# Predicting Customer Banking behavior


## Raghu Raman Nanduri


**DSC680 – Fall 2019**

https://iamnrr.github.io/

# 1. Business Objective & Understanding

We all know that, the best way to grow a business is by retaining existing customers and then acquire new customers. That's why most of the companies give importance to retain existing customers and increase their presence by cross selling other products to them.

The business objective of this project is to analyze the success rate in customer retention through direct campaign marketing by a Portuguese Bank for which this data set pertains to. It is important to find what parameters define the outcome of the direct campaign and what parameters are a must for a positive outcome leading to a subscription. In order to increase the chance of their success, if a bank can predict who are the customers that are most likely to subscribe for their new program (here the program is term deposit).

**Business Goal:**

In terms of modelling perspective, the business goal is to define a predictive model involving binary classification of customers into 2 categories –

1) Customers that might open term deposit account with the bank after the campaign.

2) Customers that will not open term deposit account with the bank.

This classification really helps banks to identify the customers that they can concentrate their campaigning resources on such that it gives them maximum chance to succeed in their campaign resulting in more term deposit accounts (indirectly increasing their program's success rate).

**Approach:**

CRISP – DM Methodology has been used to perform this supervised learning classification task.

# 2. Data Understanding

The data set consists of direct marketing campaigning results from a Portuguese bank. The direct campaigning involves their existing customers that have either an account with some balance or personal loan or a housing loan. The data set includes the demographic information about such customers like their job status, marital status, educational details. It also includes direct call campaigning details like contact type, day & month of contact, how long the telephonic conversation took place along with previous campaigning results. As data set involves, only phone calls my research includes the effectiveness of the direct marketing campaign.

Data set is available at the link - https://www.openml.org/d/1461

**Feature Variables:** There are 16 input variables, out of which some are categorical, and some are numeric. Below is the entire list

1 - age (numeric)

2 - job : type of job (categorical:
"admin.","unknown","unemployed","management","housemaid","entrepreneur",
"student","blue-collar","self-employed","retired","technician","services")

3 - marital : marital status (categorical: "married","divorced","single"; note: "divorced" means divorced or widowed)

4 - education (categorical: "unknown","secondary","primary","tertiary")

5 - default: has credit in default? (binary: "yes","no")

6 - balance: average yearly balance, in euros (numeric)

7 - housing: has housing loan? (binary: "yes","no")

8 - loan: has personal loan? (binary: "yes","no")

related with the last contact of the current campaign:
9 - contact: contact communication type (categorical: "unknown","telephone","cellular")

10 - day: last contact day of the month (numeric)

11 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")

12 - duration: last contact duration, in seconds (numeric)

other attributes:
13 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

14 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)

15 - previous: number of contacts performed before this campaign and for this client (numeric)

16 - poutcome: outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")

**Target Variable:**

Only one variable that needs to be predicted that is the customer's classification, it is Boolean type with yes/no (represented by 1, 0 in the data set). Yes, means the customer has subscribed for the term deposit and No, means the customer did not subscribe. The actual data set has values 1 & 2 which were modified as 0 & 1 before performing exploratory data analysis.

**Exploratory Data Analysis:**

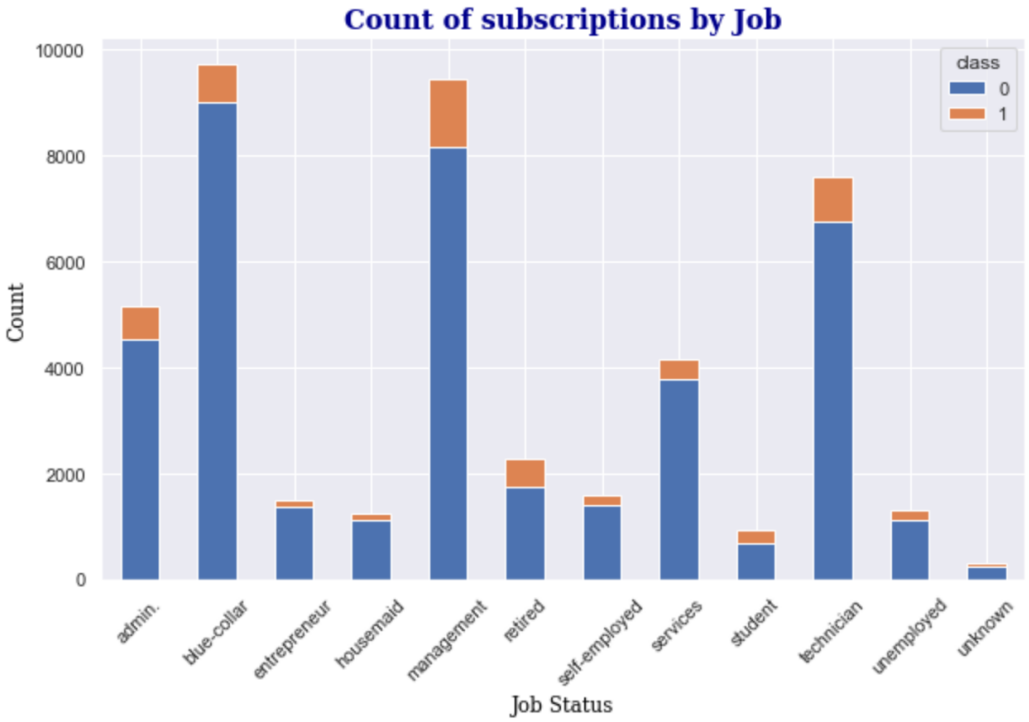
Firstly, I checked for missing values in the data set. Luckily, the available data set was very clean with out any missing values.

```
# Checking for blank values for each column of dataframe
nullcols = bankdf.isnull().sum()
print(nullcols)

```

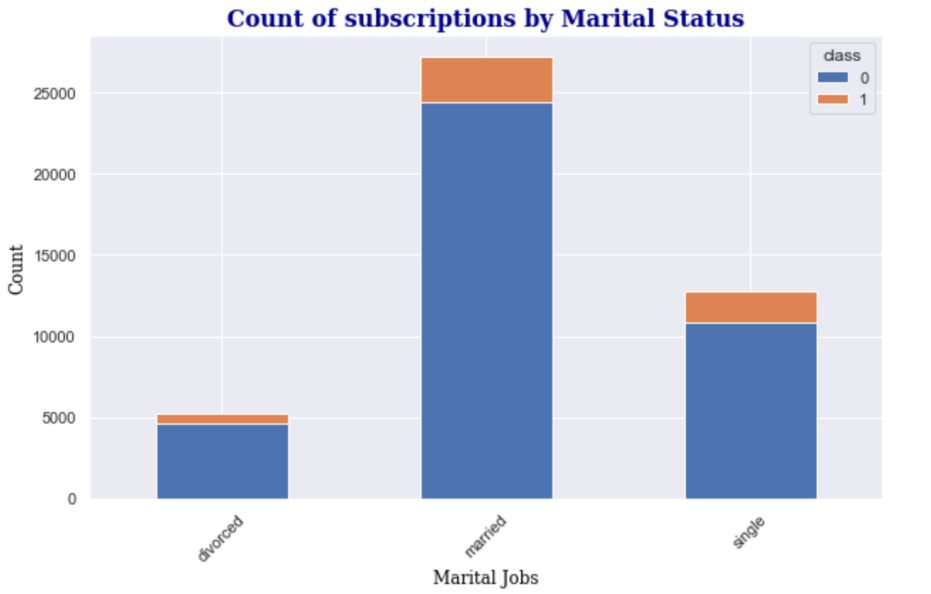
```
age             0
job             0
marital_status  0
education       0
default         0
balance         0
housing         0
loan            0
contact         0
day             0
month           0
duration        0
campaign        0
pdays           0
previous        0
poutcome        0
class           0
dtype: int64
```

For better understanding of the data, I performed the initial exploratory data analysis for all the variables.

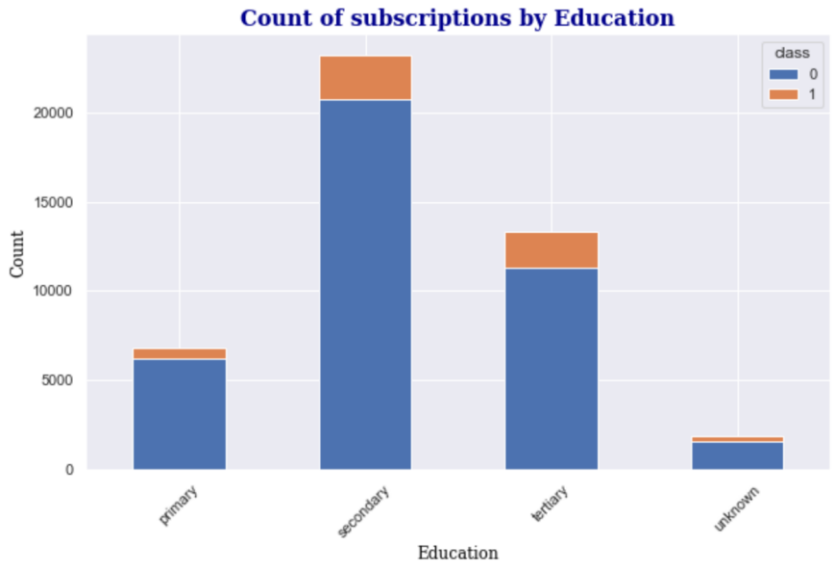
1) Job Status: Top 3 job statuses among the current customers are blue-collar, management, and technician.



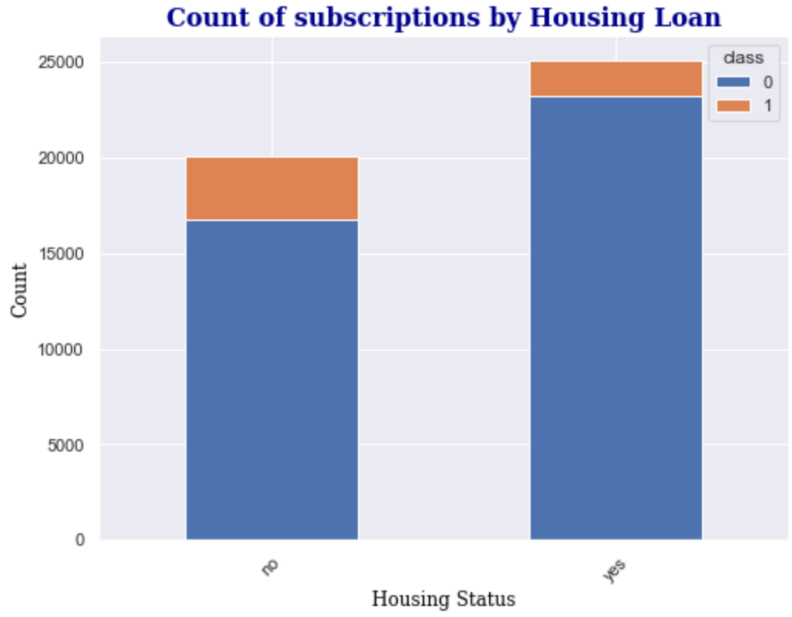

2) Marital Status: Most of the current customers are married as clearly depicted in below bar plot.



3) Education Status – Most of the customers have at least secondary education and tertiary education.

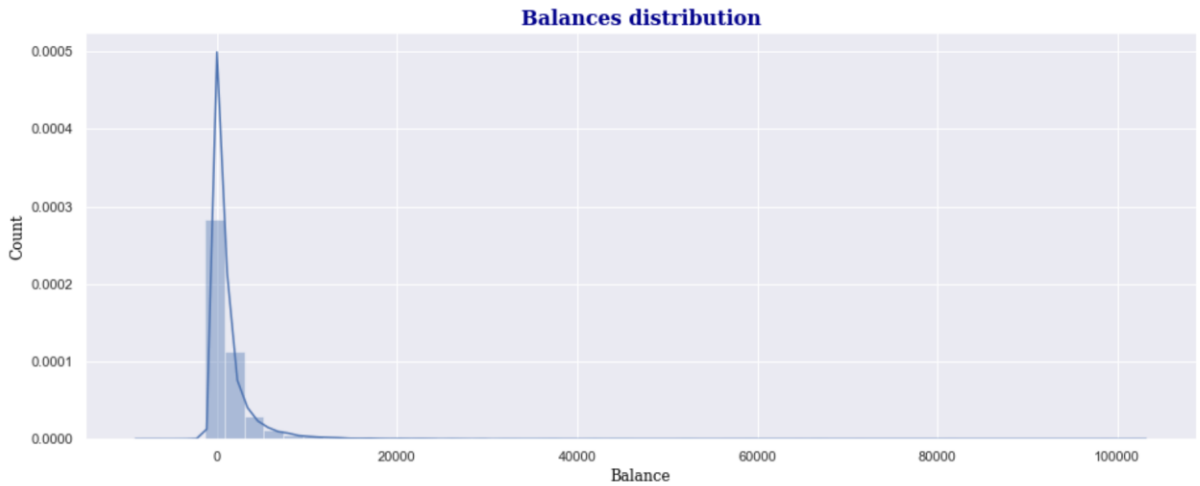



4) Housing Loan: Most of the current customer bunch does have housing loan with the bank and looks like customers that do not have housing loan with the bank are more likely to subscribe for term deposit.

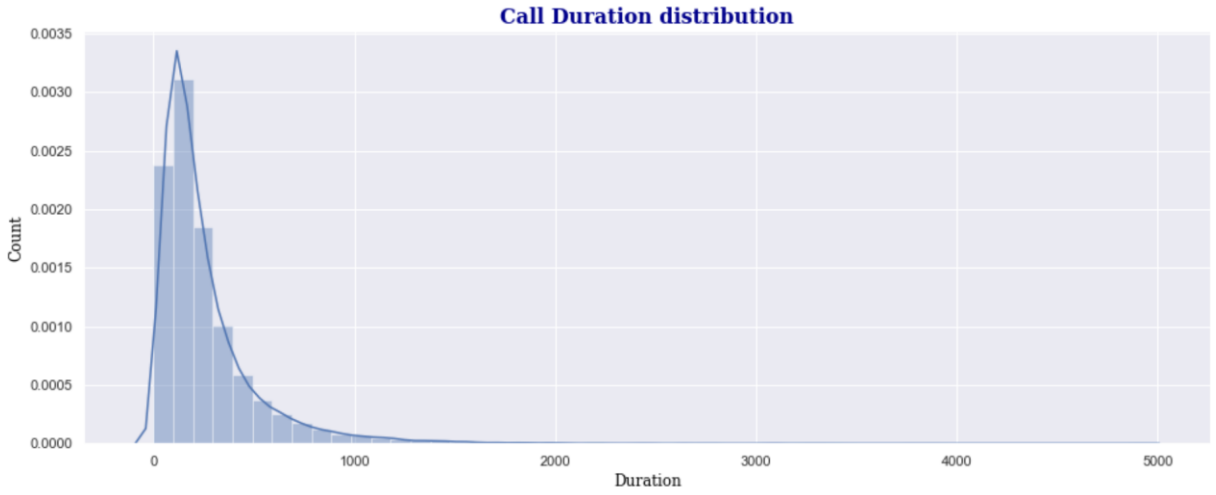


5) Personal Loan: Most of the current customer bunch does not have personal loan with the bank and looks like customers that do not have personal loan with the bank are more likely to subscribe for term deposit.

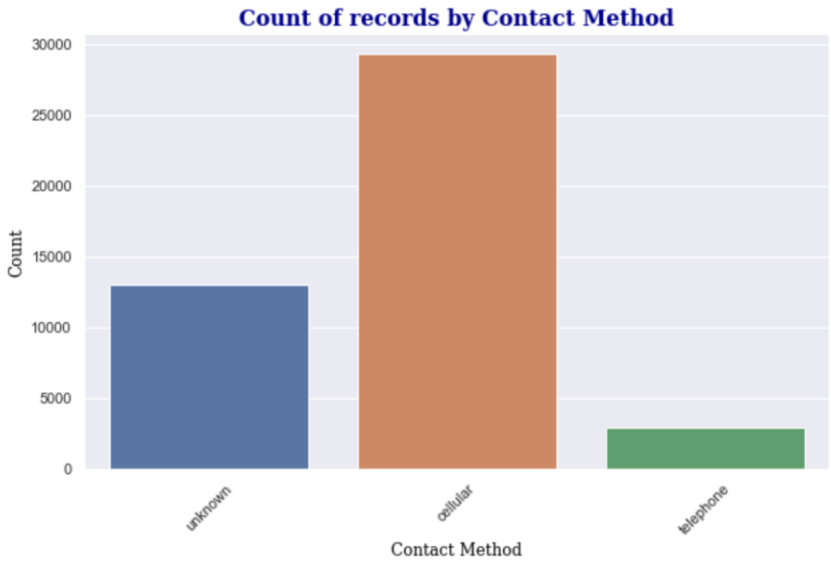

6) Balance: Balance distribution of the banks current customers is highly skewed with long tail towards higher end of the scale. We can infer that most of the current customers maintain less than 10,000 balance.



7) Last Call duration: Most of the current customers have spoken for less than 500 seconds during the last call campaign. Which means, the campaign representatives on average must spend less than 10 minutes, to get to know either the customer is interested or not in subscribing to term deposit program.

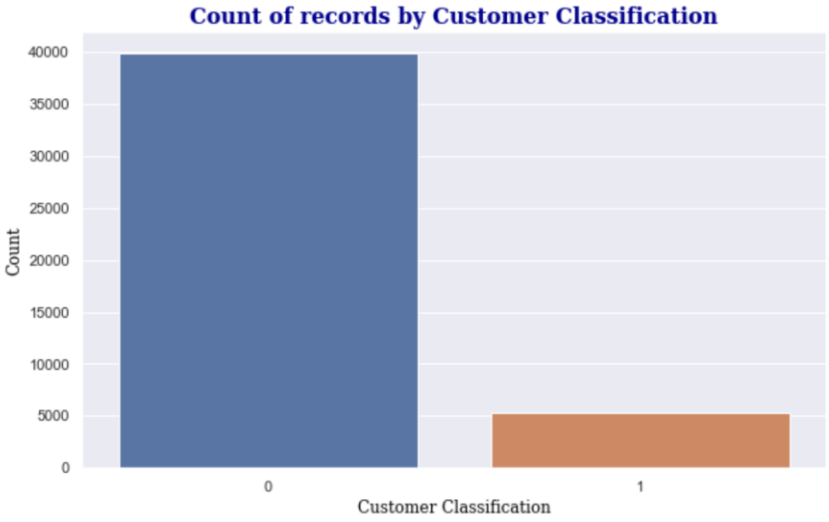



8) Contact Method: Most of the customers prefer cellular phone as the primary contact method. It is understandable as cell phones offer flexibility to be carried all the time.



9) Customer classification:

The data set currently is highly unbalanced. Most of the customers did not subscribe for the term deposit program. The predictive modeling, needs to care of this imbalance in data set.




For more detailed exploratory data analysis, please visit my Git Hub portfolio - https://iamnrr.github.io/

# 3. Data Preparation

During the early stages of the exploratory data analysis observed that the target variable – 'class', holds values 1 & 2. To be truly convert it into binary format, modified the values to 0 & 1 respectively. Due to lack of clarity in the documentation available, I have considered current value 1 represents 'NO' in binary format and so replaced with numeric 0; value 2 – represents binary 'YES' – which is replaced with numeric 1.

```
# Changing class variable values to 0 (no) or 1 (yes)

bankdf.loc[bankdf['class'] == 1, 'class'] = 0    # Changing all NO to 0
bankdf.loc[bankdf['class'] == 2, 'class'] = 1    # Changing all YES to 1
```

**Feature Engineering**

The input features currently have lot of categorical  variables. Categorical variables need to be encoded to fit the predictive models. Here, I used pandas – getdummies() function to create dummy variables for all the categorical variables.

```
bankdf_wdummy = pd.get_dummies(bankdf)
bankdf_wdummy.head()
```

To avoid dummy variable trap, one dummy variable for each categorical variable should be removed. Below dummy variable columns are removed:

'job_unknown', 'marital_status_divorced','education_unknown','default_no','housing_no', 'loan_no', 'month_dec','contact_unknown', 'poutcome_unknown'

```
# Resolving Dummy Variable Trap
bankdf_wdummy_final = bankdf_wdummy[['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous',
       'class', '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_status_married', 'marital_status_single',
       'education_primary', 'education_secondary',
       'education_tertiary',  'default_yes',
       'housing_yes', 'loan_yes', 'contact_cellular',
       'contact_telephone',  'month_apr', 'month_aug',
       'month_feb', 'month_jan', 'month_jul', 'month_jun',
       'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
       'poutcome_failure', 'poutcome_other', 'poutcome_success',
       ]]

```

# 4. Predictive Modelling

The predictive problem (predict whether customer subscribes or not), we have at our hand is in fact a binary classification problem. Also, the data set already contains the target classification, which means that this is a supervised learning classification. Due to availability of this baseline classification, using which we can train the model that we are about to build, the advantage of supervised learning applies here as well. To resolve this supervised binary classification problem, I want to try Random Forest Classifier.

**Why Random Forest?**

Random Forest is an ensemble of decision trees. Ever since, I learnt about Random Forest it has been one of my favorite models to consider, with the understanding that it may not fit to all problems. But I see the below advantages:

1) To begin with, our data has unbalanced classes and Random Forest handles unbalanced data well by minimizing the overall error rate.

2) At each random split of decision tree, the model picks random subset of features, which reduces the correlation between the features selected improving the variance of the model.

3) Random Forest is not affected with the presence of outliers and non- linear data. In the exploratory data analysis, we have observed that the variable 'balance' has outliers. By selecting Random Forest, the presence of outliers is negated.

**Feature Selection**

In order to improve the accuracy and quality of our predictions, it is necessary to select the important variables that have more say in prediction. This helps in reduction of errors and improving noise due to presence of noise in the data set.

To select the important features from our current data set, I have used the ExtraTreesClassifier algorithm.

```
# feature selection
fsmodel = ExtraTreesClassifier(n_estimators=10)
fsmodel.fit(features, target)
impfeatures = fsmodel.feature_importances_
print(impfeatures)
print(features.columns)
```

Converting feature importance metric into a data frame for easy read.

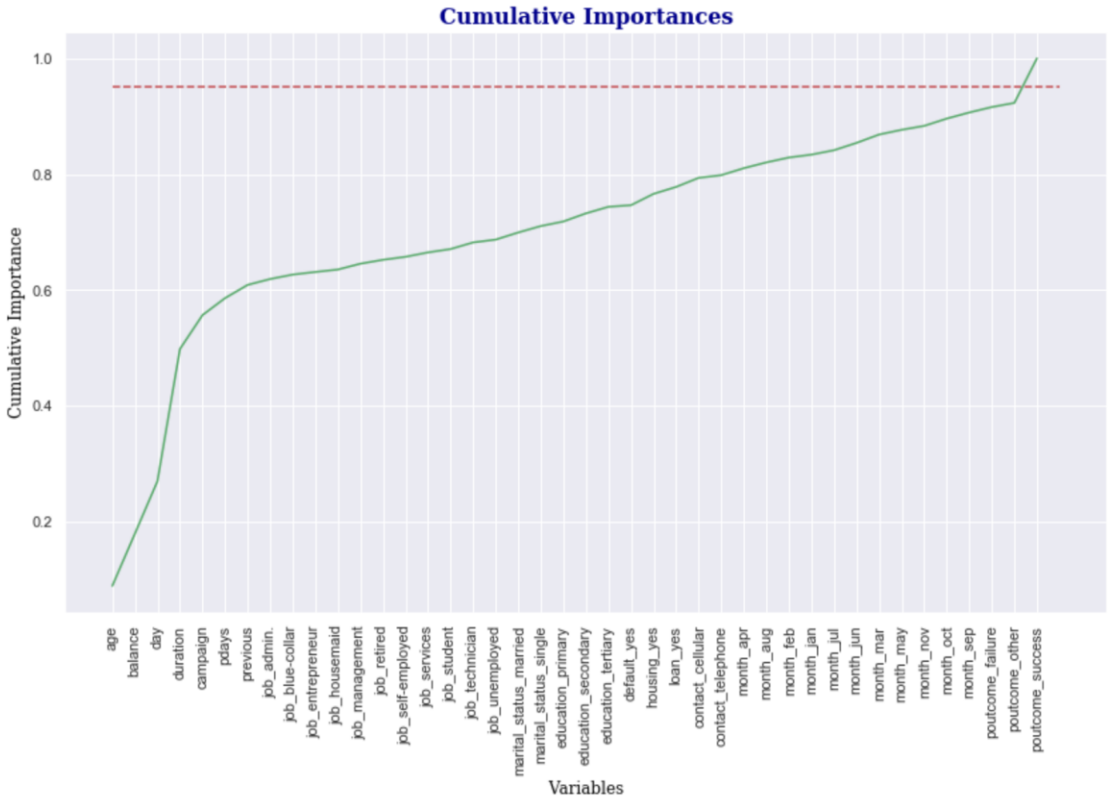
```
impfeatdf = pd.DataFrame(features.columns, impfeatures).reset_index()
impfeatdf.rename(columns = {'index':'featureimportance',0: 'featurename'}, inplace = True)
impfeatdf.sort_values('featureimportance',ascending=False)
#impfeatdf.sort_values('impfeatures',ascending=False)
```

Below is the table that shows each variable and its importance. The most important variable among that has considerable say in the outcome of predicting subscription is "duration" of call.

| Sno | featurename | featureimportance |
|---|---|---|
| 3 | duration | 0.228362 |
| 2 | day | 0.09059 |
| 1 | balance | 0.090038 |
| 0 | age | 0.089262 |
| 41 | poutcome_success | 0.076974 |
| 4 | campaign | 0.058466 |
| 5 | pdays | 0.029296 |
| 6 | previous | 0.022766 |
| 24 | housing_yes | 0.019583 |
| 26 | contact_cellular | 0.015671 |

| | | |
|---|---|---|
| 34 | month_mar | 0.01439 |
| 21 | education_secondary | 0.013993 |
| 33 | month_jun | 0.012649 |
| 37 | month_oct | 0.012563 |
| 18 | marital_status_married | 0.012281 |
| 28 | month_apr | 0.012043 |
| 25 | loan_yes | 0.011929 |
| 16 | job_technician | 0.011433 |
| 22 | education_tertiary | 0.011335 |
| 19 | marital_status_single | 0.011126 |
| 38 | month_sep | 0.010528 |
| 7 | job_admin. | 0.010175 |
| 11 | job_management | 0.010078 |
| 29 | month_aug | 0.009988 |
| 39 | poutcome_failure | 0.009375 |
| 30 | month_feb | 0.008408 |
| 35 | month_may | 0.008198 |
| 20 | education_primary | 0.007904 |
| 8 | job_blue-collar | 0.007686 |
| 32 | month_jul | 0.007633 |
| 14 | job_services | 0.007605 |
| 40 | poutcome_other | 0.007216 |
| 36 | month_nov | 0.006789 |
| 12 | job_retired | 0.006681 |
| 15 | job_student | 0.005816 |
| 13 | job_self-employed | 0.005196 |
| 31 | month_jan | 0.005011 |
| 17 | job_unemployed | 0.004839 |
| 27 | contact_telephone | 0.004639 |
| 9 | job_entrepreneur | 0.004495 |
| 10 | job_housemaid | 0.004197 |
| 23 | default_yes | 0.002793 |

From the below plot, which shows the cumulative importance of the features inorder to get to 95% of the importance, we need to consider almost all features. Hence, for this model no further feature selection is required, we need to consider all the available variables.




## Model Training

Created a base model using Random Forest Classifier with the below parameters and trained the dataset.

```
# Create random forest classifier object
randomforest = RandomForestClassifier(random_state=1,           # for consistent results
                                      n_estimators = 200,       # number of tress in forest
                                      oob_score=True,           # OOB Score to get performance
                                      bootstrap=True,
                                      n_jobs=-1,                # for using all cores
                                      class_weight="balanced"   # for handling imbalanced classes
                                     )
# Train model
basemodel = randomforest.fit(X_train, y_train)
```

# 5. Evaluation

We cannot consider Accuracy as a metric to evaluate the performance of our predictive model as our current data set has unbalanced classes. So, we need consider other metrics like confusion matrix, classification report and ROC Curve.

Below are the resultant metrics from evaluating our random classifier classification model:

**Confusion Matrix**

```
# Create confusion matrix
matrix = confusion_matrix(y_test, y_pred)
matrix
```

```
array([[9791,  207],
       [ 872,  433]], dtype=int64)
```

**Classification Report**

```
# printing classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      9998
           1       0.68      0.33      0.45      1305

   micro avg       0.90      0.90      0.90     11303
   macro avg       0.80      0.66      0.70     11303
weighted avg       0.89      0.90      0.89     11303
```

**Accuracy, Precision, Recall, F1 Score**

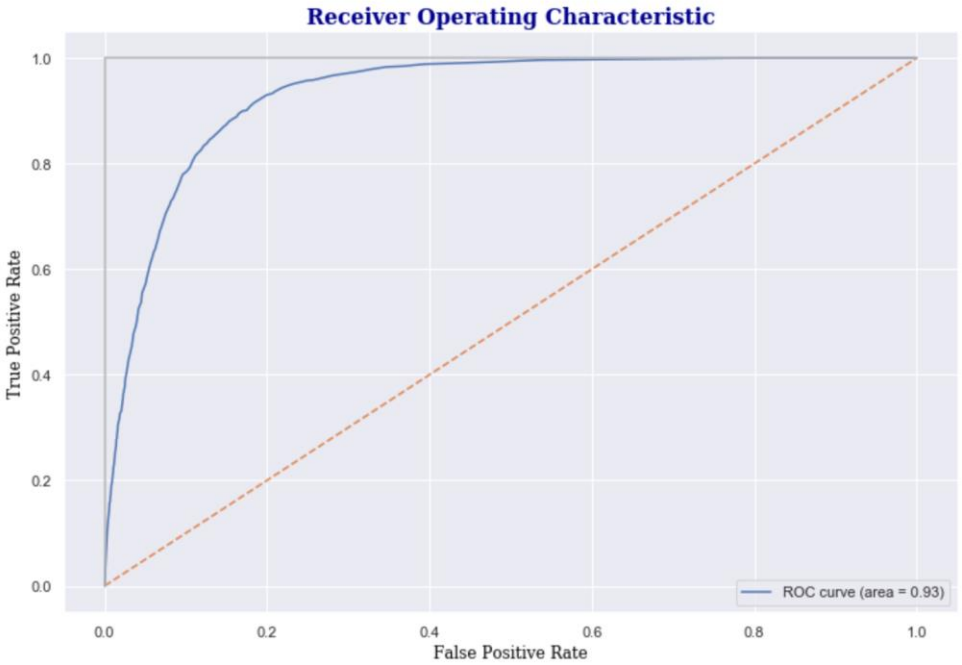
```
acc, prec, recall, f1 = fn_multiclass_metrics(y_test, y_pred)

acc, prec, recall, f1

```

```
(0.9045386180660002,
 0.8903208445956925,
 0.9045386180660002,
 0.8897556560886072)
```

Precision tells us whether the customers we predicted as those who might subscribe to the term program subscribed or not. Metric Recall tells us what factor of customers that subscribed were predicted by our model as those who might subscribe.

Our above evaluation shows that we have a Precision of 89% and Recall factor of 90%.

**ROC Curve:**




The above Receiver Operating Characteristics plot shows that the base model was created is having AUC value of 0.93%, shows that the model differentiates between customers between the two classes well.

# 6. Model Tuning

To tune the model further, I have created the below param grid and used GridSearchCV, which tries every combination of parameter given in the parameter grid and finds out the best parameter. Though we might get better accuracy with this approach, it could lot of time to search through all parameters. So, we need to be cautious consider the tradeoff between performance and accuracy before defining the parameters for GridSearchCV.

```
# Create the parameter grid based on the results of random search
param_grid = {
    'n_estimators': [100, 250, 500, 1000, 2000],
    "criterion": ["gini", "entropy"],
     }

# Create a default rf model
rf = RandomForestClassifier(random_state=1,           # for consistent results
                                    oob_score=True,           # OOB Score to get performance
                                    bootstrap=True,
                                    n_jobs=-1,                # for using all cores
                                    class_weight="balanced" )
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
```

For the given parameter grid, GridSearchCV found the below parameters as the best estimators for the Random Forest Classifier model.

```
best_grid = grid_search.best_estimator_
print(best_grid)
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=2000, n_jobs=-1, oob_score=True, random_state=1,
            verbose=0, warm_start=False)
```

# 7. Deployment

Once the model is fine-tuned with the appropriate parameters, the model is ready for deployment. We can use JOBLIB module in python to save the model on disk in pickle format and save the pickle file to fit the data in the real time.

```
# Create random forest classifier object
bestmodel = RandomForestClassifier(random_state=1,           # for consistent results
                                   n_estimators = 2000,       # number of tress in forest
                                   oob_score=True,           # OOB Score to get performance
                                   bootstrap=True,
                                   n_jobs=-1,                # for using all cores
                                   class_weight="balanced",   # for handling imbalanced classes
                                   criterion='entropy'
                                  )

```

```
# Save the model as a pickle in a file
joblib.dump(bestmodel, 'Model\\bestmodel.pkl')
```

```
['Model\\bestmodel.pkl']
```

```
# Load the model from the file
tunedmodel_from_joblib = joblib.load('Model\\bestmodel.pkl')

```

```
# Fitting deployed model on new data ( assume here X_train and y_train are new unseen features and targets)
deployed_model = tunedmodel_from_joblib.fit(X_train, y_train)
```

## 8. Conclusion

To conclude using Random Forest Classifier algorithm and hyper tuning it further using either GridSearchCV or RandomSearchCV, we should be able to predict whether a customer will subscribe or not to a term deposit program in a performant manner. Next step is to see if performance of the model can be further improved upon using Deep Learning and Neural Networks.

The code for this project can be obtained from my Git Hub repository here.

Please visit my Git Hub Portfolio to look at my other projects.

## 9. Assumptions

Due to lack of clarity in the code book and misrepresentation of the binary variable – class with values 1 & 2, I had to make below assumptions:

1) The value ‘1’ represents binary ‘no’ and hence replaced it with numeric 0.

2) The value ‘2’ represents binary ‘yes and hence replaced it with numeric 1.

## 10. Techniques

Used the below modules in python to accomplish this supervised classification task.

- pandas
- numpy
- sklearn
- matplotlib
- seaborn
- joblib
- os

## 11. References


1. Data source link - https://www.openml.org/d/1461

2. Chris Albon, April 2018, “Python Machine Learning Cookbook”, ISBN 9781491989388.

3. Jake Huneycutt, May 18 2018, "Implementing a Random Forest Classification Model in Python", https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-583891c99652

4. Will Koehrsen, Jan 9 2018, "Hyperparameter Tuning the Random Forest in Python", https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

5. Mohammed Sunasra, Nov 11, 2017, "Performance Metrics for Classification problems in Machine Learning", https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b