Bank Marketing Analysis Report
Part 2

Gabriela Almeida Monteiro - s5198626
Jason Dias - s5216366
Julio Pimentel – s5172620

Griffith University
7031ICT Applied Data Mining
Lecturer: Dr Can Wang
Assessment item: Written Assignment

Word count: 2,598
Due date: 08/10/2021

Table of Contents

1. Introduction

A Portuguese bank was experiencing a decline in its revenues, and after some investigation, it was found that customers were not making enough term deposits (Maity, 2020). Financial institutions' significant income source comes from term deposits, especially from fixed-term investments, where investors can withdraw their money only after the term ends (Chen, 2021). Term deposits allow banks to use the capital for other investments and make a profit.

Following the findings, the Portuguese bank decided to perform direct marketing campaigns by phone calls to persuade their customers to subscribe to a term deposit (Moro et al., 2014). This project aims to build a classifier that can help the bank correctly predict whether a customer will subscribe to a term deposit based on some customer's features. This information can be helpful during the marketing campaign to target new customers that could potentially have a 'yes' response. But, on the other hand, it can also be beneficial to explore strategies to target the customers who answered 'no'.

For this project, two platforms were used to developed solutions, Python and R. Both solutions contain four different models: (1) Logistic Regression, (2) Support Vector Machine, (3) Decision Tree, and (4) Random Forests. The models will be compared using recall.

The report has the following structure: Exploratory Data Analysis, Data Preprocessing, Solutions, Key Results and Metrics, Summary, References, and Individual Contribution.

2. Exploratory Data Analysis

We first used the head() function to grasp our dataset and see which type of information it holds.

Figure 1. First 5 rows of the DataFrame.

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

The info() function showed us the number of rows and columns and the data type for each column.

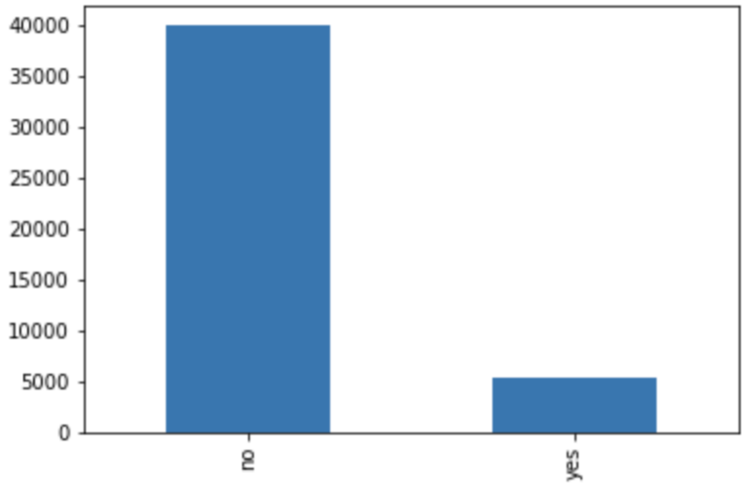
Figure 2. Summary of a DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
```

As we can see, the number of records available in the dataset is 45,211, and there are 17 columns, including the target variable “y”, which is the response (yes or no) to a term deposit subscription.

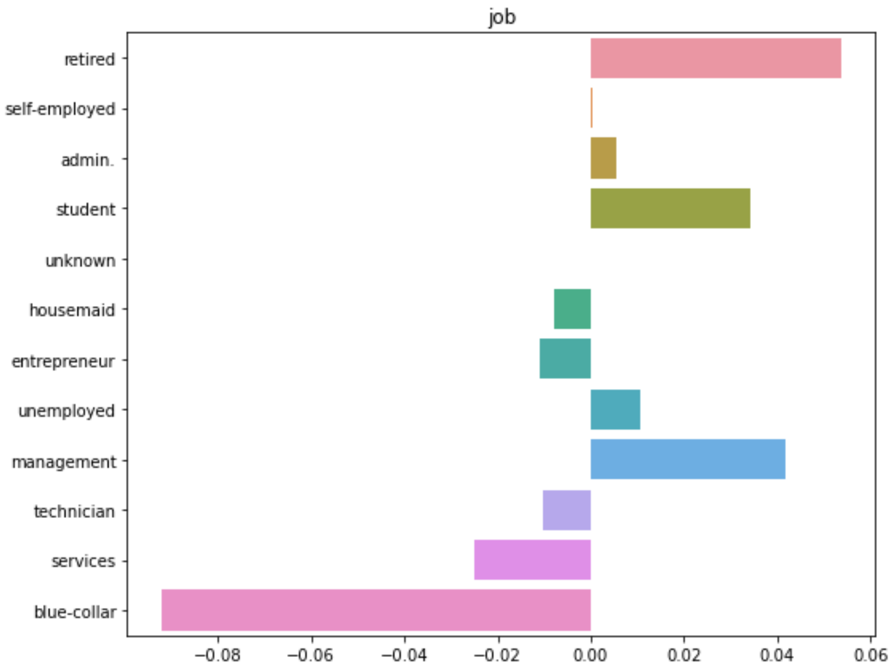
In this phase, we noted that the dataset is highly imbalanced for the target variable “y”, where 39,922 customers replied “no” to the term deposit and 5,189 customers replied “yes”. This characteristic of the dataset could impact training the model and be dealt with before fitting the models.

Figure 3. Frequency terms of ‘y’ variable.



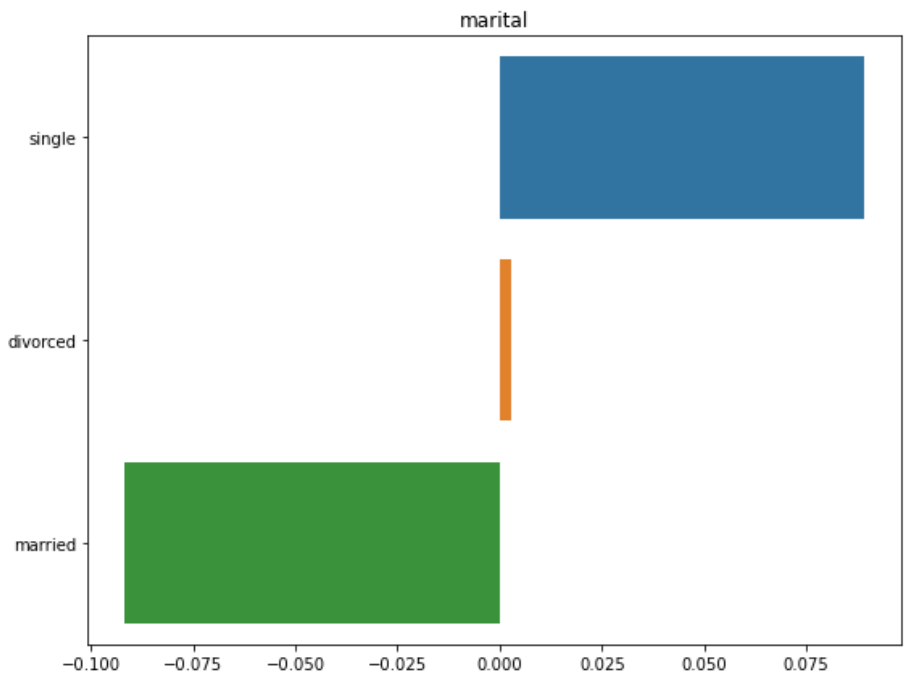
We further analyse the Job sector to know which sector could have more potential to accept term deposits.

Figure 4. Job sector analysis



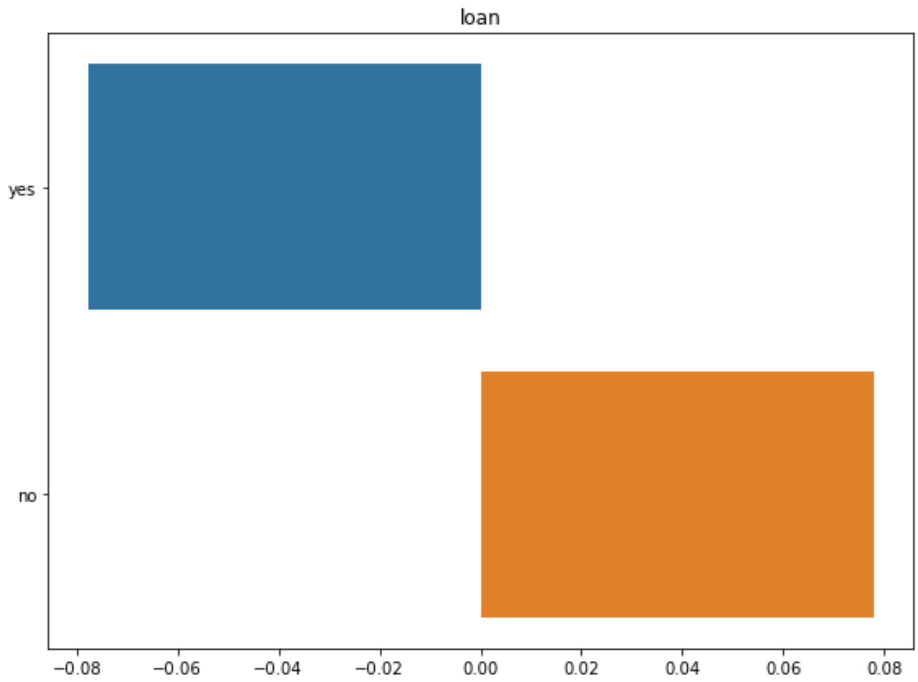
Here we can see that compared with all other sectors, the student, retired, and management sectors have the highest possibility of accepting term deposits. At the same time, blue-collar sectors would be the least targeted sectors.

Figure 5. Marital group analysis



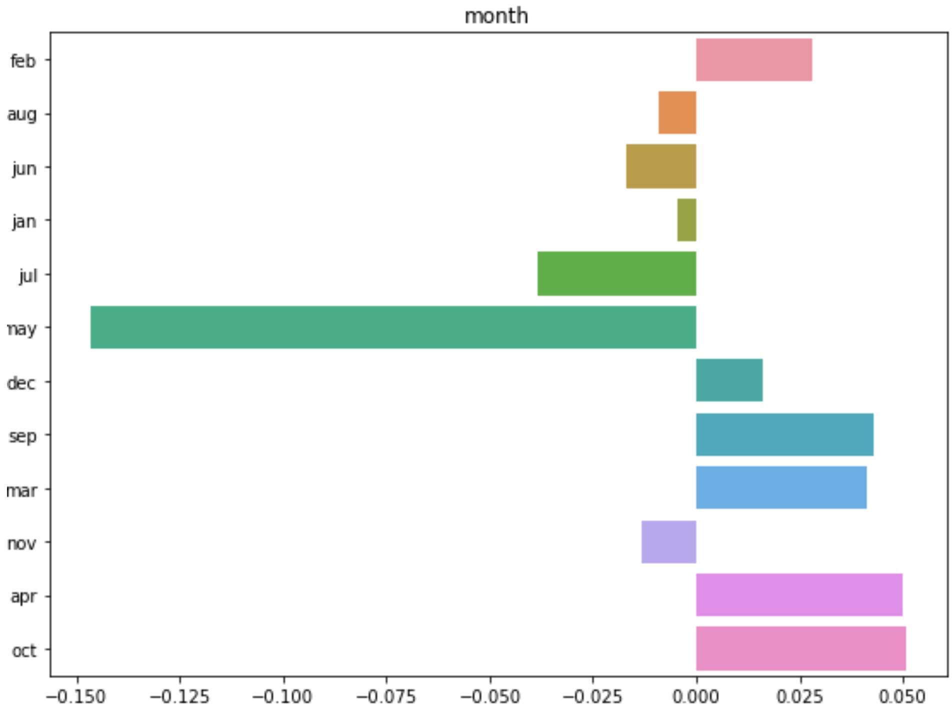
The above graph gives us the insight that a group of singles could be prioritised since the chances of them enrolling for term deposits are higher than that of the married group.

Figure 6. Loan analysis



Customers of the bank that have already a loan to their name have meagre chances of accepting a term deposit. One of the reasons could be the instalment payments of the loan, due to which they are unlikely to enrol for new investment.

Figure 7. Month analysis



Analysing the previous term deposit records, we can see that April, February, March, September, and October have seen the highest conversions. To sum up, for the bank to see their term deposit to have a positive impact and have a profitable turnover, the marketing team should focus on students and retired age group who are possible single and having no loan to their name. In addition, a promotional campaign could be rolled out in February, March, April, September and October to know how the new term deposit scheme performs.

3. Data Preprocessing

Before proceeding to data analysis, our team has applied some preprocessing techniques to the data:

- Columns with many “unknown” values were deleted since they would not contribute to the model—for example, the “poutcome” and “contact” columns.
- Categorical features were encoded. The columns ‘default’, ‘housing’, ‘loan’, and ‘y’ were transformed to ‘yes’ -> ‘1’ and ‘no’ -> ‘0’. Finally, we applied the one-hot-encoding technique for the columns ‘job’, ‘marital’, and ‘education’.
- The columns ‘age’ and ‘month’ were grouped in four and three categories, respectively.
- Numeric features were normalised. We applied a scaling transformation for the columns “balance”, “duration”, “campaign”, “pdays”, and “previous”.

After the data preprocessing techniques, the dataset was transformed from 17 to 36 columns. As a result, we created two additional datasets applying feature reduction techniques. First, we applied PCA with 15 variables. Next, we selected the top 6 correlated columns with the target variable. These additional datasets will be used to train and test the proposed models.

4. Solution

Three models were trained to predict whether a customer would subscribe or not to a term deposit based on their recorded features. The four models were Logistic Regression, Support Vector Machine, Decision Tree, and Random Forest. These models were implemented with Python (using Jupyter Notebooks) and in R (using Rstudio). Therefore, the models have different libraries used to build and perform the predictions for each environment.

4.1 Implementation in Python

Python's libraries used to develop this solution were Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, Collections, and Imblearn. In addition, we developed a function to automatise all the steps required for splitting the data, training the model, presenting the classification report, and storing the relevant performance metrics. Since all models follow the same structure, this function avoided redundant typing and helped the code be clean.

The function was called fit_classifier(), and it takes three arguments: the name of the model (model_name), the set of predictors (X) and the target variable (y). First, the function splits the dataset into 80% for training and 20% for testing, using the function train_test_split() from the Scikit-learn library. Next, it performs undersampling using the NearMiss() function, trains the model, and prints the confusion matrix, the accuracy score and the classification report with the information about precision, recall and F1 score for that model. The function also returns precision, recall, F1 score and accuracy of the model, stored in a DataFrame to compare all the models.

Figure 8. fit_classifier() function.

```
def fit_classifier(model_name, X, y):
    '''takes the name of the model, the predictors, target variable, and
    prints the confusion matrix, accuracy score and classification report.
    An undersampling strategy is used after splitting the data.'''

    # split the dataset before undersampling
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2021, train_size = 0.8, shuffle = True)


    # Apply undersampling to the training set
    # define the undersampling method
    undersample = NearMiss(version=3, n_neighbors=10, sampling_strategy=0.2)
    # transform the dataset
    X_under, y_under = undersample.fit_resample(X_train, y_train)
    X_train = X_under
    y_train = y_under

    #Training the model
    my_model = model_name # Train the model
    my_model.fit(X_train, y_train) # fit the model
    pred = my_model.predict(X_test) # Predict the response

    #Creating a confusion matrix
    conf_mat= metrics.confusion_matrix(y_test, pred)
    conf_mat_df=pd.DataFrame(conf_mat, index=['no','yes'], columns=['no','yes'] )
    print("----------- CONFUSION MATRIX ------------------------")
    print(conf_mat_df,'\n\n')

    # Model Accuracy: how often is the classifier correct?
    print("----------- ACCURACY --------------------------------")
    print("Accuracy:",metrics.accuracy_score(y_test, pred),'\n\n')
    print("----------- CLASSIFICATION REPORT -------------------")
    print(metrics.classification_report(y_test, pred))


    return [round(metrics.precision_score(y_test, pred),3),
            round(metrics.recall_score(y_test, pred),3),
            round(metrics.f1_score(y_test, pred),3),
           round(metrics.accuracy_score(y_test, pred),3)]
```

Each model was stored in a variable to be passed as an argument in the fit_classifier() function. For the Support Vector Machine, we have decided to experiment with three different kernels.

Figure 9. Proposed models to predict the target variable.

```
#Define the parameters of the model
logistic_reg = linear_model.LogisticRegression()
sup_vector_linear = svm.SVC(kernel = 'linear')
sup_vector_rbf = svm.SVC(kernel = 'rbf')
sup_vector_poly = svm.SVC(kernel = 'poly')
decision_tree=DecisionTreeClassifier()
ran_classifier = RandomForestClassifier()
```

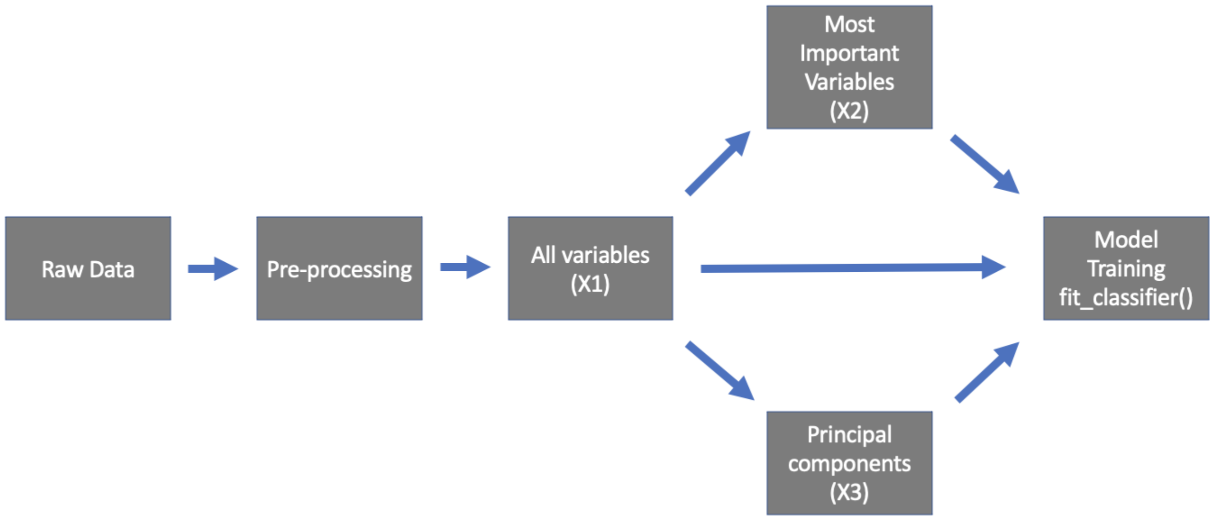
It is important to note that the dataset is imbalanced. As a result, performing a sampling technique was necessary to balance the data and avoid model bias. For this study, undersampling was chosen because it is less computationally expensive.

In this program, each model experimented with three different sets of predictors to find the best performance in each model. These predictors are explained below:

- *All variables (X1)*: all features present on the dataset were considered (except those deleted due to excessive "unknown" values).
- *Most important variables (X2)*: a correlation matrix was generated between all features and the target variable. The most correlated variables were selected.
- *Principal components (X3)*: 15 principal components were generated.

The picture below describes the process of generating the different types of variables to train the model.

Figure 10. Process of training machine learning models.



The process of training a model pursued the following structure:

a. Logistic Regression Model

- All variables (X1): fit_classifier(logistic_reg, X1, y)
- Most important variables (X2): fit_classifier(logistic_reg, X2, y)
- Principal components (X3): fit_classifier(logistic_reg, X3, y)

b. Support Vector Machine Model (with 3 different kernels)

- All variables (X1) and linear kernel: fit_classifier(sup_vector_linear), X1, y)
- Most important variables (X2) and linear kernel: fit_classifier(sup_vector_linear), X2, y)
- Principal components (X3) and linear kernel: fit_classifier(sup_vector_linear), X3, y)
- All variables (X1) and poly kernel: fit_classifier(sup_vector_poly), X1, y)
- Most important variables (X2) and poly kernel: fit_classifier(sup_vector_poly), X2, y)

- Principal components (X3) and poly kernel: fit_classifier(sup_vector_poly), X3, y)
- All variables (X1) and rbf kernel: fit_classifier(sup_vector_rbf), X1, y)
- Most important variables (X2) and rbf kernel: fit_classifier(sup_vector_rbf), X2, y)
- Principal components (X3) and rbf kernel: fit_classifier(sup_vector_rbf), X3, y)

c. Decision Tree Model

- All variables (X1): fit_classifier(decision_tree, X1, y)
- Most important variables (X2): fit_classifier(decision_tree, X2, y)
- Principal components (X3): fit_classifier(decision_tree, X3, y)

d. Random Forest Model

- All variables (X1): fit_classifier(ran_classifier, X1, y)
- Most important variables (X2): fit_classifier(ran_classifier, X2, y)
- Principal components (X3): fit_classifier(ran_classifier, X3, y)

4.2 Implementation in R

The R solution requires the train and test data sets from the previous Python implementation. The code was developed to have the same training and test sets for both solutions and compare the results. R's libraries used to create this solution were *readr, plyr, dplyr, e1071, rpart, ROSE,* and *caret*. In addition, we only trained this implementation using all the variables.

Figure 11. R libraries.

```
# --------------- 2. Loading Libraries ---------------

library(readr)
library(plyr)
library(dplyr)
library(e1071)
library(rpart)
library(ROSE)
library(caret)
```

We performed undersampling in the training set, the same sampling technique used on Python. However, in this implementation, we equalled both target classes to 3,997 records for each of them. In addition, the models were set up with 10 Cross Validations and accuracy as a performance metric. Finally, the precision, recall, F1 score, and accuracy of each model were calculated.

Figure 12. Training models on R.

```
# --------------- 4. Undersampling the training set ---------------

# Source -> https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classi
n_yes <- length(which(train$y == 1)) # finding the number of rows where y = yes

df_balanced_under <- ovun.sample(y ~ ., data = train, method = "under", N = n_yes*2, seed = 202
table(df_balanced_under$y)


# --------------- 5. Building Logistic Regression model ---------------

control <- trainControl(method="cv", number=10)
metric <- "Accuracy"

#Train model
set.seed(7)
logreg <- train(y~., data=df_balanced_under, method="lda", metric=metric, trControl=control)
y_pred <- predict(logreg, newdata = test)
```

5. Key Results and Metrics

Our group has decided that recall was the best metric to measure the performance of the models. The primary purpose of the bank is to target as many “yes” customers as possible. In other words, we would like to avoid classifying customers as “no” when, in fact, they are “yes” customers. It indicates that we need a high recall. The trade-off would be that we would end up mislabelling the “no” customers as if they were “yes” customers (low precision), but this is a better situation than missing the potential “yes” customers. Plus, if the bank invests in good sellers, there is a possibility that even the mislabelled customers could be converted to “yes” customers. As a result, a high recall would allow call-centre workers to spend their time effectively with customers that as likely to submit to the term deposit.

5.1 Models built-in Python (Jupyter Notebook)

The table below shows the performance metrics for all the models used for training in Python. The models are sorted by the highest recall, which was the metric that we have chosen to evaluate the best model. As we can see, the Decision Tree with all the variables is the one with the highest recall, followed by Random Forest. All six combinations of Decision Tree and Random Forest score remarkably high in terms of recall.

The selected cells show the best recall achieved for each model. For the sake of simplicity, we will use only these to conduct our analysis in R.

Table 1. Performance metrics per model (sorted by highest recall).

Model	Variables	Precision	Recall	F1 Score	Accuracy
Decision tree	All variables	0.254	0.495	0.336	0.768
Random forest	All variables	0.564	0.484	0.521	0.895
Decision tree	PCA variables	0.218	0.479	0.300	0.734
Random forest	PCA variables	0.535	0.407	0.462	0.888
Random forest	Most relevant variables	0.445	0.398	0.421	0.870
Decision tree	Most relevant variables	0.373	0.367	0.370	0.852
SVM rbf	All variables	0.611	0.354	0.448	0.897
SVM rbf	PCA variables	0.612	0.319	0.420	0.895
SVM rbf	Most relevant variables	0.604	0.303	0.403	0.894
SVM poly	All variables	0.580	0.294	0.390	0.891
Logistic regression	All variables	0.553	0.262	0.355	0.887
Logistic regression	PCA variables	0.581	0.229	0.328	0.889
Logistic regression	Most relevant variables	0.598	0.224	0.326	0.890
SVM linear	All variables	0.451	0.184	0.261	0.877
SVM poly	PCA variables	0.581	0.176	0.270	0.887
SVM linear	PCA variables	0.626	0.126	0.210	0.887
SVM linear	Most relevant variables	0.605	0.090	0.156	0.885
SVM poly	Most relevant variables	0.629	0.088	0.154	0.886

5.2 Models built-in R (Rstudio)

In R, we have implemented the combinations with the best recall in each model. As we can see, the results show that the Decision Tree shows exceptionally superior results, confirming what we have seen in Python. It is also worth noting that Random Forest has identical performance to the Decision tree in R, suggesting that both models could be used interchangeably. Regarding the other models, SVM with RBF is better than Logistic Regression, as we have also seen in the implementation in Python.

Table 2. Performance metrics per model.

Model	Variable	Precision	Recall	F1 Score	Accuracy
Decision tree	All variables	0.445	0.932	0.602	0.854
Random forest	All variables	0.445	0.932	0.602	0.854
SVM - rbf	All variables	0.407	0.895	0.560	0.833
Logistic regression	All variables	0.397	0.851	0.541	0.829

5.3 Comparison between Python and R

When comparing the recall metric in Python and R, we note a significant difference between them, although the ranking between the models does not change. Even though the values for the recall change, we can still see the same order where Decision Tree and Random Forest are better than SVM - RBF, which is better than logistic regression.

Table 3. Python x R

	Recall	
Model	**Python**	**R**
Decision tree	0.495	0.932
Random forest	0.484	0.932
SVM - rbf	0.354	0.895
Logistic regression	0.262	0.851

6. Summary

The data preprocessing was a vital aspect of this implementation. The binary categorical features were mapped to "1" and "0" when they had "yes" and "no". In the case of multiple labels, one-hot-encoding was the best alternative for these columns. Finally, all numerical features were normalised to improve the performance of the models. Nevertheless, when we implemented reducing dimensionality strategies, the recall performance did not improve compared with using all the variables. It was a surprising discovery because, according to previous readings, these strategies should improve the performance in our specific case by having 39 columns.

Although we trained our model in a balanced dataset, it didn't predict the "yes" customers so well in both implementations. This insight makes us think that the model learned a few patterns for the "yes" customers. Still, apart from the "typical" profile of the "yes" customer, there is also probably another important factor involved, which is the seller's ability to convince the customer to subscribe. Unfortunately, the dataset did not contain information about the person who performed the calls. Since this was a conversation, it is essential to know more about both parties involved in the phone calls. Our team believes that this information could play a critical role in predicting the "yes" customers.

It caught our attention that the dataset did not contain a column for gender. We wondered if that was how the dataset was designed or if that column was purposefully dropped before the dataset was uploaded to avoid training algorithms that could be biased. Since this is a bank in the European Union, it is reign by the General Data Protection Regulation (GDPR), the EU law on data protection and privacy. Our group considers algorithms bias and the importance of having critical thinking when building models that could increase disparities and promote injustices.

A surprising result was that the R solution had high recall results on all the models. Unfortunately, we did not explore this alternative in-depth since R was not our primary programming language. The main differences in the R solution were having the same number of target classes and implementing a Cross-Validation on the training set.

Finally, our analysis suggests that the Decision Tree is the best model to predict if a customer will subscribe or not to a term deposit. We could verify that by training the data with different models both in Python and in R, then comparing the recall metric for all models. Therefore, our recommendation for the Portuguese Bank is to use the Decision Tree in their customer database to generate a list of potential term deposit subscribers. This list could be handed to sellers to market this product and increase the bank revenues.

7. References

Analytics Vidhya. (2016, March 28). *Practical Guide to deal with Imbalanced Classification Problems in R*. Retrieved from Analytics Vidhya : https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/

Brownlee, J. (2016, February 3). *Your First Machine Learning Project in R Step-By-Step*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/machine-learning-in-r-step-by-step/

Chen, J. (2021, April 29). *Investopedia*. Retrieved from Term Deposit: https://www.investopedia.com/terms/t/termdeposit.asp

Krishnan, M. (2018, July 7). *Understanding the Classification report through sklearn*. Retrieved from Muthukrishnan : https://muthu.co/understanding-the-classification-report-in-sklearn/

Maity, A. (2020, March 22). *Predict if the client will subscribe a term deposit or not, using 'Machine learning'* . Retrieved from https://medium.com/@ashim.maity8/predict-if-the-client-will-subscribe-a-term-deposit-or-not-using-machine-learning-c6e4024c7028

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

Thanusan, S. (2021, June). *Analysis of Bank Marketing Dataset By using Support Vector Machine (SVM)*. Retrieved from Medium: https://medium.com/nerd-for-tech/analysis-of-bank-marketing-dataset-by-using-support-vector-machine-svm-1ccae6eaa782

8. Individual Contribution

Gabriela

I have contributed with:

- Creation of the function fit_classifier() in python
- Development of the R code
- Section 1, Introduction
- Section 2, Exploratory Data Analysis
- Section 3, Data Preprocessing
- Section 4.1. Implementation in Python (including design of Figure 10)
- Section 5, Key Results and Metrics
- Section 5.1, Models built-in Python
- Section 5.2, Models built-in R
- Comparison between Python and R
- Summary

9. Appendices

Appendix A.
Python program.

Data exploration

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import datasets, model_selection, linear_model, metrics
from collections import Counter
from imblearn.under_sampling import NearMiss
```

In [3]:
```
#Read bank-full.csv file
df = pd.read_csv('bank-full.csv',sep=';')
df.head()
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

1. Data Exploration

In [4]:
```
#Data type and null values for each column
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
None
```

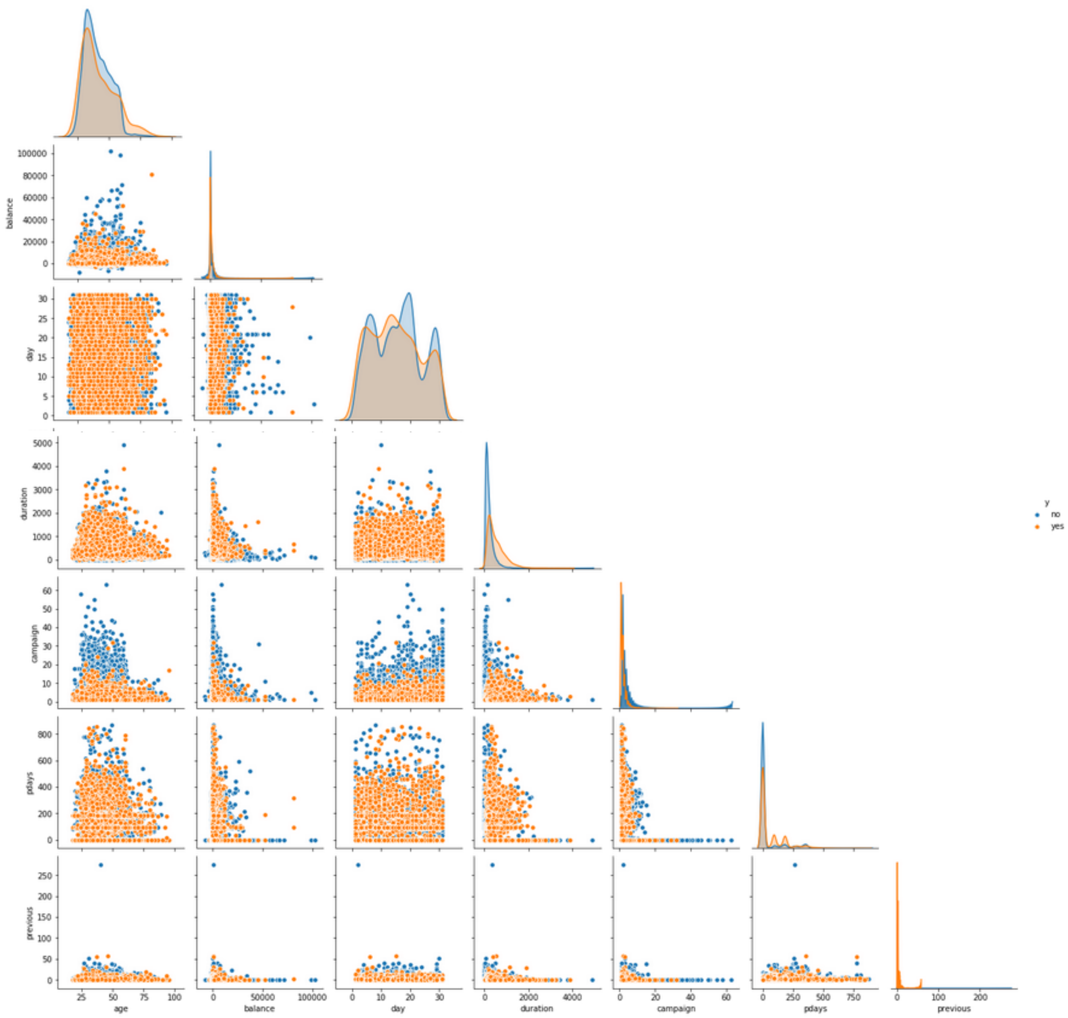
In [5]:
```python
#Review the first 5 values
df.head()
```

Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

In [61]:
```python
# #Plot numerical values
sns.pairplot(df,hue='y',corner=True)
```

Out[61]: <seaborn.axisgrid.PairGrid at 0x1a3b1e56e88>



In [7]:
```
#Statistical information from numerical columns
df.describe()
```

Out[7]:

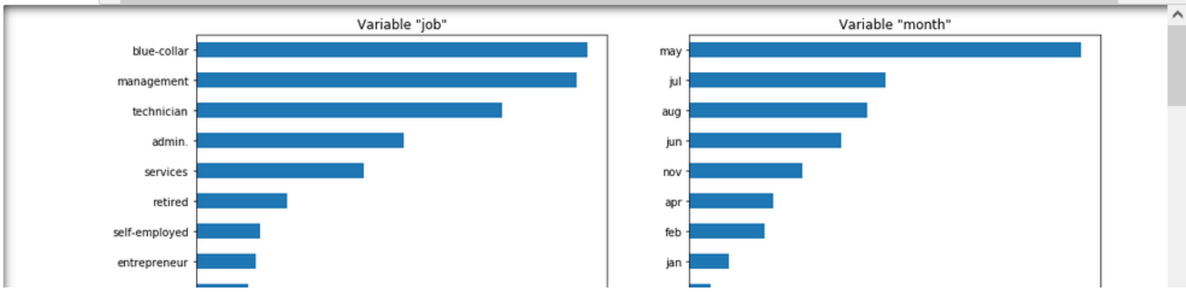
	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

Check unique values of object columns. Define if some of the columns will be transformed using one hot encoder.

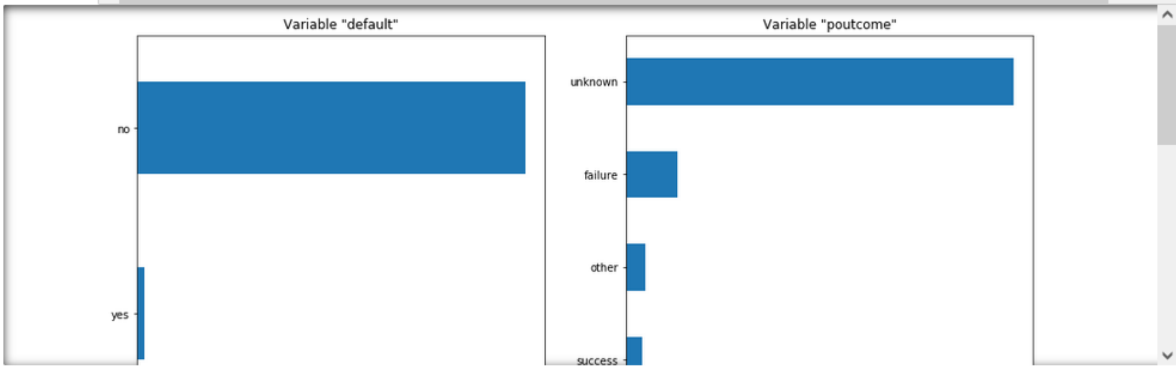
In [8]:
```
#multilabel
col=1
print('Column {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#3 labels
col=2
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#multilabel
col=3
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#2 labels
col=4
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())
#2 labels
col=6
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#2 labels
col=7
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#3 labels
col=8
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#months in text - 12 labels
col=10
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#multilabel
col=15
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())

#multilabel
col=16
print('\nColumn {} ({}) unique values:'.format(col,df.columns[col]))
print(df.iloc[:,col].value_counts())
```

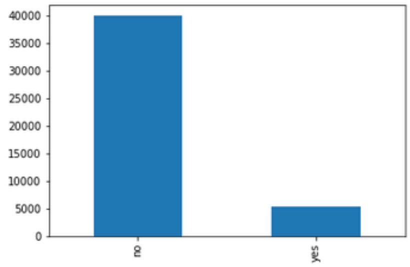
```
Column 1 (job) unique values:
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
```

In [9]:
```python
fig, axs = plt.subplots(nrows=3, ncols=2, figsize = (15,21))
plt.subplots_adjust(wspace=0.2, hspace=0.2)

col=1
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=2
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=3
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=7
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=8
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=10
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

plt.show()
```



In [10]:
```python
fig, axs = plt.subplots(nrows=2, ncols=2, figsize = (15,14))
plt.subplots_adjust(wspace=0.2, hspace=0.2)


col=4
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=6
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=15
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

col=16
df[df.columns[col]].value_counts().sort_values().plot(kind='barh',title='Variable "'+df.columns[col]+'"', ax = a

plt.show()
```



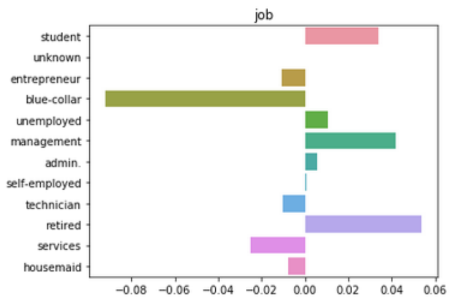
In [11]:
```
df['y'].value_counts().plot(kind = 'bar')
```
Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3b1f92708>
```


In [12]:
```
# This data set contains an imbalanced target variable.
df['y'].value_counts()
```
Out[12]:
```
no     39922
yes     5289
Name: y, dtype: int64
```

Analyse different variables to find a specific category has more chances of getting a term deposit.

Positive values imply this category favors clients that will subscribe and negative values categories that favour not buying

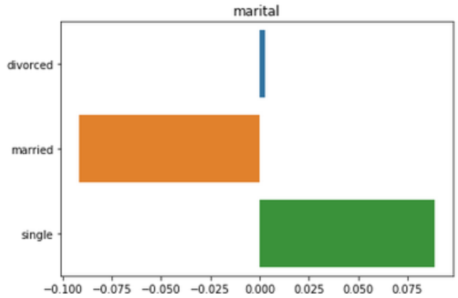
1.1 Job sector variable

In [13]:
```
feature_name="job"
pos_count = df.loc[df.y.values == 'yes', feature_name].value_counts()
neg_count = df.loc[df.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_count.index) + list(neg_count.index)))

#Count of how often each outcome was recorded
freq_pos = (df.y.values == 'yes').sum()
freq_neg = (df.y.values =='no').sum()

pos_counts = pos_count.to_dict()
neg_counts = neg_count.to_dict()
all_index= list(all_counts)

all_counts = [pos_counts.get(k,0) / freq_pos - neg_counts.get(k,0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```


Blue-collar jobs are less likely to get a term deposit. Meanwhile, retired, management and student jobs are more likely to get a term deposit.

1.2 Marital status variable

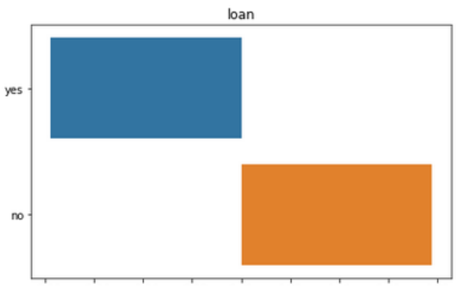
In [14]:
```
feature_name="marital"
pos_count = df.loc[df.y.values == 'yes', feature_name].value_counts()
neg_count = df.loc[df.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_count.index) + list(neg_count.index)))

#Count of how often each outcome was recorded
freq_pos = (df.y.values == 'yes').sum()
freq_neg = (df.y.values =='no').sum()

pos_counts = pos_count.to_dict()
neg_counts = neg_count.to_dict()
all_index= list(all_counts)

all_counts = [pos_counts.get(k,0) / freq_pos - neg_counts.get(k,0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```



Here we can see that customers who are single are mostly likely in accepting term deposits

1.3 Previous loan variable

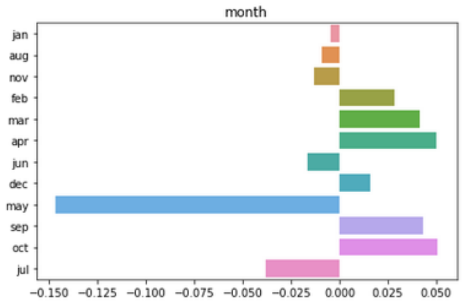
In [15]:
```
feature_name="loan"
pos_count = df.loc[df.y.values == 'yes', feature_name].value_counts()
neg_count = df.loc[df.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_count.index) + list(neg_count.index)))

#Count of how often each outcome was recorded
freq_pos = (df.y.values == 'yes').sum()
freq_neg = (df.y.values =='no').sum()

pos_counts = pos_count.to_dict()
neg_counts = neg_count.to_dict()
all_index= list(all_counts)

all_counts = [pos_counts.get(k,0) / freq_pos - neg_counts.get(k,0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```



By observing the above diagram we can say that people who have not taken a loan have applied for term deposit

1.4 Month variable

In [16]:
```
feature_name="month"
pos_count = df.loc[df.y.values == 'yes', feature_name].value_counts()
neg_count = df.loc[df.y.values == 'no', feature_name].value_counts()

all_counts = list(set(list(pos_count.index) + list(neg_count.index)))

#Count of how often each outcome was recorded
freq_pos = (df.y.values == 'yes').sum()
freq_neg = (df.y.values =='no').sum()

pos_counts = pos_count.to_dict()
neg_counts = neg_count.to_dict()
all_index= list(all_counts)

all_counts = [pos_counts.get(k,0) / freq_pos - neg_counts.get(k,0) / freq_neg for k in all_counts]

sns.barplot(all_counts, all_index)
plt.title(feature_name)
plt.tight_layout()
```



With the above analysis, we can say that during the months of March, April, September and October is more likely to get a term posit. Meanwhile, May and July are the less likely to get a term deposit.

2. Data Cleaning

In [63]:
```
#Remove 'contact' and 'poutcome' columns because they have large number of unknown values
df_clean=df.drop(columns=['contact','poutcome'])
df_clean.head()
```

Out[63]:

	age	job	marital	education	default	balance	housing	loan	day	month	duration	campaign	pdays	previous	y
0	58	management	married	tertiary	no	2143	yes	no	5	may	261	1	-1	0	no
1	44	technician	single	secondary	no	29	yes	no	5	may	151	1	-1	0	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	5	may	76	1	-1	0	no
3	47	blue-collar	married	unknown	no	1506	yes	no	5	may	92	1	-1	0	no
4	33	unknown	single	unknown	no	1	no	no	5	may	198	1	-1	0	no

2.1 Education column

In [64]:
```
#Remove rows when 'education' has 'unknown' values
df_clean=df_clean[df_clean['education']!='unknown']
print('Df has {} values'.format(len(df_clean)))
```

```
Df has 43354 values
```

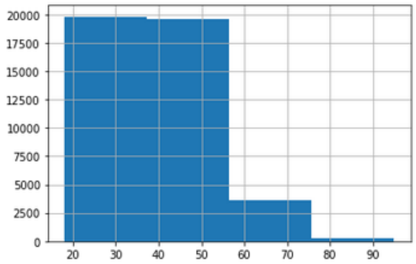
2.2 Job column

In [65]:
```
#Remove rows when 'job' has 'unknown' values

df_clean = df_clean[df_clean['job']!='unknown']
print('Df has {} values'.format(len(df_clean)))
```

```
Df has 43193 values
```

In [22]:
```
df_clean.age.hist(bins=4)
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3b262a0c8>
```



2.3 Age column

In [66]:
```
#Divide the age in 4 bins
bins=4

#Define variables to split the 'age' column
age_max=df_clean.age.max()
age_min=df_clean.age.min()
range_age=age_max-age_min
dist_age=range_age/bins

lim_min=age_min
lim_max=age_min + dist_age
sum_vals=0

for times in range(bins):
    col_name='age_'+str(int(lim_min))+'_'+str(int(lim_max))
    df_clean[col_name]=((df_clean.age >= lim_min) & (df_clean.age < lim_max)) *1
    df_temp=df_clean[(df_clean.age >= lim_min) & (df_clean.age < lim_max)]
    sum_vals+=len(df_temp)
    print(sum_vals, len(df_temp))

    lim_min=lim_max
    lim_max+=dist_age

df_clean=df_clean.drop(columns=['age'])
df_clean.head()
```

```
19804 19804
39393 19589
42966 3573
43191 225
```

Out[66]:

	job	marital	education	default	balance	housing	loan	day	month	duration	campaign	pdays	previous	y	age_18_37	age_37_56	age_5
0	management	married	tertiary	no	2143	yes	no	5	may	261	1	-1	0	no	0	0	
1	technician	single	secondary	no	29	yes	no	5	may	151	1	-1	0	no	0	1	
2	entrepreneur	married	secondary	no	2	yes	yes	5	may	76	1	-1	0	no	1	0	
5	management	married	tertiary	no	231	yes	no	5	may	139	1	-1	0	no	1	0	
6	management	single	tertiary	no	447	yes	yes	5	may	217	1	-1	0	no	1	0	

2.4 Month column

In [67]:
```
#Transform month in trimesters
df_clean['month_T1']=df_clean.month.isin(['jan','feb','mar','apr'])*1
df_clean['month_T2']=df_clean.month.isin(['may','jun','jul','ago'])*1
df_clean['month_T3']=df_clean.month.isin(['sep','oct','nov','dic'])*1

#Remove month column
df_clean=df_clean.drop(columns=['month'])
df_clean.head()
```

Out[67]:

	job	marital	education	default	balance	housing	loan	day	duration	campaign	pdays	previous	y	age_18_37	age_37_56	age_56_75	a
0	management	married	tertiary	no	2143	yes	no	5	261	1	-1	0	no	0	0	1	
1	technician	single	secondary	no	29	yes	no	5	151	1	-1	0	no	0	1	0	
2	entrepreneur	married	secondary	no	2	yes	yes	5	76	1	-1	0	no	1	0	0	
5	management	married	tertiary	no	231	yes	no	5	139	1	-1	0	no	1	0	0	
6	management	single	tertiary	no	447	yes	yes	5	217	1	-1	0	no	1	0	0	

2.5 Day column

In [68]:
```
#Group the column in 10 days bin
df_clean['day_1_10']=((df_clean.day > 0) & (df_clean.day <= 10))*1
df_clean['day_11_20']=((df_clean.day > 10) & (df_clean.day <= 20))*1
df_clean['day_21_31']=(df_clean.day > 20)*1

#Remove month column
df_clean=df_clean.drop(columns=['day'])
df_clean.head()
```

Out[68]:

	job	marital	education	default	balance	housing	loan	duration	campaign	pdays	...	age_18_37	age_37_56	age_56_75	age_75_95	month
0	management	married	tertiary	no	2143	yes	no	261	1	-1	...	0	0	1	0	
1	technician	single	secondary	no	29	yes	no	151	1	-1	...	0	1	0	0	
2	entrepreneur	married	secondary	no	2	yes	yes	76	1	-1	...	1	0	0	0	

2.6 Default, housing, loan columns

In [69]:
```
# #transform 'yes', 'no' to numerical values
df_clean.default=df_clean.default.map(dict(yes=1, no=0))
df_clean.housing=df_clean.housing.map(dict(yes=1, no=0))
df_clean.loan=df_clean.loan.map(dict(yes=1, no=0))
df_clean.y=df_clean.y.map(dict(yes=1, no=0))

df_clean.head()
```

Out[69]:

	job	marital	education	default	balance	housing	loan	duration	campaign	pdays	...	age_18_37	age_37_56	age_56_75	age_75_95	month
0	management	married	tertiary	0	2143	1	0	261	1	-1	...	0	0	1	0	
1	technician	single	secondary	0	29	1	0	151	1	-1	...	0	1	0	0	
2	entrepreneur	married	secondary	0	2	1	1	76	1	-1	...	1	0	0	0	
5	management	married	tertiary	0	231	1	0	139	1	-1	...	1	0	0	0	
6	management	single	tertiary	0	447	1	1	217	1	-1	...	1	0	0	0	

5 rows × 22 columns

In [70]:
```
df_clean.columns
```

Out[70]:
```
Index(['job', 'marital', 'education', 'default', 'balance', 'housing', 'loan',
       'duration', 'campaign', 'pdays', 'previous', 'y', 'age_18_37',
       'age_37_56', 'age_56_75', 'age_75_95', 'month_T1', 'month_T2',
       'month_T3', 'day_1_10', 'day_11_20', 'day_21_31'],
      dtype='object')
```

2.7 Job, marital, education columns

In [71]:
```
#Perform one hot encoding for the categorical features
one_hot_encoding = pd.get_dummies(df_clean.iloc[:,[0,1,2]])
df_clean=pd.concat([df_clean,one_hot_encoding],axis=1)
df_clean=df_clean.drop(columns=['job','marital','education'])

df_clean.head()
```

Out[71]:

	default	balance	housing	loan	duration	campaign	pdays	previous	y	age_18_37	...	job_services	job_student	job_technician	job_unemployed
0	0	2143	1	0	261	1	-1	0	0	0	...	0	0	0	0
1	0	29	1	0	151	1	-1	0	0	0	...	0	0	1	0
2	0	2	1	1	76	1	-1	0	0	1	...	0	0	0	0
5	0	231	1	0	139	1	-1	0	0	1	...	0	0	0	0
6	0	447	1	1	217	1	-1	0	0	1	...	0	0	0	0

5 rows × 36 columns

In [72]:
```
df_clean.columns
```

Out[72]:
```
Index(['default', 'balance', 'housing', 'loan', 'duration', 'campaign',
       'pdays', 'previous', 'y', 'age_18_37', 'age_37_56', 'age_56_75',
       'age_75_95', 'month_T1', 'month_T2', 'month_T3', 'day_1_10',
       'day_11_20', 'day_21_31', 'job_admin.', 'job_blue-collar',
       'job_entrepreneur', 'job_housemaid', 'job_management', 'job_retired',
       'job_self-employed', 'job_services', 'job_student', 'job_technician',
       'job_unemployed', 'marital_divorced', 'marital_married',
       'marital_single', 'education_primary', 'education_secondary',
       'education_tertiary'],
      dtype='object')
```

2.8 Balance, duration, campaign, pdays, previous columns

In [73]:
```
#Perform normalisatin to numerical columns
from sklearn.preprocessing import StandardScaler

df_clean[['balance', 'duration', 'campaign',
          'pdays','previous']]=StandardScaler().fit_transform(df_clean[['balance', 'duration',
                                                              'campaign', 'pdays','previous']])
df_clean.head()
```

Out[73]:

	default	balance	housing	loan	duration	campaign	pdays	previous	y	age_18_37	...	job_services	job_student	job_technician	job_unemplo
0	0	0.259354	1	0	0.010368	-0.573827	-0.412311	-0.25073	0	0	...	0	0	0	
1	0	-0.435568	1	0	-0.415726	-0.573827	-0.412311	-0.25073	0	0	...	0	0	1	
2	0	-0.444443	1	1	-0.706245	-0.573827	-0.412311	-0.25073	0	1	...	0	0	0	
5	0	-0.369166	1	0	-0.462209	-0.573827	-0.412311	-0.25073	0	1	...	0	0	0	
6	0	-0.298161	1	1	-0.160070	-0.573827	-0.412311	-0.25073	0	1	...	0	0	0	

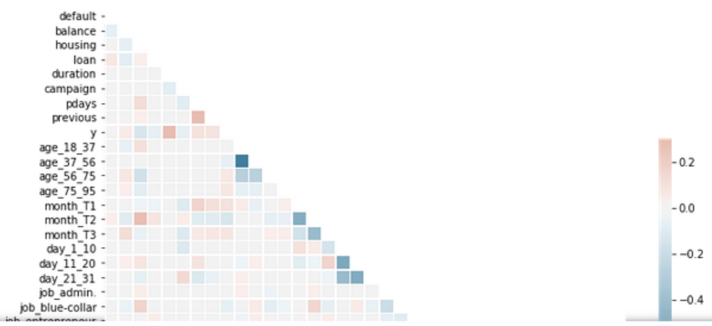
5 rows × 36 columns

2.9 Get only the most relevant variables

In [74]:
```
## Correlation matrix

import seaborn as sns

# Compute the correlation matrix
corr = df_clean.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
```

```
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x1a3b1cab5c8>



In [77]:
```
corr.y.apply(abs).sort_values(ascending=False)
```

Out[77]:
```
y                          1.000000
duration                   0.397393
housing                    0.138300
month_T2                   0.128929
pdays                      0.101446
month_T1                   0.098599
previous                   0.091764
job_retired                0.078686
```

In [75]:
```
top_variables=corr.y.apply(abs).sort_values(ascending=False)[:7].index
top_variables
```

Out[75]:
```
Index(['y', 'duration', 'housing', 'month_T2', 'pdays', 'month_T1',
       'previous'],
      dtype='object')
```

In [76]:
```
df_reduce=df_clean[top_variables]
df_reduce.head()
```

Out[76]:

	y	duration	housing	month_T2	pdays	month_T1	previous
0	0	0.010368	1	1	-0.412311	0	-0.25073
1	0	-0.415726	1	1	-0.412311	0	-0.25073
2	0	-0.706245	1	1	-0.412311	0	-0.25073
5	0	-0.462209	1	1	-0.412311	0	-0.25073
6	0	-0.160070	1	1	-0.412311	0	-0.25073

2.10 Apply Feature Reduction using Principal Components

In [78]:
```
from sklearn.decomposition import PCA

X1 = df_clean.drop(columns=['y'])
n = 15
pca = PCA(n_components= n)
principalComponents = pca.fit_transform(X1)
principalDf = pd.DataFrame(data = principalComponents)

pca.explained_variance_ratio_[:n].sum()

# 15 components explain 90% of the variance.
```

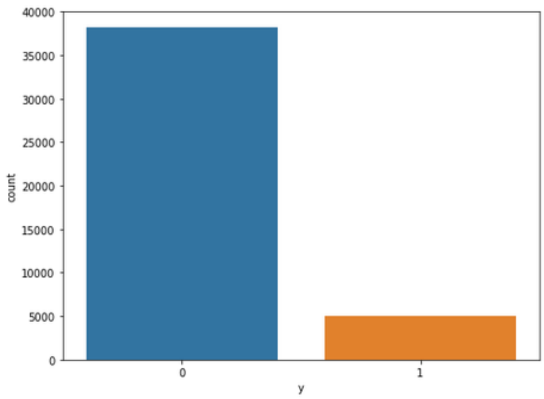
Out[78]: 0.900200653879968

2.11 Define final predictors and target variable

In [79]:
```
y = df_clean['y']
X1 = df_clean.drop(columns=['y'])
X2 = df_reduce.drop(columns=['y'])
X3 = principalDf
```

2.12 Verify Class Imbalance occurs

In [81]:
```
plt.rcParams["figure.figsize"] = (8, 6)
df_clean['y'].value_counts()
sns.countplot(x = 'y', data=df_clean)
plt.show()
```



3. Models deployment

In [172]:
```
# Function to be used for all models in order to simply the coding

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.utils import shuffle
from imblearn.over_sampling import SMOTE


def fit_classifier(model_name, X, y):
    '''takes the name of the model, the predictors, target variable, and
    prints the confusion matrix, accuracy score and classification report.
    An undersampling strategy is used after splitting the data.'''

    # split the dataset before undersampling
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2021, train_size = 0.8, shuffle = Tru


    # Apply undersampling to the training set
    # define the undersampling method
    undersample = NearMiss(version=3, n_neighbors=10, sampling_strategy=0.2)
    # transform the dataset
    X_under, y_under = undersample.fit_resample(X_train, y_train)
    X_train = X_under
    y_train = y_under

    #Training the model
    my_model = model_name # Train the model
    my_model.fit(X_train, y_train) # fit the model
    pred = my_model.predict(X_test) # Predict the response

    #Creating a confusion matrix
    conf_mat= metrics.confusion_matrix(y_test, pred)
    conf_mat_df=pd.DataFrame(conf_mat, index=['no','yes'], columns=['no','yes'] )
    print("----------- CONFUSION MATRIX ------------------------")
    print(conf_mat_df,'\n\n')
```

```python
    # Model Accuracy: how often is the classifier correct?
    print("----------- ACCURACY --------------------------------")
    print("Accuracy:",metrics.accuracy_score(y_test, pred),'\n\n')
    print("----------- CLASSIFICATION REPORT -------------------")
    print(metrics.classification_report(y_test, pred))


    return [round(metrics.precision_score(y_test, pred),3),
            round(metrics.recall_score(y_test, pred),3),
            round(metrics.f1_score(y_test, pred),3),
           round(metrics.accuracy_score(y_test, pred),3)]
```

In [205]:
```python
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, GradientBoostingClassifier


#Define the parameters of the model
logistic_reg = linear_model.LogisticRegression()
sup_vector_linear = svm.SVC(kernel = 'linear')
sup_vector_rbf = svm.SVC(kernel = 'rbf')
sup_vector_poly = svm.SVC(kernel = 'poly')
decision_tree=DecisionTreeClassifier()
ran_classifier = RandomForestClassifier()
```

3.1 Logistic regression

3.1.1 All variables

In [206]:
```python
metrics_model = fit_classifier(logistic_reg, X1, y)
models_measurements=pd.DataFrame([['Logistic regression', 'All variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7398  217
yes   756  268


----------- ACCURACY --------------------------------
Accuracy: 0.8873712235212409


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      7615
           1       0.55      0.26      0.36      1024

    accuracy                           0.89      8639
   macro avg       0.73      0.62      0.65      8639
weighted avg       0.87      0.89      0.87      8639
```

3.1.2 Most relevant variables

In [207]:
```python
metrics_model = fit_classifier(logistic_reg, X2, y)
models_measurements=models_measurements.append([['Logistic regression', 'Most relevant variables']+metrics_model
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7461  154
yes   795  229


----------- ACCURACY --------------------------------
Accuracy: 0.8901493228382915


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      7615
           1       0.60      0.22      0.33      1024

    accuracy                           0.89      8639
   macro avg       0.75      0.60      0.63      8639
weighted avg       0.87      0.89      0.87      8639
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

3.1.3 PCA variables

In [208]:
```
metrics_model = fit_classifier(logistic_reg, X3, y)
models_measurements=models_measurements.append([['Logistic regression', 'PCA variables']+metrics_model])
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7446  169
yes   790  234

----------- ACCURACY --------------------------------
Accuracy: 0.8889917814561871


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      7615
           1       0.58      0.23      0.33      1024

    accuracy                           0.89      8639
   macro avg       0.74      0.60      0.63      8639
weighted avg       0.87      0.89      0.87      8639
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

3.2 Support Vector Machine - linear

3.2.1 All variables

In [209]:
```
metrics_model = fit_classifier(sup_vector_linear, X1, y)
models_measurements=models_measurements.append([['SVM linear', 'All variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7386  229
yes   836  188

----------- ACCURACY --------------------------------
Accuracy: 0.8767218428058803


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      7615
           1       0.45      0.18      0.26      1024

    accuracy                           0.88      8639
   macro avg       0.67      0.58      0.60      8639
weighted avg       0.85      0.88      0.85      8639
```

3.2.2 Most relevant variables

In [210]:
```
metrics_model = fit_classifier(sup_vector_linear, X2, y)
models_measurements=models_measurements.append([['SVM linear', 'Most relevant variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7555   60
yes   932   92


----------- ACCURACY --------------------------------
Accuracy: 0.8851718948952425


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      7615
           1       0.61      0.09      0.16      1024

    accuracy                           0.89      8639
   macro avg       0.75      0.54      0.55      8639
weighted avg       0.86      0.89      0.85      8639
```

3.2.3 PCA variables

In [211]:
```
metrics_model = fit_classifier(sup_vector_linear, X3, y)
models_measurements=models_measurements.append([['SVM linear', 'PCA variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7538   77
yes   895  129


----------- ACCURACY --------------------------------
Accuracy: 0.8874869776594513


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      7615
           1       0.63      0.13      0.21      1024

    accuracy                           0.89      8639
   macro avg       0.76      0.56      0.57      8639
weighted avg       0.86      0.89      0.85      8639
```

3.3 Support Vector Machine - rbf

3.3.1 All variables

In [212]:
```
metrics_model = fit_classifier(sup_vector_rbf, X1, y)
models_measurements=models_measurements.append([['SVM rbf', 'All variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7385  230
yes   662  362

----------- ACCURACY --------------------------------
Accuracy: 0.8967473087162866


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      7615
           1       0.61      0.35      0.45      1024

    accuracy                           0.90      8639
   macro avg       0.76      0.66      0.70      8639
weighted avg       0.88      0.90      0.88      8639
```

3.3.2 Most relevant variables

In [213]:
```
metrics_model = fit_classifier(sup_vector_rbf, X2, y)
models_measurements=models_measurements.append([['SVM rbf', 'Most relevant variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7412  203
yes   714  310

----------- ACCURACY --------------------------------
Accuracy: 0.8938534552610256


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      7615
           1       0.60      0.30      0.40      1024

    accuracy                           0.89      8639
   macro avg       0.76      0.64      0.67      8639
weighted avg       0.88      0.89      0.88      8639
```

3.3.3 PCA variables

In [214]:
```
metrics_model = fit_classifier(sup_vector_rbf, X3, y)
models_measurements=models_measurements.append([['SVM rbf', 'PCA variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_ne
armiss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples av
ailable. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7408  207
yes   697  327


----------- ACCURACY --------------------------------
Accuracy: 0.8953582590577613


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      7615
```

3.4 Support Vector Machine - poly ¶

3.4.1 All variables

In [215]:
```
metrics_model = fit_classifier(sup_vector_poly, X1, y)
models_measurements=models_measurements.append([['SVM poly', 'All variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7397  218
yes   723  301


----------- ACCURACY --------------------------------
Accuracy: 0.891075355943975


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      7615
           1       0.58      0.29      0.39      1024

    accuracy                           0.89      8639
   macro avg       0.75      0.63      0.67      8639
weighted avg       0.87      0.89      0.87      8639
```

3.4.2 Most relevant variables

In [216]:
```
metrics_model = fit_classifier(sup_vector_poly, X2, y)
models_measurements=models_measurements.append([['SVM poly', 'Most relevant variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_ne
armiss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples av
ailable. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7562   53
yes   934   90


----------- ACCURACY --------------------------------
Accuracy: 0.8857506655862947


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      7615
```

3.4.3 PCA variables

In [217]:
```
metrics_model = fit_classifier(sup_vector_poly, X3, y)
models_measurements=models_measurements.append([['SVM poly', 'PCA variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7485  130
yes   844  180


----------- ACCURACY --------------------------------
Accuracy: 0.8872554693830305


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      7615
           1       0.58      0.18      0.27      1024

    accuracy                           0.89      8639
   macro avg       0.74      0.58      0.60      8639
weighted avg       0.86      0.89      0.86      8639
```

3.5 Decision tree classifier

3.5.1 All variables

In [218]:
```
metrics_model = fit_classifier(decision_tree,X1,y)
models_measurements=models_measurements.append([['Decision tree', 'All variables']+metrics_model])
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   6124  1491
yes   517   507


----------- ACCURACY --------------------------------
Accuracy: 0.7675656904734344


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.92      0.80      0.86      7615
           1       0.25      0.50      0.34      1024

    accuracy                           0.77      8639
   macro avg       0.59      0.65      0.60      8639
weighted avg       0.84      0.77      0.80      8639
```

click to expand output; double click to hide output

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

3.5.2 Most relevant variables

In [219]:
```
metrics_model = fit_classifier(decision_tree,X2,y)
models_measurements=models_measurements.append([['Decision tree', 'Most relevant variables']+metrics_model])
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   6983  632
yes   648  376


----------- ACCURACY --------------------------------
Accuracy: 0.8518347030906355


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      7615
           1       0.37      0.37      0.37      1024

    accuracy                           0.85      8639
   macro avg       0.64      0.64      0.64      8639
weighted avg       0.85      0.85      0.85      8639
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

3.5.3 PCA variables

In [220]:
```
metrics_model = fit_classifier(model,X3,y)
models_measurements=models_measurements.append([['Decision tree', 'PCA variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no   yes
no   5852  1763
yes   533   491


----------- ACCURACY --------------------------------
Accuracy: 0.7342284986688274


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.92      0.77      0.84      7615
           1       0.22      0.48      0.30      1024

    accuracy                           0.73      8639
   macro avg       0.57      0.62      0.57      8639
weighted avg       0.83      0.73      0.77      8639
```

3.6 Random Forest

3.6.1 All Variables

In [221]:
```
metrics_model = fit_classifier(ran_classifier,X1,y)
models_measurements=models_measurements.append([['Random forest', 'All variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7232  383
yes   528  496


----------- ACCURACY --------------------------------
Accuracy: 0.8945479800902882


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      7615
           1       0.56      0.48      0.52      1024

    accuracy                           0.89      8639
   macro avg       0.75      0.72      0.73      8639
weighted avg       0.89      0.89      0.89      8639
```

3.6.2 Relevant Variable

In [222]:
```
metrics_model = fit_classifier(ran_classifier,X2,y)
models_measurements=models_measurements.append([['Random forest', 'Most relevant variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7107  508
yes   616  408


----------- ACCURACY --------------------------------
Accuracy: 0.8698923486514643


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.92      0.93      0.93      7615
           1       0.45      0.40      0.42      1024

    accuracy                           0.87      8639
   macro avg       0.68      0.67      0.67      8639
weighted avg       0.86      0.87      0.87      8639
```

3.6.3 PCA variables

In [223]:
```
metrics_model = fit_classifier(ran_classifier,X3,y)
models_measurements=models_measurements.append([['Random forest', 'PCA variables']+metrics_model])
```

```
C:\Users\DELL\AppData\Roaming\Python\Python37\site-packages\imblearn\under_sampling\_prototype_selection\_near
miss.py:176: UserWarning: The number of the samples to be selected is larger than the number of samples availa
ble. The balancing ratio cannot be ensure and all samples will be returned.
  "The number of the samples to be selected is larger"
```

```
----------- CONFUSION MATRIX ------------------------
       no  yes
no   7252  363
yes   607  417


----------- ACCURACY --------------------------------
Accuracy: 0.8877184859358722


----------- CLASSIFICATION REPORT -------------------
              precision    recall  f1-score   support

           0       0.92      0.95      0.94      7615
           1       0.53      0.41      0.46      1024

    accuracy                           0.89      8639
   macro avg       0.73      0.68      0.70      8639
weighted avg       0.88      0.89      0.88      8639
```

In [224]:
```
models_measurements.columns=['Model','Variables','Precision','Recall','F1 Score','Accuracy']
```

In [229]:
```
#Measurements sorted by F1 score
models_measurements=models_measurements.sort_values(by=['F1 Score'],ascending = False).reset_index(drop=True)
models_measurements
```

Out[229]:

	Model	Variables	Precision	Recall	F1 Score	Accuracy
0	Random forest	All variables	0.564	0.484	0.521	0.895
1	Random forest	PCA variables	0.535	0.407	0.462	0.888
2	SVM rbf	All variables	0.611	0.354	0.448	0.897
3	Random forest	Most relevant variables	0.445	0.398	0.421	0.870
4	SVM rbf	PCA variables	0.612	0.319	0.420	0.895
5	SVM rbf	Most relevant variables	0.604	0.303	0.403	0.894
6	SVM poly	All variables	0.580	0.294	0.390	0.891
7	Decision tree	Most relevant variables	0.373	0.367	0.370	0.852
8	Logistic regression	All variables	0.553	0.262	0.355	0.887
9	Decision tree	All variables	0.254	0.495	0.336	0.768
10	Logistic regression	PCA variables	0.581	0.229	0.328	0.889
11	Logistic regression	Most relevant variables	0.598	0.224	0.326	0.890
12	Decision tree	PCA variables	0.218	0.479	0.300	0.734
13	SVM poly	PCA variables	0.581	0.176	0.270	0.887
14	SVM linear	All variables	0.451	0.184	0.261	0.877
15	SVM linear	PCA variables	0.626	0.126	0.210	0.887
16	SVM linear	Most relevant variables	0.605	0.090	0.156	0.885
17	SVM poly	Most relevant variables	0.629	0.088	0.154	0.886

In [232]:
```
#Sorted by accuracy
models_measurements.sort_values(by=['Accuracy'],ascending = False)
```

Out[232]:

	Model	Variables	Precision	Recall	F1 Score	Accuracy
2	SVM rbf	All variables	0.611	0.354	0.448	0.897
0	Random forest	All variables	0.564	0.484	0.521	0.895
4	SVM rbf	PCA variables	0.612	0.319	0.420	0.895
5	SVM rbf	Most relevant variables	0.604	0.303	0.403	0.894
6	SVM poly	All variables	0.580	0.294	0.390	0.891
11	Logistic regression	Most relevant variables	0.598	0.224	0.326	0.890
10	Logistic regression	PCA variables	0.581	0.229	0.328	0.889
1	Random forest	PCA variables	0.535	0.407	0.462	0.888
15	SVM linear	PCA variables	0.626	0.126	0.210	0.887
13	SVM poly	PCA variables	0.581	0.176	0.270	0.887
8	Logistic regression	All variables	0.553	0.262	0.355	0.887
17	SVM poly	Most relevant variables	0.629	0.088	0.154	0.886
16	SVM linear	Most relevant variables	0.605	0.090	0.156	0.885
14	SVM linear	All variables	0.451	0.184	0.261	0.877
3	Random forest	Most relevant variables	0.445	0.398	0.421	0.870
7	Decision tree	Most relevant variables	0.373	0.367	0.370	0.852
9	Decision tree	All variables	0.254	0.495	0.336	0.768
12	Decision tree	PCA variables	0.218	0.479	0.300	0.734

Appendix B.
R program.

```
# --------------- 1. Install Libraries ---------------


#install.packages("readr")
#install.packages("dplyr")
#install.packages("plyr")
#install.packages("e1071")
#install.packages("rpart")
#install.packages("ROSE")
#install.packages("caret")


# --------------- 2. Loading Libraries ---------------

library(readr)
library(plyr)
library(dplyr)
library(e1071)
library(rpart)
library(ROSE)
library(caret)

# --------------- 3. Reading the data ---------------

# NOTE: DATA WAS ALREADY SPLITTED IN PYTHON, AND VARIABLES ARE ALREADY NORMALIZED.

#Set working Directory
#Julio's computer:
#setwd("C:/Users/DELL/Griffith University/7031ICT Applied Data Mining Group Assessment - General/R files")

train <- read_csv("training_data.csv")
test <- read_csv("testing_data.csv")

#View the column names. Remove the first column. In Gabriela's computer, the column name is 'X1'.
#However, in Julio's computer the column name is '...1'
colnames(train)
colnames(test)

train$y <- as.factor(train$y)
test$y <- as.factor(test$y)

# necessary procedure in order to use the ovun.sample() function.
train <- data.frame(train)
test <- data.frame(test)


# --------------- 4. Undersampling the training set ---------------

# Source -> https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/
n_yes <- length(which(train$y == 1)) # finding the number of rows where y = yes

df_balanced_under <- ovun.sample(y ~ ., data = train, method = "under", N = n_yes*2, seed = 2021)$data
table(df_balanced_under$y)


# --------------- 5. Building Logistic Regression model ---------------

control <- trainControl(method="cv", number=10)
metric <- "Accuracy"

#Train model
set.seed(7)
logreg <- train(y~., data=df_balanced_under, method="lda", metric=metric, trControl=control)
y_pred <- predict(logreg, newdata = test)

#Confusion matrix testing set
cm = confusionMatrix(y_pred,test$y,positive="1")
cm

#Calculate precision
precision <- posPredValue(y_pred, test$y,positive="1")
precision
```

```r
#Calcualte recall
recall <- sensitivity(y_pred, test$y,positive="1")
recall

#Calculate F1 score
F1 <- (2 * precision * recall) / (precision + recall)
F1

#Calculate accuracy
accuracy = cm$overall[['Accuracy']]
accuracy

logreg_val= c("Logistic regression","All variables",precision,recall,F1,accuracy)
logreg_val

# --------------- 6. Building SVM model ---------------

set.seed(7)
svm <- train(y~., data=df_balanced_under, method="svmRadial", metric=metric, trControl=control)
y_pred <- predict(svm, newdata = test)

#Confusion matrix testing set
cm = confusionMatrix(y_pred,test$y,positive="1")
cm

#Calculate precision
precision <- posPredValue(y_pred, test$y,positive="1")
precision

#Calcualte recall
recall <- sensitivity(y_pred, test$y,positive="1")
recall

#Calculate F1 score
F1 <- (2 * precision * recall) / (precision + recall)
F1

#Calculate accuracy
accuracy = cm$overall[['Accuracy']]
accuracy

svm_val= c("SVM - radial","All variables",precision,recall,F1,accuracy)
svm_val


# --------------- 7. Building Random Forest model ---------------

set.seed(7)
rf <- train(y~., data=df_balanced_under, method="rf", metric=metric, trControl=control)
y_pred <- predict(rf, newdata = test)

#Confusion matrix testing set
cm = confusionMatrix(y_pred,test$y,positive="1")
cm

#Calculate precision
precision <- posPredValue(y_pred, test$y,positive="1")
precision

#Calcualte recall
recall <- sensitivity(y_pred, test$y,positive="1")
recall

#Calculate F1 score
F1 <- (2 * precision * recall) / (precision + recall)
F1

#Calculate accuracy
accuracy = cm$overall[['Accuracy']]
accuracy

rf_val= c("Random forest","All variables",precision,recall,F1,accuracy)
rf_val
```

```
# --------------- 8. Building Decision Tree model ---------------

set.seed(7)
cart <- train(y~., data=df_balanced_under, method="rpart", metric=metric, trControl=control)
y_pred <- predict(rf, newdata = test)

#Confusion matrix testing set
cm = confusionMatrix(y_pred,test$y,positive="1")
cm

#Calculate precision
precision <- posPredValue(y_pred, test$y,positive="1")
precision

#Calcualte recall
recall <- sensitivity(y_pred, test$y,positive="1")
recall

#Calculate F1 score
F1 <- (2 * precision * recall) / (precision + recall)
F1

#Calculate accuracy
accuracy = cm$overall[['Accuracy']]
accuracy

cart_val= c("Decision tree","All variables",precision,recall,F1,accuracy)
cart_val


# --------------- 9. Building Decision Tree model ---------------

titles=c("Model","Variable","Precision","Recall","F1 Score","Accuracy")
View(data.frame(titles,logreg_val,svm_val,rf_val,cart_val))
```