Bank Marketing Data Classification

Firas Bou Karroum[1[JCOHWM]]

[1] Eotvos Lorand University ELTE Budapest, Hungary

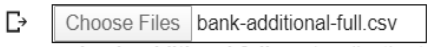
Abstract. This report shows sequentially the application of a set of Data Science techniques to gain insights from the Direct Marketing campaign of a Portuguese Banking Institution.
The data set has 45,212 instances and 20 features (input variables), which include the respondents' level of education, social status, month and day, results of a previous marketing campaign, and some macroeconomic indicators.
The predictor variable ("y") states the outcome of the marketing campaign — whether the respondent subscribed for a deposit ("yes") or no ("no").
We have 21 variables (10 numeric and 11 string), 10 categorical and 10 continuous variables. The data is almost clean and hence there are indeed no missing values.
The Portuguese Bank that initiated the telemarketing campaign, (that provided the data examined by this document), contacted potential savings account depositors for a 5-year period, 2008 through 2013. This data therefore reflects the influence of the financial crisis of 2008.). In this report I will go through the data analysis, categorizing data (graphically) with the effect of each feature on the variable y, treating imbalanced data, applying various techniques and models for training and prediction and study their outcome, and finally the conclusion of our analysis with the decision of best model and technique.

Keywords: Categorical and Continuous Variables, Random Undersampling, Random Oversampling, SMOTE, Decision Tree, Random Forest, Support Vector Machin (SVM), Logistic Regression.

1 Environment

As a first step I downloaded my data set as per requirement from UCI (Machine Learning Repository) and used the data set "bank-additional-full.csv", opened it on local machine and analyze its content. For the environment to apply my analysis and the techniques and experiments using different models I picked "Colaboratory" or "Colab" from google research that is based on Jupiter but with the feature to run it online without setting local environment on my machine, and so I created my notebook and started my work using Python3.6. I used "google.colab" library to upload my file (csv data set) to the working notebook. Then I used read_csv from pandas library applied on the output of uploaded files that was my csv file in previous step. Hence my data is ready and I am ready to start working on it.

```
[ ] from google.colab import files
    uploaded = files.upload()
```



- **bank-additional-full.csv**(application/vnd.ms-excel) - 5834924 bytes, last modified: 3/26/2014 - 100% done

```
Saving bank-additional-full.csv to bank-additional-full.csv
```

```
[ ] import io
    data = pd.read_csv(io.BytesIO(uploaded['bank-additional-full.csv']),sep=';')
    data.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	em
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0	nonexistent	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0	nonexistent	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0	nonexistent	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0	nonexistent	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	0	nonexistent	

2 Data Analysis and Graphics

Our data set is based on phone calls with the bank clients where there are clients that been contacted more than once to make sure if the bank term deposit (our target) has been subscribed or not yet which will help us a lot in our analysis. The data has 20 predictable features (columns) and 1 output. The last column “y” is our term output and has value of Boolean yes or no. Customers who received these call might not be unique and might been contacted several times.

The Bank Marketing dataset contains numeric variables, (useful for predictive analytics and machine learning), categorical variables,

discrete and continuous variables. 19 of the 20 explanatory variables initially appear useful for prediction of future Term Deposit subscriptions. The data set used has 4 types of data as per the resource UCI machine learning repository:

1) Customer data: age, job, marital status, education, default, housing and loan.

2) Telemarketing data: contact, month, day of the week, and duration.

3) Socioeconomic data: employment variation rate, consumer price index, consumer confidence index, 3 month Euribor rate, and number of employees.

4) Other data: campaign, past days, previous, and past outcome.

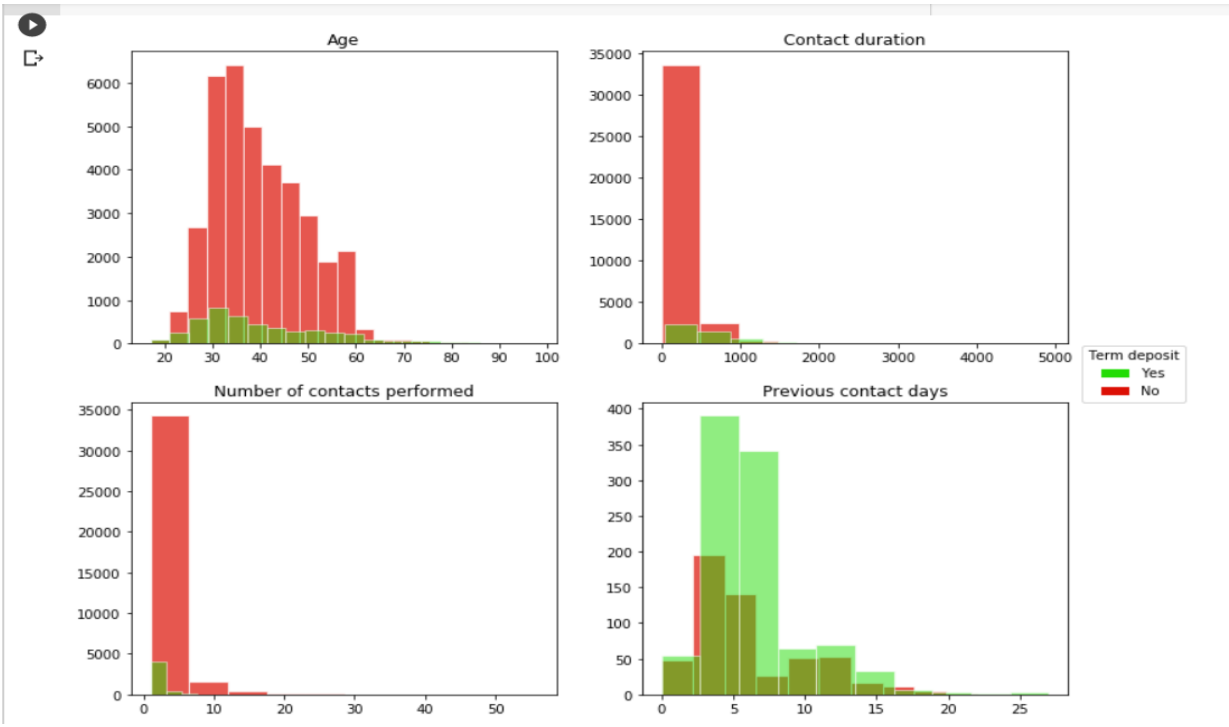
The columns (features with their definitions):
age - Age of the client- (numeric)
job - Client's occupation - (categorical)
(admin, bluecollar, entrepreneur, housemaid, management, retired, selfemployed, services, student, technician, unemployed, unknown)
marital - Client's marital status - (categorical)
(divorced, married, single, unknown, note: divorced means divorced or widowed)
education - Client's education level - (categorical)
(basic.4y, basic.6y, basic.9y, high.school, illiterate, professional.course, university.degree, unknown)
default - Indicates if the client has credit in default - (categorical)
(no, yes, unknown)
housing - Does the client as a housing loan? - (categorical)
(no, yes, unknown)
loan - Does the client as a personal loan? - (categorical)
(no, yes, unknown')
contact - Type of communication contact - (categorical)
(cellular, telephone)
month - Month of last contact with client - (categorical)
(January - December)
day_of_week - Day of last contact with client - (categorical)
(Monday - Friday)
duration - Duration of last contact with client, in seconds - (numeric)
For benchmark purposes only, and not reliable for predictive modeling
campaign - Number of client contacts during this campaign - (numeric)
(includes last contact)
pdays - Number of days from last contacted from a previous campaign - (numeric)
(999 means client was not previously contacted)
previous - Number of client contacts performed before this campaign - (numeric)
poutcome - Previous marketing campaign outcome - (categorical)
(failure, nonexistent , success)
emp.var.rate - Quarterly employment variation rate - (numeric)
cons.price.idx - Monthly consumer price index - (numeric)
cons.conf.idx - Monthly consumer confidence index - (numeric)
euribor3m - Daily euribor 3 month rate - (numeric)
nr.**employed** - Quarterly number of employees - (numeric)
Output variable (desired target) - Term Deposit - subscription verified
(binary: 'yes','no')

Now I will show the data graphics for each of the features divided into two groups: Continuous and Categorical. For the 10 continuous variables I will use unstacked histograms and for the categorical variables I will use the barplots with respect to the term deposit of subscription that will be "yes" or "no" and colored "green for yes and red for no.
Here are the 10 continuous variables unstacked histograms with the code snippets:

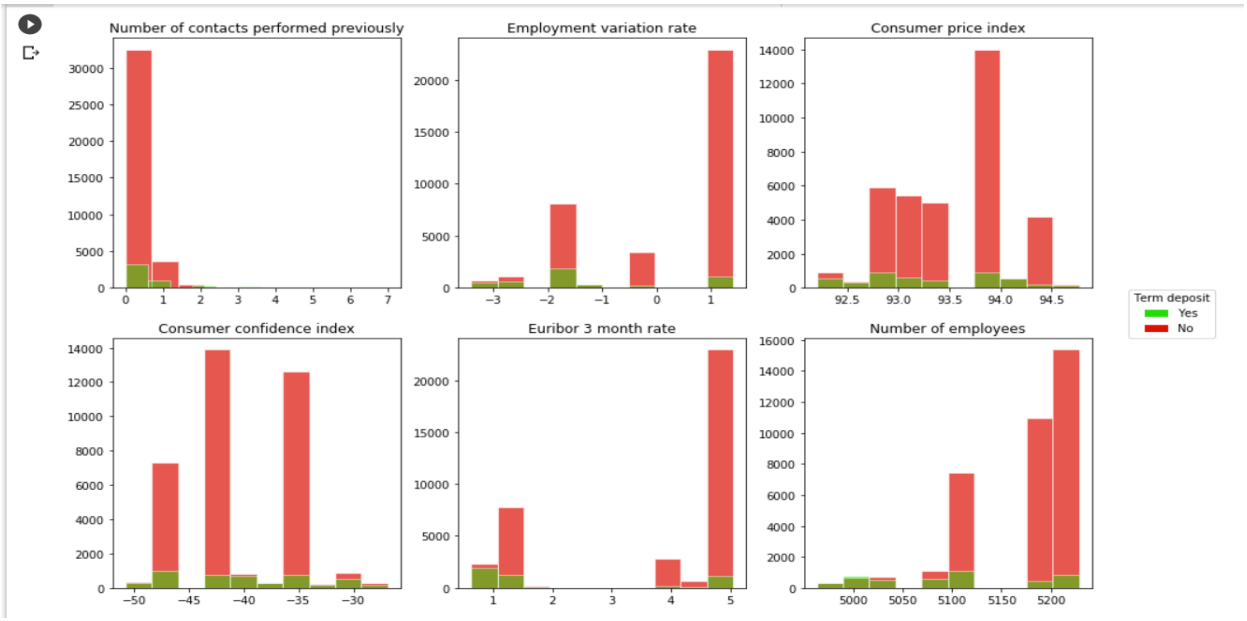
4

```
fig, ax = plt.subplots(2, 2, figsize=(12,10))

ax[0, 0].hist(data2['age'],color = '#dc1005',alpha=0.7,bins=20, edgecolor='white')
ax[0, 0].hist(data1['age'],color='#22dc05',alpha=0.5,bins=20, edgecolor='white')
ax[0, 0].title.set_text('Age')
ax[0, 1].hist(data2['duration'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[0, 1].hist(data1['duration'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[0, 1].title.set_text('Contact duration')
ax[1, 0].hist(data2['campaign'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[1, 0].hist(data1['campaign'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[1, 0].title.set_text('Number of contacts performed')
ax[1, 1].hist(data2[data2['pdays'] != 999]['pdays'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[1, 1].hist(data1[data1['pdays'] != 999]['pdays'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[1, 1].title.set_text('Previous contact days')
plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")
plt.show()
```



```
fig, ax = plt.subplots(2, 3, figsize=(15,10))
ax[0, 0].hist(data2['previous'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[0, 0].hist(data1['previous'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[0, 0].title.set_text('Number of contacts performed previously')
ax[0, 1].hist(data2['emp.var.rate'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[0, 1].hist(data1['emp.var.rate'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[0, 1].title.set_text('Employment variation rate')
ax[0, 2].hist(data2['cons.price.idx'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[0, 2].hist(data1['cons.price.idx'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[0, 2].title.set_text('Consumer price index')
ax[1, 0].hist(data2['cons.conf.idx'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[1, 0].hist(data1['cons.conf.idx'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[1, 0].title.set_text('Consumer confidence index')
ax[1, 1].hist(data2['euribor3m'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[1, 1].hist(data1['euribor3m'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[1, 1].title.set_text('Euribor 3 month rate')
ax[1, 2].hist(data2['nr.employed'],color = '#dc1005',alpha=0.7, edgecolor='white')
ax[1, 2].hist(data1['nr.employed'],color='#22dc05',alpha=0.5, edgecolor='white')
ax[1, 2].title.set_text('Number of employees')
plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")
plt.show()
```

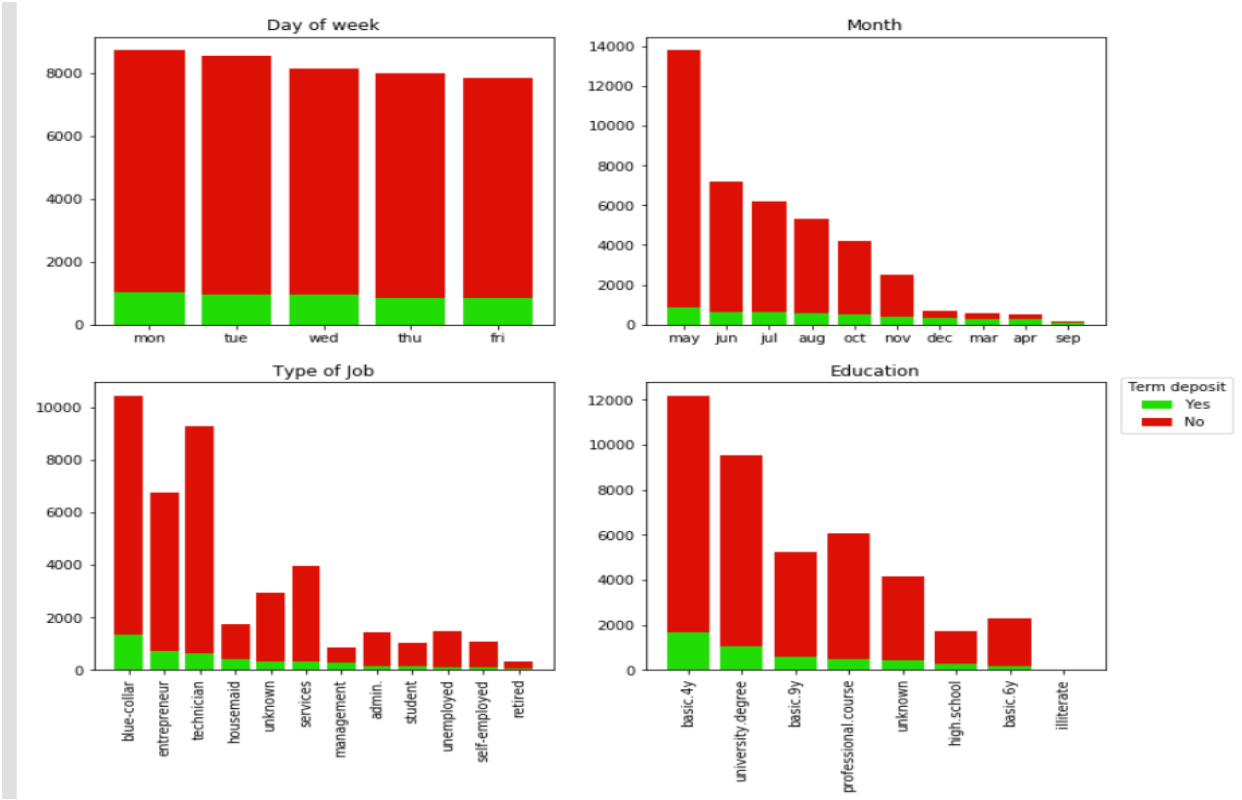


All the histograms above except for ‘previous contact days’ show that the proportion of “no” is much higher the proportion of “yes”. For ‘previous contact details’ the proportion of yes is higher.
Now for the categorical variables (features) I will do the same analysis:

6

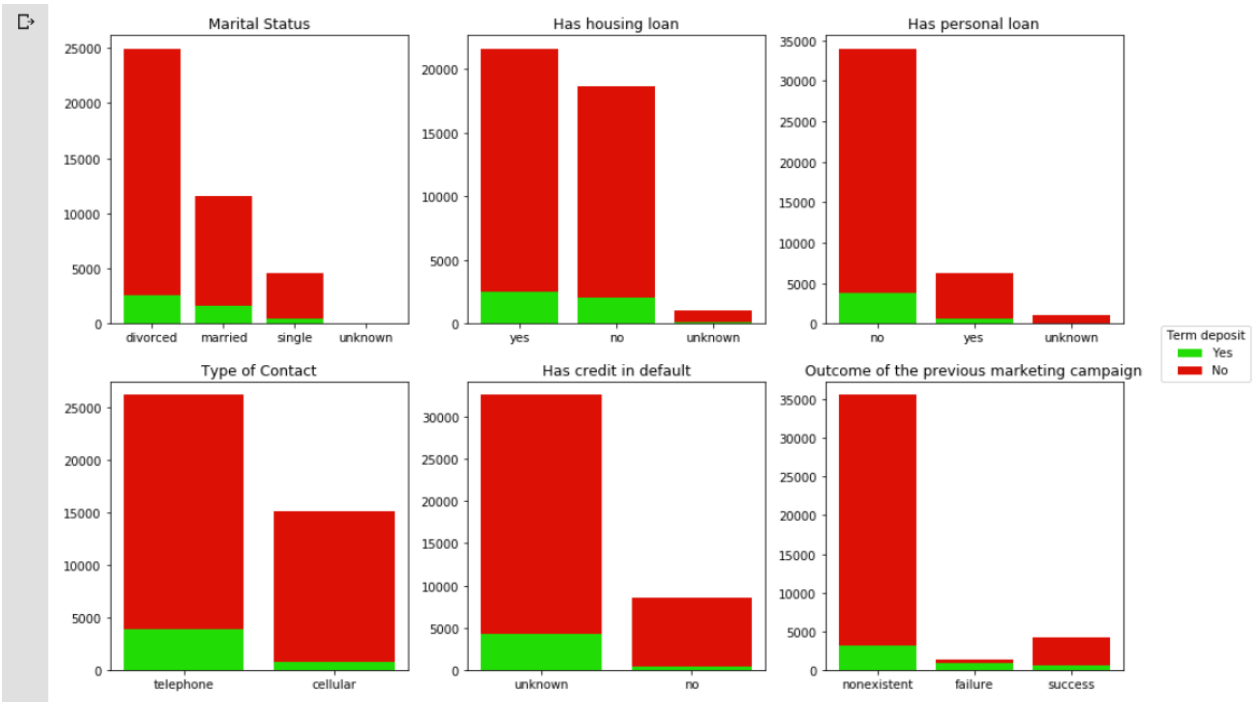
```
fig, ax = plt.subplots(2, 2, figsize=(12,10))

b1 = ax[0, 0].bar(data1['day_of_week'].unique(),height = data1['day_of_week'].value_counts(),color='#22dc05')
b2 = ax[0, 0].bar(data2['day_of_week'].unique(),height = data2['day_of_week'].value_counts(),bottom = data1['day_of_week'].value_counts(),color = '#dc1005')
ax[0, 0].title.set_text('Day of week')
#ax[0, 0].legend((b1[0], b2[0]), ('Yes', 'No'))
ax[0, 1].bar(data1['month'].unique(),height = data1['month'].value_counts(),color='#22dc05')
ax[0, 1].bar(data2['month'].unique(),height = data2['month'].value_counts(),bottom = data1['month'].value_counts(),color = '#dc1005')
ax[0, 1].title.set_text('Month')
ax[1, 0].bar(data1['job'].unique(),height = data1['job'].value_counts(),color='#22dc05')
ax[1, 0].bar(data1['job'].unique(),height = data2['job'].value_counts()[data1['job'].value_counts().index],bottom = data1['job'].value_counts(),color = '#dc1005')
ax[1, 0].title.set_text('Type of Job')
ax[1, 0].tick_params(axis='x',rotation=90)
ax[1, 1].bar(data1['education'].unique(),height = data1['education'].value_counts(),color='#22dc05') #row=0, col=1
ax[1, 1].bar(data1['education'].unique(),height = data2['education'].value_counts()[data1['education'].value_counts().index],bottom = data1['education'].value_counts(),
        color = '#dc1005')
ax[1, 1].title.set_text('Education')
ax[1, 1].tick_params(axis='x',rotation=90)
#ax[0, 1].xticks(rotation=90)
plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")
plt.show()
```



```
[ ] fig, ax = plt.subplots(2, 3, figsize=(15,10))

    b1 = ax[0, 0].bar(data1['marital'].unique(),height = data1['marital'].value_counts(),color='#22dc05')
    b2 = ax[0, 0].bar(data1['marital'].unique(),height = data2['marital'].value_counts()[data1['marital'].value_counts().index],bottom = data1['marital'].value_counts(),color = '#dc1005')
    ax[0, 0].title.set_text('Marital Status')
    #ax[0, 0].legend((b1[0], b2[0]), ('Yes', 'No'))
    ax[0, 1].bar(data1['housing'].unique(),height = data1['housing'].value_counts(),color='#22dc05')
    ax[0, 1].bar(data1['housing'].unique(),height = data2['housing'].value_counts()[data1['housing'].value_counts().index],bottom = data1['housing'].value_counts(),color = '#dc1005')
    ax[0, 1].title.set_text('Has housing loan')
    ax[0, 2].bar(data1['loan'].unique(),height = data1['loan'].value_counts(),color='#22dc05')
    ax[0, 2].bar(data1['loan'].unique(),height = data2['loan'].value_counts()[data1['loan'].value_counts().index],bottom = data1['loan'].value_counts(),color = '#dc1005')
    ax[0, 2].title.set_text('Has personal loan')
    ax[1, 0].bar(data1['contact'].unique(),height = data1['contact'].value_counts(),color='#22dc05')
    ax[1, 0].bar(data1['contact'].unique(),height = data2['contact'].value_counts()[data1['contact'].value_counts().index],bottom = data1['contact'].value_counts(),color = '#dc1005')
    ax[1, 0].title.set_text('Type of Contact')
    ax[1, 1].bar(data1['default'].unique(),height = data1['default'].value_counts(),color='#22dc05')
    ax[1, 1].bar(data1['default'].unique(),height = data2['default'].value_counts()[data1['default'].value_counts().index],bottom = data1['default'].value_counts(),color = '#dc1005')
    ax[1, 1].title.set_text('Has credit in default')
    ax[1, 2].bar(data1['poutcome'].unique(),height = data1['poutcome'].value_counts(),color='#22dc05')
    ax[1, 2].bar(data1['poutcome'].unique(),height = data2['poutcome'].value_counts()[data1['poutcome'].value_counts().index],bottom = data1['poutcome'].value_counts(),color = '#dc1005')
    ax[1, 2].title.set_text('Outcome of the previous marketing campaign')
    plt.figlegend((b1[0], b2[0]), ('Yes', 'No'),loc="right",title = "Term deposit")
    plt.show()
```



All the categories in each categorical variable has 'yes' very low compared to 'no' for term deposit subscription. However, it is clear from the observation that 'Outcome of the previous marketing campaign' variable, failure category has more 'yes' compared to 'no'.

This means, if the previous campaign calls failed to get the customers to subscribe, repeated calls got customers to subscribe to term deposit.

As we noticed that the ranking of “No” is much more then “yes” in our previous observations and hence the data is imbalanced. We need to find a way to treat the imbalanced data.

```
predictors = data.iloc[:,0:20]
predictors = predictors.drop(['pdays'],axis=1)
y = data.iloc[:,20]
X = pd.get_dummies(predictors)
```

```
[ ]  y.value_counts()

     no     36548
     yes     4640
     Name: y, dtype: int64
```

3 Imbalanced data treatment (methods)

3.1 Random Undersampling

We randomly sample the “no” category that is the majority with the “yes” category that is the minority to let the numbers equal the size of “yes” that is the minority and discard the rest of “no” occurrences using the following:

```
[2]  rus = RandomUnderSampler(random_state=0)
     X_Usampled, y_Usampled = rus.fit_resample(X, y)
     pd.Series(y_Usampled).value_counts()
```

No 4640
Yes 4640
Dtype: Int64

3.2 Random Oversampling

Here we do the opposite of previous method and that is by sampling the “yes” majority with the size of the :no” majority and so the “yes” minority will be repeated too much.

```
ros = RandomOverSampler(random_state=0)
X_Osampled, y_Osampled = ros.fit_resample(X, y)
pd.Series(y_Osampled).value_counts()
```

```
no     36548
yes    36548
dtype: int64
```

3.3 Synthetic Minority Oversampling Technique (SMOTE)

Instead of randomly repeating minority 'yes' category in the previous method, new entries are synthetically created maintaining the convexity of minority entry space. Minority category will again match the majority category samples.

```
[ ]  sm = SMOTE(random_state=0)
     X_SMOTE, y_SMOTE = sm.fit_resample(X, y)
     pd.Series(y_SMOTE).value_counts()
```

```
no     36548
yes    36548
dtype: int64
```

4 Applying Machine Learning Models

In this section I will cover the models that best fit this data set to solve it and best predict the term deposit that is required for data classification

4.1 Decision Tree Model

We can use decision trees for issues where we have continuous but also categorical input and target features like our case here.
But this model faces the problem of bad precision score and bad recall score for imbalanced data on our data set here according to the following:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
tree = DecisionTreeClassifier(criterion="entropy", max_depth=7)
model = tree.fit(X_train,y_train)
y_pred = model.predict(X_test)
y_test = label_binarize(y_test,classes=['no','yes'])
y_pred = label_binarize(y_pred,classes=['no','yes'])
print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
```

```
Precision:  0.64 Recall:  0.56
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    tree = DecisionTreeClassifier(criterion="entropy", max_depth=7)
    X_SMOTE, y_SMOTE = sm.fit_resample(X_train, y_train)
    model = tree.fit(X_SMOTE,y_SMOTE)
    y_pred = model.predict(X_test)
    y_test = label_binarize(y_test,classes=['no','yes'])
    y_pred = label_binarize(y_pred,classes=['no','yes'])
    print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
```

```
Precision:  0.48 Recall:  0.83
```

4.2 Random Forests Model

Random forest is an ensemble technique which reduces the variance in the classification technique. But this classification lessens the bias that is already in the data. Random forest performs badly in terms of precision and recall when applied on imbalanced data.

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    forest = RandomForestClassifier(n_estimators= 1000,criterion="gini", max_depth=5,min_samples_split = 0.4,min_samples_leaf=1, class_weight="balanced")
    model = forest.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    pd.Series(y_pred).value_counts()
    y_test = label_binarize(y_test,classes=['no','yes'])
    y_pred = label_binarize(y_pred,classes=['no','yes'])
    print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
```

```
Precision:  0.3 Recall:  0.78
```

Random forest classification hardly performs better when applied on SMOTE data in terms of precision and recall.

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    forest = RandomForestClassifier(n_estimators= 1000,criterion="gini", max_depth=5,min_samples_split = 0.4,min_samples_leaf=1, class_weight="balanced")
    X_SMOTE, y_SMOTE = sm.fit_resample(X_train, y_train)
    model = forest.fit(X_SMOTE,y_SMOTE)
    y_pred = model.predict(X_test)
    pd.Series(y_pred).value_counts()
    y_test = label_binarize(y_test,classes=['no','yes'])
    y_pred = label_binarize(y_pred,classes=['no','yes'])
    print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
```

```
Precision:  0.34 Recall:  0.68
```

4.3 Support Vector Machine Model (SVM)

Support Vector Machine is employed for SMOTE data using two kernels: linear and Gaussian. Gaussian kernel performs the best in terms of both precision and recall.

```
[ ]  sm = SMOTE(random_state=0)
     X_SMOTE, y_SMOTE = sm.fit_resample(X, y)
     X_train, X_test, y_train, y_test = train_test_split(X_SMOTE, y_SMOTE, test_size=0.3)
     svm = SVC(kernel='linear')
     model = svm.fit(X_train, y_train)
     y_pred = model.predict(X_test)
     y_test = label_binarize(y_test,classes=['no','yes'])
     y_pred = label_binarize(y_pred,classes=['no','yes'])
     print("Linear kernel- ","Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
     fpr_linear, tpr_linear, _ = roc_curve(y_test, y_pred)
     roc_auc_linear = auc(fpr_linear, tpr_linear)
     sm = SMOTE(random_state=0)
     X_SMOTE, y_SMOTE = sm.fit_resample(X, y)
     X_train, X_test, y_train, y_test = train_test_split(X_SMOTE, y_SMOTE, test_size=0.3)
     svm = SVC(kernel='rbf')
     model = svm.fit(X_train, y_train)
     y_pred = model.predict(X_test)
     y_test = label_binarize(y_test,classes=['no','yes'])
     y_pred = label_binarize(y_pred,classes=['no','yes'])
     print("Guassian kernel- ","Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
     fpr_rbf, tpr_rbf, _ = roc_curve(y_test, y_pred)
     roc_auc_rbf = auc(fpr_rbf, tpr_rbf)
```

```
Linear kernel-  Precision:  0.9 Recall:  0.73

Guassian kernel-  Precision:  0.83 Recall:  0.87
```

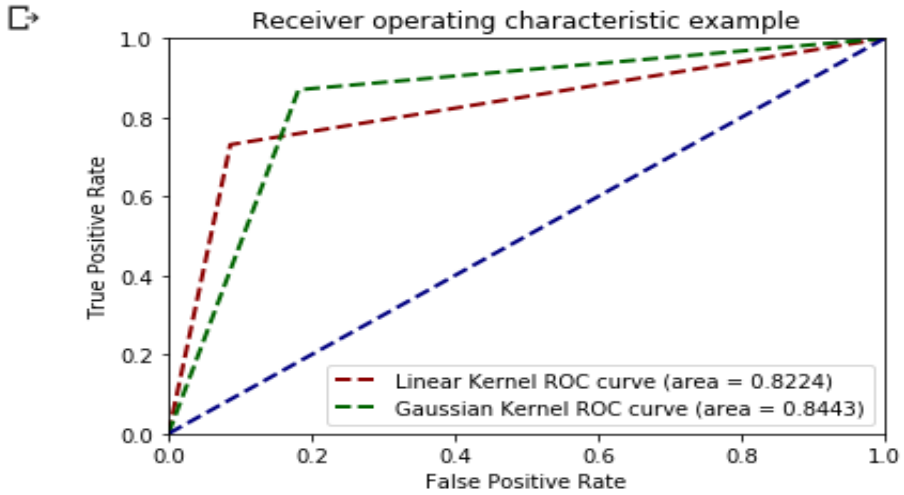
ROC Curves for the different Kernels:

```
[ ]  plt.figure()
     lw = 2

     plt.plot(fpr_linear, tpr_linear,
              label='Linear Kernel ROC curve (area = {0:0.4f})'
                    ''.format(roc_auc_linear),
              color='darkred', linestyle='--', linewidth=2)

     plt.plot(fpr_rbf, tpr_rbf,
              label='Gaussian Kernel ROC curve (area = {0:0.4f})'
                    ''.format(roc_auc_rbf),
              color='darkgreen', linestyle='--', linewidth=2)

     plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
     plt.xlim([0.0, 1.0])
     plt.ylim([0.0, 1.00])
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Receiver operating characteristic example')
     plt.legend(loc="lower right")
     plt.show()
```



4.4 Logistic Regression Model

Logistic regression was fit on imbalanced, random undersampled, random over-sampled and SMOTE data. Logistic regression on treated data performs fairly better than the first 2 models in this report but not better than SVM especially with Gaussian Kernel.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
model = lm.LogisticRegression(random_state=0, solver='lbfgs',multi_class='auto',max_iter=1000).fit(X_train,y_train)
y_pred = model.predict_proba(X_test)
y_pred = y_pred[:,1]
y_test = label_binarize(y_test,classes=['no','yes'])
fpr_imb, tpr_imb, _ = roc_curve(y_test, y_pred)
roc_auc_imb = auc(fpr_imb, tpr_imb)
y_pred = model.predict(X_test)
y_pred = label_binarize(y_pred,classes=['no','yes'])
print("Imbalanced -")
print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
# Undersampled
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
rus = RandomUnderSampler(random_state=0)
X_Usampled, y_Usampled = rus.fit_resample(X_train, y_train)
model = lm.LogisticRegression(random_state=0, solver='lbfgs',multi_class='auto',max_iter=5000).fit(X_Usampled,y_Usampled)
y_pred = model.predict_proba(X_test)
y_pred = y_pred[:,1]
y_test = label_binarize(y_test,classes=['no','yes'])
fpr_us, tpr_us, _ = roc_curve(y_test, y_pred)
roc_auc_us = auc(fpr_us, tpr_us)
y_pred = model.predict(X_test)
y_pred = label_binarize(y_pred,classes=['no','yes'])
print("Random undersampled -")
print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
```

```
Imbalanced -
Precision:  0.66 Recall:  0.43
```

```
Random undersampled -
Precision:  0.43 Recall:  0.87
```

```
# Oversampled
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
ros = RandomOverSampler(random_state=0)
X_Osampled, y_Osampled = ros.fit_resample(X_train, y_train)
model = lm.LogisticRegression(random_state=0, solver='lbfgs',multi_class='auto',max_iter=5000).fit(X_Osampled, y_Osampled)
y_pred = model.predict_proba(X_test)
y_pred = y_pred[:,1]
y_test = label_binarize(y_test,classes=['no','yes'])
fpr_os, tpr_os, _ = roc_curve(y_test, y_pred)
roc_auc_os = auc(fpr_os, tpr_os)
y_pred = model.predict(X_test)
y_pred = label_binarize(y_pred,classes=['no','yes'])
print("Random oversampled -")
print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
```

```
Random oversampled -
Precision:  0.44 Recall:  0.88
```

```
# SMOTE
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
sm = SMOTE(random_state=0)
X_SMOTE, y_SMOTE = sm.fit_resample(X_train, y_train)
model = lm.LogisticRegression(random_state=0, solver='lbfgs',multi_class='auto',max_iter=5000).fit(X_SMOTE,y_SMOTE)
y_pred = model.predict_proba(X_test)
y_pred = y_pred[:,1]
y_test = label_binarize(y_test,classes=['no','yes'])
fpr_smote, tpr_smote, _ = roc_curve(y_test, y_pred)
roc_auc_smote = auc(fpr_smote, tpr_smote)
y_pred = model.predict(X_test)
y_pred = label_binarize(y_pred,classes=['no','yes'])
print("SMOTE -")
print("Precision: ",round(precision_score(y_test,y_pred),2),"Recall: ",round(recall_score(y_test,y_pred),2))
```

```
SMOTE -
Precision:  0.44 Recall:  0.86
```

ROC curve depicts the variation of True Positive Rate to False Positive Rate.

Area under ROC curve is slightly better for over sampled data compared to imbalanced and undersampled data.

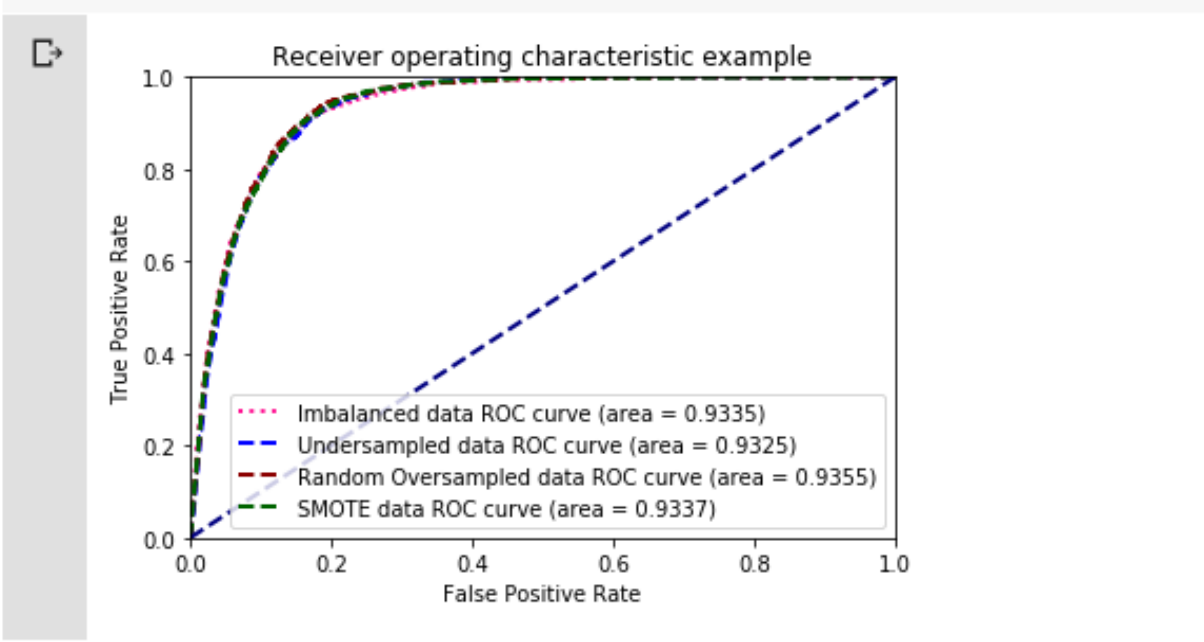
```
[ ]  plt.figure()
     lw = 2
     plt.plot(fpr_imb, tpr_imb,
              label='Imbalanced data ROC curve (area = {0:0.4f})'
                    ''.format(roc_auc_imb),
              color='deeppink', linestyle=':', linewidth=2)

     plt.plot(fpr_us, tpr_us,
              label='Undersampled data ROC curve (area = {0:0.4f})'
                    ''.format(roc_auc_us),
              color='blue', linestyle='--', linewidth=2)

     plt.plot(fpr_os, tpr_os,
              label='Random Oversampled data ROC curve (area = {0:0.4f})'
                    ''.format(roc_auc_os),
              color='darkred', linestyle='--', linewidth=2)

     plt.plot(fpr_smote, tpr_smote,
              label='SMOTE data ROC curve (area = {0:0.4f})'
                    ''.format(roc_auc_smote),
              color='darkgreen', linestyle='--', linewidth=2)

     plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
     plt.xlim([0.0, 1.0])
     plt.ylim([0.0, 1.00])
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Receiver operating characteristic example')
     plt.legend(loc="lower right")
     plt.show()
```



4.5 Results Comparison Table

MODEL	PRECISION	RECALL
Decision Tree	0.64	0.56
Decision Tree + SMOTE	0.48	0.83
Random Forest	0.3	0.78
Random Forest + SMOTE	0.34	0.68
SVM + SMOTE (Linear)	0.9	0.73
SVM + SMOTE (Gaussian)	0.83	0.87
Logistic Regression (imbalanced)	0.66	0.43
Logistic Regression (undersampled)	0.43	0.87
Logistic Regression (oversampled)	0.44	0.88
Logistic Regression + SMOTE	0.44	0.86

** Best two results are highlighted with green for precision and recall.

Conclusion

From all the observations and experiments done above, we were able to figure the best ways to treat the imbalance of the data and the best predictive model for our target "term deposit subscription" denoted as variable y. From data observation we deduce that the repeated campaign calls to customers within 20 days of previous call increases this variable. Moreover, as we saw in terms of precision and recall the best fitting model was SVM with Gaussian kernel after treating the imbalance data using SMOTE but also it was noticeable that Logistic Regression (oversampled) had the highest recall where SVM (linear) had the highest precision (almost same as SVM Gaussian).