Bank Marketing Campaign Classification

Group 8
Marcus Chan Hua Rui
Huang Cheng
Catur Indra Pratisto

Abstract

Using dataset from Portuguese banking institution from May 2008 to November 2010, we applied various machine learning classification methods to predict the effectiveness of the bank direct campaigns (by phone calls) to get potential customers to subscribe to term deposit. By comparing Type-II error (False Negative Rate), we found that Random Forest yielded best prediction.

1 Introduction

Banks regularly conduct direct marketing campaigns to get the potential customers to subscribe to certain products. It is therefore important to analyse the dataset to predict the effectiveness of direct marketing campaign.

2 Objective

The classification goal is to predict if the direct marketing campaign is effective to get the potential customers will subscribe to a bank product.

For this project, the direct marketing campaign is done by phone calls and the bank product is term deposit.

3 Dataset

All the analysis is done in R, files saved as R Markdown (.Rmd).

This dataset is based on "Bank Marketing" UCI dataset 'bank-additional-full.csv' containing 41188 number of instances. There are 20 + output attribute, the binary classification goal is to predict if the client will subscribe a bank term deposit (variable y).

Detail descriptions of each variables can be found in Appendix A.

4 Methodologies

There are several machine learning techniques employed for comparison:

- K-nearest neighbours (KNN)
- Logistic Regression
- Decision tree
- Random Forest
- Gradient Boosting
- Adaptive Boosting (AdaBoost)
- Extreme Gradient Boosting (XGboost)
- Support Vector Machine (SVM) with linear kernel
- Support Vector Machine (SVM) with polynomial kernel
- Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel

5 Pre-processing and Exploratory Analysis

5.1 Pre-processing

This dataset comprises of a significant amount of missing data which was indicated by "unknown" values. For instance, from the 41188 data points in the "default" column, there were 8597 "unknown" values, 1731 in "education", 990 in "housing" and "loan", 330 in "job", and 80 in "marital". The details of how we deal with the missing values are shown in the next section.

We did further data pre-processing to deal with the missing values as well as to reduce the amount of noise in our dataset. As our dataset was hugely imbalanced with 88% of respondents saying "no" and only 12% saying "yes", we also did oversampling, which helped to reduce the type II error that we were interested in as it helped our model better generalise for the "yes" values we wanted to characterise.

5.2 Exploratory Analysis

In the "default" column, the fields filled with "unknown" values had 32588 "no", 8597 "unknown", and 3 "yes". Inside "default's "no's, there were 87.1% that eventually said no to the bank for the marketing campaign, which is lower than 94.8% in the respondents that had unknown values. As all the yes in default replied no to the marketing campaign, the unknown and yes values were re-coded to be the same in the dataset.

For education's unknown values, the behaviour of the unknown values was assessed, finding that they are most similar to that of university students, as can be seen from our code, and therefore recoded them to be university students as well.

The "housing" and "loan" columns were found to be statistically insignificant, from a chi-squared analysis and were dropped from the dataset to reduce noise in the dataset, which seemed to improve the results.

Unknown values in "job" was found to have distinct composition of result to the marketing campaign, and hence were re-labelled as unconventional.

From reading the description of some of the fields, we found that variables 'poutcome' and 'previous' seemed to be characterising similar information. Hence it seemed to be a good idea to merge this fields to output a single column. Failures were given a weight of 0.5, non-existent = 0.2, and success given 1. These weights were then used to multiply against "previous" +1. We added 1 as we did not want to multiple any values with 0. The 2 original columns were then dropped.

Age in this dataset was particularly interesting. We noticed that the proportion of people that said yes to the marketing campaign decreased with age, except that trend became significantly different after 60, where many began to reply yes and the trend with age seemed random. Re-categorised age into 7 values, for whichever age group the individual was in to reduce noise, and those over the age of 51 would be in one group. The histograms and other exploratory analysis visualisations can be found in Appendix B.

6 Findings

Because of the highly imbalance data (only 12% said Yes to term deposit), any classification models will classify label as 'No' so much more than 'Yes', therefore resulting in very good accuracy or low classification error rate.

However, looking more closely to the problem, Bank would not want to miscategorise potential customers that will say 'Yes' as 'No'. In other words, Bank are more likely to be concerned on reducing the number of **False Negative** – the number of customers identified as saying 'No' but in fact would have said 'Yes'.

Therefore, in order to choose the best prediction models, the False Negative Rate or type-II errors are compared instead of misclassification accuracy rate.

In addition to looking at type-II errors, we also want to know if we can reduce the **False Positive** – the number of customers identified as saying 'Yes' but in fact would have said 'No'.

Receiver Operating Characteristic (ROC) curves are created to illustrate both error types for all possible probability of prediction occurrence (threshold). The x-axis on ROC curve shows the **False Positive Rate** / type-I error and the y-axis shows the **True Positive Rate** / 1 – type II error (also called recall or sensitivity).

For binary classification like this project, the prediction is 'Yes' if the probability is more than 0.5 (threshold = 0.5). Since Bank might be more concerned of not missing customers that will say 'Yes', we can reduce the threshold from 0.5 to smaller number like 0.3 or 0.2.

As the threshold is reduced, the False Negative Rate/type-II error will be reduced (True Positive Rate in y-axis increases). However, at the same time, False Positive Rate / type-I error in x-axis will increase.

So which threshold is the best and should be chosen? Unfortunately, there is no easy answer to this.

Domain knowledge is required to choose the threshold. For example: Bank needs to calculate the cost of performing more marketing campaign for more potential customer and compare it with the actual profit for each actual term deposit signup.

In addition to showing the trade-off between type-I and type-II errors, ROC curves can also be used to indicate the overall performance of classification over all possible thresholds. This is given by Area Under ROC Curve or AUC.

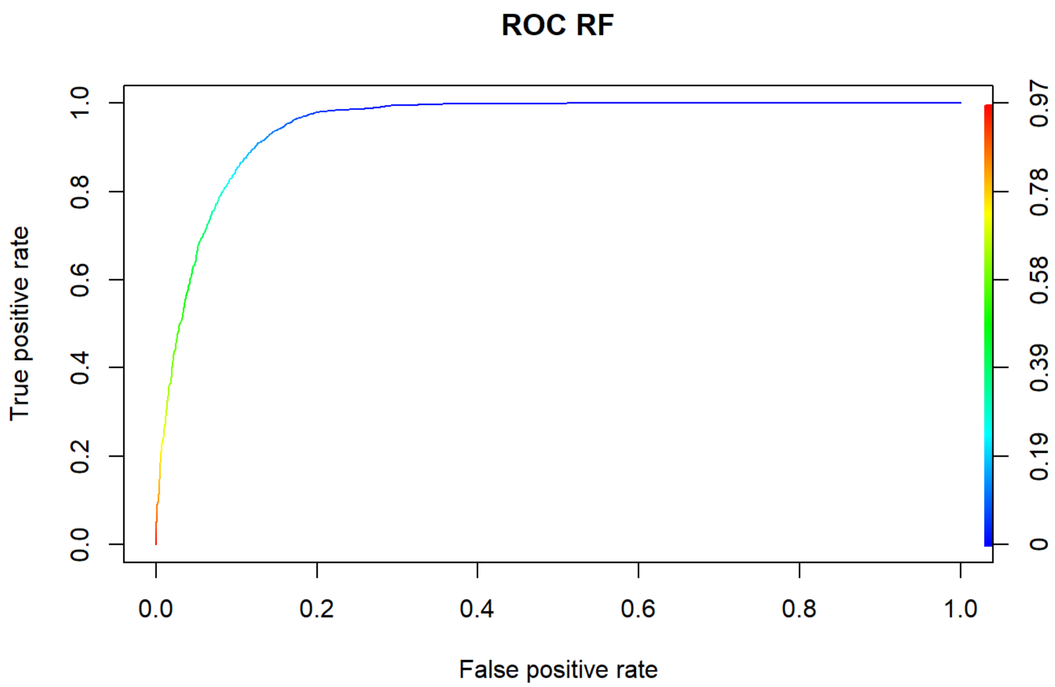
6.1 Classification Results

The following table summarizes classification results:

Table 6-1: Classification Result

	Misclass error	Type-II error	Precision	Recall
KNN	0.0920	0.5240	0.6255	0.4760
Logistic regression	0.0893	0.5827	0.6730	0.4173
Decision tree	0.0932	0.6531	0.6757	0.3469
Random forest	0.0827	**0.4482**	0.6645	0.5518
Gradient boosting	0.0819	0.4578	0.6737	0.5422
Adaboost	0.0895	0.4632	0.6241	0.5368
XGBoost	0.0816	0.4568	0.676	0.5432
SVM (linear kernel)	0.0972	0.6894	0.6525	0.3106
SVM (polynomial kernel)	0.0944	0.7375	0.7387	0.2625
SVM (RBF kernel)	0.0915	0.6553	0.6976	0.3447

Figure 6-1: Random Forest ROC Curve (**AUC = 0.951**)



From the table, Random Forest has the best classification result (lowest type-II error). Please find our R codes in Appendix C for further details.

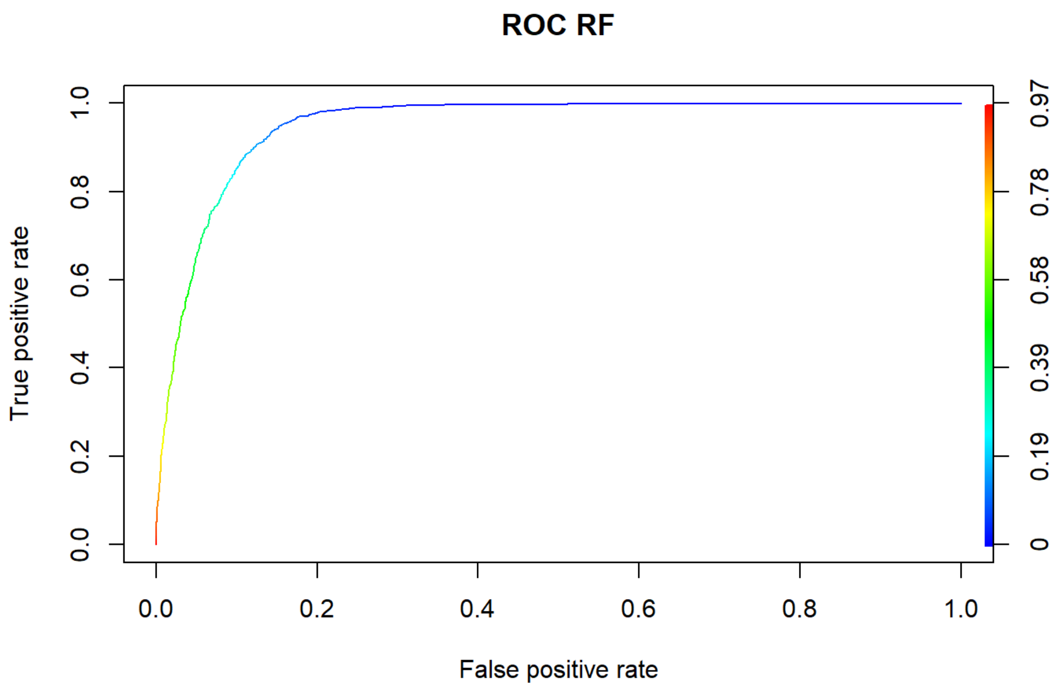
6.2 Classification with Scaled Numerical Variables Results

The following table summarizes classification results:

Table 6-2: Classification Results (Scaled Numerical Variables)

	Misclass error	Type-II error	Precision	Recall
KNN	0.0906	0.5560	0.6490	0.4440
Logistic regression	0.0893	0.5827	0.6730	0.4173
Decision tree	0.0932	0.6531	0.6757	0.3469
Random forest	0.0839	**0.4536**	0.6581	0.5464
Gradient boosting	0.0819	0.4578	0.6737	0.5422
Adaboost	0.0896	0.4642	0.6236	0.5358
XGBoost	0.0816	0.4568	0.6760	0.5432
SVM (linear kernel)	0.0972	0.6894	0.6525	0.3106
SVM (polynomial kernel)	0.0944	0.7375	0.7387	0.2625
SVM (RBF kernel)	0.0915	0.6553	0.6976	0.3447

Figure 6-2: Random Forest ROC Curve (**AUC = 0.951**)



There is slight improvement / reduction on type-II errors, but generally scaling the numerical variables do not have any significant effect to the classification results. Please find our R codes in Appendix D for further details.

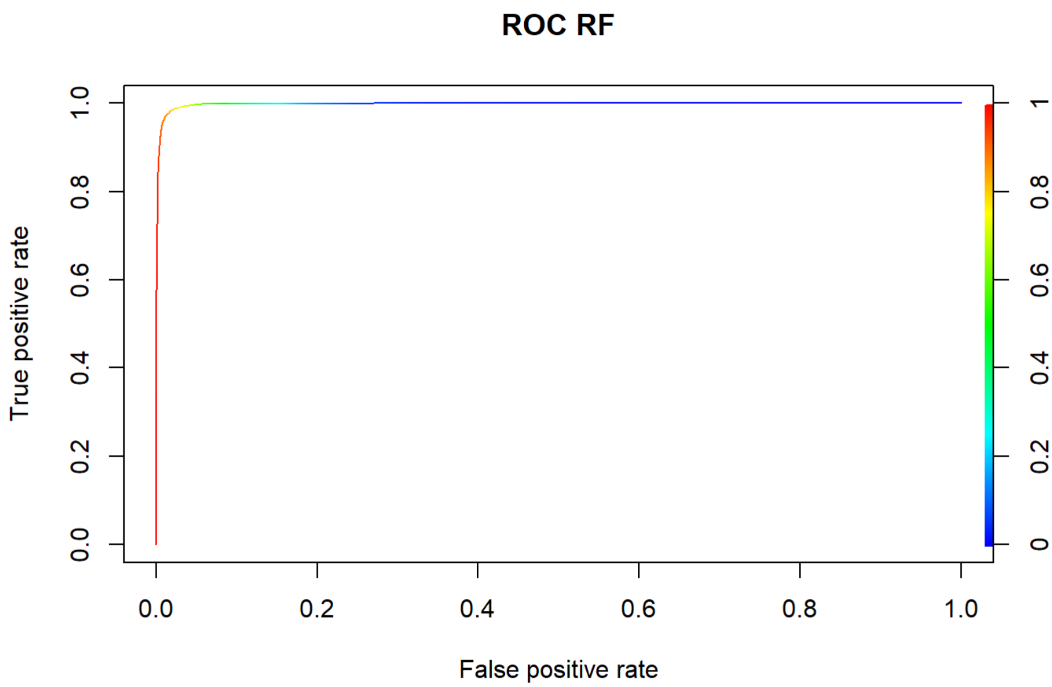
6.3 Classification with Oversampling Variables Results

The following table summarizes classification results:

Table 6-3: Classification Results (with Oversampling)

	Misclass error	**Type-II error**	**Precision**	**Recall**
KNN	0.0900	0.0106	0.8544	0.9894
Logistic regression	0.1249	0.1103	0.8654	0.8897
Decision tree	0.2064	0.0598	0.7281	0.9402
Random forest	0.0422	**0.0015**	0.9236	0.9985
Gradient boosting	0.1064	0.0655	0.8646	0.9345
Adaboost	0.0879	0.0489	0.8829	0.9511
XGBoost	0.0981	0.0504	0.8676	0.9496
SVM (linear kernel)	0.1187	0.0832	0.8569	0.9168
SVM (polynomial kernel)	0.1211	0.0891	0.8569	0.9109
SVM (RBF kernel)	0.1113	0.0598	0.8532	0.9402

Figure 6-3: Random Forest ROC Curve (**AUC = 0.998**)



Oversampling significantly reduces type-II error in Random Forest. Please find our R codes in Appendix E for further details.

6.4 Variables Importance

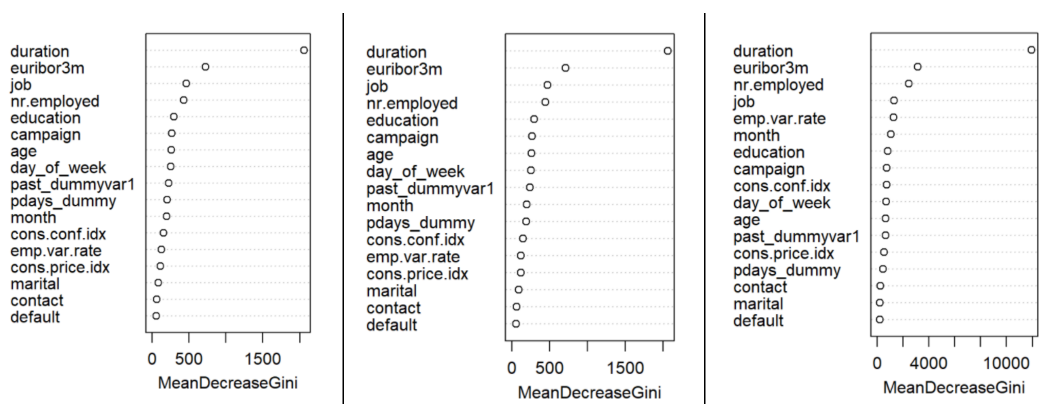


Figure 6-4: Random Forest Variables Importance

Duration call is self-explanatory: the longer the call being made, the more likely that potential customers will subscribe to term deposit.

Euribor is short for Euro Interbank Offered Rate, which is the average interest rates at which a large panel of European banks borrow funds from one another (equivalent to SIBOR – Singapore Interbank Offered Rate in Singapore). The Euribor rates are the reference rates in the European money market, to be used as basis for the price and interest rates of all kinds of financial products like interest rate swaps, interest rate futures, saving accounts and mortgages.

Therefore, it makes sense that more potential customer will subscribe to term deposit when the interest rate (based on Euribor rate) is higher.

7 Lesson Learnt

In order to assess the effectiveness of this direct marketing campaigns by phone calls of the banking institution, we have worked on the prediction on if the client will subscribe a term deposit (variable y) with the use of different models with and without scaled numerical variables and oversampling. In our evaluation of the features, we can say that the top feature is "Duration" due to its high correlation to Y. In real life situation, if duration of call is longer, it’s a strong indication of interest, thus higher chance of subscription. We have taken necessary steps for the data pre-processing and presented the results comparison and justify our selection of model. For all results we run out, we have shown results of Random forest has the higher ability of distinguishing with the maximum AUC of more than 0.95.

We have learnt ROC curve serves as a good performance measurement for classification problem at various threshold settings. ROC is a probability curve and AUC represent degree or measure of separability. Higher the AUC, better the model is at predicting YES as YES and NO as NO. By analogy, Higher the AUC, better the model is at distinguishing between customers who are willing or willing not to subscribe the term deposit. That is how we rationalise our selection of model.

We have also used Type I and Type II errors to analysis the model, the formula has been shown in the R codes. False Positive, or the Type I error means the client does not subscribe to term deposit, but the model thinks he/she does. False Negative, or the Type II error means the client subscribes to term deposit, but the model said he/she does not. In fact, for banks, false positive means the rate when banks think that they have the client but actually they have lost them. We will not think that this is something the bank is interested to know in the sense that banks are more revenue focusing. Then Type II error should be the one we are focusing on. We have actually predicted that the tree-based methods, especially Random Forest will have the lowest type-II errors since we have learnt in lectures that tree-based methods is the best for categorical variables, which we have quite a number of them for our dataset,

There are quite a number of difficulties and challenges in our project. One big challenge is that most features are categorical and there are quite a number of unknowns. Which we have attempted to solve the problem at the beginning. Another challenge we met is the hardware limitation to process large set of data, we have spent quite a long time to get the error results for certain models, e.g. KNN.

References

[1] S. Moro, P. Cortez and P. Rita (2014) A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

[2] https://towardsdatascience.com/machine-learning-case-study-a-data-driven-approach-to-predict-the-success-of-bank-telemarketing-20e37d46c31c.

[3] https://github.com/z-o-e/bank_data_analysis/blob/master/Linear_Models_Discriminants_Additive_Models_trees.R

[4] https://www.kaggle.com/janiobachmann/bank-marketing-campaign-opening-a-term-deposit

[5] https://www.kaggle.com/psqrtpsqrt/bank-marketing-eda-classification-pr-f-score#model-selection

Number of Instances: 41188 for bank-additional-full.csv

Number of Attributes: 20 + output attribute.

Input variables:

bank client data:

1 - age (numeric)

2 - job : type of job (categorical: "admin.","blue-collar","entrepreneur","housemaid","management","retired","self-employed","services","student","technician","unemployed","unknown")

3 - marital : marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed)

4 - education (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree", "unknown")

5 - default: has credit in default? (categorical: "no","yes","unknown")

6 - housing: has housing loan? (categorical: "no","yes","unknown")

7 - loan: has personal loan? (categorical: "no","yes","unknown")

related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: "cellular","telephone")

9 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")

10 - day_of_week: last contact day of the week (categorical: "mon","tue","wed","thu","fri")

11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")

social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)

17 - cons.price.idx: consumer price index - monthly indicator (numeric)

18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19 - euribor3m: euribor 3 month rate - daily indicator (numeric)

20 - nr.employed: number of employees - quarterly indicator (numeric)

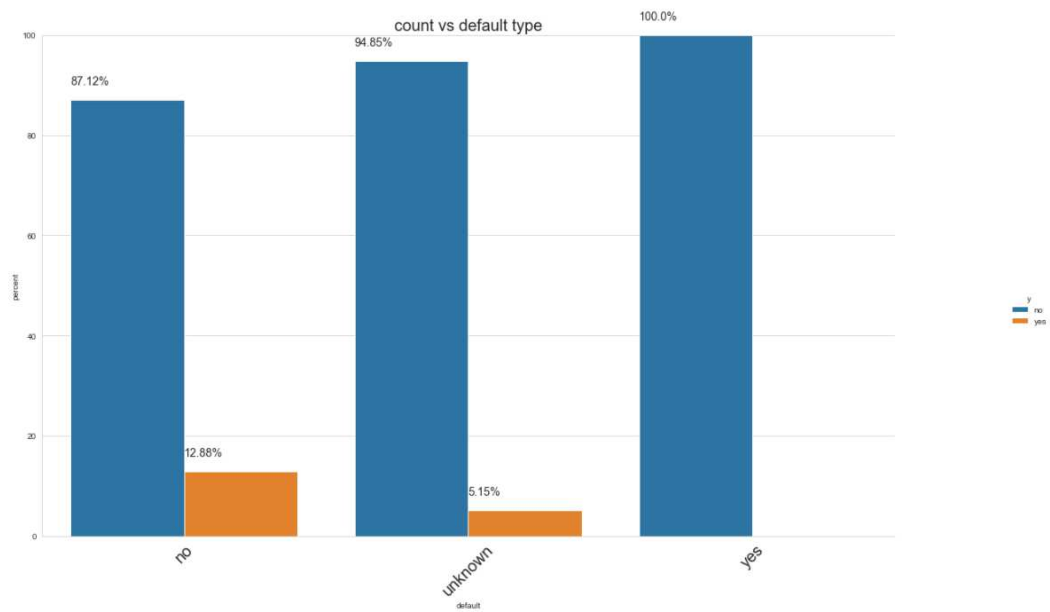
Output variable (desired target):

21 - y - has the client subscribed a term deposit? (binary: "yes","no")

Exploratory Data analysis

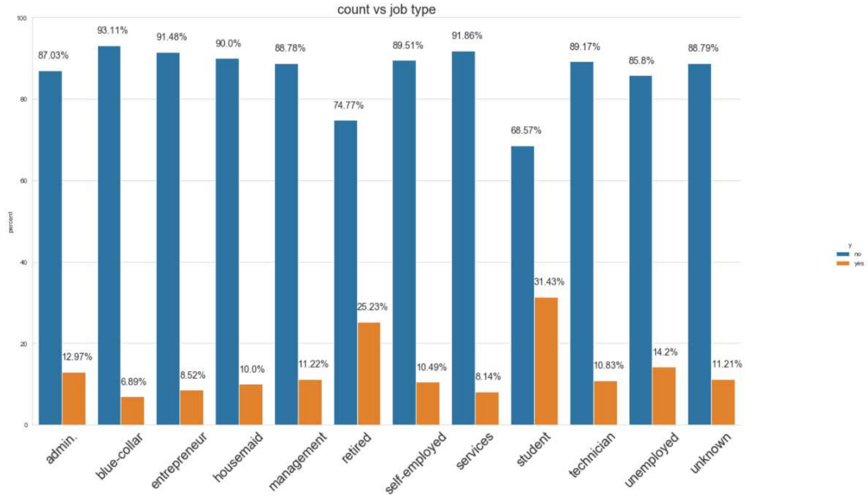
default

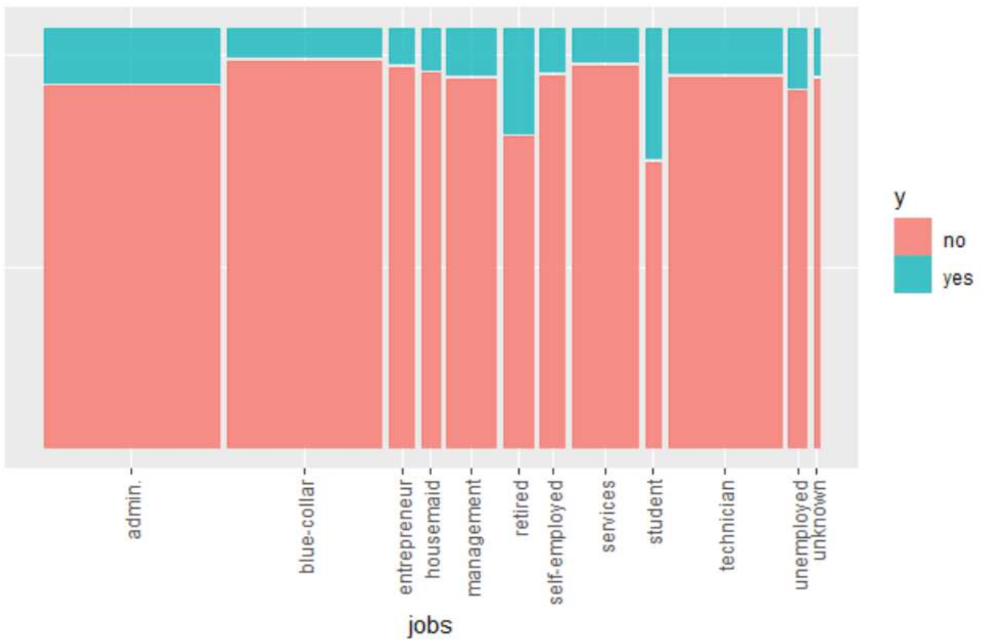
default	y: no	yes	Row Total
no	28391	4197	32588
	0.871	0.129	0.791
	0.689	0.102	
unknown	8154	443	8597
	0.948	0.052	0.209
	0.198	0.011	
yes	3	0	3
	1.000	0.000	0.000
	0.000	0.000	
Column Total	36548	4640	41188



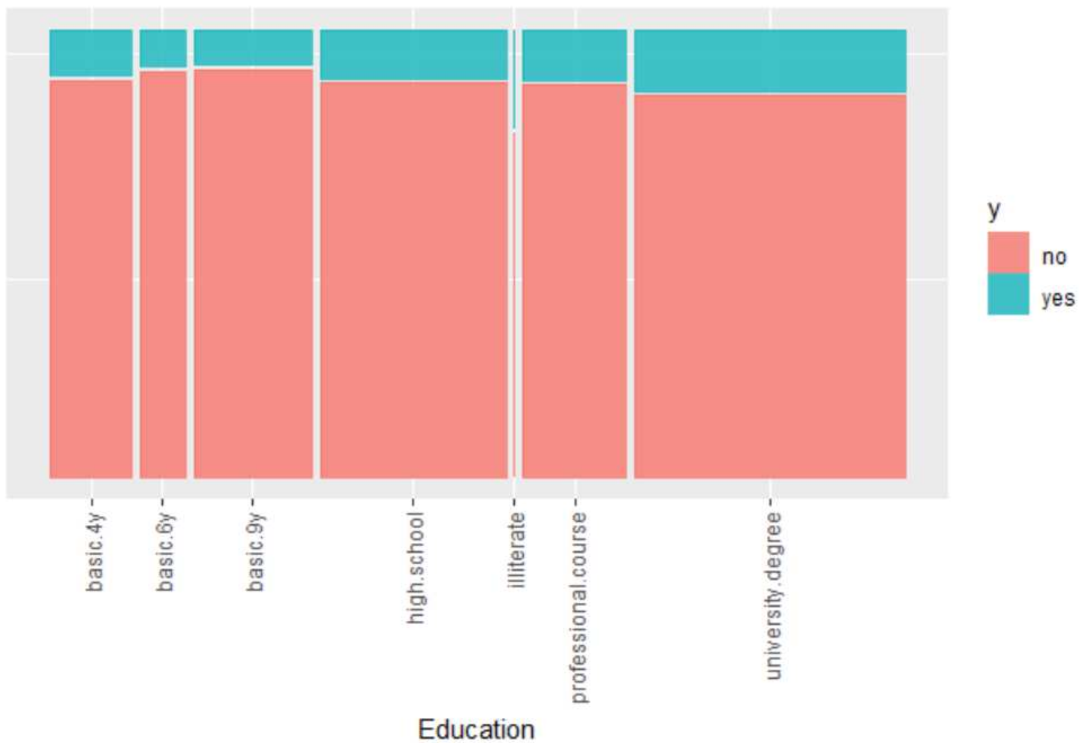
job

	y		
job	no	yes	Row Total
admin.	9070	1352	10422
	0.870	0.130	0.253
	0.220	0.033	
blue-collar	8616	638	9254
	0.931	0.069	0.225
	0.209	0.015	
entrepreneur	1332	124	1456
	0.915	0.085	0.035
	0.032	0.003	
housemaid	954	106	1060
	0.900	0.100	0.026
	0.023	0.003	
management	2596	328	2924
	0.888	0.112	0.071
	0.063	0.008	
retired	1286	434	1720
	0.748	0.252	0.042
	0.031	0.011	
self-employed	1272	149	1421
	0.895	0.105	0.035
	0.031	0.004	
services	3646	323	3969
	0.919	0.081	0.096
	0.089	0.008	
student	600	275	875
	0.686	0.314	0.021
	0.015	0.007	
technician	6013	730	6743
	0.892	0.108	0.164
	0.146	0.018	
unemployed	870	144	1014
	0.858	0.142	0.025
	0.021	0.003	
unknown	293	37	330
	0.888	0.112	0.008
	0.007	0.001	
Column Total	36548	4640	41188





education

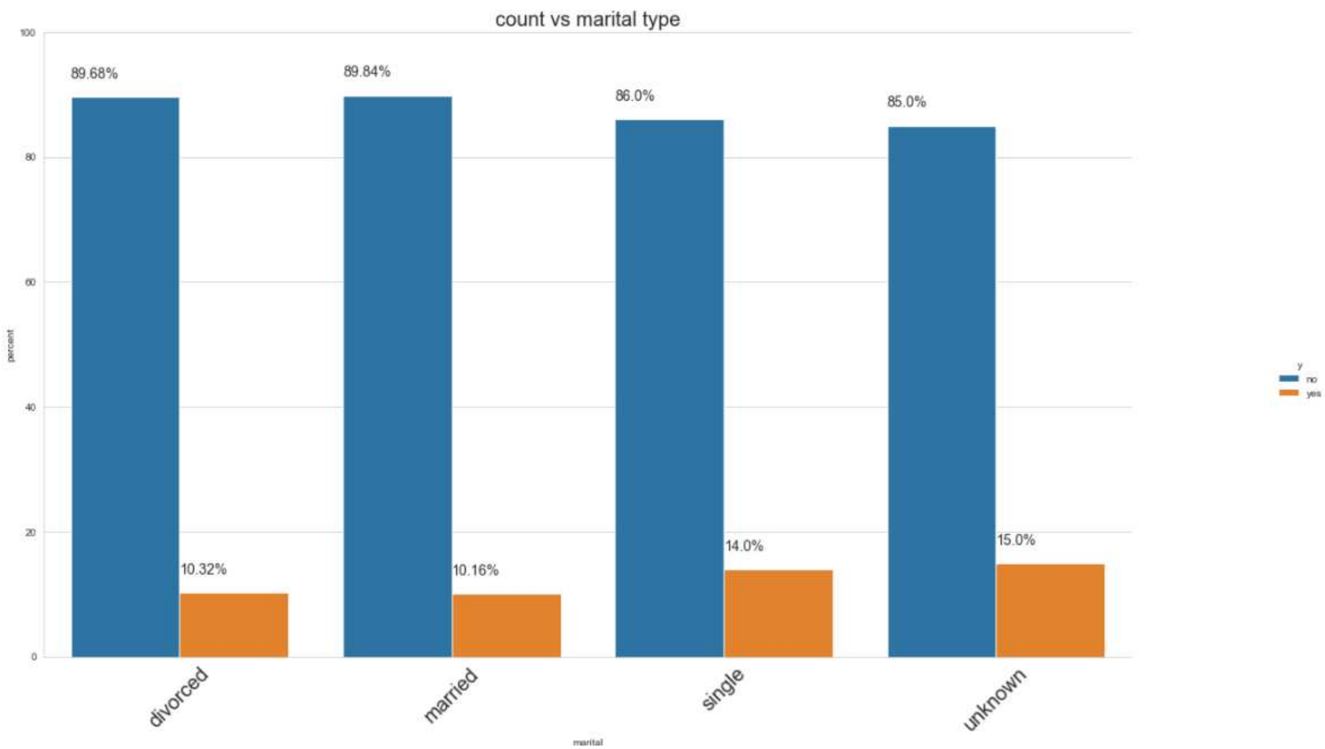


y
no
yes
basic.4y
basic.6y
basic.9y
high.school
illiterate
professional.course
university.degree
Education

Housing and loans

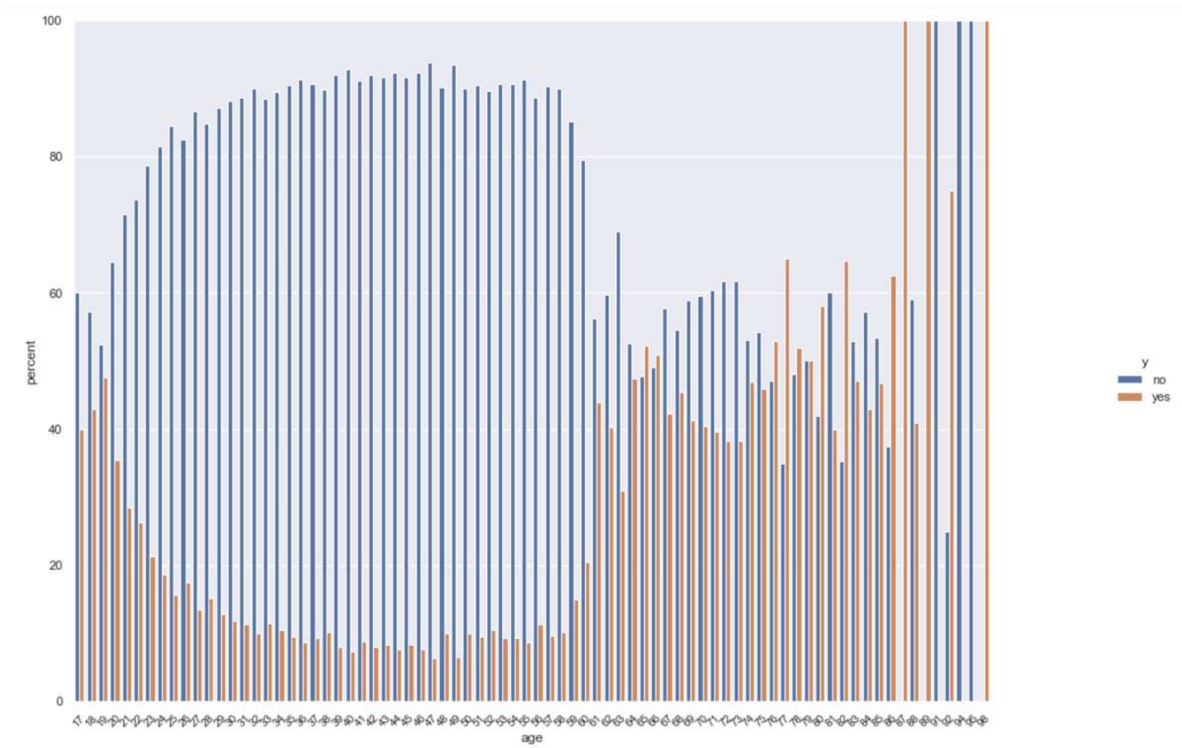
```
 Pearson's Chi-squared test

data:  bank_df$housing and bank_df$y
X-squared = 5.6845, df = 2, p-value = 0.05829


 Pearson's Chi-squared test

data:  bank_df$loan and bank_df$y
X-squared = 1.094, df = 2, p-value = 0.5787
```

marital

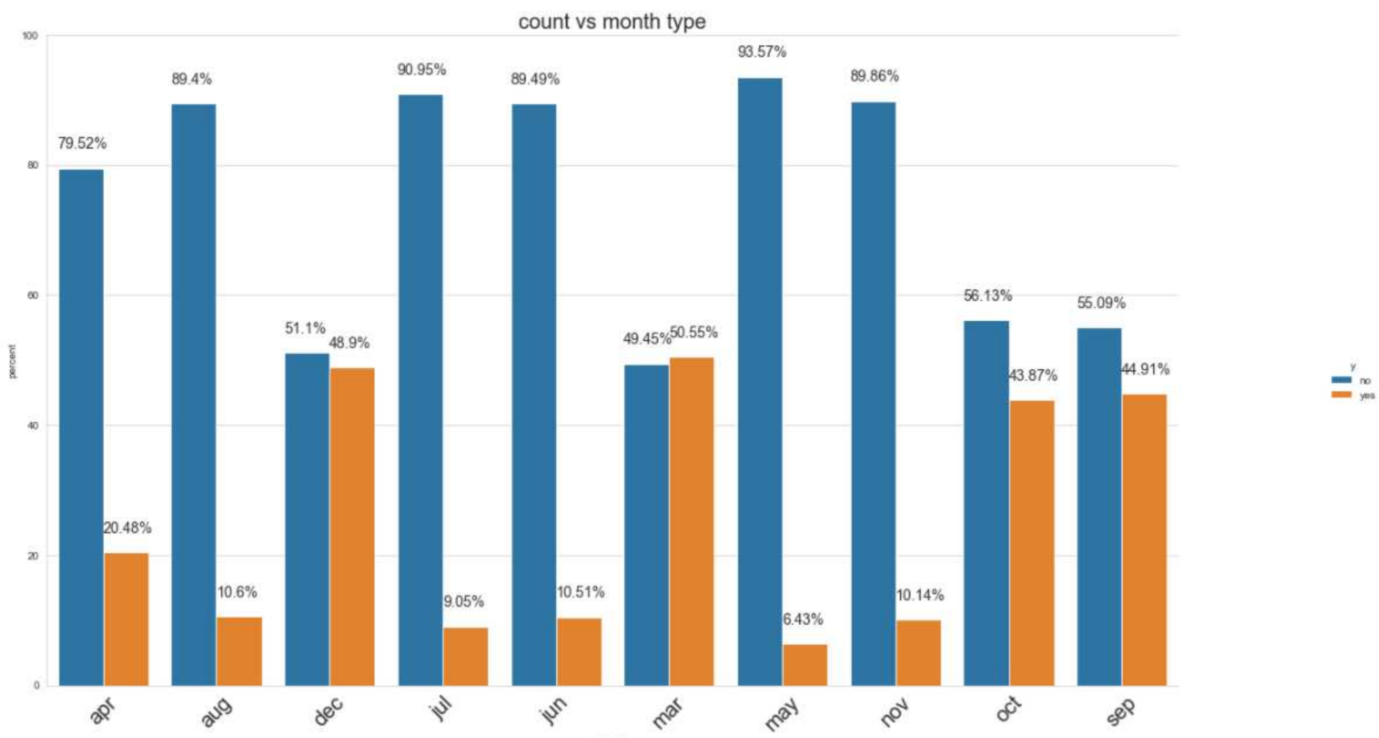


count vs marital type
100
80
60
40
20
0
percent
89.68%
89.84%
86.0%
85.0%
10.32%
10.16%
14.0%
15.0%
divorced
married
single
unknown
marital
y
no
yes

Age – percentage of yes and no with age

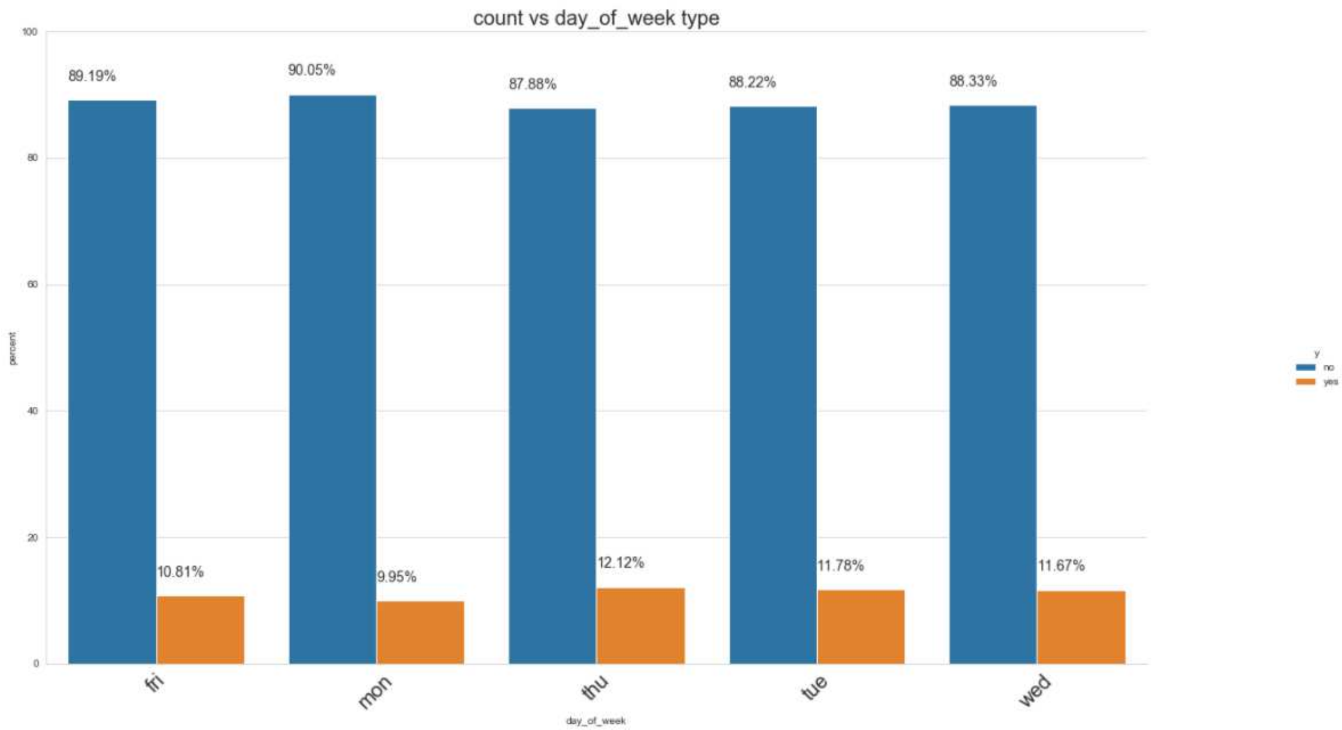


month



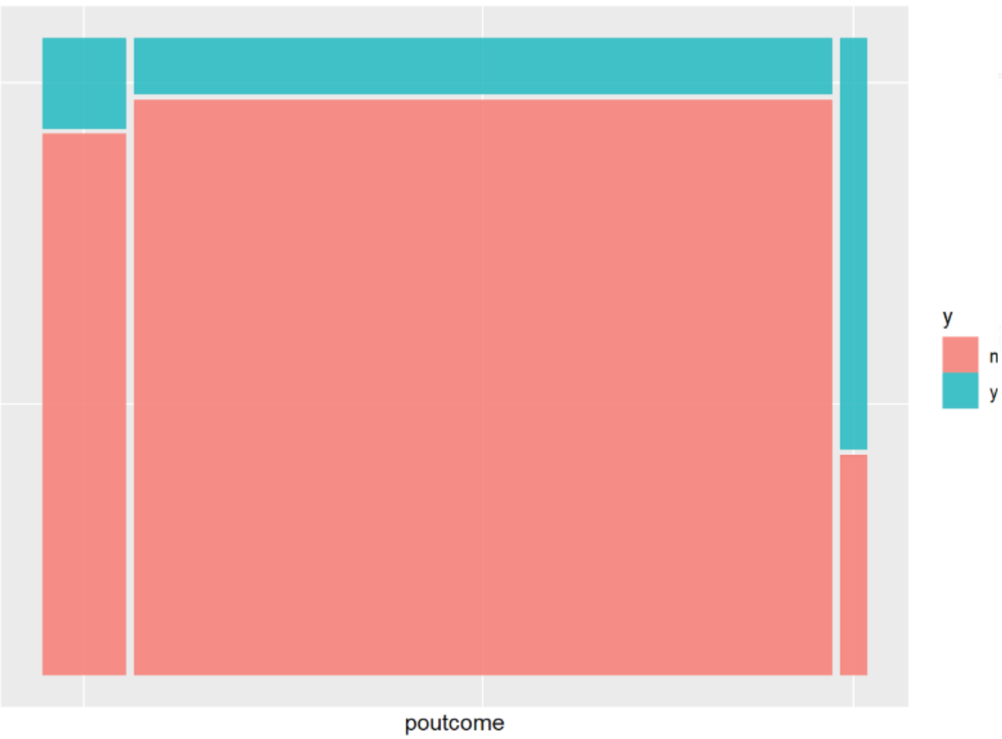
count vs month type
percent
month
79.52%
20.48%
89.4%
10.6%
51.1%
48.9%
90.95%
9.05%
89.49%
10.51%
49.45%
50.55%
93.57%
6.43%
89.86%
10.14%
56.13%
43.87%
55.09%
44.91%
apr
aug
dec
jul
jun
mar
may
nov
oct
sep
y
no
yes

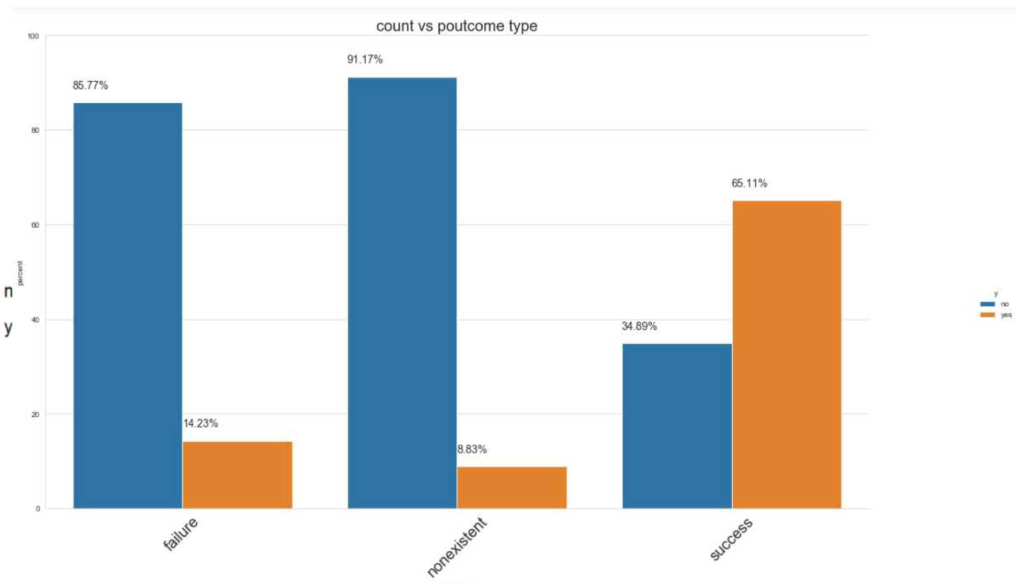
Day of week



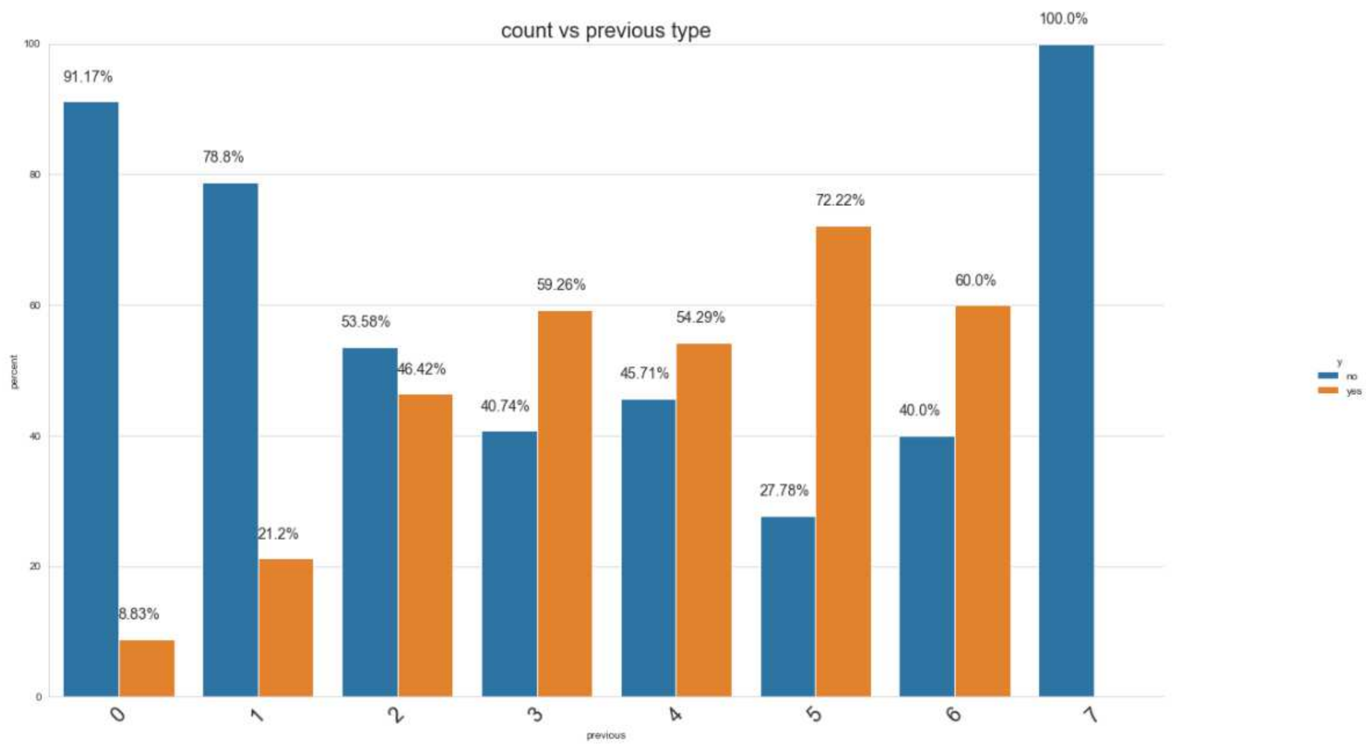
count vs day_of_week type
100
80
60
40
20
0
percent
89.19%
10.81%
90.05%
9.95%
87.88%
12.12%
88.22%
11.78%
88.33%
11.67%
fri
mon
thu
tue
wed
day_of_week
y
no
yes

Poutcome – success, failure, non-existent

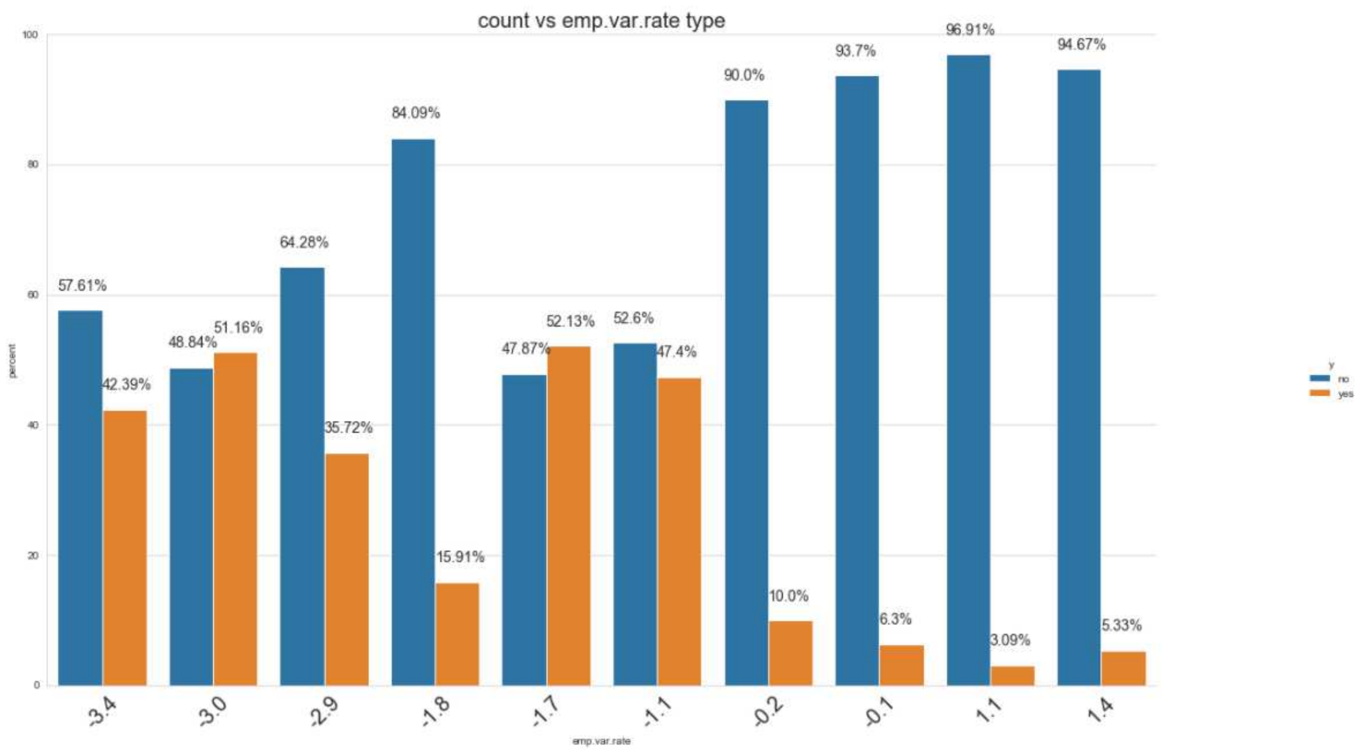




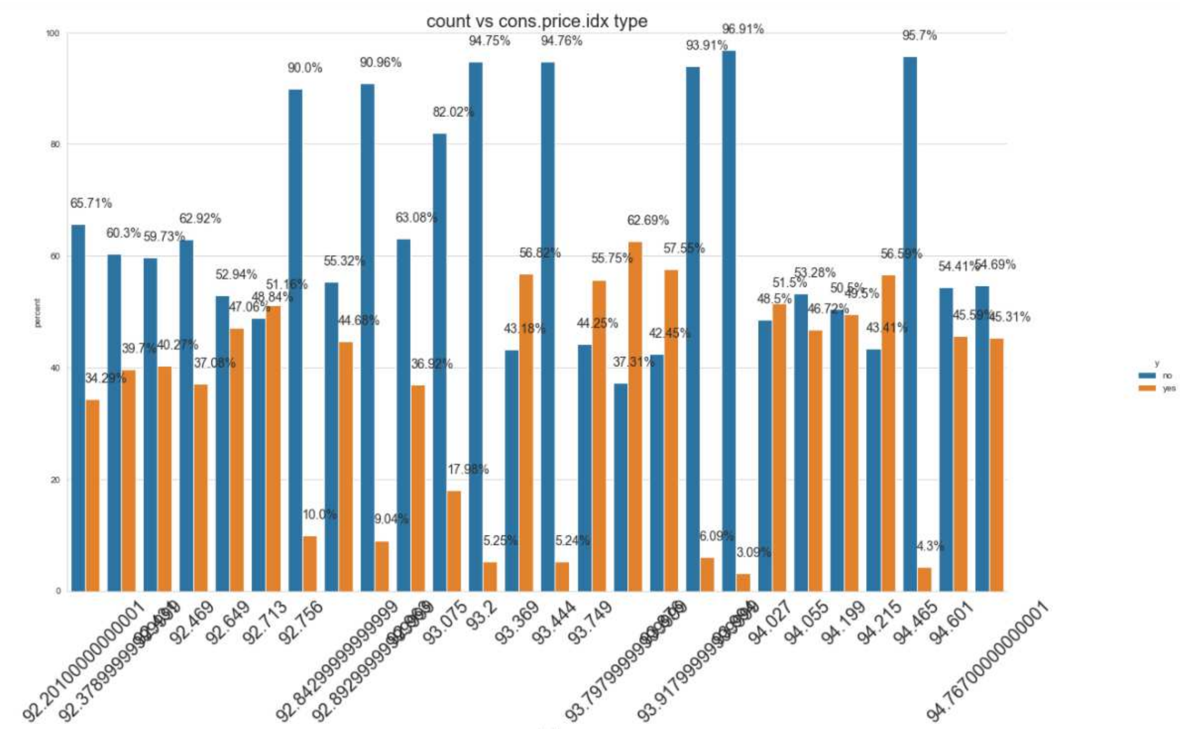
previous


count vs previous type
100.0%
91.17%
78.8%
72.22%
59.26%
60.0%
53.58%
54.29%
46.42%
45.71%
40.74%
40.0%
27.78%
21.2%
8.83%
percent
previous
0
1
2
3
4
5
6
7
100
80
60
40
20
0
y
no
yes

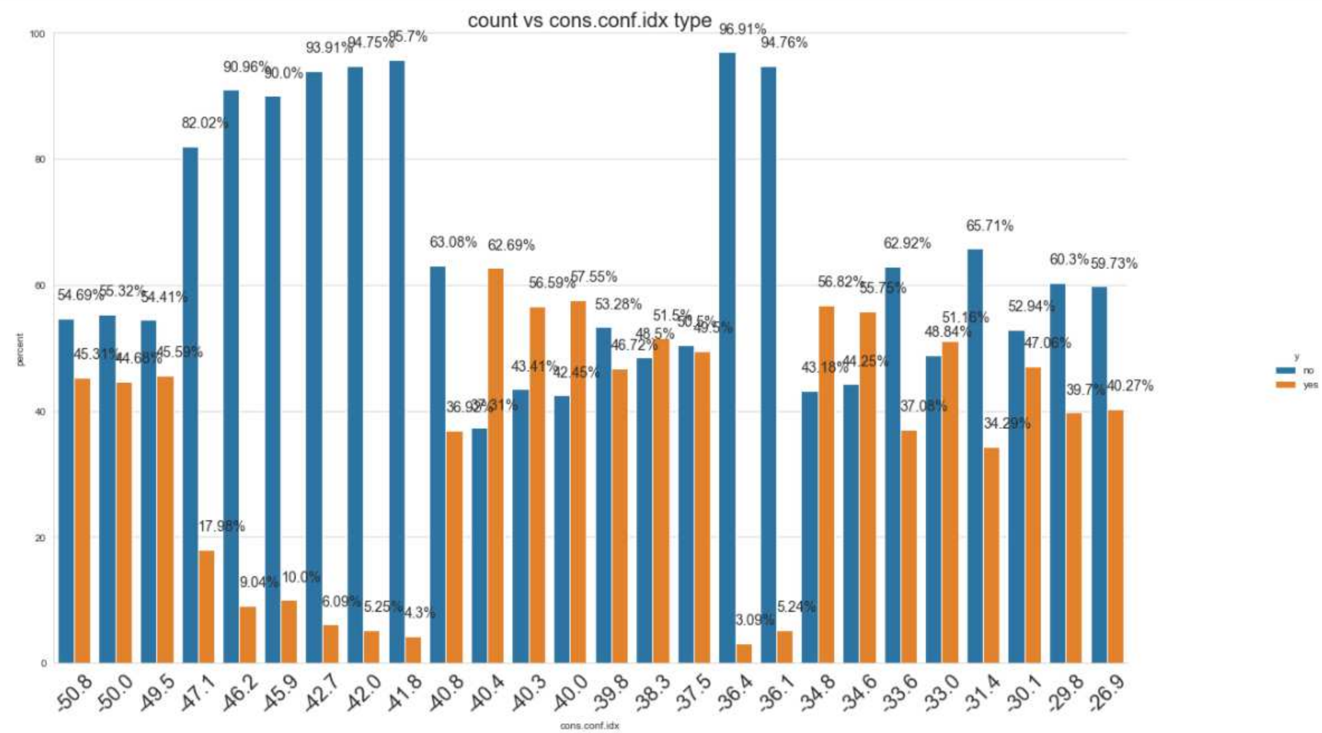
Emp.var.rate.type



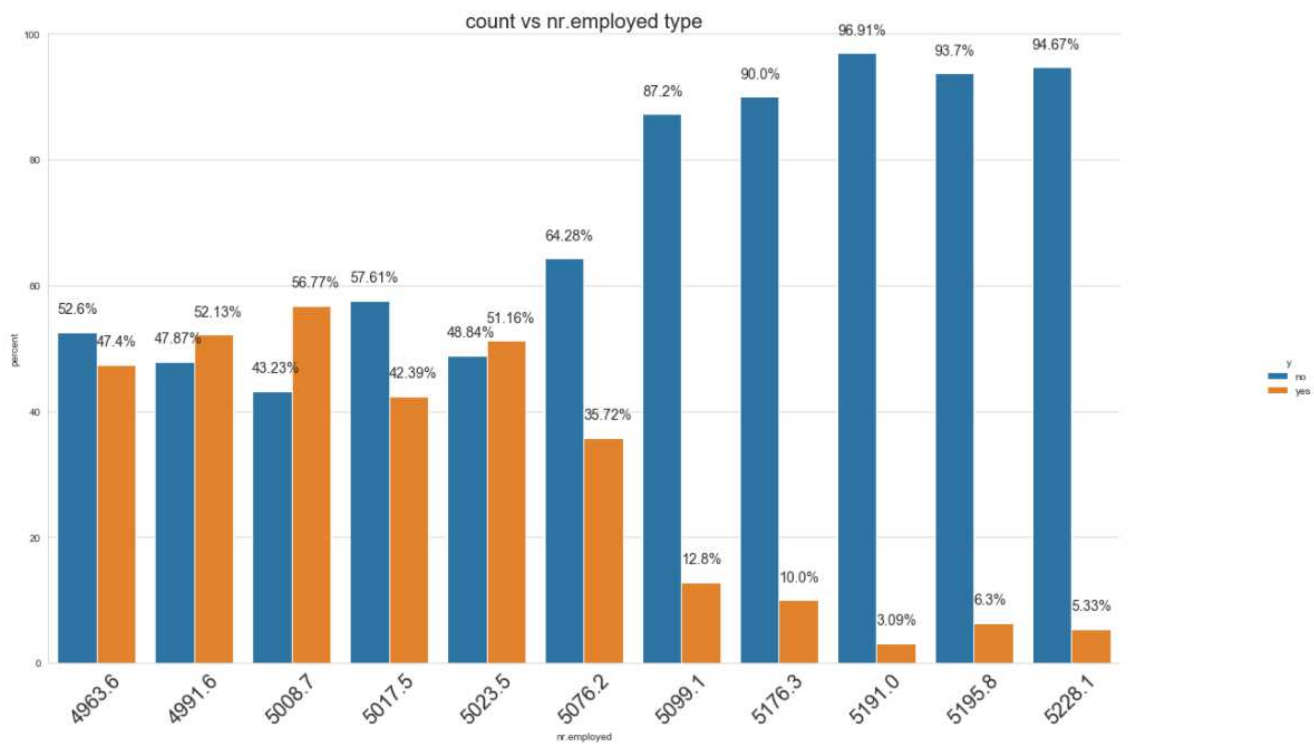
Cons.price.idx.type



Cons.conf.idx.type



Nr.employed.type



Bank Marketing Data

Group 8

Load Data

```
#Read dataset
bank_df <- read_delim("bank-additional-full.csv", delim=";")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   age = col_double(),
##   duration = col_double(),
##   campaign = col_double(),
##   pdays = col_double(),
##   previous = col_double(),
##   emp.var.rate = col_double(),
##   cons.price.idx = col_double(),
##   cons.conf.idx = col_double(),
##   euribor3m = col_double(),
##   nr.employed = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
#Assign category to all categorical variables

#2.job as category
bank_df$job <- as.factor(bank_df$job)

#3.marital status as category
bank_df$marital <- as.factor(bank_df$marital)

#4.education as category
bank_df$education <- as.factor(bank_df$education)

#5.credit default as category
bank_df$default <- as.factor(bank_df$default)

#6.housing loan as category
bank_df$housing <- as.factor(bank_df$housing)

#7.personal loan as category
bank_df$loan <- as.factor(bank_df$loan)

#8.contact communication type as category
bank_df$contact <- as.factor(bank_df$contact)

#9.last contact month of year as category
bank_df$month <- as.factor(bank_df$month)

#10.last contact day of the month as category
bank_df$day_of_week <- as.factor(bank_df$day_of_week)

#15.outcome of the previous marketing campaign as category
bank_df$poutcome <- as.factor(bank_df$poutcome)

#21.output y as binary factor
bank_df$y <- factor(bank_df$y, levels = c("no","yes"))

dim(bank_df)
```

```
## [1] 41188    21
```

Data preprocessing

```
bank_df %>%
  summarise_all(list(~sum(. == "unknown"))) %>%
  gather(key = "variable", value = "nr_unknown") %>%
  arrange(-nr_unknown)
```

```
## # A tibble: 21 x 2
##    variable    nr_unknown
##    <chr>            <int>
##  1 default           8597
##  2 education         1731
##  3 housing            990
##  4 loan               990
##  5 job                330
##  6 marital             80
##  7 age                  0
##  8 contact              0
##  9 month                0
## 10 day_of_week          0
## # ... with 11 more rows
```

```
# Analyse default
table(bank_df$default)
```

```
##
##      no unknown     yes
##   32588    8597       3
```

```
## This is not usable, too few "yes" to evaluate
```

analyse the unknown values

```
# setting default parameters for crosstables
# fun_crosstable = function(df, var1, var2){
#   # df: dataframe containing both columns to cross
#   # var1, var2: columns to cross together.
#   CrossTable(df$var1, df$var2,
#              prop.r = T,
#              prop.c = F,
#              prop.t = F,
#              prop.chisq = F,
#              dnn = c(var1, var2)) # dimension names
# }

#default
CrossTable(bank_df$default, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("default",
"y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      default |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##           no |     28391 |      4197 |     32588 |
##              |     0.871 |     0.129 |     0.791 |
##              |     0.689 |     0.102 |           |
## -------------|-----------|-----------|-----------|
##      unknown |      8154 |       443 |      8597 |
##              |     0.948 |     0.052 |     0.209 |
##              |     0.198 |     0.011 |           |
## -------------|-----------|-----------|-----------|
##          yes |         3 |         0 |         3 |
##              |     1.000 |     0.000 |     0.000 |
##              |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

```
table(bank_df$default)
```

```
##
##      no unknown     yes
##   32588    8597       3
```

```
# job
CrossTable(bank_df$job, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("job", "y"))
```

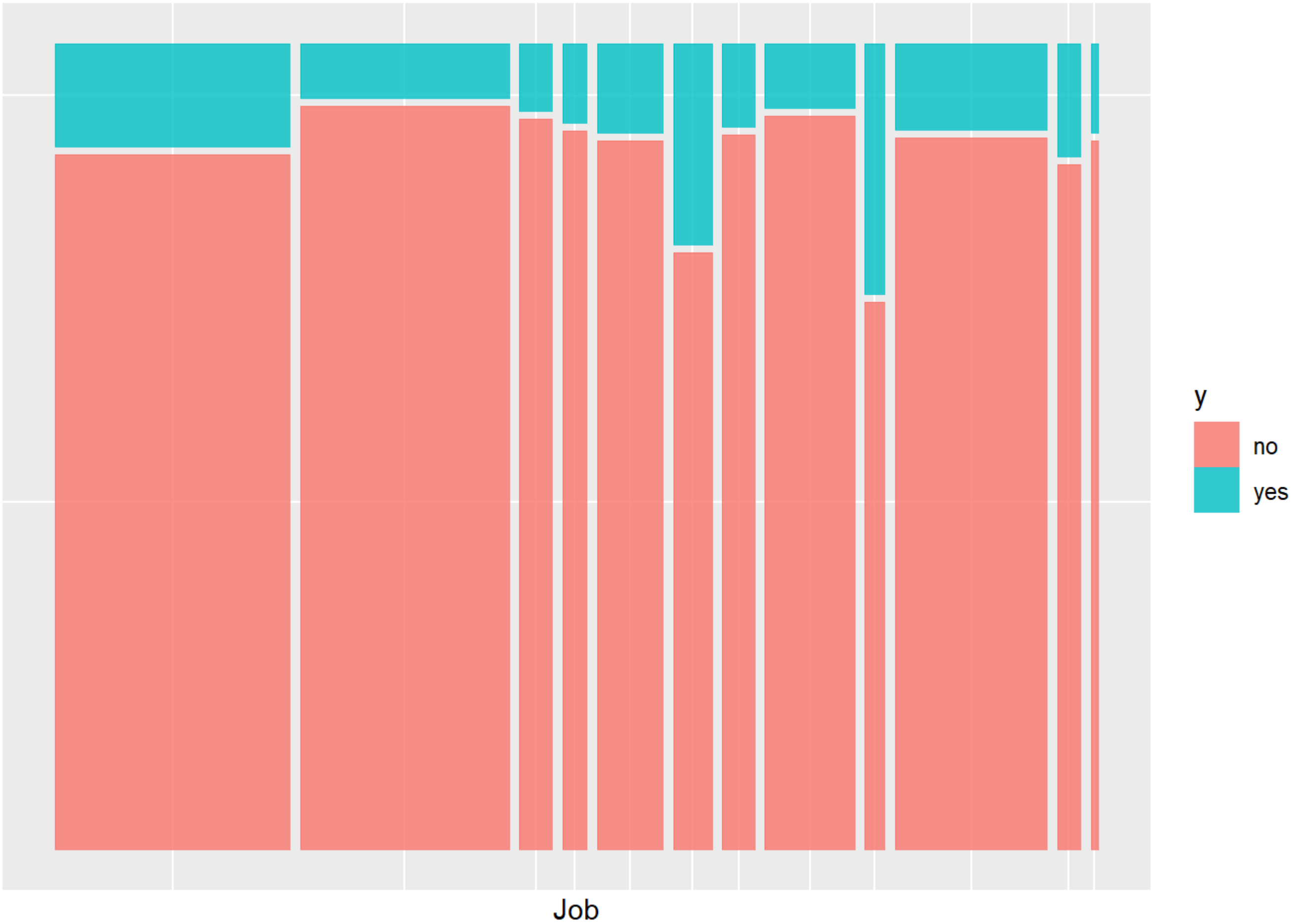
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##               | y
##           job |        no |       yes | Row Total |
## --------------|-----------|-----------|-----------|
##        admin. |      9070 |      1352 |     10422 |
##               |     0.870 |     0.130 |     0.253 |
##               |     0.220 |     0.033 |           |
## --------------|-----------|-----------|-----------|
##   blue-collar |      8616 |       638 |      9254 |
##               |     0.931 |     0.069 |     0.225 |
##               |     0.209 |     0.015 |           |
## --------------|-----------|-----------|-----------|
##  entrepreneur |      1332 |       124 |      1456 |
##               |     0.915 |     0.085 |     0.035 |
##               |     0.032 |     0.003 |           |
## --------------|-----------|-----------|-----------|
##     housemaid |       954 |       106 |      1060 |
##               |     0.900 |     0.100 |     0.026 |
##               |     0.023 |     0.003 |           |
## --------------|-----------|-----------|-----------|
##    management |      2596 |       328 |      2924 |
##               |     0.888 |     0.112 |     0.071 |
##               |     0.063 |     0.008 |           |
## --------------|-----------|-----------|-----------|
##       retired |      1286 |       434 |      1720 |
##               |     0.748 |     0.252 |     0.042 |
##               |     0.031 |     0.011 |           |
## --------------|-----------|-----------|-----------|
## self-employed |      1272 |       149 |      1421 |
##               |     0.895 |     0.105 |     0.035 |
##               |     0.031 |     0.004 |           |
## --------------|-----------|-----------|-----------|
##      services |      3646 |       323 |      3969 |
##               |     0.919 |     0.081 |     0.096 |
##               |     0.089 |     0.008 |           |
## --------------|-----------|-----------|-----------|
##       student |       600 |       275 |       875 |
##               |     0.686 |     0.314 |     0.021 |
##               |     0.015 |     0.007 |           |
## --------------|-----------|-----------|-----------|
##    technician |      6013 |       730 |      6743 |
```

```
##              |     0.892 |     0.108 |     0.164 |
##              |     0.146 |     0.018 |           |
## -------------|-----------|-----------|-----------|
##   unemployed |       870 |       144 |      1014 |
##              |     0.858 |     0.142 |     0.025 |
##              |     0.021 |     0.003 |           |
## -------------|-----------|-----------|-----------|
##      unknown |       293 |        37 |       330 |
##              |     0.888 |     0.112 |     0.008 |
##              |     0.007 |     0.001 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

```
table(bank_df$job)
```

```
##
##        admin.   blue-collar  entrepreneur     housemaid    management
##         10422          9254          1456          1060          2924
##       retired self-employed      services       student    technician
##          1720          1421          3969           875          6743
##    unemployed       unknown
##          1014           330
```

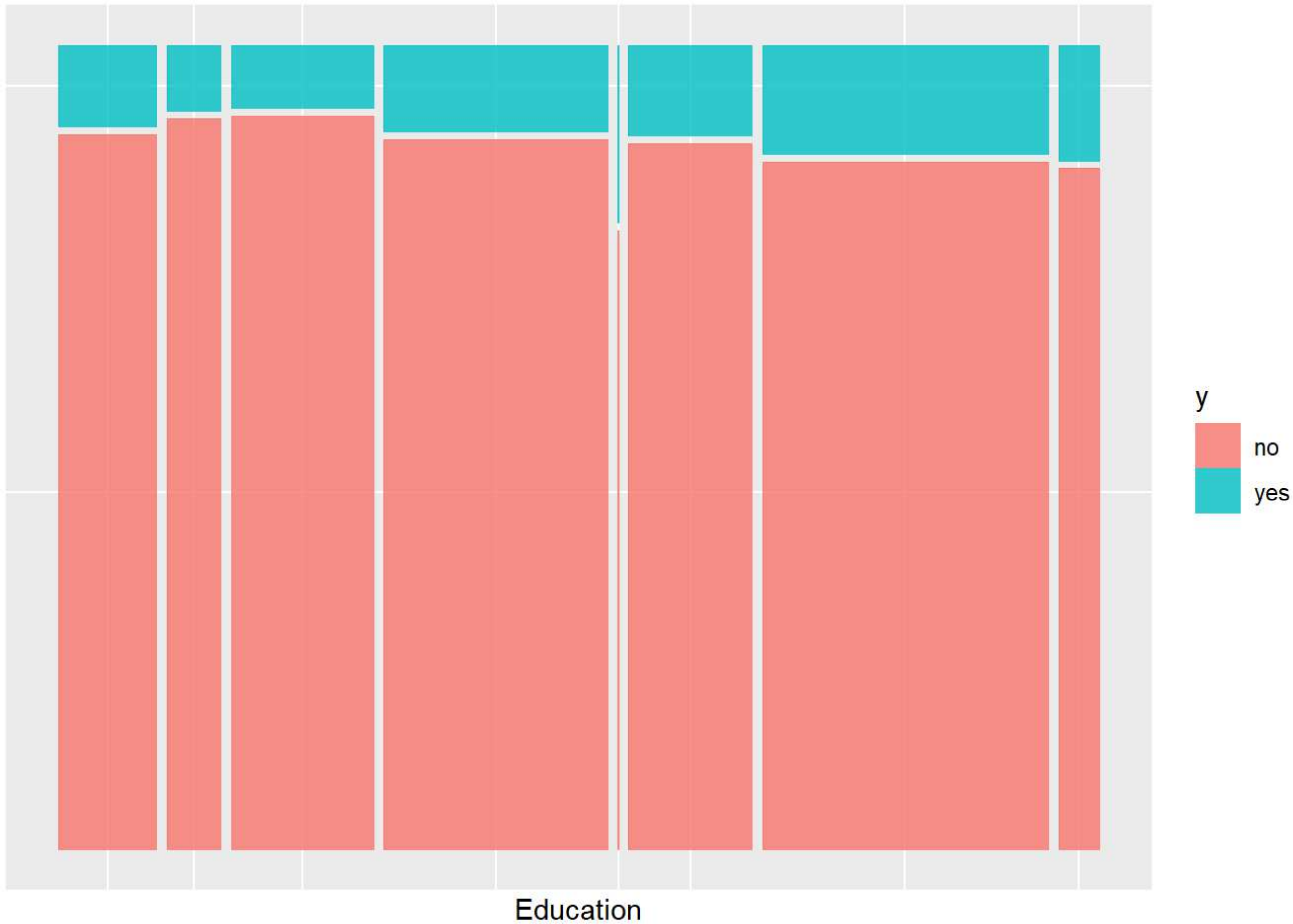
```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, job), fill = y)) +
  #mosaic_theme +
  xlab("Job") +
  ylab(NULL)
```




```
bank_df <- bank_df %>%
  mutate(job = recode(job, "unknown" = "unconventional"))


# marital
CrossTable(bank_df$marital, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("marital",
"y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      marital |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##     divorced |      4136 |       476 |      4612 |
##              |     0.897 |     0.103 |     0.112 |
##              |     0.100 |     0.012 |           |
## -------------|-----------|-----------|-----------|
##      married |     22396 |      2532 |     24928 |
##              |     0.898 |     0.102 |     0.605 |
##              |     0.544 |     0.061 |           |
## -------------|-----------|-----------|-----------|
##       single |      9948 |      1620 |     11568 |
##              |     0.860 |     0.140 |     0.281 |
##              |     0.242 |     0.039 |           |
## -------------|-----------|-----------|-----------|
##      unknown |        68 |        12 |        80 |
##              |     0.850 |     0.150 |     0.002 |
##              |     0.002 |     0.000 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

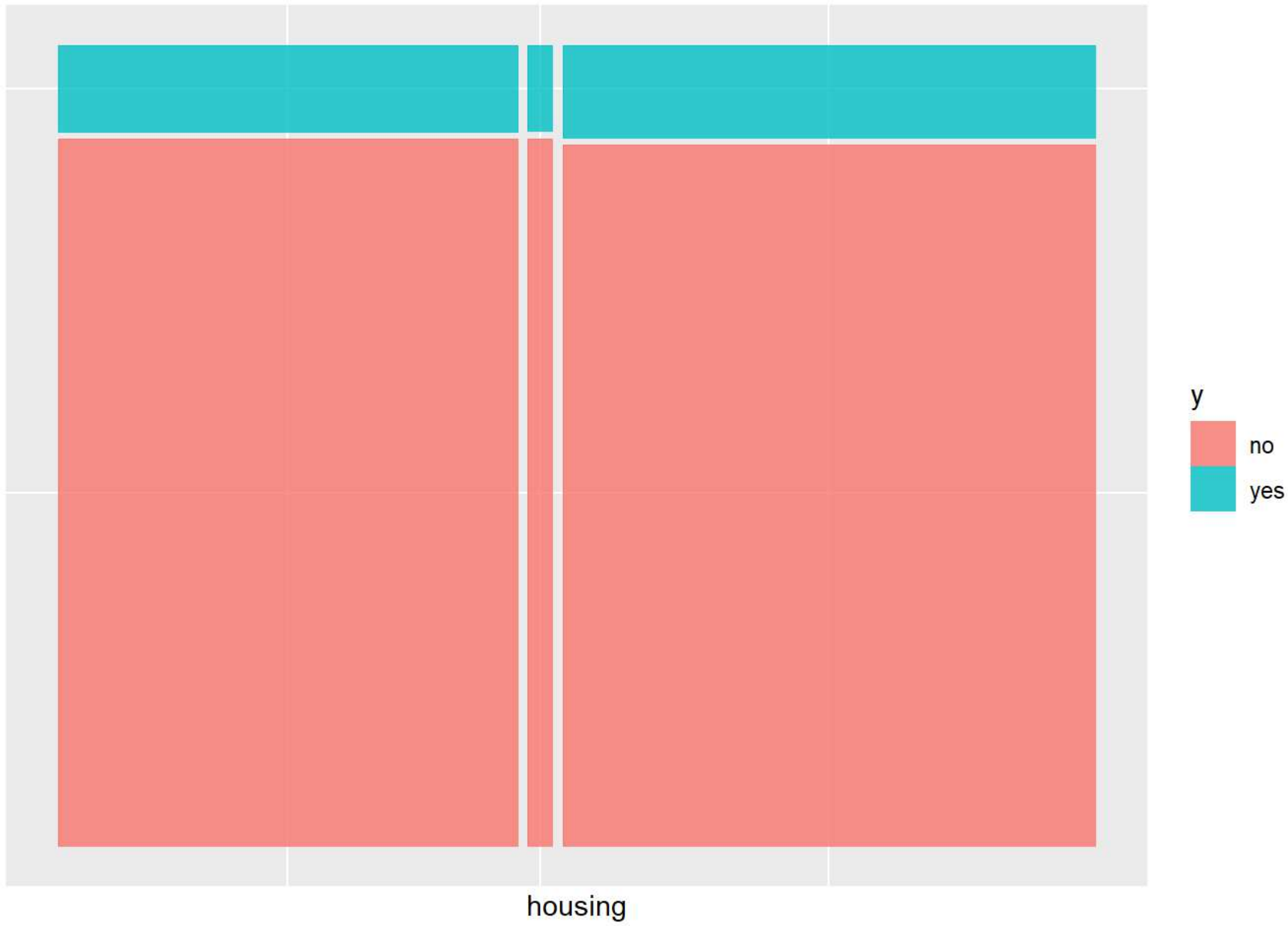
```
## can merge single+unknown, married+divorced since values are similar
bank_df = bank_df %>%
  mutate(marital = recode(marital, "unknown" = "single", "divorced"="married"))

# education
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, education),  fill = y)) +
  #mosaic_theme +
  xlab("Education") +
  ylab(NULL)
```




```
## recode unknown as univeristy degree because proportions are similar
bank_df = bank_df %>%
  mutate(education = recode(education, "unknown" = "university.degree"))

# housing
CrossTable(bank_df$housing, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("housing",
"y"))
```

```
## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
## 
## 
## Total Observations in Table:  41188 
## 
## 
##              | y 
##      housing |        no |       yes | Row Total | 
## -------------|-----------|-----------|-----------|
##           no |     16596 |      2026 |     18622 | 
##              |     0.891 |     0.109 |     0.452 | 
##              |     0.403 |     0.049 |           | 
## -------------|-----------|-----------|-----------|
##      unknown |       883 |       107 |       990 | 
##              |     0.892 |     0.108 |     0.024 | 
##              |     0.021 |     0.003 |           | 
## -------------|-----------|-----------|-----------|
##          yes |     19069 |      2507 |     21576 | 
##              |     0.884 |     0.116 |     0.524 | 
##              |     0.463 |     0.061 |           | 
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 | 
## -------------|-----------|-----------|-----------|
## 
## 
```

```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, housing),  fill = y)) +
  #mosaic_theme +
  xlab("housing") +
  ylab(NULL)
```



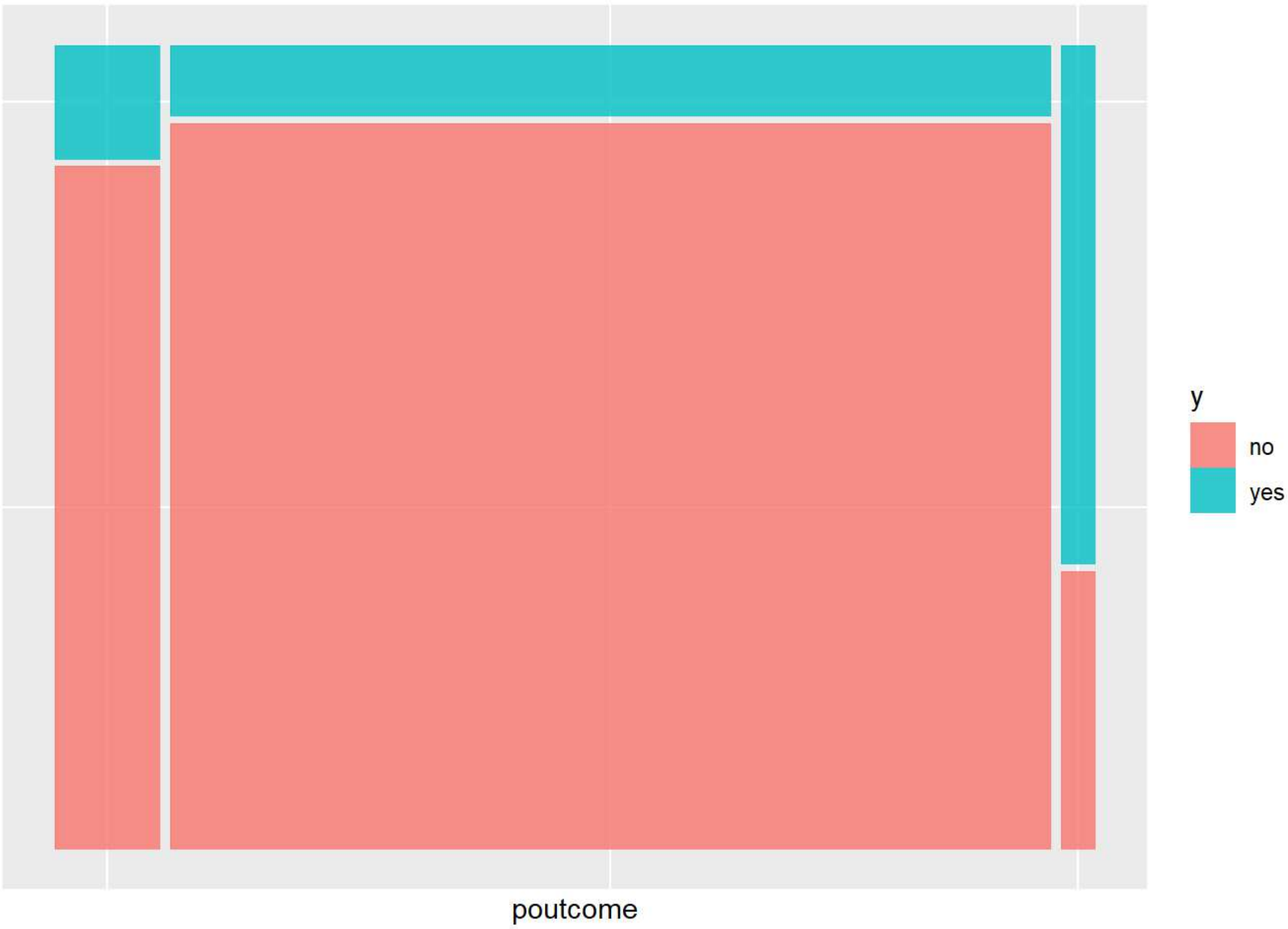

```
## the plot looks very similar, do chisquared test to see if there are differences
chisq.test(bank_df$housing, bank_df$y) # drop this column
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$housing and bank_df$y
## X-squared = 5.6845, df = 2, p-value = 0.05829
```

```
bank_df$housing <- NULL

# loan
chisq.test(bank_df$loan, bank_df$y) # drop col, pvalue >0.1
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$loan and bank_df$y
## X-squared = 1.094, df = 2, p-value = 0.5787
```

```
bank_df$loan <- NULL

# pdays
# poutcome
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, poutcome),  fill = y)) +
  #mosaic_theme +
  xlab("poutcome") +
  ylab(NULL)
```




```
bank_df = bank_df %>%
  mutate(past_dummyvar = recode(poutcome, "failure" = 0.5, "nonexistent"=0.2, "success"=1))
# combining previous and poutcome
bank_df$past_dummyvar1 = bank_df$past_dummyvar*(bank_df$previous+1)
chisq.test(bank_df$past_dummyvar1, bank_df$y)
```

```
## Warning in chisq.test(bank_df$past_dummyvar1, bank_df$y): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$past_dummyvar1 and bank_df$y
## X-squared = 4383.4, df = 11, p-value < 2.2e-16
```

```
bank_df$previous <-NULL
bank_df$poutcome <-NULL
bank_df$past_dummyvar <-NULL

bank_df = bank_df %>%
  mutate(pdays_dummy = if_else(pdays == 999, "0", "1")) %>%
  select(-pdays)
bank_df$pdays<-NULL

#resolve default, let yes become unknown
bank_df = bank_df %>%
  mutate(default = recode(default, "yes"="unknown"))


# dayofweek
bank_df = bank_df %>%
  mutate(day_of_week = recode(day_of_week, "mon"=1, "tue"=2,"wed"=3,"thu"=4,"fri"=5))

# age
bank_df = bank_df %>%
  mutate(age = if_else(
    age<20, 1, if_else(
      age<23, 2, if_else(
        age<26, 3, if_else(
          age<31, 4, if_else(
            age<41, 5, if_else(age<51, 6, 7)))))))

#dataset after preprocessing
dim(bank_df)
```

```
## [1] 41188    18
```

```
summary(bank_df)
```

```
##       age                   job           marital
##  Min.   :1.000   admin.     :10422   married:29540
##  1st Qu.:5.000   blue-collar: 9254   single :11648
##  Median :5.000   technician : 6743
##  Mean   :5.367   services   : 3969
##  3rd Qu.:6.000   management : 2924
##  Max.   :7.000   retired    : 1720
##                  (Other)    : 6156
##               education        default           contact          month
##  basic.4y           : 4176   no     :32588   cellular :26144   may    :13769
##  basic.6y           : 2292   unknown: 8600   telephone:15044   jul    : 7174
##  basic.9y           : 6045                                     aug    : 6178
##  high.school        : 9515                                     jun    : 5318
##  illiterate         :   18                                     nov    : 4101
##  professional.course: 5243                                     apr    : 2632
##  university.degree  :13899                                     (Other): 2016
##   day_of_week      duration         campaign       emp.var.rate
##  Min.   :1.00   Min.   :   0.0   Min.   : 1.000   Min.   :-3.40000
##  1st Qu.:2.00   1st Qu.: 102.0   1st Qu.: 1.000   1st Qu.:-1.80000
##  Median :3.00   Median : 180.0   Median : 2.000   Median : 1.10000
##  Mean   :2.98   Mean   : 258.3   Mean   : 2.568   Mean   : 0.08189
##  3rd Qu.:4.00   3rd Qu.: 319.0   3rd Qu.: 3.000   3rd Qu.: 1.40000
##  Max.   :5.00   Max.   :4918.0   Max.   :56.000   Max.   : 1.40000
##
##  cons.price.idx  cons.conf.idx     euribor3m      nr.employed     y
##  Min.   :92.20   Min.   :-50.8   Min.   :0.634   Min.   :4964   no :36548
##  1st Qu.:93.08   1st Qu.:-42.7   1st Qu.:1.344   1st Qu.:5099   yes: 4640
##  Median :93.75   Median :-41.8   Median :4.857   Median :5191
##  Mean   :93.58   Mean   :-40.5   Mean   :3.621   Mean   :5167
##  3rd Qu.:93.99   3rd Qu.:-36.4   3rd Qu.:4.961   3rd Qu.:5228
##  Max.   :94.77   Max.   :-26.9   Max.   :5.045   Max.   :5228
##
##  past_dummyvar1   pdays_dummy
##  Min.   :0.2000   Length:41188
##  1st Qu.:0.2000   Class :character
##  Median :0.2000   Mode  :character
##  Mean   :0.3703
##  3rd Qu.:0.2000
##  Max.   :8.0000
##
```

```
# splitting train and test
library(caTools)
set.seed(1)
smp_size <- floor(0.8*nrow(bank_df))
train_ind <- sample(seq_len(nrow(bank_df)), size = smp_size)
train <- bank_df[train_ind, ]
test <- bank_df[-train_ind, ]
```

KNN

```r
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1 tim
es

# knn.fit <- train(y ~., data = train, method = "knn",
#                  trControl=trctrl, tuneLength = 10) # tuneLength parameter tells the algorithm
to try different default values for the main parameter

knn.fit <- train(y ~., data = train, method = "knn",
                 trControl=trctrl) # tuneLength parameter tells the algorithm to try different d
efault values for the main parameter

# knn.fit <- train(y ~., data = train, method = "knn")#by default bootstrap is used to find tuni
ng parameter -> trCtrl
knn.fit
```

```
## k-Nearest Neighbors
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9003035  0.4532526
##   7  0.9020941  0.4586713
##   9  0.9035811  0.4636005
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```r
#predict using test data
knn.pred <- predict(knn.fit, newdata = test)
#knn.pred

#confusion matrix
cm.knn <- table(knn.pred, test$y)
cm.knn
```

```
##
## knn.pred   no  yes
##      no  7034  491
##      yes  267  446
```

```
TP <- cm.knn[2,2]
TN <- cm.knn[1,1]
FP <- cm.knn[2,1]
FN <- cm.knn[1,2]

#FPR / Type I error
FPR.knn = FP/(FP+TN)
FPR.knn
```

```
## [1] 0.03657033
```

```
#FNR / Type II error
FNR.knn = FN/(FN+TP)
FNR.knn
```

```
## [1] 0.5240128
```

```
#Precision
precis.knn = TP/(TP+FP)
precis.knn
```

```
## [1] 0.6255259
```

```
#Recall / sensitivity
recall.knn = TP/(TP+FN)
recall.knn
```

```
## [1] 0.4759872
```

```
#misclassification error
test.err.knn = 1-(sum(diag(cm.knn))/sum(cm.knn))
test.err.knn
```

```
## [1] 0.09201262
```

Logistic Regression

```
set.seed(8)
glm.fit <- glm(y ~., data = train, family = binomial)
summary(glm.fit)
```

```
## 
## Call:
## glm(formula = y ~ ., family = binomial, data = train)
## 
## Deviance Residuals: 
##     Min       1Q   Median       3Q      Max  
## -5.9280  -0.3025  -0.1891  -0.1390   3.2894  
## 
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)    
## (Intercept)                  -2.134e+02  4.262e+01  -5.007 5.53e-07 ***
## age                          -5.169e-02  2.515e-02  -2.055 0.039852 *  
## jobblue-collar               -2.468e-01  8.713e-02  -2.833 0.004609 ** 
## jobentrepreneur              -1.451e-01  1.389e-01  -1.045 0.296230    
## jobhousemaid                  3.990e-02  1.628e-01   0.245 0.806388    
## jobmanagement                -2.838e-02  9.460e-02  -0.300 0.764129    
## jobretired                    3.526e-01  1.066e-01   3.309 0.000938 ***
## jobself-employed             -1.506e-01  1.328e-01  -1.133 0.257060    
## jobservices                  -1.278e-01  9.358e-02  -1.366 0.172040    
## jobstudent                    5.965e-02  1.250e-01   0.477 0.633192    
## jobtechnician                 6.576e-03  7.838e-02   0.084 0.933134    
## jobunemployed                -8.935e-02  1.460e-01  -0.612 0.540417    
## jobunconventional            -4.305e-02  2.753e-01  -0.156 0.875763    
## maritalsingle                -3.191e-02  5.727e-02  -0.557 0.577456    
## educationbasic.6y             3.766e-02  1.364e-01   0.276 0.782449    
## educationbasic.9y             4.568e-02  1.057e-01   0.432 0.665517    
## educationhigh.school          5.549e-02  1.018e-01   0.545 0.585865    
## educationilliterate           1.559e+00  7.406e-01   2.105 0.035325 *  
## educationprofessional.course  1.016e-01  1.125e-01   0.904 0.366216    
## educationuniversity.degree    1.519e-01  9.881e-02   1.538 0.124123    
## defaultunknown               -2.954e-01  7.457e-02  -3.962 7.45e-05 ***
## contacttelephone             -6.306e-01  8.568e-02  -7.360 1.84e-13 ***
## monthaug                      7.204e-01  1.333e-01   5.406 6.43e-08 ***
## monthdec                      3.302e-01  2.295e-01   1.438 0.150305    
## monthjul                      7.833e-02  1.062e-01   0.737 0.460947    
## monthjun                     -4.725e-01  1.401e-01  -3.374 0.000742 ***
## monthmar                      1.946e+00  1.598e-01  12.179  < 2e-16 ***
## monthmay                     -5.366e-01  9.107e-02  -5.892 3.83e-09 ***
## monthnov                     -5.657e-01  1.347e-01  -4.200 2.67e-05 ***
## monthoct                      5.546e-02  1.708e-01   0.325 0.745431    
## monthsep                      2.865e-01  1.991e-01   1.439 0.150154    
## day_of_week                   2.378e-02  1.601e-02   1.486 0.137376    
## duration                      4.604e-03  8.228e-05  55.950  < 2e-16 ***
## campaign                     -3.926e-02  1.274e-02  -3.082 0.002057 ** 
## emp.var.rate                 -1.668e+00  1.585e-01 -10.527  < 2e-16 ***
## cons.price.idx                2.030e+00  2.808e-01   7.231 4.80e-13 ***
## cons.conf.idx                 1.945e-02  8.552e-03   2.274 0.022956 *  
## euribor3m                     3.852e-01  1.436e-01   2.683 0.007290 ** 
## nr.employed                   3.749e-03  3.461e-03   1.083 0.278780    
## past_dummyvar1               -2.045e-01  5.328e-02  -3.838 0.000124 ***
## pdays_dummy1                  1.854e+00  1.320e-01  14.052  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23162  on 32949  degrees of freedom
## Residual deviance: 13829  on 32909  degrees of freedom
## AIC: 13911
##
## Number of Fisher Scoring iterations: 6
```

```
#predict using test data
glm.prob <- predict(glm.fit, type = "response", newdata = test)

#check which one is 'Yes'
contrasts(test$y)#Yes = 1, Low = 0
```

```
##     yes
## no    0
## yes   1
```

```
glm.pred <- rep('no', nrow(test))
glm.pred[glm.prob > 0.5] <- 'yes' #yes = 1, no = 0

#confusion matrix
cm.reg = table(glm.pred, test$y)
cm.reg
```

```
##
## glm.pred   no  yes
##       no  7111  546
##       yes  190  391
```

```
TP <- cm.reg[2,2]
TN <- cm.reg[1,1]
FP <- cm.reg[2,1]
FN <- cm.reg[1,2]

#FPR / Type I error
FPR.reg = FP/(FP+TN)
FPR.reg
```

```
## [1] 0.02602383
```

```
#FNR / Type II error
FNR.reg = FN/(FN+TP)
FNR.reg
```

```
## [1] 0.5827108
```

```
#Precision
precis.reg = TP/(TP+FP)
precis.reg
```

```
## [1] 0.6729776
```

```
#Recall / sensitivity
recall.reg = TP/(TP+FN)
recall.reg
```

```
## [1] 0.4172892
```

```
#misclassification error
test.err.reg = 1-(sum(diag(cm.reg))/sum(cm.reg))
test.err.reg
```

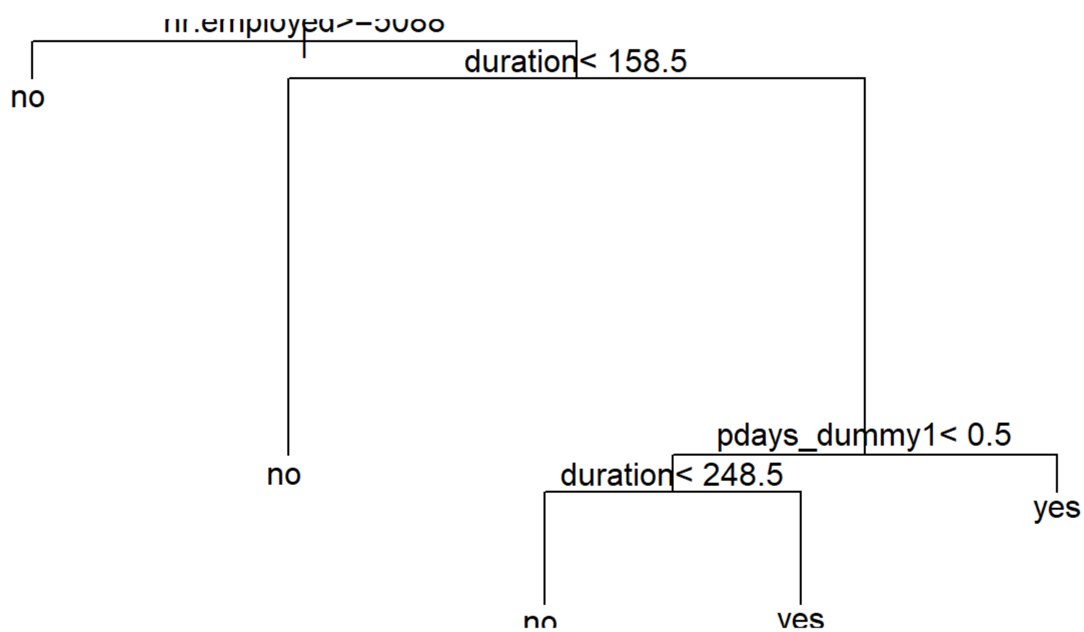
```
## [1] 0.08934207
```

Decision Tree

```
# train.tree <- data.frame(train)
# test.tree <-data.frame(test)

#Decision Tree using rpart()
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1 tim
es

cls.tree1 = train(y ~ ., data=train, method="rpart",
                  trControl=trctrl)# tuneLength parameter tells the algorithm to try different d
efault values for the main parameter

# cls.tree1 = train(y ~ ., data=train.tree, method="rpart")#by default bootstrap is used to find
tuning parameter -> trCtrl

cls.tree1
```

```
## CART
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01876857  0.9085282  0.4758751
##   0.02106400  0.9058881  0.4220528
##   0.07061842  0.8964786  0.2597907
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01876857.
```

```
plot(cls.tree1$finalModel)
text(cls.tree1$finalModel)
```



```r
#predict using test data
tree.pred1 <- predict(cls.tree1, newdata = test)
#tree.pred1

#confusion matrix
cm.tree1 <- table(tree.pred1, test$y)
cm.tree1
```

```
## 
## tree.pred1   no  yes
##        no  7145  612
##        yes  156  325
```

```r
TP <- cm.tree1[2,2]
TN <- cm.tree1[1,1]
FP <- cm.tree1[2,1]
FN <- cm.tree1[1,2]

#FPR / Type I error
FPR.tree1 = FP/(FP+TN)
FPR.tree1
```

```
## [1] 0.02136694
```

```r
#FNR / Type II error
FNR.tree1 = FN/(FN+TP)
FNR.tree1
```

```
## [1] 0.6531483
```

```r
#Precision
precis.tree1 = TP/(TP+FP)
precis.tree1
```

```
## [1] 0.6756757
```

```r
#Recall / sensitivity
recall.tree1 = TP/(TP+FN)
recall.tree1
```

```
## [1] 0.3468517
```

```r
#misclassification error
test.err.tree1 = 1-(sum(diag(cm.tree1))/sum(cm.tree1))
test.err.tree1
```

```
## [1] 0.09322651
```

Random Forest

```
#Random forest with 500 bootstrapped trees
#p = 16
sqrt(16) # ntree = 4
```

```
## [1] 4
```

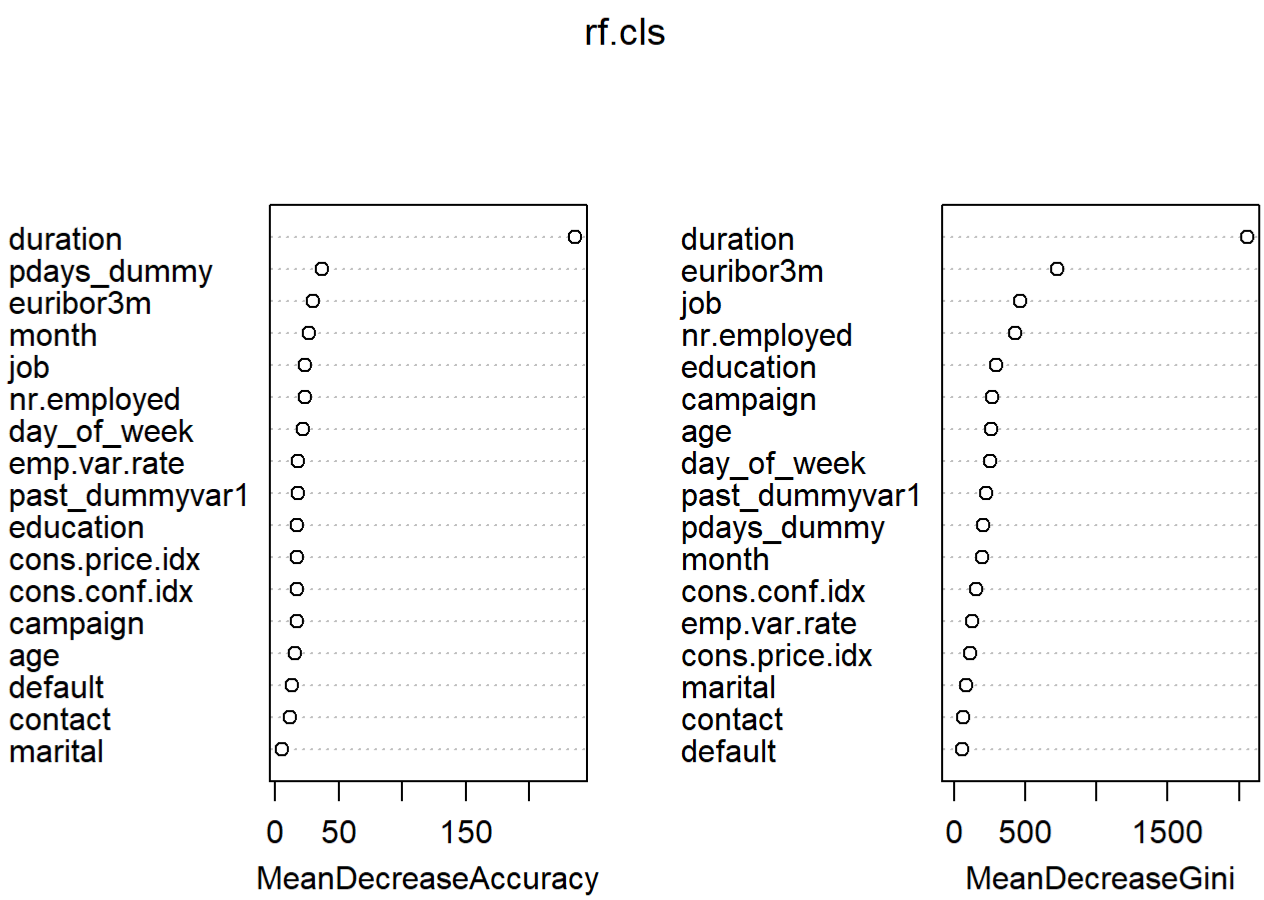
```
set.seed(8)
rf.cls <- randomForest(y ~ .,
                       data = train,
                       mtry = 4,
                       ntree = 500,
                       importance = TRUE)
rf.cls
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train, mtry = 4, ntree = 500,      importance = TRUE)
##               Type of random forest: classification
##                     Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 8.53%
## Confusion matrix:
##        no  yes class.error
## no  28169 1078  0.03685848
## yes  1733 1970  0.46799892
```

```
#ls(rf.cls)
importance(rf.cls)
```

```
##                         no         yes MeanDecreaseAccuracy MeanDecreaseGini
## age              16.618209   2.7787032            15.783747        260.22417
## job              32.300664  -8.6716373            23.574378        469.06925
## marital           6.877160  -0.8560175             5.093426         88.98439
## education        20.600919   1.3791306            17.335005        299.50452
## default           7.651229   8.8613381            13.279836         57.17673
## contact           6.327229  31.2466857            11.892202         64.56189
## month            25.719212   4.8937613            26.677099        197.79415
## day_of_week      20.444460   6.9352422            21.521504        255.79902
## duration        145.535945 264.8679961           236.250246       2061.58784
## campaign          8.720273  14.5443996            16.810597        271.53416
## emp.var.rate     17.261693   7.4390000            18.286362        131.81957
## cons.price.idx   16.910552  -3.5721354            16.876427        114.99034
## cons.conf.idx    15.841751   4.6209383            16.844132        154.88268
## euribor3m        26.833225  15.4508925            30.102870        727.14260
## nr.employed      18.716957  22.0890773            23.222963        429.78559
## past_dummyvar1    8.833548  23.9317166            18.129254        226.81197
## pdays_dummy       1.442605  60.5843448            36.882401        208.74446
```

```
varImpPlot(rf.cls)
```



```r
#predict using test data
rf.pred <- predict(rf.cls, newdata = test, type = "class")
#rf.pred

#confusion matrix
cm.rf <- table(rf.pred, test$y)
cm.rf
```

```
##
## rf.pred   no  yes
##      no  7040  420
##      yes  261  517
```

```r
TP <- cm.rf[2,2]
TN <- cm.rf[1,1]
FP <- cm.rf[2,1]
FN <- cm.rf[1,2]

#FPR / Type I error
FPR.rf = FP/(FP+TN)
FPR.rf
```

```
## [1] 0.03574853
```

```r
#FNR / Type II error
FNR.rf = FN/(FN+TP)
FNR.rf
```

```
## [1] 0.4482391
```

```r
#Precision
precis.rf = TP/(TP+FP)
precis.rf
```

```
## [1] 0.6645244
```

```r
#Recall / sensitivity
recall.rf = TP/(TP+FN)
recall.rf
```

```
## [1] 0.5517609
```

```r
#misclassification error
test.err.rf = 1-(sum(diag(cm.rf))/sum(cm.rf))
test.err.rf
```

```
## [1] 0.0826657
```

Gradient Boosting

```
#Gradient boosting
set.seed(8)

#Use K-fold CV to find best trControl
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 1) #5 folds repeated 1 times

gbm.fit <- train(y ~ ., data = train,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)

# gbm.fit <- train(y ~ ., data = train,
#                   method = "gbm",
#                   verbose = FALSE) #by default bootstrap is used to find tuning parameter -> tr
Ctrl
gbm.fit
```

```
## Stochastic Gradient Boosting
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9054326  0.3425904
##   1                  100      0.9085587  0.4064470
##   1                  150      0.9098333  0.4385889
##   2                   50      0.9091959  0.4340753
##   2                  100      0.9124735  0.4971532
##   2                  150      0.9133840  0.5092936
##   3                   50      0.9117148  0.4927134
##   3                  100      0.9135356  0.5149861
##   3                  150      0.9157512  0.5309642
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
#predict using test data
gbm.pred <- predict(gbm.fit, newdata = test)
#gbm.pred

#confusion matrix
cm.gbm <- table(gbm.pred, test$y)
cm.gbm
```

```
##
## gbm.pred   no  yes
##      no  7055  429
##      yes  246  508
```

```
TP <- cm.gbm[2,2]
TN <- cm.gbm[1,1]
FP <- cm.gbm[2,1]
FN <- cm.gbm[1,2]

#FPR / Type I error
FPR.gbm = FP/(FP+TN)
FPR.gbm
```

```
## [1] 0.03369401
```

```
#FNR / Type II error
FNR.gbm = FN/(FN+TP)
FNR.gbm
```

```
## [1] 0.4578442
```

```
#Precision
precis.gbm = TP/(TP+FP)
precis.gbm
```

```
## [1] 0.6737401
```

```
#Recall / sensitivity
recall.gbm = TP/(TP+FN)
recall.gbm
```

```
## [1] 0.5421558
```

```
#misclassification error
test.err.gbm = 1-(sum(diag(cm.gbm))/sum(cm.gbm))
test.err.gbm
```

```
## [1] 0.08193736
```

AdaBoost

```
#AdaBoost
set.seed(8)
x.trainA = model.matrix(data=train, y~.-1)
y.trainA = rep(1, nrow(train))
y.trainA [train$y=="no"]=-1 #for Adaboost

x.testA = model.matrix(data=test, y~.-1)
y.testA = rep(1, nrow(test))
y.testA [test$y=="no"]=-1 #for Adaboost

ada.cls <- adaboost(x.trainA, y.trainA, tree_depth=5, n_rounds=500)
ada.cls
```

```
## AdaBoost: tree_depth =  5  rounds =  500
##
##
##  In-sample confusion matrix:
##     yhat
## y        -1     1
##   -1 28415   832
##   1   1430  2273
```

```
#predict using test data
ada.pred <- predict(ada.cls, x.testA)
#ada.pred

#confusion matrix
cm.ada <- table(ada.pred, y.testA) #-1 is "no", 1 is "yes"
cm.ada
```

```
##         y.testA
## ada.pred   -1    1
##       -1 6998  434
##       1   303  503
```

```
TP <- cm.ada[2,2]
TN <- cm.ada[1,1]
FP <- cm.ada[2,1]
FN <- cm.ada[1,2]

#FPR / Type I error
FPR.ada = FP/(FP+TN)
FPR.ada
```

```
## [1] 0.04150116
```

```
#FNR / Type II error
FNR.ada = FN/(FN+TP)
FNR.ada
```

```
## [1] 0.4631804
```

```
#Precision
precis.ada = TP/(TP+FP)
precis.ada
```

```
## [1] 0.6240695
```

```
#Recall / sensitivity
recall.ada = TP/(TP+FN)
recall.ada
```

```
## [1] 0.5368196
```

```
#misclassification error
test.err.ada = 1-(sum(diag(cm.ada))/sum(cm.ada))
test.err.ada
```

```
## [1] 0.08946346
```

XGBoost

```
#XGBoost
set.seed(8)
x.trainXG =model.matrix(data=train,y~.-1)
y.trainXG = rep(1, nrow(train))
y.trainXG[train$y=="no"]=0 #for XGBoost

x.testXG = model.matrix(data=test, y~.-1)
y.testXG = rep(1, nrow(test))
y.testXG[test$y=="no"]=0 #for XGBoost

xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=5,eta=0.01,nrounds=500,verbose=FALSE
)
#xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=10,nrounds=500,verbose=FALSE)
xgb.cls
```

```
## ##### xgb.Booster
## raw: 1 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 5, eta = 0.01)
## params (as set within xgb.train):
##   max_depth = "5", eta = "0.01", validate_parameters = "1"
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 41
## niter: 500
## nfeatures : 41
## evaluation_log:
##     iter train_rmse
##        1   0.496162
##        2   0.492358
## ---
##      499   0.225786
##      500   0.225769
```

```
#predict using test data
xgb.pred.prob<-predict(xgb.cls,x.testXG)

xgb.pred<-as.numeric(xgb.pred.prob>0.5) #convert to 0 ("no") or 1 ("yes")

#confusion matrix
cm.xgb<-table(xgb.pred,y.testXG) #0 is "no", 1 is "yes"
cm.xgb
```

```
##         y.testXG
## xgb.pred    0    1
##        0 7057  428
##        1  244  509
```

```
TP <- cm.xgb[2,2]
TN <- cm.xgb[1,1]
FP <- cm.xgb[2,1]
FN <- cm.xgb[1,2]

#FPR / Type I error
FPR.xgb = FP/(FP+TN)
FPR.xgb
```

```
## [1] 0.03342008
```

```
#FNR / Type II error
FNR.xgb = FN/(FN+TP)
FNR.xgb
```

```
## [1] 0.4567769
```

```
#Precision
precis.xgb = TP/(TP+FP)
precis.xgb
```

```
## [1] 0.6759628
```

```
#Recall / sensitivity
recall.xgb = TP/(TP+FN)
recall.xgb
```

```
## [1] 0.5432231
```

```
#misclassification error
test.err.xgb = 1-sum(diag(cm.xgb))/sum(cm.xgb)
test.err.xgb
```

```
## [1] 0.0815732
```

SVM with linear kernel

```
set.seed(8)
svm.fit <- svm(y~., data=train, kernel='linear', cost=1)
#summary(svm.fit)

#CV for tuning the cost parameter
set.seed(8)
tune.out1 <- tune(svm, y~.,
              data=train,
              kernel="linear",
              )

#tune.out1 <- tune(svm, y~.,
#               data=train,
#               kernel="linear",
#               ranges=list(cost=c(0.01, 0.1, 1, 10, 100)),tunecontrol=tune.control(cross=10))
summary(tune.out1)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09732929
```

```r
svm.lin.best <- tune.out1$best.model
summary(svm.lin.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  6641
##
##  ( 3329 3312 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```r
#predict using test data
lin.pred <- predict(svm.lin.best, test)

#confusion matrix
cm.lin <- table(lin.pred, test$y)
cm.lin
```

```
##
## lin.pred   no  yes
##      no  7146  646
##      yes  155  291
```

```r
TP <- cm.lin[2,2]
TN <- cm.lin[1,1]
FP <- cm.lin[2,1]
FN <- cm.lin[1,2]

#FPR / Type I error
FPR.lin = FP/(FP+TN)
FPR.lin
```

```
## [1] 0.02122997
```

```
#FNR / Type II error
FNR.lin = FN/(FN+TP)
FNR.lin
```

```
## [1] 0.6894344
```

```
#Precision
precis.lin = TP/(TP+FP)
precis.lin
```

```
## [1] 0.6524664
```

```
#Recall / sensitivity
recall.lin = TP/(TP+FN)
recall.lin
```

```
## [1] 0.3105656
```

```
#misclassification error
test.err.lin = 1-(sum(diag(cm.lin))/sum(cm.lin))
test.err.lin
```

```
## [1] 0.09723234
```

SVM with polynomial kernel

```
set.seed(8)
tune.out2 <- tune(svm, y~.,
              data=train,
              kernel="polynomial",
              )

#tune.out2 <- tune(svm, y~.,
#              data=train,
#              kernel="polynomial",
#              ranges=list(cost=c(0.1,1,5,10,15,20,50,100),
#                          degree=c(2,3,4)))
summary(tune.out2)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09456753
```

```
svm.poly.best <- tune.out2$best.model
summary(svm.poly.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  6712
##
##  ( 3399 3313 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
poly.pred <- predict(svm.poly.best, test)

#confusion matrix
cm.poly <- table(poly.pred, test$y)
cm.poly
```

```
##
## poly.pred   no  yes
##       no  7214  691
##       yes   87  246
```

```
TP <- cm.poly[2,2]
TN <- cm.poly[1,1]
FP <- cm.poly[2,1]
FN <- cm.poly[1,2]

#FPR / Type I error
FPR.poly = FP/(FP+TN)
FPR.poly
```

```
## [1] 0.01191618
```

```
#FNR / Type II error
FNR.poly = FN/(FN+TP)
FNR.poly
```

```
## [1] 0.73746
```

```
#Precision
precis.poly = TP/(TP+FP)
precis.poly
```

```
## [1] 0.7387387
```

```
#Recall / sensitivity
recall.poly = TP/(TP+FN)
recall.poly
```

```
## [1] 0.26254
```

```
#misclassification error
test.err.poly = 1-(sum(diag(cm.poly))/sum(cm.poly))
test.err.poly
```

```
## [1] 0.0944404
```

SVM with rbf kernel

```
set.seed(8)
tune.out3 <- tune(svm, y~.,
              data=train,
              kernel="radial",)

#tune.out3 <- tune(svm, y~.,
#              data=train,
#              kernel="radial",
#              ranges=list(cost=c(0.1,1,5,10),
#                          gamma=c(0.01,0.1,1,5,10,100)))
summary(tune.out3)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09125948
```

```
svm.rbf.best <- tune.out3$best.model
summary(svm.rbf.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  6573
##
##  ( 3334 3239 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
rbf.pred <- predict(svm.rbf.best, test)

#confusion matrix
cm.rbf <- table(rbf.pred, test$y)
cm.rbf
```

```
##
## rbf.pred   no  yes
##      no  7161  614
##      yes  140  323
```

```
TP <- cm.rbf[2,2]
TN <- cm.rbf[1,1]
FP <- cm.rbf[2,1]
FN <- cm.rbf[1,2]

#FPR / Type I error
FPR.rbf = FP/(FP+TN)
FPR.rbf
```

```
## [1] 0.01917546
```

```
#FNR / Type II error
FNR.rbf = FN/(FN+TP)
FNR.rbf
```

```
## [1] 0.6552828
```

```
#Precision
precis.rbf = TP/(TP+FP)
precis.rbf
```

```
## [1] 0.6976242
```

```
#Recall / sensitivity
recall.rbf = TP/(TP+FN)
recall.rbf
```

```
## [1] 0.3447172
```

```
#misclassification error
test.err.rbf = 1-(sum(diag(cm.rbf))/sum(cm.rbf))
test.err.rbf
```

```
## [1] 0.09152707
```

Result Summary

```
options(digits = 3)
cl.err <- matrix(c(test.err.knn,FNR.knn,precis.knn,recall.knn,
                   test.err.reg,FNR.reg,precis.reg,recall.reg,
                   test.err.tree1,FNR.tree1,precis.tree1,recall.tree1,
                   test.err.rf,FNR.rf,precis.rf,recall.rf,
                   test.err.gbm,FNR.gbm,precis.gbm,recall.gbm,
                   test.err.ada,FNR.ada,precis.ada,recall.ada,
                   test.err.xgb,FNR.xgb,precis.xgb,recall.xgb,
                   test.err.lin,FNR.lin,precis.lin,recall.lin,
                   test.err.poly,FNR.poly,precis.poly,recall.poly,
                   test.err.rbf,FNR.rbf,precis.rbf,recall.rbf),
                   ncol=4, byrow=TRUE)
colnames(cl.err) <- c('misclass error','type-II error','precision','recall')
rownames(cl.err) <- c('KNN',
                      'Logistic regression',
                      'Decision tree with rpart',
                      'Random forest',
                      'Gradient boosting',
                      'Adaboost',
                      'XGBoost',
                      'SVM with linear kernel',
                      'SVM with polynomial kernel',
                      'SVM with radial kernel')
as.table(cl.err)
```

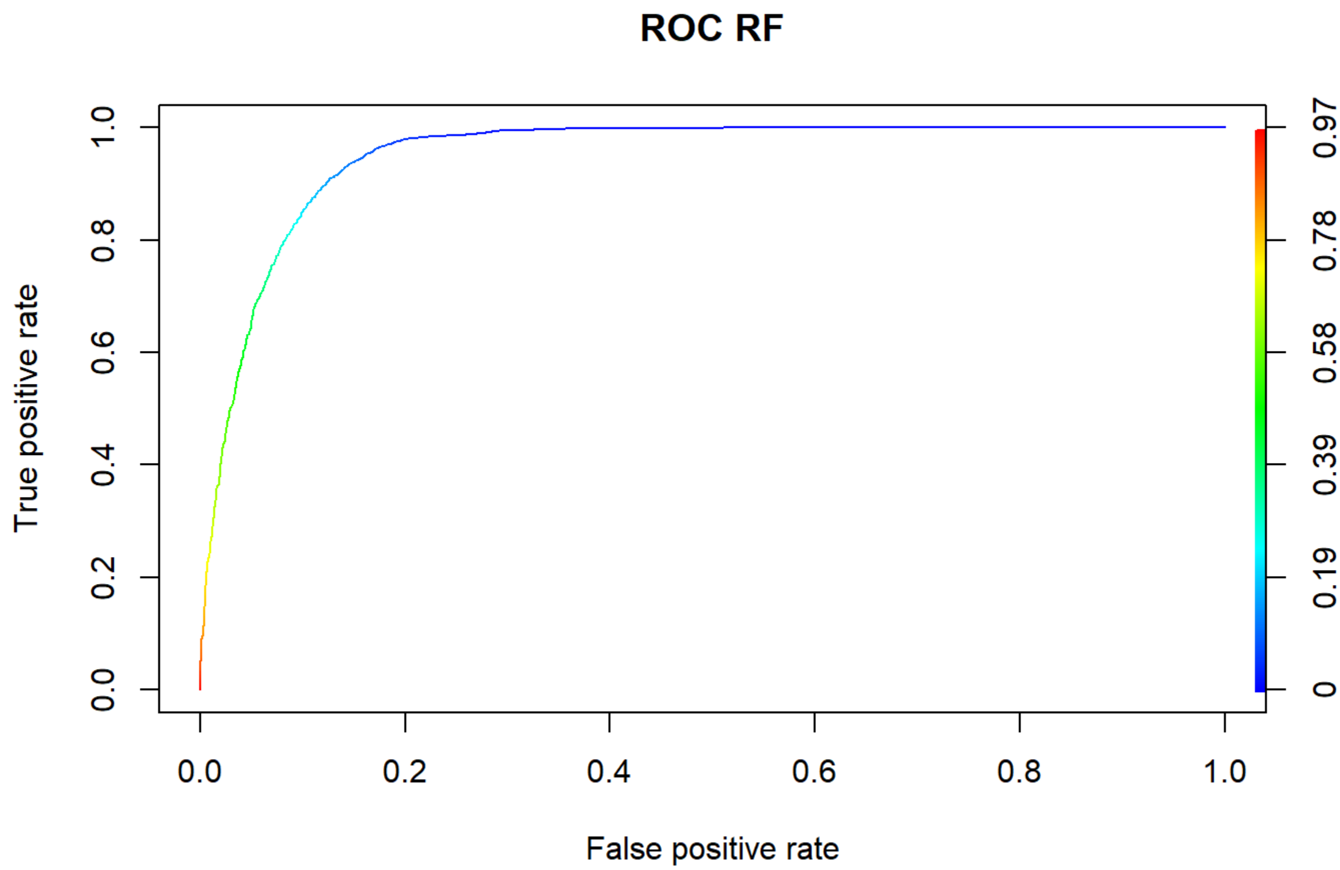
```
##                              misclass error type-II error precision recall
## KNN                                  0.0920        0.5240    0.6255 0.4760
## Logistic regression                  0.0893        0.5827    0.6730 0.4173
## Decision tree with rpart             0.0932        0.6531    0.6757 0.3469
## Random forest                        0.0827        0.4482    0.6645 0.5518
## Gradient boosting                    0.0819        0.4578    0.6737 0.5422
## Adaboost                             0.0895        0.4632    0.6241 0.5368
## XGBoost                              0.0816        0.4568    0.6760 0.5432
## SVM with linear kernel               0.0972        0.6894    0.6525 0.3106
## SVM with polynomial kernel           0.0944        0.7375    0.7387 0.2625
## SVM with radial kernel               0.0915        0.6553    0.6976 0.3447
```

Based on Type-II error comparison, best models are shortlisted: Random Forest, XGBoost, Adaboost, Gradient boosting.

ROC and AUC

Random Forest

```
#Prepare model for ROC curve
rf.pred <- predict(rf.cls, newdata = test, type = "prob")

forestpred = prediction(rf.pred[,2], test$y)

roc.perf.rf = performance(forestpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.rf, main='ROC RF', colorize=T)
```




```r
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.rf, forestpred))
roc.result = as.data.frame((opt.cut(roc.perf.rf, forestpred)))
roc.result
```

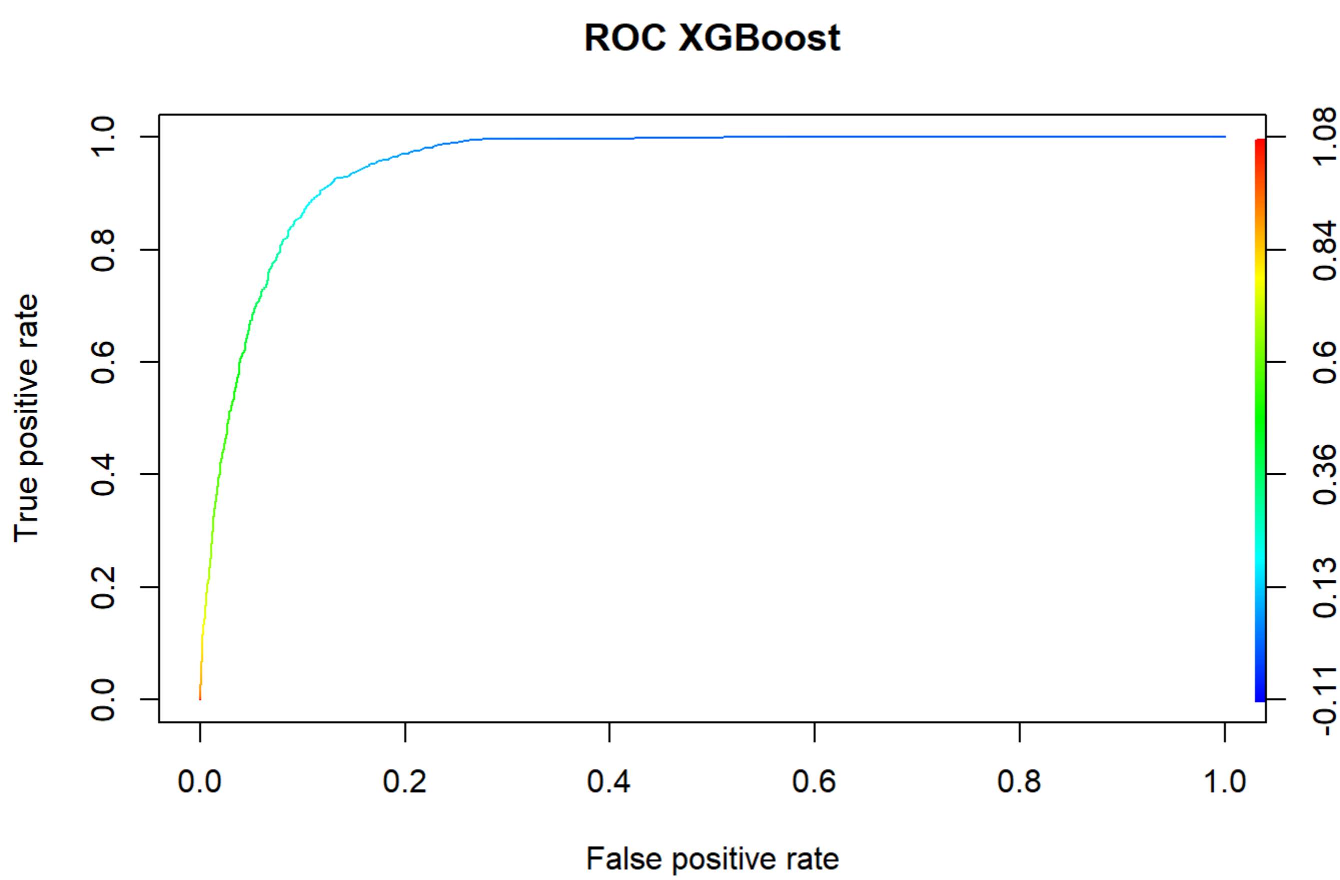
```
##                V1
## sensitivity 0.909
## specificity 0.873
## cutoff      0.146
```

```
rf.sens = roc.result[1,]
rf.spec = roc.result[2,]
rf.cutoff = roc.result[3,]

auc.perf.rf = performance(forestpred, measure = 'auc')
auc.rf = auc.perf.rf@y.values
auc.rf
```

```
## [[1]]
## [1] 0.951
```

XGBoost

```
#Prepare model for ROC curve
xgbpred = prediction(xgb.pred.prob, test$y)

roc.perf.xgb = performance(xgbpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.xgb, main='ROC XGBoost', colorize=T)
```



```r
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.xgb, xgbpred))
roc.result = as.data.frame((opt.cut(roc.perf.xgb, xgbpred)))
roc.result
```

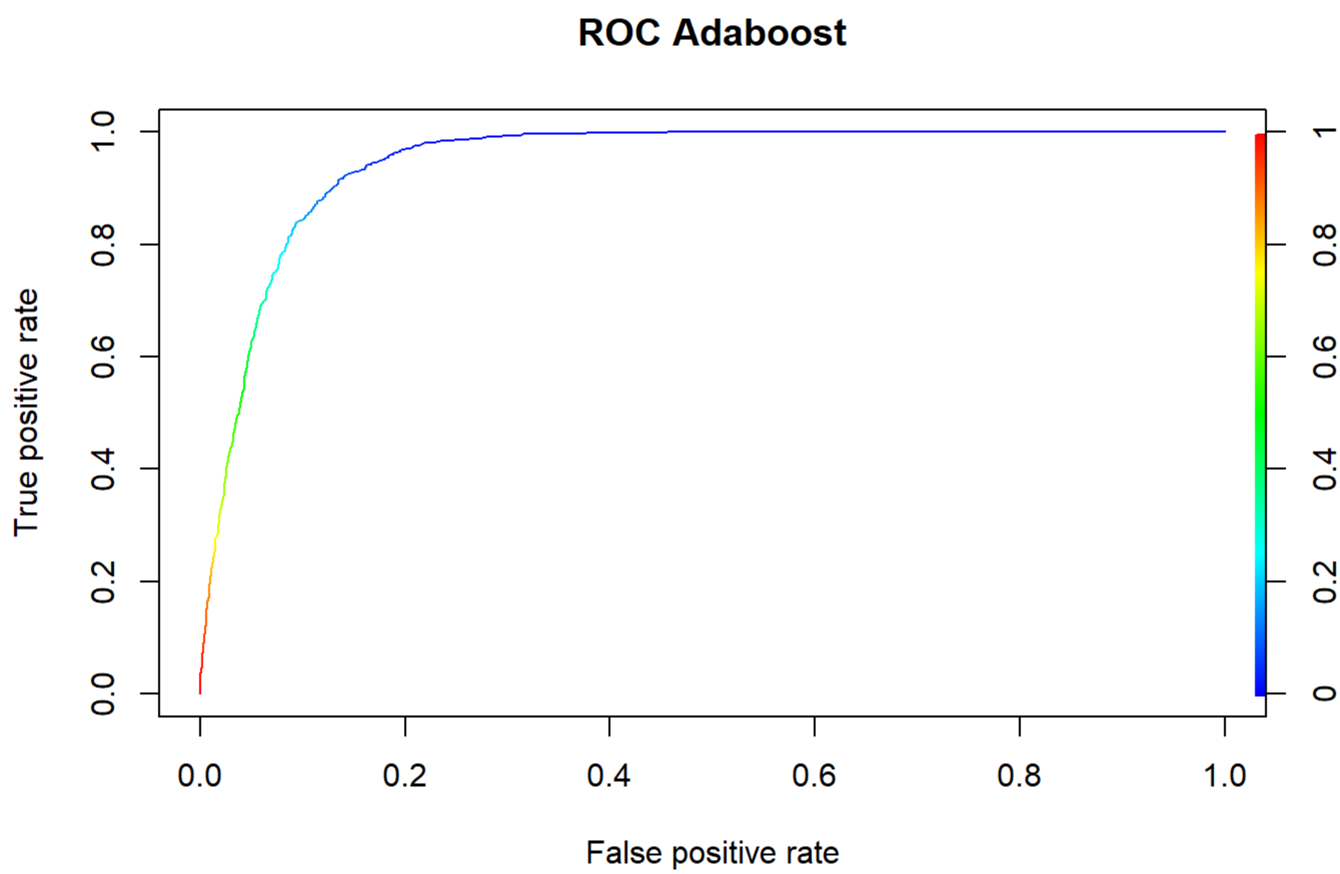
```
##                V1
## sensitivity 0.906
## specificity 0.883
## cutoff      0.166
```

```r
xgb.sens = roc.result[1,]
xgb.spec = roc.result[2,]
xgb.cutoff = roc.result[3,]

auc.perf.xgb = performance(xgbpred, measure = 'auc')
auc.xgb = auc.perf.xgb@y.values
auc.xgb
```

```
## [[1]]
## [1] 0.953
```

Adaboost

```r
#Prepare model for ROC curve
ada.pred <- predict(ada.cls, x.testA, type = "prob")

adapred = prediction(ada.pred, test$y)

roc.perf.ada = performance(adapred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.ada, main='ROC Adaboost', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.ada, adapred))
roc.result = as.data.frame((opt.cut(roc.perf.ada, adapred)))
roc.result
```

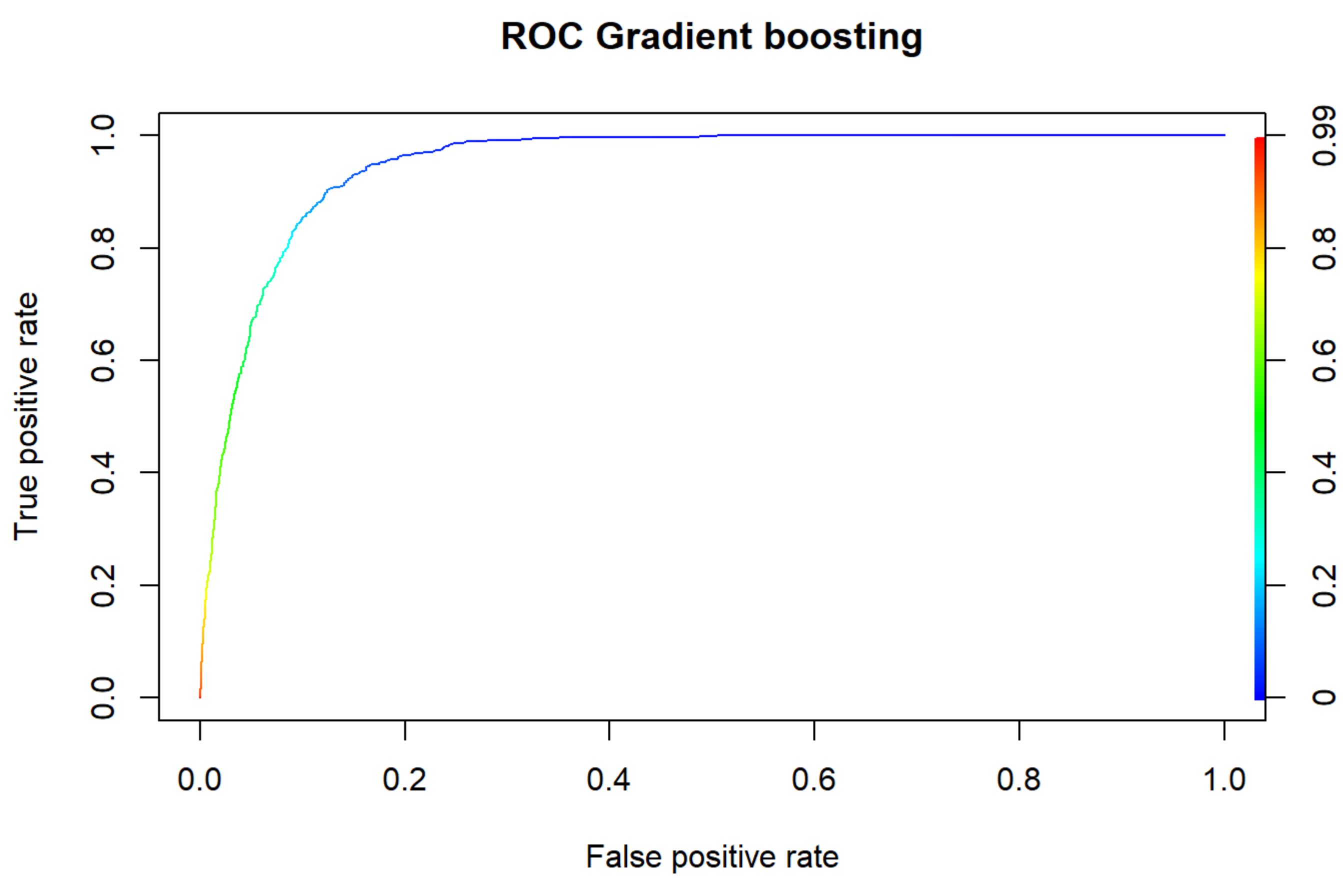
```
##                   V1
## sensitivity 0.9168
## specificity 0.8645
## cutoff      0.0876
```

```
ada.sens = roc.result[1,]
ada.spec = roc.result[2,]
ada.cutoff = roc.result[3,]

auc.perf.ada = performance(adapred, measure = 'auc')
auc.ada = auc.perf.ada@y.values
auc.ada
```

```
## [[1]]
## [1] 0.946
```

Gradient boosting

```
#Prepare model for ROC curve
gbm.pred <- predict (gbm.fit, test, type = "prob")

gbmpred = prediction(gbm.pred[,2], test$y)

roc.perf.gbm = performance(gbmpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.gbm, main='ROC Gradient boosting', colorize=T)
```



```r
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.gbm, gbmpred))
roc.result = as.data.frame((opt.cut(roc.perf.gbm, gbmpred)))
roc.result
```

```
##                V1
## sensitivity 0.904
## specificity 0.876
## cutoff      0.136
```

```r
gbm.sens = roc.result[1,]
gbm.spec = roc.result[2,]
gbm.cutoff = roc.result[3,]

auc.perf.gbm = performance(gbmpred, measure = 'auc')
auc.gbm = auc.perf.gbm@y.values
auc.gbm
```

```
## [[1]]
## [1] 0.949
```

AUC Summary

```r
options(digits = 3)
cl.err <- matrix(c(
                   test.err.rf, FNR.rf, rf.cutoff, (1-rf.sens), auc.rf,
                   test.err.xgb, FNR.xgb, xgb.cutoff, (1-xgb.sens), auc.xgb,
                   test.err.ada, FNR.ada, ada.cutoff, (1-ada.sens), auc.ada,
                   test.err.gbm, FNR.gbm, gbm.cutoff, (1-gbm.sens), auc.gbm
                   ),
                   ncol=5, byrow=TRUE)
colnames(cl.err) <- c('misclass err','Type-II err@0.5', 'cutoff', 'Type-II err@cutoff','AUC')
rownames(cl.err) <- c(
                      'RandomForest',
                      'XGBoost',
                      'Adaboost',
                      'Gradboost')
as.matrix(cl.err)
```

```
##               misclass err Type-II err@0.5 cutoff Type-II err@cutoff AUC
## RandomForest 0.0827       0.448           0.146  0.0907             0.951
## XGBoost      0.0816       0.457           0.166  0.0939             0.953
## Adaboost     0.0895       0.463           0.0876 0.0832             0.946
## Gradboost    0.0819       0.458           0.136  0.0961             0.949
```

Bank Marketing Data

Group 8

Load Data

```
#Read dataset
bank_df <- read_delim("bank-additional-full.csv", delim=";")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   age = col_double(),
##   duration = col_double(),
##   campaign = col_double(),
##   pdays = col_double(),
##   previous = col_double(),
##   emp.var.rate = col_double(),
##   cons.price.idx = col_double(),
##   cons.conf.idx = col_double(),
##   euribor3m = col_double(),
##   nr.employed = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
#Assign category to all categorical variables

#2.job as category
bank_df$job <- as.factor(bank_df$job)

#3.marital status as category
bank_df$marital <- as.factor(bank_df$marital)

#4.education as category
bank_df$education <- as.factor(bank_df$education)

#5.credit default as category
bank_df$default <- as.factor(bank_df$default)

#6.housing loan as category
bank_df$housing <- as.factor(bank_df$housing)

#7.personal loan as category
bank_df$loan <- as.factor(bank_df$loan)

#8.contact communication type as category
bank_df$contact <- as.factor(bank_df$contact)

#9.last contact month of year as category
bank_df$month <- as.factor(bank_df$month)

#10.last contact day of the month as category
bank_df$day_of_week <- as.factor(bank_df$day_of_week)

#15.outcome of the previous marketing campaign as category
bank_df$poutcome <- as.factor(bank_df$poutcome)

#21.output y as binary factor
bank_df$y <- factor(bank_df$y, levels = c("no","yes"))

dim(bank_df)
```

```
## [1] 41188    21
```

Data preprocessing

```
bank_df %>%
  summarise_all(list(~sum(. == "unknown"))) %>%
  gather(key = "variable", value = "nr_unknown") %>%
  arrange(-nr_unknown)
```

```
## # A tibble: 21 x 2
##    variable    nr_unknown
##    <chr>            <int>
##  1 default           8597
##  2 education         1731
##  3 housing            990
##  4 loan               990
##  5 job                330
##  6 marital             80
##  7 age                  0
##  8 contact              0
##  9 month                0
## 10 day_of_week          0
## # ... with 11 more rows
```

```
# Analyse default
table(bank_df$default)
```

```
## 
##      no unknown     yes 
##   32588    8597       3
```

```
## This is not usable, too few "yes" to evaluate
```

analyse the unknown values

```
# setting default parameters for crosstables
# fun_crosstable = function(df, var1, var2){
#   # df: dataframe containing both columns to cross
#   # var1, var2: columns to cross together.
#   CrossTable(df$var1, df$var2,
#              prop.r = T,
#              prop.c = F,
#              prop.t = F,
#              prop.chisq = F,
#              dnn = c(var1, var2)) # dimension names
# }

#default
CrossTable(bank_df$default, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("default", "y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      default |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##           no |     28391 |      4197 |     32588 |
##              |     0.871 |     0.129 |     0.791 |
##              |     0.689 |     0.102 |           |
## -------------|-----------|-----------|-----------|
##      unknown |      8154 |       443 |      8597 |
##              |     0.948 |     0.052 |     0.209 |
##              |     0.198 |     0.011 |           |
## -------------|-----------|-----------|-----------|
##          yes |         3 |         0 |         3 |
##              |     1.000 |     0.000 |     0.000 |
##              |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

```
table(bank_df$default)
```

```
##
##      no unknown     yes
##   32588    8597       3
```

```
# job
CrossTable(bank_df$job, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("job", "y"))
```

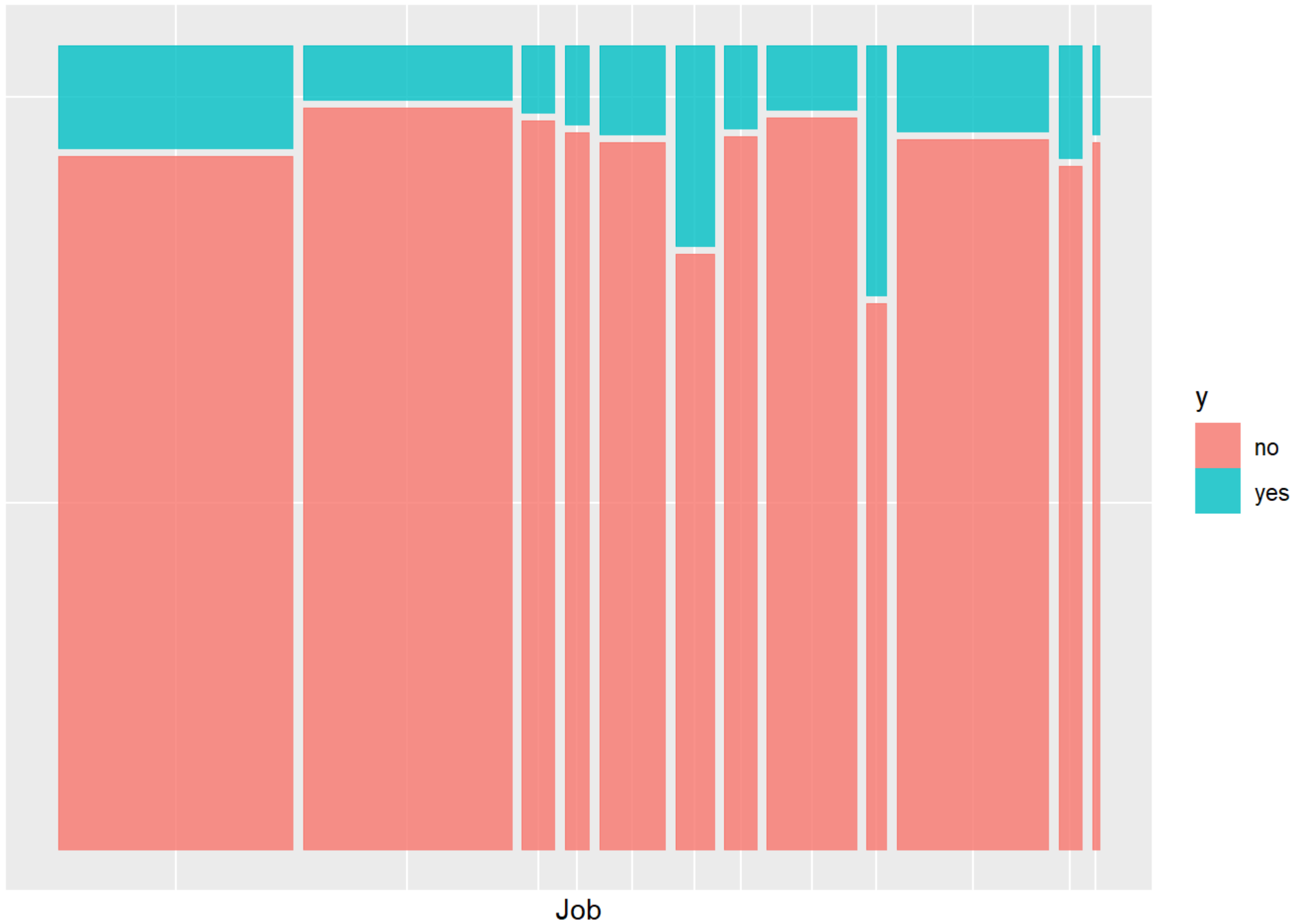
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##               | y
##           job |        no |       yes | Row Total |
## --------------|-----------|-----------|-----------|
##        admin. |      9070 |      1352 |     10422 |
##               |     0.870 |     0.130 |     0.253 |
##               |     0.220 |     0.033 |           |
## --------------|-----------|-----------|-----------|
##   blue-collar |      8616 |       638 |      9254 |
##               |     0.931 |     0.069 |     0.225 |
##               |     0.209 |     0.015 |           |
## --------------|-----------|-----------|-----------|
##  entrepreneur |      1332 |       124 |      1456 |
##               |     0.915 |     0.085 |     0.035 |
##               |     0.032 |     0.003 |           |
## --------------|-----------|-----------|-----------|
##     housemaid |       954 |       106 |      1060 |
##               |     0.900 |     0.100 |     0.026 |
##               |     0.023 |     0.003 |           |
## --------------|-----------|-----------|-----------|
##    management |      2596 |       328 |      2924 |
##               |     0.888 |     0.112 |     0.071 |
##               |     0.063 |     0.008 |           |
## --------------|-----------|-----------|-----------|
##       retired |      1286 |       434 |      1720 |
##               |     0.748 |     0.252 |     0.042 |
##               |     0.031 |     0.011 |           |
## --------------|-----------|-----------|-----------|
## self-employed |      1272 |       149 |      1421 |
##               |     0.895 |     0.105 |     0.035 |
##               |     0.031 |     0.004 |           |
## --------------|-----------|-----------|-----------|
##      services |      3646 |       323 |      3969 |
##               |     0.919 |     0.081 |     0.096 |
##               |     0.089 |     0.008 |           |
## --------------|-----------|-----------|-----------|
##       student |       600 |       275 |       875 |
##               |     0.686 |     0.314 |     0.021 |
##               |     0.015 |     0.007 |           |
## --------------|-----------|-----------|-----------|
##    technician |      6013 |       730 |      6743 |
```

```
##                |     0.892 |     0.108 |     0.164 |
##                |     0.146 |     0.018 |           |
## --------------|-----------|-----------|-----------|
##    unemployed |       870 |       144 |      1014 |
##               |     0.858 |     0.142 |     0.025 |
##               |     0.021 |     0.003 |           |
## --------------|-----------|-----------|-----------|
##       unknown |       293 |        37 |       330 |
##               |     0.888 |     0.112 |     0.008 |
##               |     0.007 |     0.001 |           |
## --------------|-----------|-----------|-----------|
##  Column Total |     36548 |      4640 |     41188 |
## --------------|-----------|-----------|-----------|
##
##
```

```
table(bank_df$job)
```

```
##
##        admin.   blue-collar  entrepreneur     housemaid    management
##         10422          9254          1456          1060          2924
##       retired self-employed      services       student    technician
##          1720          1421          3969           875          6743
##    unemployed       unknown
##          1014           330
```

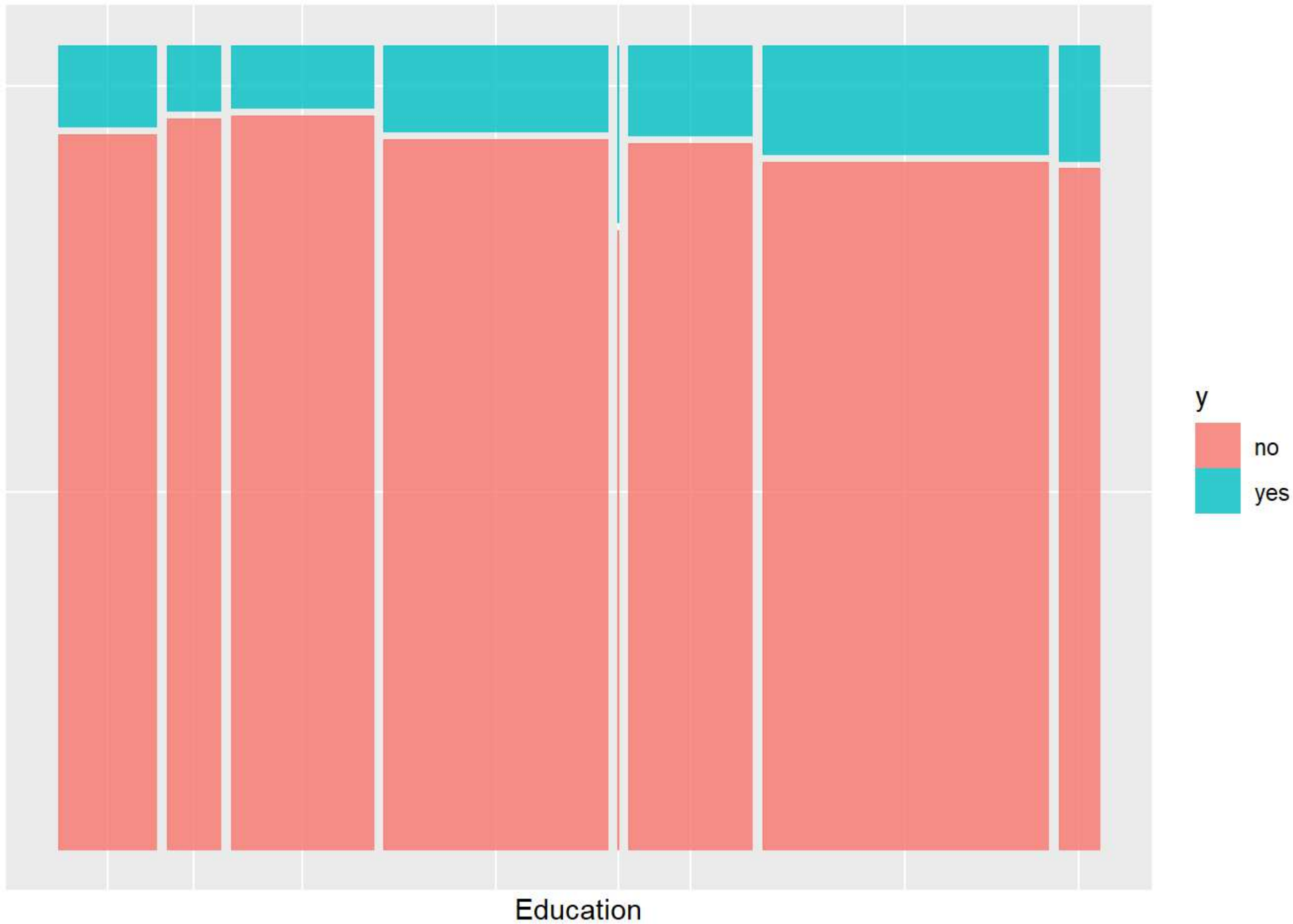
```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, job), fill = y)) +
  #mosaic_theme +
  xlab("Job") +
  ylab(NULL)
```




```
bank_df <- bank_df %>%
  mutate(job = recode(job, "unknown" = "unconventional"))


# marital
CrossTable(bank_df$marital, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("marital",
"y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      marital |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##     divorced |      4136 |       476 |      4612 |
##              |     0.897 |     0.103 |     0.112 |
##              |     0.100 |     0.012 |           |
## -------------|-----------|-----------|-----------|
##      married |     22396 |      2532 |     24928 |
##              |     0.898 |     0.102 |     0.605 |
##              |     0.544 |     0.061 |           |
## -------------|-----------|-----------|-----------|
##       single |      9948 |      1620 |     11568 |
##              |     0.860 |     0.140 |     0.281 |
##              |     0.242 |     0.039 |           |
## -------------|-----------|-----------|-----------|
##      unknown |        68 |        12 |        80 |
##              |     0.850 |     0.150 |     0.002 |
##              |     0.002 |     0.000 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

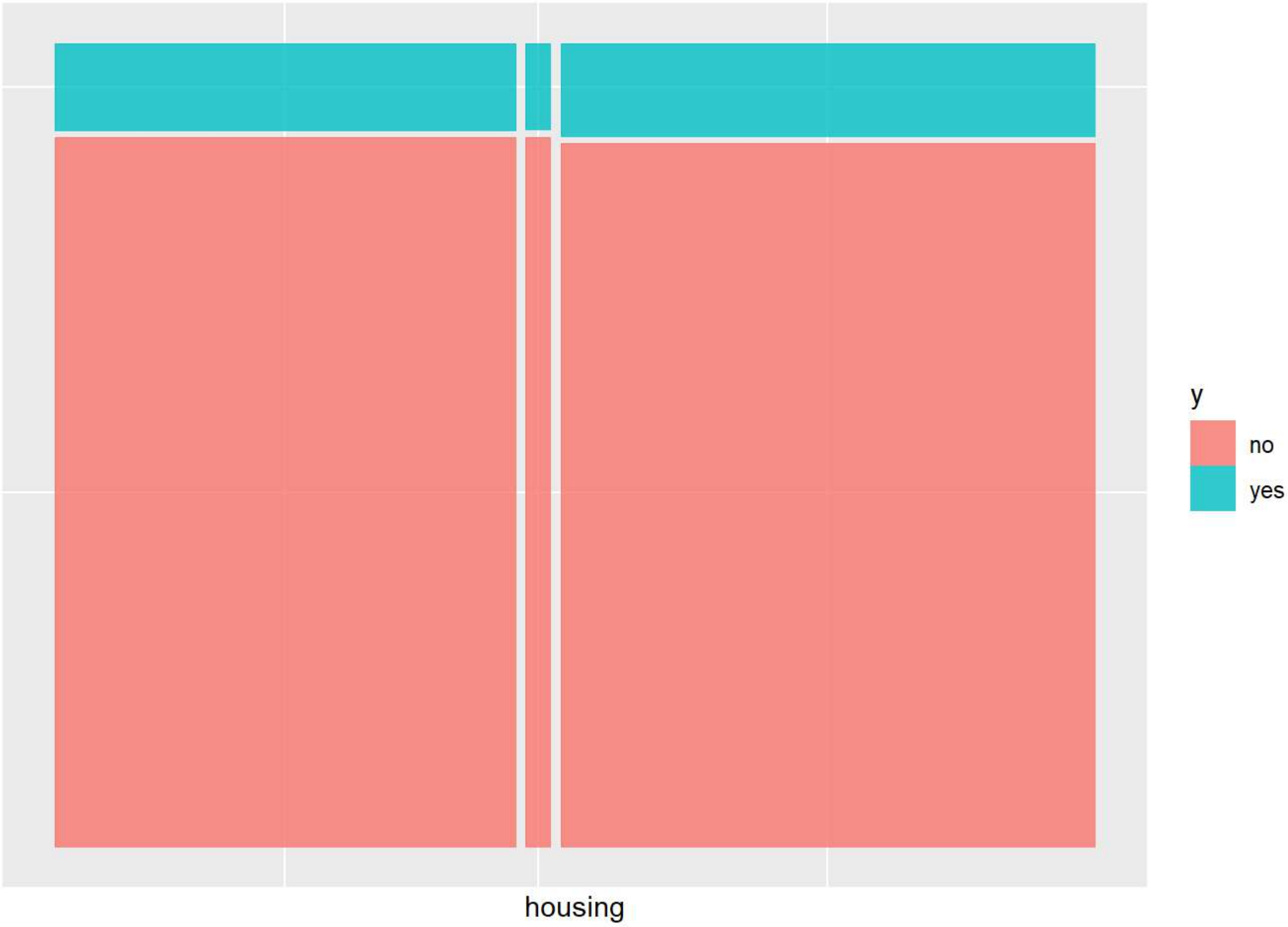
```
## can merge single+unknown, married+divorced since values are similar
bank_df = bank_df %>%
  mutate(marital = recode(marital, "unknown" = "single", "divorced"="married"))

# education
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, education),  fill = y)) +
  #mosaic_theme +
  xlab("Education") +
  ylab(NULL)
```




```
    ## recode unknown as univeristy degree because proportions are similar
bank_df = bank_df %>%
  mutate(education = recode(education, "unknown" = "university.degree"))

# housing
CrossTable(bank_df$housing, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("housing",
"y"))
```

```
## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
## 
## 
## Total Observations in Table:  41188 
## 
## 
##              | y 
##      housing |        no |       yes | Row Total | 
## -------------|-----------|-----------|-----------|
##           no |     16596 |      2026 |     18622 | 
##              |     0.891 |     0.109 |     0.452 | 
##              |     0.403 |     0.049 |           | 
## -------------|-----------|-----------|-----------|
##      unknown |       883 |       107 |       990 | 
##              |     0.892 |     0.108 |     0.024 | 
##              |     0.021 |     0.003 |           | 
## -------------|-----------|-----------|-----------|
##          yes |     19069 |      2507 |     21576 | 
##              |     0.884 |     0.116 |     0.524 | 
##              |     0.463 |     0.061 |           | 
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 | 
## -------------|-----------|-----------|-----------|
## 
## 
```

```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, housing),  fill = y)) +
  #mosaic_theme +
  xlab("housing") +
  ylab(NULL)
```



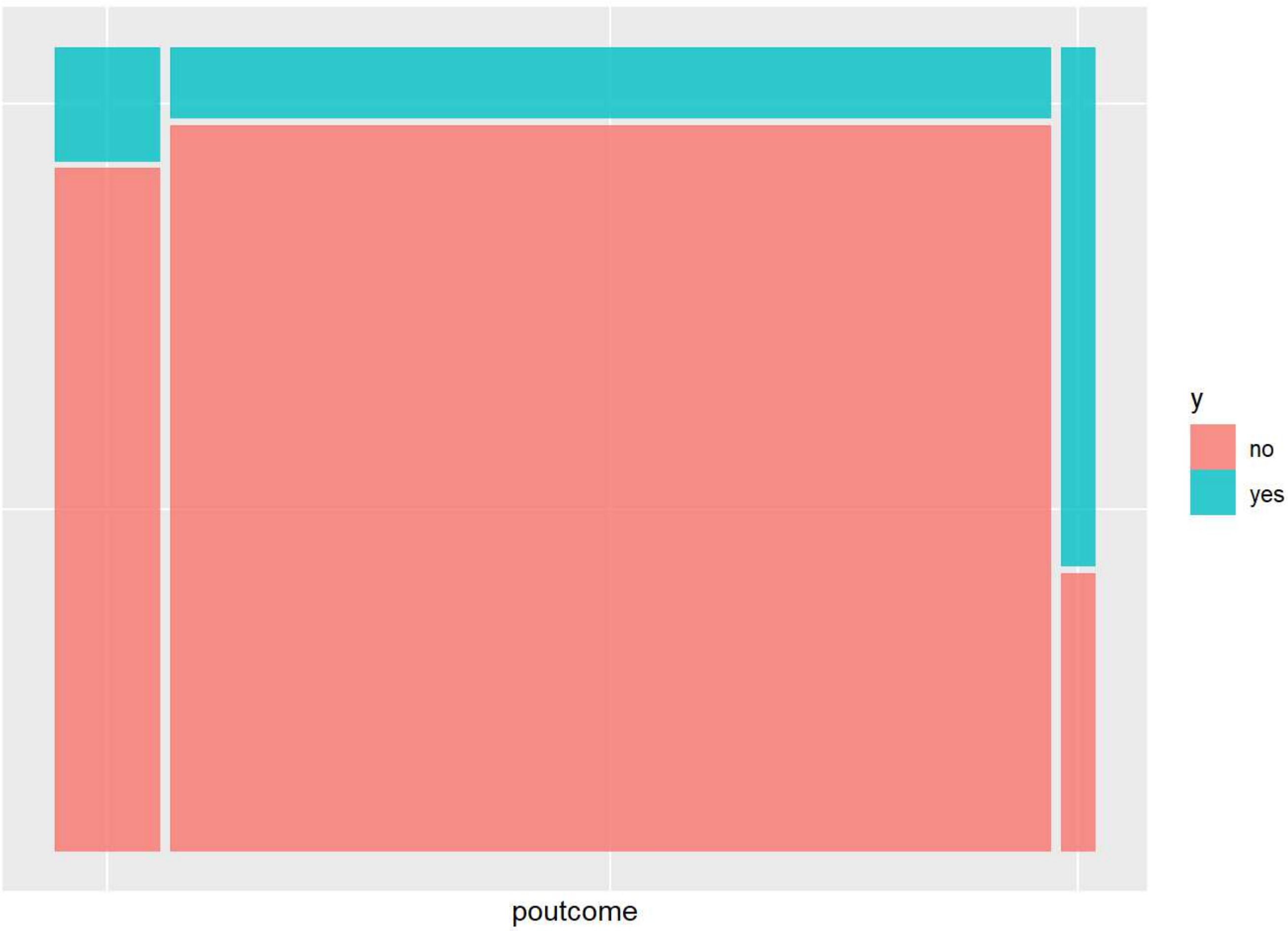

```
## the plot looks very similar, do chisquared test to see if there are differences
chisq.test(bank_df$housing, bank_df$y) # drop this column
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$housing and bank_df$y
## X-squared = 5.6845, df = 2, p-value = 0.05829
```

```
bank_df$housing <- NULL

# loan
chisq.test(bank_df$loan, bank_df$y) # drop col, pvalue >0.1
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$loan and bank_df$y
## X-squared = 1.094, df = 2, p-value = 0.5787
```

```
bank_df$loan <- NULL

# pdays
# poutcome
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, poutcome),  fill = y)) +
  #mosaic_theme +
  xlab("poutcome") +
  ylab(NULL)
```




```
bank_df = bank_df %>%
  mutate(past_dummyvar = recode(poutcome, "failure" = 0.5, "nonexistent"=0.2, "success"=1))
# combining previous and poutcome
bank_df$past_dummyvar1 = bank_df$past_dummyvar*(bank_df$previous+1)
chisq.test(bank_df$past_dummyvar1, bank_df$y)
```

```
## Warning in chisq.test(bank_df$past_dummyvar1, bank_df$y): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$past_dummyvar1 and bank_df$y
## X-squared = 4383.4, df = 11, p-value < 2.2e-16
```

```
bank_df$previous <-NULL
bank_df$poutcome <-NULL
bank_df$past_dummyvar <-NULL

bank_df = bank_df %>%
  mutate(pdays_dummy = if_else(pdays == 999, "0", "1")) %>%
  select(-pdays)
bank_df$pdays<-NULL

#resolve default, let yes become unknown
bank_df = bank_df %>%
  mutate(default = recode(default, "yes"="unknown"))


# dayofweek
bank_df = bank_df %>%
  mutate(day_of_week = recode(day_of_week, "mon"=1, "tue"=2,"wed"=3,"thu"=4,"fri"=5))

# age
bank_df = bank_df %>%
  mutate(age = if_else(
    age<20, 1, if_else(
      age<23, 2, if_else(
        age<26, 3, if_else(
          age<31, 4, if_else(
            age<41, 5, if_else(age<51, 6, 7)))))))

#dataset after preprocessing
dim(bank_df)
```

```
## [1] 41188    18
```

```
summary(bank_df)
```

```
##       age                  job           marital
##  Min.   :1.000   admin.     :10422   married:29540
##  1st Qu.:5.000   blue-collar: 9254   single :11648
##  Median :5.000   technician : 6743
##  Mean   :5.367   services   : 3969
##  3rd Qu.:6.000   management : 2924
##  Max.   :7.000   retired    : 1720
##                  (Other)    : 6156
##                education        default           contact          month
##  basic.4y           : 4176   no     :32588   cellular :26144   may    :13769
##  basic.6y           : 2292   unknown: 8600   telephone:15044   jul    : 7174
##  basic.9y           : 6045                                     aug    : 6178
##  high.school        : 9515                                     jun    : 5318
##  illiterate         :   18                                     nov    : 4101
##  professional.course: 5243                                     apr    : 2632
##  university.degree  :13899                                     (Other): 2016
##   day_of_week      duration         campaign       emp.var.rate
##  Min.   :1.00   Min.   :   0.0   Min.   : 1.000   Min.   :-3.40000
##  1st Qu.:2.00   1st Qu.: 102.0   1st Qu.: 1.000   1st Qu.:-1.80000
##  Median :3.00   Median : 180.0   Median : 2.000   Median : 1.10000
##  Mean   :2.98   Mean   : 258.3   Mean   : 2.568   Mean   : 0.08189
##  3rd Qu.:4.00   3rd Qu.: 319.0   3rd Qu.: 3.000   3rd Qu.: 1.40000
##  Max.   :5.00   Max.   :4918.0   Max.   :56.000   Max.   : 1.40000
##
##  cons.price.idx  cons.conf.idx     euribor3m      nr.employed     y
##  Min.   :92.20   Min.   :-50.8   Min.   :0.634   Min.   :4964   no :36548
##  1st Qu.:93.08   1st Qu.:-42.7   1st Qu.:1.344   1st Qu.:5099   yes: 4640
##  Median :93.75   Median :-41.8   Median :4.857   Median :5191
##  Mean   :93.58   Mean   :-40.5   Mean   :3.621   Mean   :5167
##  3rd Qu.:93.99   3rd Qu.:-36.4   3rd Qu.:4.961   3rd Qu.:5228
##  Max.   :94.77   Max.   :-26.9   Max.   :5.045   Max.   :5228
##
##  past_dummyvar1   pdays_dummy
##  Min.   :0.2000   Length:41188
##  1st Qu.:0.2000   Class :character
##  Median :0.2000   Mode  :character
##  Mean   :0.3703
##  3rd Qu.:0.2000
##  Max.   :8.0000
##
```

```
#Standardize the numeric features
num.ind <- sapply(bank_df, is.numeric)
bank_df.mean <- apply(bank_df[,num.ind], 2, mean)
bank_df.sd <- apply(bank_df[,num.ind], 2, sd)

bank_df.scaled <- bank_df

bank_df.scaled[,num.ind] <- scale(bank_df[,num.ind], center=bank_df.mean, scale=bank_df.sd)
```

```
# splitting train and test
library(caTools)
set.seed(1)
smp_size <- floor(0.8*nrow(bank_df.scaled))
train_ind <- sample(seq_len(nrow(bank_df.scaled)), size = smp_size)
train <- bank_df.scaled[train_ind, ]
test <- bank_df.scaled[-train_ind, ]
```

KNN

```
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1 tim
es

# knn.fit <- train(y ~., data = train, method = "knn",
#                   trControl=trctrl, tuneLength = 10) # tuneLength parameter tells the algorithm
to try different default values for the main parameter

knn.fit <- train(y ~., data = train, method = "knn",
                 trControl=trctrl) # tuneLength parameter tells the algorithm to try different d
efault values for the main parameter

# knn.fit <- train(y ~., data = train, method = "knn")#by default bootstrap is used to find tuni
ng parameter -> trCtrl
knn.fit
```

```
## k-Nearest Neighbors
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.8994540  0.4346305
##   7  0.9020639  0.4392127
##   9  0.9047954  0.4458783
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```r
#predict using test data
knn.pred <- predict(knn.fit, newdata = test)
#knn.pred

#confusion matrix
cm.knn <- table(knn.pred, test$y)
cm.knn
```

```
##
## knn.pred   no  yes
##      no  7076  521
##      yes  225  416
```

```r
TP <- cm.knn[2,2]
TN <- cm.knn[1,1]
FP <- cm.knn[2,1]
FN <- cm.knn[1,2]

#FPR / Type I error
FPR.knn = FP/(FP+TN)
FPR.knn
```

```
## [1] 0.0308177
```

```r
#FNR / Type II error
FNR.knn = FN/(FN+TP)
FNR.knn
```

```
## [1] 0.5560299
```

```r
#Precision
precis.knn = TP/(TP+FP)
precis.knn
```

```
## [1] 0.648986
```

```r
#Recall / sensitivity
recall.knn = TP/(TP+FN)
recall.knn
```

```
## [1] 0.4439701
```

```r
#misclassification error
test.err.knn = 1-(sum(diag(cm.knn))/sum(cm.knn))
test.err.knn
```

```
## [1] 0.09055596
```

Logistic Regression

```
set.seed(8)
glm.fit <- glm(y ~., data = train, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.9280  -0.3025  -0.1891  -0.1390   3.2894
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -2.750785   0.134765 -20.412  < 2e-16 ***
## age                          -0.055930   0.027213  -2.055 0.039852 *
## jobblue-collar               -0.246842   0.087127  -2.833 0.004609 **
## jobentrepreneur              -0.145059   0.138872  -1.045 0.296230
## jobhousemaid                  0.039898   0.162792   0.245 0.806388
## jobmanagement                -0.028385   0.094597  -0.300 0.764129
## jobretired                    0.352578   0.106563   3.309 0.000938 ***
## jobself-employed             -0.150559   0.132842  -1.133 0.257060
## jobservices                  -0.127794   0.093575  -1.366 0.172040
## jobstudent                    0.059646   0.124981   0.477 0.633192
## jobtechnician                 0.006576   0.078381   0.084 0.933134
## jobunemployed                -0.089351   0.145955  -0.612 0.540417
## jobunconventional            -0.043046   0.275330  -0.156 0.875763
## maritalsingle                -0.031908   0.057275  -0.557 0.577456
## educationbasic.6y             0.037662   0.136393   0.276 0.782449
## educationbasic.9y             0.045684   0.105675   0.432 0.665517
## educationhigh.school          0.055489   0.101846   0.545 0.585865
## educationilliterate           1.558755   0.740636   2.105 0.035325 *
## educationprofessional.course  0.101641   0.112486   0.904 0.366216
## educationuniversity.degree    0.151933   0.098805   1.538 0.124123
## defaultunknown               -0.295427   0.074573  -3.962 7.45e-05 ***
## contacttelephone             -0.630566   0.085678  -7.360 1.84e-13 ***
## monthaug                      0.720398   0.133252   5.406 6.43e-08 ***
## monthdec                      0.330156   0.229521   1.438 0.150305
## monthjul                      0.078329   0.106239   0.737 0.460947
## monthjun                     -0.472526   0.140064  -3.374 0.000742 ***
## monthmar                      1.945546   0.159750  12.179  < 2e-16 ***
## monthmay                     -0.536562   0.091073  -5.892 3.83e-09 ***
## monthnov                     -0.565715   0.134703  -4.200 2.67e-05 ***
## monthoct                      0.055455   0.170805   0.325 0.745431
## monthsep                      0.286532   0.199121   1.439 0.150154
## day_of_week                   0.033569   0.022596   1.486 0.137376
## duration                      1.193618   0.021334  55.950  < 2e-16 ***
## campaign                     -0.108741   0.035285  -3.082 0.002057 **
## emp.var.rate                 -2.620715   0.248940 -10.527  < 2e-16 ***
## cons.price.idx                1.175153   0.162518   7.231 4.80e-13 ***
## cons.conf.idx                 0.090009   0.039579   2.274 0.022956 *
## euribor3m                     0.668129   0.248996   2.683 0.007290 **
## nr.employed                   0.270856   0.250083   1.083 0.278780
## past_dummyvar1               -0.106698   0.027802  -3.838 0.000124 ***
## pdays_dummy1                  1.854235   0.131958  14.052  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23162  on 32949  degrees of freedom
## Residual deviance: 13829  on 32909  degrees of freedom
## AIC: 13911
##
## Number of Fisher Scoring iterations: 6
```

```
#predict using test data
glm.prob <- predict(glm.fit, type = "response", newdata = test)

#check which one is 'Yes'
contrasts(test$y)#Yes = 1, Low = 0
```

```
##     yes
## no    0
## yes   1
```

```
glm.pred <- rep('no', nrow(test))
glm.pred[glm.prob > 0.5] <- 'yes' #yes = 1, no = 0

#confusion matrix
cm.reg = table(glm.pred, test$y)
cm.reg
```

```
##
## glm.pred   no  yes
##      no  7111  546
##      yes  190  391
```

```
TP <- cm.reg[2,2]
TN <- cm.reg[1,1]
FP <- cm.reg[2,1]
FN <- cm.reg[1,2]

#FPR / Type I error
FPR.reg = FP/(FP+TN)
FPR.reg
```

```
## [1] 0.02602383
```

```
#FNR / Type II error
FNR.reg = FN/(FN+TP)
FNR.reg
```

```
## [1] 0.5827108
```

```
#Precision
precis.reg = TP/(TP+FP)
precis.reg
```

```
## [1] 0.6729776
```

```
#Recall / sensitivity
recall.reg = TP/(TP+FN)
recall.reg
```

```
## [1] 0.4172892
```

```
#misclassification error
test.err.reg = 1-(sum(diag(cm.reg))/sum(cm.reg))
test.err.reg
```

```
## [1] 0.08934207
```

Decision Tree

```
# train.tree <- data.frame(train)
# test.tree <-data.frame(test)

#Decision Tree using rpart()
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1 tim
es

cls.tree1 = train(y ~ ., data=train, method="rpart",
                  trControl=trctrl)# tuneLength parameter tells the algorithm to try different d
efault values for the main parameter

# cls.tree1 = train(y ~ ., data=train.tree, method="rpart")#by default bootstrap is used to find
tuning parameter -> trCtrl

cls.tree1
```

```
## CART
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01876857  0.9085282  0.4758751
##   0.02106400  0.9058881  0.4220528
##   0.07061842  0.8964786  0.2597907
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01876857.
```

```
#plot(cls.tree1$finalModel)
#text(cls.tree1$finalModel)

#predict using test data
tree.pred1 <- predict(cls.tree1, newdata = test)
#tree.pred1

#confusion matrix
cm.tree1 <- table(tree.pred1, test$y)
cm.tree1
```

```
##
## tree.pred1   no  yes
##        no  7145  612
##        yes  156  325
```

```
TP <- cm.tree1[2,2]
TN <- cm.tree1[1,1]
FP <- cm.tree1[2,1]
FN <- cm.tree1[1,2]

#FPR / Type I error
FPR.tree1 = FP/(FP+TN)
FPR.tree1
```

```
## [1] 0.02136694
```

```
#FNR / Type II error
FNR.tree1 = FN/(FN+TP)
FNR.tree1
```

```
## [1] 0.6531483
```

```
#Precision
precis.tree1 = TP/(TP+FP)
precis.tree1
```

```
## [1] 0.6756757
```

```
#Recall / sensitivity
recall.tree1 = TP/(TP+FN)
recall.tree1
```

```
## [1] 0.3468517
```

```
#misclassification error
test.err.tree1 = 1-(sum(diag(cm.tree1))/sum(cm.tree1))
test.err.tree1
```

```
## [1] 0.09322651
```

Random Forest

```
#Random forest with 500 bootstrapped trees
#p = 16
sqrt(16) # ntree = 4
```

```
## [1] 4
```

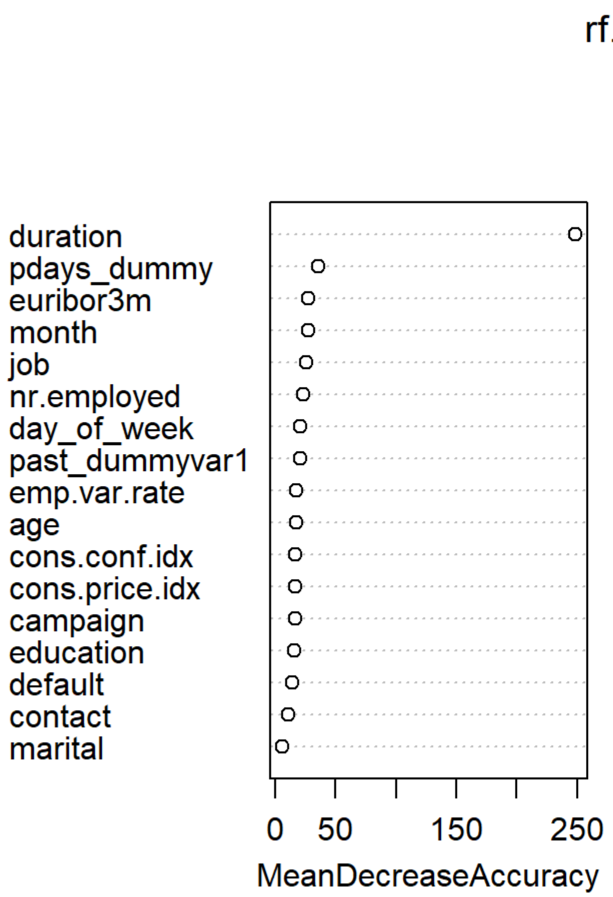
```
set.seed(8)
rf.cls <- randomForest(y ~ .,
                       data = train,
                       mtry = 4,
                       ntree = 500,
                       importance = TRUE)
rf.cls
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train, mtry = 4, ntree = 500,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 8.58%
## Confusion matrix:
##        no  yes class.error
## no  28159 1088   0.0372004
## yes  1739 1964   0.4696192
```

```
#ls(rf.cls)
importance(rf.cls)
```

```
##                         no         yes MeanDecreaseAccuracy MeanDecreaseGini
## age             17.5994852   2.6431157             17.17497        261.27791
## job             33.4598069  -8.3279418             25.53862        472.00539
## marital          8.4804094  -2.0770280              5.63656         90.42659
## education       18.2285426   0.2902509             15.85282        296.38671
## default          7.9851672   9.9466921             14.06632         57.38234
## contact          5.8083405  31.6780656             10.75700         65.39431
## month           26.1934824   5.7403113             27.21203        201.40406
## day_of_week     19.7355665   7.1906377             20.93292        256.55362
## duration       152.0502038 246.1841391            248.91491       2063.97786
## campaign         9.1802979  14.2948341             16.52478        270.40819
## emp.var.rate    16.4877524   7.0310918             17.40129        124.77060
## cons.price.idx  16.6662412  -3.0556567             16.77137        121.28844
## cons.conf.idx   15.9713846   3.4665630             16.86955        147.71796
## euribor3m       24.9134347  13.3710567             27.65883        709.60922
## nr.employed     19.1091313  23.1302593             23.16106        444.41888
## past_dummyvar1  10.4070327  24.5936232             20.68765        239.75494
## pdays_dummy      0.5297726  55.5552773             36.07905        194.92174
```

```
varImpPlot(rf.cls)
```




```
#predict using test data
rf.pred <- predict(rf.cls, newdata = test, type = "class")
#rf.pred

#confusion matrix
cm.rf <- table(rf.pred, test$y)
cm.rf
```

```
##
## rf.pred   no  yes
##      no 7035  425
##      yes 266  512
```

```
TP <- cm.rf[2,2]
TN <- cm.rf[1,1]
FP <- cm.rf[2,1]
FN <- cm.rf[1,2]

#FPR / Type I error
FPR.rf = FP/(FP+TN)
FPR.rf
```

```
## [1] 0.03643337
```

```
#FNR / Type II error
FNR.rf = FN/(FN+TP)
FNR.rf
```

```
## [1] 0.4535752
```

```
#Precision
precis.rf = TP/(TP+FP)
precis.rf
```

```
## [1] 0.6580977
```

```
#Recall / sensitivity
recall.rf = TP/(TP+FN)
recall.rf
```

```
## [1] 0.5464248
```

```
#misclassification error
test.err.rf = 1-(sum(diag(cm.rf))/sum(cm.rf))
test.err.rf
```

```
## [1] 0.08387958
```

Gradient Boosting

```
#Gradient boosting
set.seed(8)

#Use K-fold CV to find best trControl
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 1) #5 folds repeated 1 times

gbm.fit <- train(y ~ ., data = train,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)

# gbm.fit <- train(y ~ ., data = train,
#                  method = "gbm",
#                  verbose = FALSE) #by default bootstrap is used to find tuning parameter -> tr
Ctrl
gbm.fit
```

```
## Stochastic Gradient Boosting
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9054326  0.3425904
##   1                  100      0.9085587  0.4064470
##   1                  150      0.9098333  0.4385889
##   2                   50      0.9091959  0.4340753
##   2                  100      0.9124735  0.4971532
##   2                  150      0.9133840  0.5092936
##   3                   50      0.9117148  0.4927134
##   3                  100      0.9135356  0.5149861
##   3                  150      0.9157512  0.5309642
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
#predict using test data
gbm.pred <- predict(gbm.fit, newdata = test)
#gbm.pred

#confusion matrix
cm.gbm <- table(gbm.pred, test$y)
cm.gbm
```

```
##
## gbm.pred   no  yes
##      no  7055  429
##      yes  246  508
```

```
TP <- cm.gbm[2,2]
TN <- cm.gbm[1,1]
FP <- cm.gbm[2,1]
FN <- cm.gbm[1,2]

#FPR / Type I error
FPR.gbm = FP/(FP+TN)
FPR.gbm
```

```
## [1] 0.03369401
```

```
#FNR / Type II error
FNR.gbm = FN/(FN+TP)
FNR.gbm
```

```
## [1] 0.4578442
```

```
#Precision
precis.gbm = TP/(TP+FP)
precis.gbm
```

```
## [1] 0.6737401
```

```
#Recall / sensitivity
recall.gbm = TP/(TP+FN)
recall.gbm
```

```
## [1] 0.5421558
```

```
#misclassification error
test.err.gbm = 1-(sum(diag(cm.gbm))/sum(cm.gbm))
test.err.gbm
```

```
## [1] 0.08193736
```

AdaBoost

```
#AdaBoost
set.seed(8)
x.trainA = model.matrix(data=train, y~.-1)
y.trainA = rep(1, nrow(train))
y.trainA [train$y=="no"]=-1 #for Adaboost

x.testA = model.matrix(data=test, y~.-1)
y.testA = rep(1, nrow(test))
y.testA [test$y=="no"]=-1 #for Adaboost

ada.cls <- adaboost(x.trainA, y.trainA, tree_depth=5, n_rounds=500)
ada.cls
```

```
## AdaBoost: tree_depth =  5  rounds =  500
##
##
##  In-sample confusion matrix:
##     yhat
## y        -1     1
##   -1 28415   832
##   1   1430  2273
```

```
#predict using test data
ada.pred <- predict(ada.cls, x.testA)
#ada.pred

#confusion matrix
cm.ada <- table(ada.pred, y.testA) #-1 is "no", 1 is "yes"
cm.ada
```

```
##         y.testA
## ada.pred   -1    1
##       -1 6998  435
##       1   303  502
```

```
TP <- cm.ada[2,2]
TN <- cm.ada[1,1]
FP <- cm.ada[2,1]
FN <- cm.ada[1,2]

#FPR / Type I error
FPR.ada = FP/(FP+TN)
FPR.ada
```

```
## [1] 0.04150116
```

```
#FNR / Type II error
FNR.ada = FN/(FN+TP)
FNR.ada
```

```
## [1] 0.4642476
```

```
#Precision
precis.ada = TP/(TP+FP)
precis.ada
```

```
## [1] 0.6236025
```

```
#Recall / sensitivity
recall.ada = TP/(TP+FN)
recall.ada
```

```
## [1] 0.5357524
```

```
#misclassification error
test.err.ada = 1-(sum(diag(cm.ada))/sum(cm.ada))
test.err.ada
```

```
## [1] 0.08958485
```

XGBoost

```
#XGBoost
set.seed(8)
x.trainXG =model.matrix(data=train,y~.-1)
y.trainXG = rep(1, nrow(train))
y.trainXG[train$y=="no"]=0 #for XGBoost

x.testXG = model.matrix(data=test, y~.-1)
y.testXG = rep(1, nrow(test))
y.testXG[test$y=="no"]=0 #for XGBoost

xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=5,eta=0.01,nrounds=500,verbose=FALSE
)
#xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=10,nrounds=500,verbose=FALSE)
xgb.cls
```

```
## ##### xgb.Booster
## raw: 1 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 5, eta = 0.01)
## params (as set within xgb.train):
##   max_depth = "5", eta = "0.01", validate_parameters = "1"
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 41
## niter: 500
## nfeatures : 41
## evaluation_log:
##     iter train_rmse
##        1   0.496162
##        2   0.492358
## ---
##      499   0.225786
##      500   0.225769
```

```
#predict using test data
xgb.pred.prob<-predict(xgb.cls,x.testXG)

xgb.pred<-as.numeric(xgb.pred.prob>0.5) #convert to 0 ("no") or 1 ("yes")

#confusion matrix
cm.xgb<-table(xgb.pred,y.testXG) #0 is "no", 1 is "yes"
cm.xgb
```

```
##         y.testXG
## xgb.pred    0    1
##        0 7057  428
##        1  244  509
```

```
TP <- cm.xgb[2,2]
TN <- cm.xgb[1,1]
FP <- cm.xgb[2,1]
FN <- cm.xgb[1,2]

#FPR / Type I error
FPR.xgb = FP/(FP+TN)
FPR.xgb
```

```
## [1] 0.03342008
```

```
#FNR / Type II error
FNR.xgb = FN/(FN+TP)
FNR.xgb
```

```
## [1] 0.4567769
```

```
#Precision
precis.xgb = TP/(TP+FP)
precis.xgb
```

```
## [1] 0.6759628
```

```
#Recall / sensitivity
recall.xgb = TP/(TP+FN)
recall.xgb
```

```
## [1] 0.5432231
```

```
#misclassification error
test.err.xgb = 1-sum(diag(cm.xgb))/sum(cm.xgb)
test.err.xgb
```

```
## [1] 0.0815732
```

SVM with linear kernel

```
set.seed(8)
svm.fit <- svm(y~., data=train, kernel='linear', cost=1)
#summary(svm.fit)

#CV for tuning the cost parameter
set.seed(8)
tune.out1 <- tune(svm, y~.,
              data=train,
              kernel="linear",
              )

#tune.out1 <- tune(svm, y~.,
#              data=train,
#              kernel="linear",
#              ranges=list(cost=c(0.01, 0.1, 1, 10, 100)),tunecontrol=tune.control(cross=10))
summary(tune.out1)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09729894
```

```
svm.lin.best <- tune.out1$best.model
summary(svm.lin.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  6635
##
##  ( 3324 3311 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
lin.pred <- predict(svm.lin.best, test)

#confusion matrix
cm.lin <- table(lin.pred, test$y)
cm.lin
```

```
##
## lin.pred   no  yes
##      no  7146  646
##      yes  155  291
```

```
TP <- cm.lin[2,2]
TN <- cm.lin[1,1]
FP <- cm.lin[2,1]
FN <- cm.lin[1,2]

#FPR / Type I error
FPR.lin = FP/(FP+TN)
FPR.lin
```

```
## [1] 0.02122997
```

```
#FNR / Type II error
FNR.lin = FN/(FN+TP)
FNR.lin
```

```
## [1] 0.6894344
```

```
#Precision
precis.lin = TP/(TP+FP)
precis.lin
```

```
## [1] 0.6524664
```

```
#Recall / sensitivity
recall.lin = TP/(TP+FN)
recall.lin
```

```
## [1] 0.3105656
```

```
#misclassification error
test.err.lin = 1-(sum(diag(cm.lin))/sum(cm.lin))
test.err.lin
```

```
## [1] 0.09723234
```

SVM with polynomial kernel

```
set.seed(8)
tune.out2 <- tune(svm, y~.,
              data=train,
              kernel="polynomial",
              )

#tune.out2 <- tune(svm, y~.,
#               data=train,
#               kernel="polynomial",
#               ranges=list(cost=c(0.1,1,5,10,15,20,50,100),
#                           degree=c(2,3,4)))
summary(tune.out2)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09456753
```

```
svm.poly.best <- tune.out2$best.model
summary(svm.poly.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  6712
##
##  ( 3399 3313 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
poly.pred <- predict(svm.poly.best, test)

#confusion matrix
cm.poly <- table(poly.pred, test$y)
cm.poly
```

```
##
## poly.pred   no  yes
##       no  7214  691
##       yes   87  246
```

```
TP <- cm.poly[2,2]
TN <- cm.poly[1,1]
FP <- cm.poly[2,1]
FN <- cm.poly[1,2]

#FPR / Type I error
FPR.poly = FP/(FP+TN)
FPR.poly
```

```
## [1] 0.01191618
```

```
#FNR / Type II error
FNR.poly = FN/(FN+TP)
FNR.poly
```

```
## [1] 0.73746
```

```
#Precision
precis.poly = TP/(TP+FP)
precis.poly
```

```
## [1] 0.7387387
```

```
#Recall / sensitivity
recall.poly = TP/(TP+FN)
recall.poly
```

```
## [1] 0.26254
```

```
#misclassification error
test.err.poly = 1-(sum(diag(cm.poly))/sum(cm.poly))
test.err.poly
```

```
## [1] 0.0944404
```

SVM with rbf kernel

```
set.seed(8)
tune.out3 <- tune(svm, y~.,
              data=train,
              kernel="radial",)

#tune.out3 <- tune(svm, y~.,
#              data=train,
#              kernel="radial",
#              ranges=list(cost=c(0.1,1,5,10),
#                          gamma=c(0.01,0.1,1,5,10,100)))
summary(tune.out3)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09125948
```

```
svm.rbf.best <- tune.out3$best.model
summary(svm.rbf.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  6573
##
##  ( 3334 3239 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
rbf.pred <- predict(svm.rbf.best, test)

#confusion matrix
cm.rbf <- table(rbf.pred, test$y)
cm.rbf
```

```
##
## rbf.pred   no  yes
##      no  7161  614
##      yes  140  323
```

```
TP <- cm.rbf[2,2]
TN <- cm.rbf[1,1]
FP <- cm.rbf[2,1]
FN <- cm.rbf[1,2]

#FPR / Type I error
FPR.rbf = FP/(FP+TN)
FPR.rbf
```

```
## [1] 0.01917546
```

```
#FNR / Type II error
FNR.rbf = FN/(FN+TP)
FNR.rbf
```

```
## [1] 0.6552828
```

```
#Precision
precis.rbf = TP/(TP+FP)
precis.rbf
```

```
## [1] 0.6976242
```

```
#Recall / sensitivity
recall.rbf = TP/(TP+FN)
recall.rbf
```

```
## [1] 0.3447172
```

```
#misclassification error
test.err.rbf = 1-(sum(diag(cm.rbf))/sum(cm.rbf))
test.err.rbf
```

```
## [1] 0.09152707
```

Result Summary

```
options(digits = 3)
cl.err <- matrix(c(test.err.knn,FNR.knn,precis.knn,recall.knn,
                   test.err.reg,FNR.reg,precis.reg,recall.reg,
                   test.err.tree1,FNR.tree1,precis.tree1,recall.tree1,
                   test.err.rf,FNR.rf,precis.rf,recall.rf,
                   test.err.gbm,FNR.gbm,precis.gbm,recall.gbm,
                   test.err.ada,FNR.ada,precis.ada,recall.ada,
                   test.err.xgb,FNR.xgb,precis.xgb,recall.xgb,
                   test.err.lin,FNR.lin,precis.lin,recall.lin,
                   test.err.poly,FNR.poly,precis.poly,recall.poly,
                   test.err.rbf,FNR.rbf,precis.rbf,recall.rbf),
                   ncol=4, byrow=TRUE)
colnames(cl.err) <- c('misclass error','type-II error','precision','recall')
rownames(cl.err) <- c('KNN',
                      'Logistic regression',
                      'Decision tree with rpart',
                      'Random forest',
                      'Gradient boosting',
                      'Adaboost',
                      'XGBoost',
                      'SVM with linear kernel',
                      'SVM with polynomial kernel',
                      'SVM with radial kernel')
as.table(cl.err)
```

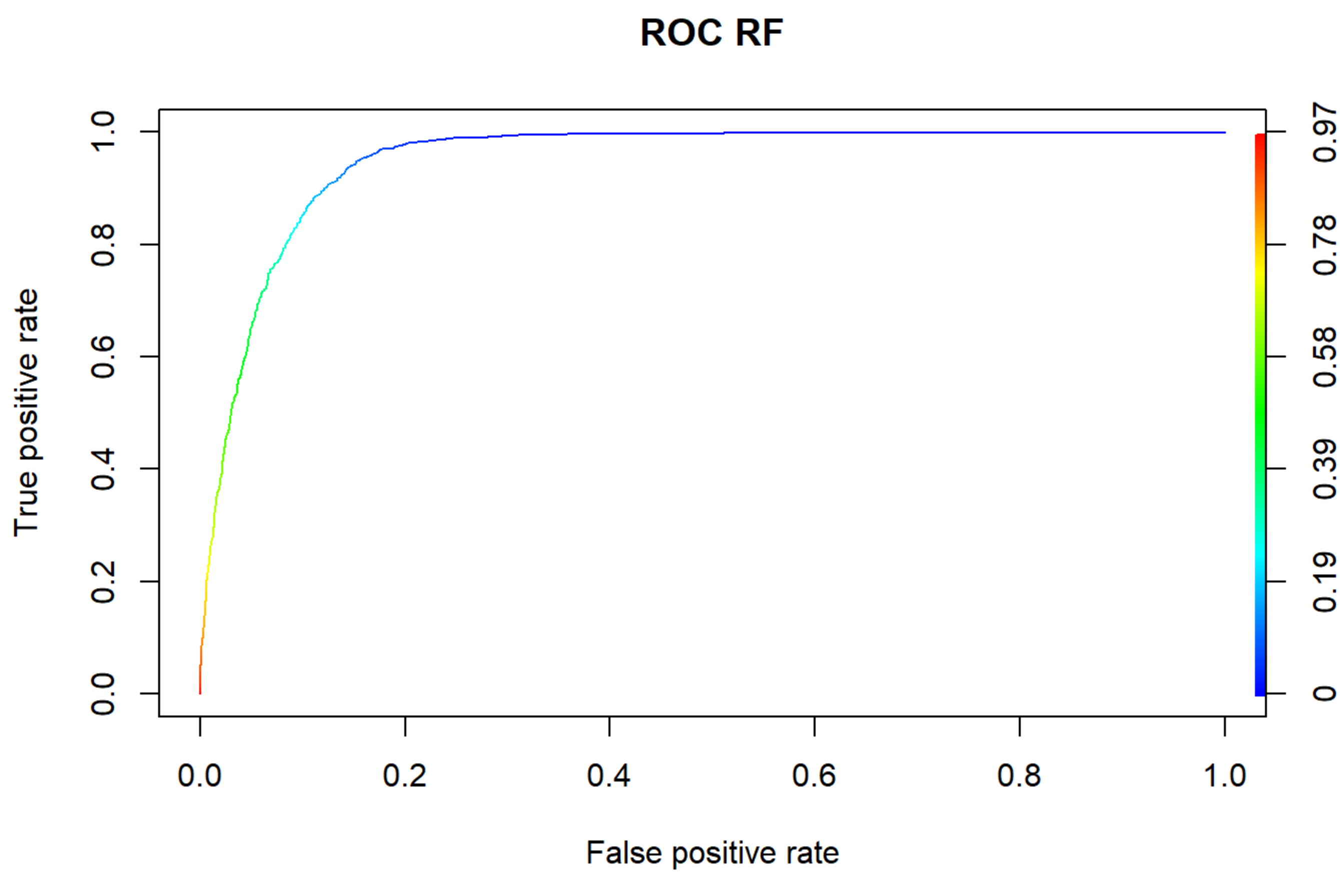
```
##                              misclass error type-II error precision recall
## KNN                                  0.0906        0.5560    0.6490 0.4440
## Logistic regression                  0.0893        0.5827    0.6730 0.4173
## Decision tree with rpart             0.0932        0.6531    0.6757 0.3469
## Random forest                        0.0839        0.4536    0.6581 0.5464
## Gradient boosting                    0.0819        0.4578    0.6737 0.5422
## Adaboost                             0.0896        0.4642    0.6236 0.5358
## XGBoost                              0.0816        0.4568    0.6760 0.5432
## SVM with linear kernel               0.0972        0.6894    0.6525 0.3106
## SVM with polynomial kernel           0.0944        0.7375    0.7387 0.2625
## SVM with radial kernel               0.0915        0.6553    0.6976 0.3447
```

Based on Type-II error comparison, best models are shortlisted: Random Forest, XGBoost, Adaboost, Gradient boosting.

ROC and AUC

Random Forest

```
#Prepare model for ROC curve
rf.pred <- predict(rf.cls, newdata = test, type = "prob")

forestpred = prediction(rf.pred[,2], test$y)

roc.perf.rf = performance(forestpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.rf, main='ROC RF', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.rf, forestpred))
roc.result = as.data.frame((opt.cut(roc.perf.rf, forestpred)))
roc.result
```

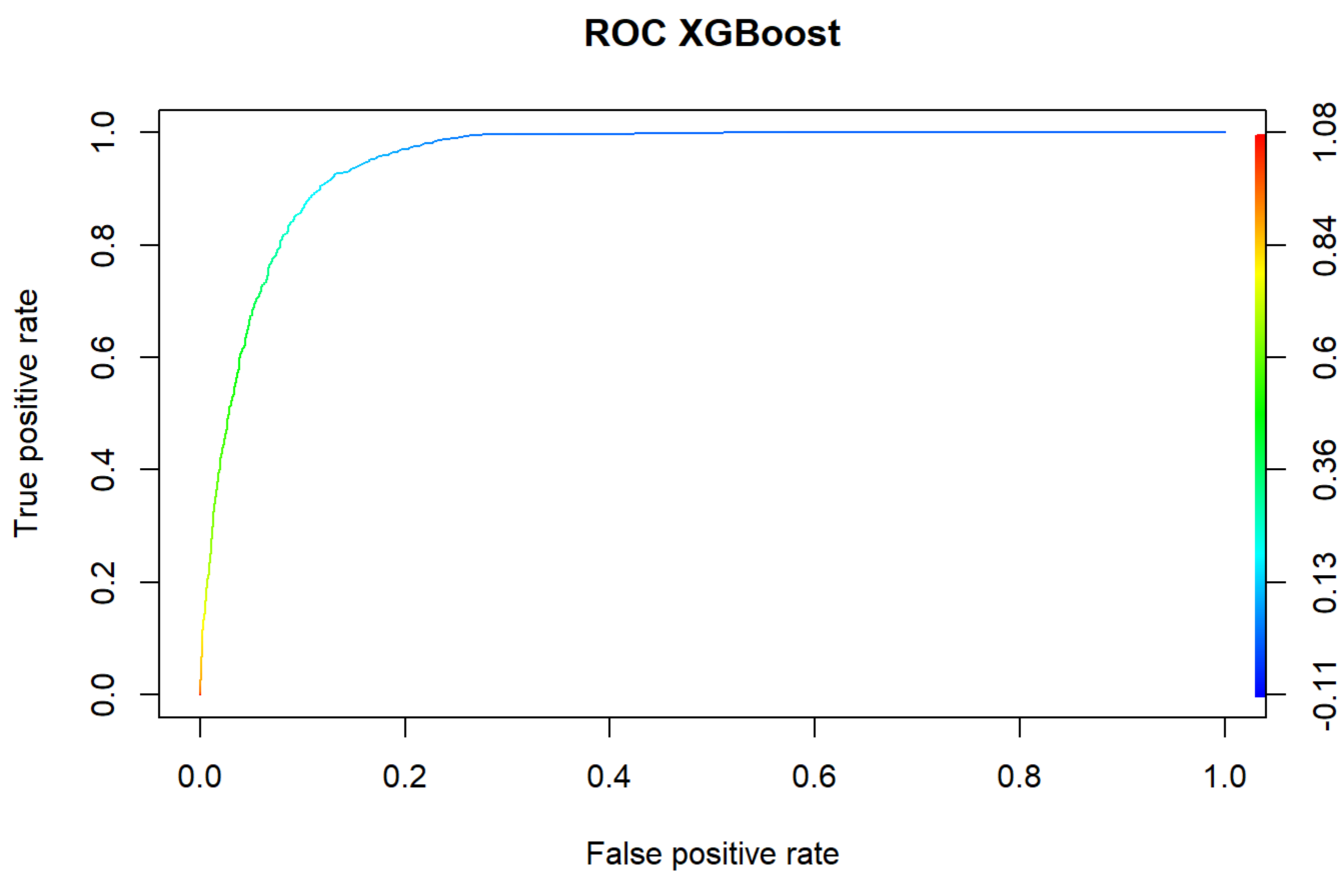
```
##                 V1
## sensitivity 0.908
## specificity 0.875
## cutoff      0.154
```

```
rf.sens = roc.result[1,]
rf.spec = roc.result[2,]
rf.cutoff = roc.result[3,]

auc.perf.rf = performance(forestpred, measure = 'auc')
auc.rf = auc.perf.rf@y.values
auc.rf
```

```
## [[1]]
## [1] 0.951
```

XGBoost

```
#Prepare model for ROC curve
xgbpred = prediction(xgb.pred.prob, test$y)

roc.perf.xgb = performance(xgbpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.xgb, main='ROC XGBoost', colorize=T)
```



```r
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.xgb, xgbpred))
roc.result = as.data.frame((opt.cut(roc.perf.xgb, xgbpred)))
roc.result
```

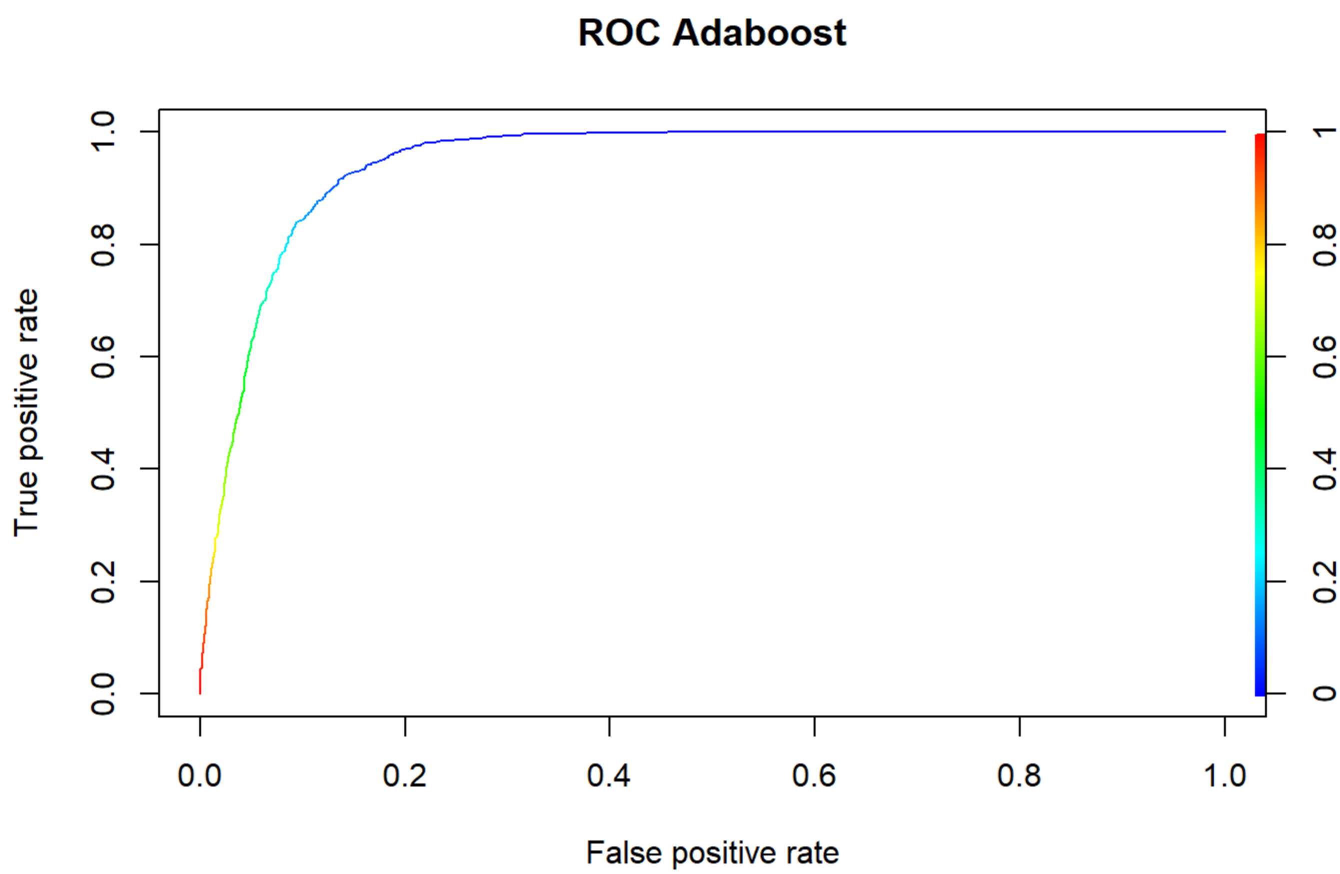
```
##                V1
## sensitivity 0.906
## specificity 0.883
## cutoff      0.166
```

```r
xgb.sens = roc.result[1,]
xgb.spec = roc.result[2,]
xgb.cutoff = roc.result[3,]

auc.perf.xgb = performance(xgbpred, measure = 'auc')
auc.xgb = auc.perf.xgb@y.values
auc.xgb
```

```
## [[1]]
## [1] 0.953
```

Adaboost

```r
#Prepare model for ROC curve
ada.pred <- predict(ada.cls, x.testA, type = "prob")

adapred = prediction(ada.pred, test$y)

roc.perf.ada = performance(adapred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.ada, main='ROC Adaboost', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.ada, adapred))
roc.result = as.data.frame((opt.cut(roc.perf.ada, adapred)))
roc.result
```

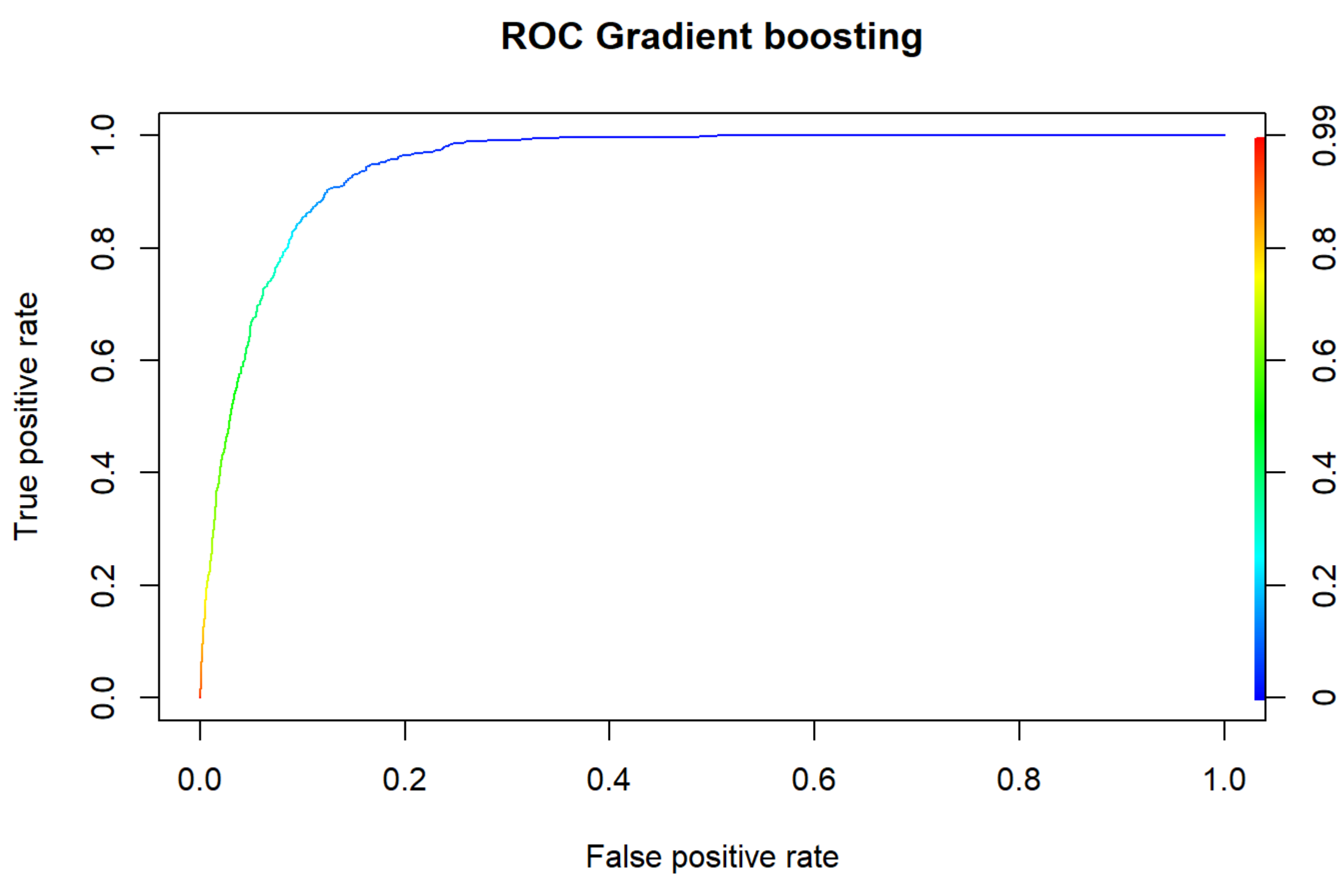
```
##                   V1
## sensitivity 0.9168
## specificity 0.8645
## cutoff      0.0876
```

```
ada.sens = roc.result[1,]
ada.spec = roc.result[2,]
ada.cutoff = roc.result[3,]

auc.perf.ada = performance(adapred, measure = 'auc')
auc.ada = auc.perf.ada@y.values
auc.ada
```

```
## [[1]]
## [1] 0.946
```

Gradient boosting

```
#Prepare model for ROC curve
gbm.pred <- predict (gbm.fit, test, type = "prob")

gbmpred = prediction(gbm.pred[,2], test$y)

roc.perf.gbm = performance(gbmpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.gbm, main='ROC Gradient boosting', colorize=T)
```



```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.gbm, gbmpred))
roc.result = as.data.frame((opt.cut(roc.perf.gbm, gbmpred)))
roc.result
```

```
##                V1
## sensitivity 0.904
## specificity 0.876
## cutoff      0.136
```

```
gbm.sens = roc.result[1,]
gbm.spec = roc.result[2,]
gbm.cutoff = roc.result[3,]

auc.perf.gbm = performance(gbmpred, measure = 'auc')
auc.gbm = auc.perf.gbm@y.values
auc.gbm
```

```
## [[1]]
## [1] 0.949
```

AUC Summary

```
options(digits = 3)
cl.err <- matrix(c(
                   test.err.rf, FNR.rf, rf.cutoff, (1-rf.sens), auc.rf,
                   test.err.xgb, FNR.xgb, xgb.cutoff, (1-xgb.sens), auc.xgb,
                   test.err.ada, FNR.ada, ada.cutoff, (1-ada.sens), auc.ada,
                   test.err.gbm, FNR.gbm, gbm.cutoff, (1-gbm.sens), auc.gbm
                   ),
                   ncol=5, byrow=TRUE)
colnames(cl.err) <- c('misclass err','Type-II err@0.5', 'cutoff', 'Type-II err@cutoff','AUC')
rownames(cl.err) <- c(
                      'RandomForest',
                      'XGBoost',
                      'Adaboost',
                      'Gradboost')
as.matrix(cl.err)
```

```
##               misclass err Type-II err@0.5 cutoff Type-II err@cutoff AUC
## RandomForest 0.0839       0.454           0.154  0.0918             0.951
## XGBoost      0.0816       0.457           0.166  0.0939             0.953
## Adaboost     0.0896       0.464           0.0876 0.0832             0.946
## Gradboost    0.0819       0.458           0.136  0.0961             0.949
```

Bank Marketing Data

Group 8

Load Data

```
#Read dataset
bank_df <- read_delim("bank-additional-full.csv", delim=";")
```

```
## Warning in gzfile(file, mode): cannot open compressed file 'C:/Users/yohci/
## AppData/Local/Temp/RtmpMtZd46\file4eac5ef453ff', probable reason 'No such file
## or directory'
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   age = col_double(),
##   duration = col_double(),
##   campaign = col_double(),
##   pdays = col_double(),
##   previous = col_double(),
##   emp.var.rate = col_double(),
##   cons.price.idx = col_double(),
##   cons.conf.idx = col_double(),
##   euribor3m = col_double(),
##   nr.employed = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```r
#Assign category to all categorical variables

#2.job as category
bank_df$job <- as.factor(bank_df$job)

#3.marital status as category
bank_df$marital <- as.factor(bank_df$marital)

#4.education as category
bank_df$education <- as.factor(bank_df$education)

#5.credit default as category
bank_df$default <- as.factor(bank_df$default)

#6.housing loan as category
bank_df$housing <- as.factor(bank_df$housing)

#7.personal loan as category
bank_df$loan <- as.factor(bank_df$loan)

#8.contact communication type as category
bank_df$contact <- as.factor(bank_df$contact)

#9.last contact month of year as category
bank_df$month <- as.factor(bank_df$month)

#10.last contact day of the month as category
bank_df$day_of_week <- as.factor(bank_df$day_of_week)

#15.outcome of the previous marketing campaign as category
bank_df$poutcome <- as.factor(bank_df$poutcome)

#21.output y as binary factor
bank_df$y <- factor(bank_df$y, levels = c("no","yes"))

dim(bank_df)
```

```
## [1] 41188    21
```

Data preprocessing

```r
bank_df %>%
  summarise_all(list(~sum(. == "unknown"))) %>%
  gather(key = "variable", value = "nr_unknown") %>%
  arrange(-nr_unknown)
```

```
## # A tibble: 21 x 2
##    variable    nr_unknown
##    <chr>            <int>
##  1 default           8597
##  2 education         1731
##  3 housing            990
##  4 loan               990
##  5 job                330
##  6 marital             80
##  7 age                  0
##  8 contact              0
##  9 month                0
## 10 day_of_week          0
## # ... with 11 more rows
```

```
# Analyse default
table(bank_df$default)
```

```
##
##      no unknown     yes
##   32588    8597       3
```

```
## This is not usable, too few "yes" to evaluate
```

analyse the unknown values

```
# setting default parameters for crosstables
# fun_crosstable = function(df, var1, var2){
#   # df: dataframe containing both columns to cross
#   # var1, var2: columns to cross together.
#   CrossTable(df$var1, df$var2,
#              prop.r = T,
#              prop.c = F,
#              prop.t = F,
#              prop.chisq = F,
#              dnn = c(var1, var2)) # dimension names
# }

#default
CrossTable(bank_df$default, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("default",
"y"))
```

```
## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
## 
## 
## Total Observations in Table:  41188 
## 
## 
##              | y 
##      default |        no |       yes | Row Total | 
## -------------|-----------|-----------|-----------|
##           no |     28391 |      4197 |     32588 | 
##              |     0.871 |     0.129 |     0.791 | 
##              |     0.689 |     0.102 |           | 
## -------------|-----------|-----------|-----------|
##      unknown |      8154 |       443 |      8597 | 
##              |     0.948 |     0.052 |     0.209 | 
##              |     0.198 |     0.011 |           | 
## -------------|-----------|-----------|-----------|
##          yes |         3 |         0 |         3 | 
##              |     1.000 |     0.000 |     0.000 | 
##              |     0.000 |     0.000 |           | 
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 | 
## -------------|-----------|-----------|-----------|
## 
## 
```

```
table(bank_df$default)
```

```
## 
##      no unknown     yes 
##   32588    8597       3
```

```
# job
CrossTable(bank_df$job, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("job", "y"))
```

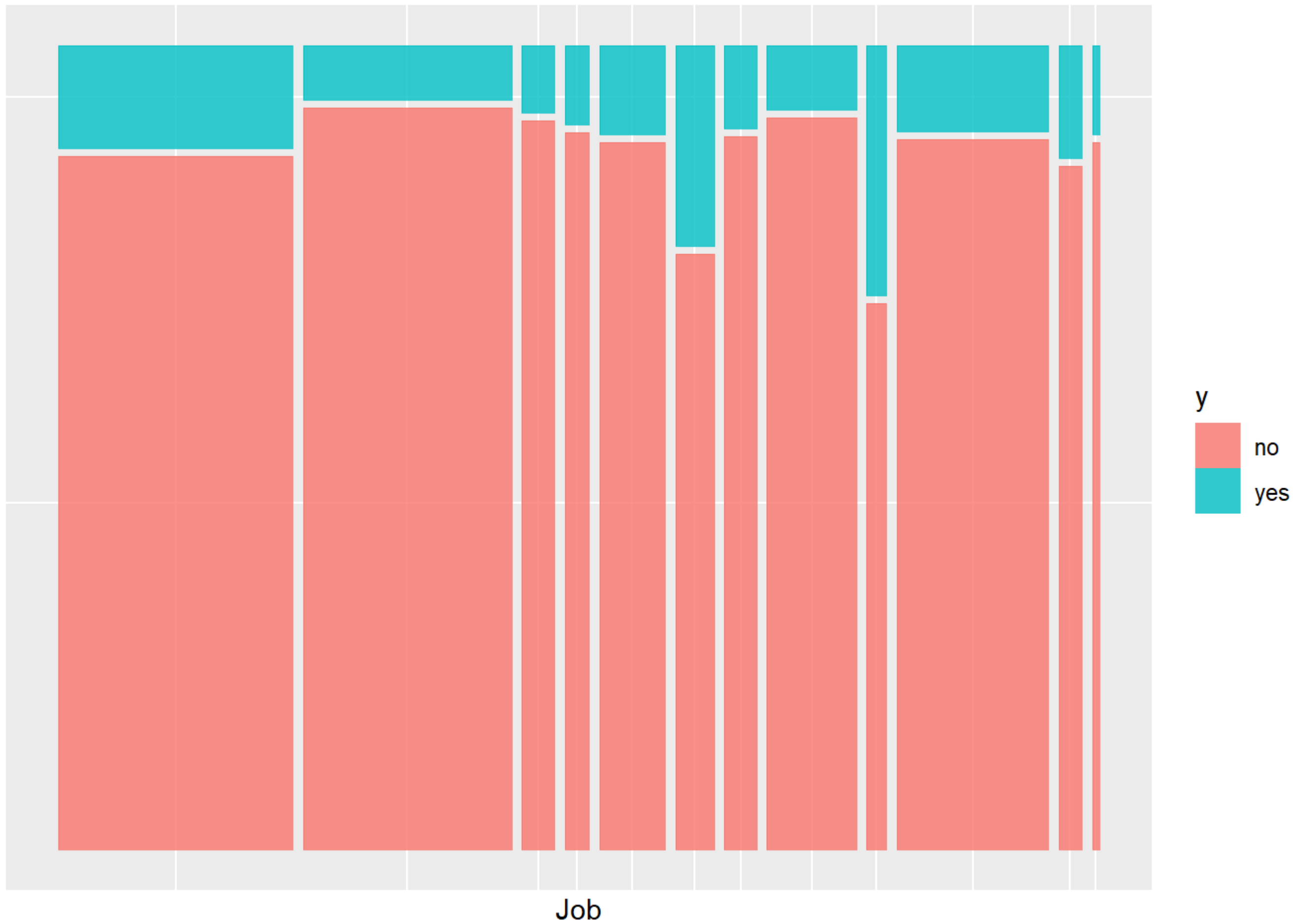
```
## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
## 
## 
## Total Observations in Table:  41188 
## 
## 
##               | y 
##           job |        no |       yes | Row Total | 
## --------------|-----------|-----------|-----------|
##        admin. |      9070 |      1352 |     10422 | 
##               |     0.870 |     0.130 |     0.253 | 
##               |     0.220 |     0.033 |           | 
## --------------|-----------|-----------|-----------|
##   blue-collar |      8616 |       638 |      9254 | 
##               |     0.931 |     0.069 |     0.225 | 
##               |     0.209 |     0.015 |           | 
## --------------|-----------|-----------|-----------|
##  entrepreneur |      1332 |       124 |      1456 | 
##               |     0.915 |     0.085 |     0.035 | 
##               |     0.032 |     0.003 |           | 
## --------------|-----------|-----------|-----------|
##     housemaid |       954 |       106 |      1060 | 
##               |     0.900 |     0.100 |     0.026 | 
##               |     0.023 |     0.003 |           | 
## --------------|-----------|-----------|-----------|
##    management |      2596 |       328 |      2924 | 
##               |     0.888 |     0.112 |     0.071 | 
##               |     0.063 |     0.008 |           | 
## --------------|-----------|-----------|-----------|
##       retired |      1286 |       434 |      1720 | 
##               |     0.748 |     0.252 |     0.042 | 
##               |     0.031 |     0.011 |           | 
## --------------|-----------|-----------|-----------|
## self-employed |      1272 |       149 |      1421 | 
##               |     0.895 |     0.105 |     0.035 | 
##               |     0.031 |     0.004 |           | 
## --------------|-----------|-----------|-----------|
##      services |      3646 |       323 |      3969 | 
##               |     0.919 |     0.081 |     0.096 | 
##               |     0.089 |     0.008 |           | 
## --------------|-----------|-----------|-----------|
##       student |       600 |       275 |       875 | 
##               |     0.686 |     0.314 |     0.021 | 
##               |     0.015 |     0.007 |           | 
## --------------|-----------|-----------|-----------|
##    technician |      6013 |       730 |      6743 | 
```

```
##                |     0.892 |     0.108 |     0.164 |
##                |     0.146 |     0.018 |           |
## --------------|-----------|-----------|-----------|
##    unemployed |       870 |       144 |      1014 |
##               |     0.858 |     0.142 |     0.025 |
##               |     0.021 |     0.003 |           |
## --------------|-----------|-----------|-----------|
##       unknown |       293 |        37 |       330 |
##               |     0.888 |     0.112 |     0.008 |
##               |     0.007 |     0.001 |           |
## --------------|-----------|-----------|-----------|
##  Column Total |     36548 |      4640 |     41188 |
## --------------|-----------|-----------|-----------|
##
##
```

```
table(bank_df$job)
```

```
##
##        admin.   blue-collar  entrepreneur     housemaid    management
##         10422          9254          1456          1060          2924
##       retired self-employed      services       student    technician
##          1720          1421          3969           875          6743
##    unemployed       unknown
##          1014           330
```

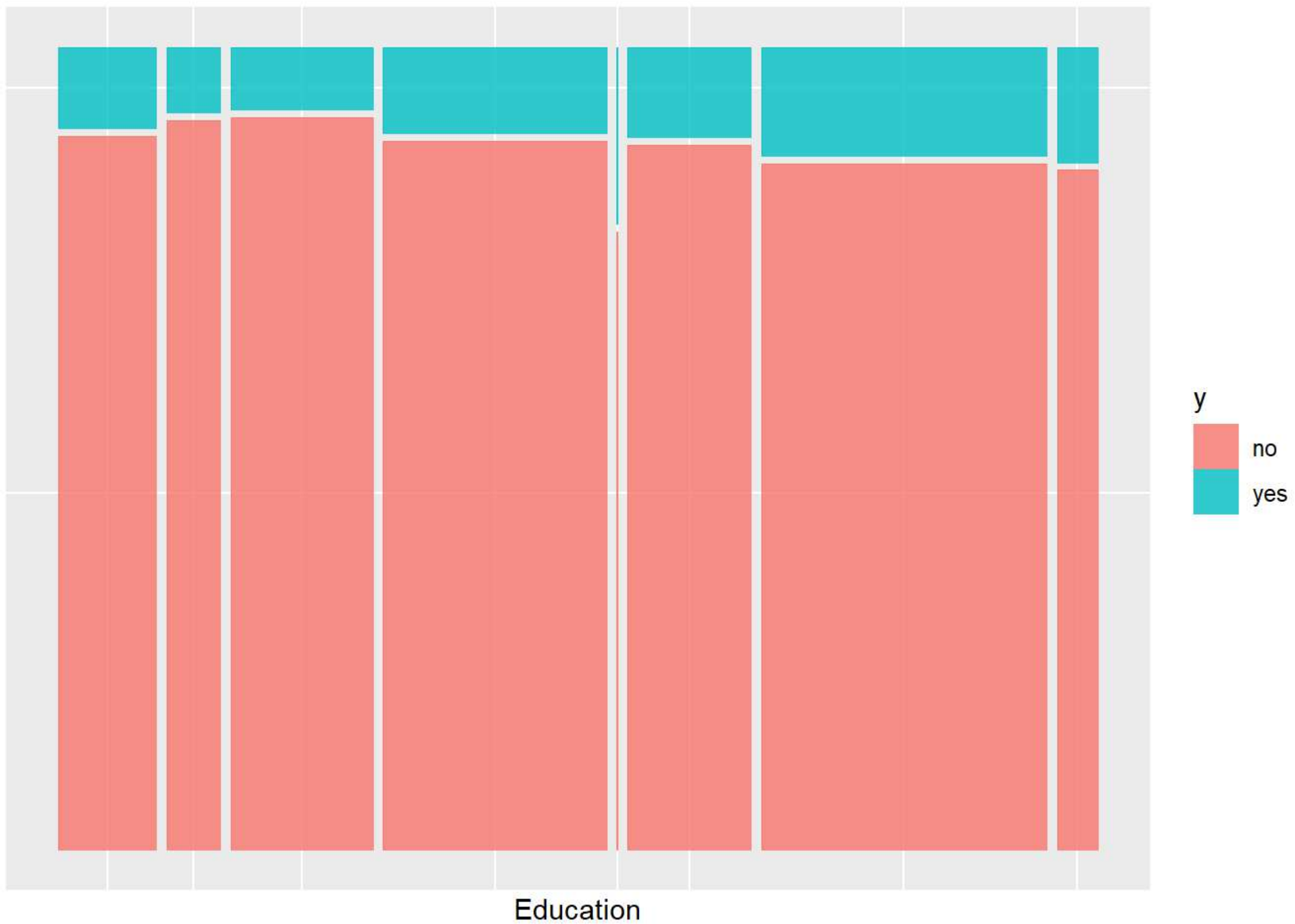
```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, job), fill = y)) +
  #mosaic_theme +
  xlab("Job") +
  ylab(NULL)
```




```
bank_df <- bank_df %>%
  mutate(job = recode(job, "unknown" = "unconventional"))


# marital
CrossTable(bank_df$marital, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("marital",
"y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      marital |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##     divorced |      4136 |       476 |      4612 |
##              |     0.897 |     0.103 |     0.112 |
##              |     0.100 |     0.012 |           |
## -------------|-----------|-----------|-----------|
##      married |     22396 |      2532 |     24928 |
##              |     0.898 |     0.102 |     0.605 |
##              |     0.544 |     0.061 |           |
## -------------|-----------|-----------|-----------|
##       single |      9948 |      1620 |     11568 |
##              |     0.860 |     0.140 |     0.281 |
##              |     0.242 |     0.039 |           |
## -------------|-----------|-----------|-----------|
##      unknown |        68 |        12 |        80 |
##              |     0.850 |     0.150 |     0.002 |
##              |     0.002 |     0.000 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

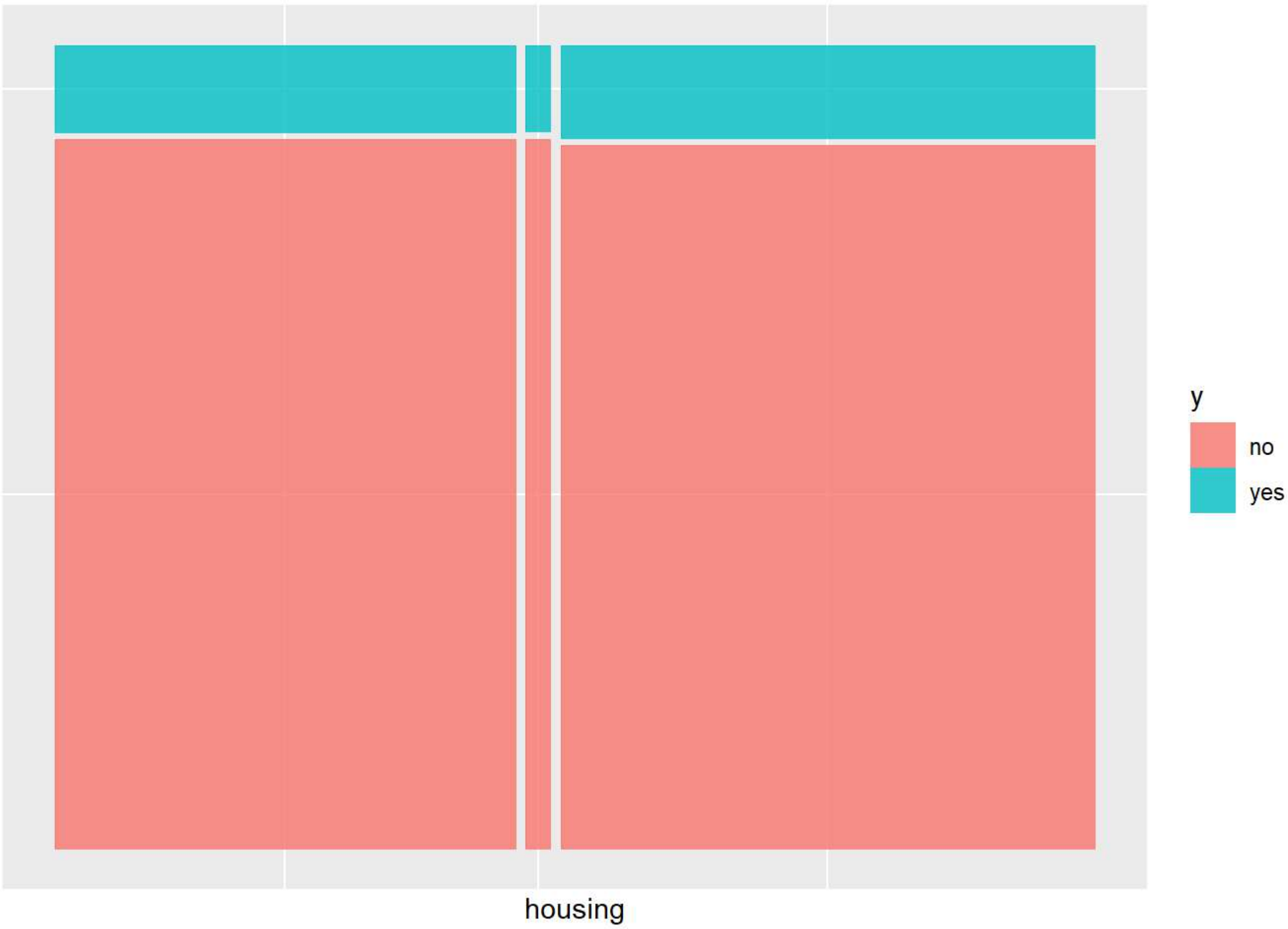
```
## can merge single+unknown, married+divorced since values are similar
bank_df = bank_df %>%
  mutate(marital = recode(marital, "unknown" = "single", "divorced"="married"))

# education
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, education),  fill = y)) +
  #mosaic_theme +
  xlab("Education") +
  ylab(NULL)
```




```
    ## recode unknown as univeristy degree because proportions are similar
bank_df = bank_df %>%
  mutate(education = recode(education, "unknown" = "university.degree"))

# housing
CrossTable(bank_df$housing, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("housing",
"y"))
```

```
## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
## 
## 
## Total Observations in Table:  41188 
## 
## 
##              | y 
##      housing |        no |       yes | Row Total | 
## -------------|-----------|-----------|-----------|
##           no |     16596 |      2026 |     18622 | 
##              |     0.891 |     0.109 |     0.452 | 
##              |     0.403 |     0.049 |           | 
## -------------|-----------|-----------|-----------|
##      unknown |       883 |       107 |       990 | 
##              |     0.892 |     0.108 |     0.024 | 
##              |     0.021 |     0.003 |           | 
## -------------|-----------|-----------|-----------|
##          yes |     19069 |      2507 |     21576 | 
##              |     0.884 |     0.116 |     0.524 | 
##              |     0.463 |     0.061 |           | 
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 | 
## -------------|-----------|-----------|-----------|
## 
## 
```

```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, housing),  fill = y)) +
  #mosaic_theme +
  xlab("housing") +
  ylab(NULL)
```



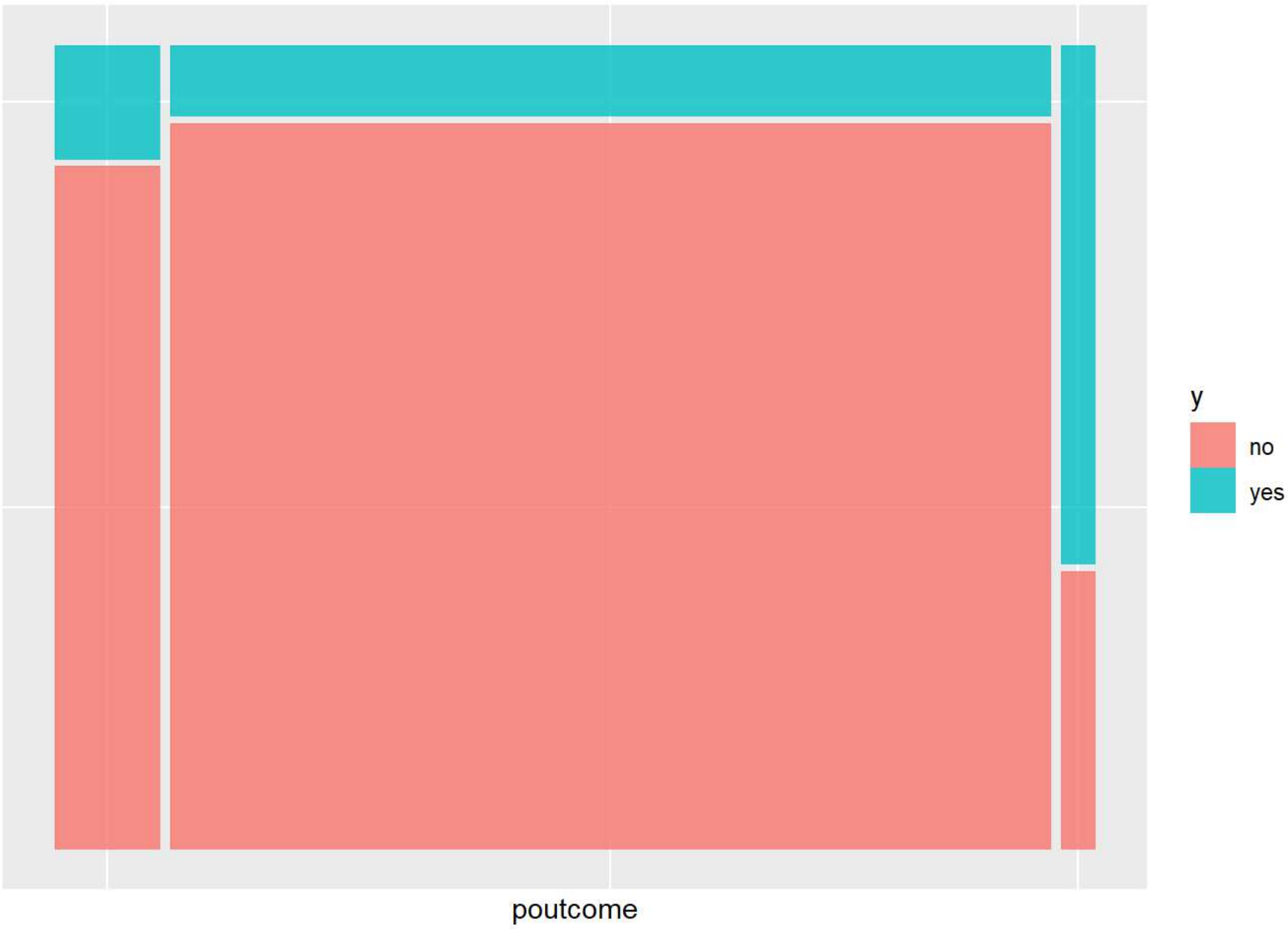

```
  ## the plot looks very similar, do chisquared test to see if there are differences
chisq.test(bank_df$housing, bank_df$y) # drop this column
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$housing and bank_df$y
## X-squared = 5.6845, df = 2, p-value = 0.05829
```

```
bank_df$housing <- NULL

# loan
chisq.test(bank_df$loan, bank_df$y) # drop col, pvalue >0.1
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$loan and bank_df$y
## X-squared = 1.094, df = 2, p-value = 0.5787
```

```
bank_df$loan <- NULL

# pdays
# poutcome
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, poutcome),  fill = y)) +
  #mosaic_theme +
  xlab("poutcome") +
  ylab(NULL)
```




```
bank_df = bank_df %>%
  mutate(past_dummyvar = recode(poutcome, "failure" = 0.5, "nonexistent"=0.2, "success"=1))
# combining previous and poutcome
bank_df$past_dummyvar1 = bank_df$past_dummyvar*(bank_df$previous+1)
chisq.test(bank_df$past_dummyvar1, bank_df$y)
```

```
## Warning in chisq.test(bank_df$past_dummyvar1, bank_df$y): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$past_dummyvar1 and bank_df$y
## X-squared = 4383.4, df = 11, p-value < 2.2e-16
```

```
bank_df$previous <-NULL
bank_df$poutcome <-NULL
bank_df$past_dummyvar <-NULL

bank_df = bank_df %>%
  mutate(pdays_dummy = if_else(pdays == 999, "0", "1")) %>%
  select(-pdays)
bank_df$pdays<-NULL

#resolve default, let yes become unknown
bank_df = bank_df %>%
  mutate(default = recode(default, "yes"="unknown"))


# dayofweek
bank_df = bank_df %>%
  mutate(day_of_week = recode(day_of_week, "mon"=1, "tue"=2,"wed"=3,"thu"=4,"fri"=5))

# age
bank_df = bank_df %>%
  mutate(age = if_else(
    age<20, 1, if_else(
      age<23, 2, if_else(
        age<26, 3, if_else(
          age<31, 4, if_else(
            age<41, 5, if_else(age<51, 6, 7)))))))

#dataset after preprocessing
dim(bank_df)
```

```
## [1] 41188    18
```

```
summary(bank_df)
```

```
##       age                  job           marital
##  Min.   :1.000   admin.     :10422   married:29540
##  1st Qu.:5.000   blue-collar: 9254   single :11648
##  Median :5.000   technician : 6743
##  Mean   :5.367   services   : 3969
##  3rd Qu.:6.000   management : 2924
##  Max.   :7.000   retired    : 1720
##                  (Other)    : 6156
##               education        default           contact          month
##  basic.4y           : 4176   no     :32588   cellular :26144   may    :13769
##  basic.6y           : 2292   unknown: 8600   telephone:15044   jul    : 7174
##  basic.9y           : 6045                                     aug    : 6178
##  high.school        : 9515                                     jun    : 5318
##  illiterate         :   18                                     nov    : 4101
##  professional.course: 5243                                     apr    : 2632
##  university.degree  :13899                                     (Other): 2016
##   day_of_week      duration         campaign       emp.var.rate
##  Min.   :1.00   Min.   :   0.0   Min.   : 1.000   Min.   :-3.40000
##  1st Qu.:2.00   1st Qu.: 102.0   1st Qu.: 1.000   1st Qu.:-1.80000
##  Median :3.00   Median : 180.0   Median : 2.000   Median : 1.10000
##  Mean   :2.98   Mean   : 258.3   Mean   : 2.568   Mean   : 0.08189
##  3rd Qu.:4.00   3rd Qu.: 319.0   3rd Qu.: 3.000   3rd Qu.: 1.40000
##  Max.   :5.00   Max.   :4918.0   Max.   :56.000   Max.   : 1.40000
##
##  cons.price.idx  cons.conf.idx     euribor3m      nr.employed     y
##  Min.   :92.20   Min.   :-50.8   Min.   :0.634   Min.   :4964   no :36548
##  1st Qu.:93.08   1st Qu.:-42.7   1st Qu.:1.344   1st Qu.:5099   yes: 4640
##  Median :93.75   Median :-41.8   Median :4.857   Median :5191
##  Mean   :93.58   Mean   :-40.5   Mean   :3.621   Mean   :5167
##  3rd Qu.:93.99   3rd Qu.:-36.4   3rd Qu.:4.961   3rd Qu.:5228
##  Max.   :94.77   Max.   :-26.9   Max.   :5.045   Max.   :5228
##
##  past_dummyvar1   pdays_dummy
##  Min.   :0.2000   Length:41188
##  1st Qu.:0.2000   Class :character
##  Median :0.2000   Mode  :character
##  Mean   :0.3703
##  3rd Qu.:0.2000
##  Max.   :8.0000
##
```

Oversampling

```
n <- nrow(bank_df); n
```

```
## [1] 41188
```

```r
majorind <- (1:n)[bank_df$y == "no"]
minorind <- (1:n)[bank_df$y == "yes"]
majorn <- length(majorind)
minorn <- length(minorind)

#sample(data_index, numberofdata, replacement?)
OSind<-sample(minorind,majorn-minorn,replace=TRUE)
OSdata<-bank_df[OSind,] # length 4244
# Get the new combined and scaled dataset
OSdata_combined <- rbind(bank_df, bank_df[OSind,]) # length 9066 = 4822+4244
table(OSdata_combined$y) # 4533 points each
```

```
##
##    no   yes
## 36548 36548
```

```r
# splitting train and test
library(caTools)
set.seed(1)
smp_size <- floor(0.8*nrow(OSdata_combined))
train_ind <- sample(seq_len(nrow(OSdata_combined)), size = smp_size)
train <- OSdata_combined[train_ind, ]
test <- OSdata_combined[-train_ind, ]
table(train$y)
```

```
##
##    no   yes
## 29269 29207
```

```r
table(test$y)
```

```
##
##   no  yes
## 7279 7341
```

KNN

```
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1 tim
es

# knn.fit <- train(y ~., data = train, method = "knn",
#                   trControl=trctrl, tuneLength = 10) # tuneLength parameter tells the algorithm
to try different default values for the main parameter

knn.fit <- train(y ~., data = train, method = "knn",
                 trControl=trctrl) # tuneLength parameter tells the algorithm to try different d
efault values for the main parameter

# knn.fit <- train(y ~., data = train, method = "knn")#by default bootstrap is used to find tuni
ng parameter -> trCtrl
knn.fit
```

```
## k-Nearest Neighbors
##
## 58476 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 46781, 46780, 46780, 46781, 46782
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9011048  0.8022425
##   7  0.8927595  0.7855535
##   9  0.8885355  0.7771034
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
#predict using test data
knn.pred <- predict(knn.fit, newdata = test)
#knn.pred

#confusion matrix
cm.knn <- table(knn.pred, test$y)
cm.knn
```

```
##
## knn.pred   no  yes
##      no  6046   93
##      yes 1233 7248
```

```
TP <- cm.knn[2,2]
TN <- cm.knn[1,1]
FP <- cm.knn[2,1]
FN <- cm.knn[1,2]

#FPR / Type I error
FPR.knn = FP/(FP+TN)
FPR.knn
```

```
## [1] 0.1693914
```

```
#FNR / Type II error
FNR.knn = FN/(FN+TP)
FNR.knn
```

```
## [1] 0.01266857
```

```
#Precision
precis.knn = TP/(TP+FP)
precis.knn
```

```
## [1] 0.8546162
```

```
#Recall / sensitivity
recall.knn = TP/(TP+FN)
recall.knn
```

```
## [1] 0.9873314
```

```
#misclassification error
test.err.knn = 1-(sum(diag(cm.knn))/sum(cm.knn))
test.err.knn
```

```
## [1] 0.09069767
```

Logistic Regression

```
set.seed(8)
glm.fit <- glm(y ~., data = train, family = binomial)
```

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

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.3793  -0.1126   0.4861   2.9677
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -2.721e+02  2.326e+01 -11.700  < 2e-16 ***
## age                          -4.289e-02  1.448e-02  -2.962 0.003059 **
## jobblue-collar               -2.391e-01  4.974e-02  -4.808 1.52e-06 ***
## jobentrepreneur              -1.097e-01  7.658e-02  -1.433 0.151953
## jobhousemaid                  9.750e-03  9.554e-02   0.102 0.918720
## jobmanagement                -6.804e-02  5.453e-02  -1.248 0.212142
## jobretired                    5.638e-01  6.612e-02   8.526  < 2e-16 ***
## jobself-employed             -2.345e-01  7.518e-02  -3.119 0.001817 **
## jobservices                  -1.739e-01  5.406e-02  -3.217 0.001294 **
## jobstudent                    2.936e-01  7.886e-02   3.723 0.000197 ***
## jobtechnician                -3.499e-02  4.619e-02  -0.758 0.448709
## jobunemployed                 1.733e-01  8.409e-02   2.061 0.039328 *
## jobunconventional             2.087e-01  1.459e-01   1.430 0.152748
## maritalsingle                 8.718e-02  3.282e-02   2.656 0.007904 **
## educationbasic.6y             1.427e-02  7.680e-02   0.186 0.852612
## educationbasic.9y            -5.231e-02  6.047e-02  -0.865 0.387027
## educationhigh.school          1.075e-02  5.888e-02   0.183 0.855151
## educationilliterate           1.308e+00  5.038e-01   2.597 0.009411 **
## educationprofessional.course  1.366e-01  6.543e-02   2.088 0.036825 *
## educationuniversity.degree    2.348e-01  5.764e-02   4.073 4.64e-05 ***
## defaultunknown               -3.289e-01  4.085e-02  -8.052 8.15e-16 ***
## contacttelephone             -4.912e-01  4.927e-02  -9.970  < 2e-16 ***
## monthaug                      1.095e+00  8.526e-02  12.847  < 2e-16 ***
## monthdec                      1.925e-01  1.624e-01   1.185 0.236050
## monthjul                     -6.024e-03  6.205e-02  -0.097 0.922660
## monthjun                     -9.603e-01  7.873e-02 -12.198  < 2e-16 ***
## monthmar                      2.154e+00  9.995e-02  21.554  < 2e-16 ***
## monthmay                     -7.900e-01  5.171e-02 -15.279  < 2e-16 ***
## monthnov                     -6.764e-01  7.673e-02  -8.815  < 2e-16 ***
## monthoct                      4.405e-01  1.009e-01   4.366 1.27e-05 ***
## monthsep                      4.440e-01  1.170e-01   3.794 0.000148 ***
## day_of_week                  -2.724e-03  9.377e-03  -0.290 0.771461
## duration                      6.965e-03  6.679e-05 104.283  < 2e-16 ***
## campaign                     -2.446e-02  6.991e-03  -3.498 0.000469 ***
## emp.var.rate                 -2.325e+00  8.776e-02 -26.498  < 2e-16 ***
## cons.price.idx                2.557e+00  1.547e-01  16.528  < 2e-16 ***
## cons.conf.idx                 3.846e-03  5.471e-03   0.703 0.482038
## euribor3m                     6.296e-01  8.431e-02   7.468 8.13e-14 ***
## nr.employed                   5.489e-03  1.908e-03   2.877 0.004019 **
## past_dummyvar1               -2.865e-01  3.438e-02  -8.332  < 2e-16 ***
## pdays_dummy1                  2.157e+00  9.588e-02  22.501  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 81065  on 58475  degrees of freedom
## Residual deviance: 38198  on 58435  degrees of freedom
## AIC: 38280
##
## Number of Fisher Scoring iterations: 6
```

```
#predict using test data
glm.prob <- predict(glm.fit, type = "response", newdata = test)

#check which one is 'Yes'
contrasts(test$y)#Yes = 1, Low = 0
```

```
##     yes
## no    0
## yes   1
```

```
glm.pred <- rep('no', nrow(test))
glm.pred[glm.prob > 0.5] <- 'yes' #yes = 1, no = 0

#confusion matrix
cm.reg = table(glm.pred, test$y)
cm.reg
```

```
##
## glm.pred   no  yes
##      no  6274  849
##      yes 1005 6492
```

```
TP <- cm.reg[2,2]
TN <- cm.reg[1,1]
FP <- cm.reg[2,1]
FN <- cm.reg[1,2]

#FPR / Type I error
FPR.reg = FP/(FP+TN)
FPR.reg
```

```
## [1] 0.1380684
```

```
#FNR / Type II error
FNR.reg = FN/(FN+TP)
FNR.reg
```

```
## [1] 0.1156518
```

```
#Precision
precis.reg = TP/(TP+FP)
precis.reg
```

```
## [1] 0.8659464
```

```
#Recall / sensitivity
recall.reg = TP/(TP+FN)
recall.reg
```

```
## [1] 0.8843482
```

```
#misclassification error
test.err.reg = 1-(sum(diag(cm.reg))/sum(cm.reg))
test.err.reg
```

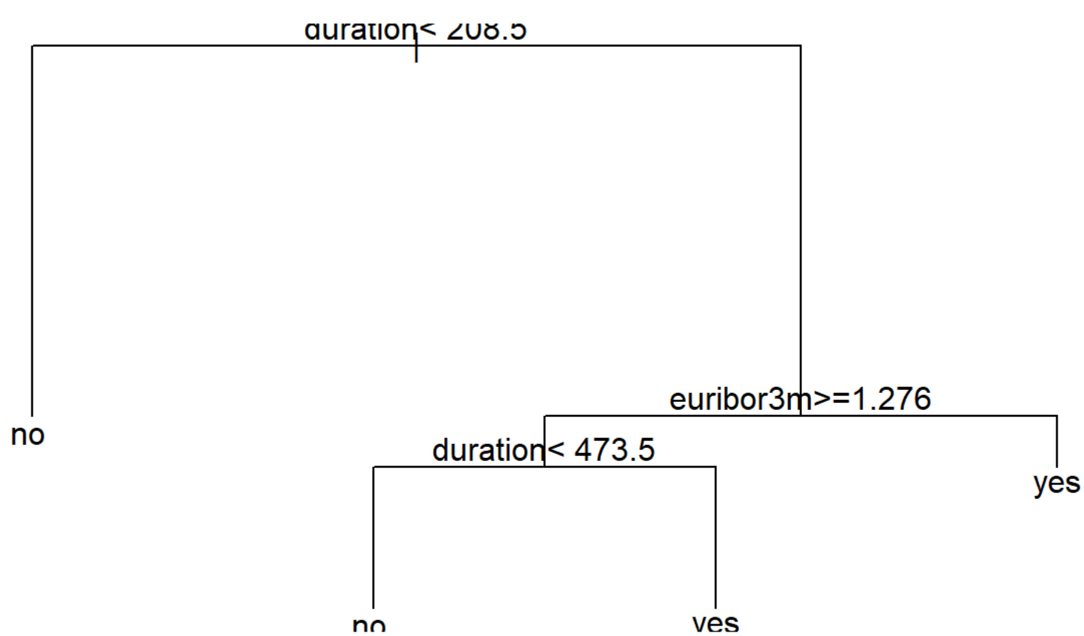
```
## [1] 0.1268126
```

Decision Tree

```
train.tree <- data.frame(train)
test.tree <-data.frame(test)

#Decision Tree using rpart()
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1 tim
es

cls.tree1 = train(y ~ ., data=train.tree, method="rpart",
                  trControl=trctrl)# tuneLength parameter tells the algorithm to try different d
efault values for the main parameter

# cls.tree1 = train(y ~ ., data=train.tree, method="rpart")#by default bootstrap is used to find
tuning parameter -> trCtrl

cls.tree1
```

```
## CART
##
## 58476 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 46781, 46780, 46780, 46781, 46782
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.07155819  0.8109814  0.6219926
##   0.08609238  0.7862363  0.5725275
##   0.45174102  0.6809321  0.3617189
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.07155819.
```

```
plot(cls.tree1$finalModel)
text(cls.tree1$finalModel)
```



```r
#predict using test data
tree.pred1 <- predict(cls.tree1, newdata = test.tree)
#tree.pred1

#confusion matrix
cm.tree1 <- table(tree.pred1, test$y)
cm.tree1
```

```
##
## tree.pred1   no  yes
##        no  6441 1946
##        yes  838 5395
```

```r
TP <- cm.tree1[2,2]
TN <- cm.tree1[1,1]
FP <- cm.tree1[2,1]
FN <- cm.tree1[1,2]

#FPR / Type I error
FPR.tree1 = FP/(FP+TN)
FPR.tree1
```

```
## [1] 0.1151257
```

```r
#FNR / Type II error
FNR.tree1 = FN/(FN+TP)
FNR.tree1
```

```
## [1] 0.2650865
```

```r
#Precision
precis.tree1 = TP/(TP+FP)
precis.tree1
```

```
## [1] 0.8655543
```

```r
#Recall / sensitivity
recall.tree1 = TP/(TP+FN)
recall.tree1
```

```
## [1] 0.7349135
```

```r
#misclassification error
test.err.tree1 = 1-(sum(diag(cm.tree1))/sum(cm.tree1))
test.err.tree1
```

```
## [1] 0.1904241
```

Random Forest

```
#Random forest with 500 bootstrapped trees
#p = 16
sqrt(16) # ntree = 4
```

```
## [1] 4
```

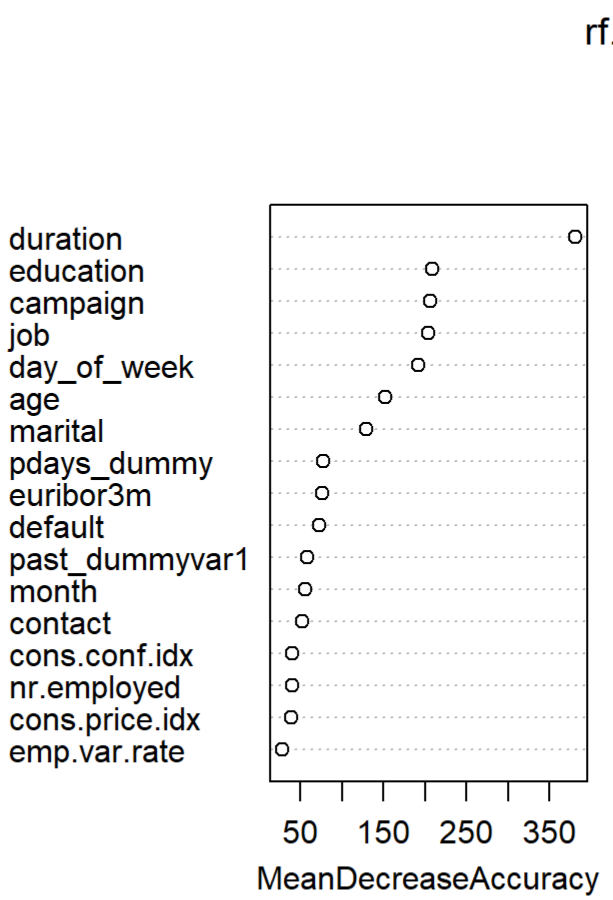
```
set.seed(8)
rf.cls <- randomForest(y ~ .,
                       data = train,
                       mtry = 4,
                       ntree = 500,
                       importance = TRUE)
rf.cls
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train, mtry = 4, ntree = 500,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 4.37%
## Confusion matrix:
##        no   yes  class.error
## no  26729  2540 0.0867812361
## yes    14 29193 0.0004793371
```

```
#ls(rf.cls)
importance(rf.cls)
```

```
##                        no       yes MeanDecreaseAccuracy MeanDecreaseGini
## age              12.429526 153.40158            152.30127        681.2940
## job              23.572414 200.25575            203.72387       1323.2002
## marital           2.934999 129.92328            129.00311        236.0142
## education         4.331264 209.88006            208.62744        825.6247
## default          -7.766226  73.69850             73.11891        199.4638
## contact           8.627092  57.53414             51.95514        247.0378
## month            31.939679  54.66100             56.11903       1072.9966
## day_of_week      24.404648 203.07014            192.57306        696.6359
## duration        310.415999 362.55453            381.64679      11950.4967
## campaign         -4.966380 208.09632            206.54586        764.4524
## emp.var.rate     20.019480  24.03849             27.90471       1272.2180
## cons.price.idx   19.580034  42.07286             39.21067        533.6724
## cons.conf.idx    18.503076  45.29669             40.77544        744.0300
## euribor3m        28.065364  89.57287             76.13213       3135.1825
## nr.employed      22.469122  38.78348             39.93334       2446.3922
## past_dummyvar1    8.633164  63.41333             58.71083        668.3224
## pdays_dummy       5.335515  86.15948             77.43937        467.9722
```

```
varImpPlot(rf.cls)
```



```r
#predict using test data
rf.pred <- predict(rf.cls, newdata = test, type = "class")
#rf.pred

#confusion matrix
cm.rf <- table(rf.pred, test$y)
cm.rf
```

```
##
## rf.pred   no  yes
##      no  6677    1
##      yes  602 7340
```

```r
TP <- cm.rf[2,2]
TN <- cm.rf[1,1]
FP <- cm.rf[2,1]
FN <- cm.rf[1,2]

#FPR / Type I error
FPR.rf = FP/(FP+TN)
FPR.rf
```

```
## [1] 0.08270367
```

```r
#FNR / Type II error
FNR.rf = FN/(FN+TP)
FNR.rf
```

```
## [1] 0.0001362212
```

```r
#Precision
precis.rf = TP/(TP+FP)
precis.rf
```

```
## [1] 0.9242005
```

```r
#Recall / sensitivity
recall.rf = TP/(TP+FN)
recall.rf
```

```
## [1] 0.9998638
```

```r
#misclassification error
test.err.rf = 1-(sum(diag(cm.rf))/sum(cm.rf))
test.err.rf
```

```
## [1] 0.04124487
```

Gradient Boosting

```
#Gradient boosting
set.seed(8)

#Use K-fold CV to find best trControl
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 1) #5 folds repeated 1 times

gbm.fit <- train(y ~ ., data = train,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)

# gbm.fit <- train(y ~ ., data = train,
#                  method = "gbm",
#                  verbose = FALSE) #by default bootstrap is used to find tuning parameter -> tr
Ctrl
gbm.fit
```

```
## Stochastic Gradient Boosting
##
## 58476 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 46781, 46780, 46780, 46781, 46782
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.8572065  0.7144247
##   1                  100      0.8711268  0.7422659
##   1                  150      0.8738971  0.7478057
##   2                   50      0.8727513  0.7455228
##   2                  100      0.8804125  0.7608453
##   2                  150      0.8846365  0.7692931
##   3                   50      0.8796088  0.7592395
##   3                  100      0.8853205  0.7706612
##   3                  150      0.8886039  0.7772268
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
#predict using test data
gbm.pred <- predict(gbm.fit, newdata = test)
#gbm.pred

#confusion matrix
cm.gbm <- table(gbm.pred, test$y)
cm.gbm
```

```
##
## gbm.pred   no  yes
##      no  6182  522
##      yes 1097 6819
```

```
TP <- cm.gbm[2,2]
TN <- cm.gbm[1,1]
FP <- cm.gbm[2,1]
FN <- cm.gbm[1,2]

#FPR / Type I error
FPR.gbm = FP/(FP+TN)
FPR.gbm
```

```
## [1] 0.1507075
```

```
#FNR / Type II error
FNR.gbm = FN/(FN+TP)
FNR.gbm
```

```
## [1] 0.07110748
```

```
#Precision
precis.gbm = TP/(TP+FP)
precis.gbm
```

```
## [1] 0.8614199
```

```
#Recall / sensitivity
recall.gbm = TP/(TP+FN)
recall.gbm
```

```
## [1] 0.9288925
```

```
#misclassification error
test.err.gbm = 1-(sum(diag(cm.gbm))/sum(cm.gbm))
test.err.gbm
```

```
## [1] 0.1107387
```

AdaBoost

```
#AdaBoost
set.seed(8)
x.trainA = model.matrix(data=train, y~.-1)
y.trainA = rep(1, nrow(train))
y.trainA [train$y=="no"]=-1 #for Adaboost

x.testA = model.matrix(data=test, y~.-1)
y.testA = rep(1, nrow(test))
y.testA [test$y=="no"]=-1 #for Adaboost

ada.cls <- adaboost(x.trainA, y.trainA, tree_depth=5, n_rounds=500)
ada.cls
```

```
## AdaBoost: tree_depth =  5  rounds =  500
##
##
##  In-sample confusion matrix:
##     yhat
## y        -1     1
##   -1 25704  3565
##   1   1187 28020
```

```
#predict using test data
ada.pred <- predict(ada.cls, x.testA)
#ada.pred

#confusion matrix
cm.ada <- table(ada.pred, y.testA) #-1 is "no", 1 is "yes"
cm.ada
```

```
##         y.testA
## ada.pred   -1    1
##       -1 6373  350
##       1   906 6991
```

```
TP <- cm.ada[2,2]
TN <- cm.ada[1,1]
FP <- cm.ada[2,1]
FN <- cm.ada[1,2]

#FPR / Type I error
FPR.ada = FP/(FP+TN)
FPR.ada
```

```
## [1] 0.1244676
```

```
#FNR / Type II error
FNR.ada = FN/(FN+TP)
FNR.ada
```

```
## [1] 0.04767743
```

```
#Precision
precis.ada = TP/(TP+FP)
precis.ada
```

```
## [1] 0.8852729
```

```
#Recall / sensitivity
recall.ada = TP/(TP+FN)
recall.ada
```

```
## [1] 0.9523226
```

```
#misclassification error
test.err.ada = 1-(sum(diag(cm.ada))/sum(cm.ada))
test.err.ada
```

```
## [1] 0.08590971
```

XGBoost

```
#XGBoost
set.seed(8)
x.trainXG =model.matrix(data=train,y~.-1)
y.trainXG = rep(1, nrow(train))
y.trainXG[train$y=="no"]=0 #for XGBoost

x.testXG = model.matrix(data=test, y~.-1)
y.testXG = rep(1, nrow(test))
y.testXG[test$y=="no"]=0 #for XGBoost

xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=5,eta=0.01,nrounds=500,verbose=FALSE
)
#xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=10,nrounds=500,verbose=FALSE)
xgb.cls
```

```
## ##### xgb.Booster
## raw: 1.1 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 5, eta = 0.01)
## params (as set within xgb.train):
##   max_depth = "5", eta = "0.01", validate_parameters = "1"
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 41
## niter: 500
## nfeatures : 41
## evaluation_log:
##     iter train_rmse
##        1   0.496814
##        2   0.493676
## ---
##      499   0.272070
##      500   0.272022
```

```
#predict using test data
xgb.pred.prob<-predict(xgb.cls,x.testXG)

xgb.pred<-as.numeric(xgb.pred.prob>0.5) #convert to 0 ("no") or 1 ("yes")

#confusion matrix
cm.xgb<-table(xgb.pred,y.testXG) #0 is "no", 1 is "yes"
cm.xgb
```

```
##         y.testXG
## xgb.pred    0    1
##        0 6226  391
##        1 1053 6950
```

```
TP <- cm.xgb[2,2]
TN <- cm.xgb[1,1]
FP <- cm.xgb[2,1]
FN <- cm.xgb[1,2]

#FPR / Type I error
FPR.xgb = FP/(FP+TN)
FPR.xgb
```

```
## [1] 0.1446627
```

```
#FNR / Type II error
FNR.xgb = FN/(FN+TP)
FNR.xgb
```

```
## [1] 0.0532625
```

```
#Precision
precis.xgb = TP/(TP+FP)
precis.xgb
```

```
## [1] 0.8684243
```

```
#Recall / sensitivity
recall.xgb = TP/(TP+FN)
recall.xgb
```

```
## [1] 0.9467375
```

```
#misclassification error
test.err.xgb = 1-sum(diag(cm.xgb))/sum(cm.xgb)
test.err.xgb
```

```
## [1] 0.09876881
```

SVM with linear kernel

```
set.seed(8)
svm.fit <- svm(y~., data=train, kernel='linear', cost=1)
#summary(svm.fit)

#CV for tuning the cost parameter
set.seed(8)
tune.out1 <- tune(svm, y~.,
              data=train,
              kernel="linear",
              )

#tune.out1 <- tune(svm, y~.,
#               data=train,
#               kernel="linear",
#               ranges=list(cost=c(0.01, 0.1, 1, 10, 100)),tunecontrol=tune.control(cross=10))
summary(tune.out1)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.1225287
```

```r
svm.lin.best <- tune.out1$best.model
summary(svm.lin.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  18932
##
##  ( 9447 9485 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```r
#predict using test data
lin.pred <- predict(svm.lin.best, test)

#confusion matrix
cm.lin <- table(lin.pred, test$y)
cm.lin
```

```
##
## lin.pred   no  yes
##      no  6149  639
##      yes 1130 6702
```

```r
TP <- cm.lin[2,2]
TN <- cm.lin[1,1]
FP <- cm.lin[2,1]
FN <- cm.lin[1,2]

#FPR / Type I error
FPR.lin = FP/(FP+TN)
FPR.lin
```

```
## [1] 0.1552411
```

```
#FNR / Type II error
FNR.lin = FN/(FN+TP)
FNR.lin
```

```
## [1] 0.08704536
```

```
#Precision
precis.lin = TP/(TP+FP)
precis.lin
```

```
## [1] 0.8557201
```

```
#Recall / sensitivity
recall.lin = TP/(TP+FN)
recall.lin
```

```
## [1] 0.9129546
```

```
#misclassification error
test.err.lin = 1-(sum(diag(cm.lin))/sum(cm.lin))
test.err.lin
```

```
## [1] 0.1209986
```

SVM with polynomial kernel

```
set.seed(8)
tune.out2 <- tune(svm, y~.,
              data=train,
              kernel="polynomial",
              )

#tune.out2 <- tune(svm, y~.,
#               data=train,
#               kernel="polynomial",
#               ranges=list(cost=c(0.1,1,5,10,15,20,50,100),
#                           degree=c(2,3,4)))
summary(tune.out2)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.124085
```

```
svm.poly.best <- tune.out2$best.model
summary(svm.poly.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  24284
##
##  ( 12052 12232 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
poly.pred <- predict(svm.poly.best, test)

#confusion matrix
cm.poly <- table(poly.pred, test$y)
cm.poly
```

```
##
## poly.pred   no  yes
##       no  6150  685
##       yes 1129 6656
```

```
TP <- cm.poly[2,2]
TN <- cm.poly[1,1]
FP <- cm.poly[2,1]
FN <- cm.poly[1,2]

#FPR / Type I error
FPR.poly = FP/(FP+TN)
FPR.poly
```

```
## [1] 0.1551037
```

```
#FNR / Type II error
FNR.poly = FN/(FN+TP)
FNR.poly
```

```
## [1] 0.09331154
```

```
#Precision
precis.poly = TP/(TP+FP)
precis.poly
```

```
## [1] 0.8549775
```

```
#Recall / sensitivity
recall.poly = TP/(TP+FN)
recall.poly
```

```
## [1] 0.9066885
```

```
#misclassification error
test.err.poly = 1-(sum(diag(cm.poly))/sum(cm.poly))
test.err.poly
```

```
## [1] 0.1240766
```

SVM with rbf kernel

```
set.seed(8)
tune.out3 <- tune(svm, y~.,
              data=train,
              kernel="radial",)

#tune.out3 <- tune(svm, y~.,
#              data=train,
#              kernel="radial",
#              ranges=list(cost=c(0.1,1,5,10),
#                          gamma=c(0.01,0.1,1,5,10,100)))
summary(tune.out3)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.1132942
```

```
svm.rbf.best <- tune.out3$best.model
summary(svm.rbf.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  17897
##
##  ( 8758 9139 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
rbf.pred <- predict(svm.rbf.best, test)

#confusion matrix
cm.rbf <- table(rbf.pred, test$y)
cm.rbf
```

```
##
## rbf.pred   no  yes
##      no  6090  463
##      yes 1189 6878
```

```
TP <- cm.rbf[2,2]
TN <- cm.rbf[1,1]
FP <- cm.rbf[2,1]
FN <- cm.rbf[1,2]

#FPR / Type I error
FPR.rbf = FP/(FP+TN)
FPR.rbf
```

```
## [1] 0.1633466
```

```
#FNR / Type II error
FNR.rbf = FN/(FN+TP)
FNR.rbf
```

```
## [1] 0.06307043
```

```
#Precision
precis.rbf = TP/(TP+FP)
precis.rbf
```

```
## [1] 0.8526094
```

```
#Recall / sensitivity
recall.rbf = TP/(TP+FN)
recall.rbf
```

```
## [1] 0.9369296
```

```
#misclassification error
test.err.rbf = 1-(sum(diag(cm.rbf))/sum(cm.rbf))
test.err.rbf
```

```
## [1] 0.1129959
```

Result Summary

```
options(digits = 3)
cl.err <- matrix(c(test.err.knn,FNR.knn,precis.knn,recall.knn,
                   test.err.reg,FNR.reg,precis.reg,recall.reg,
                   test.err.tree1,FNR.tree1,precis.tree1,recall.tree1,
                   test.err.rf,FNR.rf,precis.rf,recall.rf,
                   test.err.gbm,FNR.gbm,precis.gbm,recall.gbm,
                   test.err.ada,FNR.ada,precis.ada,recall.ada,
                   test.err.xgb,FNR.xgb,precis.xgb,recall.xgb,
                   test.err.lin,FNR.lin,precis.lin,recall.lin,
                   test.err.poly,FNR.poly,precis.poly,recall.poly,
                   test.err.rbf,FNR.rbf,precis.rbf,recall.rbf),
                   ncol=4, byrow=TRUE)
colnames(cl.err) <- c('misclass error','type-II error','precision','recall')
rownames(cl.err) <- c('KNN',
                      'Logistic regression',
                      'Decision tree with rpart',
                      'Random forest',
                      'Gradient boosting',
                      'Adaboost',
                      'XGBoost',
                      'SVM with linear kernel',
                      'SVM with polynomial kernel',
                      'SVM with radial kernel')
as.table(cl.err)
```

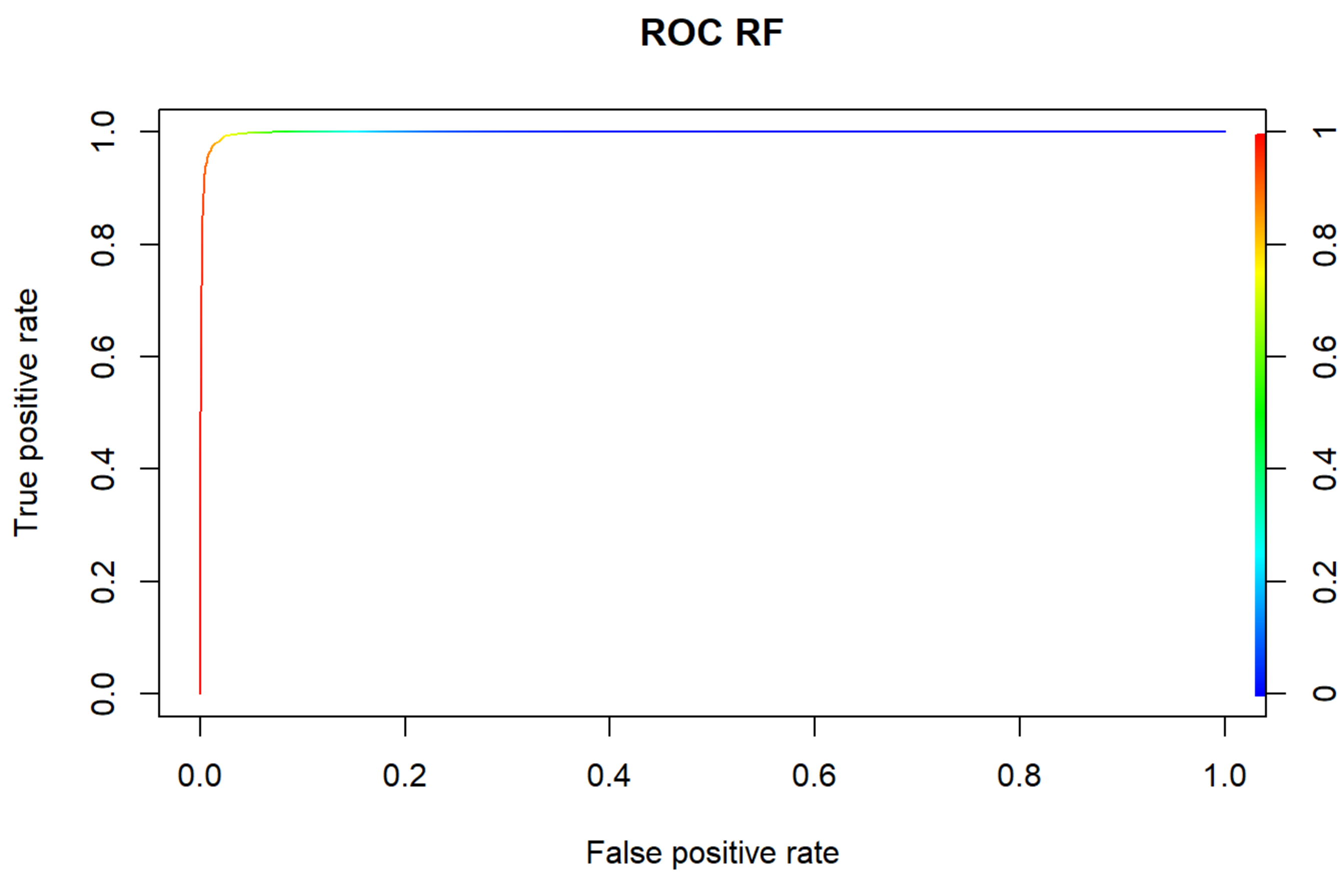
```
##                              misclass error type-II error precision   recall
## KNN                                0.090698      0.012669  0.854616 0.987331
## Logistic regression                0.126813      0.115652  0.865946 0.884348
## Decision tree with rpart           0.190424      0.265087  0.865554 0.734913
## Random forest                      0.041245      0.000136  0.924200 0.999864
## Gradient boosting                  0.110739      0.071107  0.861420 0.928893
## Adaboost                           0.085910      0.047677  0.885273 0.952323
## XGBoost                            0.098769      0.053262  0.868424 0.946738
## SVM with linear kernel             0.120999      0.087045  0.855720 0.912955
## SVM with polynomial kernel         0.124077      0.093312  0.854978 0.906688
## SVM with radial kernel             0.112996      0.063070  0.852609 0.936930
```

Based on Type-II error comparison, best model is: Random Forest.

ROC and AUC

Random Forest

```
#Prepare model for ROC curve
rf.pred <- predict(rf.cls, newdata = test, type = "prob")

forestpred = prediction(rf.pred[,2], test$y)

roc.perf.rf = performance(forestpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.rf, main='ROC RF', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.rf, forestpred))
roc.result = as.data.frame((opt.cut(roc.perf.rf, forestpred)))
roc.result
```

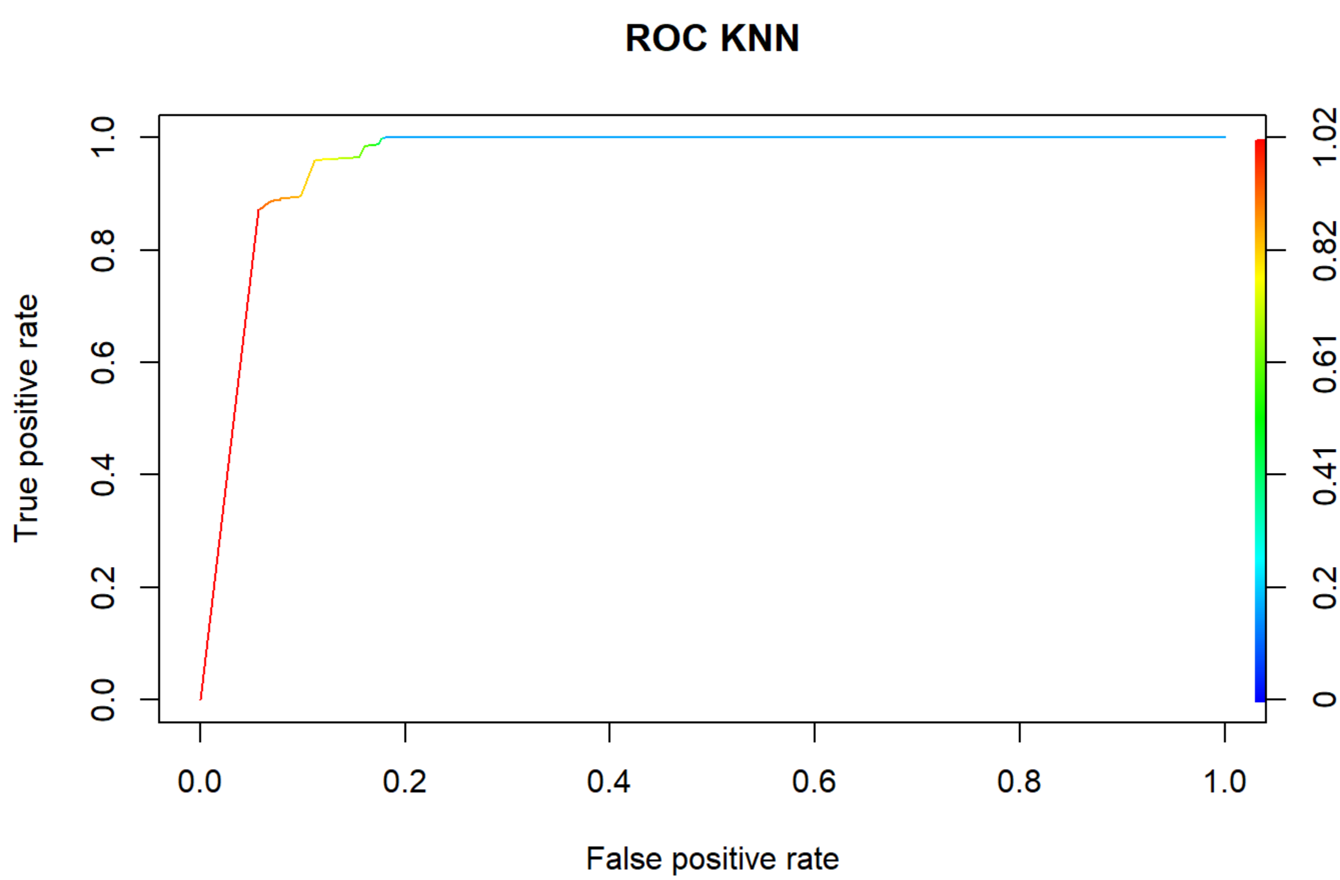
```
##                V1
## sensitivity 0.990
## specificity 0.979
## cutoff      0.784
```

```
rf.sens = roc.result[1,]
rf.spec = roc.result[2,]
rf.cutoff = roc.result[3,]

auc.perf.rf = performance(forestpred, measure = 'auc')
auc.rf = auc.perf.rf@y.values
auc.rf
```

```
## [[1]]
## [1] 0.998
```

KNN

```
#Prepare model for ROC curve
knn.pred <- predict (knn.fit, test, type = "prob")

knnpred = prediction(knn.pred[,2], test$y)

roc.perf.knn = performance(knnpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.knn, main='ROC KNN', colorize=T)
```



```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.knn, knnpred))
roc.result = as.data.frame((opt.cut(roc.perf.knn, knnpred)))
roc.result
```

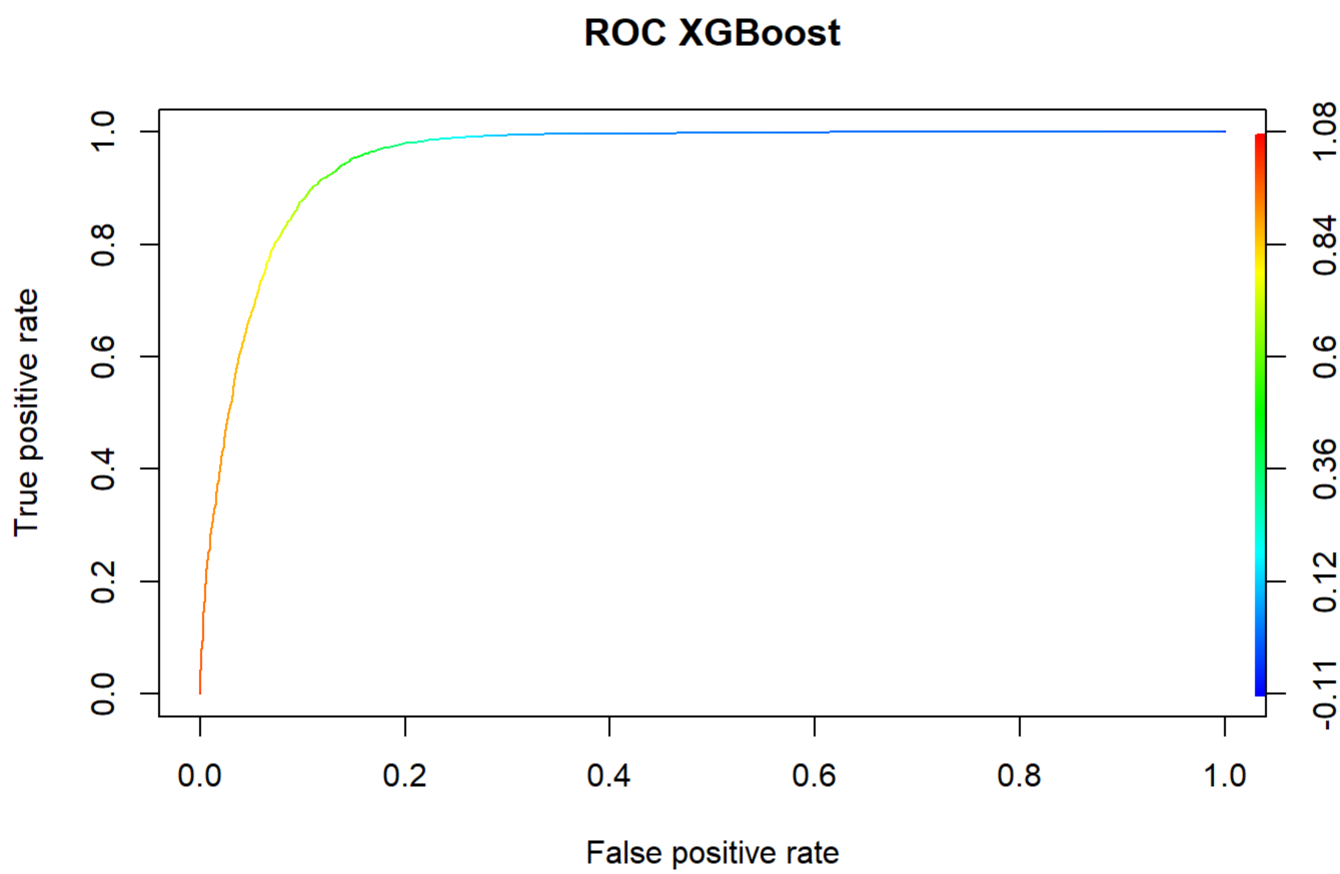
```
##                V1
## sensitivity 0.960
## specificity 0.888
## cutoff      0.800
```

```
knn.sens = roc.result[1,]
knn.spec = roc.result[2,]
knn.cutoff = roc.result[3,]

auc.perf.knn = performance(knnpred, measure = 'auc')
auc.knn = auc.perf.knn@y.values
auc.knn
```

```
## [[1]]
## [1] 0.96
```

XGBoost

```
#Prepare model for ROC curve
xgbpred = prediction(xgb.pred.prob, test$y)

roc.perf.xgb = performance(xgbpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.xgb, main='ROC XGBoost', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.xgb, xgbpred))
roc.result = as.data.frame((opt.cut(roc.perf.xgb, xgbpred)))
roc.result
```

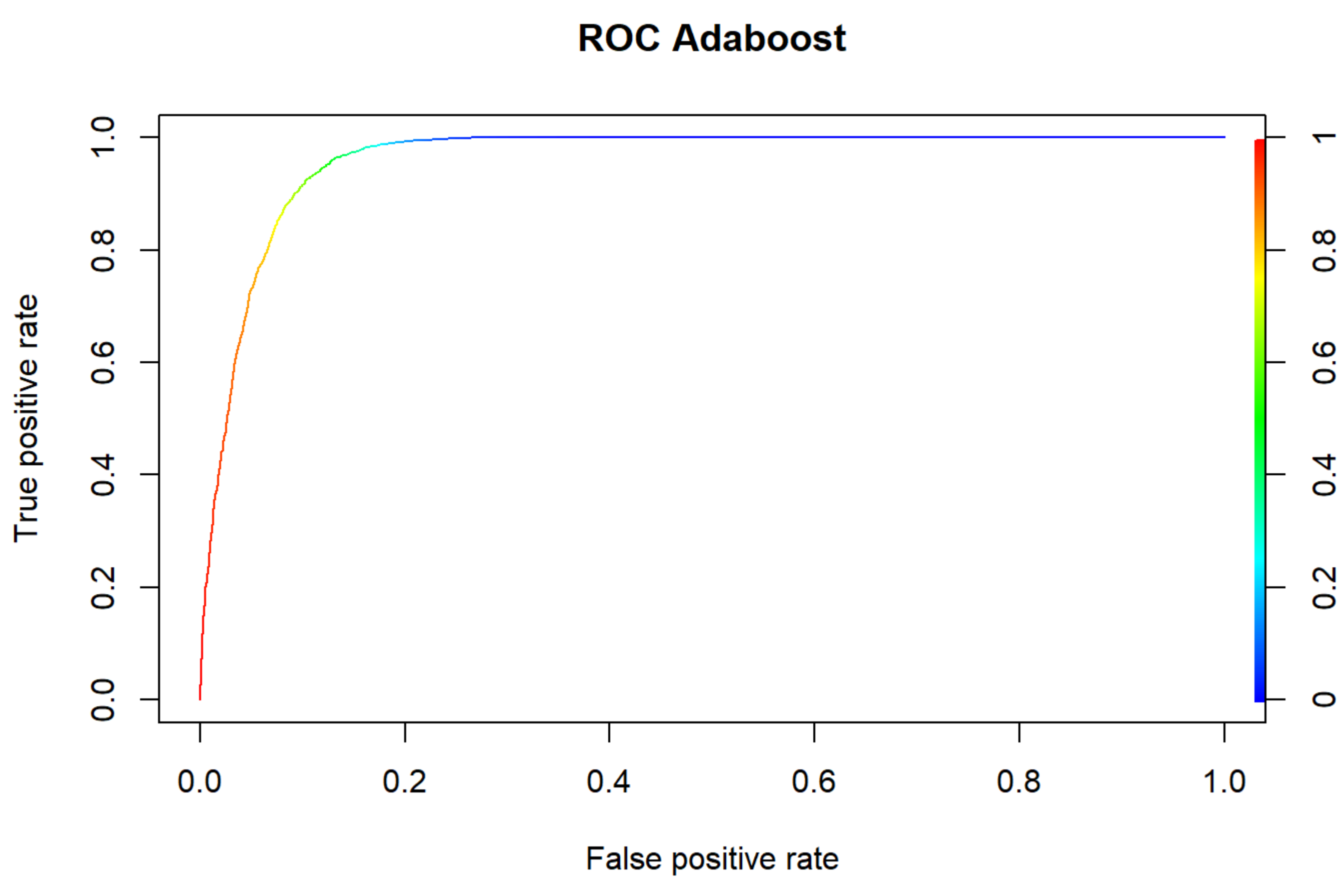
```
##                 V1
## sensitivity 0.915
## specificity 0.883
## cutoff      0.597
```

```
xgb.sens = roc.result[1,]
xgb.spec = roc.result[2,]
xgb.cutoff = roc.result[3,]

auc.perf.xgb = performance(xgbpred, measure = 'auc')
auc.xgb = auc.perf.xgb@y.values
auc.xgb
```

```
## [[1]]
## [1] 0.954
```

Adaboost

```
#Prepare model for ROC curve
ada.pred <- predict(ada.cls, x.testA, type = "prob")

adapred = prediction(ada.pred, test$y)

roc.perf.ada = performance(adapred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.ada, main='ROC Adaboost', colorize=T)
```



```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.ada, adapred))
roc.result = as.data.frame((opt.cut(roc.perf.ada, adapred)))
roc.result
```

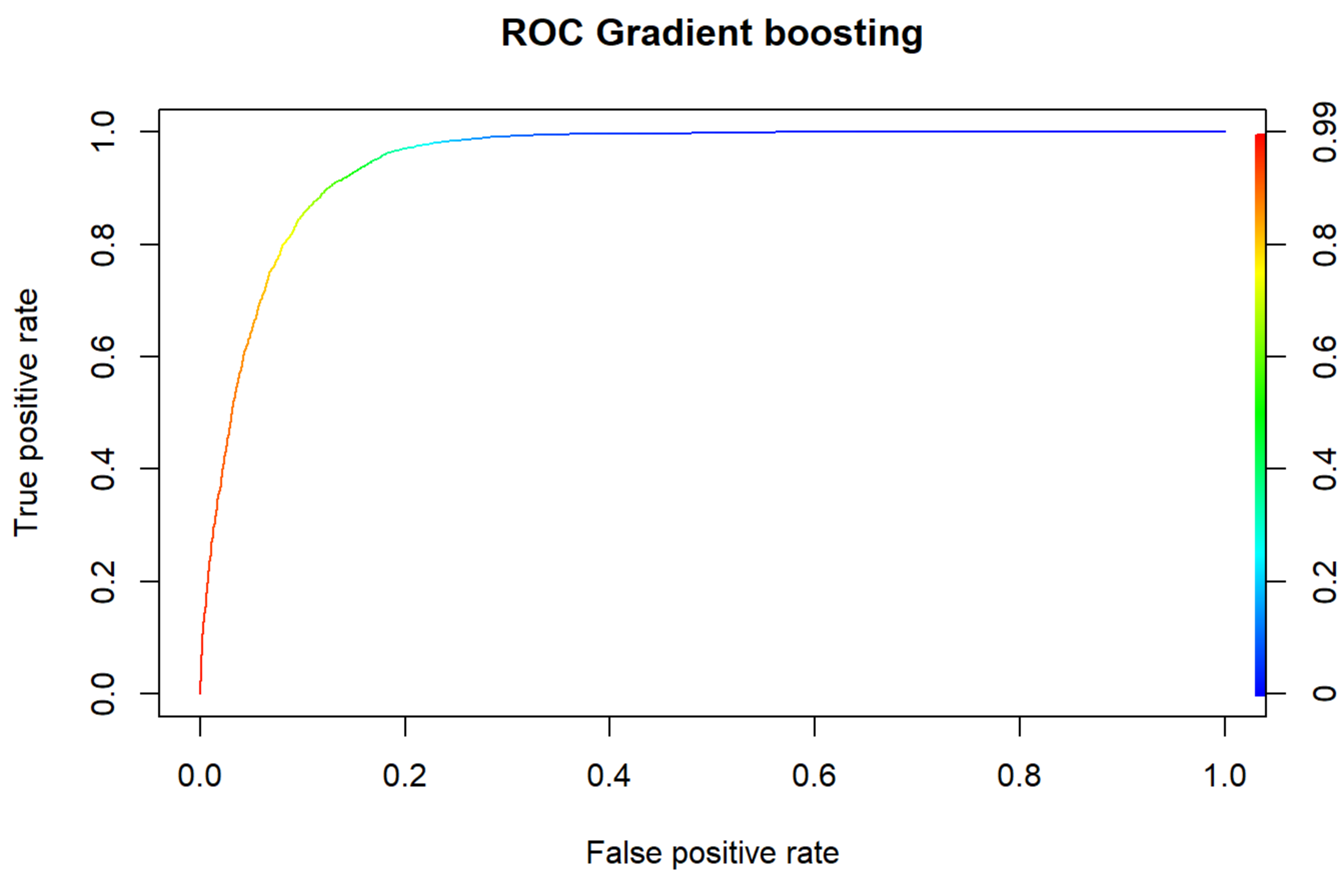
```
##                V1
## sensitivity 0.926
## specificity 0.896
## cutoff      0.605
```

```
ada.sens = roc.result[1,]
ada.spec = roc.result[2,]
ada.cutoff = roc.result[3,]

auc.perf.ada = performance(adapred, measure = 'auc')
auc.ada = auc.perf.ada@y.values
auc.ada
```

```
## [[1]]
## [1] 0.961
```

Gradient boosting

```
#Prepare model for ROC curve
gbm.pred <- predict (gbm.fit, test, type = "prob")

gbmpred = prediction(gbm.pred[,2], test$y)

roc.perf.gbm = performance(gbmpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.gbm, main='ROC Gradient boosting', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.gbm, gbmpred))
roc.result = as.data.frame((opt.cut(roc.perf.gbm, gbmpred)))
roc.result
```

```
##                V1
## sensitivity 0.911
## specificity 0.868
## cutoff      0.563
```

```
gbm.sens = roc.result[1,]
gbm.spec = roc.result[2,]
gbm.cutoff = roc.result[3,]

auc.perf.gbm = performance(gbmpred, measure = 'auc')
auc.gbm = auc.perf.gbm@y.values
auc.gbm
```

```
## [[1]]
## [1] 0.949
```

AUC Summary

```
options(digits = 3)
cl.err <- matrix(c(test.err.rf, FNR.rf, rf.cutoff, (1-rf.sens), auc.rf,
                   test.err.knn, FNR.knn, knn.cutoff, (1-knn.sens), auc.knn,
                   test.err.xgb, FNR.xgb, xgb.cutoff, (1-xgb.sens), auc.xgb,
                   test.err.ada, FNR.ada, ada.cutoff, (1-ada.sens), auc.ada,
                   test.err.gbm, FNR.gbm, gbm.cutoff, (1-gbm.sens), auc.gbm
                   ),
                   ncol=5, byrow=TRUE)
colnames(cl.err) <- c('misclass err','Type-II err@0.5', 'cutoff', 'Type-II err@cutoff','AUC')
rownames(cl.err) <- c('RandomForest',
                      'KNN',
                      'XGBoost',
                      'Adaboost',
                      'Gradboost')
as.matrix(cl.err)
```

```
##              misclass err Type-II err@0.5 cutoff Type-II err@cutoff AUC
## RandomForest 0.0412       0.000136        0.784  0.0101             0.998
## KNN          0.0907       0.0127          0.8    0.0398             0.96
## XGBoost      0.0988       0.0533          0.597  0.0854             0.954
## Adaboost     0.0859       0.0477          0.605  0.0737             0.961
## Gradboost    0.111        0.0711          0.563  0.089              0.949
```