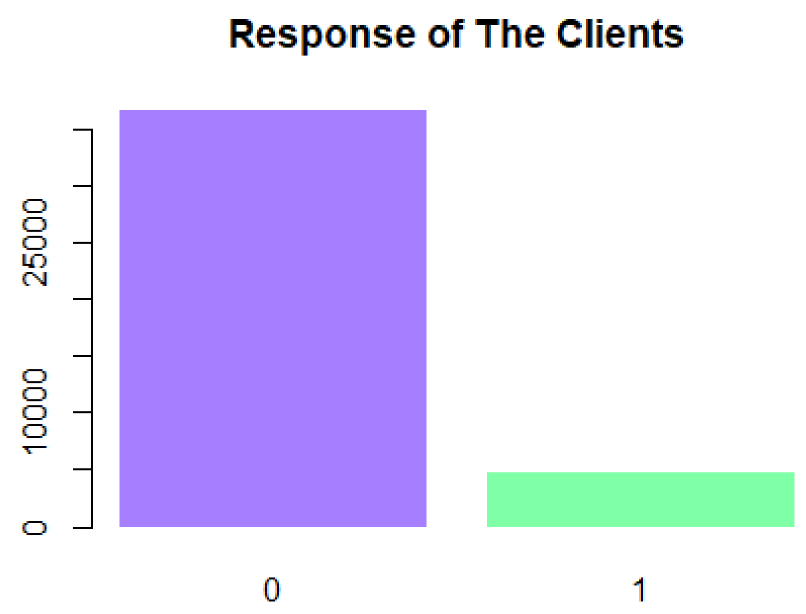
R Notebook

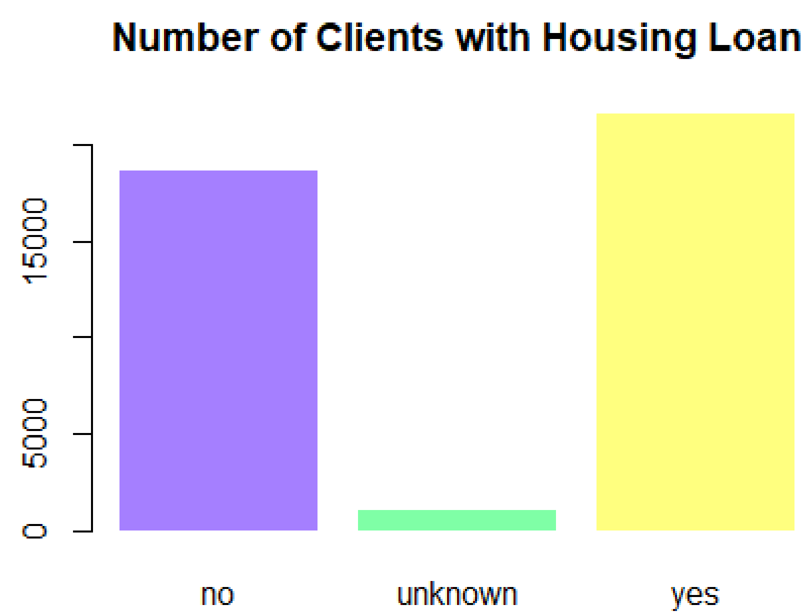
```
# We will use 2 basic models to demonstrate the impact of Synthetic Minority
Oversampling Technique (SMOTE)

# We observed in the Amazon Alexa Reviews dataset that unbalanced dataset can
lead to models with poor predictability for minority class. Here we will use
SMOTE, which creates synthetic minority class examples

# Here we have an unbalanced dataset, which has a number of customer, economy
related data and information about whther customer responded positively to a
marketing call for term deposit (y).

# y = 1 is the minority classs,
# As we will see in the process, we have only 4640 1s in the dataset with
41188 rows
# i.e only about 11.2% of the data has a positive outcome

library(readxl)
df_bank <- read.csv("C:/MBA 2nd Year/R/Data/Portuguese Bank/Portuguese
Bank.csv")
df_bank = data.frame(df_bank)
View(df_bank)

# Let's look at the variables in the dataset
names(df_bank)

##  [1] "age"            "job"            "marital"        "education"
##  [5] "default"        "housing"        "loan"           "contact"
##  [9] "month"          "day_of_week"    "duration"       "campaign"
## [13] "pdays"          "previous"       "poutcome"       "emp_var_rate"
## [17] "cons_price_idx" "cons_conf_idx"  "euribor3m"      "nr_employed"
## [21] "y"

# Variable Description

#age: Client age
#job: Job Type
#marital: Marital status
#education: Education level
#default: Whether the client has defaulted
#housing: Whether the client has housing loan
#loan: Whether the client has a personal loan
#contact: Type of communication
#month: Last month of the contact
#day_of_week: Last contact day of the week
#duration: Duration of last contact (seconds)
#campaign: Number of times client contacted in the campaign
```

```r
#pdays: Number of days since the client was last contacted
#previous: Number of times the client was contacted in earlier campaigns
#poutcome: Outcome of the previous marketinig campaign
#emp.var.rate: Employment variation rate
#cons.price.idx: Consumer price index
#cons.conf.idx: Consumer confidence index
#euribor3m: Euribor 3 month rate
#nr.employed: Number of employees
#y: Whether the client subscribed for a term deposit

table(df_bank$y)

##
##     0     1
## 36548  4640

# The imbalance in the dataset can be observed above
# Low number of 1s can impact the sensitivity of our models

# Rows and Columns in the dataframe
dim(df_bank)

## [1] 41188    21

# Checking the data types of the variables
str(df_bank)

## 'data.frame':    41188 obs. of  21 variables:
##  $ age           : int  44 53 28 39 55 30 37 39 36 27 ...
##  $ job           : Factor w/ 12 levels "admin.","blue-collar",..: 2 10 5 8
6 5 2 2 1 2 ...
##  $ marital       : Factor w/ 4 levels "divorced","married",..: 2 2 3 2 2 1
2 1 2 3 ...
##  $ education     : Factor w/ 8 levels "basic.4y","basic.6y",..: 1 8 7 4 1
1 1 3 7 1 ...
##  $ default       : Factor w/ 3 levels "no","unknown",..: 2 1 1 1 1 1 1 1 1
1 ...
##  $ housing       : Factor w/ 3 levels "no","unknown",..: 3 1 3 1 3 3 3 3 1
3 ...
##  $ loan          : Factor w/ 3 levels "no","unknown",..: 1 1 1 1 1 1 1 1 1
1 ...
##  $ contact       : Factor w/ 2 levels "cellular","telephone": 1 1 1 1 1 1
1 1 1 1 ...
##  $ month         : Factor w/ 10 levels "apr","aug","dec",..: 2 8 5 1 2 4 7
7 5 1 ...
##  $ day_of_week   : Factor w/ 5 levels "fri","mon","thu",..: 3 1 3 1 1 4 3
1 2 3 ...
##  $ duration      : int  210 138 339 185 137 68 204 191 174 191 ...
##  $ campaign      : int  1 1 3 2 1 8 1 1 1 2 ...
##  $ pdays         : int  999 999 6 999 3 999 999 999 3 999 ...
##  $ previous      : int  0 0 2 0 1 0 0 0 1 1 ...
```

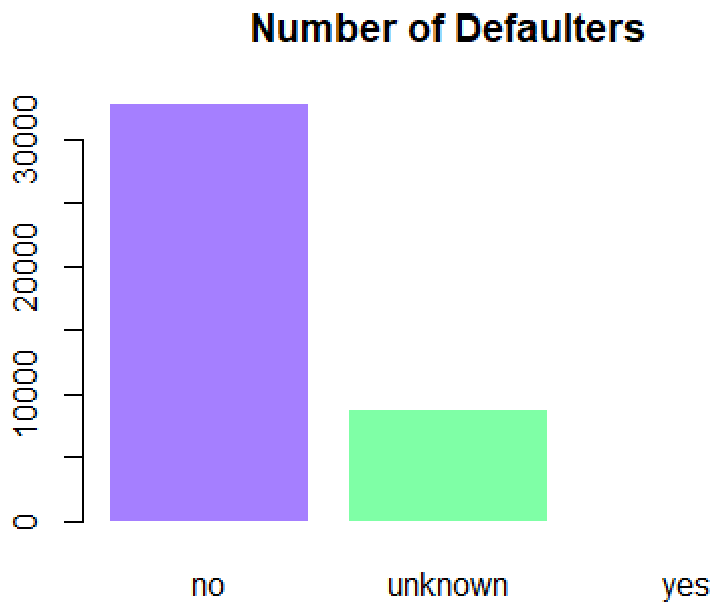
```
##  $ poutcome      : Factor w/ 3 levels "failure","nonexistent",..: 2 2 3 2
3 2 2 2 3 1 ...
##  $ emp_var_rate  : num  1.4 -0.1 -1.7 -1.8 -2.9 1.4 -1.8 -1.8 -2.9 -1.8
...
##  $ cons_price_idx: num  93.4 93.2 94.1 93.1 92.2 ...
##  $ cons_conf_idx : num  -36.1 -42 -39.8 -47.1 -31.4 -42.7 -46.2 -46.2 -
40.8 -47.1 ...
##  $ euribor3m     : num  4.963 4.021 0.729 1.405 0.869 ...
##  $ nr_employed   : num  5228 5196 4992 5099 5076 ...
##  $ y             : int  0 0 1 0 1 0 0 0 1 0 ...

# Checking for missing values
sapply(df_bank, function(x) sum(is.na(x)))

##            age            job        marital      education        default
##              0              0              0              0              0
##        housing           loan        contact          month    day_of_week
##              0              0              0              0              0
##       duration       campaign          pdays       previous       poutcome
##              0              0              0              0              0
##   emp_var_rate cons_price_idx  cons_conf_idx      euribor3m    nr_employed
##              0              0              0              0              0
##              y
##              0

# The dataset has no missing values

# In 'pdays', 999 indicates that the client was never contacted
# Hence, we will replace 999 with 'Not_contacted' and convert the variable
into factor type

df_bank$pdays <- cut(df_bank$pdays, breaks=c(0,5,10,15,20,25,30))
df_bank$pdays <- as.character(df_bank$pdays)
df_bank$pdays[is.na(df_bank$pdays)] <- "Not_contacted"
df_bank$pdays <- as.factor(df_bank$pdays)

# Checking the data type of 'pdays'
str(df_bank$pdays)

##  Factor w/ 7 levels "(0,5]","(10,15]",..: 7 7 6 7 1 7 7 7 1 7 ...

# Exploratory Data Analysis

barplot(table(df_bank$y), main="Response of The Clients", col =
topo.colors(3, alpha=0.5), border = NA)
```



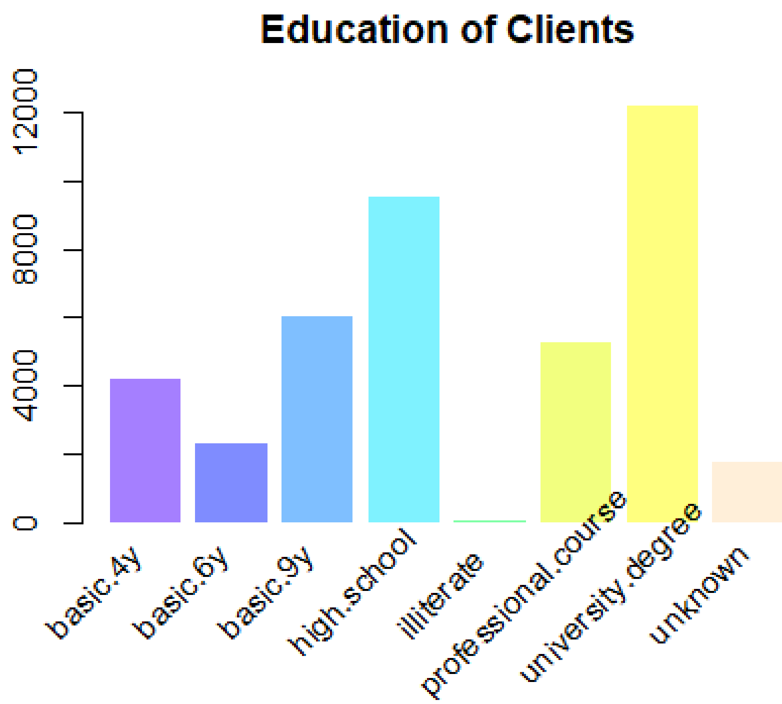

```r
barplot(table(df_bank$housing), main="Number of Clients with Housing Loan",
col = topo.colors(3, alpha=0.5), border = NA)
```



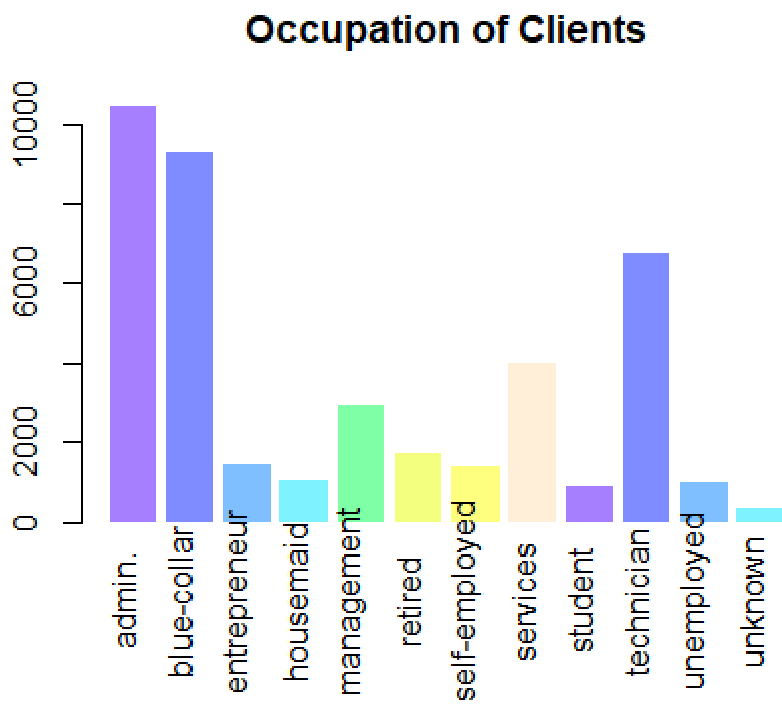
```r
barplot(table(df_bank$default), main="Number of Defaulters", col =
topo.colors(3, alpha=0.5), border = NA)
```



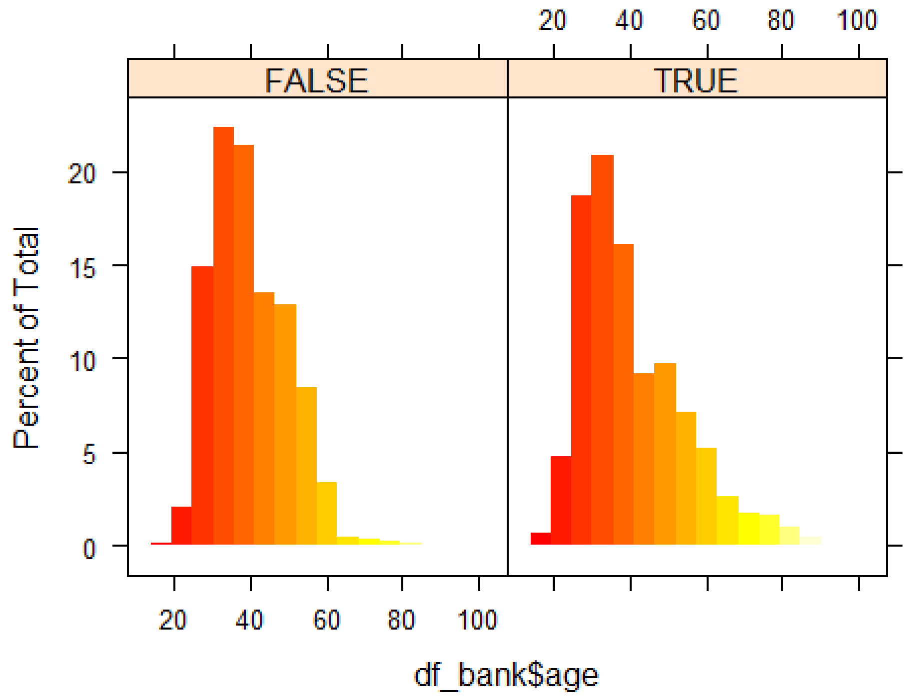

```r
# It appears that the number of defaulters is almost negligible. Let's check.
table(df_bank$default)

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

x <- barplot(table(df_bank$education), main="Education of Clients", col =
topo.colors(8, alpha=0.5), border = NA, xaxt="n")
labs <- paste(names(table(df_bank$education)))
text(cex=1, x=x-.75, y=-2000, labs, xpd=TRUE, srt=45)
```



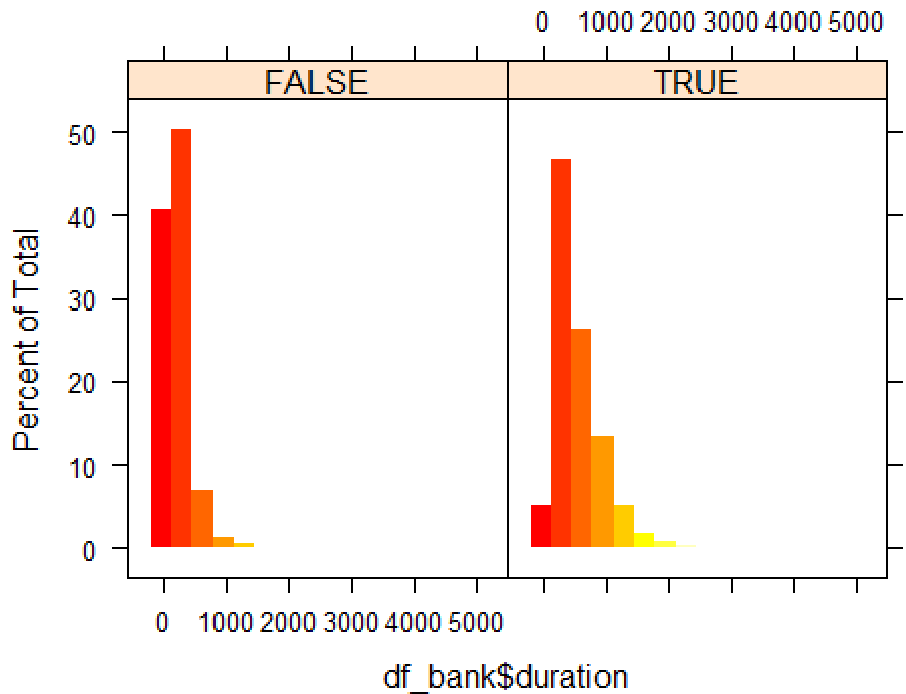

```
x <- barplot(table(df_bank$job), main = "Occupation of Clients", col =
topo.colors(8, alpha=0.5), border = NA, xaxt="n")
labs <- paste(names(table(df_bank$job)))
text(cex=1, x=x-.25, y=-2000, labs, xpd=TRUE, srt=90)
```



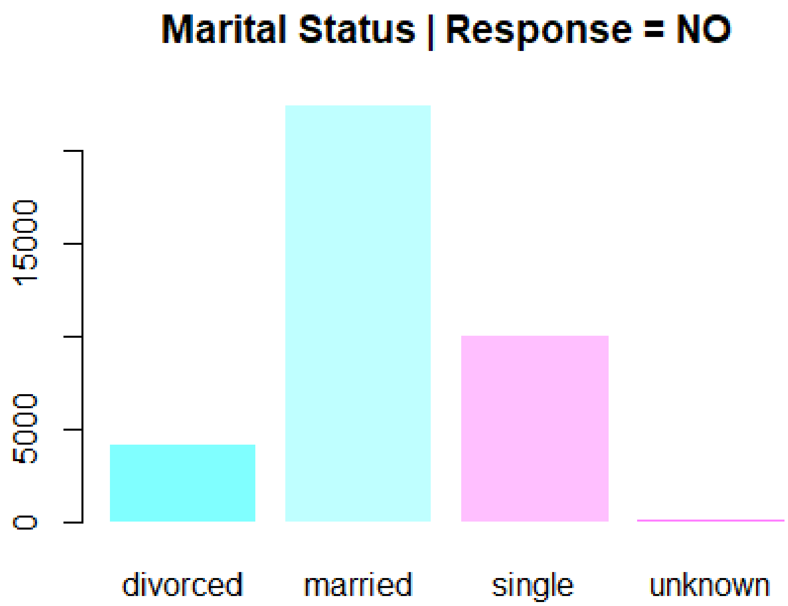

```
# Let's split the visualization according to the response to the marketing
calls

df_banky = subset(df_bank, y == 1)
df_bankn = subset(df_bank, y == 0)
View(df_bankn)

# Age distribution

library(lattice)
histogram(~df_bank$age|df_bank$y==1, col=heat.colors(14), border=NA)
```



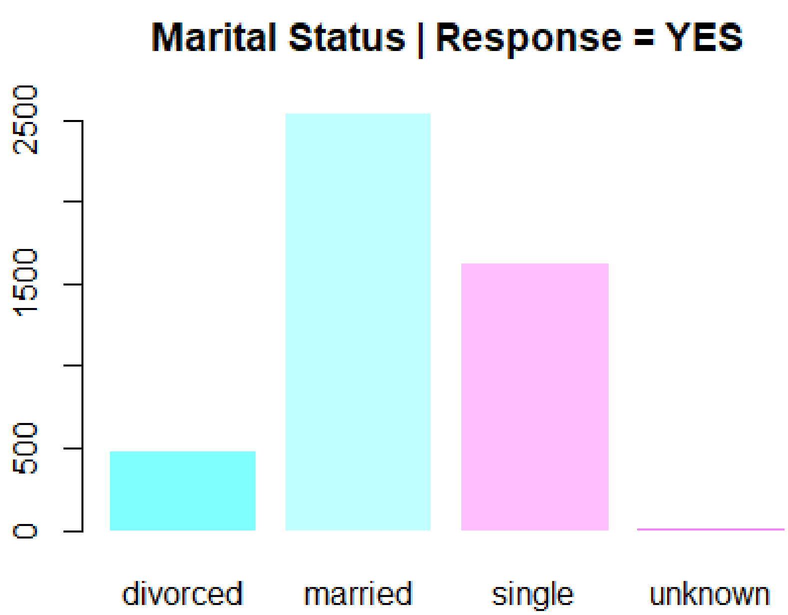

```
# The age range of people who responded negatively is comparatively more narrow; however, the difference looks to be marginal

# Length of the call duration

histogram(~df_bank$duration|df_bank$y==1, col=heat.colors(8), border=NA)
```



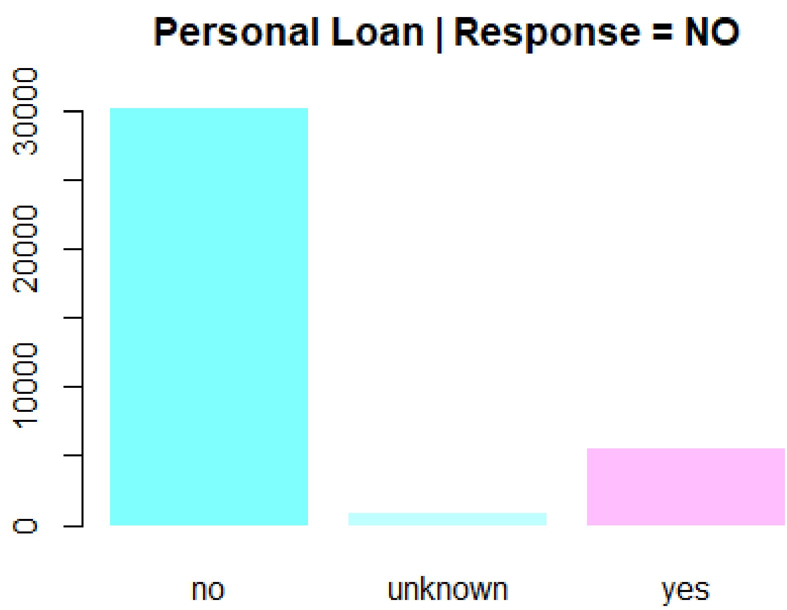

```r
# People responding positively tend to stay on the call for comparatively longer duration

# Marital Status

barplot(table(df_bankn$marital), main="Marital Status | Response = NO", col = cm.colors(4), border = NA)
```



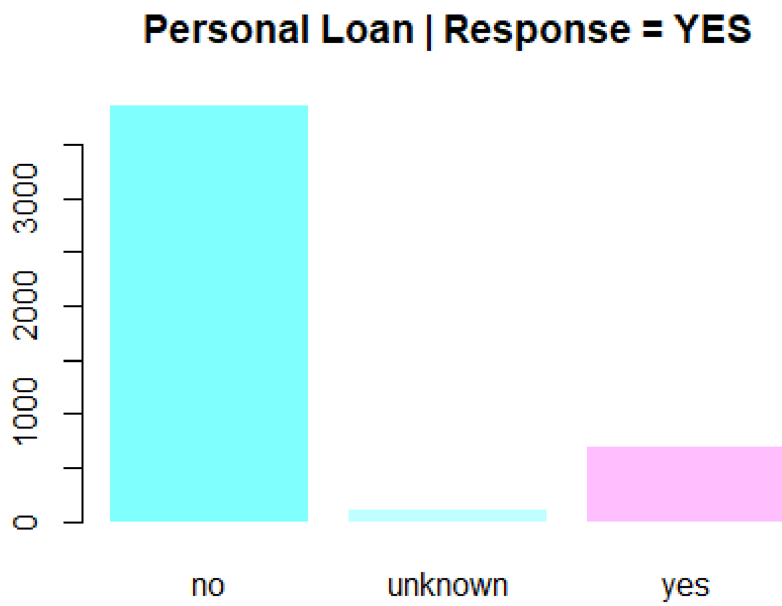

```
barplot(table(df_banky$marital), main="Marital Status | Response = YES", col
= cm.colors(4), border = NA)
```



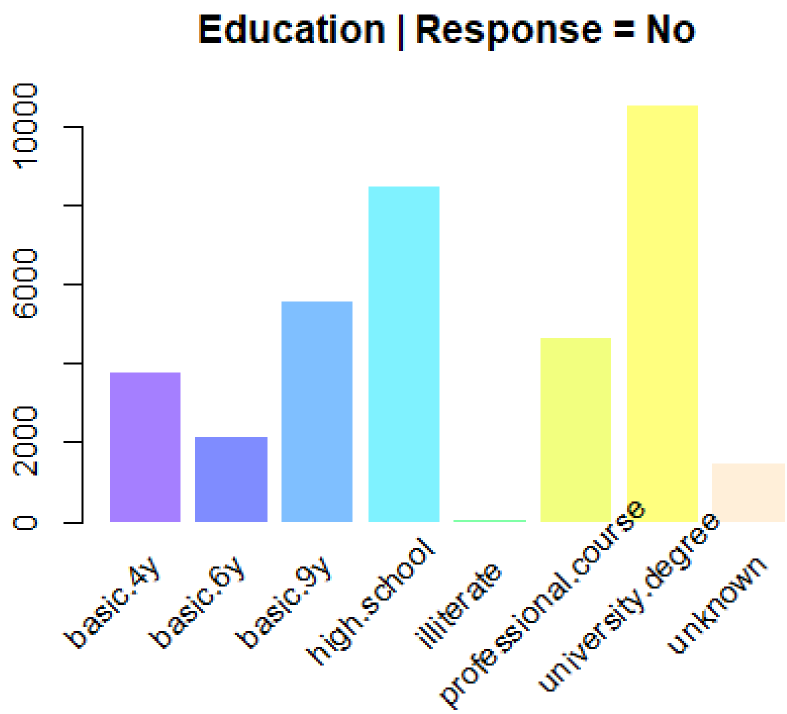
```r
# Clients responding positively have a greater proportion of single people

# Personal Loan

barplot(table(df_bankn$loan), main="Personal Loan | Response = NO", col =
cm.colors(4), border = NA)
```



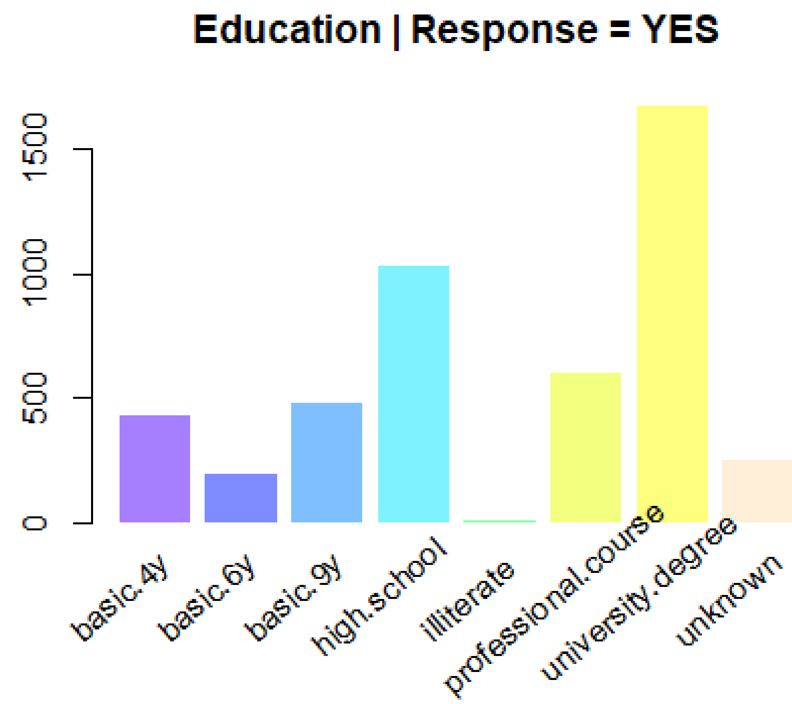

```r
barplot(table(df_banky$loan), main="Personal Loan | Response = YES", col =
cm.colors(4), border = NA)
```




```
# Distribution of the clients seems to be similar for clients responding
positively and negatively

# Day of The Week

x <- barplot(table(df_bankn$education), main = "Education | Response = No",
col = topo.colors(8, alpha=0.5), border = NA, xaxt="n")
labs <- paste(names(table(df_bankn$education)))
text(cex=1, x=x-.5, y=-2000, labs, xpd=TRUE, srt=45)
```

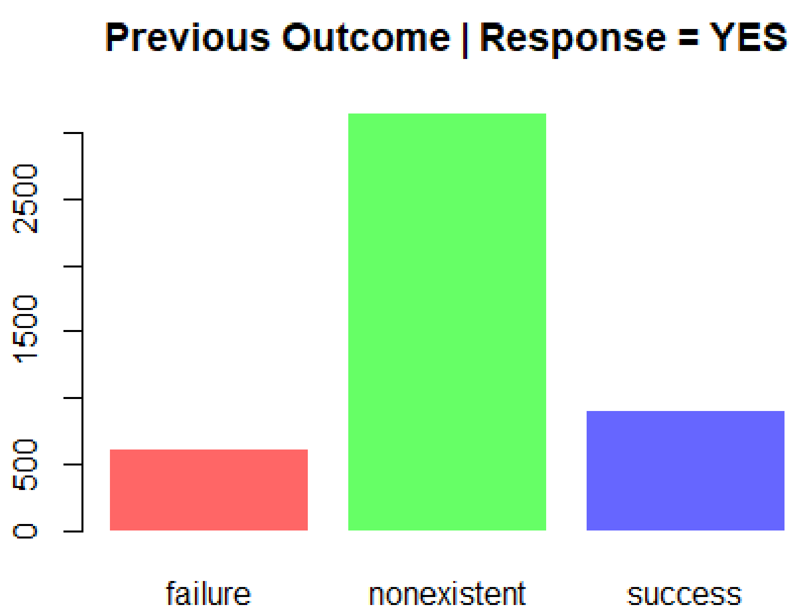



```
x <- barplot(table(df_banky$education), main = "Education | Response = YES",
col = topo.colors(8, alpha=0.5), border = NA, xaxt="n")
labs <- paste(names(table(df_banky$education)))
text(cex=1, x=x-.5, y=-300, labs, xpd=TRUE, srt=40)
```




```
# Groups of both the responses seem to have similar composition of education
levels

barplot(table(df_bankn$poutcome), main="Previous Outcome | Response = NO",
col = rainbow(3, alpha = 0.6), border = NA)
```


```r
barplot(table(df_banky$poutcome), main="Previous Outcome | Response = YES", col = rainbow(3, alpha = 0.6), border = NA)
```



```r
# More proportion of the people who said yes in the previous campaign
responded positively to this campaign

# Building Machine Learning Models

# As call duration can't be decided before initiating a call, this variable
can't be included in our prediction models
df_bank_model <- subset(df_bank, select = -c(duration))
View(df_bank_model)

# Converting contact into dummy variable. This variable had only 2 levels.

df_bank_model$contact=ifelse(df_bank_model$contact=="cellular",1,0)

# To ensure that the dummy variables are automatically made in the models, in
case they were not imported properly. These variables have more than 2
levels.

df_bank_model$job = as.factor(df_bank_model$job)
df_bank_model$marital = as.factor(df_bank_model$marital)
df_bank_model$education = as.factor(df_bank_model$education)
df_bank_model$default = as.factor(df_bank_model$default)
df_bank_model$housing = as.factor(df_bank_model$housing)
df_bank_model$loan = as.factor(df_bank_model$loan)
df_bank_model$month = as.factor(df_bank_model$month)
df_bank_model$day_of_week = as.factor(df_bank_model$day_of_week)
df_bank_model$pdays = as.factor(df_bank_model$pdays)

# Splitting the dataset into test (30%) and train

require(caTools)

## Loading required package: caTools

set.seed(1)

# Train and Test
sample = sample.split(df_bank_model, SplitRatio = 0.7)
df_train = subset(df_bank_model, sample==TRUE)
df_test = subset(df_bank_model, sample==FALSE)

# Check the number of 0s and 1s in the training data
table(df_train$y)

##
##     0     1
## 25564  3267

# Creating SMOTEd dataset

# We will use SMOTE to make the dataset balanced
# AS the SMOTE function needs the target variable in factor format, we will
```

```r
convert 'y' into a factor
library(DMwR)

## Loading required package: grid

df_train_smote <- df_train

df_train_smote$y = as.factor(df_train_smote$y)

df_train_smote <- SMOTE(y~.,df_train_smote,perc.over = 100,perc.under = 200)
table(df_train_smote$y)

##
##    0    1
## 6534 6534

# How perc.over and perc.under are used

# How the no. of rows for the minority class are decided
# New no. of minority class rows = Original no. of minority class rows X [1 +
(perc.over/100)]

# How the no. of rows for the majority class are decided
# New no. of majority class rows = [New no. of minority class rows - Original
no. of minority class rows]*(perc.under/100)

# For training our models we will convert the target variable - 'y' - back
into integer
df_train_smote$y=ifelse(df_train_smote$y=="1",1,0)
summary(df_train_smote$y)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     0.5     0.5     1.0     1.0

# LOGISTIC REGRESSION

logreg = glm(y~.,df_train,family="binomial")
y_pred_logreg = predict(logreg,df_test,type="response")

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

print("Logistic Regression Confusion Matrix from the unbalanced data")

## [1] "Logistic Regression Confusion Matrix from the unbalanced data"

conf_matrix_logreg <- table(df_test[,c(20)], y_pred_logreg>0.5)
conf_matrix_logreg

##
##     FALSE  TRUE
```

```
##   0 10826   158
##   1  1051   322
```

```r
library(InformationValue)

logreg_accuracy = sum(diag(conf_matrix_logreg))/sum(conf_matrix_logreg)


logreg_sen = sensitivity(df_test[,c(20)], y_pred_logreg, threshold = 0.5)
logreg_spe = specificity(df_test[,c(20)], y_pred_logreg, threshold = 0.5)
logreg_pre = precision(df_test[,c(20)], y_pred_logreg, threshold = 0.5)


print("Logistic Regression Results from the unbalanced dataset")
```

```
## [1] "Logistic Regression Results from the unbalanced dataset"
```

```r
print(paste0("Accuracy: ", round(logreg_accuracy*100,2),"%"))
```

```
## [1] "Accuracy: 90.22%"
```

```r
print(paste0("Sensitivity: ", round(logreg_sen*100,2),"%"))
```

```
## [1] "Sensitivity: 23.45%"
```

```r
print(paste0("Specificity: ", round(logreg_spe*100,2),"%"))
```

```
## [1] "Specificity: 98.56%"
```

```r
print(paste0("Precision: ", round(logreg_pre*100,2),"%"))
```

```
## [1] "Precision: 67.08%"
```

```r
# LOGISTIC REGRESSION on the SMOTEd dataset

logreg_smote <- glm(y~.,df_train_smote,family="binomial")
y_pred_logreg_smote <- predict(logreg_smote,df_test,type="response")

print("Logistic Regression Confusion Matrix after using SMOTE")
```

```
## [1] "Logistic Regression Confusion Matrix after using SMOTE"
```

```r
conf_matrix_logreg_smote <- table(df_test[,c(20)], y_pred_logreg_smote>0.5)
conf_matrix_logreg_smote
```

```
##
##     FALSE TRUE
##   0  9636 1348
##   1   546  827
```

```r
logreg_accuracy_smote =
sum(diag(conf_matrix_logreg_smote))/sum(conf_matrix_logreg_smote)
```

```r
logreg_sen_smote = sensitivity(df_test[,c(20)], y_pred_logreg_smote,
threshold = 0.5)
logreg_spe_smote = specificity(df_test[,c(20)], y_pred_logreg_smote,
threshold = 0.5)
logreg_pre_smote = precision(df_test[,c(20)], y_pred_logreg_smote, threshold
= 0.5)

print("Logistic Regression Results after using SMOTE")

## [1] "Logistic Regression Results after using SMOTE"

print(paste0("Accuracy: ", round(logreg_accuracy_smote*100,2),"%"))

## [1] "Accuracy: 84.67%"

print(paste0("Sensitivity: ", round(logreg_sen_smote*100,2),"%"))

## [1] "Sensitivity: 60.23%"

print(paste0("Specificity: ", round(logreg_spe_smote*100,2),"%"))

## [1] "Specificity: 87.73%"

print(paste0("Precision: ", round(logreg_pre_smote*100,2),"%"))

## [1] "Precision: 38.02%"

# DECISION TREE

library(rpart)

dectree = rpart(y~.,df_train,control = rpart.control(cp = 0.001))
y_pred_dectree = predict(dectree,df_test)

print("Decision Tree Confusion Matrix from the unbalanced data")

## [1] "Decision Tree Confusion Matrix from the unbalanced data"

conf_matrix_dectree <- table(df_test[,c(20)], y_pred_dectree>0.5)
conf_matrix_dectree

##
##     FALSE  TRUE
##   0 10781   203
##   1   996   377

dectree_accuracy = sum(diag(conf_matrix_dectree))/sum(conf_matrix_dectree)
dectree_sen = sensitivity(df_test[,c(20)], y_pred_dectree, threshold = 0.5)
dectree_spe = specificity(df_test[,c(20)], y_pred_dectree, threshold = 0.5)
dectree_pre = precision(df_test[,c(20)], y_pred_dectree, threshold = 0.5)

print("Decision Tree Results from the unbalanced dataset")
```

```r
## [1] "Decision Tree Results from the unbalanced dataset"

print(paste0("Accuracy: ", round(dectree_accuracy*100,2),"%"))

## [1] "Accuracy: 90.3%"

print(paste0("Sensitivity: ", round(dectree_sen*100,2),"%"))

## [1] "Sensitivity: 27.46%"

print(paste0("Specificity: ", round(dectree_spe*100,2),"%"))

## [1] "Specificity: 98.15%"

print(paste0("Precision: ", round(dectree_pre*100,2),"%"))

## [1] "Precision: 65%"

# DECISION TREE on the SMOTEd dataset

dectree_smote = rpart(y~.,df_train_smote,control = rpart.control(cp = 0.001))
y_pred_dectree_smote = predict(dectree_smote,df_test)

print("Decision Tree Confusion Matrix after using SMOTE")

## [1] "Decision Tree Confusion Matrix after using SMOTE"

conf_matrix_dectree_smote <- table(df_test[,c(20)], y_pred_dectree_smote>0.5)
conf_matrix_dectree_smote

##
##      FALSE TRUE
##   0  9923 1061
##   1   574  799

dectree_accuracy_smote =
sum(diag(conf_matrix_dectree_smote))/sum(conf_matrix_dectree_smote)


dectree_sen_smote = sensitivity(df_test[,c(20)], y_pred_dectree_smote,
threshold = 0.5)
dectree_spe_smote = specificity(df_test[,c(20)], y_pred_dectree_smote,
threshold = 0.5)
dectree_pre_smote = precision(df_test[,c(20)], y_pred_dectree_smote,
threshold = 0.5)


print("Decision Tree Results after using SMOTE")

## [1] "Decision Tree Results after using SMOTE"

print(paste0("Accuracy: ", round(dectree_accuracy_smote*100,2),"%"))
```

```
## [1] "Accuracy: 86.77%"

print(paste0("Sensitivity: ", round(dectree_sen_smote*100,2),"%"))

## [1] "Sensitivity: 58.19%"

print(paste0("Specificity: ", round(dectree_spe_smote*100,2),"%"))

## [1] "Specificity: 90.34%"

print(paste0("Precision: ", round(dectree_pre_smote*100,2),"%"))

## [1] "Precision: 42.96%"

# After using SMOTE, though the accuracy has declined (mainly as specificity
has been impacted), the sensitivity (improvement in which is needed)  has
improved greatly
```