

580Project

Zian Shang, Tasfia Shaikh, Thomas Huang, Nicole Dona, Matthew Davison

2025-04-23

```
# include packages
library(fastDummies)
library(caret)
```

```
## Loading required package: ggplot2
```

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

```
library(ggplot2)
library(ggrepel)
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
```

```
## v dplyr      1.1.4      v readr      2.1.5
## v forcats    1.0.0      v stringr   1.5.1
## v lubridate  1.9.4      v tibble    3.2.1
## v purrr      1.0.2      v tidyr     1.3.1
```

```
## -- Conflicts ----- tidyverse_conflicts() --
```

```
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x purrr::lift()    masks caret::lift()
```

```
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(randomForest)
```

```
## randomForest 4.7-1.2
```

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

```
##
```

```
## Attaching package: 'randomForest'
```

```
##
```

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

```
##
```

```
##      combine
```

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

```
library(nnet)
library(rpart)
library(rpart.plot)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(vcdExtra)
```

```
## Loading required package: vcd
## Loading required package: grid
## Loading required package: gnm
##
## Attaching package: 'gnm'
##
## The following object is masked from 'package:lattice':
##
##     barley
##
## Attaching package: 'vcdExtra'
##
```

```
## The following object is masked from 'package:dplyr':  
##  
##     summarise
```

```
library(vcd)  
library(rpart)  
library(rattle)
```

```
## Loading required package: bitops  
## Rattle: A free graphical interface for data science with R.  
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.  
##  
## Attaching package: 'rattle'  
##  
## The following object is masked from 'package:xgboost':  
##  
##     xgboost  
##  
## The following object is masked from 'package:randomForest':  
##  
##     importance
```

```
library(partykit)
```

```
## Loading required package: libcoin  
## Loading required package: mvtnorm  
##  
## Attaching package: 'mvtnorm'  
##  
## The following object is masked from 'package:gnm':  
##  
##     Mult
```

```
library(NeuralNetTools) # for neural net plots
```

```
# read train.data  
train.data <- read.csv("train.csv",  
                      header = T,  
                      na.strings = " ?")  
  
# remove NA  
train.data <- na.omit(train.data)  
  
# remove empty space in front of each char cell  
# factorize char columns  
  
# remove fnlwgt  
train.data <- subset(train.data, select = -c(fnlwgt))  
  
train.data[] <- lapply(train.data,  
                      function(x) {
```

```

        if(is.character(x))
          as.factor(trimws(x))
        else x
      })

# View(train.data)
# str(train.data)

# read test data
test.data <- read.csv("~/Desktop/IncomeClassification580/test.csv",
                     header = T,
                     na.strings = " ?")

# remove NA
test.data <- na.omit(test.data)

# remove fnlwgt
test.data <- subset(test.data, select = -c(fnlwgt))

# remove empty space in front of each char cell
test.data[] <- lapply(test.data,
                     function(x) {
                       if(is.character(x))
                         as.factor(trimws(x))
                       else x
                     })
test.data$native.country <- factor(test.data$native.country,
                                   levels = levels(train.data$native.country))

# View(test.data)
# str(test.data)

```

variable selection, identify important features

outliers -> Matthew

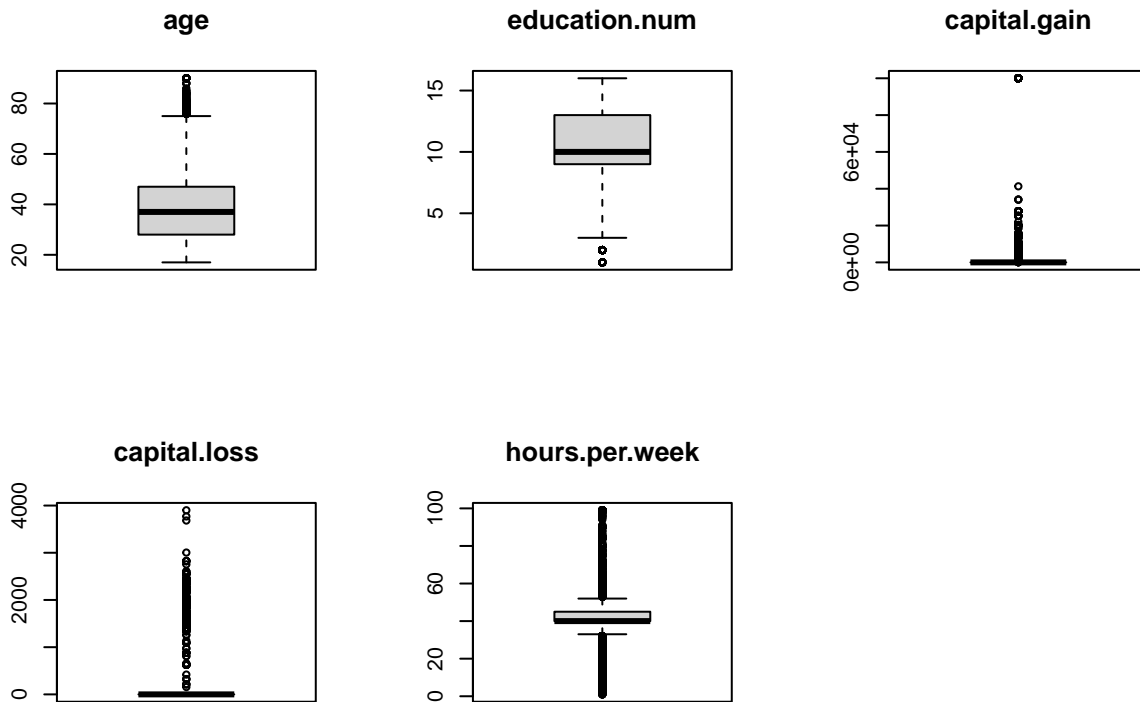
```

# Start with box plots to visualize which variables have outliers
numeric_vars <- names(train.data)[sapply(train.data, is.numeric)]
numeric_data <- train.data[, numeric_vars]

par(mfrow = c(2,3))
for(n in numeric_vars) {
  boxplot(train.data[[n]], main = n)
}

par(mfrow = c(1,1))

```



Capital gain and loss are highly skewed variables with many 0s, treat as binary?

Age has a decent amount of observations ≥ 80 , far from the average

Education.num seems clean

fnlwgt might not have any interpretable meaning, drop?

hours.per.week is centered around 40, but with considerable values < 20 and > 60 , what is reasonable?

z-score method

```
z.scores <- scale(numeric_data)
```

```
outlier.indices <- which(apply(abs(z.scores), 1, function(x) any(x>3)))
```

```
outliers <- numeric_data[outlier.indices, ]
```

```
str(outliers)
```

```
## 'data.frame': 1809 obs. of 5 variables:
```

```
## $ age : int 37 43 39 45 47 79 48 20 24 45 ...
```

```
## $ education.num : int 10 7 9 13 15 10 16 10 9 11 ...
```

```
## $ capital.gain : int 0 0 0 0 0 0 0 0 0 0 ...
```

```
## $ capital.loss : int 0 2042 0 1408 1902 0 1902 1719 1762 1564 ...
```

```
## $ hours.per.week: int 80 40 80 40 60 20 60 28 40 40 ...
```

```
rm(z.scores, outlier.indices, outliers)
```

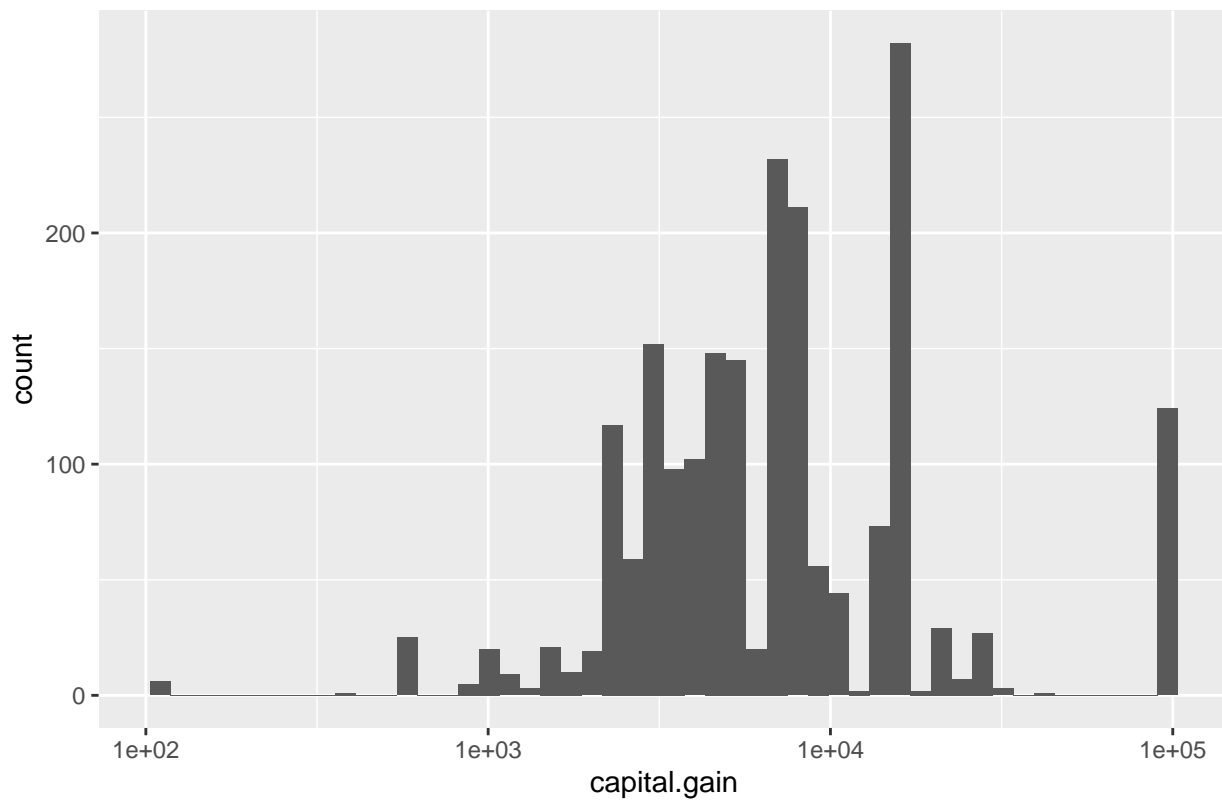
```
# look at distributions of some vars
```

```
ggplot(train.data, aes(x = capital.gain)) +  
  geom_histogram(bins = 50) +  
  scale_x_log10() +  
  ggtitle("Capital Gain Distribution (log scale)")
```

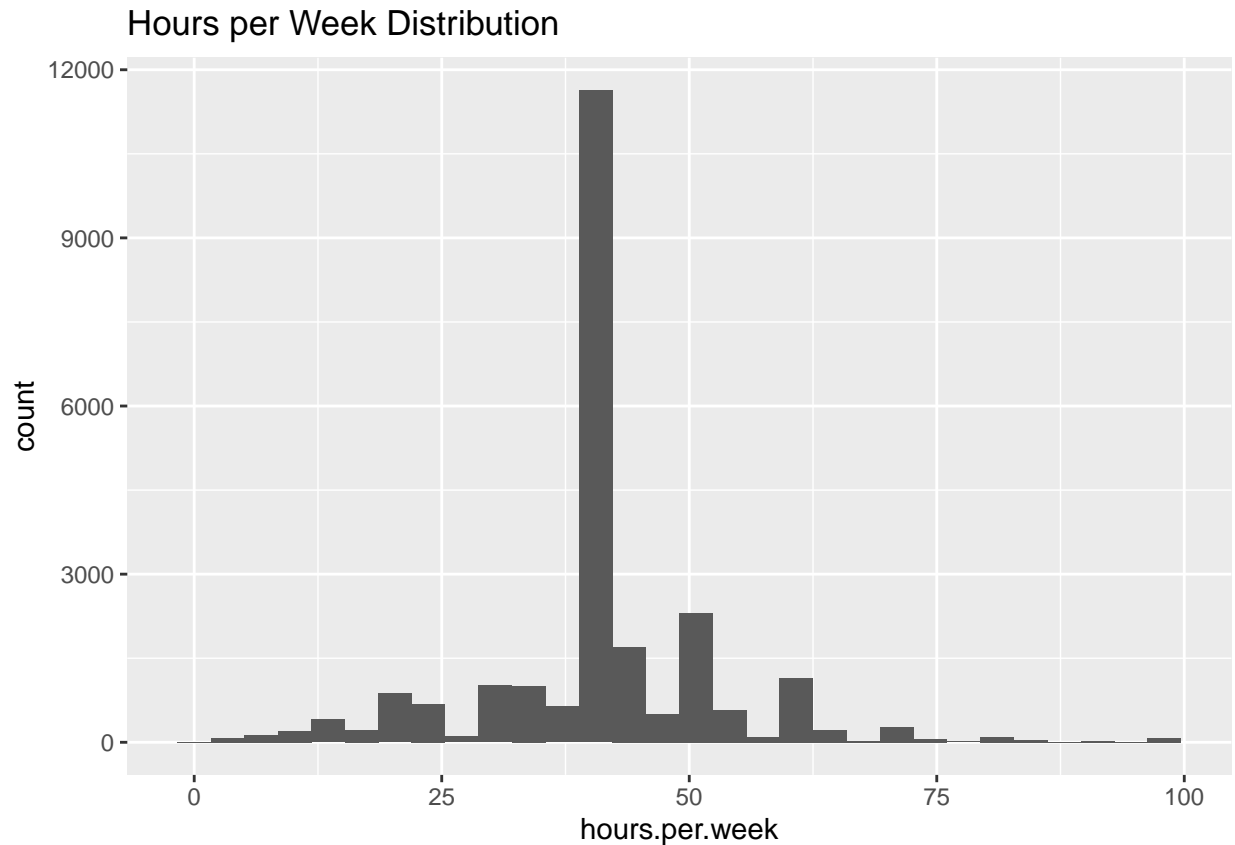
```
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
```

```
## Warning: Removed 22094 rows containing non-finite outside the scale range  
## ('stat_bin()').
```

Capital Gain Distribution (log scale)



```
ggplot(train.data, aes(x = hours.per.week)) +  
  geom_histogram(bins = 30) +  
  ggtitle("Hours per Week Distribution")
```



If keeping capital.gain, maybe remove values <1000 or >50000 Consider trimming hours per week of values <20 or >60, there are not many

Multivariate outliers

```
numeric_data <- na.omit(train.data[, numeric_vars])

dists <- mahalanobis(numeric_data, colMeans(numeric_data), cov(numeric_data))
threshold <- qchisq(0.975, df = length(numeric_vars))

mv.outliers <- numeric_data[which(dists > threshold), ]
str(mv.outliers)
```

```
## 'data.frame': 1773 obs. of 5 variables:
## $ age : int 43 47 79 48 20 24 45 64 27 51 ...
## $ education.num : int 7 15 10 16 10 9 11 7 9 10 ...
## $ capital.gain : int 0 0 0 0 0 0 0 0 0 0 ...
## $ capital.loss : int 2042 1902 0 1902 1719 1762 1564 2179 1980 1977 ...
## $ hours.per.week: int 40 60 20 60 28 40 40 40 40 40 ...
```

```
rm(numeric_data, dists, threshold, mv.outliers)
```

IQR method to remove outliers for column hours.per.week → lowered model accuracy, withdraw

```
# Q1 <- quantile(train.data$hours.per.week, 0.25, na.rm = TRUE)
# Q3 <- quantile(train.data$hours.per.week, 0.75, na.rm = TRUE)
```

```
# IQR <- Q3 - Q1

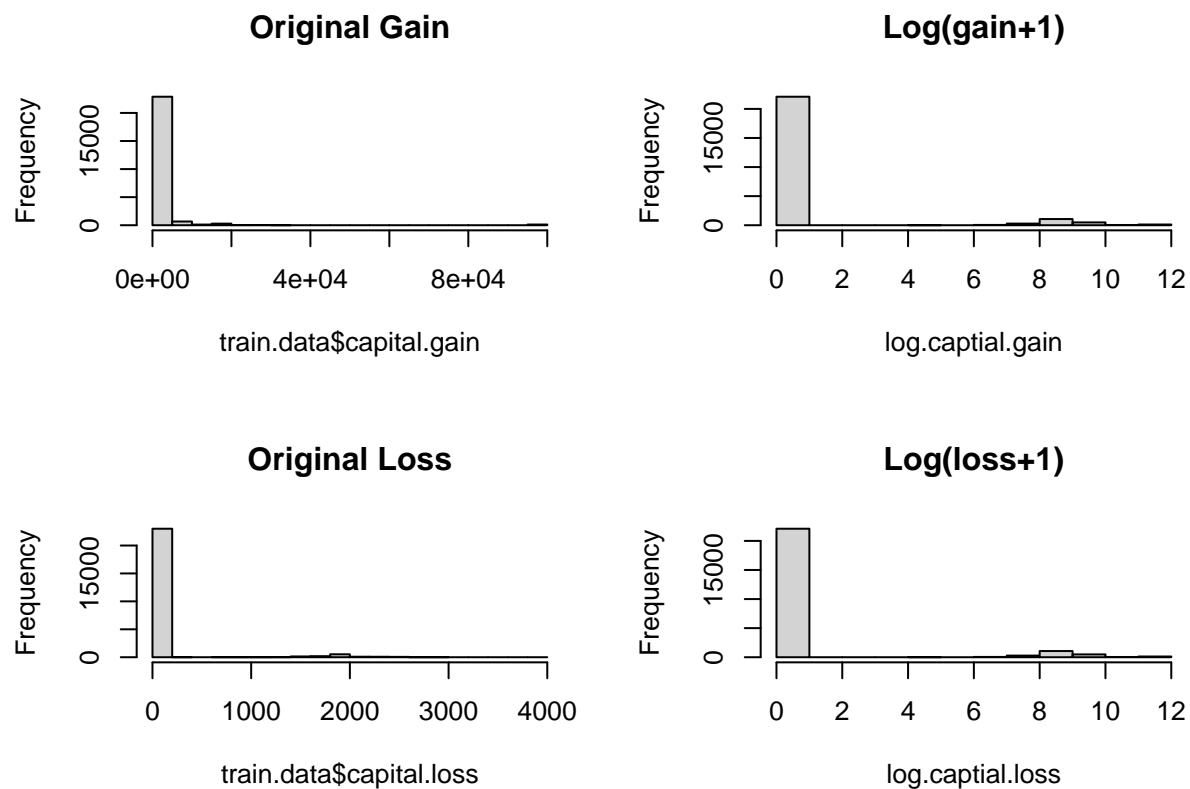
# lower <- Q1 - 1.5 * IQR
# upper <- Q3 + 1.5 * IQR

# train.data <- train.data[train.data$hours.per.week >= lower & train.data$hours.per.week <= upper, ]
```

Testing log transform – Doesn't seem to help

```
log.captial.gain <- log(train.data$capital.gain + 1)
log.captial.loss <- log(train.data$capital.loss + 1)

par(mfrow = c(2,2))
hist(train.data$capital.gain, main = "Original Gain")
hist(log.captial.gain, main = "Log(gain+1)")
hist(train.data$capital.loss, main = "Original Loss")
hist(log.captial.loss, main = "Log(loss+1)")
```



```
par(mfrow = c(1,1))

rm(log.captial.gain, log.captial.loss)
```

Use binary flags instead


```

has.gain <- as.factor(ifelse(train.data$capital.gain > 0, 1, 0))
has.loss <- as.factor(ifelse(train.data$capital.loss > 0, 1, 0))

test.has.gain <- as.factor(ifelse(test.data$capital.gain > 0, 1, 0))
test.has.loss <- as.factor(ifelse(test.data$capital.loss > 0, 1, 0))

```

correlation analysis-> Tasfia

```

# Subset only numeric predictors
numeric_vars <- names(train.data)[sapply(train.data, is.numeric)]
numeric_data <- train.data[, numeric_vars]

```

```

# Compute correlation matrix
cor_matrix <- cor(numeric_data)
cor_matrix

```

```

##              age education.num capital.gain capital.loss
## age          1.00000000    0.03781210    0.08064292    0.05948878
## education.num 0.03781210    1.00000000    0.12437130    0.08487161
## capital.gain  0.08064292    0.12437130    1.00000000   -0.03230727
## capital.loss  0.05948878    0.08487161   -0.03230727    1.00000000
## hours.per.week 0.09908376    0.14973533    0.08135083    0.05005427
##              hours.per.week
## age          0.09908376
## education.num 0.14973533
## capital.gain  0.08135083
## capital.loss  0.05005427
## hours.per.week 1.00000000

```

```

# Display heat map
# Display heat map
corrplot(cor_matrix,
  method = "color",
  col = colorRampPalette(c("white", "lightpink", "deeppink4"))(10),
  tl.col = "black",
  tl.cex = 0.9,
  addCoef.col = "black",
  number.cex = 0.7)

```



*# no strong linear dependencies among numeric predictors (all $|r| < 0.15$),
so no further variable removal due to multicollinearity is needed*

```
rm(numeric_vars, numeric_data, cor_matrix)
```

Correlation Matrix for Categorical Variables (just testing this out): -> Tasfia

```
# Select categorical columns
cat_vars <- train.data[, sapply(train.data, is.factor)]

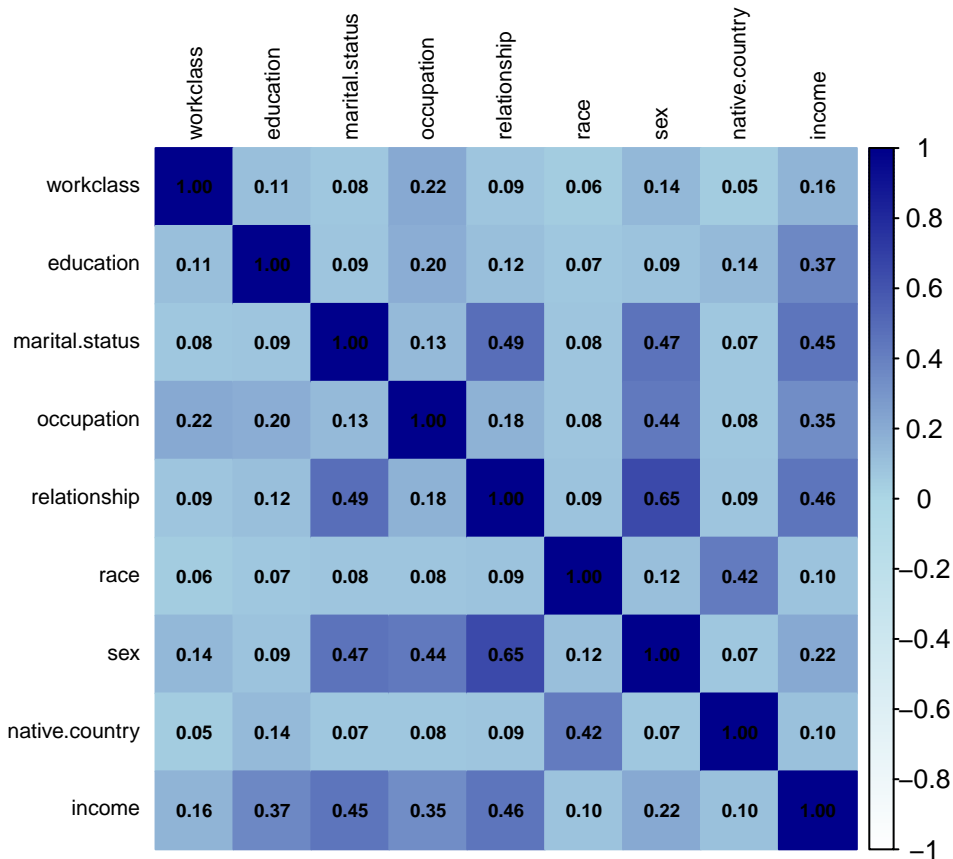
# Function to compute Cramér's V matrix
cramers_v_matrix <- function(df) {
  vars <- names(df)
  mat <- matrix(NA, ncol = length(vars), nrow = length(vars),
               dimnames = list(vars, vars))
  for (i in vars) {
    for (j in vars) {
      tbl <- table(df[[i]], df[[j]])
      mat[i, j] <- assocstats(tbl)$cramer
    }
  }
  return(as.data.frame(mat))
}

cramers_v_df <- crammers_v_matrix(cat_vars)
```

```

cramer_mat <- as.matrix(cramers_v_df)
corrplot(cramer_mat,
  method = "color",
  col = colorRampPalette(c("white", "lightblue", "darkblue"))(200),
  type = "full",
  tl.col = "black",
  tl.cex = 0.7,
  na.label = " ",
  addCoef.col = "black",
  number.cex = 0.6)

```



```
rm(cat_vars, crammers_v_df, cramer_mat)
```

Variance Thresholding -> Zian

```

# get all numeric features
numeric.data <- train.data[, sapply(train.data, is.numeric)]

# look for numbers of columns with variance near 0
nzv <- nearZeroVar(numeric.data)

# display low variance columns
head(numeric.data[, nzv], 2)

## capital.gain capital.loss

```

```
## 1      0      0
## 2      0      0
```

NOTE: ### 'capital.gain' & 'capital.loss' are mostly 0's but have some very large values.
They are classic long-tailed features, KEEP them for now.

tree-based models (CART, RF, XGBoost) -> dataset 1

```
# -> use education(categorical, factorized)
# -> no scaling
# -> no dummy encoding

### NOTE: ### Tree-based models can natively handle categorical variables (factors).
# So keep 'education' factor var, remove 'education.num'.
# No scaling or dummy encoding is needed for these models.

train.data.1 <- subset(train.data, select = -c(education.num))
test.data.1 <- subset(test.data, select = -c(education.num))
# identical(names(train.data.1), names(test.data.1))      # true
```

Linear/Logistic/NN/KNN -> dataset 2

```
# -> use education.num(numeric, original)
# -> scaling
# -> dummy encode all categorical variables

### preprocess train.data
train.data.2 <- subset(train.data, select = -c(education))
train.data.2$capital.gain <- has.gain
train.data.2$capital.loss <- has.loss

# obtain numeric & categorical column names
num_cols <- names(train.data.2)[sapply(train.data.2, is.numeric)]
num_cols <- num_cols[num_cols != "education.num"]
### education.num is treated as an ordinal categorical variable (1-16), so we do not scale it.

cat_cols <- names(train.data.2)[sapply(train.data.2, is.factor)]
cat_cols <- setdiff(cat_cols, "income") # exclude target

# scale numeric data
scaled_data <- scale(train.data.2[, num_cols])
train.data.2[, num_cols] <- scaled_data

# obtain means & stds for KNN
means <- attr(scaled_data, "scaled:center")
sds <- attr(scaled_data, "scaled:scale")

# dummy encode all categorical variables
### note: it is normal that there are more columns being created ###
```

```

train.data.2 <- dummy_cols(
  train.data.2,
  select_columns = cat_cols,
  remove_first_dummy = TRUE,
  remove_selected_columns = TRUE
)

# -----

### preprocess test.data in the same way
test.data.2 <- subset(test.data, select = -c(education))

# num_cols are the same

test.data.2$capital.gain <- test.has.gain
test.data.2$capital.loss <- test.has.loss

# scale test data using means & stds from the train data
test.data.2[, num_cols] <- sweep(test.data.2[, num_cols], 2, means, "-")
test.data.2[, num_cols] <- sweep(test.data.2[, num_cols], 2, sds, "/")

# dummy encode all categorical variables
test.data.2 <- dummy_cols(
  test.data.2,
  select_columns = cat_cols,
  remove_first_dummy = TRUE,
  remove_selected_columns = TRUE
)

# check missing columns
# setdiff(names(train.data.2), names(test.data.2))

### NOTE: ### level "Holand-Netherlands" appeared in the training set but
# not in the test set, so dummy encoding did not create this column in the test data.
# We manually add the missing dummy variable and set it to 0 for all rows.
test.data.2[["native.country_Holand-Netherlands"]] <- 0

### NOTE: ### check if column names of train and test datasets are identical.
# model prediction requires the exact same column order, so align test.data
# columns to match train.data.

# identical(names(train.data.2), names(test.data.2))      # false
test.data.2 <- test.data.2[, names(train.data.2)]

# -----

# testing: variance thresholding for preprocessed train.data.2

# get all numeric features
numeric.data <- train.data.2[, sapply(train.data.2, is.numeric)]

# look for numbers of columns with variance near 0

```

```

nzv <- nearZeroVar(numeric.data)

# obtain low variance columns, except for "capital.loss", "capital.gain"
low.var.cols <- names(numeric.data[, nzv])
low.var.cols <- setdiff(low.var.cols, c("capital.loss", "capital.gain"))
# low.var.cols

# remove them from train.data.2 & test.data.2
train.data.2 <- train.data.2[, !(names(train.data.2) %in% low.var.cols)]
test.data.2 <- test.data.2[, !(names(test.data.2) %in% low.var.cols)]

# check if columns are in identical order
# identical(names(train.data.2), names(test.data.2))    --> true

# -----

# remove temp variables
rm(scaled_data, low.var.cols, nzv)

```

PCA test for variable importance, only on train.data.2 -> Zian

```

# reason: all vars are numeric, either dummy or continuous, and is standardized
# for building Linear/Logistic/NN/KNN models

# obtain new numeric features (after all the processes above)
numeric.data <- train.data.2[, sapply(train.data.2, is.numeric)]

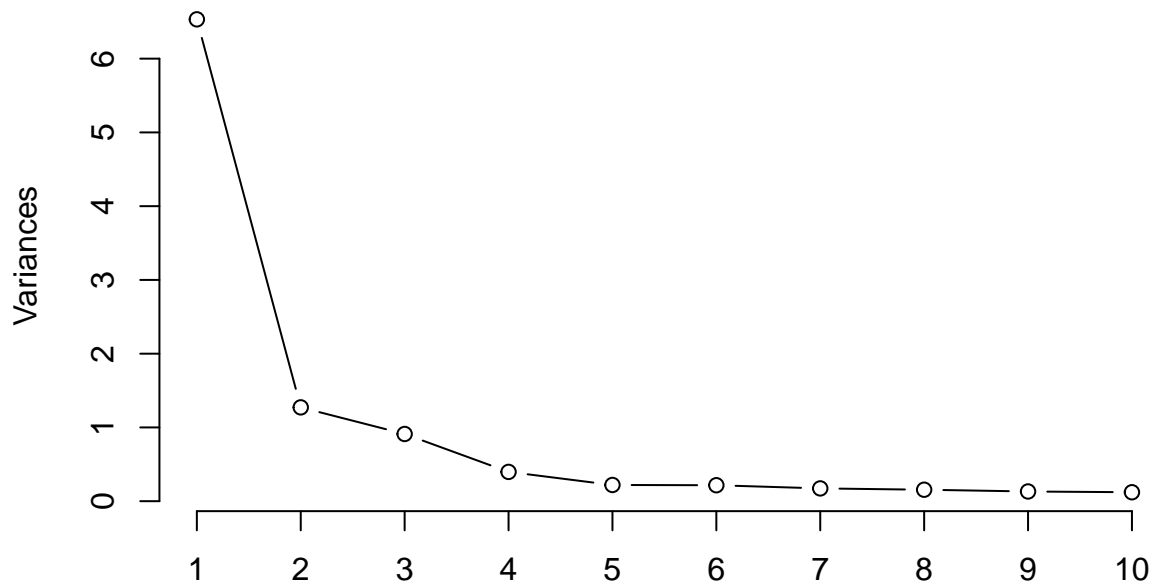
# get pca result
pca_result <- prcomp(numeric.data, center = TRUE, scale. = FALSE)
# summary(pca_result)

# -----

plot(pca_result, type = "l", main = "Scree Plot: Variance Explained by Each Principal Component")

```

Scree Plot: Variance Explained by Each Principal Component



```
# first principal component (PC1) explains a significantly larger portion of the variance
```

```
# check variable loading of PC 1,2,3,4
# they explain most of the variance before the scree plot flattens.
loadings <- round(pca_result$rotation[, 1:4], 3)

# get all vars with loadings < 0.01 (low contribution)
low.contrib.vars <- rownames(loadings)[apply(abs(loadings) < 0.01, 1, all)]

cat("Low-contribution variables (loading < 0.01 across PC1-PC4):\n")
```

```
## Low-contribution variables (loading < 0.01 across PC1-PC4):
```

```
print(low.contrib.vars)
```

```
## character(0)
```

```
# remove vars with low contribution (loading near 0)
# to reduce dimensionality without significant information loss
train.data.2 <- train.data.2[, !(names(train.data.2) %in% low.contrib.vars)]
test.data.2 <- test.data.2[, !(names(test.data.2) %in% low.contrib.vars)]

# -----
```

```
# check if columns are in identical order
identical(names(train.data.2), names(test.data.2))
```

```
## [1] TRUE
```

```
dim(train.data.2)
```

```
## [1] 24147    24
```

```
dim(test.data.2)
```

```
## [1] 6015    24
```

```
# -----
# new pca result
pca_result <- prcomp(train.data.2[, sapply(train.data.2, is.numeric)],
                     center = TRUE,
                     scale. = FALSE)

# remove temp variables
rm(numeric.data, loadings, low.contrib.vars)
```

Optional: PCA datasets for modeling(if implementing)

```
# -> remove all categorical variables
# -> scaling
# train.data.3 <- train.data[, sapply(train.data, is.numeric)]
# train.data.3 <- scale(train.data.3)
```

*** remember to optimize model parameters***

model 1: multiple linear model -> Thomas

```
### while testing, do not only run this code chunk,
###clear the environment & run from the beginning

# Convert income column to **numeric**
train.data.2$income <- as.numeric(ifelse(train.data.2$income == ">50K",1,0))
test.data.2$income <- as.numeric(ifelse(test.data.2$income == ">50K",1,0))

# -----

#fit mlr
mlr_model <- lm(income ~ ., data = train.data.2)
#prediction
pred_mlr <- predict(mlr_model, test.data.2)
```



```

# find y and y_hat
y <- test.data.2$income
yhat <- pred_mlr

# -----
# Predict class
yhat_binary <- ifelse(pred_mlr > 0.5, 1, 0)

# MSE
mse_mlr <- mean((y - yhat)^2)
print(paste("MLR MSE:", mse_mlr))

## [1] "MLR MSE: 0.122564581358856"

# Accuracy
accuracy_mlr <- mean(y == yhat_binary)
print(paste("MLR Accuracy:", accuracy_mlr))

## [1] "MLR Accuracy: 0.828927680798005"

cat("\n")

# -----
cm_mlr <- confusionMatrix(factor(yhat_binary), factor(y), positive = "1")
cm_mlr

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 4210  711
##           1  318  776
##
##           Accuracy : 0.8289
##           95% CI : (0.8192, 0.8384)
##           No Information Rate : 0.7528
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.4956
##
##           McNemar's Test P-Value : < 2.2e-16
##
##           Sensitivity : 0.5219
##           Specificity : 0.9298
##           Pos Pred Value : 0.7093
##           Neg Pred Value : 0.8555
##           Prevalence : 0.2472
##           Detection Rate : 0.1290
##           Detection Prevalence : 0.1819
##           Balanced Accuracy : 0.7258
##
##           'Positive' Class : 1
##

```

```
# remove temp variables
rm(y, yhat, yhat_binary, mse_mlr, accuracy_mlr, pred_mlr)
```

model 2: logistic model -> Zian

```
# build full model
lr_model0 <- glm(income~. , data = train.data.2, family = binomial)
# summary(lr_model0)
# race, sex_Male, occupation_Craft-repair seemed to be not significant

# step-wise variable selection, both direction
suppressWarnings({
  lr_model <- stepAIC(lr_model0, k = log(nrow(train.data.2)), trace = 0, direction = "both")
})
summary(lr_model)
```

```
##
## Call:
## glm(formula = income ~ age + education.num + hours.per.week +
##     'workclass_Self-emp-not-inc' + 'marital.status_Married-civ-spouse' +
##     'marital.status_Never-married' + 'occupation_Craft-repair' +
##     'occupation_Exec-managerial' + 'occupation_Other-service' +
##     'occupation_Prof-specialty' + occupation_Sales + 'relationship_Own-child' +
##     relationship_Unmarried + capital.gain_1 + 'native.country_United-States',
##     family = binomial, data = train.data.2)
##
## Coefficients:
##
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -6.36559    0.14854 -42.856 < 2e-16 ***
## age                0.34803    0.02291  15.191 < 2e-16 ***
## education.num      0.31585    0.01008  31.333 < 2e-16 ***
## hours.per.week     0.33426    0.02086  16.024 < 2e-16 ***
## 'workclass_Self-emp-not-inc' -0.53817    0.06528  -8.244 < 2e-16 ***
## 'marital.status_Married-civ-spouse' 1.99706    0.06765  29.519 < 2e-16 ***
## 'marital.status_Never-married' -0.40956    0.08638  -4.741 2.13e-06 ***
## 'occupation_Craft-repair'      0.20050    0.05933   3.380 0.000726 ***
## 'occupation_Exec-managerial'    0.92491    0.05833  15.857 < 2e-16 ***
## 'occupation_Other-service'    -0.76667    0.11448  -6.697 2.12e-11 ***
## 'occupation_Prof-specialty'    0.59283    0.06476   9.154 < 2e-16 ***
## occupation_Sales      0.44105    0.06390   6.902 5.13e-12 ***
## 'relationship_Own-child'     -1.14661    0.15152  -7.567 3.81e-14 ***
## relationship_Unmarried    -0.46675    0.10433  -4.474 7.68e-06 ***
## capital.gain_1          1.54901    0.06218  24.910 < 2e-16 ***
## 'native.country_United-States' 0.31710    0.07620   4.161 3.16e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 27123  on 24146  degrees of freedom
## Residual deviance: 16931  on 24131  degrees of freedom
```

```

## AIC: 16963
##
## Number of Fisher Scoring iterations: 7

# predict test data, obtain confusion matrix
probs <- lr_model %>% predict(newdata = test.data.2, type = "response")
pred.income <- ifelse(probs > 0.5, 1, 0)

confusionMatrix(factor(pred.income), factor(test.data.2$income), positive = "1")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 4173  657
##           1  355  830
##
##           Accuracy : 0.8318
##           95% CI : (0.8221, 0.8411)
##       No Information Rate : 0.7528
##       P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.5149
##
##  McNemar's Test P-Value : < 2.2e-16
##
##           Sensitivity : 0.5582
##           Specificity : 0.9216
##           Pos Pred Value : 0.7004
##           Neg Pred Value : 0.8640
##           Prevalence : 0.2472
##           Detection Rate : 0.1380
##       Detection Prevalence : 0.1970
##           Balanced Accuracy : 0.7399
##
##           'Positive' Class : 1
##

y_true <- test.data.2$income
y_pred <- pred.income
y_prob <- probs

# AUC
roc_obj <- roc(y_true, y_prob)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

auc_val <- auc(roc_obj)
cat("AUC:", auc_val, "\n")

```

```
## AUC: 0.882659
```

```
# precision
precision <- posPredValue(factor(y_pred), factor(y_true), positive = "1")
cat("precision:", precision, "\n")
```

```
## precision: 0.7004219
```

```
# F1 score
recall <- sensitivity(factor(y_pred), factor(y_true), positive = "1")
cat("recall/sensitivity:", recall, "\n")
```

```
## recall/sensitivity: 0.5581708
```

```
f1_score <- 2 * precision * recall / (precision + recall)
cat("F1 Score:", f1_score, "\n")
```

```
## F1 Score: 0.6212575
```

```
rm(y_true, y_pred, y_prob, roc_obj, probs, pred.income)
```

model 3: NN model -> Tasfia

After fitting neural network model with reduced features, I found that full model without any feature selection has the highest accuracy.

```
# Find the best parameter for size and decay
grid_full <- expand.grid(size = c(5, 7, 9, 11), decay = c(0.0001, 0.001, 0.01, 0.05, 0.1, 0.2))

# convert training target to factor
train.data.2$income <- as.factor(train.data.2$income)

set.seed(123)
nn_model_full_tuned <- train(
  income ~ .,
  data = train.data.2,
  method = "nnet",
  tuneGrid = grid_full,
  trControl = trainControl(method = "cv", number = 10),
  maxit = 300,
  trace = FALSE
)

# View best parameters
nn_model_full_tuned$bestTune
```

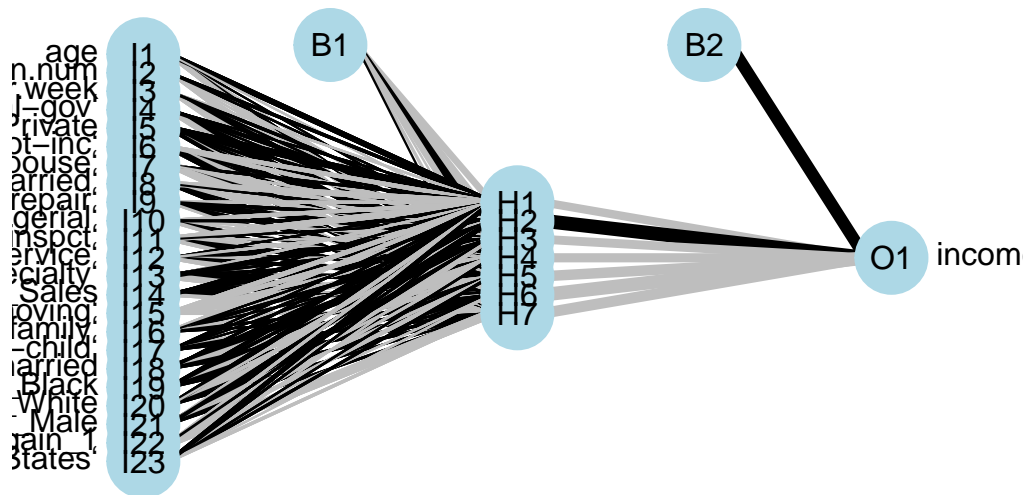
```
##   size decay
## 5     5   0.1
```

```
nn_model_full_tuned$results[order(-nn_model_full_tuned$results$Accuracy), ]
```

##	size	decay	Accuracy	Kappa	AccuracySD	KappaSD
## 5	5	1e-01	0.8401039	0.5515309	0.007331712	0.02198639
## 14	9	1e-03	0.8399796	0.5507479	0.009086261	0.02638011
## 10	7	5e-02	0.8396483	0.5505965	0.007574341	0.02140873
## 6	5	2e-01	0.8395654	0.5477418	0.008429799	0.02477449
## 18	9	2e-01	0.8393998	0.5491467	0.008633895	0.02367458
## 7	7	1e-04	0.8393173	0.5473520	0.006886058	0.02178971
## 12	7	2e-01	0.8390685	0.5477977	0.006885396	0.02008731
## 17	9	1e-01	0.8389855	0.5476737	0.008783059	0.02604875
## 9	7	1e-02	0.8389031	0.5490860	0.007381543	0.02062062
## 21	11	1e-02	0.8387373	0.5484209	0.007379578	0.02154746
## 4	5	5e-02	0.8384889	0.5445200	0.007242593	0.01991706
## 2	5	1e-03	0.8384470	0.5434987	0.009008495	0.02722371
## 3	5	1e-02	0.8384056	0.5443249	0.009194373	0.02751126
## 1	5	1e-04	0.8381159	0.5429089	0.008349358	0.02501060
## 13	9	1e-04	0.8380746	0.5451302	0.009450365	0.02630044
## 23	11	1e-01	0.8379918	0.5467605	0.008441624	0.02331797
## 16	9	5e-02	0.8374536	0.5440370	0.007410746	0.02230717
## 15	9	1e-02	0.8374118	0.5453188	0.007489830	0.02260251
## 24	11	2e-01	0.8370809	0.5426092	0.009258162	0.02617271
## 19	11	1e-04	0.8370808	0.5417726	0.006019846	0.01922713
## 20	11	1e-03	0.8364597	0.5416143	0.008541778	0.02437839
## 8	7	1e-03	0.8353002	0.5387094	0.007997153	0.01982071
## 22	11	5e-02	0.8293340	0.4928910	0.028635696	0.17448515
## 11	7	1e-01	0.8260241	0.5262863	0.028387333	0.03640006

```
# Neural Network Full Model (includes all predictors) from nnet package
train.data.2$income <- factor(train.data.2$income, levels = c("0", "1"))
test.data.2$income <- factor(test.data.2$income, levels = c("0", "1"))

# build model
set.seed(123)
nn_model_full <- nnet(
  income ~ .,
  data = train.data.2,
  size = 7,
  decay = 0.2,
  maxit = 300,
  trace = FALSE
)
# plot full model
plotnet(nn_model_full)
```



```
# predictions
nn_pred_full <- predict(nn_model_full, newdata = test.data.2, type = "class")
nn_pred_full <- factor(nn_pred_full, levels = c("0", "1"))

# confusion matrix
cm_full <- confusionMatrix(data = nn_pred_full, reference = test.data.2$income,
                             positive = "1")
cm_full
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 4163  600
##           1  365  887
##
##               Accuracy : 0.8396
##               95% CI : (0.83, 0.8488)
##           No Information Rate : 0.7528
##           P-Value [Acc > NIR] : < 2.2e-16
##
##               Kappa : 0.5448
##
##  Mcnemar's Test P-Value : 4.969e-14
##
##               Sensitivity : 0.5965
```

```
##           Specificity : 0.9194
##           Pos Pred Value : 0.7085
##           Neg Pred Value : 0.8740
##           Prevalence : 0.2472
##           Detection Rate : 0.1475
##           Detection Prevalence : 0.2081
##           Balanced Accuracy : 0.7579
##
##           'Positive' Class : 1
##
```

```
# Accuracy : 0.8396
# Sensitivity : 0.5965
# Specificity : 0.9194
```

```
# Neural Network from caret package
# set.seed(123)

# build model
# nn_model_full <- train(
#   income ~ .,
#   data = train.data.2,
#   method = "nnet",
#   tuneGrid = data.frame(size = 7, decay = 0.2),
#   maxit = 300,
#   trace = FALSE
# )

# predictions
# nn_pred_full <- predict(nn_model_full, newdata = test.data.2, type = "raw")
# nn_pred_full <- factor(nn_pred_full, levels = c("0", "1"))

# confusion matrix
# cm_full <- confusionMatrix(
#   data = nn_pred_full,
#   reference = test.data.2$income,
#   positive = "1"
# )
# cm_full

# Accuracy : 0.8394
# Sensitivity : 0.6005
# Specificity : 0.9178
```

model 4: CART model -> Nicole

```
# feature selection
cart_model <- rpart(income ~ ., data = train.data.1, control=rpart.control(cp=0), method = "class")
importance <- as.data.frame(cart_model$variable.importance)
print(importance)
```

```
##           cart_model$variable.importance
```

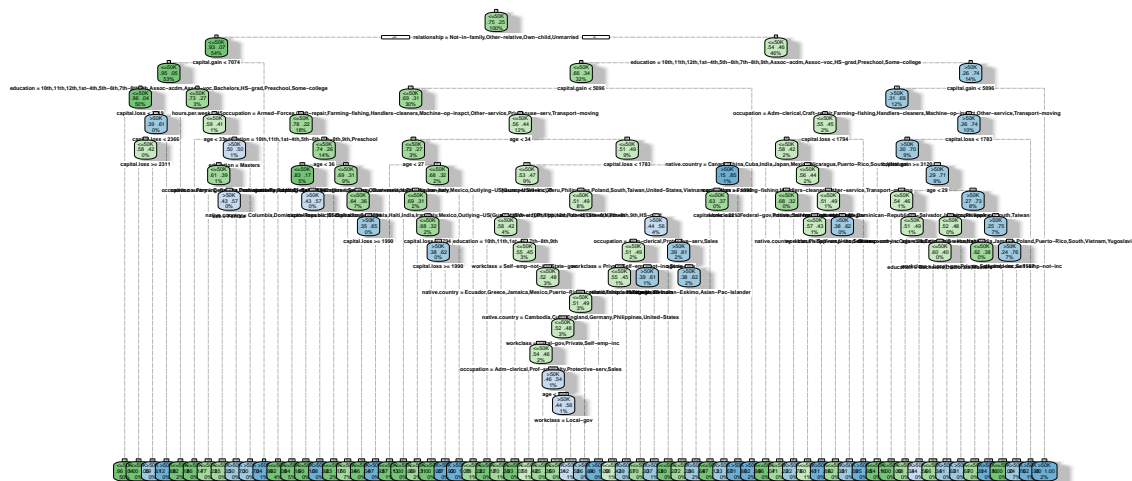
```
## relationship                1931.53200
## marital.status              1868.25395
## education                   1011.32541
## occupation                  938.38895
## capital.gain                906.68851
## age                        852.52286
## sex                         648.35841
## hours.per.week              499.97596
## capital.loss                258.30455
## workclass                   156.18319
## native.country              138.78359
## race                        38.63108
```

```
# prune
# printcp(modelCART)

best_cp <- cart_model$cptable[which.min(cart_model$cptable[, "xerror"]), "CP"]
pruned_model <- prune(cart_model, cp = best_cp)

fancyRpartPlot(pruned_model)
```

Warning: labs do not fit even at cex 0.15, there may be some overplotting




```

# helper function - compare model performances
comparemodels <- function(model1, model2, test_data, target_col) {
  preds1 <- predict(model1, test_data, type = "class")
  preds2 <- predict(model2, test_data, type = "class")

  actuals <- test_data[[target_col]]

  cm1 <- confusionMatrix(preds1, actuals, positive = levels(actuals)[2])
  cm2 <- confusionMatrix(preds2, actuals, positive = levels(actuals)[2])

  metrics <- data.frame(
    Model = c("Model 1", "Model 2"),
    Accuracy = c(cm1$overall["Accuracy"], cm2$overall["Accuracy"]),
    Sensitivity = c(cm1$byClass["Sensitivity"], cm2$byClass["Sensitivity"]),
    Specificity = c(cm1$byClass["Specificity"], cm2$byClass["Specificity"])
  )
  return(metrics)
}

comparemodels(cart_model, pruned_model, test.data.1, 'income')

```

```

##      Model  Accuracy Sensitivity Specificity
## 1 Model 1 0.8412303   0.6274378   0.9114399
## 2 Model 2 0.8543641   0.5938130   0.9399293

```

```

# pruned model is better, now lets remove unimportant variables
test.data.1a <- subset(test.data.1, select = -c(race, native.country))
train.data.1a <- subset(train.data.1, select = -c(race, native.country))

cart_model_a <- rpart(income ~ ., data = train.data.1a, control=rpart.control(cp=0), method = "class")
best_cp <- cart_model_a$cptable[which.min(cart_model_a$cptable[, "xerror"]), "CP"]
pruned_model_a <- prune(cart_model_a, cp = best_cp)
comparemodels(cart_model_a, pruned_model_a, test.data.1, 'income')

```

```

##      Model  Accuracy Sensitivity Specificity
## 1 Model 1 0.8433915   0.6314728   0.9129859
## 2 Model 2 0.8568579   0.5965030   0.9423587

```

```

# -----

preds <- predict(pruned_model_a, test.data.1, type = "class")
confusionMatrix(preds, test.data.1$income, positive = ">50K")

```

```

## Confusion Matrix and Statistics
##
##              Reference
## Prediction <=50K >50K
##      <=50K  4267  600
##      >50K   261  887
##
##              Accuracy : 0.8569
##              95% CI : (0.8478, 0.8656)

```

```
##      No Information Rate : 0.7528
##      P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.5835
##
##      McNemar's Test P-Value : < 2.2e-16
##
##              Sensitivity : 0.5965
##              Specificity : 0.9424
##              Pos Pred Value : 0.7726
##              Neg Pred Value : 0.8767
##              Prevalence : 0.2472
##              Detection Rate : 0.1475
##      Detection Prevalence : 0.1909
##      Balanced Accuracy : 0.7694
##
##      'Positive' Class : >50K
##
```

```
# -----
rm(test.data.1a, train.data.1a)
```

model 5: random forest model -> Tasfia

```
# Fitting Random Forest Model before feature selection:
set.seed(123)
rf_model <- randomForest(income ~ ., data = train.data.1, importance = TRUE, ntree = 500)

# Predictions on training data
test_preds <- predict(rf_model, newdata = test.data.1)

# Confusion matrix
cm <- confusionMatrix(test_preds, test.data.1$income, positive = ">50K")
cm
```

```
## Confusion Matrix and Statistics
##
##      Reference
## Prediction <=50K >50K
##      <=50K  4207  504
##      >50K    321  983
##
##              Accuracy : 0.8628
##              95% CI : (0.8539, 0.8714)
##      No Information Rate : 0.7528
##      P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.6156
##
##      McNemar's Test P-Value : 2.352e-10
##
```

```
##           Sensitivity : 0.6611
##           Specificity : 0.9291
##           Pos Pred Value : 0.7538
##           Neg Pred Value : 0.8930
##           Prevalence : 0.2472
##           Detection Rate : 0.1634
##           Detection Prevalence : 0.2168
##           Balanced Accuracy : 0.7951
##
##           'Positive' Class : >50K
##
```

Feature importance scores using random forest:

```
# Extract importance scores
importance_df <- as.data.frame(rf_model$importance)

# Add variable names as columns
importance_df$Variable <- rownames(importance_df)

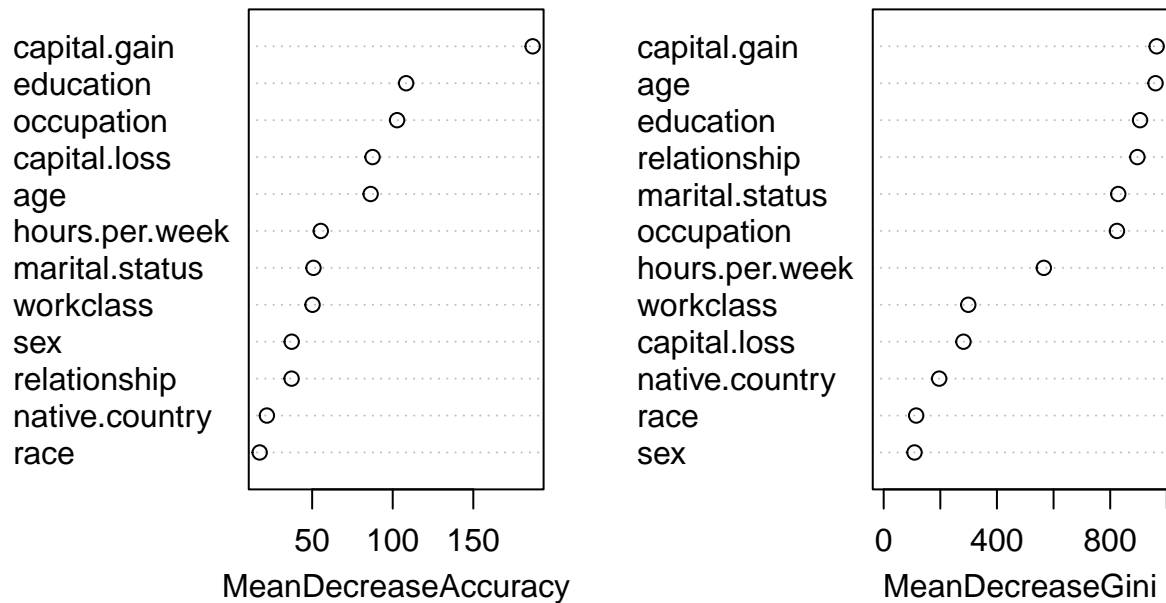
# Reorder columns
importance_df <- importance_df[, c("Variable", "MeanDecreaseAccuracy", "MeanDecreaseGini")]

# Sort by MeanDecreaseGini (most commonly used)
importance_df <- importance_df[order(-importance_df$MeanDecreaseGini), ]
importance_df
```

```
##           Variable MeanDecreaseAccuracy MeanDecreaseGini
## capital.gain      capital.gain          0.038796788      964.0763
## age               age              0.018801182      960.0870
## education         education         0.029168280      905.4104
## relationship      relationship      0.041906098      895.9086
## marital.status    marital.status    0.047587706      828.3922
## occupation        occupation        0.027308810      823.8177
## hours.per.week    hours.per.week    0.009858466      565.5229
## workclass         workclass         0.005912831      298.7715
## capital.loss      capital.loss      0.008538429      281.7051
## native.country    native.country    0.001699666      196.0093
## race              race              0.001136251      114.8469
## sex              sex              0.006968736      108.8781
```

```
# Plot variable importance
varImpPlot(rf_model,
  main = "Random Forest Variable Importance (All Features)")
```

Random Forest Variable Importance (All Features)



Random Forest with Reduced Features: -> don't need anymore

```
# train.rf.reduced <- subset(train.data.1, select = -c(sex))
#
# set.seed(123)
# rf_reduced <- randomForest(income ~ .,
#                             data = train.rf.reduced,
#                             ntree = 500,
#                             importance = TRUE)
#
# rf_reduced$importance
#
# # Plot variable importance
# varImpPlot(rf_reduced,
#             main = "Random Forest Variable Importance (Reduced Features)")
```

Confusion Matrix for RF w/ reduced features:

```
# test.rf.reduced <- subset(test.data.1, select = -c(sex))
#
# preds_reduced <- predict(rf_reduced, newdata = test.rf.reduced)
#
# cm <- confusionMatrix(preds_reduced, test.data.1$income, positive = ">50K")
# cm
#
# Accuracy : 0.862
```

```
# Sensitivity : 0.6604
# Specificity : 0.9282
```

model 6: extra model, XgBoost classification -> Thomas

```
# if testing this code chunk, run train.data.1 first

train.data.1$income <- factor(ifelse(train.data.1$income == ">50K", 1, 0))
test.data.1$income <- factor(ifelse(test.data.1$income == ">50K", 1, 0))

# Convert to Matrix
train.dummy <- model.matrix(income ~ . - 1, data = train.data.1)
test.dummy <- model.matrix(income ~ . - 1, data = test.data.1)

# Extract Label
train.label <- as.numeric(as.character(train.data.1$income))
test.label <- as.numeric(as.character(test.data.1$income))

# Convert to xgboost matrix format
dtrain <- xgb.DMatrix(data = train.dummy, label = train.label)
dtest <- xgb.DMatrix(data = test.dummy, label = test.label)

# Set XGBoost parameters for binary classification
params <- list(
  booster = "gbtree",
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8
)

# -----

# Train
set.seed(123)
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
  watchlist = list(train = dtrain, test = dtest),
  verbose = 0,
  early_stopping_rounds = 10
)

# -----

# Prediction
xgb_probs <- predict(xgb_model, dtest)
# Convert probabilities to binary
```

```
xgb_pred <- ifelse(xgb_probs > 0.5, 1, 0)
```

```
# Confusion Matrix
```

```
cm_xgb <- confusionMatrix(factor(xgb_pred), factor(test.label), positive = "1")
cm_xgb
```

Without Dealing with Imbalance

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 4275  537
##           1  253  950
##
##           Accuracy : 0.8687
##           95% CI : (0.8599, 0.8771)
##       No Information Rate : 0.7528
##       P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.6229
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##           Sensitivity : 0.6389
##           Specificity : 0.9441
##           Pos Pred Value : 0.7897
##           Neg Pred Value : 0.8884
##           Prevalence : 0.2472
##           Detection Rate : 0.1579
##       Detection Prevalence : 0.2000
##           Balanced Accuracy : 0.7915
##
##           'Positive' Class : 1
##
```

```
# check for data imbalance
table(train.data$income)
```

With Dealing with Imbalance

```
##
## <=50K  >50K
## 18126  6021
```

```
prop.table(table(train.data$income))
```

```
##
## <=50K  >50K
## 0.7506523 0.2493477
```

```

# somewhat imbalanced

### ***if model predicts income>50K poorly, try handle imbalance***

# if testing this code chunk, run train.data.1 first

# Set XGBoost parameters for binary classification

# Calculate the weight
neg <- sum(train.label == 0)
pos <- sum(train.label == 1)
scale_weight <- neg / pos

params <- list(
  booster = "gbtree",
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8,
  scale_pos_weight = scale_weight
)

# -----

# Train
set.seed(123)
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
  watchlist = list(train = dtrain, test = dtest),
  verbose = 0,
  early_stopping_rounds = 10
)

# -----

# Prediction
xgb_probs <- predict(xgb_model, dtest)
# Convert probabilities to binary
xgb_pred <- ifelse(xgb_probs > 0.5, 1, 0)

# Confusion Matrix
cm_xgb <- confusionMatrix(factor(xgb_pred), factor(test.label), positive = "1")
cm_xgb

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1

```

```
##          0 3681 212
##          1  847 1275
##
##          Accuracy : 0.8239
##          95% CI : (0.8141, 0.8335)
##    No Information Rate : 0.7528
##    P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 0.5863
##
## Mcnemar's Test P-Value : < 2.2e-16
##
##          Sensitivity : 0.8574
##          Specificity : 0.8129
##    Pos Pred Value : 0.6008
##    Neg Pred Value : 0.9455
##          Prevalence : 0.2472
##    Detection Rate : 0.2120
##    Detection Prevalence : 0.3528
##    Balanced Accuracy : 0.8352
##
##    'Positive' Class : 1
##
```

```
# Remove Temporary Variables
```

```
rm(
  train.dummy,
  test.dummy,
  train.label,
  test.label,
  dtrain,
  dtest,
  params,
  xgb_probs,
  xgb_pred,
  neg,
  pos,
  scale_weight
)
```