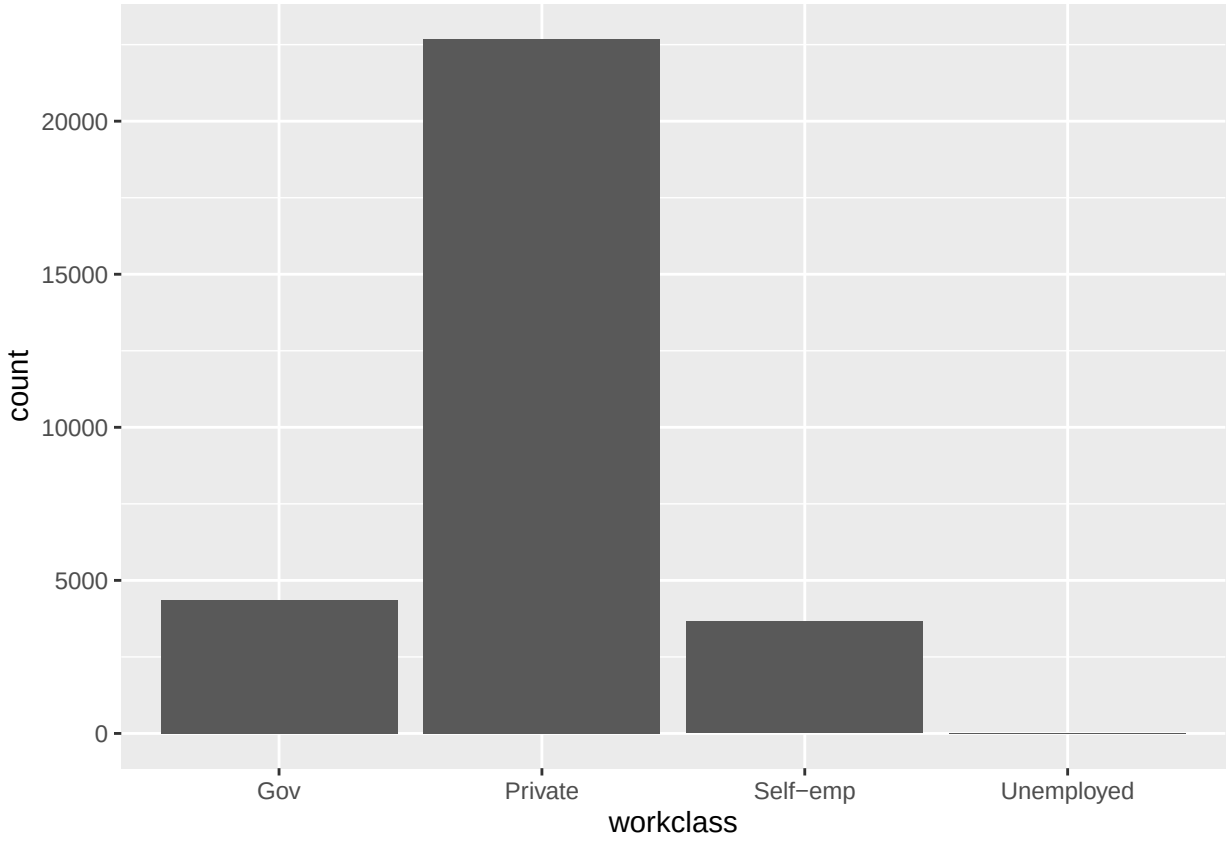
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   income   avg
##   <fct>  <dbl>
## 1 0      0.751
## 2 1      0.249
```

The entries in our dataset are divided as follows: 75% with income less than 50k and 25% with income greater than 50k.

We can now check how are the entries divided between the different work classes:

```
incomes_ds %>% ggplot(aes(x=workclass))+ geom_bar()
```



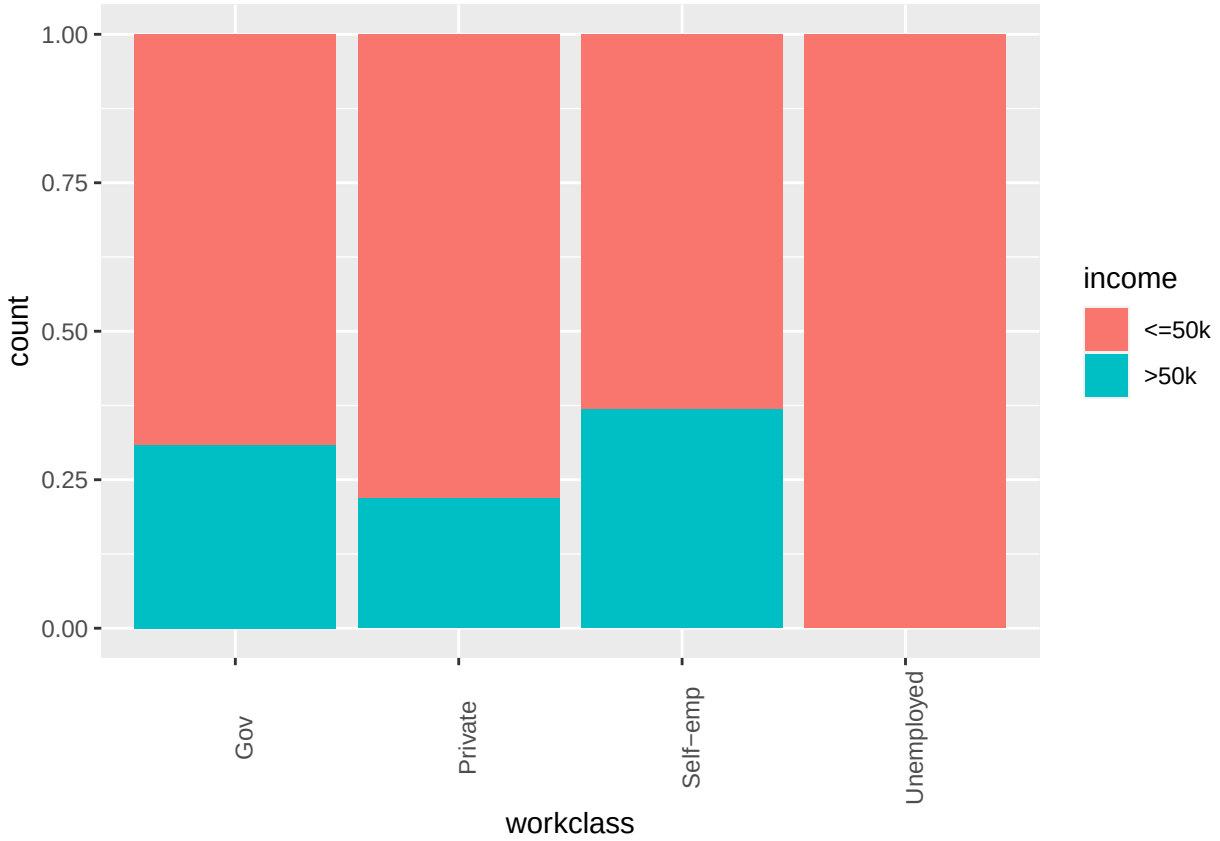

```
incomes_ds %>% group_by(workclass)%>% summarise(count=n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 4 x 2
##   workclass  count
##   <fct>      <int>
## 1 Gov         4351
## 2 Private    22696
## 3 Self-emp    3657
## 4 Unemployed    14
```

We can also see that the majority of the workers in the dataset work in the private sector. We can then see the percentage of people with higher income in each of these work classes.

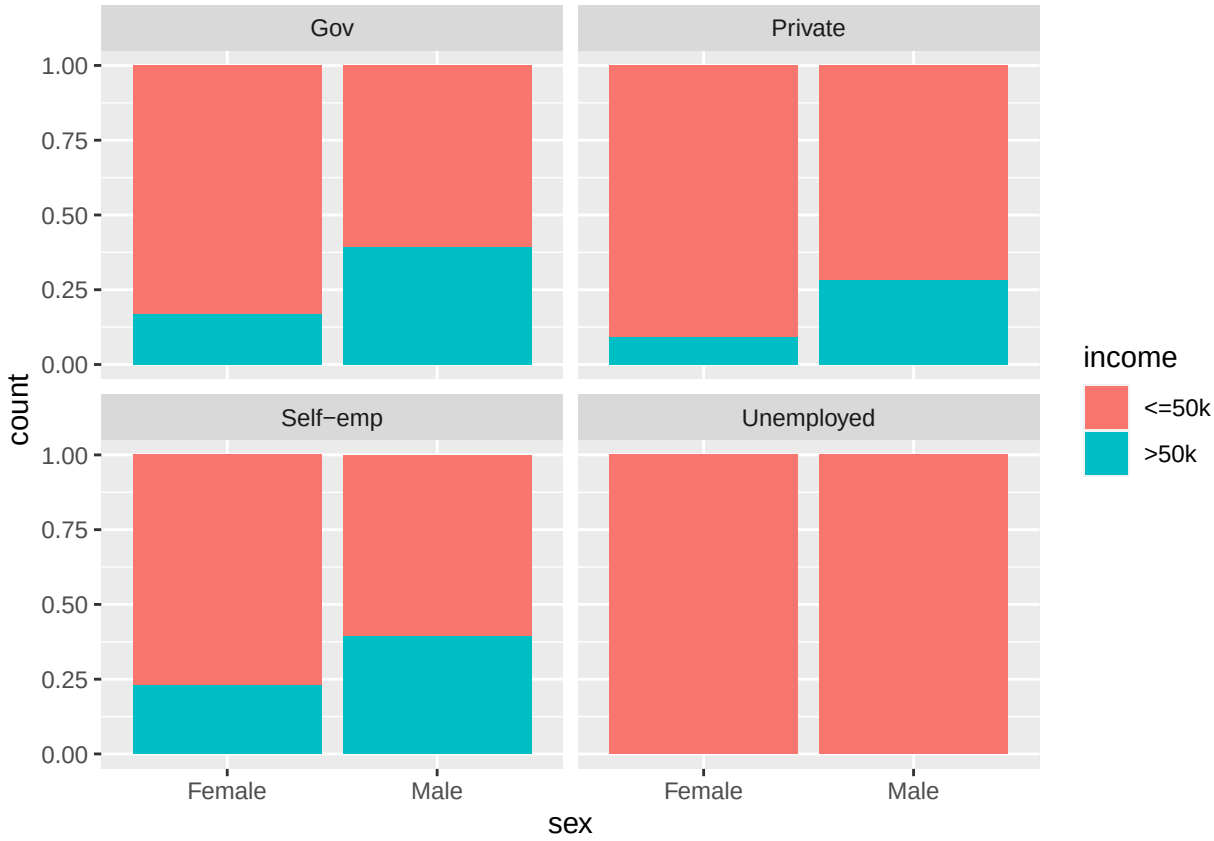
```
incomes_ds %>% ggplot(aes(x=workclass,fill=income)) + geom_bar(position = "fill")+ scale_fill_discrete(
```



We can see that the highest proportion of well paid workers is the self-employed workclass.

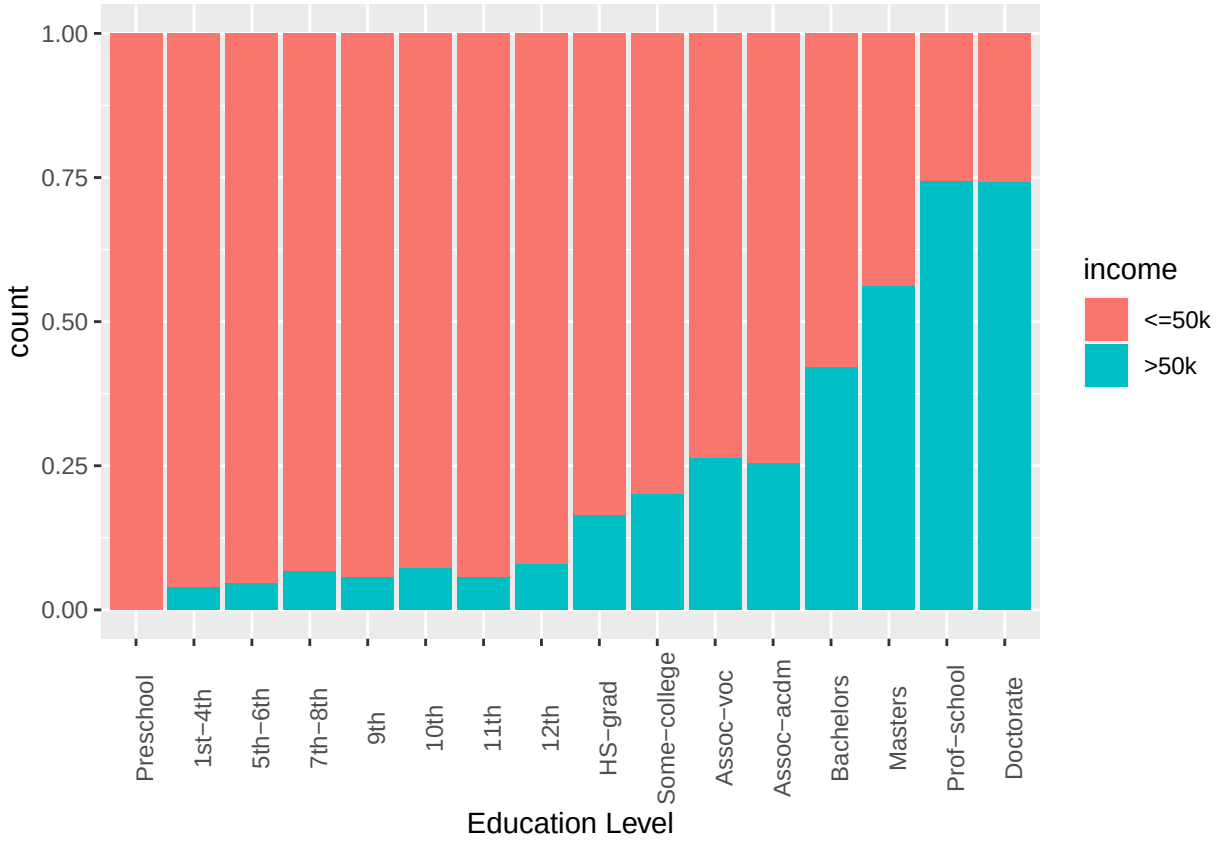
We will now compare income between different genders and work classes.

```
incomes_ds %>% ggplot(aes(x=sex,fill=income)) + geom_bar(position = "fill")+facet_wrap(incomes_ds$workcl
```



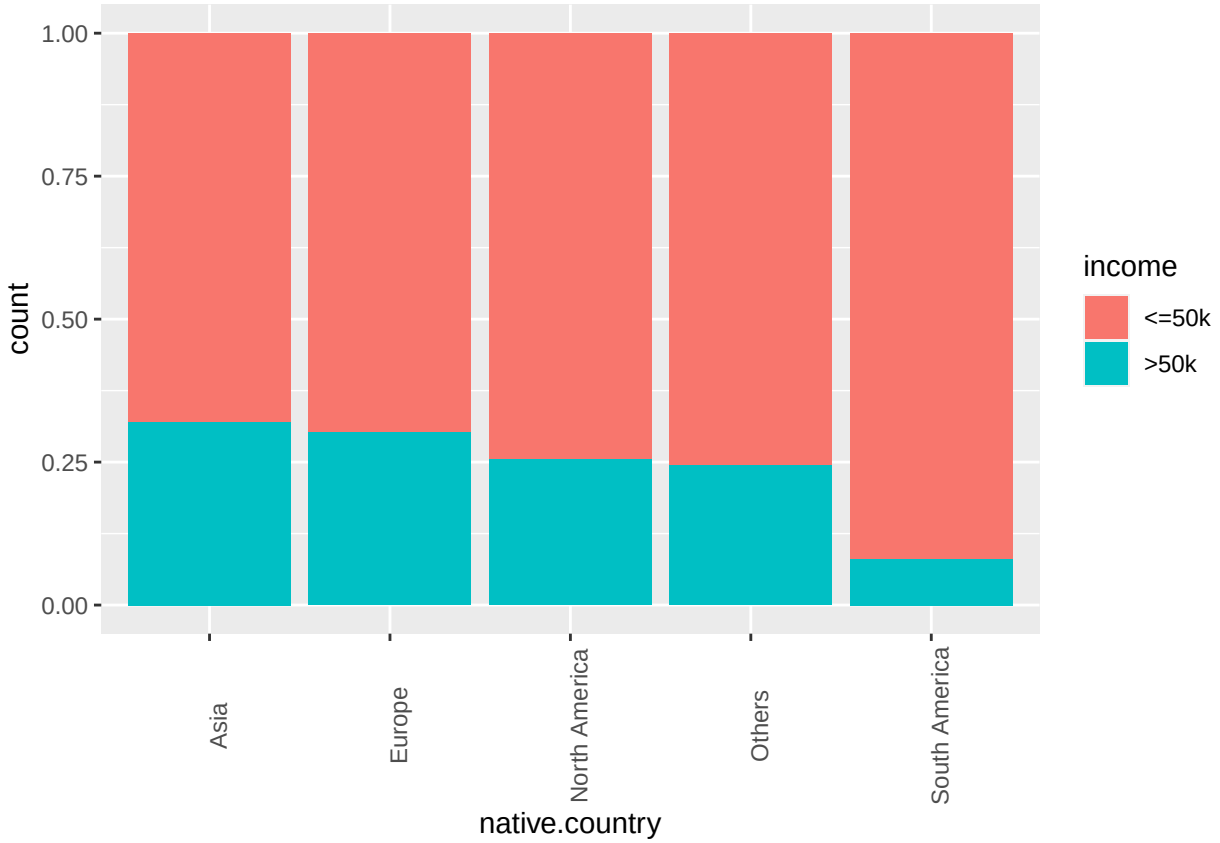
We can see that, in a particular sector, the proportion of high income within male is higher than within female. It's also interesting to compare the income between different education levels.

```
incomes_ds %>% ggplot(aes(x=education.num,fill=income)) + geom_bar(position = "fill")+ scale_fill_discr
                        labels=incomes_ds$education,name="Education Level")
```



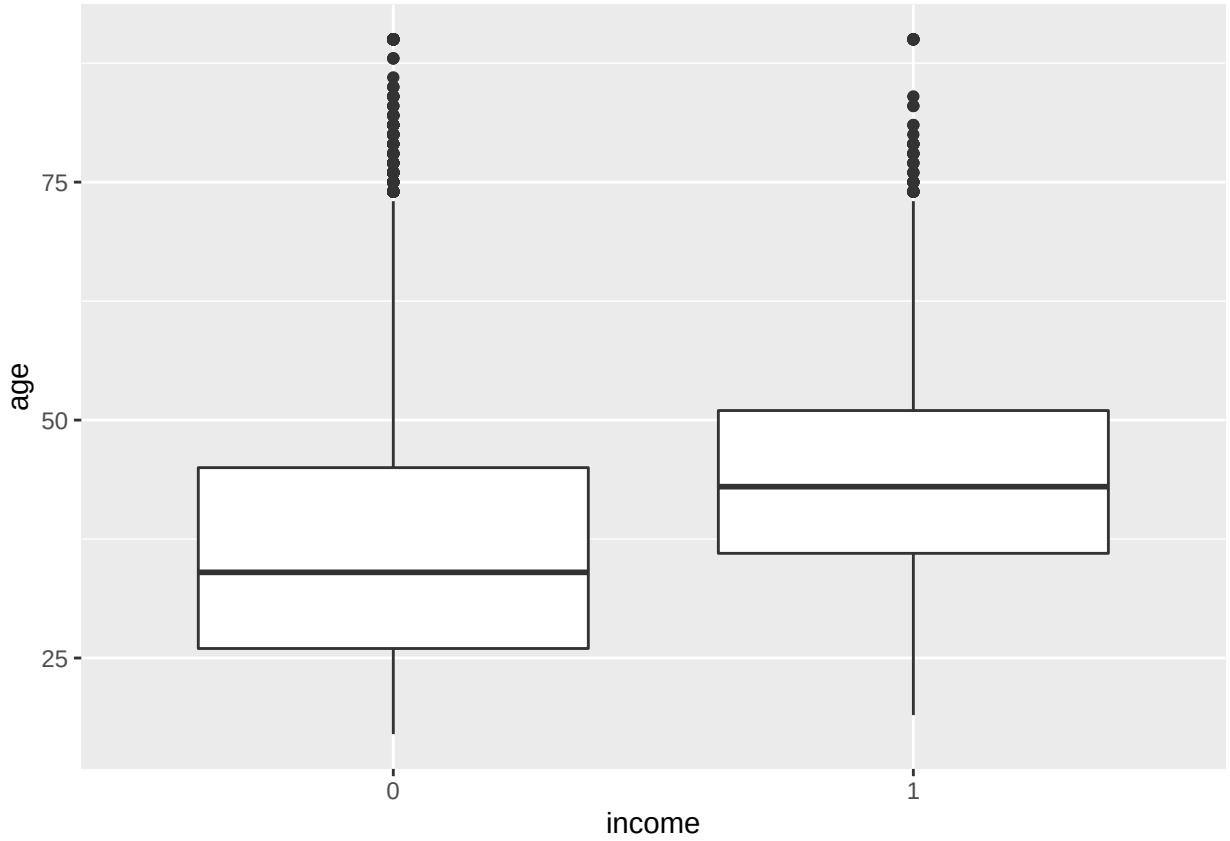
We can clearly the relation between the education level and the percentage of people with income higher than 50K: the more a person is educated, the higher her chance to earn more than 50k. We can look at the distribution with respect to the native country:

```
incomes_ds %>% ggplot(aes(x=native.country,fill=income)) + geom_bar(position = "fill")+ scale_fill_disc
```



We can see that the proportion of people with a annual salary is higher than 50k is the highest in Asia. This is result is to be taken with caution because some countries in Asia are rich while others are poor. A better approach if we are interested in those details is not to keep the country column and add another column with the region. Finally, we show the relation between the age and the income status. First, we will make a boxplot of age vs income:

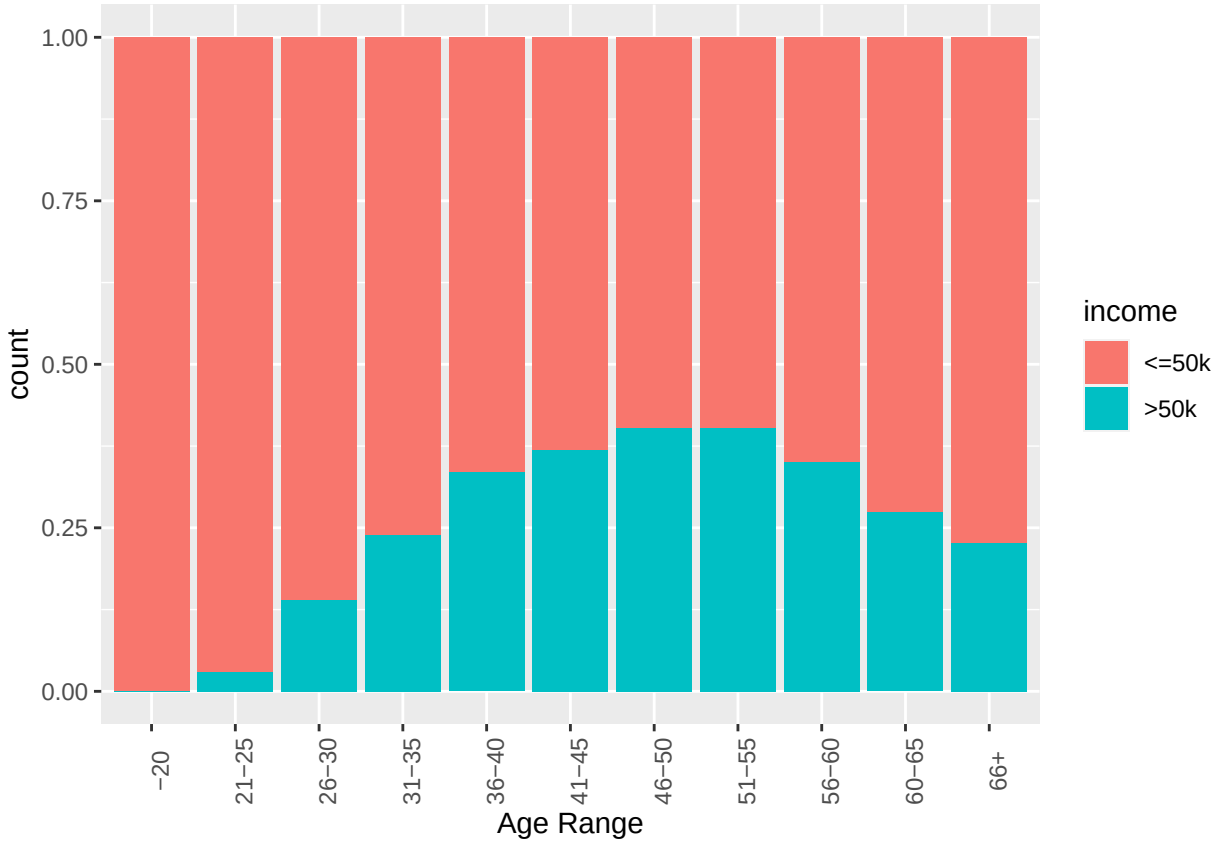
```
incomes_ds %>% ggplot(aes(x=income,y=age))+geom_boxplot()
```



We observe that the median age of people with annual salary that exceeds 50k is significantly larger than the median age of those with annual salary lower than 50k.

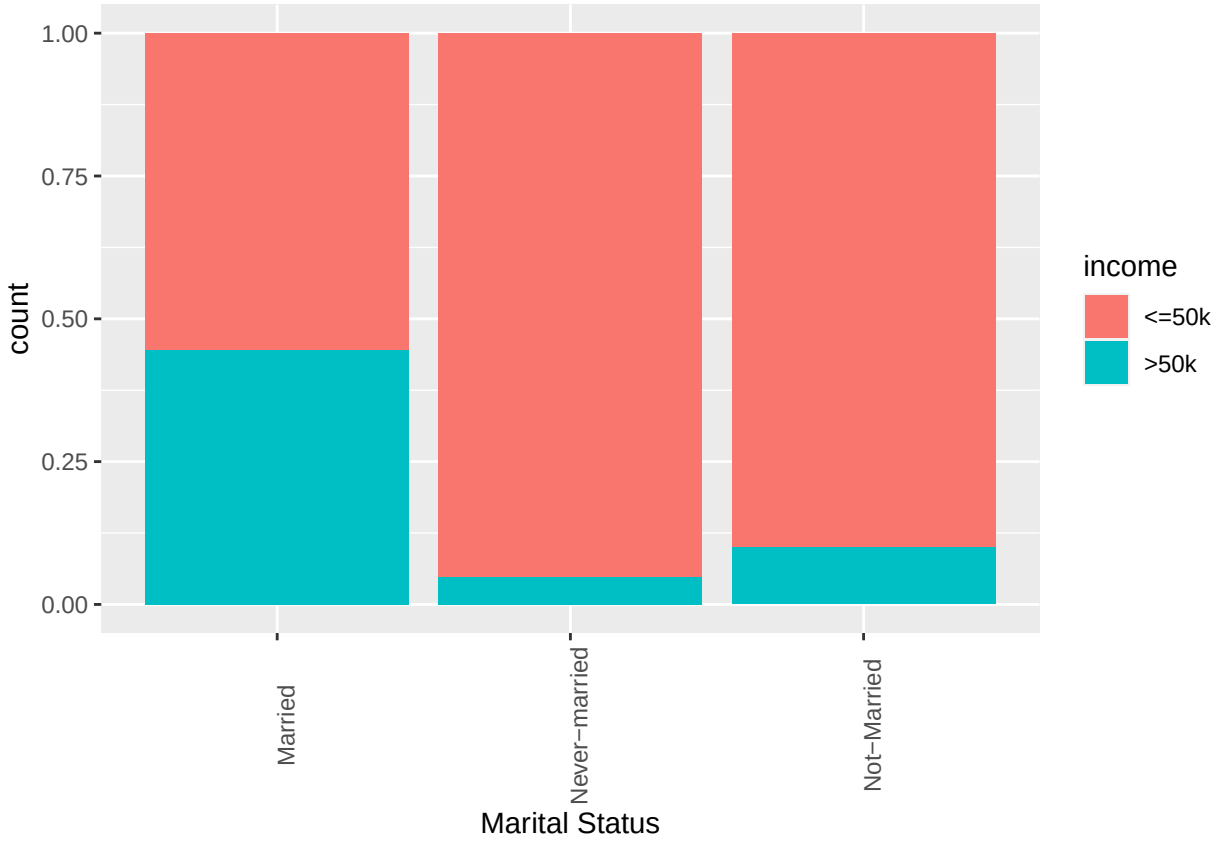
We can also make an observation by dividing the age into different ranges by 5-year increment and then we plot the proportion of income lower or higher than 50k.

```
range <- as.factor(c("-20","21-25","26-30","31-35","36-40","41-45","46-50","51-55","56-60","60-65","66+
age_range <- sapply(incomes_ds$age,function(age) {
  if(age <= 20) range[1]
  else if (age <= 25) range[2]
  else if (age <= 30) range[3]
  else if (age <= 35) range[4]
  else if (age <= 40) range[5]
  else if (age <= 45) range[6]
  else if (age <= 50) range[7]
  else if (age <= 55) range[8]
  else if (age <= 60) range[9]
  else if (age <= 65) range[10]
  else range[11]
})
incomes_ds <- incomes_ds %>% mutate(agerange=age_range)
incomes_ds %>% ggplot(aes(x=agerange,fill=income)) + geom_bar(position = "fill")+ scale_fill_discrete(la
```



We now plot the proportion of higher paid people relatively to their marital status:

```
incomes_ds %>% ggplot(aes(x=marital.status,fill=income)) + geom_bar(position = "fill")+ scale_fill_disc
```



and we see that the highest percentage of well paid people is between the ones who are married, while the lowest is between the ones who never married.

Finally, in order to simplify our analysis, we will keep the columns `agerange`, `workclass`, `education`, `marital.status`, `occupation`, `native.country`, `sex`, `race`, and `income` for our study:

```
incomes_ds <- incomes_ds %>% select(agerange,workclass,education,marital.status,occupation,native.countr
```

This dataset needs a lot of time for processing, therefore, we will randomly select 5000 entries and make our tests on this sample.

```
incomes_ds <- incomes_ds[sample(nrow(incomes_ds), 5000), ]
```

Prediction methods

We will test 10 machine learning models and then We will use an ensemble prediction by majority vote to predict the income of a person knowing his age, education level, gender.

The models that we will use are the following: “naive_bayes”, “glm”, “knn”, “rpart”, “rf”, “multinom”, and “xgbTree”.

Results

We begin by dividing our dataset into a train set and a test set:

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
```

```
## used
```

```
test_index <- createDataPartition(y = incomes_ds$income, times = 1, p = 0.15, list = FALSE)
train_set <- incomes_ds[-test_index,]
test_set <- incomes_ds[test_index,]
```

Then, we use the function train of the caret to train these models on our train set:

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
fit_naive <- train(income ~ . , method = "naive_bayes", data = train_set)
predict_naive <- predict(fit_naive,newdata = test_set)
accuracy_naive <- confusionMatrix(predict_naive,test_set$income)$overall['Accuracy']
accuracy_naive
```

```
##  Accuracy
## 0.7523302
```

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
fit_glm <- train(income ~ . , method = "glm", data = train_set,family = binomial(logit))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
predict_glm <- predict(fit_glm,newdata = test_set)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```r
accuracy_glm <- confusionMatrix(predict_glm,test_set$income)$overall['Accuracy']
accuracy_glm
```

```
##  Accuracy
## 0.8348868
```

```r
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
fit_multinom <- train(income ~ . , method = "multinom", data = train_set)
```

```
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1638.759843
## iter  20 value 1510.161870
## iter  30 value 1496.836128
## iter  40 value 1489.219491
## iter  50 value 1487.867410
## iter  60 value 1487.651316
## final  value 1487.650526
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1642.108023
## iter  20 value 1516.842763
## iter  30 value 1505.931647
## iter  40 value 1502.915864
## iter  50 value 1502.833927
## final  value 1502.833671
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
```

```
## iter  10 value 1638.763280
## iter  20 value 1510.169366
## iter  30 value 1496.848185
## iter  40 value 1489.251991
## iter  50 value 1487.942043
## iter  60 value 1487.796037
## final  value 1487.792294
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1614.652711
## iter  20 value 1504.510820
## iter  30 value 1495.418549
## iter  40 value 1490.552238
## iter  50 value 1489.352581
## iter  60 value 1489.106526
## final  value 1489.105695
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1619.671014
## iter  20 value 1513.464939
## iter  30 value 1506.172427
## iter  40 value 1504.040275
## iter  50 value 1503.917290
## final  value 1503.916734
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1614.657874
## iter  20 value 1504.521148
## iter  30 value 1495.433067
## iter  40 value 1490.576652
## iter  50 value 1489.398302
## iter  60 value 1489.210325
## final  value 1489.203132
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1661.411113
## iter  20 value 1547.496997
## iter  30 value 1534.847988
## iter  40 value 1530.260877
## iter  50 value 1528.953805
## iter  60 value 1528.584628
## final  value 1528.582304
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1666.794029
## iter  20 value 1556.873756
## iter  30 value 1546.074973
## iter  40 value 1544.225955
## iter  50 value 1544.046644
```

```
## final  value 1544.045618
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1661.416714
## iter  20 value 1547.507981
## iter  30 value 1534.862958
## iter  40 value 1530.287239
## iter  50 value 1528.996613
## iter  60 value 1528.685520
## final  value 1528.680360
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1690.141967
## iter  20 value 1564.172254
## iter  30 value 1553.685240
## iter  40 value 1549.316960
## iter  50 value 1547.832764
## iter  60 value 1547.404751
## final  value 1547.401505
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1696.202349
## iter  20 value 1573.894392
## iter  30 value 1566.253785
## iter  40 value 1564.910440
## iter  50 value 1564.828776
## final  value 1564.828647
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1690.148294
## iter  20 value 1564.183480
## iter  30 value 1553.703023
## iter  40 value 1549.345963
## iter  50 value 1547.880259
## iter  60 value 1547.514457
## final  value 1547.508626
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1638.384361
## iter  20 value 1519.641638
## iter  30 value 1509.183183
## iter  40 value 1504.467256
## iter  50 value 1503.236230
## iter  60 value 1503.106052
## final  value 1503.105642
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1642.945107
```

```
## iter  20 value 1527.423612
## iter  30 value 1518.828143
## iter  40 value 1517.053801
## iter  50 value 1516.982815
## final  value 1516.982591
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1638.389067
## iter  20 value 1519.650436
## iter  30 value 1509.195389
## iter  40 value 1504.490248
## iter  50 value 1503.292084
## iter  60 value 1503.209796
## final  value 1503.204127
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1580.465216
## iter  20 value 1483.592257
## iter  30 value 1471.613978
## iter  40 value 1466.376183
## iter  50 value 1465.177368
## iter  60 value 1464.919647
## iter  70 value 1464.914329
## final  value 1464.914276
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1584.170439
## iter  20 value 1489.064414
## iter  30 value 1480.590346
## iter  40 value 1478.079596
## iter  50 value 1478.005080
## final  value 1478.004875
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1580.469022
## iter  20 value 1483.600806
## iter  30 value 1471.625744
## iter  40 value 1466.396497
## iter  50 value 1465.219651
## iter  60 value 1465.012035
## final  value 1465.004755
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1674.699422
## iter  20 value 1573.039979
## iter  30 value 1564.644913
## iter  40 value 1560.937572
## iter  50 value 1559.836627
## iter  60 value 1559.461310
```

```
## iter  70 value 1559.459962
## final  value 1559.459402
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1678.267728
## iter  20 value 1579.444259
## iter  30 value 1573.061967
## iter  40 value 1571.687694
## iter  50 value 1571.607804
## iter  50 value 1571.607790
## iter  50 value 1571.607790
## final  value 1571.607790
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1674.703088
## iter  20 value 1573.047234
## iter  30 value 1564.655920
## iter  40 value 1560.957292
## iter  50 value 1559.871697
## iter  60 value 1559.551895
## iter  60 value 1559.551880
## iter  60 value 1559.551880
## final  value 1559.551880
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1616.091868
## iter  20 value 1513.610811
## iter  30 value 1500.438662
## iter  40 value 1493.255600
## iter  50 value 1491.561741
## iter  60 value 1491.310260
## final  value 1491.309106
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1620.726703
## iter  20 value 1524.366652
## iter  30 value 1514.309996
## iter  40 value 1511.590825
## iter  50 value 1511.537263
## final  value 1511.537110
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1616.096631
## iter  20 value 1513.623583
## iter  30 value 1500.458740
## iter  40 value 1493.297287
## iter  50 value 1491.655681
## iter  60 value 1491.477970
## final  value 1491.471888
```

```
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1685.969012
## iter  20 value 1532.774410
## iter  30 value 1521.834828
## iter  40 value 1516.687162
## iter  50 value 1515.764156
## iter  60 value 1515.303166
## final  value 1515.297721
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1691.258003
## iter  20 value 1540.999748
## iter  30 value 1532.413051
## iter  40 value 1530.445570
## iter  50 value 1530.343715
## iter  50 value 1530.343707
## iter  50 value 1530.343706
## final  value 1530.343706
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1685.974495
## iter  20 value 1532.783705
## iter  30 value 1521.848903
## iter  40 value 1516.713471
## iter  50 value 1515.803427
## iter  60 value 1515.396051
## iter  70 value 1515.393116
## iter  70 value 1515.393111
## iter  70 value 1515.393111
## final  value 1515.393111
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1674.164456
## iter  20 value 1510.825933
## iter  30 value 1500.211226
## iter  40 value 1494.367982
## iter  50 value 1492.899277
## iter  60 value 1492.671109
## final  value 1492.668282
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1677.973294
## iter  20 value 1517.436664
## iter  30 value 1508.616656
## iter  40 value 1506.094660
## iter  50 value 1505.946959
## final  value 1505.946074
## converged
```

```
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1674.168373
## iter  20 value 1510.833255
## iter  30 value 1500.221570
## iter  40 value 1494.389334
## iter  50 value 1492.947469
## iter  60 value 1492.777732
## final  value 1492.769048
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1633.450269
## iter  20 value 1512.527081
## iter  30 value 1497.541507
## iter  40 value 1492.652700
## iter  50 value 1491.184011
## iter  60 value 1490.816414
## final  value 1490.813403
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1641.386490
## iter  20 value 1519.971647
## iter  30 value 1505.521295
## iter  40 value 1503.578526
## iter  50 value 1503.418204
## final  value 1503.418001
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1633.458578
## iter  20 value 1512.534376
## iter  30 value 1497.551646
## iter  40 value 1492.672024
## iter  50 value 1491.222303
## iter  60 value 1490.915553
## final  value 1490.912305
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1641.218803
## iter  20 value 1508.905489
## iter  30 value 1497.477212
## iter  40 value 1492.302940
## iter  50 value 1490.882068
## iter  60 value 1490.454432
## iter  70 value 1490.448289
## iter  70 value 1490.448276
## iter  70 value 1490.448276
## final  value 1490.448276
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
```

```
## iter  10 value 1645.522508
## iter  20 value 1515.740683
## iter  30 value 1505.749820
## iter  40 value 1503.179652
## iter  50 value 1503.073071
## final  value 1503.072919
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1641.223227
## iter  20 value 1508.913260
## iter  30 value 1497.487689
## iter  40 value 1492.322610
## iter  50 value 1490.916138
## iter  60 value 1490.544739
## iter  70 value 1490.541611
## final  value 1490.541520
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1630.649826
## iter  20 value 1500.005733
## iter  30 value 1487.282820
## iter  40 value 1480.750401
## iter  50 value 1479.328062
## iter  60 value 1479.046093
## final  value 1479.045540
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1634.904156
## iter  20 value 1507.301593
## iter  30 value 1498.316487
## iter  40 value 1495.527795
## iter  50 value 1495.440860
## final  value 1495.440709
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1630.654186
## iter  20 value 1500.013865
## iter  30 value 1487.298101
## iter  40 value 1480.782890
## iter  50 value 1479.396261
## iter  60 value 1479.189143
## final  value 1479.186059
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1691.828859
## iter  20 value 1550.705172
## iter  30 value 1532.812902
## iter  40 value 1526.662340
## iter  50 value 1524.949038
```

```
## iter  60 value 1524.529815
## final  value 1524.528589
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1693.317183
## iter  20 value 1556.876497
## iter  30 value 1543.105937
## iter  40 value 1540.218238
## iter  50 value 1540.098082
## final  value 1540.097131
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1691.830297
## iter  20 value 1550.715275
## iter  30 value 1532.825533
## iter  40 value 1526.685576
## iter  50 value 1524.993695
## iter  60 value 1524.638465
## final  value 1524.634230
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1586.154145
## iter  20 value 1452.837416
## iter  30 value 1440.110682
## iter  40 value 1433.714804
## iter  50 value 1432.354519
## iter  60 value 1432.046676
## final  value 1432.045027
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1591.225021
## iter  20 value 1461.432104
## iter  30 value 1451.057358
## iter  40 value 1447.162348
## iter  50 value 1446.992837
## final  value 1446.992234
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1586.159409
## iter  20 value 1452.847228
## iter  30 value 1440.125827
## iter  40 value 1433.737668
## iter  50 value 1432.402161
## iter  60 value 1432.147965
## iter  70 value 1432.141672
## iter  70 value 1432.141670
## iter  70 value 1432.141670
## final  value 1432.141670
## converged
```

```
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1619.367763
## iter  20 value 1500.737797
## iter  30 value 1488.848583
## iter  40 value 1483.072189
## iter  50 value 1481.804088
## iter  60 value 1481.607955
## iter  70 value 1481.605096
## iter  70 value 1481.605091
## iter  70 value 1481.605091
## final  value 1481.605091
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1623.898494
## iter  20 value 1509.936266
## iter  30 value 1500.796911
## iter  40 value 1497.761815
## iter  50 value 1497.714882
## final  value 1497.714756
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1619.372403
## iter  20 value 1500.748404
## iter  30 value 1488.864536
## iter  40 value 1483.098419
## iter  50 value 1481.857411
## iter  60 value 1481.709727
## iter  70 value 1481.707008
## iter  70 value 1481.706996
## iter  70 value 1481.706996
## final  value 1481.706996
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1677.440557
## iter  20 value 1540.270514
## iter  30 value 1527.465780
## iter  40 value 1521.707805
## iter  50 value 1520.055926
## iter  60 value 1519.677963
## final  value 1519.675982
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1681.656188
## iter  20 value 1548.219365
## iter  30 value 1537.947409
## iter  40 value 1535.080458
## iter  50 value 1534.980846
## final  value 1534.980218
## converged
```

```
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1677.444877
## iter  20 value 1540.279508
## iter  30 value 1527.479600
## iter  40 value 1521.730996
## iter  50 value 1520.104748
## iter  60 value 1519.792383
## final  value 1519.785419
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1662.538830
## iter  20 value 1515.395617
## iter  30 value 1503.393634
## iter  40 value 1498.329351
## iter  50 value 1497.281732
## iter  60 value 1496.888269
## final  value 1496.883758
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1654.090419
## iter  20 value 1522.531354
## iter  30 value 1513.075873
## iter  40 value 1511.005349
## iter  50 value 1510.861139
## final  value 1510.860762
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1662.544573
## iter  20 value 1515.404564
## iter  30 value 1503.405949
## iter  40 value 1498.354137
## iter  50 value 1497.324051
## iter  60 value 1496.987430
## iter  70 value 1496.982452
## iter  70 value 1496.982449
## iter  70 value 1496.982449
## final  value 1496.982449
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1583.043492
## iter  20 value 1476.558191
## iter  30 value 1464.032380
## iter  40 value 1457.883797
## iter  50 value 1456.516620
## iter  60 value 1456.222787
## final  value 1456.219490
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
```

```
## iter  10 value 1586.632513
## iter  20 value 1483.296925
## iter  30 value 1472.204784
## iter  40 value 1469.166186
## iter  50 value 1468.965960
## final  value 1468.965756
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1583.047202
## iter  20 value 1476.565899
## iter  30 value 1464.043115
## iter  40 value 1457.904217
## iter  50 value 1456.557246
## iter  60 value 1456.315922
## iter  70 value 1456.314224
## iter  70 value 1456.314217
## iter  70 value 1456.314217
## final  value 1456.314217
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1591.645942
## iter  20 value 1463.283451
## iter  30 value 1450.374788
## iter  40 value 1444.050563
## iter  50 value 1442.511913
## iter  60 value 1442.214385
## final  value 1442.212443
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1597.537233
## iter  20 value 1472.161084
## iter  30 value 1460.773436
## iter  40 value 1457.415739
## iter  50 value 1457.314971
## final  value 1457.314682
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1591.652046
## iter  20 value 1463.293550
## iter  30 value 1450.387964
## iter  40 value 1444.073050
## iter  50 value 1442.566560
## iter  60 value 1442.322605
## final  value 1442.316251
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1593.244263
## iter  20 value 1473.421827
## iter  30 value 1459.307308
```

```
## iter  40 value 1453.794488
## iter  50 value 1452.135402
## iter  60 value 1451.782053
## final  value 1451.780890
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1598.066563
## iter  20 value 1483.108937
## iter  30 value 1471.803702
## iter  40 value 1468.938034
## iter  50 value 1468.789147
## final  value 1468.788814
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1593.249227
## iter  20 value 1473.433170
## iter  30 value 1459.324105
## iter  40 value 1453.821059
## iter  50 value 1452.185948
## iter  60 value 1451.894529
## final  value 1451.890093
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1659.026171
## iter  20 value 1523.117340
## iter  30 value 1504.673556
## iter  40 value 1499.747786
## iter  50 value 1497.962607
## iter  60 value 1497.514804
## final  value 1497.513592
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1667.033585
## iter  20 value 1527.653857
## iter  30 value 1513.857555
## iter  40 value 1511.427328
## iter  50 value 1511.094677
## final  value 1511.093212
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1659.034590
## iter  20 value 1523.120272
## iter  30 value 1504.684970
## iter  40 value 1499.766398
## iter  50 value 1497.999794
## iter  60 value 1497.607770
## final  value 1497.605189
## converged
## # weights:  54 (53 variable)
```

```
## initial  value 2945.182370
## iter  10 value 1643.111135
## iter  20 value 1510.137631
## iter  30 value 1499.494683
## iter  40 value 1494.580936
## iter  50 value 1493.205332
## iter  60 value 1492.891734
## final  value 1492.891310
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1647.346900
## iter  20 value 1516.323513
## iter  30 value 1507.142201
## iter  40 value 1504.814335
## iter  50 value 1504.580341
## final  value 1504.579561
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1643.115478
## iter  20 value 1510.144427
## iter  30 value 1499.503946
## iter  40 value 1494.598183
## iter  50 value 1493.240930
## iter  60 value 1492.983466
## final  value 1492.981906
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1620.142941
## iter  20 value 1510.750983
## iter  30 value 1500.056725
## iter  40 value 1495.049584
## iter  50 value 1494.015185
## iter  60 value 1493.727934
## final  value 1493.727370
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1623.868220
## iter  20 value 1516.106081
## iter  30 value 1507.930002
## iter  40 value 1505.633930
## iter  50 value 1505.564055
## final  value 1505.563889
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1620.146749
## iter  20 value 1510.756662
## iter  30 value 1500.066341
## iter  40 value 1495.068772
## iter  50 value 1494.052926
```

```
## final  value 1493.821285
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1680.660985
## iter  20 value 1532.176325
## iter  30 value 1519.502491
## iter  40 value 1514.442905
## iter  50 value 1513.104893
## iter  60 value 1512.700683
## iter  70 value 1512.696071
## iter  70 value 1512.696063
## iter  70 value 1512.696063
## final  value 1512.696063
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1686.613858
## iter  20 value 1540.259785
## iter  30 value 1530.312358
## iter  40 value 1528.463633
## iter  50 value 1528.341745
## final  value 1528.341296
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1680.667193
## iter  20 value 1532.185713
## iter  30 value 1519.517140
## iter  40 value 1514.470088
## iter  50 value 1513.154577
## iter  60 value 1512.804751
## iter  70 value 1512.799121
## final  value 1512.799082
## converged
## # weights:  54 (53 variable)
## initial  value 2945.182370
## iter  10 value 1655.174391
## iter  20 value 1538.700052
## iter  30 value 1530.691589
## iter  40 value 1528.605094
## iter  50 value 1528.503900
## final  value 1528.503496
## converged
```

```
predict_multinom <- predict(fit_multinom,newdata = test_set)
accuracy_multinom <- confusionMatrix(predict_multinom,test_set$income)$overall['Accuracy']
accuracy_multinom
```

```
##  Accuracy
## 0.8322237
```

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
```

```
## used
```

```
fit_knn <- train(income ~ . , method = "knn", data = train_set,tuneGrid = data.frame(k = c(3,5,7,9)))
predict_knn <- predict(fit_knn,newdata = test_set)
accuracy_knn <- confusionMatrix(predict_knn,test_set$income)$overall['Accuracy']
accuracy_knn
```

```
##  Accuracy
## 0.8268975
```

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
fit_rpart <- train(income ~ . ,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                     data = train_set)
predict_rpart <- predict(fit_rpart,newdata = test_set)
accuracy_rpart <- confusionMatrix(predict_rpart,test_set$income)$overall['Accuracy']
accuracy_rpart
```

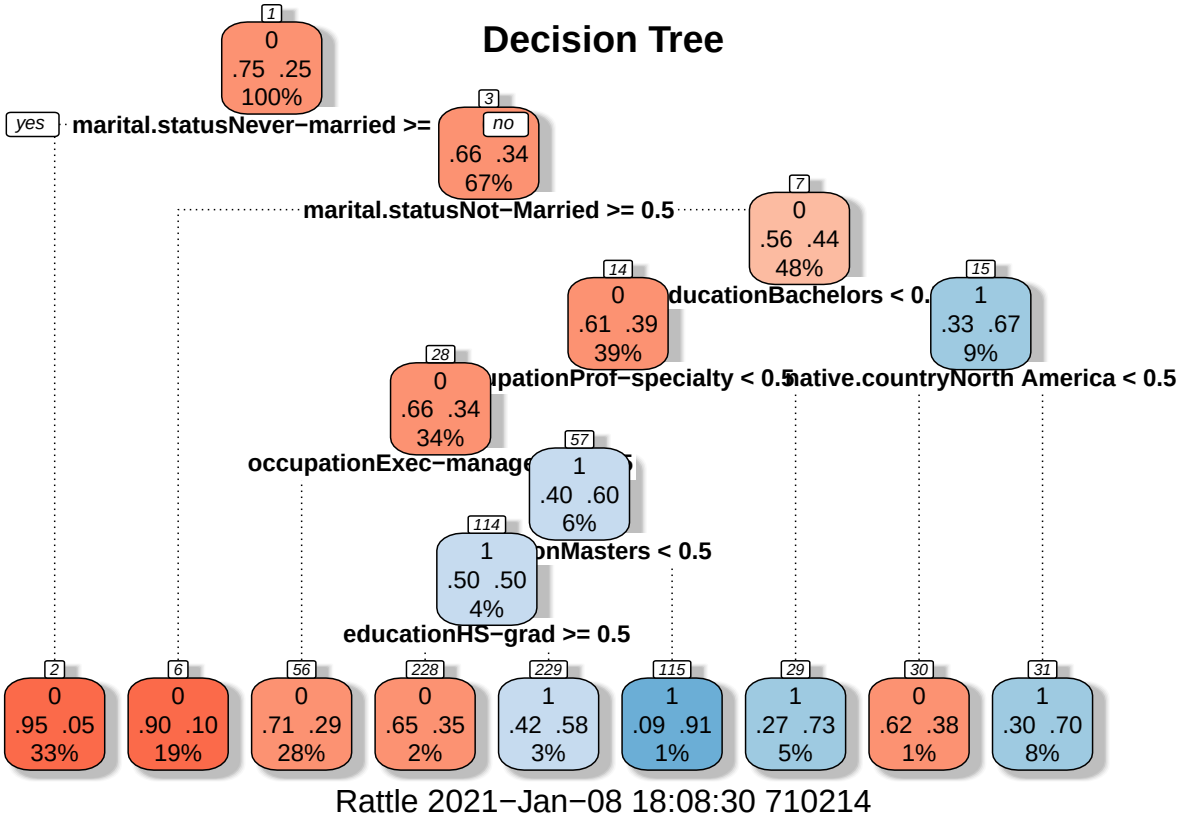
```
##  Accuracy
## 0.8175766
```

```
library(rattle)
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
fancyRpartPlot(fit_rpart$finalModel,cex=0.75,main="Decision Tree",palettes = "Reds")
```




```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
fit_rf <- train(income ~ . , method = "rf", data = train_set,tuneGrid = data.frame(mtry = c(1, 5, 10)))
predict_rf <- predict(fit_rf,newdata = test_set)
accuracy_rf <- confusionMatrix(predict_rf,test_set$income)$overall['Accuracy']
accuracy_rf
```

```
##  Accuracy
## 0.8308921
```

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
fit_xgbTree <- train(income ~ . , method = "xgbTree", data = train_set)
predict_xgbTree <- predict(fit_xgbTree,newdata = test_set)
accuracy_xgbTree <- confusionMatrix(predict_xgbTree,test_set$income)$overall['Accuracy']
accuracy_xgbTree
```

```
##  Accuracy
## 0.8362184
```

We will build now an ensemble prediction by majority vote. We will vote that the income is higher than 50k if more than 50% of the models are predicting that, and lower otherwise.

```
pos_glm <- predict_glm == 1
pos_knn <- predict_knn == 1
pos_naive <- predict_naive == 1
pos_rf <- predict_rf == 1
pos_rpart <- predict_rpart == 1
pos_multinom <- predict_multinom == 1
pos_xgbTree <- predict_xgbTree == 1
predictions <- cbind(pos_glm,pos_knn,pos_naive,pos_rf,pos_rpart,pos_xgbTree,pos_multinom)
votes <- rowMeans(predictions)
y_hat <- ifelse(votes > 0.5, "1", "0")
accuracy_ens <- mean(y_hat == test_set$income)
accuracy_ens
```

```
## [1] 0.8362184
```

We notice that the ensemble accuracy is higher than the accuracy of the "knn"-model, the "naive-bayes"-model, "rf"-model, "rpart"-model, and the "xgbTree"-model but lower than the accuracy of the "glm"-model and the "multinom"-model.

Conclusion and Perspectives

In conclusion, we have tried in this report 7 different models to predict if a certain person earns more than 50k per year based on the following parameters: Age range, Work-class, Education level, Marital status, Occupation, Native country, Gender, and Race. The models that we tried are the following: "naive_bayes", "glm", "knn", "rpart", "rf", "multinom", and "xgbTree". Then, we built an ensemble prediction by majority vote, and an ensemble prediction by removing the models that gave low accuracy. The best accuracy was achieved using the "glm"-model.

This work is just the beginning of the analysis and there's many steps that can be carried further. - The first main step is to make the analysis using the whole dataset, which I was not able to carry because of the limitation of my machine. - One idea is to better analyze each of the attributes and choose the ones with the highest influence. This step can reduce the number of parameters and hence improve the computation time. - Another weak point of this analysis is the distribution of the observations between continents with the vast majority coming from North America and specifically from the United States. It might be more interesting to make the study for the North American residents only. - It's also important to note that this data is old and the results and conclustions that we derive might not be applied to the current time.

I end this report with a quote that I saw while searching for my project: "In ancient times, the ability to predict the future was called precognition. Nowadays we call it machine learning."