Bank Campaign Classification - Advanced R Group Project

Group 1- Louis Dubaere, Hatem Hassan, Federico Loguercio, Alberto Lombatti, Gerald Walravens, Anwita Bhure

6/6/2019

We are given a dataset that has information about whether a bank's customer responds positively to a marketing campaign and subscribes to a term deposit. The data has multiple variables of the customer, which can broadly be divided into three segments - Demographics, Transactional and Campaign-specific.

Data Exploration

We begin by undertaking basic quality and cosistency checks on the data.

Check if there are any missing values in the training set and the submission set

```
any(is.na(train))
```

```
[1] FALSE
```

```
any(is.na(submission))
```

```
[1] FALSE
```

As both return FALSE, there are no missing values

Check for the structure of the dataset

```
glimpse(train)
```

```
Observations: 36,168
Variables: 17
$ age       <int> 50, 47, 56, 36, 41, 32, 26, 60, 39, 55, 32, 30, 35, 53…
$ job       <fct> entrepreneur, technician, housemaid, blue-collar, mana…
$ marital   <fct> married, married, married, married, married, single, s…
$ education <fct> primary, secondary, primary, primary, primary, tertiar…
$ default   <fct> yes, no, no, no, no, no, no, no, no, no, no, no, no, n…
$ balance   <int> 537, -938, 605, 4608, 362, 0, 782, 193, 2140, 873, 0, …
$ housing   <fct> yes, yes, no, yes, yes, no, no, yes, yes, yes, no, yes…
$ loan      <fct> no, no, no, no, no, no, no, no, no, yes, no, no, no, n…
$ contact   <fct> unknown, unknown, cellular, cellular, cellular, cellul…
$ day       <int> 20, 28, 19, 14, 12, 4, 29, 12, 16, 3, 19, 27, 21, 8, 1…
$ month     <fct> jun, may, aug, may, may, feb, jan, may, apr, jun, aug,…
$ duration  <int> 11, 176, 207, 284, 217, 233, 297, 89, 539, 131, 103, 1…
$ campaign  <int> 15, 2, 6, 7, 3, 3, 1, 2, 1, 1, 4, 3, 1, 2, 1, 8, 1, 2,…
$ pdays     <int> -1, -1, -1, -1, -1, 276, -1, -1, -1, -1, -1, -1, -1, -…
$ previous  <int> 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ poutcome  <fct> unknown, unknown, unknown, unknown, unknown, failure, …
$ y         <fct> no, no, no, no, no, yes, no, no, no, no, no, no, no, n…
```

```
glimpse(submission)
```

```
Observations: 9,043
Variables: 16
$ age       <int> 58, 43, 51, 56, 32, 54, 58, 54, 32, 38, 57, 51, 35, 57…
$ job       <fct> management, technician, retired, management, blue-coll…
$ marital   <fct> married, single, married, married, single, married, ma…
$ education <fct> tertiary, secondary, primary, tertiary, primary, secon…
$ default   <fct> no, no, no, no, no, no, no, no, no, no, no, no, no, no…
$ balance   <int> 2143, 593, 229, 779, 23, 529, -364, 1291, 0, 424, 249,…
$ housing   <fct> yes, yes, yes, yes, yes, yes, yes, yes, yes, yes, yes,…
$ loan      <fct> no, no, no, no, yes, no, no, no, no, no, no, no, yes, …
$ contact   <fct> unknown, unknown, unknown, unknown, unknown, unknown, …
$ day       <int> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, …
$ month     <fct> may, may, may, may, may, may, may, may, may, may, may,…
$ duration  <int> 261, 55, 353, 164, 160, 1492, 355, 266, 179, 104, 164,…
$ campaign  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
$ pdays     <int> -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1…
$ previous  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
$ poutcome  <fct> unknown, unknown, unknown, unknown, unknown, unknown, …
```

```
unique(train$y)
```

```
[1] no  yes
Levels: no yes
```

The target variable is recorded as y, a factor with two levels either yes or no. It is therefore a binary classification problem.

Before moving to modeling, we check the proportion of yes and no in the data.

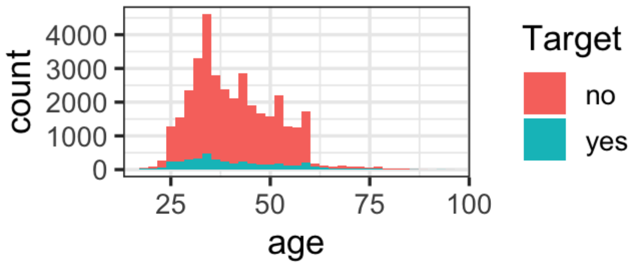
Check for imbalance

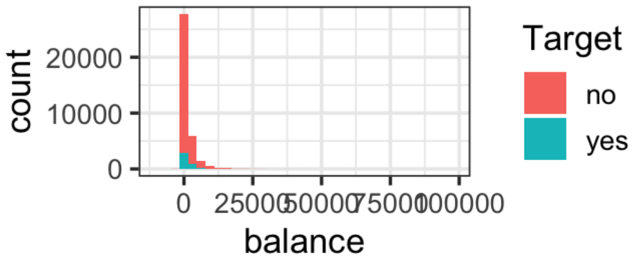
```
nrow(subset(train, y =='yes')) / nrow(train)
```

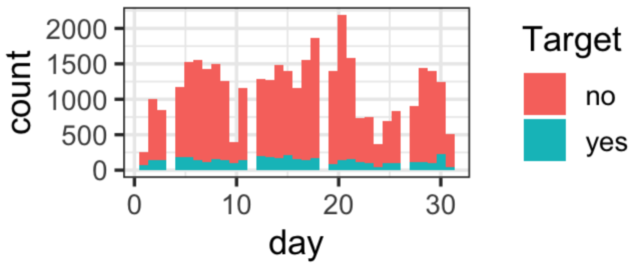
```
[1] 0.1159865
```

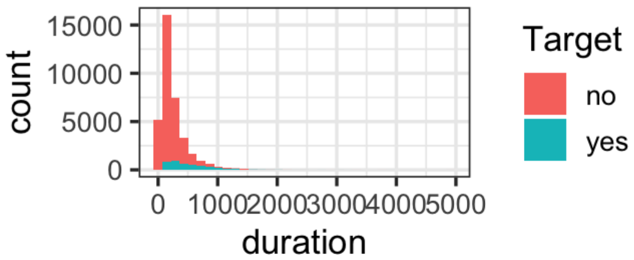
Proportion of nos in the dataset is very low, ~11%. We may need to do resampling in order to get the right models.

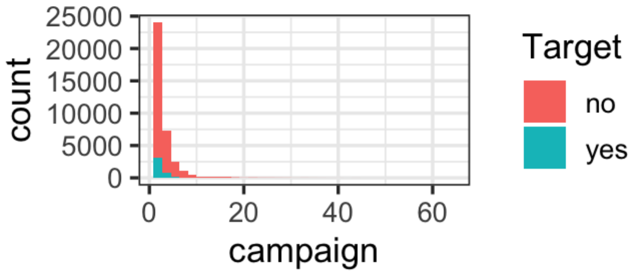
Numerical Features

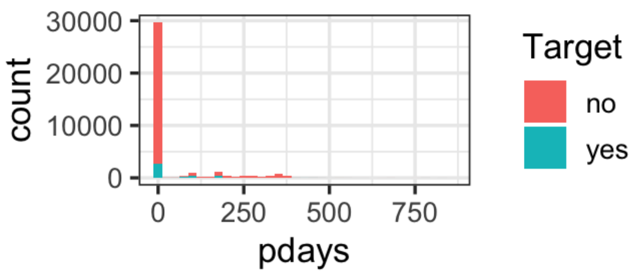


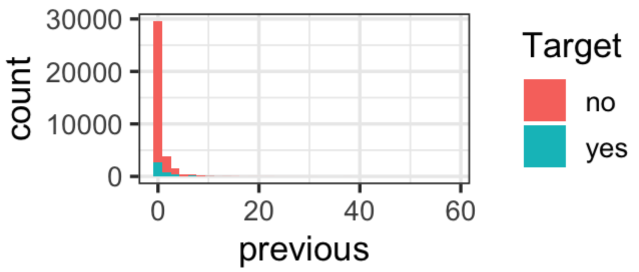








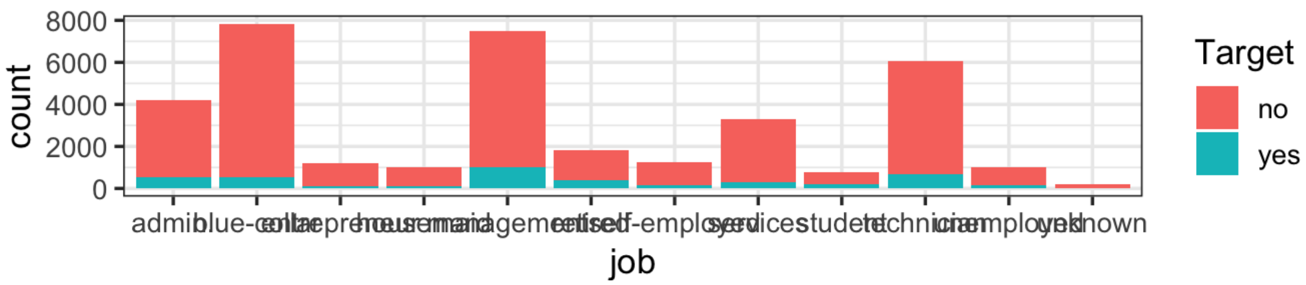


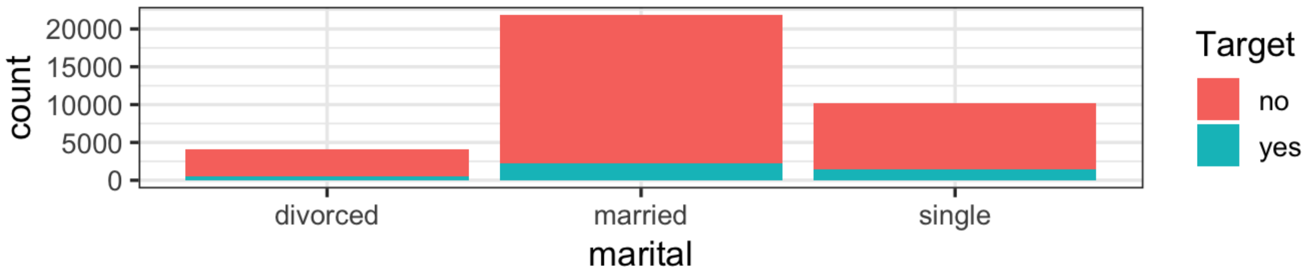


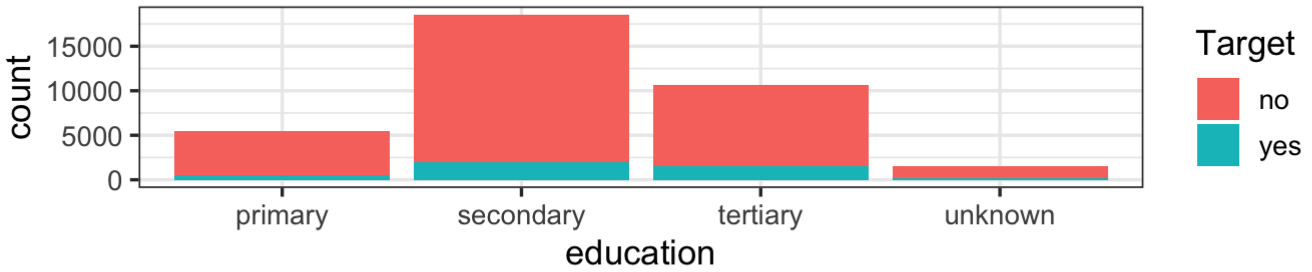
As can be seen from the histograms, most of the numerical features are heavily skewed, and need to be normalized

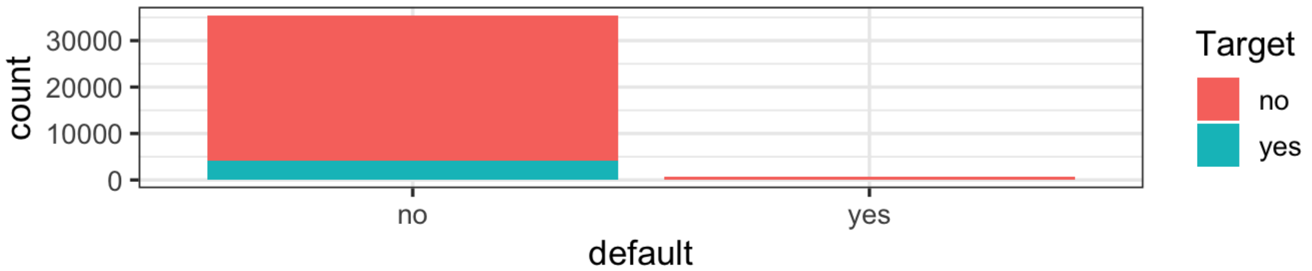
---

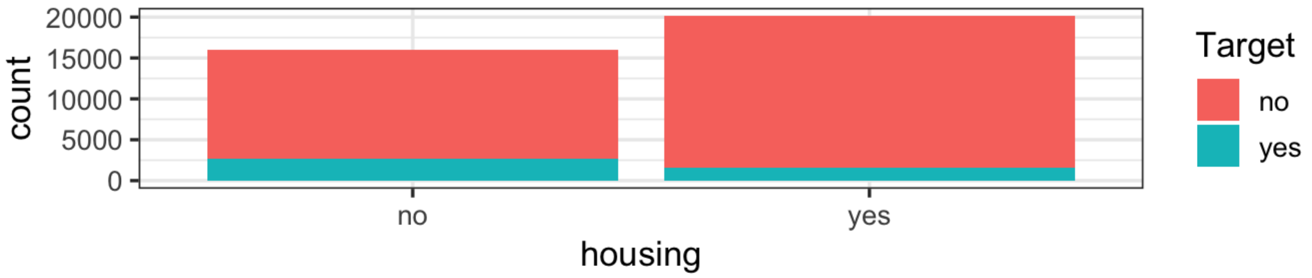
Categorical Features

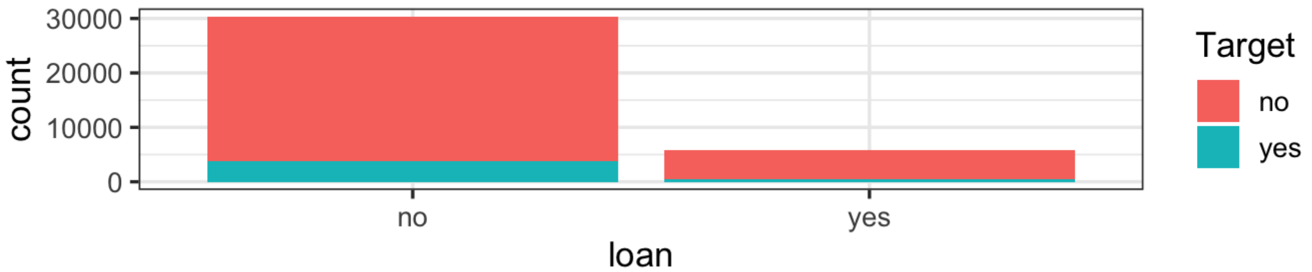


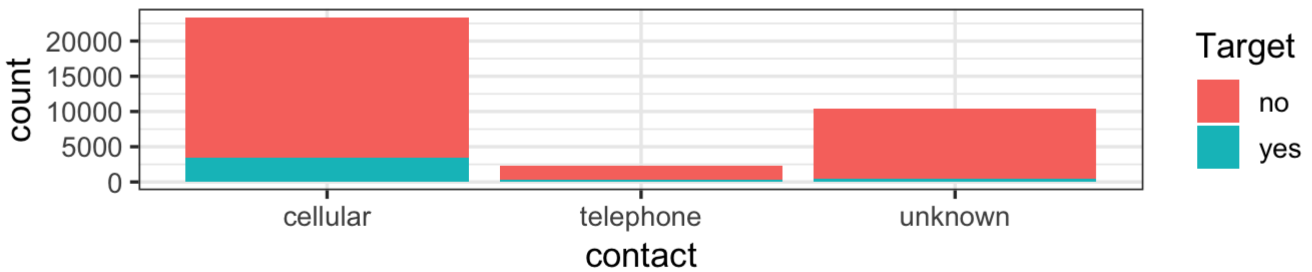

count
20000
15000
10000
5000
0
divorced
married
single
marital
Target
no
yes

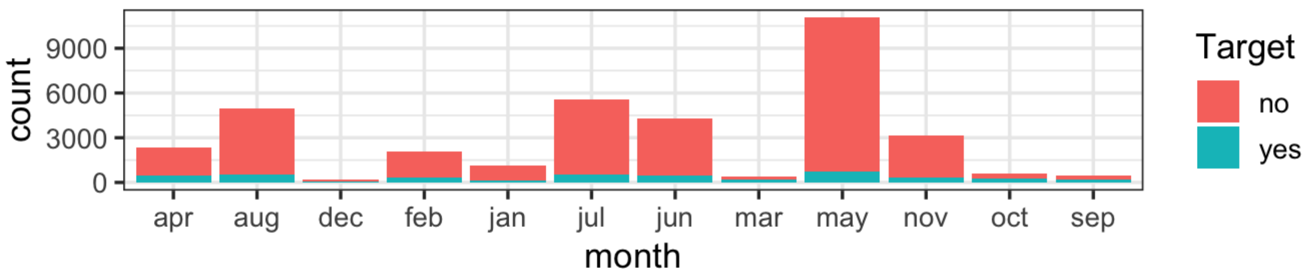

count
15000
10000
5000
0
primary
secondary
tertiary
unknown
education
Target
no
yes

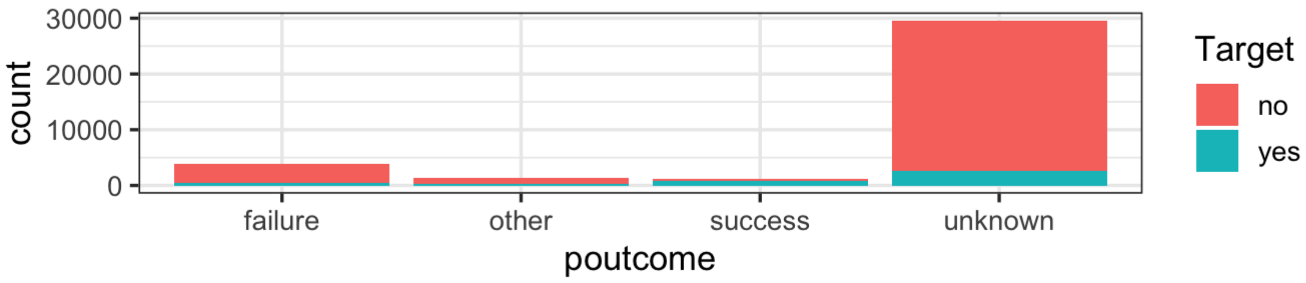

count
30000
20000
10000
0
no
yes
default
Target
no
yes


count
20000
15000
10000
5000
0
no
yes
housing
Target
no
yes


count
30000
20000
10000
0
no
yes
loan
Target
no
yes

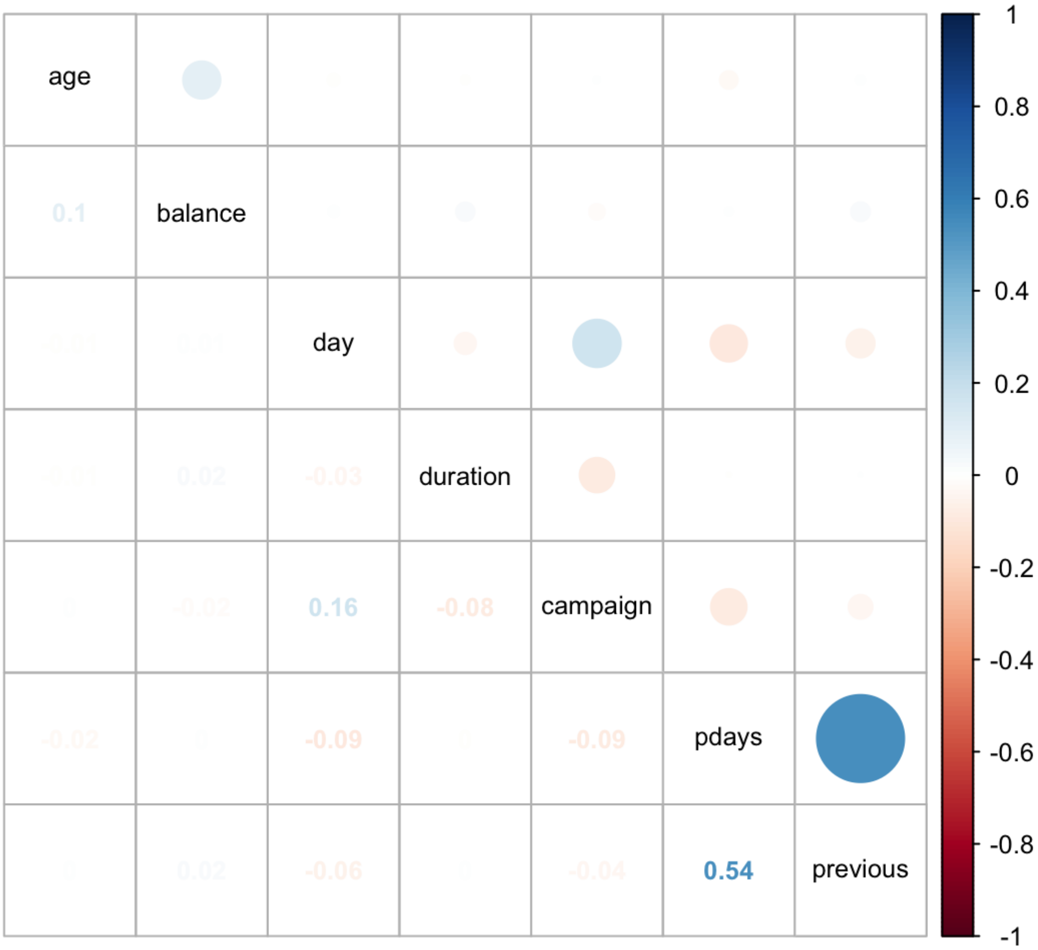

count
20000
15000
10000
5000
0
cellular
telephone
unknown
contact
Target
no
yes


count
9000
6000
3000
0
apr
aug
dec
feb
jan
jul
jun
mar
may
nov
oct
sep
month
Target
no
yes



Correlation between pairs of numerical features

```
train_num <- dplyr::select_if(train, is.numeric)
res <- cor(train_num)
corrplot.mixed(
          res,
          upper="circle",
          lower="number",
          tl.col = "black",
          number.cex = .8,
          tl.cex=.8)
```

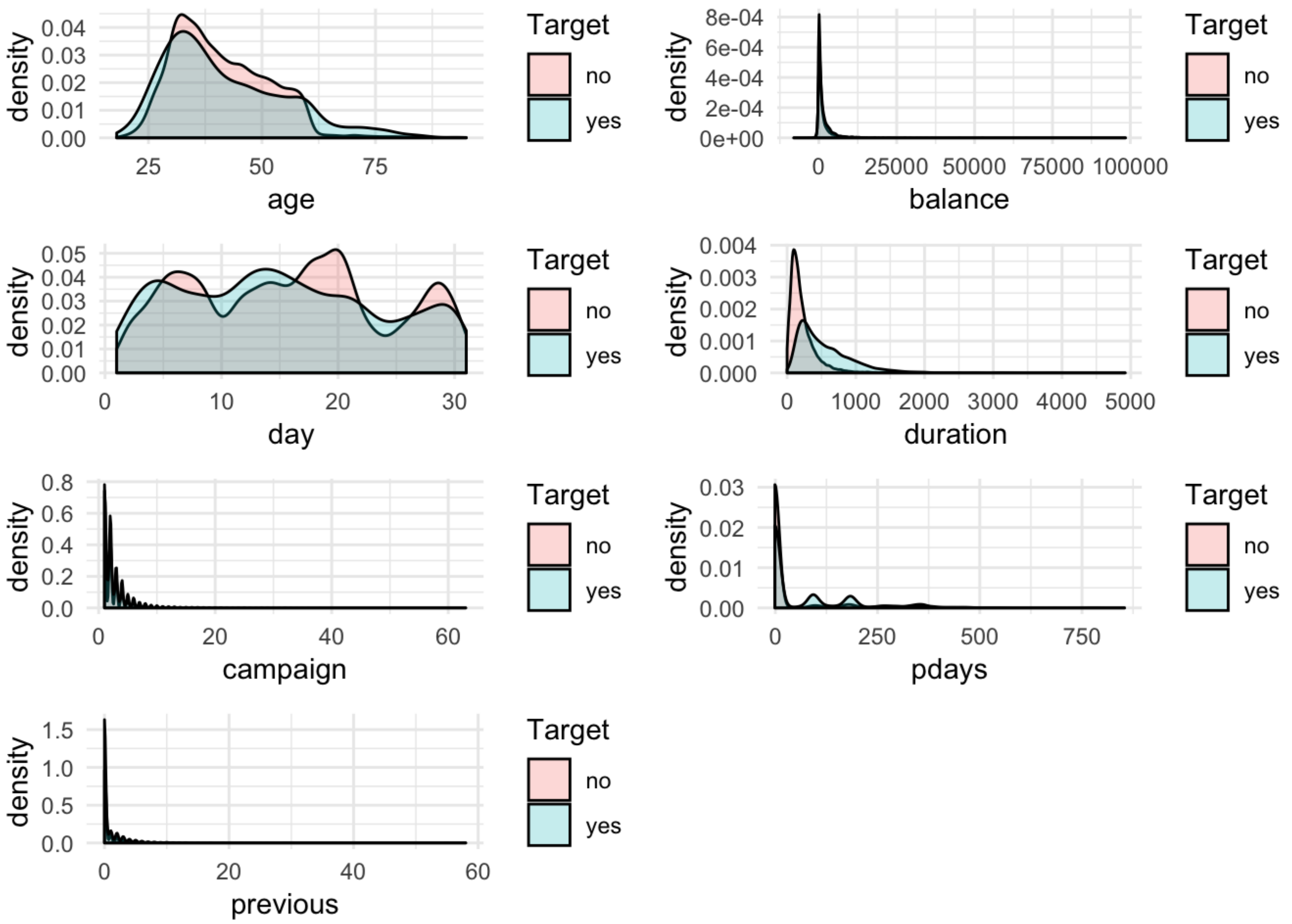


As shown from the result, there are no strong correlations between most of the pairs

Except the one for pdays and previous = 0.54

Density Plots

```
numeric_vars <- unlist(lapply(train, is.numeric))
histograms = list()
for (i in colnames(train[numeric_vars])){
  histograms[[i]] <-
    ggplot(
      train[numeric_vars],
      aes_string(x=i,fill=train$y)) +
      geom_density(alpha = 0.25) +
      theme_minimal() +
      labs(fill = "Target")
}
plot_grid(plotlist = histograms, ncol = 2)
```



Apart from duration, none of the numerical variables clearly separates the target classes

---

Preprocessing

```
# encode target variable as 1 and 0 and convert it to factor
train$label <- as.factor(ifelse(train$y == 'yes', 1, 0))
train$y <- NULL

# split train data further into train and test (hold out)
test_proportion <- 0.2
set.seed(7)
train_index <-sample(nrow(train), floor(nrow(train)*(1-test_proportion)), replace = F
ALSE)

df_train <- train[train_index,]
df_test <- train[-train_index,]
```

Baseline - Binary Logistic

```
# Define custom summary function so we can get recall as output always

recallSummary <- function (data,
    lev = NULL,
    model = NULL) {
    c(Accuracy = MLmetrics::Accuracy(data$pred, data$obs),
      recall = MLmetrics::Recall(data$obs, data$pred, positive = 1))
}

ctrl_base <- trainControl(
  method = "cv", # k-fold cross val
  number = 5,
  savePredictions=TRUE,
  summaryFunction = recallSummary
)

bsline <- train(label~.,
             data = df_train,
             method = 'glm',
             family = "binomial",
             trControl = ctrl_base)

bsline
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23148, 23146, 23148, 23147
Resampling results:

  Accuracy   recall
  0.9034008  0.354236
```

The model performs poorly in terms of recall, probably due to the unbalanced target variable. This can be tackled by either undersampling the prevalent class or oversampling the rare class or both.

Baseline with Resampling

The target variable is highly unbalanced. We will therefore resample it in order try to make the model better capture the underrepresented class.

Resampling is defined through the control object instead of outside, such that cross-validation performance will be evaluated on data that has not been resampled. Failing to do so would artifially inflate cv-recall, with a model that would perform far worse on an unbalanced test set.

```
# define basic train control object
ctrl <- trainControl(method = "cv",
                     number = 5,
                     savePredictions=TRUE,
                     summaryFunction = recallSummary,
                     sampling = "down")
```

Undersampling

```
set.seed(7)
log_under <- train(label~. ,
                   data = df_train,
                   method = 'glm',
                   family = "binomial",
                   trControl = ctrl)

log_under
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using down-sampling

Resampling results:

  Accuracy   recall
  0.8439209  0.8291449
```

This is a game changer. By simply undersampling, we are able to highly improve recall.

Let’s try oversampling instead:

Oversampling

```
ctrl$sampling <- "up"

set.seed(7)
log_up <- train(label~.,
                data = df_train,
                method = 'glm',
                family = "binomial",
                trControl = ctrl)

log_up
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using up-sampling

Resampling results:

  Accuracy   recall
  0.8460982  0.8228726
```

Next: “smote” for sampling

SMOTE

```
ctrl$sampling <- "smote"

set.seed(7)
log_smt <- train(label~.,
                 data = df_train,
                 method = 'glm',
                 family = "binomial",
                 trControl = ctrl)

log_smt
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using SMOTE

Resampling results:

  Accuracy   recall
  0.8681483  0.7631346
```

Smote resampling takes accuracy up but brings down recall substantially. SMOTE stands for Synthetic Minority Oversampling Tecniques. It uses nearest neighbours algorithm to generate new and synthetic data. 1 (https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18)

Summary of all resampling results

```
resample_models <- list(baseline = bsline,
                        undersampling = log_under,
                        oversampling = log_up,
                        SMOTE = log_smt
                        )

# do a resampling on the models and generate distribution of accuracy and recall metr
ics
resampling <- resamples(resample_models)
summ1 <- summary(resampling)
summ1
```

```
Call:
summary.resamples(object = resampling)

Models: baseline, undersampling, oversampling, SMOTE
Number of resamples: 5

Accuracy
                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
baseline      0.8999482 0.9030418 0.9040954 0.9034008 0.9042516 0.9056669
undersampling 0.8351477 0.8422326 0.8434422 0.8439209 0.8481078 0.8506740
oversampling  0.8424054 0.8429238 0.8451702 0.8460982 0.8491446 0.8508469
SMOTE         0.8622775 0.8655607 0.8667703 0.8681483 0.8719323 0.8742008
              NA's
baseline         0
undersampling    0
oversampling     0
SMOTE            0

recall
                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
baseline      0.3402985 0.3408072 0.3497758 0.3542360 0.3656716 0.3746269
undersampling 0.7982063 0.8313433 0.8328358 0.8291449 0.8370703 0.8462687
oversampling  0.7907324 0.8238806 0.8298507 0.8228726 0.8325859 0.8373134
SMOTE         0.7309417 0.7537313 0.7698057 0.7631346 0.7701493 0.7910448
              NA's
baseline         0
undersampling    0
oversampling     0
SMOTE            0
```

```r
# or just print the train results
summ2 <- NULL
for (i in 1:length(resample_models)) {
  res <- resample_models[[i]]$results[1:3]
  summ2 <- rbind(summ2, res)
}
summ2$parameter <- names(resample_models)
datatable(summ2)
```

Show 10 entries Search:

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957

Showing 1 to 4 of 4 entries Previous 1 Next

Variable Transformations

Now that we have established that resampling improves model results significantly, we will keep it as the base and move to feature selection and feature engineering. We use undersampling method.

Day as a categorical variable

There is no numerical relevance to day of the month. Hence, it is converted as a factor.

```
train$day <- as.factor(train$day)
```

Age as a categorical variable

Proceed by grouping together age-ranges

```
# create a new preprocessed dataset
train_preproc <- train

# bin age variable
train_preproc$age_bin <- cut(train_preproc$age, breaks = c(-Inf, 20, 40, 60, Inf),
                             labels = c('<20', "20-40", "40-60", ">60"))
train_preproc$age <- NULL
```

```
df_train_preproc <- train_preproc[train_index,]
df_test_preproc <- train_preproc[-train_index,]
```

```
ctrl$sampling <- "down"

set.seed(7)
log_agebin <- train(label~.,
                    data = df_train_preproc,
                    method = 'glm',
                    family = "binomial",
                    trControl = ctrl)

log_agebin
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using down-sampling

Resampling results:

  Accuracy   recall
  0.8464784  0.8303442
```

```
# update pipeline
summ2 <- rbind(summ2, log_agebin$results[1:3])
summ2$parameter[nrow(summ2)] <- "age binned"
datatable(summ2)
```

Show 10 entries Search:

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006

Showing 1 to 5 of 5 entries Previous 1 Next

The results after binning age improve slightly. We now move on to the next variable, jobs.

Reducing the number of categories in job

```
levels(df_train_preproc$job)
```

```
 [1] "admin."        "blue-collar"   "entrepreneur"  "housemaid"
 [5] "management"    "retired"       "self-employed" "services"
 [9] "student"       "technician"    "unemployed"    "unknown"
```

Job currently has 12 different levels, but the proportion of observations in each of them is not uniform. We try merging some levels together to see if it has any impact on the model performance. We realise that reducing the levels will result in loss of granularity but it is assumed that strategies are not going to be planned at such minute levels and a general idea of customer's occupation would be enough.

```
train_preproc$job_binned <- car::recode(train_preproc$job, "c('retired', 'student') =
'rs';
                                        c('blue-collar', 'entrepreneur')= 'be';
                                        c('housemaid', 'services')= 'hs';
                                        c('admin.', 'unknown')= 'au';
                        c('management', 'self-employed','technician', 'unemployed') =
'others'")
train_preproc$job_binned <- factor(train_preproc$job_binned)

train_preproc$job <- NULL
```

```
df_train_preproc <- train_preproc[train_index,]
df_test_preproc <- train_preproc[-train_index,]
```

```
set.seed(7)
log_agejobbin <- train(label~.,
                    data = df_train_preproc,
                    method = 'glm',
                    family = "binomial",
                    trControl = ctrl)

log_agejobbin
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using down-sampling

Resampling results:

  Accuracy   recall
  0.8459601  0.8267599
```

```
# update pipeline
summ2 <- rbind(summ2, log_agejobbin$results[1:3])
summ2$parameter[nrow(summ2)] <- "age & job binned"
datatable(summ2)
```

Show 10 **entries** **Search:**

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288

Showing 1 to 6 of 6 entries Previous 1 Next

The model recall decreases marginally. However, we can still go ahead with these two new features.

Converting 'balance' to categorical

This variable is highly skewed with negative values and majority 0. There are 3052 negative values and 2799 0s. We group the variable into ranges in order to cope with this.

```
train_preproc$balance_bin <- cut(train_preproc$balance,
                                 breaks = c(-Inf, 0, 500, 1000, 5000, Inf),
                                 labels = c("<=0", "1-500", "501-1000", "1001-5000",
">5000"))


train_preproc$balance <- NULL
```

```
df_train_preproc <- train_preproc[train_index,]
df_test_preproc <- train_preproc[-train_index,]
```

```
set.seed(7)
log_bal <- train(label~.,
                 data = df_train_preproc,
                 method = 'glm',
                 family = "binomial",
                 trControl = ctrl)

log_bal
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using down-sampling

Resampling results:

  Accuracy   recall
  0.8467204  0.8279535
```

```
# update pipeline
summ2 <- rbind(summ2, log_bal$results[1:3])
summ2$parameter[nrow(summ2)] <- "balance binned"
datatable(summ2)
```

Show 10 entries Search:

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957

	parameter	Accuracy	recall
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538

Showing 1 to 7 of 7 entries

Previous 1 Next

Binning balance improves accuracy and recall.

Modify Day Variable

The day variable is categorical not, but that means that it has 31 different levels. It can be imagined that certain days of the month have a slight effect on the target but most do not hold any explanatory power.

It is reasonable to argue that the first and the last day of the month are somewhat special. We group them together and put the rest into one category.

```
train_preproc$day_binned <- car::recode(train_preproc$day, "c(1,30) = 'special';
                        c(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
                        22,23,24,25,26,27,28,29,31) = 'others'")
train_preproc$day_binned <- factor(train_preproc$day_binned)

train_preproc$day <- NULL

df_train_preproc <- train_preproc[train_index,]
df_test_preproc <- train_preproc[-train_index,]
```

```
log_day_binned <- train(label~.,
                 data = df_train_preproc,
                 method = 'glm',
                 family = "binomial",
                 trControl = ctrl)

log_day_binned
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23146, 23148, 23147, 23148
Addtional sampling using down-sampling

Resampling results:

  Accuracy   recall
  0.8464442  0.8249729
```

```
summ2 <- rbind(summ2, log_day_binned$results[1:3])
summ2$parameter[nrow(summ2)] <- "Day binned"
datatable(summ2)
```

Show 10 entries Search:

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631

Showing 1 to 8 of 8 entries Previous 1 Next

Model performance goes down but we are significantly reducing the number of variables which will shorten runtimes.

pday binned

```
train_preproc$pdays_binned <- cut(train_preproc$pdays, breaks = c(-Inf, 0, 99, 299, 4
99, Inf ),
                                  labels = c("notcontacted", "0-100", "101-300", "301
-500", ">500"))
train_preproc$pdays <- NULL

df_train_preproc <- train_preproc[train_index,]
df_test_preproc <- train_preproc[-train_index,]
```

```
log_pday_binned <- train(label~.,
                 data = df_train_preproc,
                 method = 'glm',
                 family = "binomial",
                 trControl = ctrl)

log_pday_binned
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23146, 23148
Addtional sampling using down-sampling

Resampling results:

  Accuracy   recall
  0.8474805  0.8243728
```

```
summ2 <- rbind(summ2, log_pday_binned$results[1:3])
summ2$parameter[nrow(summ2)] <- "PDays binned"
datatable(summ2)
```

Show 10 entries Search:

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631
9	PDays binned	0.847480515652526	0.824372755058787

Showing 1 to 9 of 9 entries Previous 1 Next

Redesigning how numeric variables are modelled

We now fit the model again, however, adding in Yeo-Johnson through preprocessing to fix skewness in all numeric variables. No other preprocessing is done.

The Yeo-Johnson transformation is a variant of the BoxCox Transformation suited for variables with non-positive values. Variables are transformed to a more normal-like distribution with more stable variance.

```
ctrl$sampling <- "down"

log_yj <- train(label~.,
                data = df_train,
                method = 'glm',
                family = "binomial",
                preProcess = "YeoJohnson",
                trControl = ctrl)

log_yj
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

Pre-processing: Yeo-Johnson transformation (42)
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23148, 23146, 23147, 23148, 23147
Addtional sampling using down-sampling prior to pre-processing

Resampling results:

  Accuracy   recall
  0.8450953  0.8228856
```

The results remain mostly unchanged.

```
# update pipeline
summ2 <- rbind(summ2, log_yj$results[1:3])
summ2$parameter[nrow(summ2)] <- "Yeo-Johnson"
datatable(summ2)
```

Show 10 **entries** **Search:**

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631

	parameter	Accuracy	recall
9	PDays binned	0.847480515652526	0.824372755058787
10	Yeo-Johnson	0.845095292962903	0.822885572139303

Showing 1 to 10 of 10 entries Previous 1 Next

The Yeo-Johnson transformation does not appear to help with fixing skewness and outlier issues in certain variables.

Logarithm

Let's try taking the logarithm instead

```
# Logarithmise skewed variables
to_log <- function(df, l_varlist){
  for(l_var in l_varlist){
    df[,l_var] <- log(df[,l_var])
  }
  return(df)
}

# Select the numeric variables that are skewed
log_varlist <- list('duration', 'campaign', 'previous')

train_log <- train_preproc

# Add constant in order to avoid non-positive values in log (monotonic transformatio
n)
train_log$duration <- train_log$duration + 1

train_log$previous <- train_log$previous + 1

train_log <- to_log(train_log, log_varlist)

df_train_preproc <- train_log[train_index,]
df_test_preproc <- train_log[-train_index,]
```

```
logarithm <- train(label~.,
                data = df_train_preproc,
                method = 'glm',
                family = "binomial",
                trControl = ctrl)

print(logarithm)
```

```
Generalized Linear Model

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23147, 23147, 23148
Addtional sampling using down-sampling

Resampling results:

  Accuracy   recall
  0.8270202  0.8491618
```

```
summ2 <- rbind(summ2, logarithm$results[1:3])
summ2$parameter[nrow(summ2)] <- "Logarithm"
datatable(summ2)
```

Show 10 **entries** **Search:**

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631
9	PDays binned	0.847480515652526	0.824372755058787
10	Yeo-Johnson	0.845095292962903	0.822885572139303

Showing 1 to 10 of 11 entries Previous 1 2 Next

Parallelize

```
cl <- makePSOCKcluster(4) # I have 4 cores, this can be adapted
registerDoParallel(cl)
# All subsequent models are run in parallel
```

Optimization

Different models & hyperparameter tuning

Random Forest

```
set.seed(7)
# Several grids were explored, adapting the parameters in the direction where perform
ance was improving.
# For faster runtimes, only a small grid is ran.
tuneGrid_ranger <- data.table(expand.grid(mtry=c(round(sqrt(length(df_train_preproc))
+ 2),
                                                  round(1/2 * (length(df_train_prepro
c))),
                                                  round(2/3 * (length(df_train_prepro
c)))
                                                  ), #standard choices for mtry
                              splitrule= 'gini',
                              min.node.size=c(200, 300, 500))) # We found that very l
arge values for min.node.size lead to the highest recall. Lower values optimize
# accuracy at the cost of recall.

ranger <- train(label~.,
                data = df_train_preproc,
                method = "ranger", num.trees=500, # increasing the number of trees, t
he model did not improve further
                tuneGrid = tuneGrid_ranger,
                metric = "recall",
                importance = "permutation",
                trControl = ctrl)

ranger
```

```
Random Forest

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using down-sampling

Resampling results across tuning parameters:

  mtry  min.node.size  Accuracy   recall
   6    200            0.8096357  0.8891881
   6    300            0.8052464  0.8876911
   6    500            0.8003044  0.8799215
   8    200            0.8076660  0.8918743
   8    300            0.8085990  0.8847038
   8    500            0.8024818  0.8772345
  11    200            0.8104998  0.8966535
  11    300            0.8106728  0.8873931
  11    500            0.8020669  0.8841064

Tuning parameter 'splitrule' was held constant at a value of gini
recall was used to select the optimal model using the largest value.
The final values used for the model were mtry = 11, splitrule = gini
 and min.node.size = 200.
```

```
o1 <- data.frame(ranger$results[7,3:5])
names(o1) <- c("parameter", "Accuracy", "recall")
summ2 <- rbind(summ2, o1)
summ2$parameter[nrow(summ2)] <- "Random Forest"
datatable(summ2)
```

Show 10 entries Search:

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631

	parameter	Accuracy	recall
9	PDays binned	0.847480515652526	0.824372755058787
10	Yeo-Johnson	0.845095292962903	0.822885572139303

Showing 1 to 10 of 12 entries Previous 1 2 Next

Calculate test set performance

```
pred_ranger <- predict(ranger, newdata = df_test_preproc)
Accuracy(pred_ranger, df_test_preproc$label)
```

```
[1] 0.8164224
```

```
Recall(df_test_preproc$label, pred_ranger, positive = 1)
```

```
[1] 0.8902007
```

Boosting trees

Deepboost is a crazy variant of boosting. It is optimized to be able to grow ensembles of very deep (very large) trees without overfitting like AdaBoost will when growing very deep trees. We grow very shallow trees with it and only select it because it allows to choose between logistic and exponential for the loss function. Logistic loss never assigns zero penalty to any points, making it more sensitive to outliers. Exponential loss penalizes incorrect corrections more strongly.

```
# Again here, a far more exhaustive grid was explored iteratively.
tunegrid_deepbost <- expand.grid(num_iter = c(100), #more and larger trees improved a
ccuracy at the cost of recall
                                 tree_depth = c(3),
                                 beta = c(0.1),
                                 lambda = 0.1,
                                 loss_type = c("e", "l"))

boosting <- train(label~.,
                data = df_train_preproc,
                method = "deepboost",
                tuneGrid = tunegrid_deepbost,
                trControl = ctrl,
                verbose = FALSE)

boosting
```

```
DeepBoost

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23146, 23147, 23147, 23148, 23148
Addtional sampling using down-sampling

Resampling results across tuning parameters:

  loss_type  Accuracy   recall
  e          0.8260515  0.8667889
  l          0.8050730  0.8330330

Tuning parameter 'num_iter' was held constant at a value of 100

Tuning parameter 'beta' was held constant at a value of 0.1

Tuning parameter 'lambda' was held constant at a value of 0.1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were num_iter = 100, tree_depth =
 3, beta = 0.1, lambda = 0.1 and loss_type = e.
```

```
o2 <- data.frame(boosting$results[1,5:7])
names(o2) <- c("parameter", "Accuracy", "recall")
summ2 <- rbind(summ2, o2)
summ2$parameter[nrow(summ2)] <- "Boosting"
datatable(summ2)
```

Show 10 **entries** **Search:**

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631
9	PDays binned	0.847480515652526	0.824372755058787
10	Yeo-Johnson	0.845095292962903	0.822885572139303

Showing 1 to 10 of 13 entries

Previous 1 2 Next

Deepboost is not able to reach the levels of recall obtained with the random forest.

XGBoost

```
set.seed(7)

tunegrid_xgb <- expand.grid(nrounds = c(100, 80),
                       max_depth = c(5),
                       colsample_bytree = c(0.7),
                       eta = c(0.05),
                       gamma = c(0.1, 0.2),
                       min_child_weight = c(1),
                       subsample = 1
                      )

xgb <- train(label~.,
                data = df_train_preproc,
                method = "xgbTree",
                tuneGrid = tunegrid_xgb,
             metric = "recall",
                trControl = ctrl)

xgb
```

```
eXtreme Gradient Boosting

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using down-sampling

Resampling results across tuning parameters:

  gamma  nrounds  Accuracy   recall
  0.1     80      0.8282645  0.8754479
  0.1    100      0.8303036  0.8787355
  0.2     80      0.8288176  0.8757410
  0.2    100      0.8324811  0.8793271

Tuning parameter 'max_depth' was held constant at a value of 5
 0.7
Tuning parameter 'min_child_weight' was held constant at a value of
 1
Tuning parameter 'subsample' was held constant at a value of 1
recall was used to select the optimal model using the largest value.
The final values used for the model were nrounds = 100, max_depth = 5,
 eta = 0.05, gamma = 0.2, colsample_bytree = 0.7, min_child_weight = 1
 and subsample = 1.
```

```
o3 <- data.frame(xgb$results[4,7:9])
names(o3) <- c("parameter", "Accuracy", "recall")
summ2 <- rbind(summ2, o3)
summ2$parameter[nrow(summ2)] <- "XGBoost"
datatable(summ2)
```

Show 10 entries Search:

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631
9	PDays binned	0.847480515652526	0.824372755058787
10	Yeo-Johnson	0.845095292962903	0.822885572139303

Showing 1 to 10 of 14 entries Previous 1 2 Next

XGBoost performs quite well but still worse than random forest.

```
mypred_xgb <- predict(xgb, newdata = df_test_preproc)
Accuracy(mypred_xgb, df_test_preproc$label)
```

```
[1] 0.8249931
```

```
Recall(df_test_preproc$label, mypred_xgb, positive = 1)
```

```
[1] 0.8713105
```

Logistic Regression with regularization

Since we have seen that recall seems to be highest the less granularly the model fits the data, we will try whether a regularized logistic regression performs well.

```
tuneGrid_reg = expand.grid(alpha = c(0, 0.5, 1), #alpha = 0 for Ridge and alpha = 1 f
or lasso
                                             lambda = c(0, 0.2, 0.5)) # higher lambda: more
regularization

log_reg <- train(label~.,
                data = df_train_preproc,
                method = 'glmnet',
                family = "binomial",
                trControl = ctrl,
                tuneGrid = tuneGrid_reg)

print(log_reg)
```

```
glmnet

28934 samples
   16 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 23147, 23147, 23148, 23147, 23147
Addtional sampling using down-sampling

Resampling results across tuning parameters:

  alpha  lambda  Accuracy   recall
  0.0    0.0     0.8257758  0.8536372
  0.0    0.2     0.8220432  0.8318323
  0.0    0.5     0.8168592  0.8016706
  0.5    0.0     0.8278149  0.8488557
  0.5    0.2     0.7543025  0.7879263
  0.5    0.5     0.7164233  0.7305843
  1.0    0.0     0.8290936  0.8488588
  1.0    0.2     0.7193956  0.7261053
  1.0    0.5     0.4232205  0.6000000

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were alpha = 1 and lambda = 0.
```

```
o4 <- data.frame(log_reg$results[1,2:4])
names(o4) <- c("parameter", "Accuracy", "recall")
summ2 <- rbind(summ2, o4)
summ2$parameter[nrow(summ2)] <- "Logistic with Regularizations"
datatable(summ2)
```

Show 10 **entries** **Search:**

	parameter	Accuracy	recall
1	baseline	0.903400781979263	0.354235994913326
2	undersampling	0.843920880388484	0.82914485866631

	parameter	Accuracy	recall
3	oversampling	0.846098180296242	0.822872632353925
4	SMOTE	0.868148330127882	0.763134551457957
5	age binned	0.846478444271584	0.830344242911006
6	age & job binned	0.845960052899956	0.826759922361288
7	balance binned	0.846720407631418	0.827953506012538
8	Day binned	0.846444168850983	0.824972893380631
9	PDays binned	0.847480515652526	0.824372755058787
10	Yeo-Johnson	0.845095292962903	0.822885572139303

Showing 1 to 10 of 15 entries

Previous 1 2 Next

Regularization significantly worsens performance.

Final Prediction

Make the final prediction on the submission set

```
ranger_finalGrid <- expand.grid(mtry=round(1/2 * (length(df_train_preproc))),
                               splitrule= 'gini',
                               min.node.size=200)
set.seed(7)

ranger_final <- train(label~.,
                data = train_log,
                method = "ranger", num.trees=500,
                tuneGrid = ranger_finalGrid,
                metric = "recall",
                importance = "permutation",
                trControl = ctrl)

final_pred <- predict(ranger_final, newdata = submission)
```