Bank Marketing: Predicting results of telemarketing campaigns

Alex V. Hadar

7/28/2021

Introduction

Using analytics to predict the target groups or the success of marketing campaigns and product acquisition is a widely used method in the banking and e-commerce/retail industries. The decision predicted for a (potential) customer is based on customer data and information from previous campaigns, but also on domain knowledge shared between the stakeholders of the analysis.

We analyze data from the telemarketing campaign collected by a bank in Portugal. This data, provided by Moro et al. (see Moro, Cortez, and Rita (2014)), is available in the UCI repository under https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank.zip. It includes customer personal information and data on past campaigns for the customers.

We use several classification models and algorithms to predict the outcome: customer accepts ("yes") or rejects the term deposit offer ("no"). Then the results are interpreted according to the desired business outcome: predict customers who are more likely to subscribe to the deposit.

Analysis

We start the analysis with a summary, followed by the visual exploration of the telemarketing data. In the next steps, we explore different classification models and find those which best suit the business purpose. To measure the model performance, we use the confusion matrix - especially accuracy and specificity - and related values, like balanced accuracy or F-score.

Exploratory data analysis (EDA)

The bank marketing dataset can be downloaded and unpacked using the following code:

```
# Bank Marketing dataset:
# https://archive.ics.uci.edu/ml/datasets/bank+marketing
# https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank.zip

dl <- tempfile()
download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank.zip",
              dl)
bank <- read_csv2(unzip(dl, "bank-full.csv"))
bank.small <- read_csv2(unzip(dl, "bank.csv")) # small dataset: 10% of "bank-full.csv"
# bank <- read_csv2("bank-full.csv")
# bank.small <- read_csv2("bank.csv")
# convert to data frame
bank <- as.data.frame(bank)
bank.small <- as.data.frame(bank.small)
```

The data consists of 45211 observations for 17 variables collected during 5 years.
The predictors are both numerical (e.g. age, balance, duration) and discrete/categorical (e.g. job, marital, education, housing, poutcome); the outcome is categorical ("yes"/"no").

More details can be found in the dataset description at the UCI repository.

```
##       age                 job           marital          education
##  Min.   :18.00   blue-collar:9732   divorced: 5207   primary  : 6851
##  1st Qu.:33.00   management :9458   married :27214   secondary:23202
##  Median :39.00   technician :7597   single  :12790   tertiary :13301
##  Mean   :40.94   admin.     :5171                    unknown  : 1857
##  3rd Qu.:48.00   services   :4154
##  Max.   :95.00   retired    :2264
##                  (Other)    :6835
##  default        balance       housing      loan            contact
##  no :44396   Min.   : -8019   no :20081   no :37967   cellular :29285
##  yes:  815   1st Qu.:    72   yes:25130   yes: 7244   telephone: 2906
##              Median :   448                           unknown  :13020
##              Mean   :  1362
##              3rd Qu.:  1428
##              Max.   :102127
##
##       day            month          duration         campaign
##  Min.   : 1.00   may    :13766   Min.   :   0.0   Min.   : 1.000
##  1st Qu.: 8.00   jul    : 6895   1st Qu.: 103.0   1st Qu.: 1.000
##  Median :16.00   aug    : 6247   Median : 180.0   Median : 2.000
##  Mean   :15.81   jun    : 5341   Mean   : 258.2   Mean   : 2.764
##  3rd Qu.:21.00   nov    : 3970   3rd Qu.: 319.0   3rd Qu.: 3.000
##  Max.   :31.00   apr    : 2932   Max.   :4918.0   Max.   :63.000
##                  (Other): 6060
##      pdays          previous           poutcome       y
##  Min.   : -1.0   Min.   :  0.0000   failure: 4901   no :39922
##  1st Qu.: -1.0   1st Qu.:  0.0000   other  : 1840   yes: 5289
##  Median : -1.0   Median :  0.0000   success: 1511
##  Mean   : 40.2   Mean   :  0.5803   unknown:36959
##  3rd Qu.: -1.0   3rd Qu.:  0.0000
##  Max.   :871.0   Max.   :275.0000
##
```

As mentioned by the authors of the dataset, the distribution of the outcome variable y shows that the data is not balanced: 78% "no" responses, 12% "yes" responses. This situation is encountered often in real-world data. To correct the imbalance a number of approaches have been proposed (see Fawcett (2016)): * undersampling
* upsampling
* weighting

We will use undersampling to generate a more balanced dataset in the modeling part, but the alternatives are also useful. Then, models will run on the original dataset and on the balanced dataset in order to compare the performance metrics.

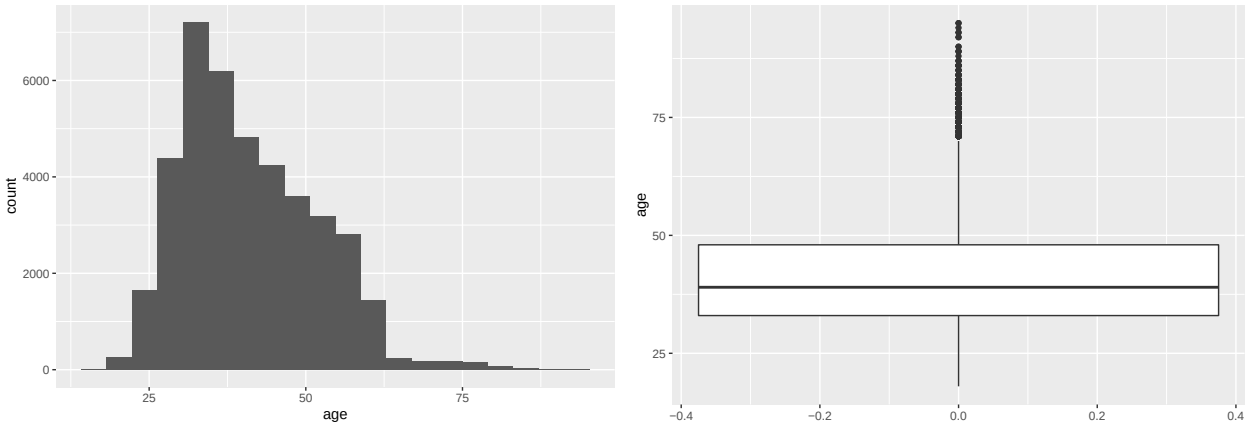
For the purpose of analysis and modeling we split the data into training and test data, with 90% allocated to training, 10% to test. In case of small datasets, a higher percentage of test data might be useful, but this dataset includes sufficient data for training and then evaluation on the remaining 10%.

Additionally, there are unknown values for some variables. These will be described below and will be treated as a category by themselves (no imputation for missing values).

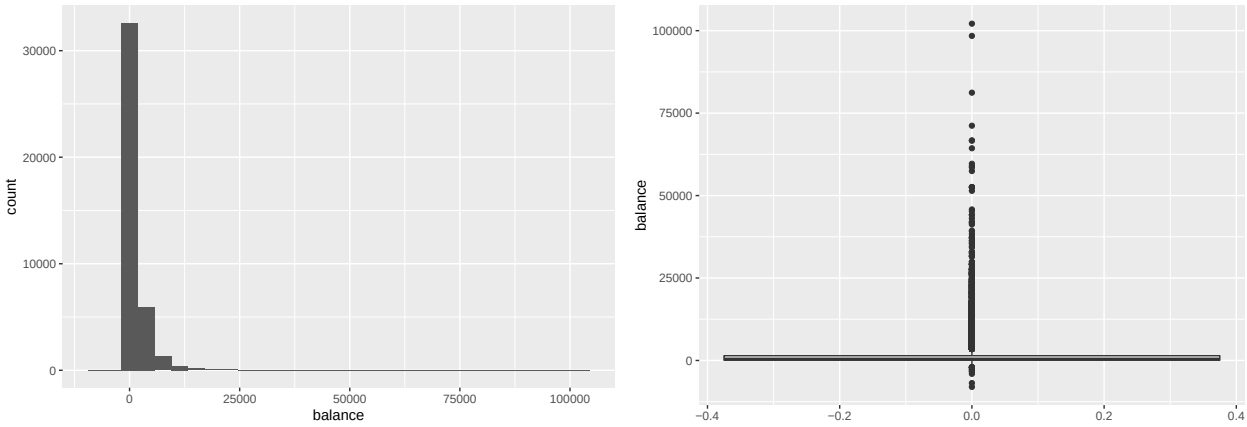
The next step is the visual exploration of the training data.

For the given features we could do feature selection, but this is out of scope for this analysis. However, based on domain knowledge we would expect balance, age, housing, loan, duration of contact and previous outcome to influence the prediction.

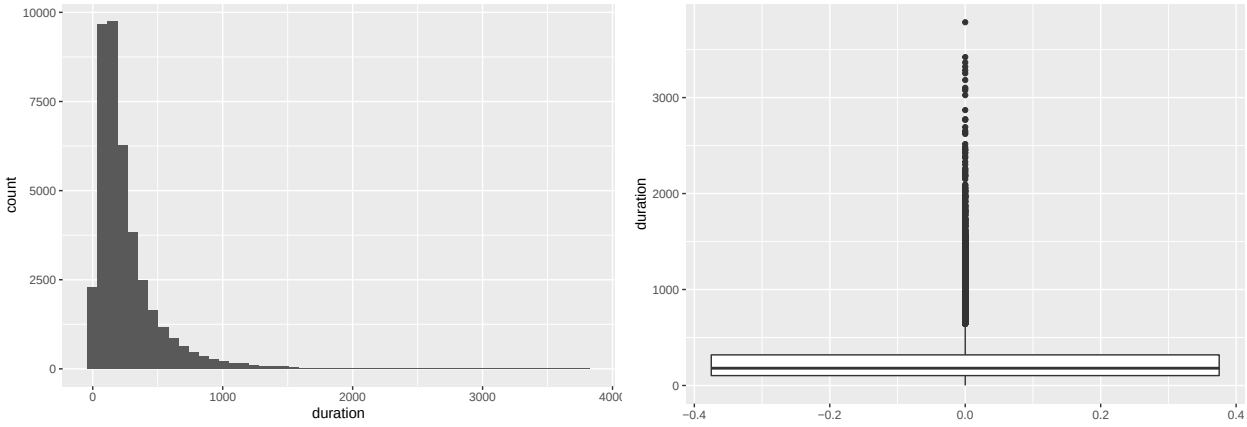
Numerical data The age histogram shows mean and median close together, the distribution is right skewed.



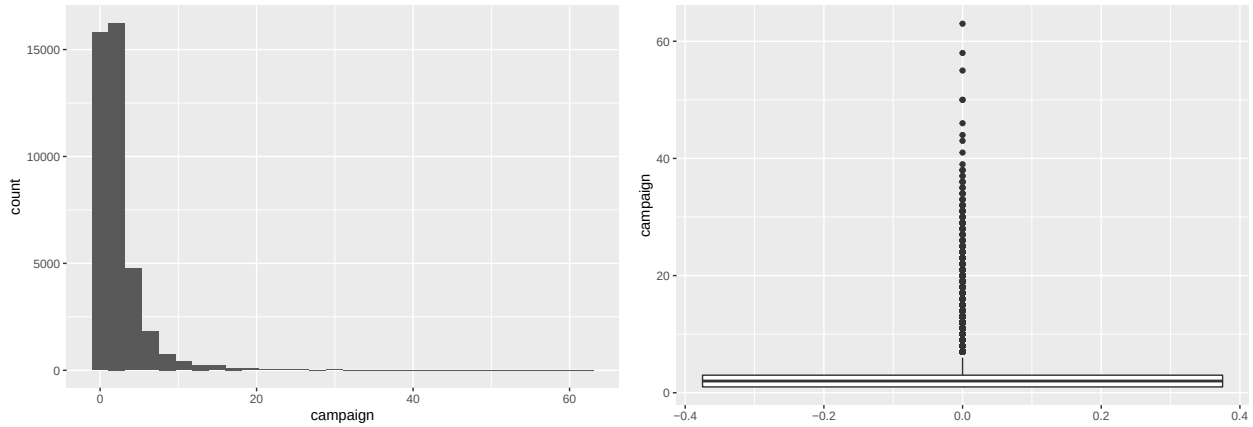
The balance distribution is strongly right skewed with many outliers.



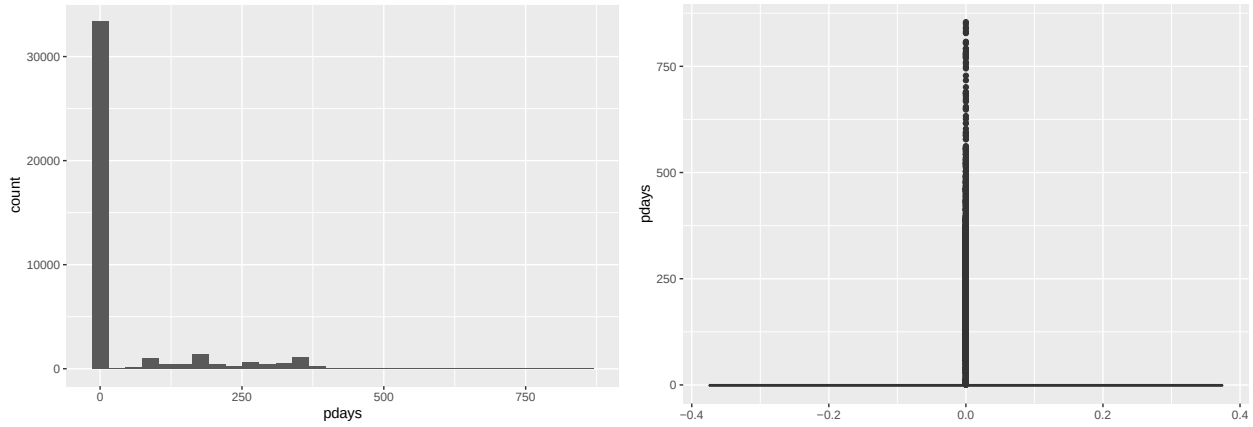
The contact duration data is also right skewed with many outliers.



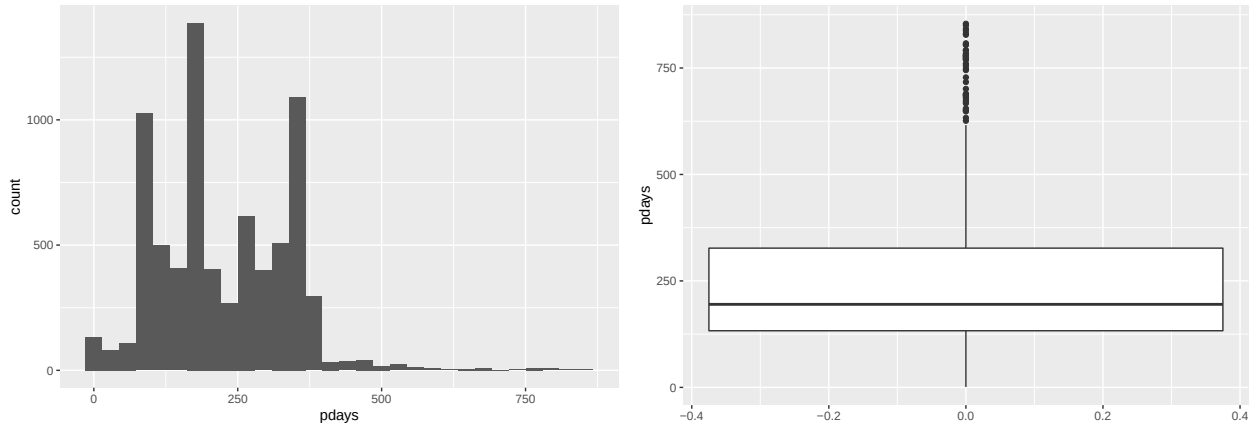
The same right skew is visible for the campaign data, which stores the number of contacts performed during this campaign and for this client.



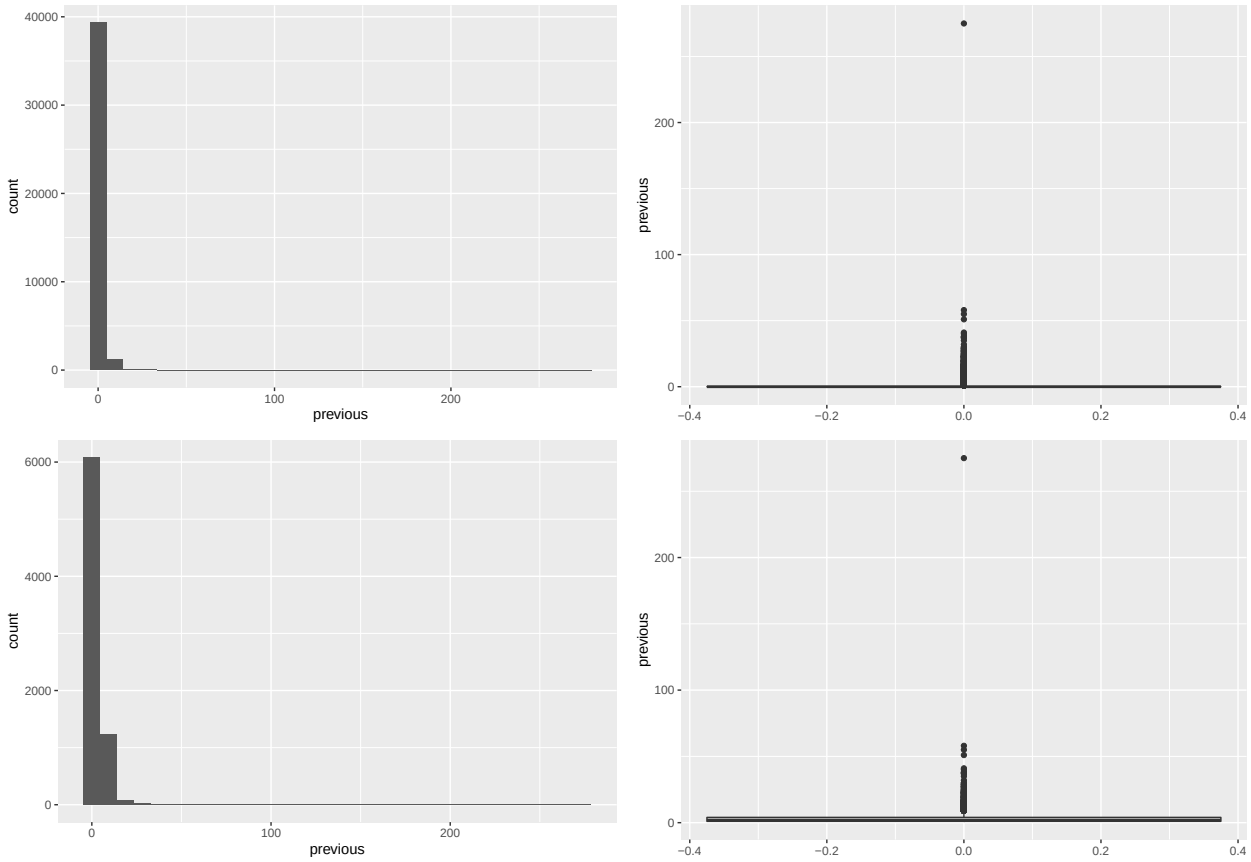
The pdays data consists of many -1 values and is right skewed with outliers. It stores the number of days that passed by after the client was last contacted. Here, -1 means the client was not previously contacted.



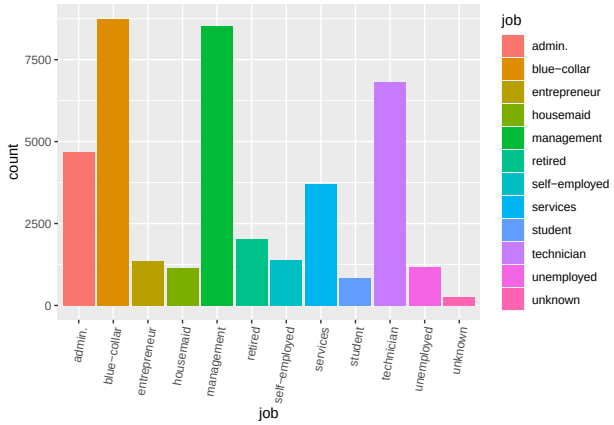
The outliers are still present when removing the -1 values.



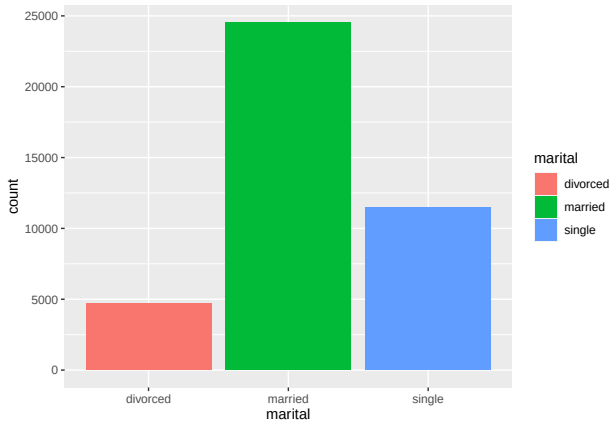
The "previous" variable is similar to the pdays variable from above, except that 0 values are used instead of -1. It stores the number of contacts performed before this campaign and for this client



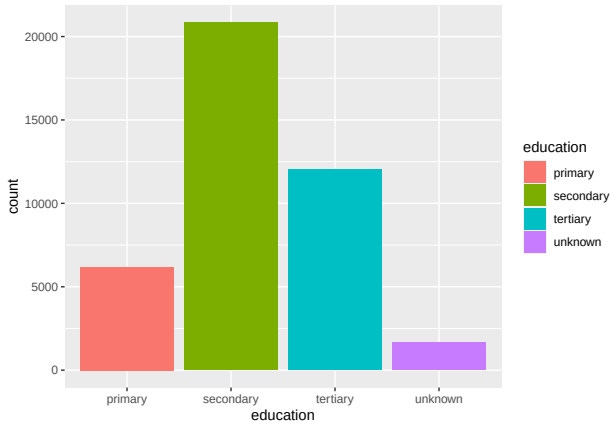
Categorical data Most people work in blue-collar jobs, management, technician, admin and services.



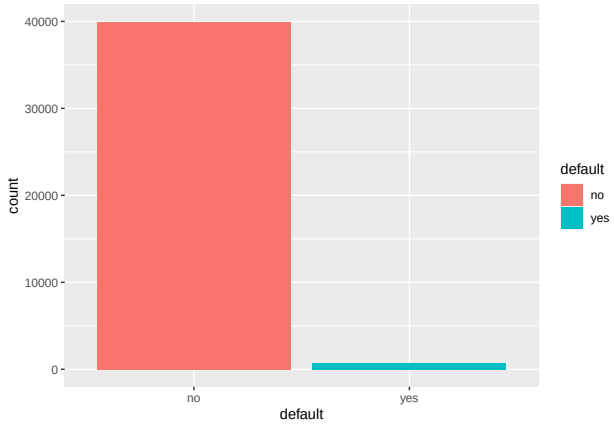
Most people in the dataset are married.



Most people completed at least secondary education.

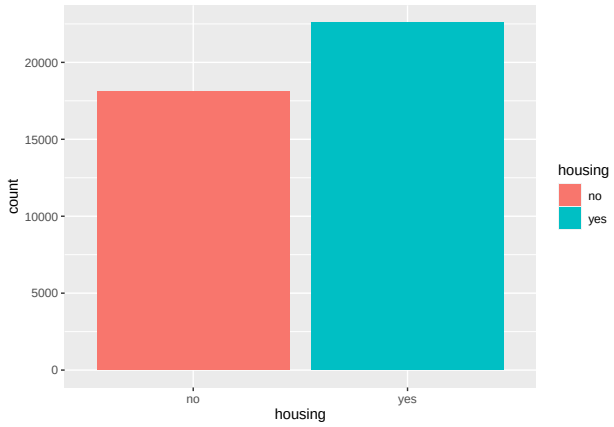


The majority of people did not default.

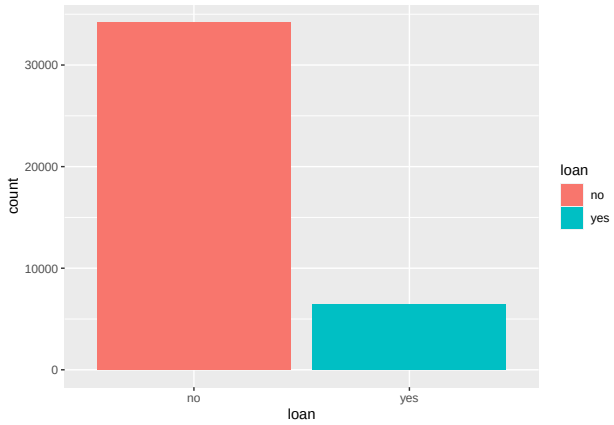


More than half of the people have housing loans, but the proportions are more similar than in other factors (55% yes versus 45% no).

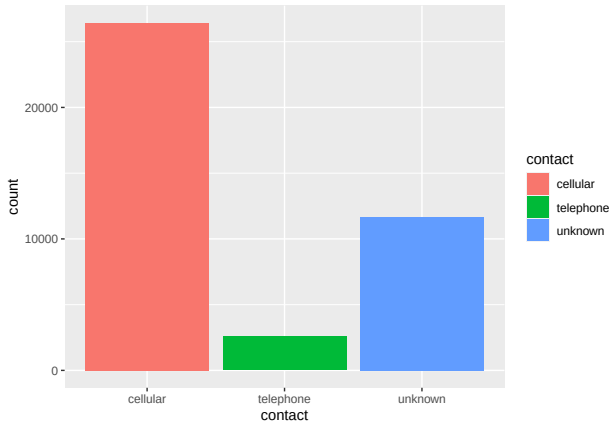
```
##
##    no   yes
## 18109 22580
```



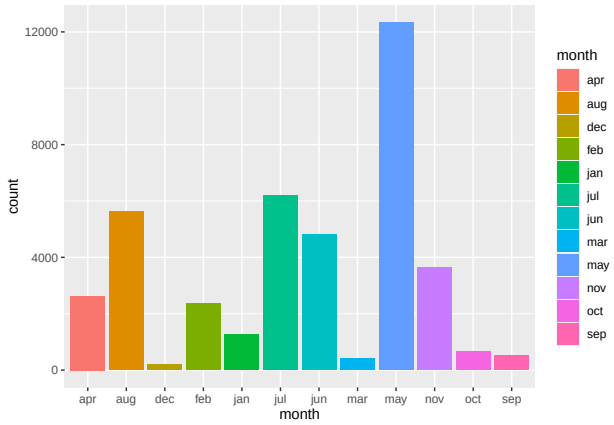
Most people do not have personal loans.



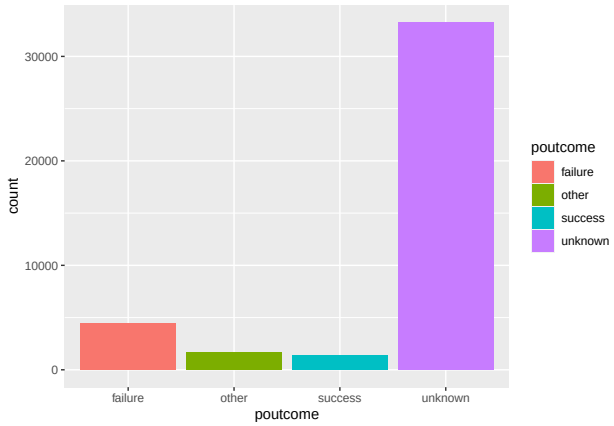
Most people were contacted using cellular (mobile phones), but the number of unknown values is relatively high.



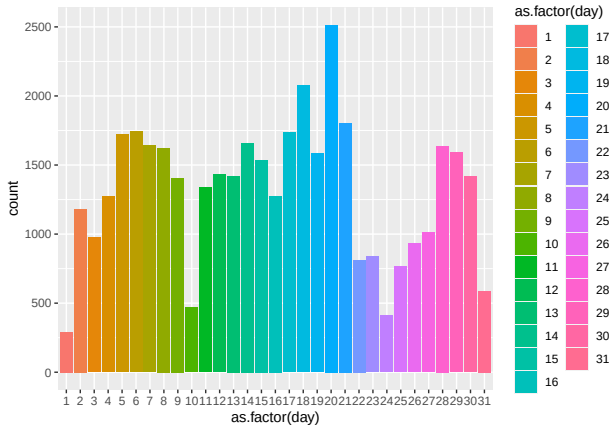
Most people were contacted during May, July, June and August, the fewest during December and March.



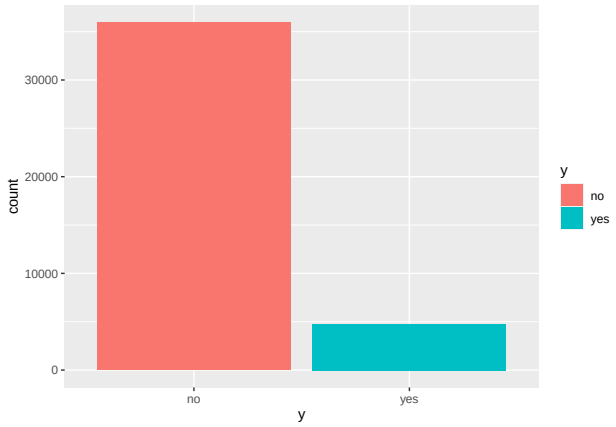
Previous outcome is mostly failure for known data, but most of the entries are unknown.



Most people were contacted during on 18th, 19th and 20th, the fewest on 1st, 10th and 24th day of the month.

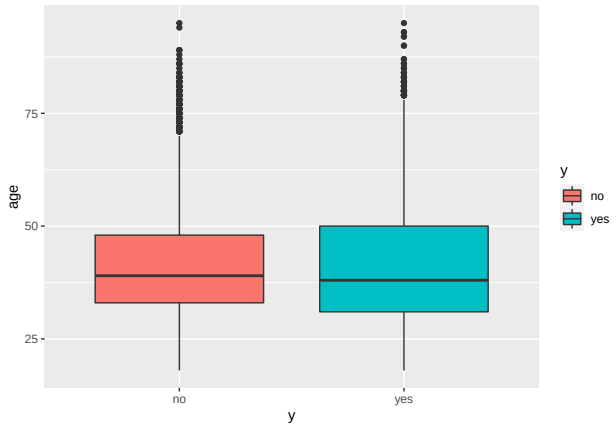


For the dependent variable, the outcome y, there are more "no" than "yes" values.

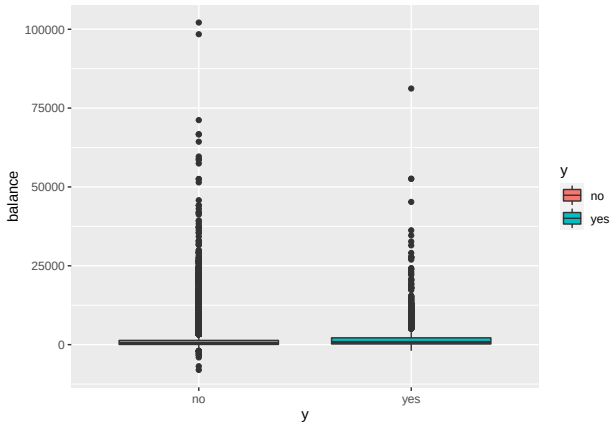


Bivariate plots When looking at the relationships between the features and the outcome y, we find the following aspects:

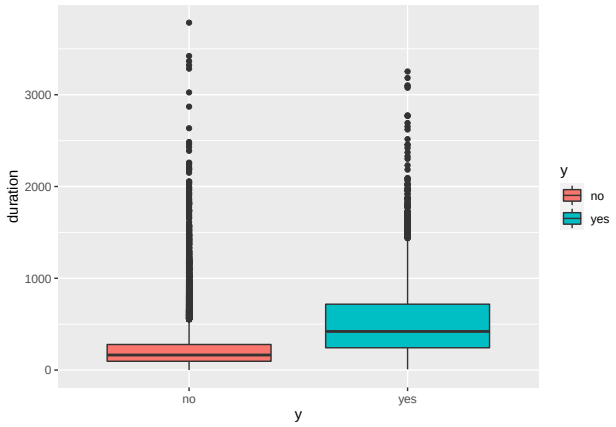
The median age for "yes" answers is slightly lower, but the data is spread wider.



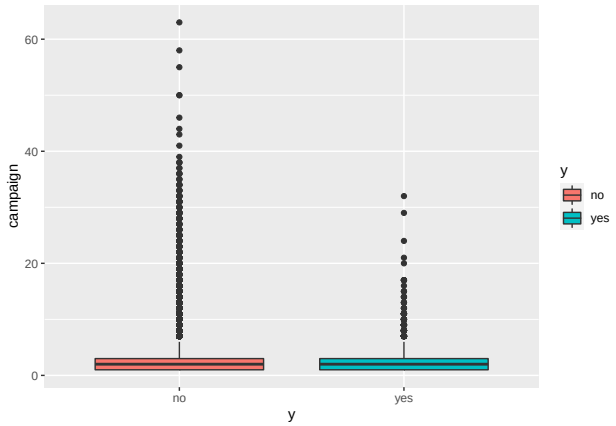
The balance distribution is similar for "yes" and "no," except higher outliers for "no."



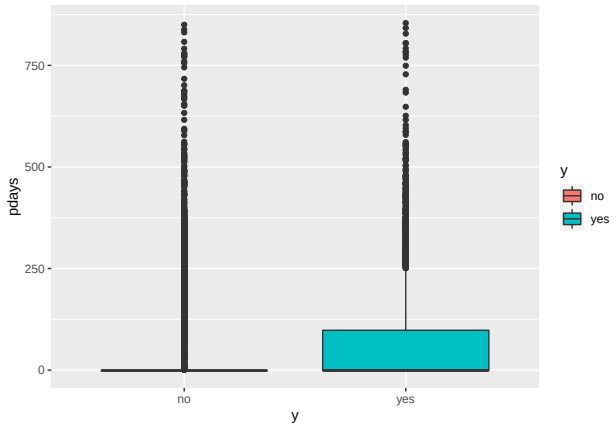
The median duration is higher for "yes" than for "no," even if we consider the wider spread of the data. This suggests that contact duration is a significant factor for the predicted outcome.



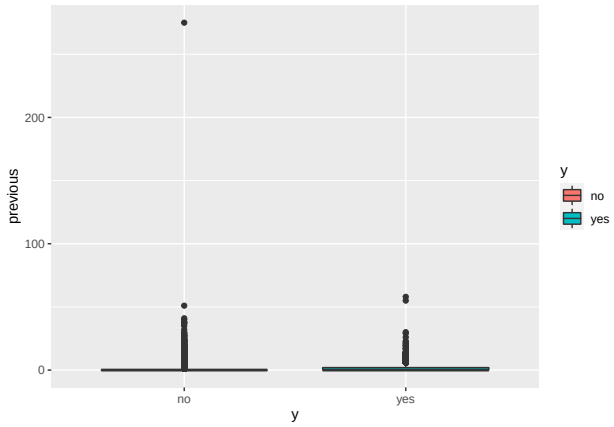
The campaign distribution is similar for “yes” and “no,” except higher outliers for “no.”



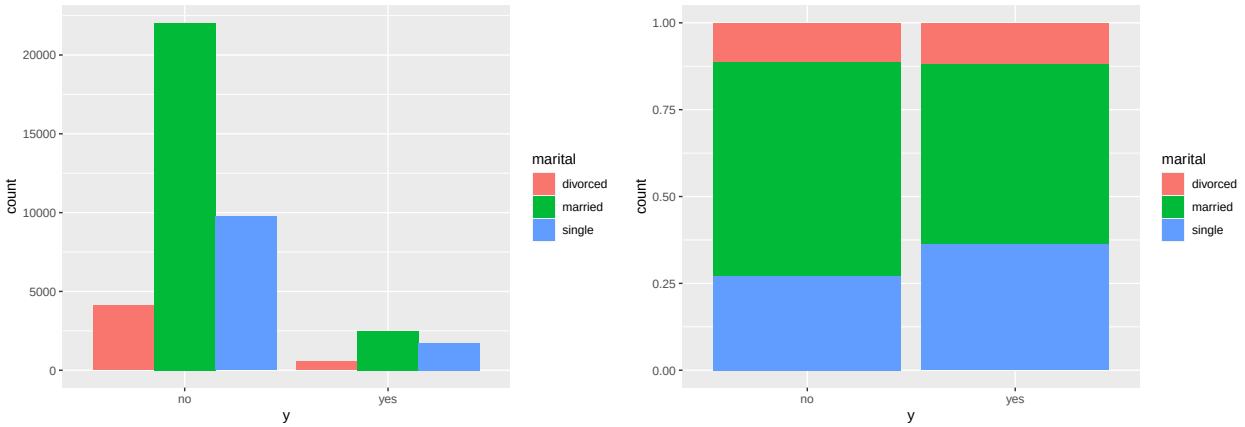
The pdays values for “yes” have wider spread.



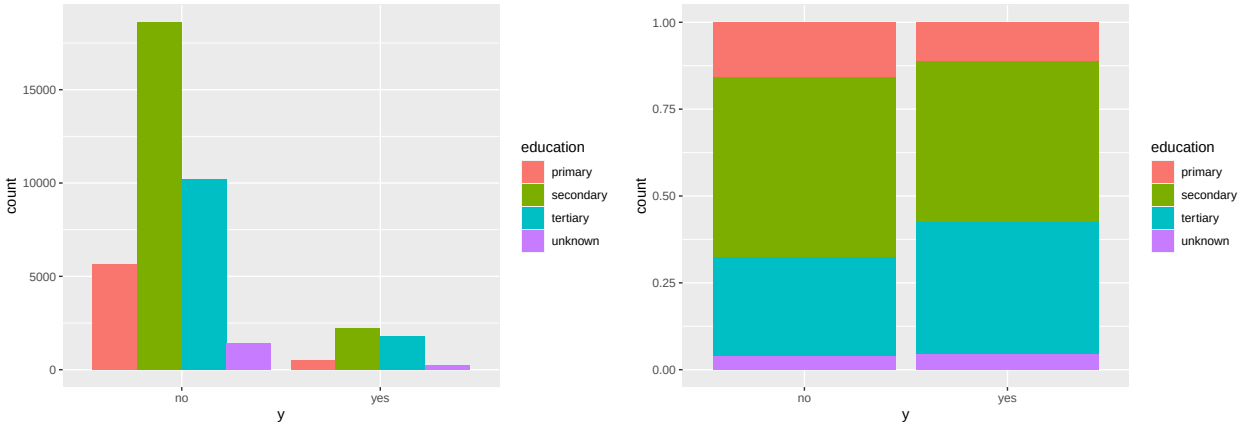
The “previous” distribution is similar for “yes” and “no,” except one high outlier for “no.”



While married people are the majority in both groups, single persons said "yes" more often.

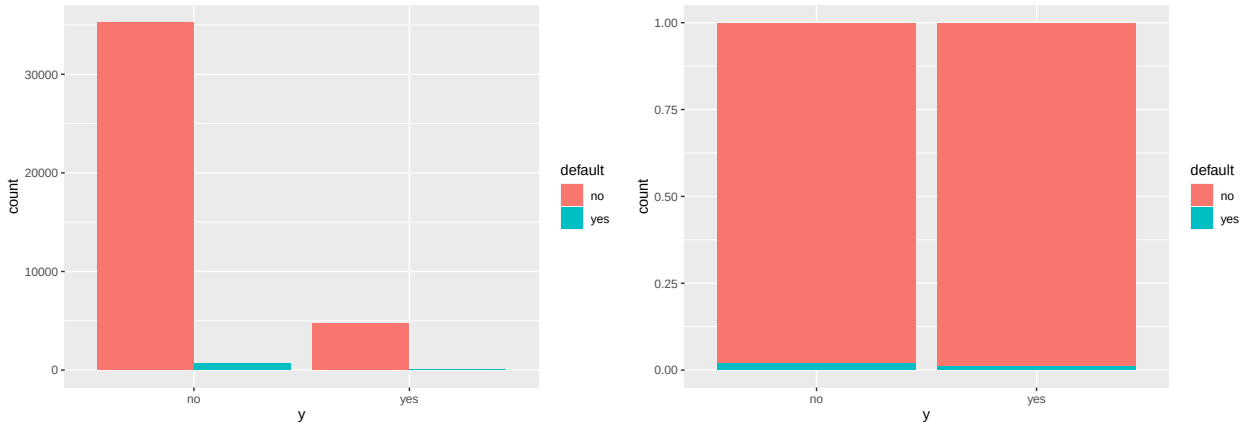


People with secondary-education are the majority in both groups. Tertiary-educated people said "yes" more often as proportion of answers, but primary-educated said "no" more often.

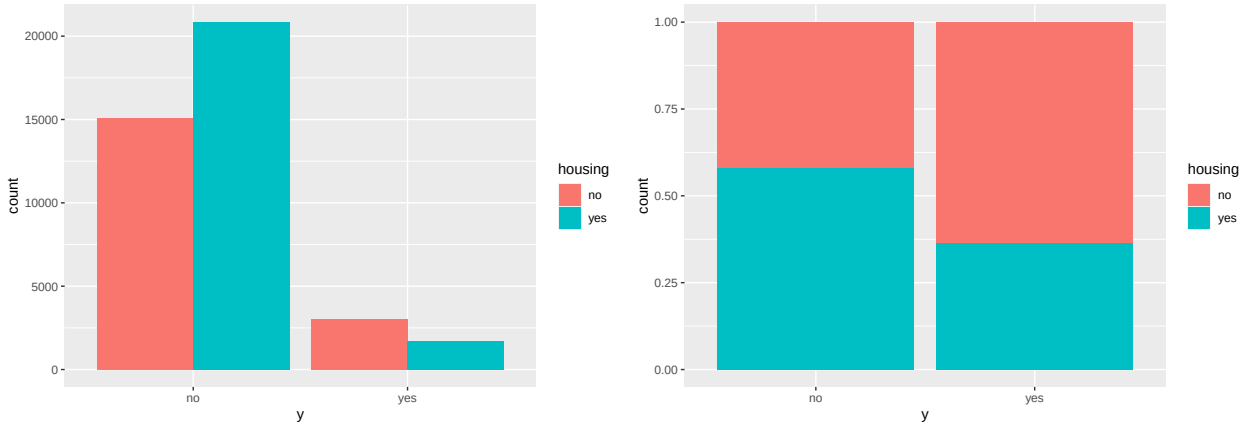


The majority of "yes" come from persons which did not default, but the situation is similar for "no" answers.

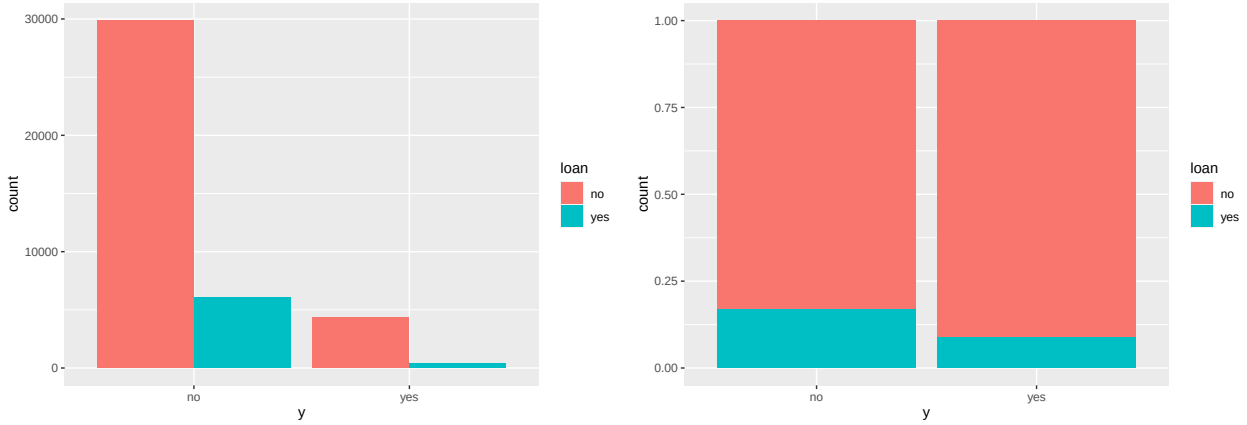
```
##
##          no   yes
##    no  35234   695
##    yes  4711    49
```



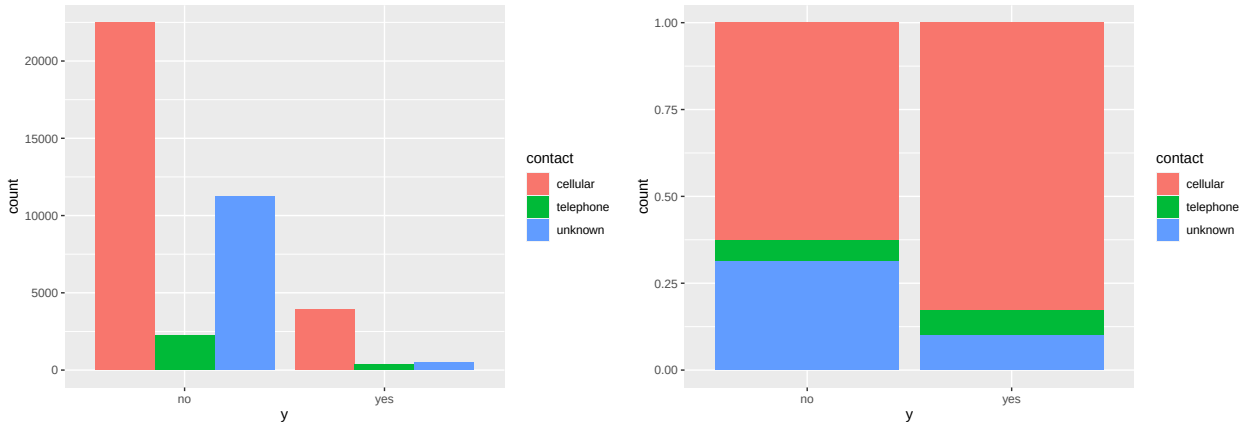
Most "yes" answers come from persons with no housing loans, while most "no" answers come from persons with housing loans. This seems aligned with the purpose of finding new customers.



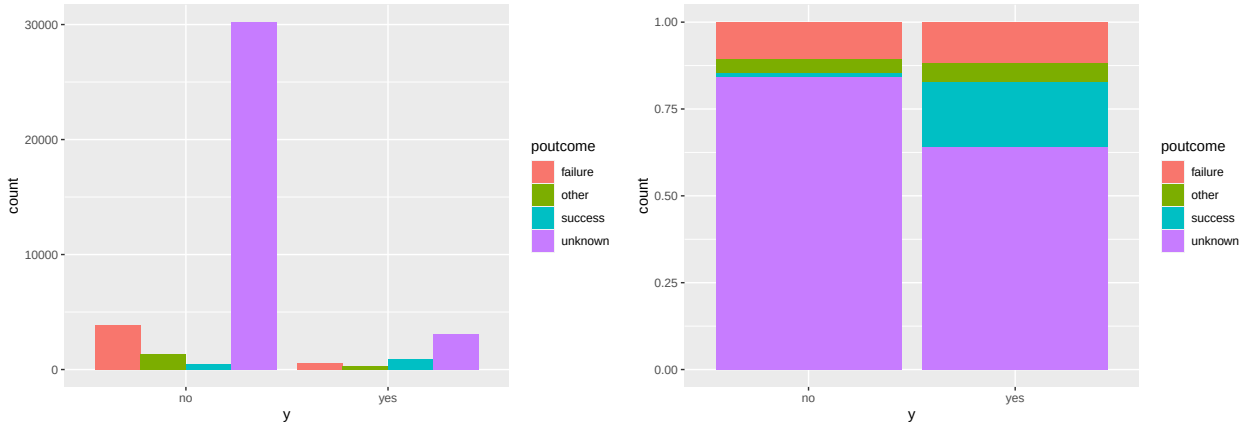
The majority of "yes" answers come from persons who do not have a loan. This seems aligned with the purpose of finding new customers.



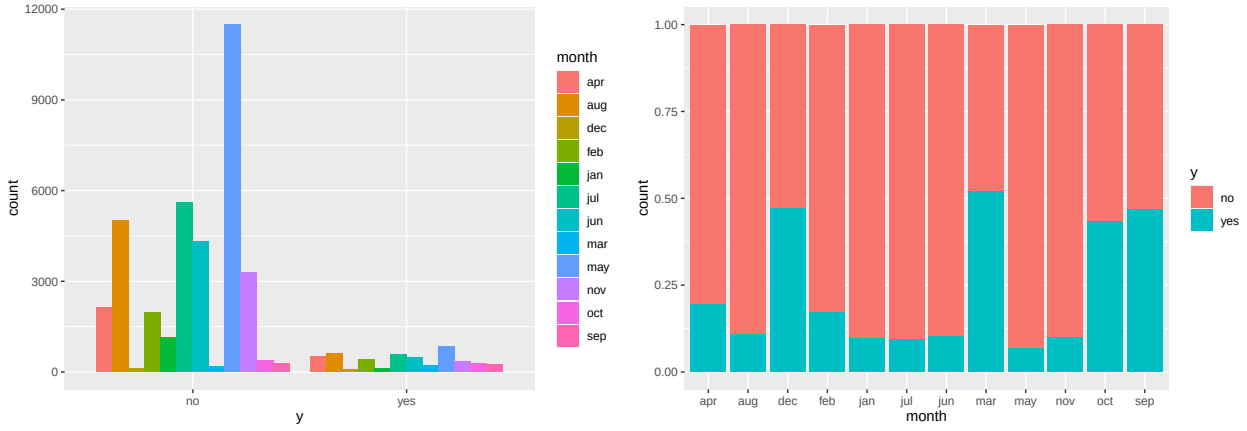
Most "yes" answers come from persons contacted via cellular(mobile phone).



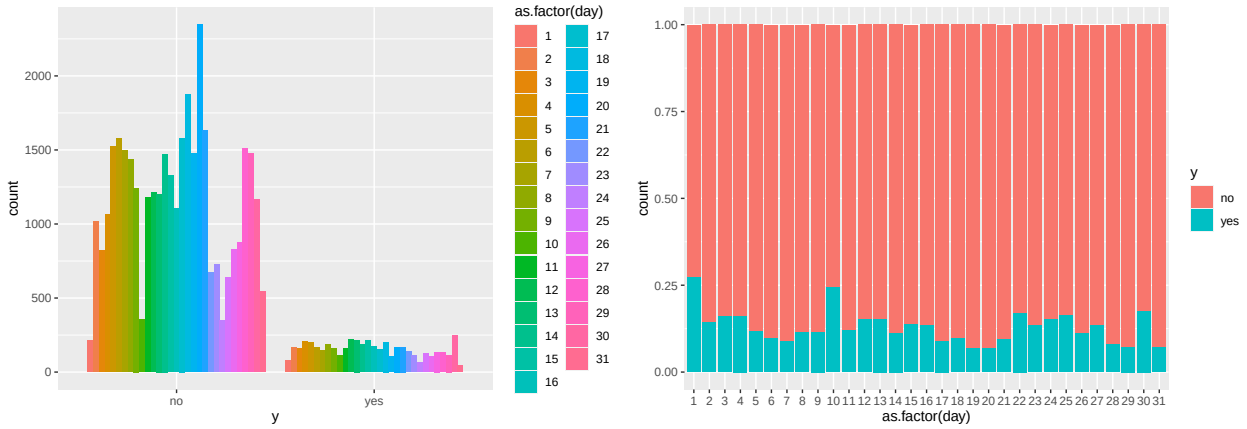
Many "yes" answers come from persons who had succesful previous outcomes. This might be related to a higher trust factor, but this is outside the scope of the analysis.



The highest proportion of yes is in March, Dec, Oct, Sep - these are months when fewer calls are made. During months with the highest number of calls, the proportion is relatively constant.



The highest proportion of yes is on the 1st, 10th and 30th of the month. While the 1st and 10th are days with fewer calls, the 30th is a day with many calls.



Correlation When looking at the correlation between numeric variables and the outcome y, we find a stronger positive correlation for duration, pdays, and previous.

```
##                [,1]
## age         0.02456114
## balance     0.05208930
## day        -0.02980028
## duration    0.39267830
## campaign   -0.07264472
## pdays       0.10372753
## previous    0.09305597
```

Models

Preparation: Undersampling to improve data balance

As noted previously, the dataset is imbalanced, including only ~12% "yes" outcomes. These outcomes, however, are the relevant ones from a business perspective. Acquiring new customers or extending the product range for existing customers are the goals of sales/marketing campaigns. In data with few observations for a specific class, learning the features which describe this class is more difficult than in data where each class has approximately the same number of observations. The reason is that similar features will be present in the other predominant classes. This makes it harder to distinguish the relevant features.

We generate a more balanced dataset by undersampling and use this for modeling along with the original dataset: the new dataset will include the "yes" outcomes and a similar number of "no" outcomes sampled from the original dataset.

```
# generate reduced, more balanced dataset
# split data by outcome
train.yes <- train.bank %>% filter(y == "yes")
train.no <- train.bank %>% filter(y == "no")
# undersample the "no" data to match the amount of "yes" data
set.seed(1, sample.kind="Rounding")
ssize <- dim(train.yes)[1] # get size of the "yes" data
# (alternatively - not done here: round size up to the nearest thousand)
rows.no <- sample(rownames(train.no), size = ssize, replace = FALSE) # sampled "no" data
reduced.no <- train.no[rows.no, ]
# merge yes and no data in a new, more balanced dataset
```

```
train.balanced <- rbind(train.yes, reduced.no)
summary(train.balanced)
```

```
##       age                job           marital         education    default
##  Min.   :18.0   management :2068   divorced:1098   primary  :1320   no :9387
##  1st Qu.:32.0   blue-collar:1747   married :5372   secondary:4711   yes: 133
##  Median :38.0   technician :1554   single  :3050   tertiary :3065
##  Mean   :41.1   admin.     :1138                   unknown  : 424
##  3rd Qu.:49.0   services   : 762
##  Max.   :95.0   retired    : 671
##                 (Other)    :1580
##     balance      housing     loan           contact          day
##  Min.   :-1944   no :5032   no :8314   cellular :6907   Min.   : 1.00
##  1st Qu.:  118   yes:4488   yes:1206   telephone: 644   1st Qu.: 8.00
##  Median :  557                         unknown  :1969   Median :15.00
##  Mean   : 1532                                          Mean   :15.42
##  3rd Qu.: 1740                                          3rd Qu.:21.00
##  Max.   :81204                                          Max.   :31.00
##
##      month         duration       campaign          pdays
##  may    :2407   Min.   :   0   Min.   : 1.000   Min.   : -1.00
##  jul    :1358   1st Qu.: 142   1st Qu.: 1.000   1st Qu.: -1.00
##  aug    :1276   Median : 259   Median : 2.000   Median : -1.00
##  jun    :1046   Mean   : 376   Mean   : 2.463   Mean   : 52.56
##  apr    : 800   3rd Qu.: 496   3rd Qu.: 3.000   3rd Qu.: 59.25
##  nov    : 714   Max.   :3253   Max.   :46.000   Max.   :854.00
##  (Other):1919
##     previous          poutcome      y
##  Min.   : 0.0000   failure:1059   no :4760
##  1st Qu.: 0.0000   other  : 460   yes:4760
##  Median : 0.0000   success: 949
##  Mean   : 0.8188   unknown:7052
##  3rd Qu.: 1.0000
##  Max.   :58.0000
##
```

Majority class for customers

A very simple model predicts "no" (the majority class) for each outcome.

The proportion of "no" outcomes is `rmean(train.bank$y == "no")` in the original training set, 0.5 in the more balanced dataset and 0.8830164 in the test set.

```
# simple model: always predict "no" - original dataset
mean(train.bank$y == "no") # 0.883
mean(test.bank$y == "no") # 0.883

# simple model: always predict "no" - balanced dataset
mean(train.balanced$y == "no")
mean(test.bank$y == "no") # 0.883
```

Model performance is determined using the confusion matrix and stored:

```r
# helper function which adds a row to the results dataframe
add_result <- function(result_df, meth, cmat, fscore){
  result_df <- bind_rows(result_df,
                         data.frame(method = meth,  acc = cmat$overall[["Accuracy"]],
                           sens = cmat$byClass[["Sensitivity"]], sp = cmat$byClass[["Specificity"]],
                           balanced.acc = cmat$byClass[["Balanced Accuracy"]] , Fscore = fscore))
  result_df
}

n <- length(test.bank$y)
# "trick" to generate y_hat with one value but two factor levels:
# create with one value of another level, then remove it
y_hat_naive <- as.factor( rep( c("no", "yes"), c(n, 1) ) )
y_hat_naive <- y_hat_naive[1:n]
cmat_naive <- confusionMatrix(y_hat_naive, test.bank$y)
# F-score
f_naive <- F_meas(data = y_hat_naive, reference = test.bank$y)

# original dataset
model_results <- data.frame(method = "Always no (one class)",  acc = cmat_naive$overall[["Accuracy"]],
                  sens = cmat_naive$byClass[["Sensitivity"]], sp = cmat_naive$byClass[["Specificity"]]
                  balanced.acc = cmat_naive$byClass[["Balanced Accuracy"]] , Fscore = f_naive)

# more balanced dataset
model_results_bal <- data.frame(method = "Always no (one class)",  acc = cmat_naive$overall[["Accuracy"]
                      sens = cmat_naive$byClass[["Sensitivity"]], sp = cmat_naive$byClass[["Specificity
                      balanced.acc = cmat_naive$byClass[["Balanced Accuracy"]] , Fscore = f_naive)
```

Confusion matrix interpretation

In order to better understand the confusion matrix and what metrics are relevant for all models, we analyze the matrix for this simple model.

```r
cmat_naive
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  3993  529
##        yes    0    0
##
##                Accuracy : 0.883
##                  95% CI : (0.8733, 0.8922)
##     No Information Rate : 0.883
##     P-Value [Acc > NIR] : 0.5116
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000
```

```
##             Specificity : 0.000
##          Pos Pred Value : 0.883
##          Neg Pred Value :   NaN
##              Prevalence : 0.883
##          Detection Rate : 0.883
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : no
##
```

The model has high accuracy, even though no "yes" reference answer is guessed correctly. This seems unusual at first, but all "no" reference answers are guessed correctly, which is shown in sensitivity 1.0 and high accuracy.

The absence of correct "yes" answers is reflected by specificity 0.0 and overall balanced accuracy of only 0.5. Note that in R the 'Positive' class for the confusion matrix is "no." This means, accuracy is not the most suitable metric for model evaluation in this scenario: specificity and balanced accuracy are better choices.

Logistic regression model

For classification problems with binary outcome (yes/no) logistic regression is the usual, efficient choice. The model yields a prediction based on probability. A probability threshold is used for "yes"/"no" decision.

For the original data we have:

```
# logistic regression (with all variables) - original dataset
set.seed(1, sample.kind = "Rounding")
fit_glm <- train(y ~ ., method = "glm", data = train.bank)
y_hat_glm <- predict(fit_glm, test.bank, type = "raw")
confusionMatrix(y_hat_glm, test.bank$y)$overall[["Accuracy"]] # 0.9029191
```

```
## [1] 0.9029191
```

```
# but yes is detected in too few cases (low specificity: 0.3762) - due to data imbalance
cmat_glm <- confusionMatrix(y_hat_glm, test.bank$y)
# F-score
f_glm <- F_meas(data = y_hat_glm, reference = test.bank$y)

model_results <- add_result(model_results,"Logistic Regression", cmat_glm, f_glm)
```

For the more balanced data we find:

```
# logistic regression (with all variables) - balanced dataset
set.seed(1, sample.kind = "Rounding")
fit_glm <- train(y ~ ., method = "glm", data = train.balanced)
y_hat_glm <- predict(fit_glm, test.bank, type = "raw")
confusionMatrix(y_hat_glm, test.bank$y)$overall[["Accuracy"]] # 0.8427687
```

```
## [1] 0.8427687
```

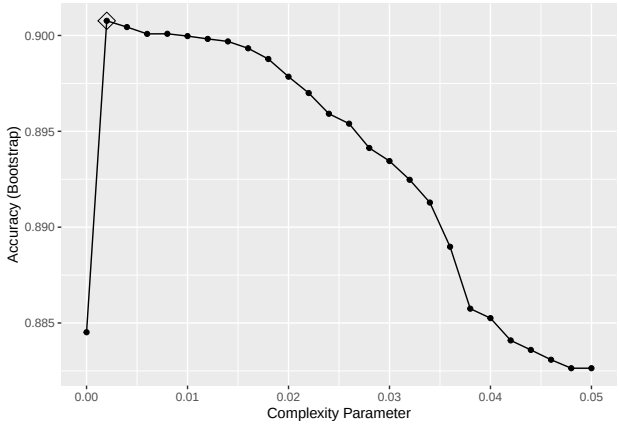
```
# yes is detected in many cases (high specificity: 0.8318) - due to better data balance
cmat_glm <- confusionMatrix(y_hat_glm, test.bank$y)
# F-score
f_glm <- F_meas(data = y_hat_glm, reference = test.bank$y)

model_results_bal <- add_result(model_results_bal,"Logistic Regression", cmat_glm, f_glm)
```

Classification/Decision tree and Random forest models

Classification trees and random forests are another popular choice for this type of problem. Trees provide models that are relatively easy to explain. Random forests average predictions over multiple trees and can generate a more reliable prediction. The random forest model also provides a ranking of feature importance for the chosen model, which can be used to better understand the classification mechanism. Both model types can detect non-linear relationships in the data.

Classification tree For the original dataset we have:

```
# decision trees - original dataset
set.seed(10, sample.kind = "Rounding")
fit_cl_tree <- train(y ~ ., method = "rpart", data = train.bank,
                     tuneGrid = data.frame(cp = seq(0, 0.05, 0.002)))
ggplot(fit_cl_tree, highlight = TRUE)
```

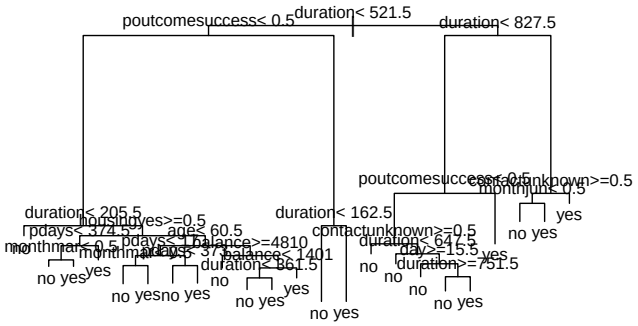



```
# fit_cl_tree$bestTune # cp = 0.002 -> optimal value for the complexity parameter
y_hat_cl_tree <- predict(fit_cl_tree, test.bank, type = "raw")
confusionMatrix(y_hat_cl_tree, test.bank$y)$overall["Accuracy"] # 0.9071207
```

```
##  Accuracy
## 0.9071207
```

```
# but yes is detected in too few cases (low specificity: 0.4234) - due to data imbalance
cmat_tree <- confusionMatrix(y_hat_cl_tree, test.bank$y)
# fit_cl_tree$finalModel
```

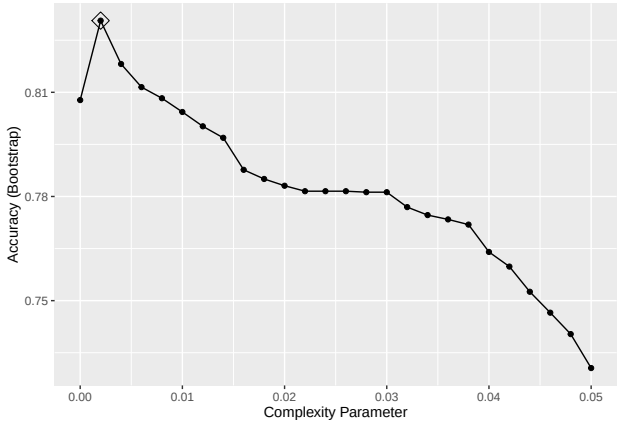
```
# plot the tree
plot(fit_cl_tree$finalModel, margin = 0.1)
text(fit_cl_tree$finalModel, cex = 0.75)
```




```
# F-score
f_tree <- F_meas(data = y_hat_cl_tree, reference = test.bank$y)

model_results <- add_result(model_results, "Classification Tree", cmat_tree, f_tree)
```

For the more balanced dataset we have:

```
# decision trees - balanced dataset
set.seed(10, sample.kind = "Rounding")
fit_cl_tree <- train(y ~ ., method = "rpart", data = train.balanced, tuneGrid = data.frame(cp = seq(0, 
ggplot(fit_cl_tree, highlight = TRUE)
```

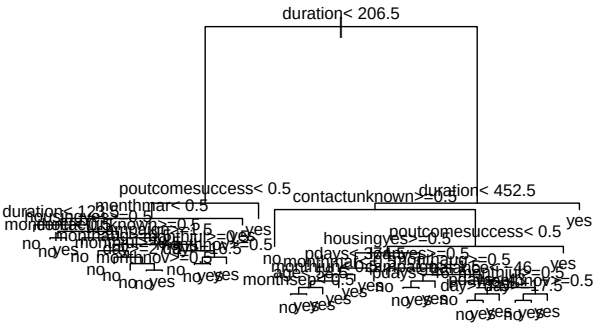


```
# fit_cl_tree$bestTune # cp = 0.002 -> optimal value for the complexity parameter
y_hat_cl_tree <- predict(fit_cl_tree, test.bank, type = "raw")
confusionMatrix(y_hat_cl_tree, test.bank$y)$overall["Accuracy"] # 0.8318
```

```
##  Accuracy
## 0.8272888
```

```
# yes is detected in many cases (high specificity: 0.8904) - due to better data balance
cmat_tree <- confusionMatrix(y_hat_cl_tree, test.bank$y)
# fit_cl_tree$finalModel
```

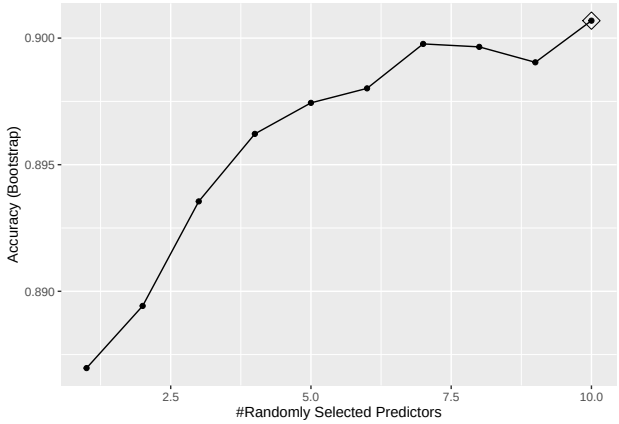
```
# plot the tree
plot(fit_cl_tree$finalModel, margin = 0.1)
text(fit_cl_tree$finalModel, cex = 0.75)
```




```
# F-score
f_tree <- F_meas(data = y_hat_cl_tree, reference = test.bank$y)

model_results_bal <- add_result(model_results_bal, "Classification Tree", cmat_tree, f_tree)
```

Random forest The model requires a large amount of computing time and memory if we use the original dataset. Therefore only a subset of the original data was used for training.

```
# random forest (classification) -> warning: long runtime - original dataset
set.seed(14, sample.kind = "Rounding")
# fit_rf <- train(y ~ ., method = "rf", data = train.bank, tuneGrid = data.frame(mtry = seq(1, 7)), ntr
# training with reduced dataset (bank.small)
fit_rf <- train(y ~ ., method = "rf", data = bank.small, tuneGrid = data.frame(mtry = seq(1, 10)),
                ntree = 100)
ggplot(fit_rf, highlight = TRUE)
```




```
# fit_rf$bestTune # mtry = 8 -> optimal value for mtry parameter
y_hat_rf <- predict(fit_rf, test.bank, type = "raw")
confusionMatrix(y_hat_rf, test.bank$y)$overall["Accuracy"] # 0.9170721
```

```
##  Accuracy
## 0.9159664
```

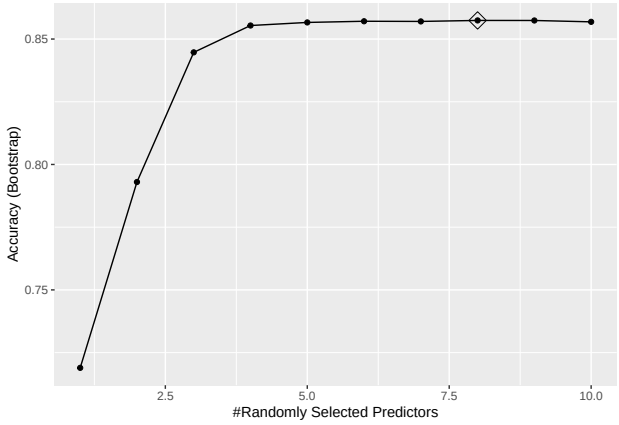
```
cmat_rf <- confusionMatrix(y_hat_rf, test.bank$y)
# but yes is detected in too few cases (low specificity: 0.4783) - due to data imbalance
# importance of variables in the random forest model
imp <- varImp(fit_rf)
imp
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 42)
##
##                     Overall
## duration            100.000
## balance              32.696
## age                  31.705
## day                  29.083
## poutcomesuccess      15.980
## pdays                14.540
## campaign             12.033
## previous              7.508
## monthoct              5.900
## housingyes            5.518
## maritalmarried        5.051
## educationtertiary     4.568
## monthaug              4.278
## contactunknown        4.142
## educationsecondary    3.960
## monthjun              3.784
## jobtechnician         3.753
## monthmar              3.732
## jobmanagement         3.556
## monthmay              3.553
```

```r
# F-score
f_rf <- F_meas(data = y_hat_rf, reference = test.bank$y)

model_results <- add_result(model_results, "Random Forest",  cmat_rf, f_rf)
```

The more balanced dataset is smaller and has lower resource usage.

```r
# random forest (classification) -> warning: long runtime - balanced dataset
set.seed(14, sample.kind = "Rounding")
# training with balanced dataset
fit_rf <- train(y ~ ., method = "rf", data = train.balanced, tuneGrid = data.frame(mtry = seq(1, 10)),
                ntree = 100)
ggplot(fit_rf, highlight = TRUE)
```




```r
# fit_rf$bestTune # mtry = 8 -> optimal value for mtry parameter
y_hat_rf <- predict(fit_rf, test.bank, type = "raw")
confusionMatrix(y_hat_rf, test.bank$y)$overall["Accuracy"] # 0.8398939
```

```
##  Accuracy
## 0.8398939
```

```r
cmat_rf <- confusionMatrix(y_hat_rf, test.bank$y)
# yes is detected in many cases (high specificity: 0.9130) - due to better data balance
# importance of variables in the random forest model
imp <- varImp(fit_rf)
imp
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 42)
##
##                    Overall
## duration           100.000
## balance             19.938
## age                 19.399
## day                 17.281
## poutcomesuccess     10.827
```

```
## contactunknown     10.436
## pdays               9.998
## campaign            8.651
## housingyes          7.469
## previous            5.002
## monthmay            3.665
## monthmar            3.404
## monthjul            3.198
## monthaug            3.091
## monthoct            2.812
## poutcomeunknown     2.767
## maritalmarried      2.625
## educationtertiary   2.520
## loanyes             2.483
## monthjun            2.398
```

```
# F-score
f_rf <- F_meas(data = y_hat_rf, reference = test.bank$y)

model_results_bal <- add_result(model_results_bal, "Random Forest",  cmat_rf, f_rf)
```

In both cases, the duration of contact, balance, age, day and successful previous outcome are the best predictors.

Support vector machines (SVM) model

In order to detect a proper separation between the prediction classes, we use Support Vector Machines (SVM) for categorical outcomes. These models maximize the margin between class boundaries. An important part of the SVM model is the kernel: either a linear or a radial kernel can be used. We use both kernel types separately. In addition, the gamma parameter for the radial kernel is provided. The radial kernel is more suitable for this scenario, as shown below.

For the original dataset we have:

```
# svm - original dataset
set.seed(1, sample.kind = "Rounding")
fit_svm <- svm(y ~ ., data = train.bank, kernel = "linear") # linear kernel
y_hat_svm <- predict(fit_svm, test.bank)
confusionMatrix(y_hat_svm, test.bank$y)$overall[["Accuracy"]] # 0.8911986
```

```
## [1] 0.8911986
```

```
# confusionMatrix(y_hat_svm, test.bank$y)
# but yes is detected in very few cases (low specificity: 0.1796) - due to data imbalance
fit_svmr <- svm(y ~ ., data = train.bank, kernel = "radial",
                gamma = 0.1) # radial kernel, choose gamma
y_hat_svm <- predict(fit_svmr, test.bank)
confusionMatrix(y_hat_svm, test.bank$y)$overall[["Accuracy"]] # 0.9066785
```

```
## [1] 0.9066785
```

```
cmat_svm <- confusionMatrix(y_hat_svm, test.bank$y)
# but yes is detected in very few cases (low specificity: 0.3932) - due to data imbalance
# F-score
f_svm <- F_meas(data = y_hat_svm, reference = test.bank$y)

model_results <- add_result(model_results, "SVM", cmat_svm, f_svm)
```

For the more balanced dataset we have:

```
# svm - balanced dataset
set.seed(1, sample.kind = "Rounding")
fit_svm <- svm(y ~ ., data = train.balanced, kernel = "linear") # linear kernel
y_hat_svm <- predict(fit_svm, test.bank)
confusionMatrix(y_hat_svm, test.bank$y)$overall[["Accuracy"]] # 0.8348076
```

```
## [1] 0.8348076
```

```
# confusionMatrix(y_hat_svm, test.bank$y)
# yes is detected in many cases (high specificity: 0.8507) - due to better data balance
fit_svmr <- svm(y ~ ., data = train.balanced, kernel = "radial",
                gamma = 0.1) # radial kernel, choose gamma
y_hat_svm <- predict(fit_svmr, test.bank)
confusionMatrix(y_hat_svm, test.bank$y)$overall[["Accuracy"]] # 0.8283945
```

```
## [1] 0.8283945
```

```
cmat_svm <- confusionMatrix(y_hat_svm, test.bank$y)
# yes is detected in many cases (high specificity: 0.9093) - due to better data balance
# F-score
f_svm <- F_meas(data = y_hat_svm, reference = test.bank$y)

model_results_bal <- add_result(model_results_bal, "SVM", cmat_svm, f_svm)
```

Models based on discriminant analysis

Discriminant analysis methods allow the model/algorithm to discriminate between relevant classes by selecting class means and maximizing the separation of these means. We use both linear and quadratic discriminant models for the data.

Linear discriminant analysis For the original dataset we have:

```
# lda - original dataset
set.seed(1, sample.kind = "Rounding")
fit_lda <- train(y ~ ., method = "lda", data = train.bank)
y_hat_lda <- predict(fit_lda, test.bank, type = "raw")
confusionMatrix(y_hat_lda, test.bank$y)$overall[["Accuracy"]] # 0.9029191
```

```
## [1] 0.9029191
```

```
cmat_lda <- confusionMatrix(y_hat_lda, test.bank$y)
# but yes is detected in too few cases (low specificity: 0.4820) - due to data imbalance
# F-score
f_lda <- F_meas(data = y_hat_lda, reference = test.bank$y)

model_results <- add_result(model_results, "LDA", cmat_lda, f_lda)
```

For the more balanced dataset we have:

```
# lda - balanced dataset
set.seed(1, sample.kind = "Rounding")
fit_lda <- train(y ~ ., method = "lda", data = train.balanced)
y_hat_lda <- predict(fit_lda, test.bank, type = "raw")
confusionMatrix(y_hat_lda, test.bank$y)$overall[["Accuracy"]] # 0.8549314
```

```
## [1] 0.8549314
```

```
cmat_lda <- confusionMatrix(y_hat_lda, test.bank$y)
# yes is detected in relatively many cases (specificity: 0.8053)
# F-score
f_lda <- F_meas(data = y_hat_lda, reference = test.bank$y)

model_results_bal <- add_result(model_results_bal, "LDA", cmat_lda, f_lda)
```

Quadratic discriminant analysis For the original dataset we have:

```
# qda - original dataset
set.seed(1, sample.kind = "Rounding")
fit_qda <- train(y ~ ., method = "qda", data = train.bank)
y_hat_qda <- predict(fit_qda, test.bank, type = "raw")
confusionMatrix(y_hat_qda, test.bank$y)$overall[["Accuracy"]] # 0.8690845
```

```
## [1] 0.8690845
```

```
cmat_qda <- confusionMatrix(y_hat_qda, test.bank$y)
# but yes is detected in too few cases (low specificity: 0.5047) - due to data imbalance
# F-score
f_qda <- F_meas(data = y_hat_qda, reference = test.bank$y)

model_results <- add_result(model_results, "QDA", cmat_qda, f_qda)
```

For the more balanced dataset we have:

```
# qda - balanced dataset
set.seed(1, sample.kind = "Rounding")
fit_qda <- train(y ~ ., method = "qda", data = train.balanced)
y_hat_qda <- predict(fit_qda, test.bank, type = "raw")
confusionMatrix(y_hat_qda, test.bank$y)$overall[["Accuracy"]] # 0.8533835
```

```
## [1] 0.8533835
```

```
cmat_qda <- confusionMatrix(y_hat_qda, test.bank$y)
# yes is detected in relatively few cases (specificity: 0.5784)
# F-score
f_qda <- F_meas(data = y_hat_qda, reference = test.bank$y)

model_results_bal <- add_result(model_results_bal, "QDA", cmat_qda, f_qda)
```

K-Nearest Neighbors (KNN) model

With K-nearest neighbors we use for a given customer the outcomes of its k nearest neighbors to predict the outcome class. The value of k is usually tuned and model performance can be improved by combining tuning with cross-validation. The model also requires a distance metric - to define what is "near."

For the original dataset we have:

```
# knn - original dataset
# using cross-validation to find the value of k (number of neighbors used)
set.seed(8, sample.kind = "Rounding")
control <- trainControl(method = "cv", number = 10)
# use 10-fold cross validation -> each partition consists of 10% of the total
# train_knn_cv <- train(y ~ ., method = "knn", data = train.bank,
#                       tuneGrid = data.frame(k = seq(3, 51, 2)),
#                       trControl = control)
# training with reduced dataset (bank.small)
train_knn_cv <- train(y ~ ., method = "knn", data = bank.small,
                      tuneGrid = data.frame(k = seq(3, 51, 2)),
                      trControl = control)
# ggplot(train_knn_cv, highlight = TRUE)
# train_knn_cv$bestTune # k = 37 -> optimal value for the number of neighbors
y_hat_knn_cv <- predict(train_knn_cv, test.bank, type = "raw")
confusionMatrix(y_hat_knn_cv, test.bank$y)$overall["Accuracy"] # 0.8865546
```

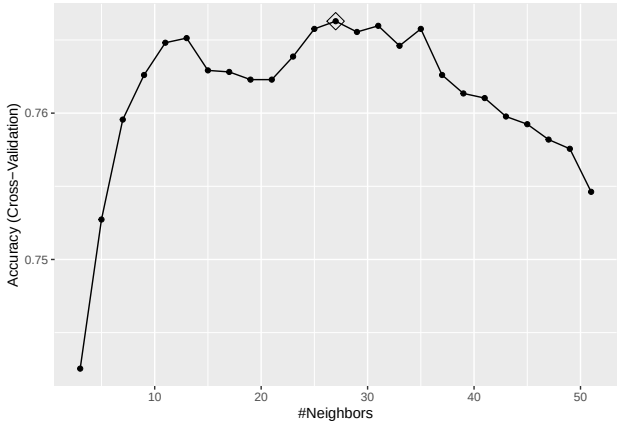
```
##  Accuracy
## 0.8865546
```

```
cmat_knn_cv <- confusionMatrix(y_hat_knn_cv, test.bank$y)
# but yes is detected in few cases (low specificity: 0.1210) - due to data imbalance
# F-score
f_knn_cv <- F_meas(data = y_hat_knn_cv, reference = test.bank$y)

model_results <- add_result(model_results, "KNN (with CV)",  cmat_knn_cv,
                            f_knn_cv)
```

For the more balanced dataset we have:

```
# knn - balanced dataset
# using cross-validation to find the value of k (number of neighbors used)
set.seed(8, sample.kind = "Rounding")
control <- trainControl(method = "cv", number = 10)
# use 10-fold cross validation -> each partition consists of 10% of the total
# training with balanced dataset
train_knn_cv <- train(y ~ ., method = "knn", data = train.balanced,
```

```
                      tuneGrid = data.frame(k = seq(3, 51, 2)),
                      trControl = control)
ggplot(train_knn_cv, highlight = TRUE)
```




```
# train_knn_cv$bestTune # k = 27 -> optimal value for the number of neighbors
y_hat_knn_cv <- predict(train_knn_cv, test.bank, type = "raw")
confusionMatrix(y_hat_knn_cv, test.bank$y)$overall["Accuracy"] # 0.7720035
```

```
##  Accuracy
## 0.7720035
```

```
cmat_knn_cv <- confusionMatrix(y_hat_knn_cv, test.bank$y)
# yes is detected in relatively many cases (specificity: 0.7543)
# F-score
f_knn_cv <- F_meas(data = y_hat_knn_cv, reference = test.bank$y)

model_results_bal <- add_result(model_results_bal, "KNN (with CV)",  cmat_knn_cv,
                                f_knn_cv)
```

Ensemble

Ensemble methods work by aggregating prediction results from different models. Then amjority vote is used to generate the final outcome of the ensemble. It may improve model performance by averaging multiple results and cancelling out errors, but this is not always the case.

For the bank marketing data and our models, the ensemble model doesn't improve results significantly. Performance metrics can be found in the next section.

Results

The following table shows various metrics which are relevant for classification problems:

- accuracy $(\frac{TP+TN}{TP+TN+FP+FN})$
- sensitivity $(\frac{TP}{TP+FN})$

- specificity ($\frac{TN}{FP+TN}$)
- balanced accuracy ($\frac{sensitivity + specificity}{2}$)
- F-score (combines precision and recall)

For the original dataset we have the following results:

method	acc	sens	sp	balanced.acc	Fscore
Always no (one class)	0.8830164	1.0000000	0.0000000	0.5000000	0.9378743
Logistic Regression	0.9029191	0.9727022	0.3761815	0.6744419	0.9465091
Classification Tree	0.9071207	0.9711996	0.4234405	0.6973200	0.9486301
Random Forest	0.9159664	0.9754570	0.4669187	0.7211879	0.9534884
SVM	0.9066785	0.9747057	0.3931947	0.6839502	0.9485742
LDA	0.9029191	0.9586777	0.4820416	0.7203596	0.9457690
QDA	0.8690845	0.9173554	0.5047259	0.7110406	0.9252336
KNN (with CV)	0.8865546	0.9879790	0.1209830	0.5544810	0.9389504
Ensemble (w/o naive)	0.9080053	0.9724518	0.4215501	0.6970009	0.9491567

The more balanced dataset provides the following metrics:

method	acc	sens	sp	balanced.acc	Fscore
Always no (one class)	0.8830164	1.0000000	0.0000000	0.5000000	0.9378743
Logistic Regression	0.8427687	0.8442274	0.8317580	0.8379927	0.9046022
Classification Tree	0.8272888	0.8189331	0.8903592	0.8546462	0.8933206
Random Forest	0.8398939	0.8302029	0.9130435	0.8716232	0.9015502
SVM	0.8283945	0.8176809	0.9092628	0.8634719	0.8937859
LDA	0.8549314	0.8615076	0.8052930	0.8334003	0.9129512
QDA	0.8533835	0.8898072	0.5784499	0.7341285	0.9146608
KNN (with CV)	0.7720035	0.7743551	0.7542533	0.7643042	0.8571033
Ensemble (w/o naive)	0.8454224	0.8422239	0.8695652	0.8558946	0.9058586

In the models built with R the "positive" class is "no," but we are mainly interested in "yes" for business outcome (customer accepts offer). The data imbalance present in the original data must also be taken into account for evaluation.

Therefore, from a business perspective the most suitable algorithms seem to be Random forest/classification trees and SVM. These models show a high balanced accuracy and also high specificity. High specificity means for this setting that more potential "yes" customers will be targeted.

Conclusion

We explored the Bank marketing data and we tested different approaches for customer response classification. Tree-based models and Support Vector Machines proved the most effective and delivered the best balanced accuracy for the intended results. Nonetheless, such models are also computationally expensive for large amounts of data.

Beside the methods used in this analysis, promising alternatives could be neural networks or time-series methods. These may capture more complex relations between features and also time-related effects.

However, in any real-world scenario the available data and its structure, domain knowledge and the business goals need to be taken into account for model selection and later for best performance during deployment.

References

Fawcett, Tom. 2016. "Learning from Imbalanced Classes." http://www.svds.com/learning-imbalanced-classes.

Irizarry, R. A. 2019. *Introduction to Data Science: Data Analysis and Prediction Algorithms with r*. Chapman & Hall/CRC Data Science Series. CRC Press.

Moro, Sérgio, Paulo Cortez, and Paulo Rita. 2014. "A Data-Driven Approach to Predict the Success of Bank Telemarketing." *Decision Support Systems* 62: 22–31.