Subscribe term deposit prediction with Bank data

About data

Source:

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

Abstract: The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

Data Set Information:

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

There are four datasets:

- bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010), very close to the data analyzed in [Moro et al., 2014]
- bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.
- bank-full.csv with all examples and 17 inputs, ordered by date (older version of this dataset with less inputs).
- bank.csv with 10% of the examples and 17 inputs, randomly selected from 3 (older version of this dataset with less inputs). The smallest datasets are provided to test more computationally demanding machine learning algorithms (e.g., SVM).

The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Attribute Information:

Input variables:

bank client data:

- age (numeric)
- job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
- marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
- education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
- default: has credit in default? (categorical: 'no','yes','unknown')
- housing: has housing loan? (categorical: 'no','yes','unknown')
- loan: has personal loan? (categorical: 'no','yes','unknown') #### related with the last contact of the current campaign:
- contact: contact communication type (categorical: 'cellular','telephone')
- month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
- duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model. #### other attributes:
- campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
- pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- previous: number of contacts performed before this campaign and for this client (numeric)
- poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success') #### social and economic context attributes
- emp.var.rate: employment variation rate - quarterly indicator (numeric)
- cons.price.idx: consumer price index - monthly indicator (numeric)
- cons.conf.idx: consumer confidence index - monthly indicator (numeric)
- euribor3m: euribor 3 month rate - daily indicator (numeric)
- nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

- y - has the client subscribed a term deposit? (binary: 'yes','no')

Goal of the case study

In the above research paper, the reserchers have mentioned that the best results acheived is of AUC-0.8, so our goal for this case study will be to achieve similar results as of the research paper. Though it is to be mentioned that after going through the research paper I realised that the data does not have all the features that are mentioned in the paper.

Type of Machine Learning Problem:

It is a binary classification problem where our objective is to predict whether a customer will subscribe a term deposit or not given the data of the customer.

Performance Metric:

The performance metric used in this case study is:

- AUC Score

Get the data

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
root_path = "drive/My Drive/BankMarketing_CaseStudy"

import os
os.chdir(root_path)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
!unzip "bank-additional.zip"
```

```
Archive:  bank-additional.zip
   creating: bank-additional/
  inflating: bank-additional/.DS_Store
   creating: __MACOSX/
   creating: __MACOSX/bank-additional/
  inflating: __MACOSX/bank-additional/._.DS_Store
  inflating: bank-additional/.Rhistory
  inflating: bank-additional/bank-additional-full.csv
  inflating: bank-additional/bank-additional-names.txt
  inflating: bank-additional/bank-additional.csv
  inflating: __MACOSX/._bank-additional
```

In [0]:

```
# Import the libraries
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
```

Basic info of the dataset

In [0]:

```
# Loading the dataset
data = pd.read_csv("/content/drive/My Drive/BankMarketing_CaseStudy/bank-additional/bank-additional-full.csv", sep=";")
data.info()
```

In [0]:

```
data.describe()
```

Out[0]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.00000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.62129
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.73444
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.63400
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.34400
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.85700
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.96100
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.04500

There are no missing values in the dataset. So we don't need to handle missing data for this case study

In [0]:

```
data.head()
```

Out[0]:

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

In [0]:

```
print(data["job"].value_counts())
print("*"*25)
print(data["marital"].value_counts())
print("*"*25)
print(data["education"].value_counts())
```

```
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
```

```
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: job, dtype: int64
*************************
married     24928
single      11568
divorced     4612
unknown        80
Name: marital, dtype: int64
*************************
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: education, dtype: int64
```

In [0]:

```
print(data["y"].value_counts())
```

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

From the above distribution we can be sure that the data is imbalanced, as the number of "no"s are also 8 times the number of "yes"

In [0]:

```
index.shape
```

Out[0]:

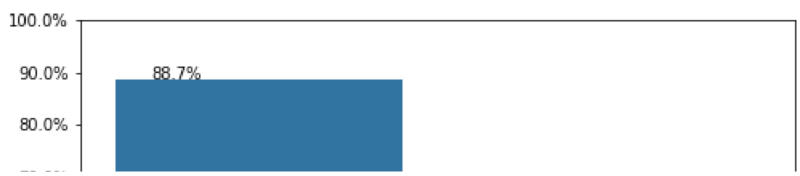
```
(41188,)
```

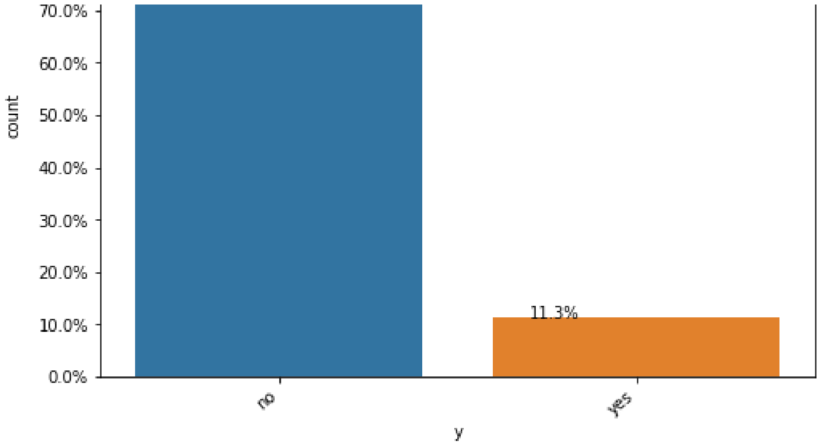
Exploratory Data Analysis

Distribution of Class variable

In [0]:

```
plt.figure(figsize=(8,6))
Y = data["y"]
total = len(Y)*1.
ax=sns.countplot(x="y", data=data)
for p in ax.patches:
  ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

#put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
ax.yaxis.set_ticks(np.linspace(0, total, 11))
#adjust the ticklabel to the desired format, without changing the position of the ticks.
ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
# ax.legend(labels=["no","yes"])
plt.show()
```





Univariate Analysis

In [0]:

```
def countplot(label, dataset):
  plt.figure(figsize=(15,10))
  Y = data[label]
  total = len(Y)*1.
  ax=sns.countplot(x=label, data=dataset)
  for p in ax.patches:
    ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

  #put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
  ax.yaxis.set_ticks(np.linspace(0, total, 11))
  #adjust the ticklabel to the desired format, without changing the position of the ticks.
  ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
  ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
  # ax.legend(labels=["no","yes"])
  plt.show()
```

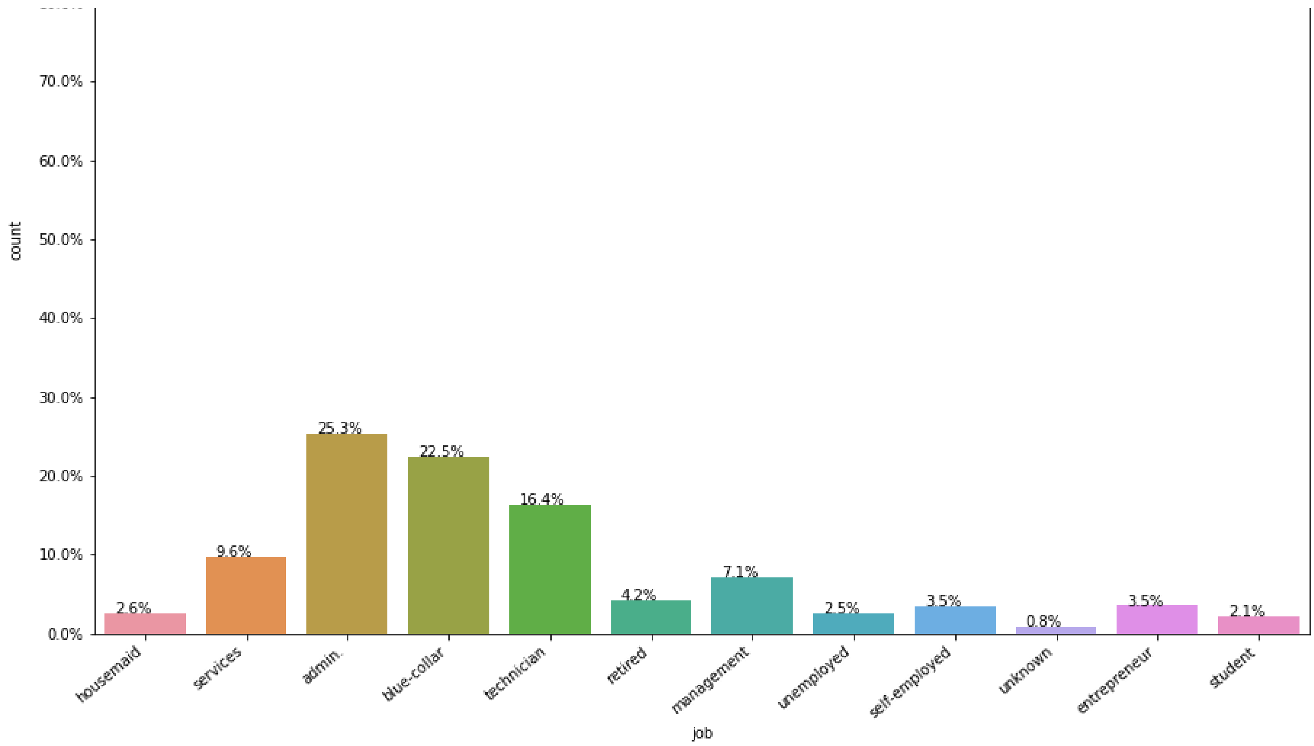
In [0]:

```
%matplotlib inline

def countplot_withY(label, dataset):
  plt.figure(figsize=(20,10))
  Y = data[label]
  total = len(Y)*1.
  ax=sns.countplot(x=label, data=dataset, hue="y")
  for p in ax.patches:
    ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

  #put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
  ax.yaxis.set_ticks(np.linspace(0, total, 11))
  #adjust the ticklabel to the desired format, without changing the position of the ticks.
  ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
  ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
  # ax.legend(labels=["no","yes"])
  plt.show()
```

Feature: Job (Categorical variable)

In [0]:

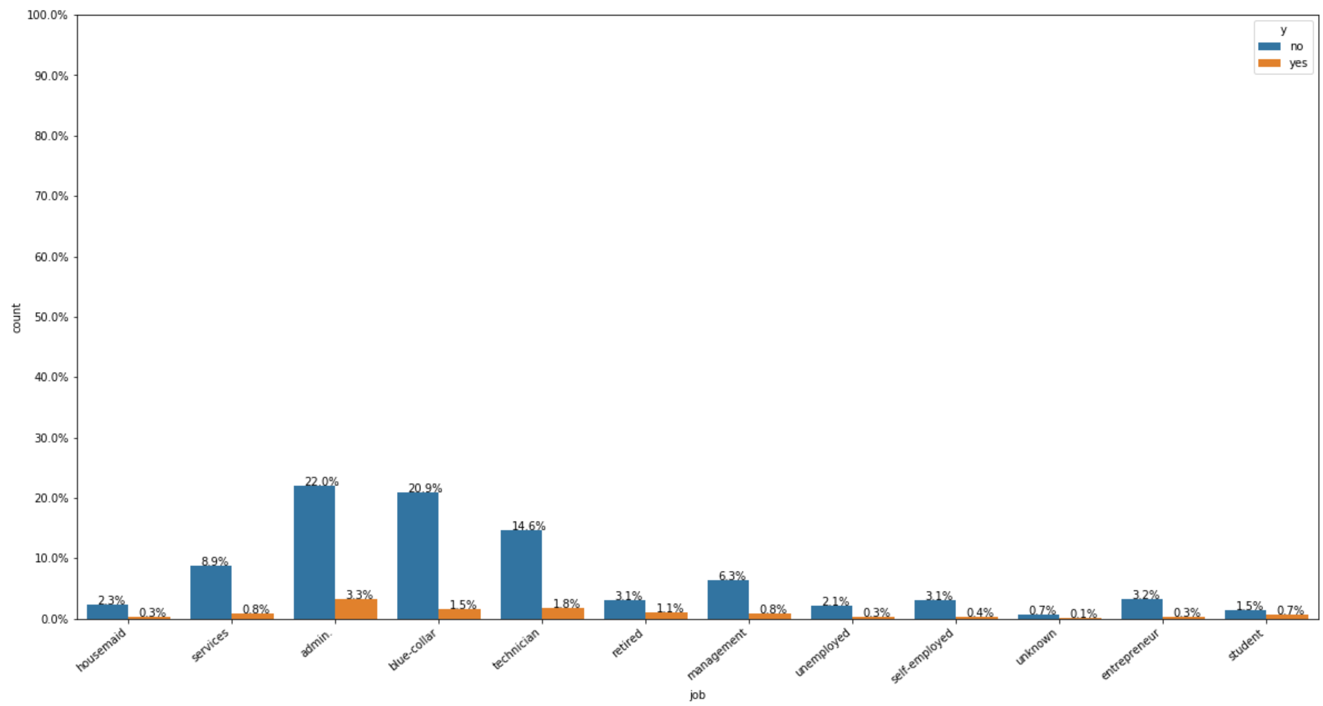
```
countplot("job", data)
```



From the above distribution we can see that most of the customers have jobs as "admin", "blue-collar" or "technician". One interesting thing to find out would be to see the distribution for each classes as well. For example, how many people who work as an admin have subscribed a term deposit.

In [0]:

```
countplot_withY("job", data)
```



From the above plot, we can see that the customers who have a job of admin have the highest rate of subscribing a term deposit, but they are also the highest when it comes to not subscribing. This is simply because we have more customers working as admin than any other profession.

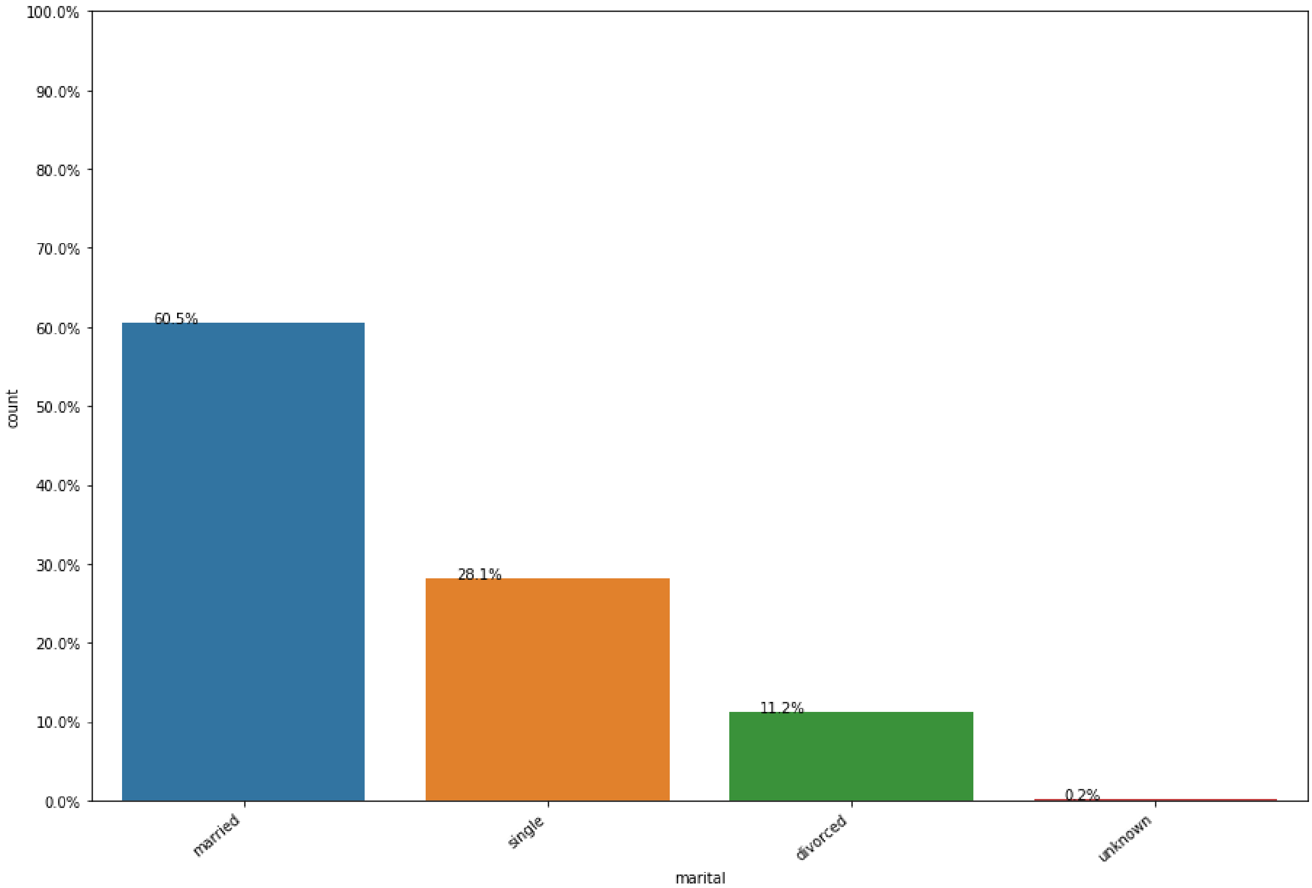
We can find out the odds or ratio of subscribing and not subscribing based on the profession, to find out which profession has the highest odds of subscribing given the data. At this point we are not sure if there is any correlation between job and target variable.

Idea: If we find that odds of one profession subscribing is greater than other job, we can use the odds or log(odds) as a feature by replacing jobs field with the odds, instead of doing one hot encoding.

Feature: Marital (Categorical feature)

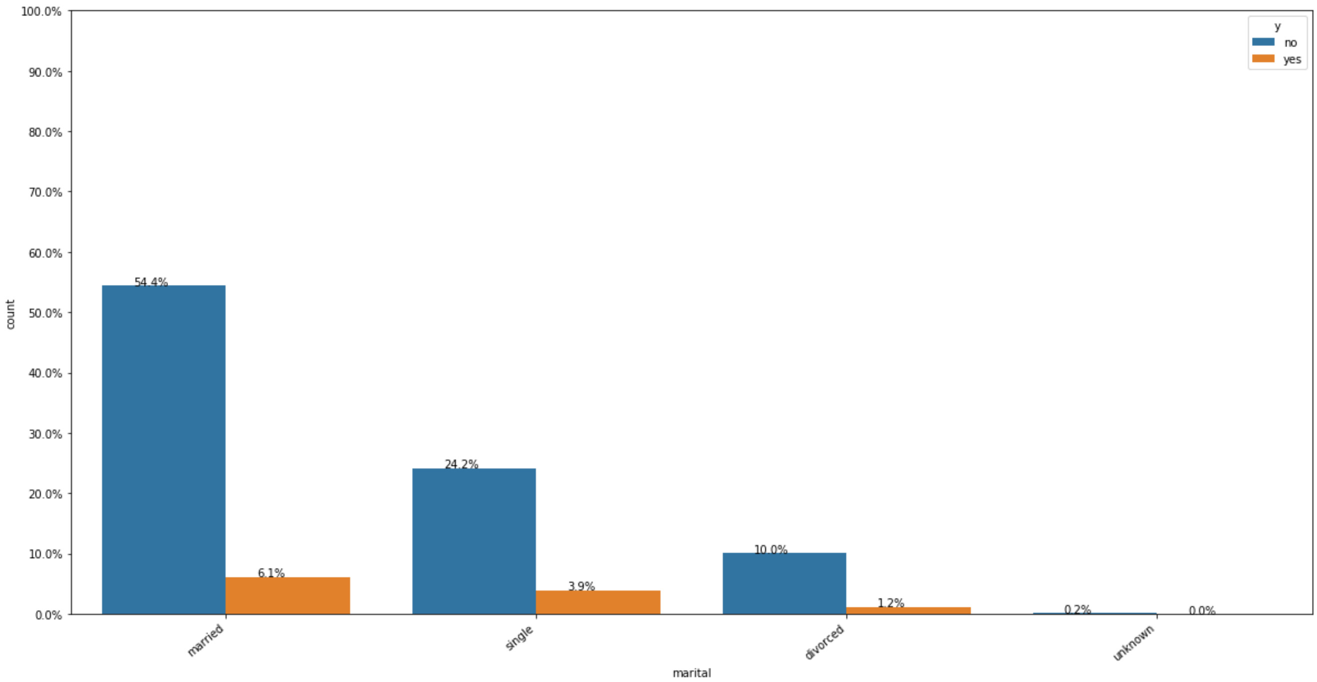
In [0]:

```
countplot("marital", data)
```



In [0]:

```
countplot_withY("marital", data)
```

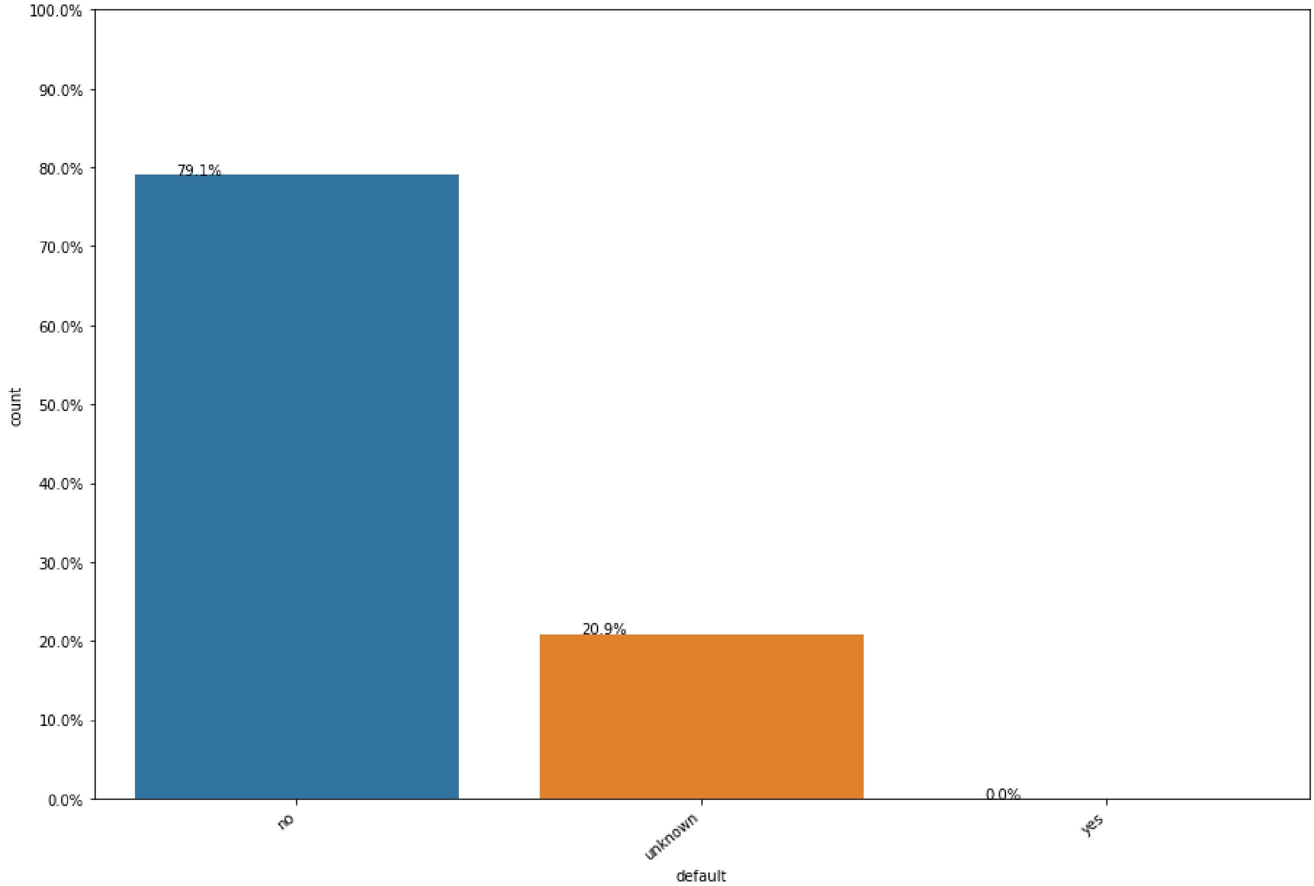


Feature: default (categorical)

This is a categorical feature which means "has credit in default", with the values "yes" and "no" and "unknown".

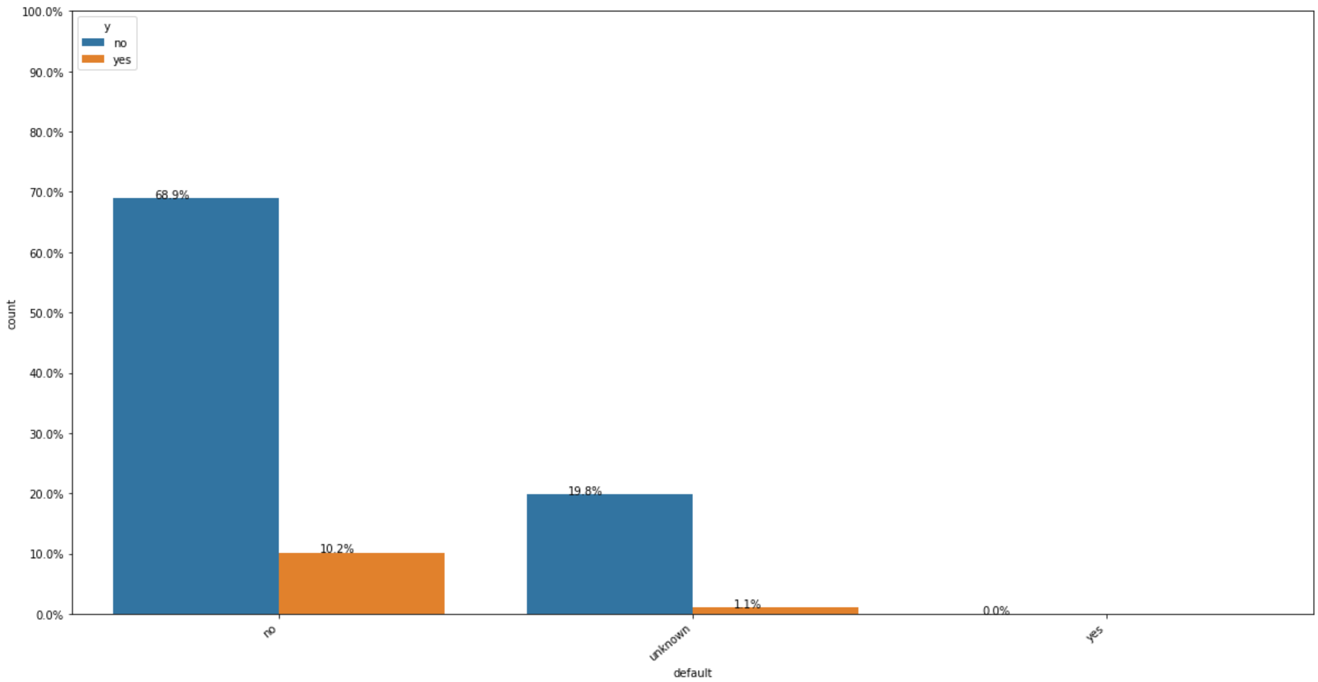
In [0]:

```
countplot("default", data)
```



In [0]:

```
countplot_withY("default", data)
```



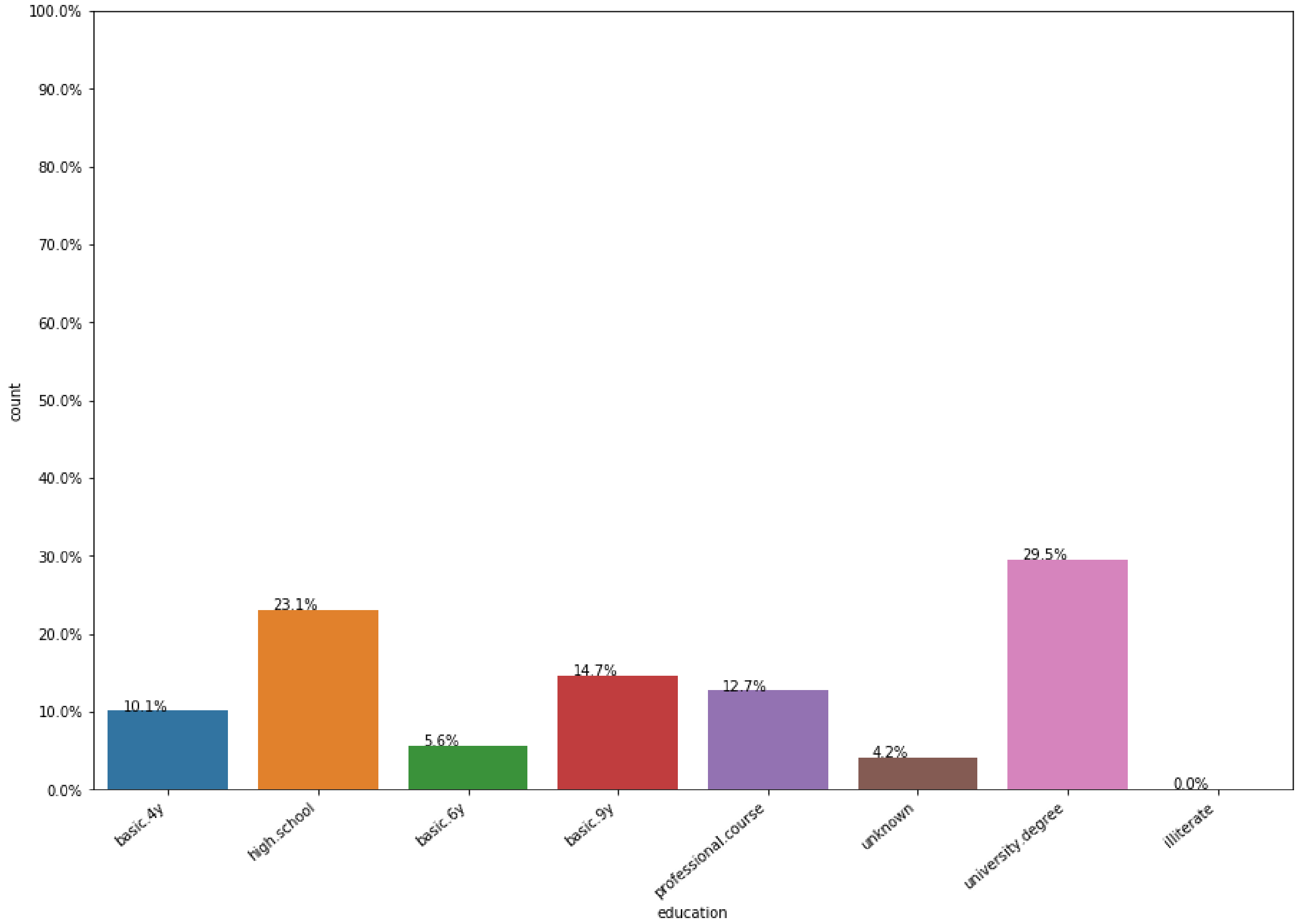
There is no customer with who has credit in default. Majority of the customers don't have, and the for the rest of the customers this

There is no customer with who has credit in default. Majority of the customers don't have, and the for the rest of the customers this field is unknown.

Feature: Education

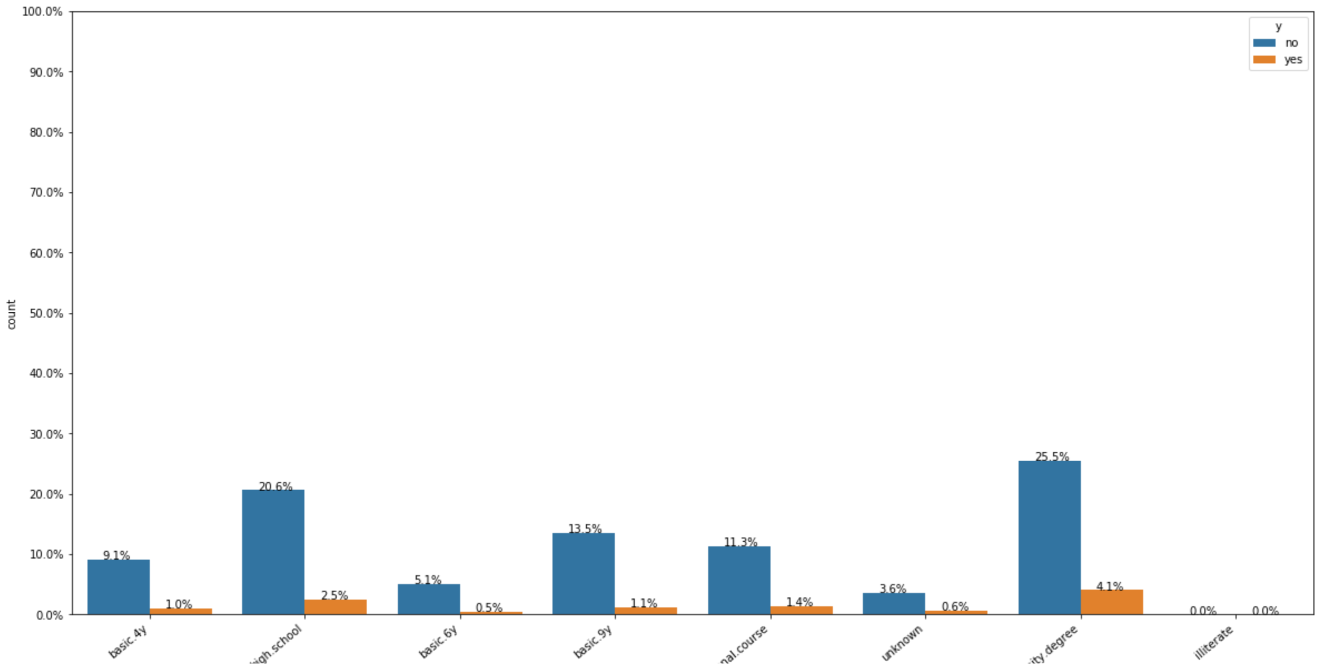
In [0]:

```
countplot("education",data)
```



In [0]:

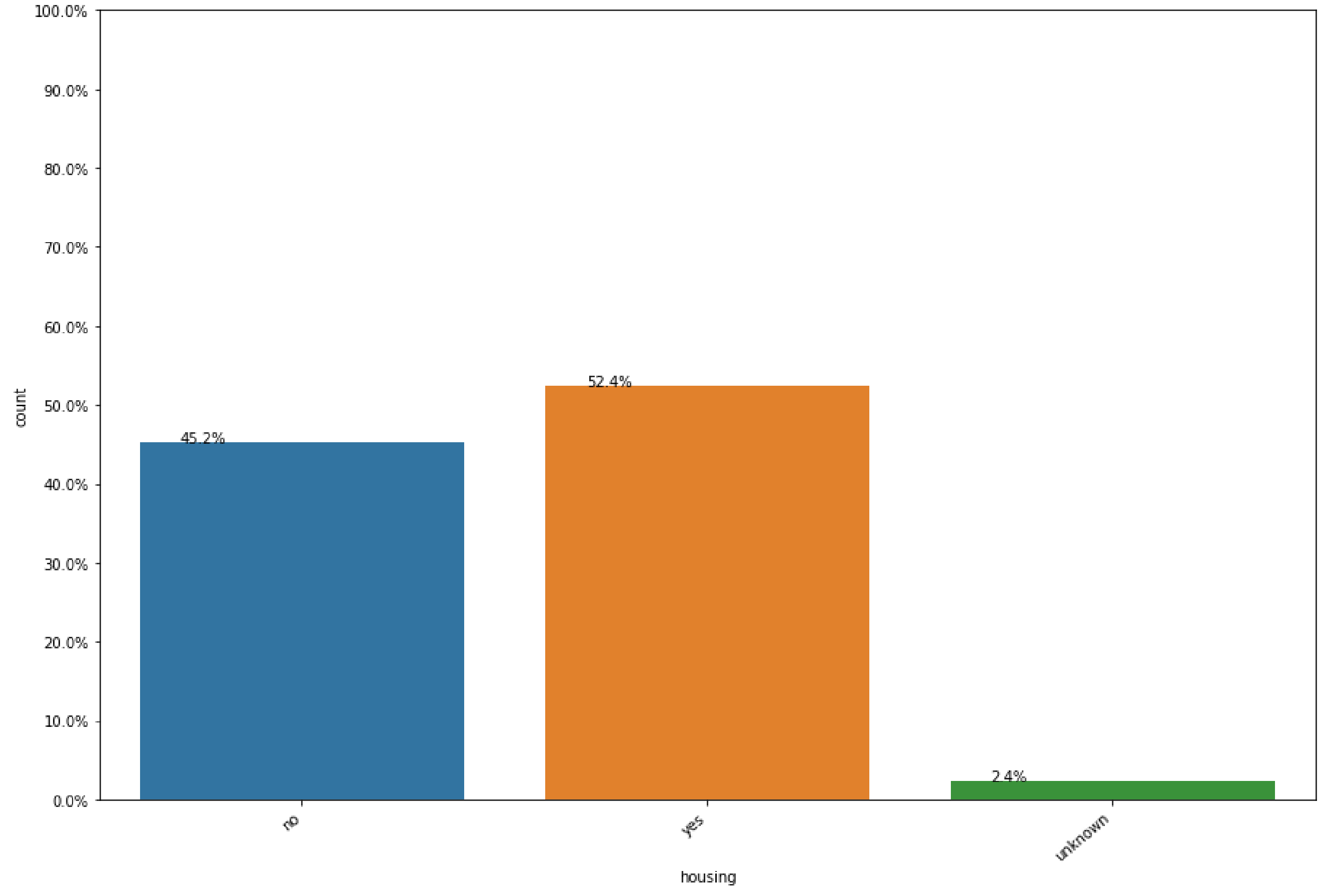
```
countplot_withY("education", data)
```



Feature: housing (Categorical)

In [0]:

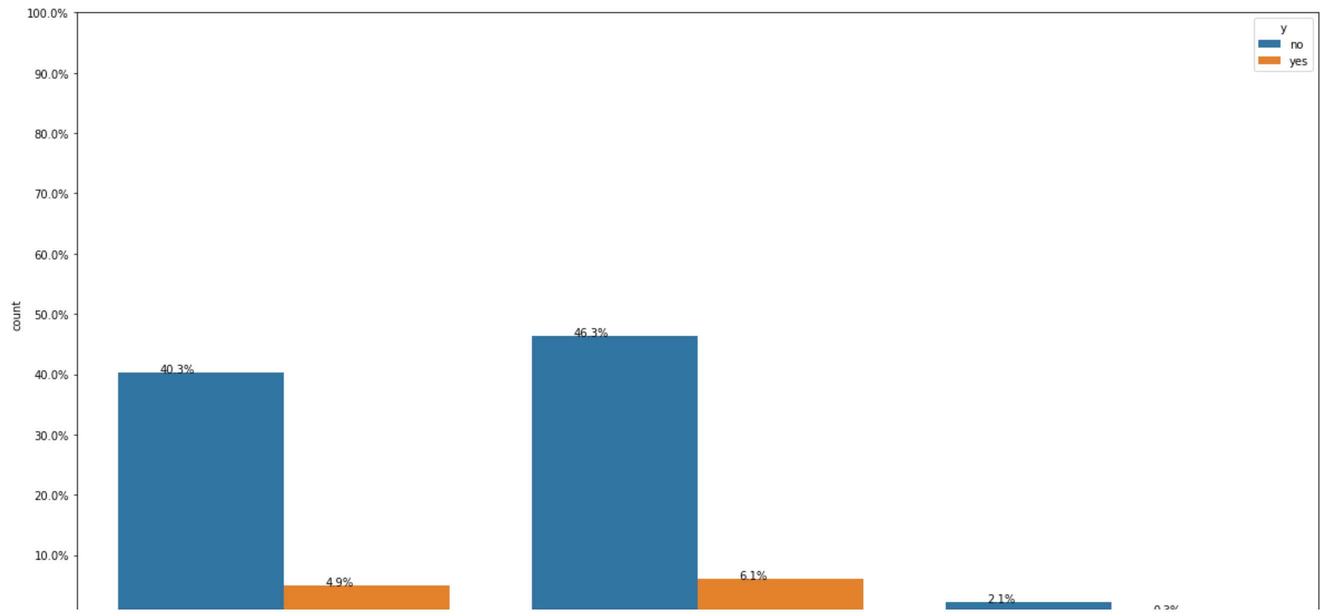
```
countplot("housing", data)
```

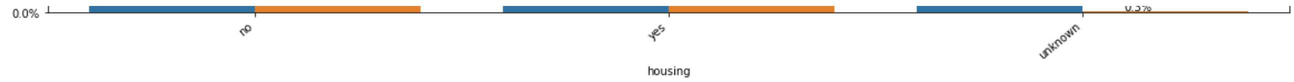


Majority of the customers have a housing loan.

In [0]:

```
countplot_withY("housing", data)
```

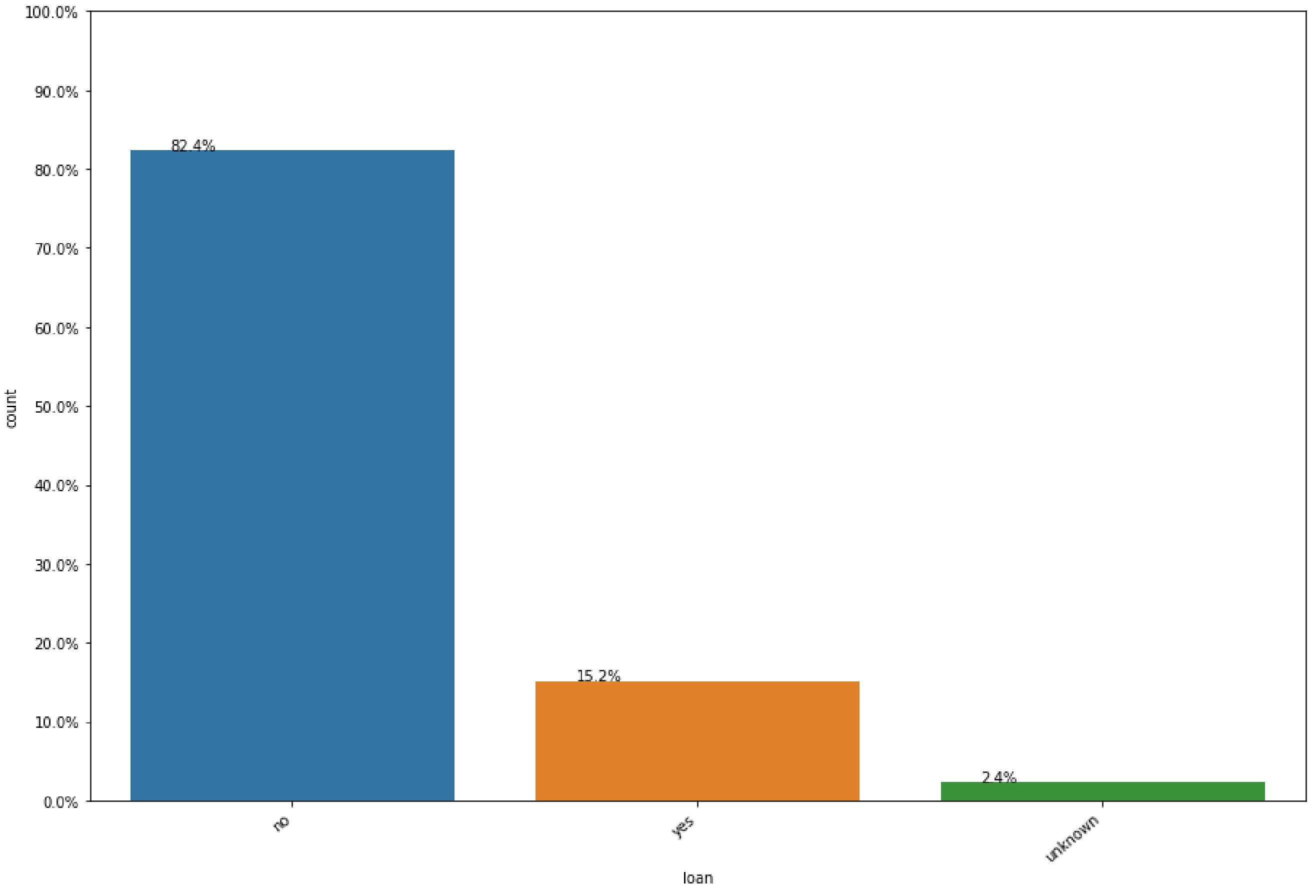




Feature: loan (Categorical)

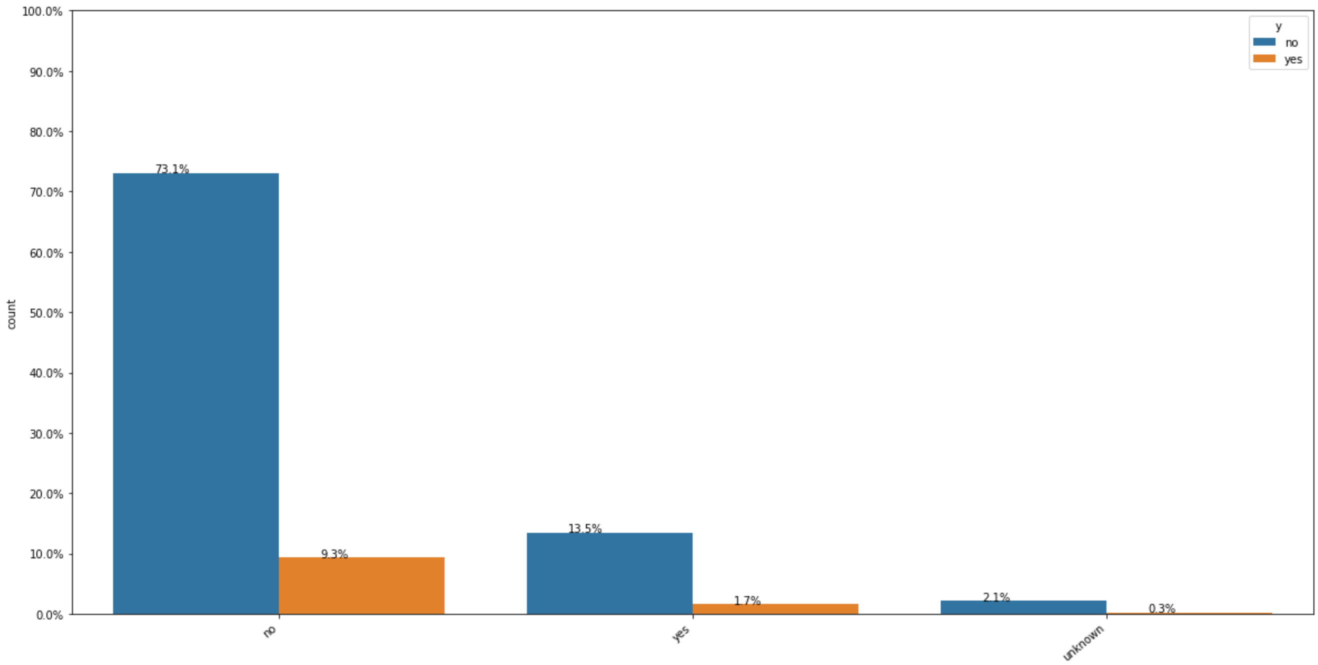
In [0]:

```
countplot("loan", data)
```



In [0]:

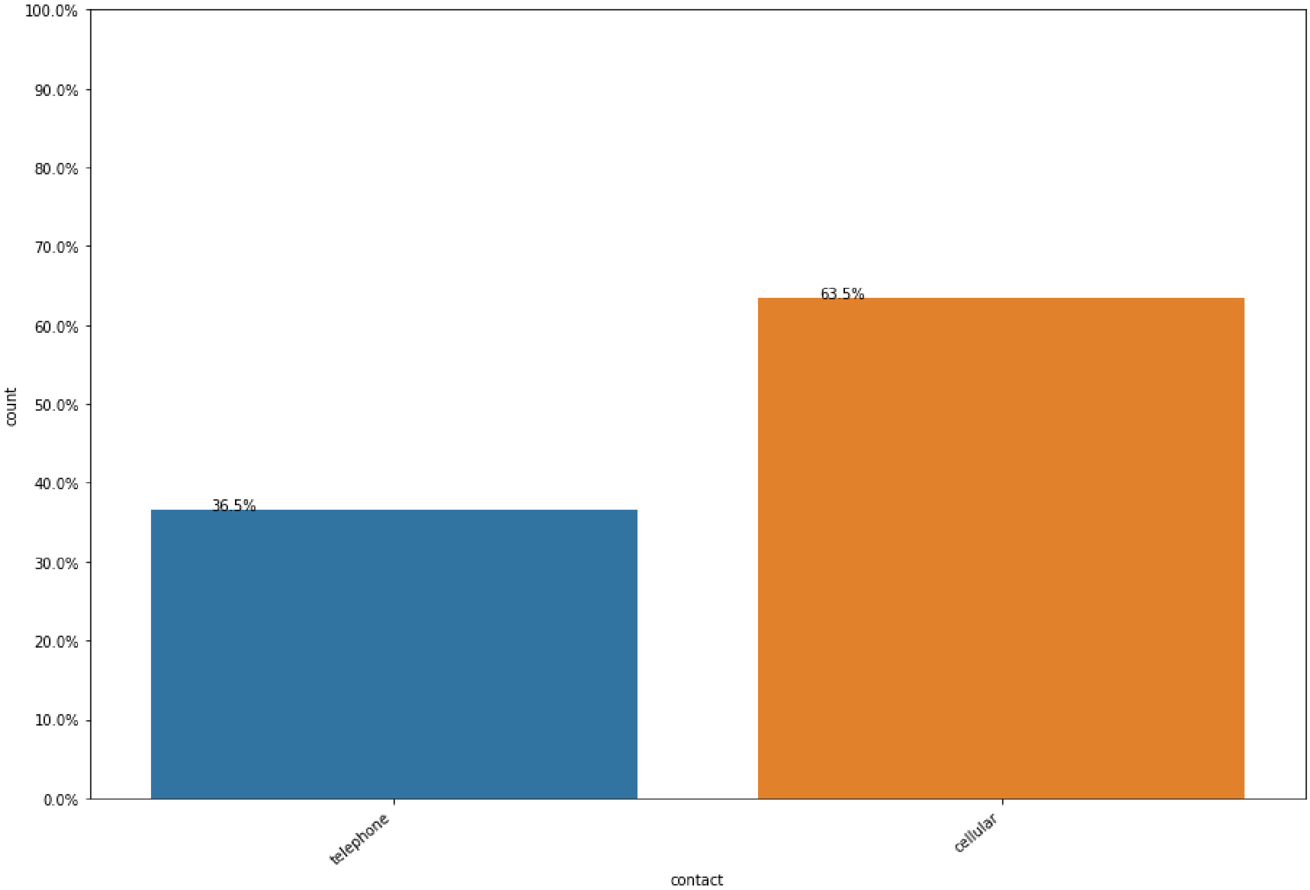
```
countplot_withY("loan", data)
```



Feature: contact (Categorical)

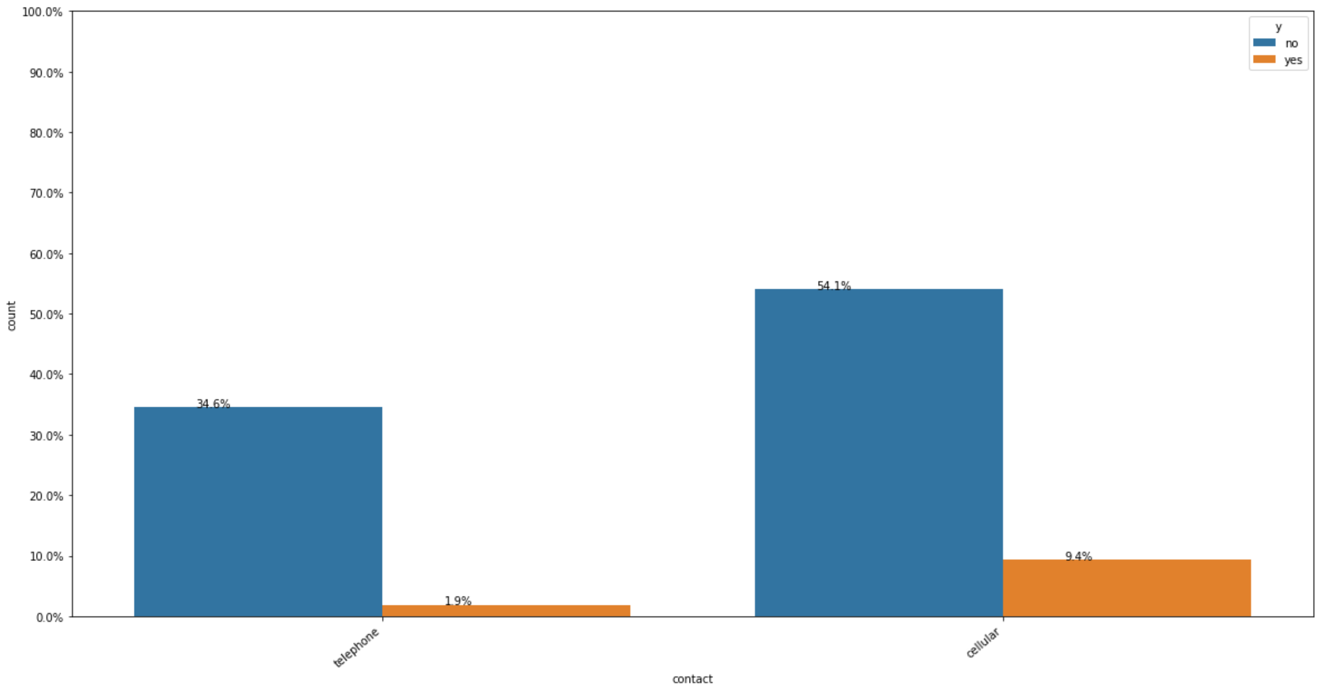
In [0]:

```
countplot("contact", data)
```



In [0]:

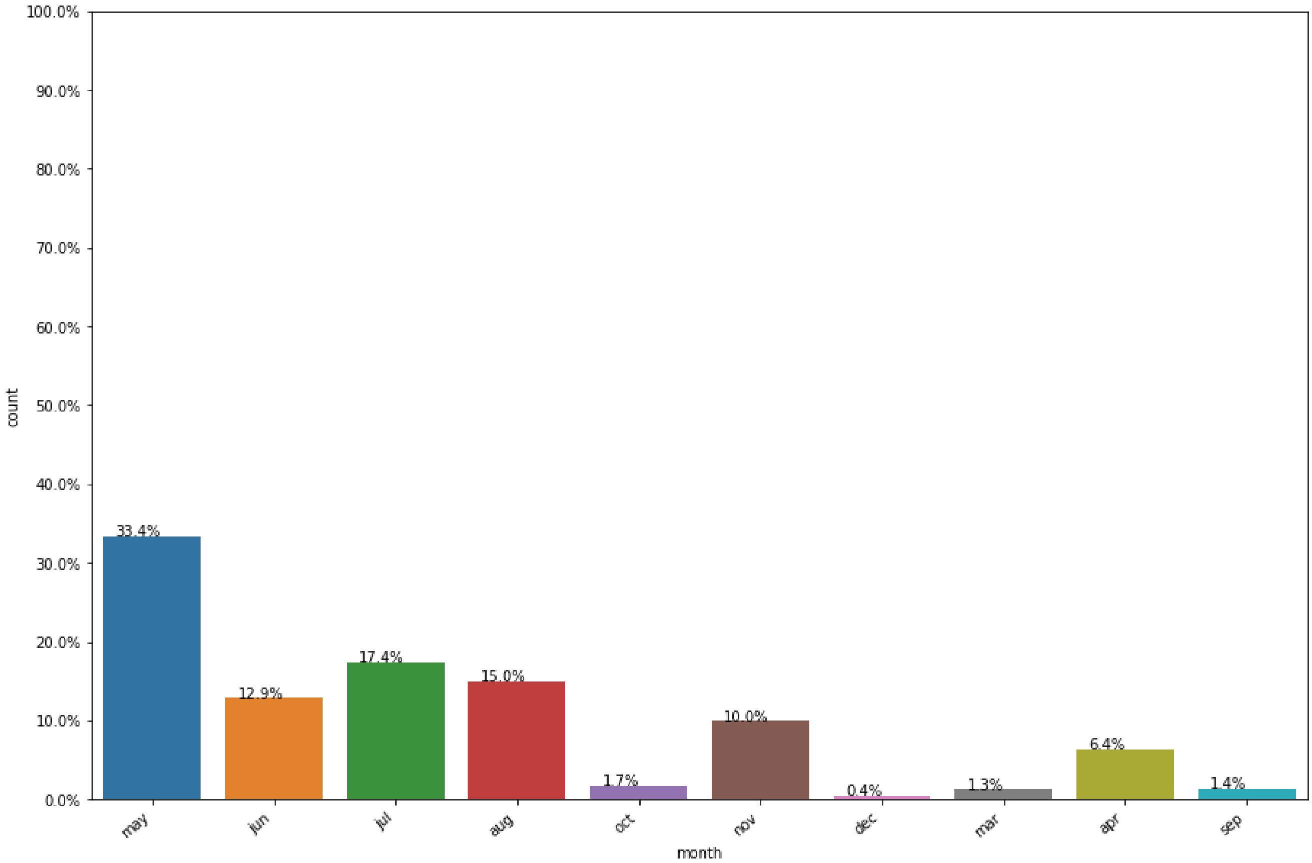
```
countplot_withY("contact", data)
```



Feature: month (Categorical)

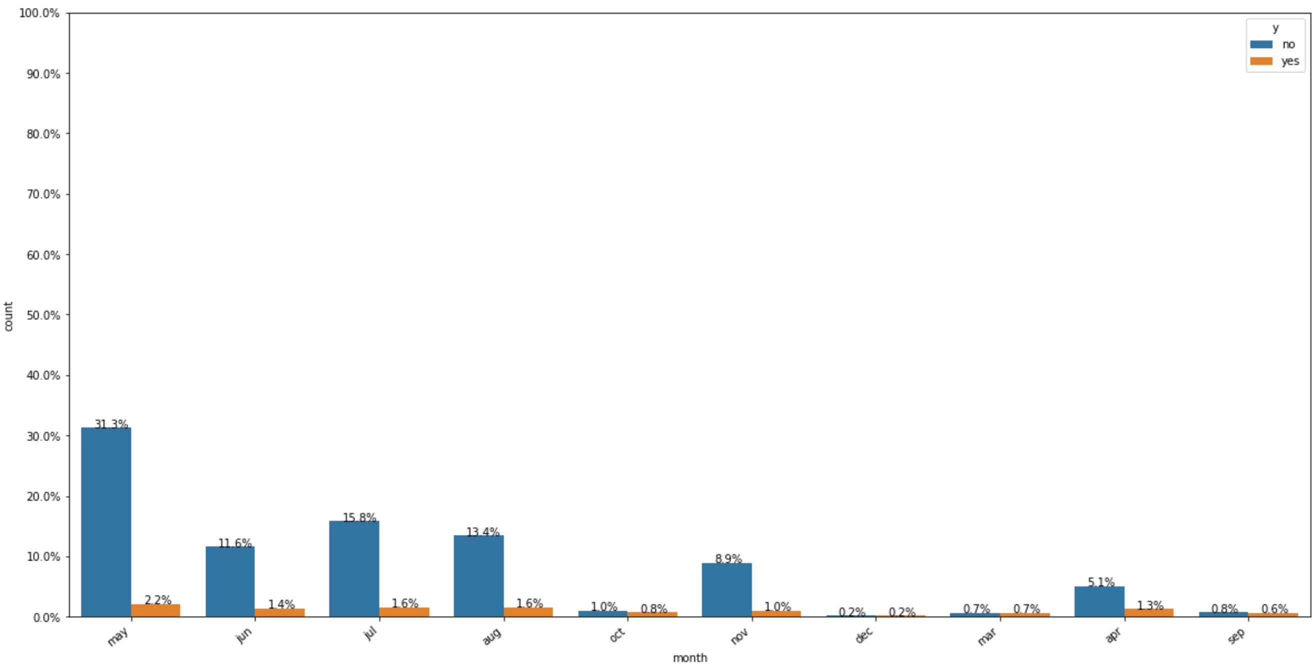
In [0]:

```
countplot("month", data)
```



In [0]:

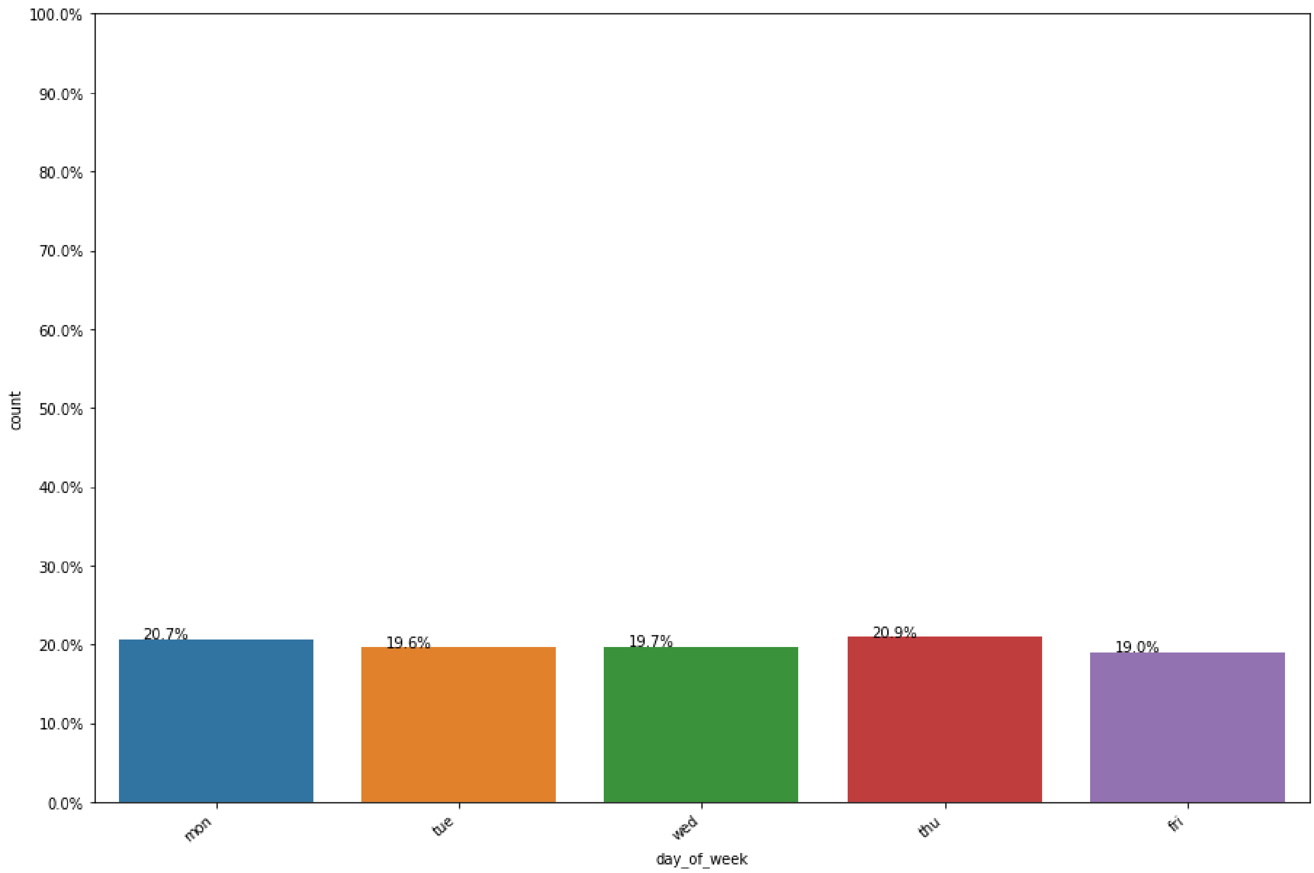
```
countplot_withY("month", data)
```



Feature: day_of_week (Categorical)

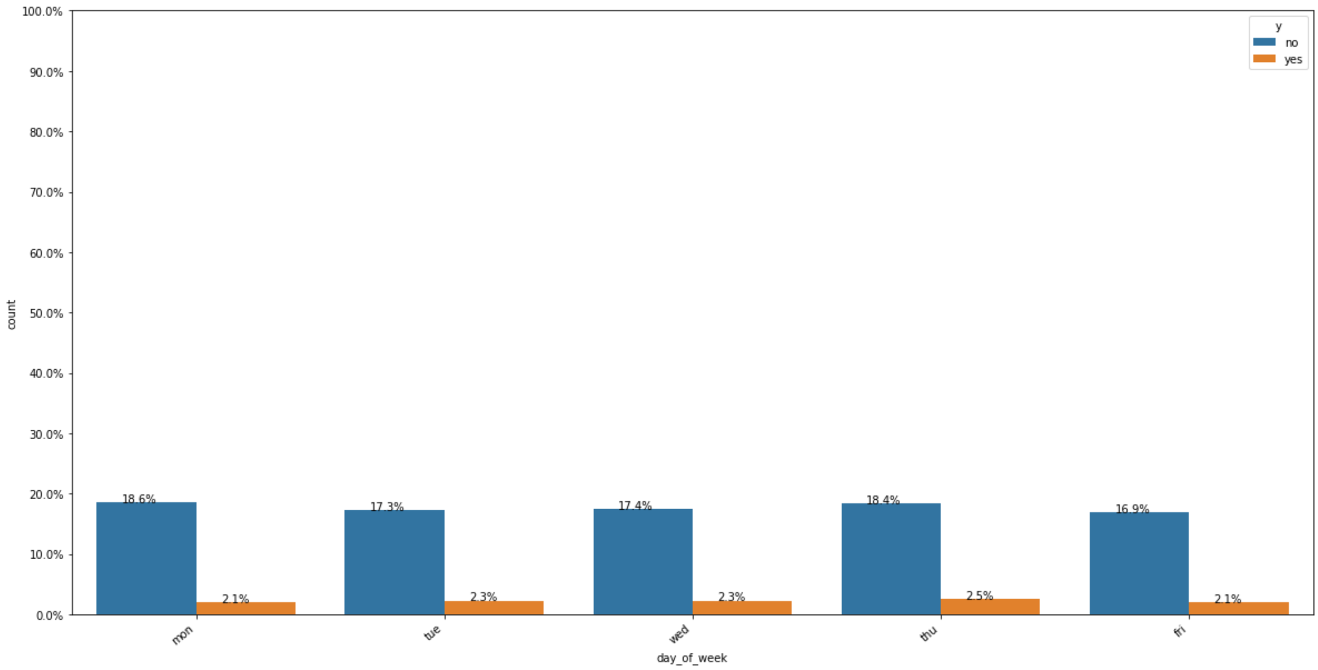
In [0]:

```
countplot("day_of_week", data)
```



In [0]:

```
countplot_withY("day_of_week", data)
```



The day of the week seems to be irrelevent as we have the same amount of data for all the days of the week, and no:yes ratio is also almost same.

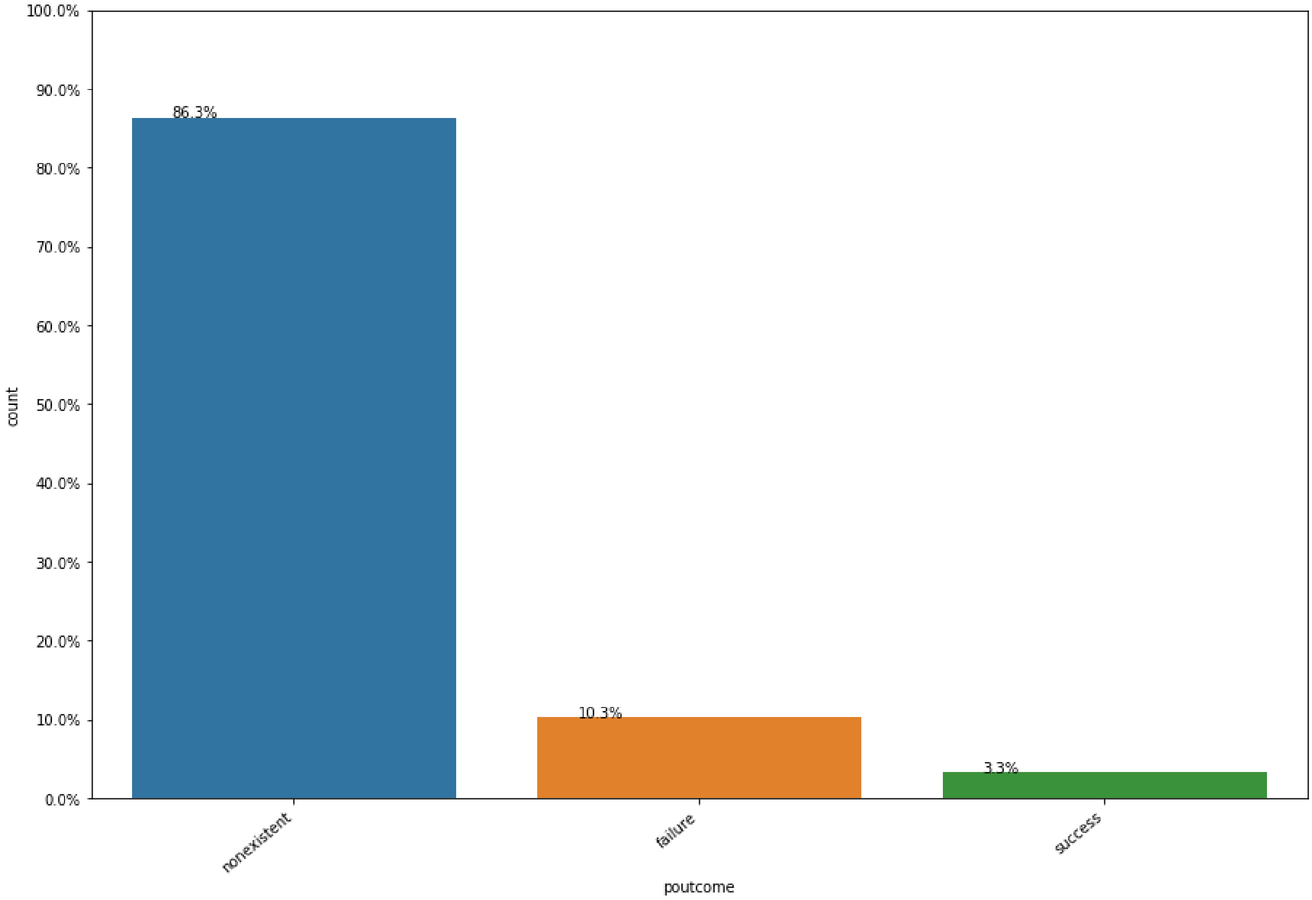
Feature: poutcome (Categorical)

Feature: poutcome (Categorical)

This feature indicates the outcome of the previous marketing campaign

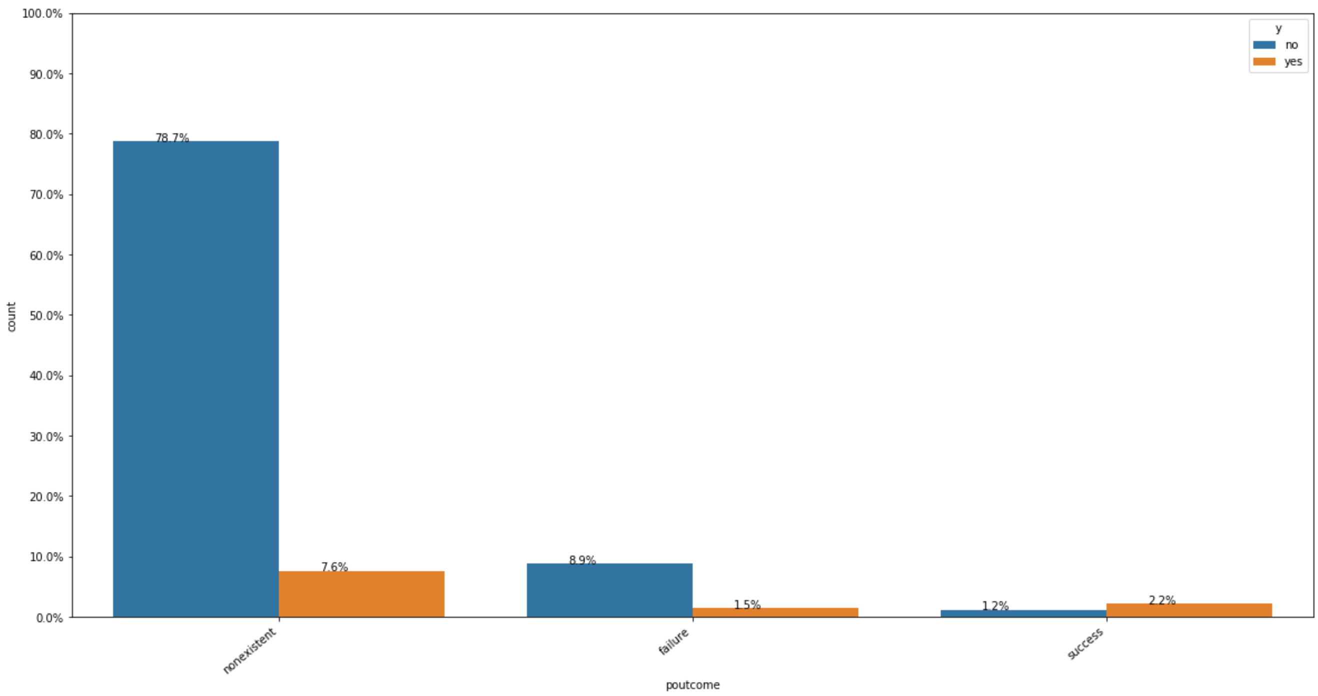
In [0]:

```
countplot("poutcome", data)
```



In [0]:

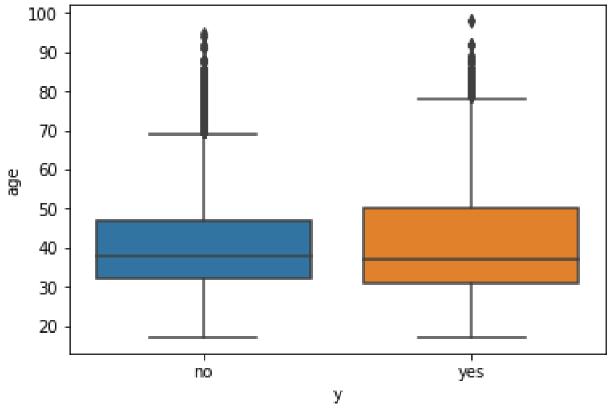
```
countplot_withY("poutcome", data)
```



Feature: Age (Numeric)

In [0]:

```
%matplotlib inline
sns.boxplot(data=data, x="y", y="age")
plt.show()
```



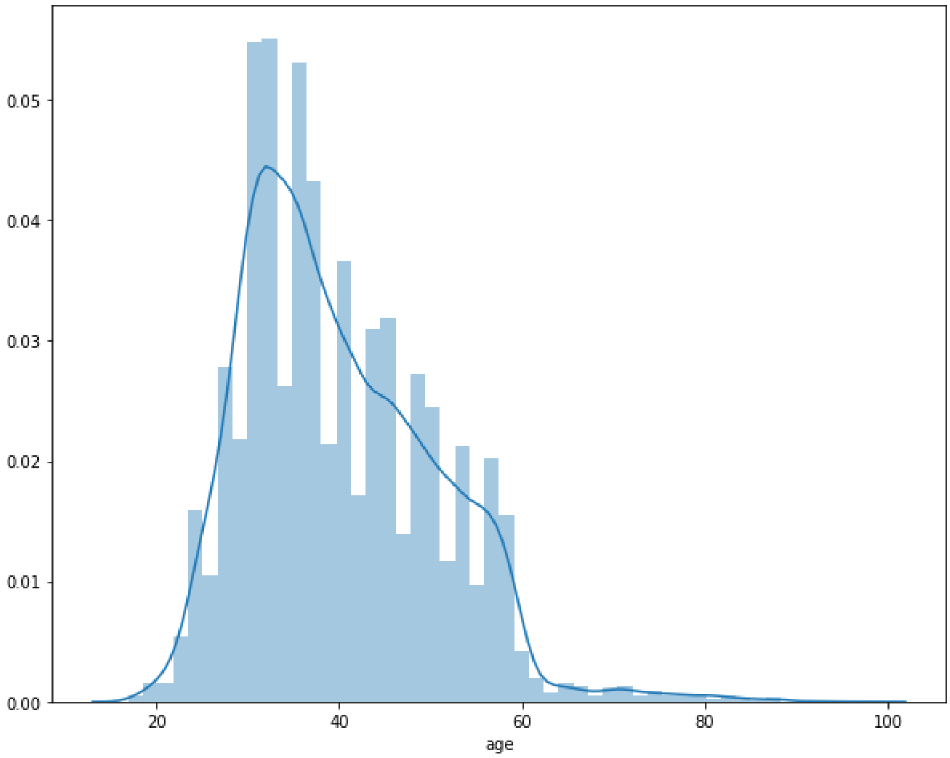
From the above boxplot we know that for both the customers that subscibed or didn't subscribe a term deposit, has a median age of around 38-40. And the boxplot for both the classes overlap quite a lot, which means that age isn't necessarily a good indicator for which customer will subscribe and which customer will not.

In [0]:

```
plt.figure(figsize=(10,8))
sns.distplot(data["age"])
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68dafa2278>
```

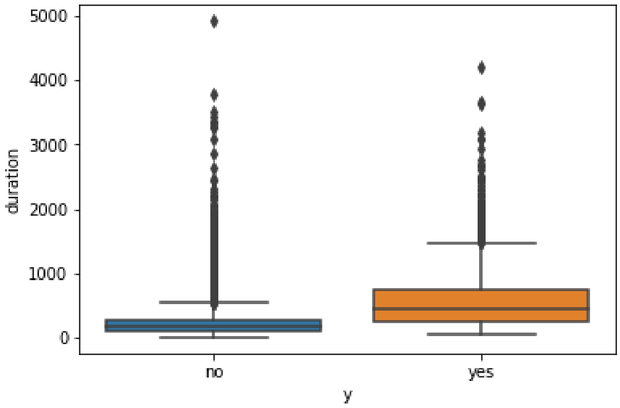


As we can see in the above distribution also, that most of the customers are in the age range of 30-40.

Feature: duration (numeric)

In [0]:

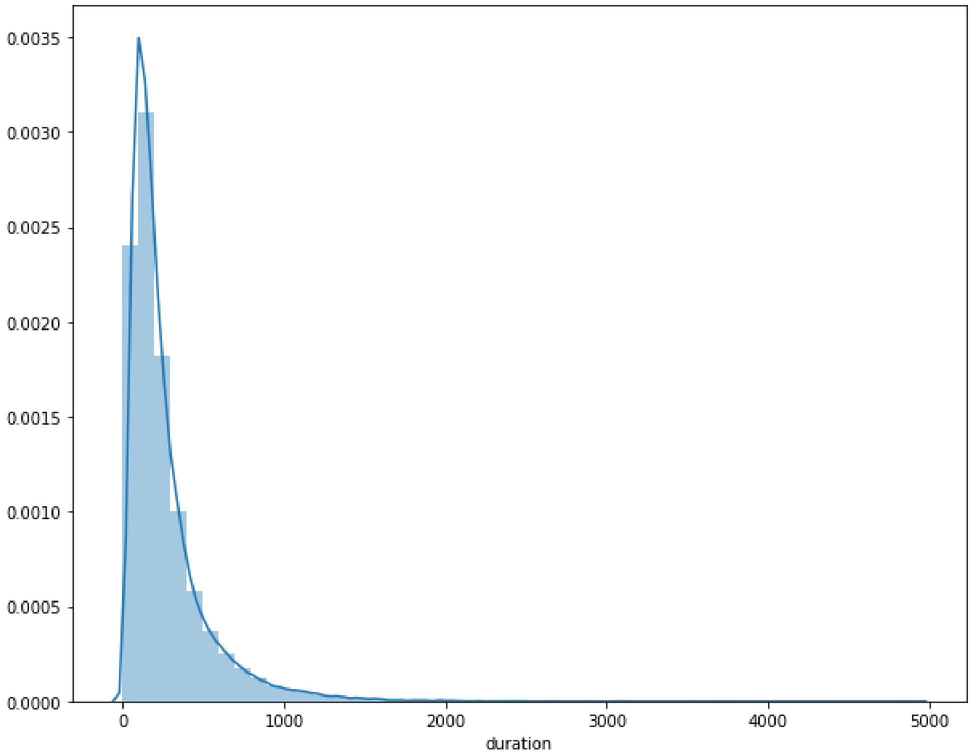
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="duration")
plt.show()
```



From the above plot it is clear that, the duration (last contact duration) of a customer can be useful for predicting the target variable. It is expected because it is already mentioned in the data overview that this field highely affects the target variable and should only be used for benchmark purposes.

In [0]:

```
plt.figure(figsize=(10,8))
sns.distplot(data["duration"])
plt.show()
```



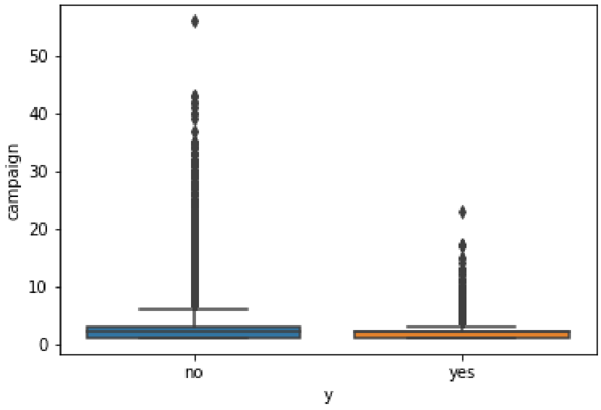
This seems like a powerlaw distribution where most the values are very low and very few have high values.

Feature: campaign (numeric)

In [0]:

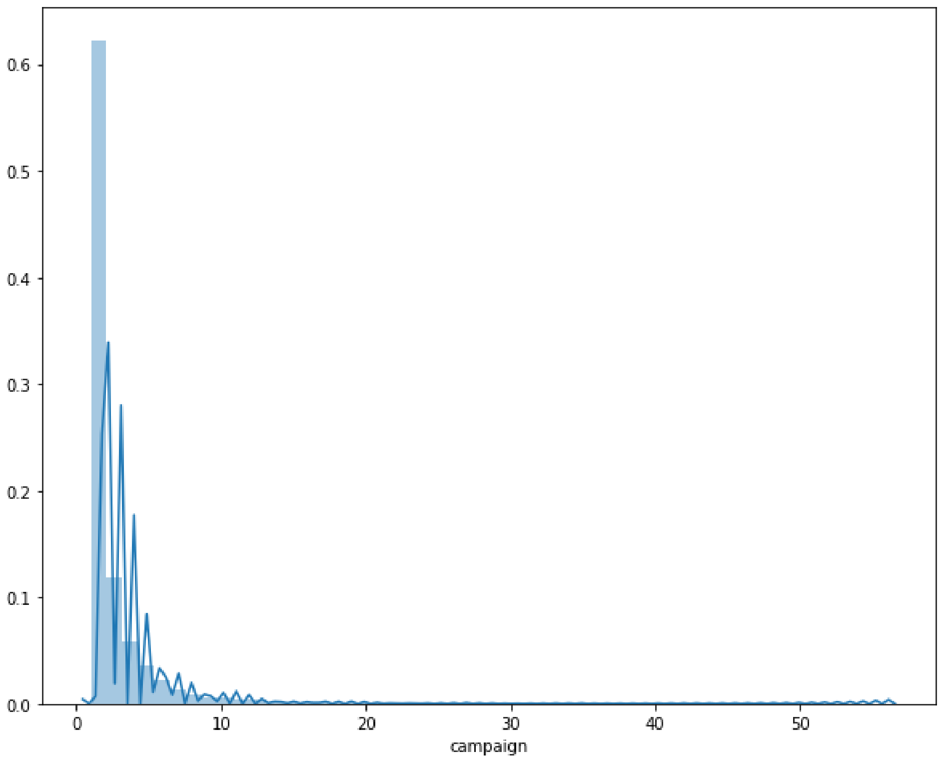
```
%matplotlib inline
```

```
sns.boxplot(data=data, x="y", y="campaign")
plt.show()
```



In [0]:

```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["campaign"])
plt.show()
```



Feature: pdays (numeric)

In [0]:

```
data["pdays"].unique()
```

Out[0]:

```
array([999,   6,   4,   3,   5,   1,   0,  10,   7,   8,   9,  11,   2,
        12,  13,  14,  15,  16,  21,  17,  18,  22,  25,  26,  19,  27,
        20])
```

In [0]:

```
data["pdays"].value_counts()
```

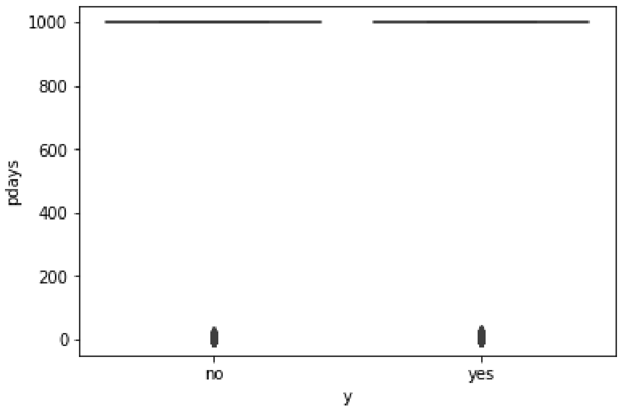
Out[0]:

```
999    39673
3        439
6        412
4        118
9         64
2         61
7         60
12        58
10        52
5         46
13        36
11        28
1         26
15        24
14        20
8         18
0         15
16        11
17         8
18         7
19         3
22         3
21         2
26         1
20         1
25         1
27         1
Name: pdays, dtype: int64
```

Most of the values are 999, which means that the most of the customers have never been contacted before.

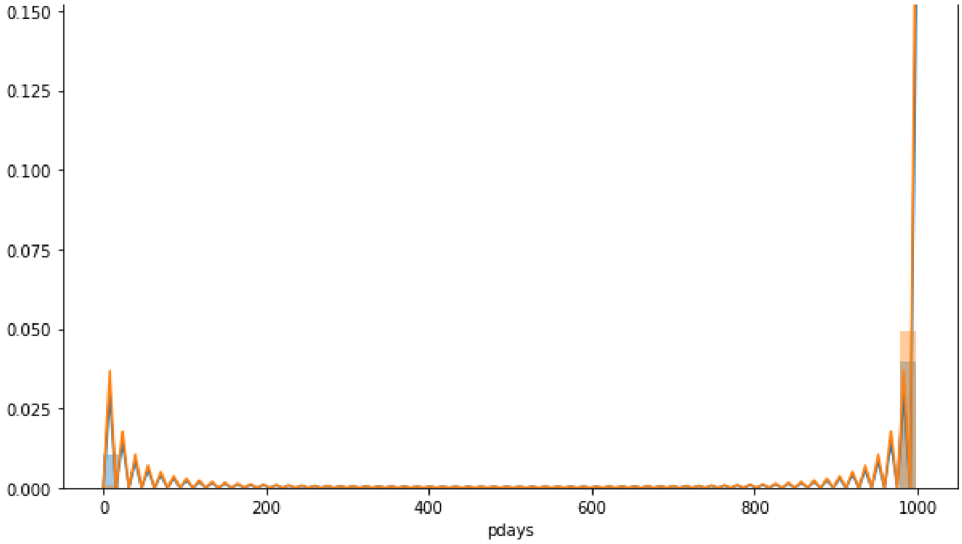
In [0]:

```
%matplotlib inline
sns.boxplot(data=data, x="y", y="pdays")
plt.show()
```



In [0]:

```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data[data["y"]=="yes"]["pdays"])
sns.distplot(data[data["y"]=="no"]["pdays"])
plt.show()
```



Feature: previous (numeric)

In [0]:

```
data["previous"].unique()
```

Out[0]:

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [0]:

```
data["previous"].value_counts()
```

Out[0]:

```
0    35563
1     4561
2      754
3      216
4       70
5       18
6        5
7        1
Name: previous, dtype: int64
```

In [0]:

```
data[data["y"]=="yes"]["previous"].value_counts()
```

Out[0]:

```
0    3141
1     967
2     350
3     128
4      38
5      13
6       3
Name: previous, dtype: int64
```

In [0]:

```
data[data["y"]=="no"]["previous"].value_counts()
```

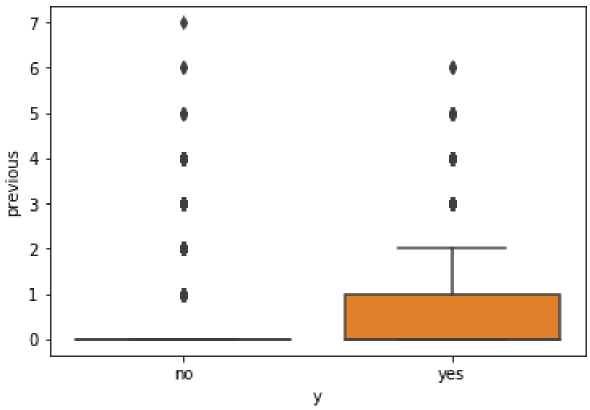
Out[0]:

```
0    32422
1     3594
2      404
```

```
2        404
3         88
4         32
5          5
6          2
7          1
Name: previous, dtype: int64
```

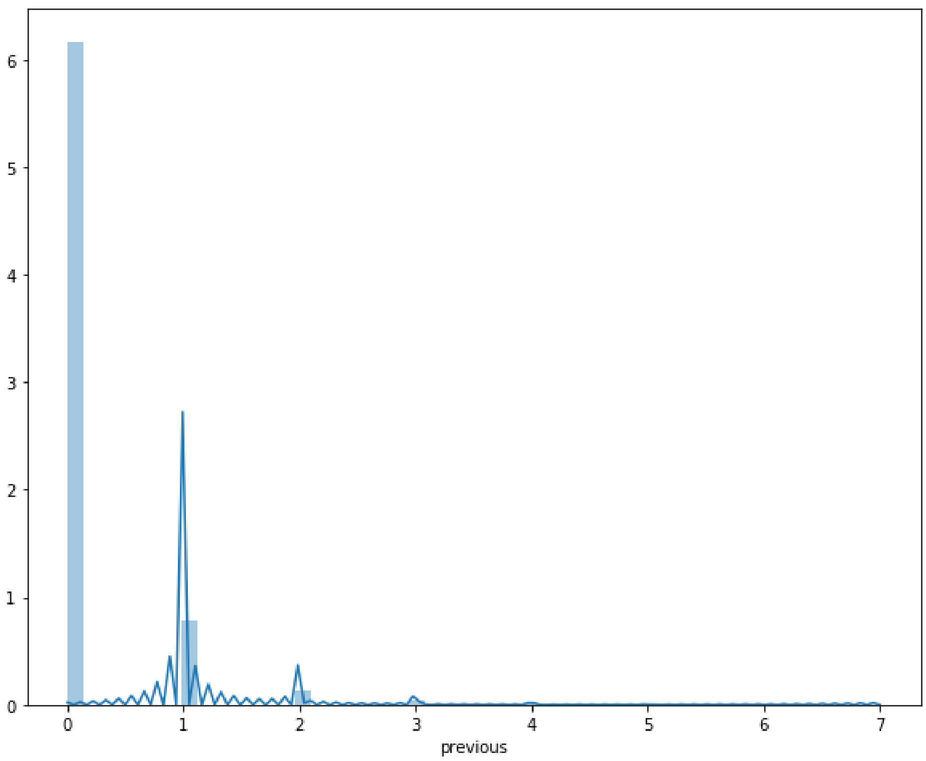
In [0]:

```
%matplotlib inline
sns.boxplot(data=data, x="y", y="previous")
plt.show()
```



In [0]:

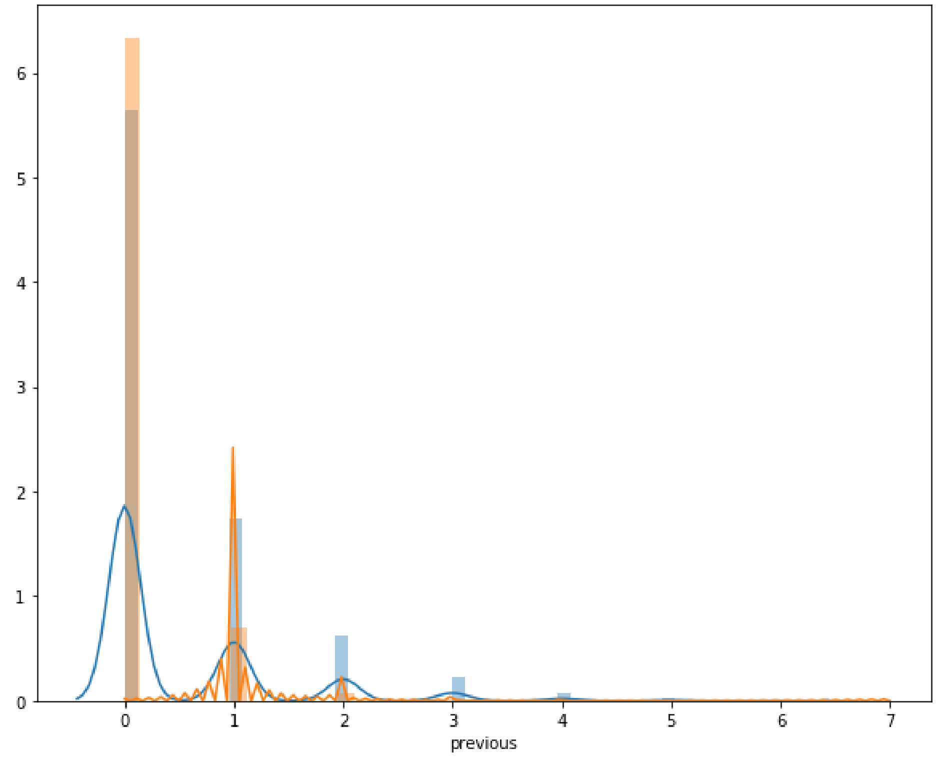
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["previous"])
plt.show()
```



In [0]:

```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data[data["y"]=="yes"]["previous"])
sns.distplot(data[data["y"]=="no"]["previous"])
plt.show()
```

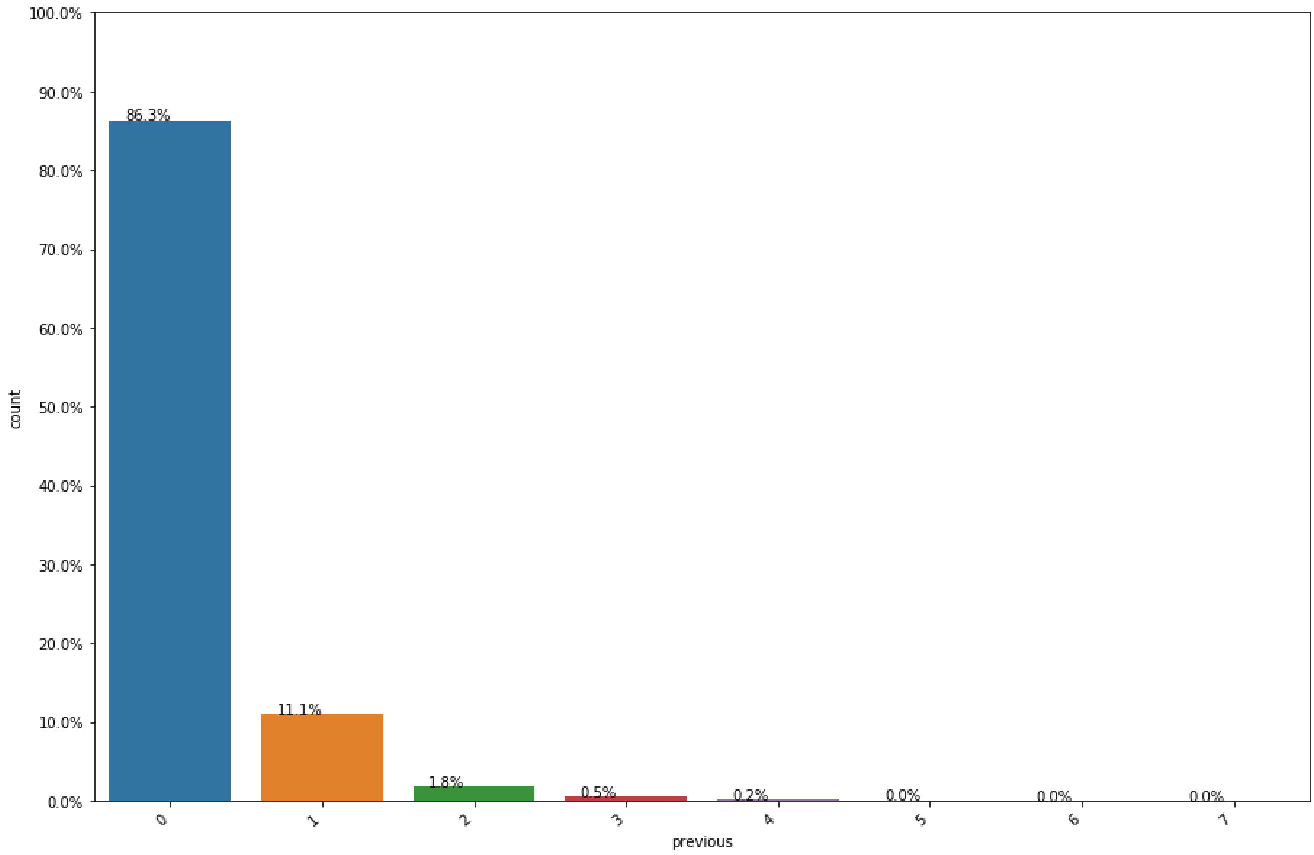
```
plt.show()
```



The previous feature is very similarly distributed for both the classes in the target variable. From basic EDA it is not sure how much value this individual feature have on the target variable.

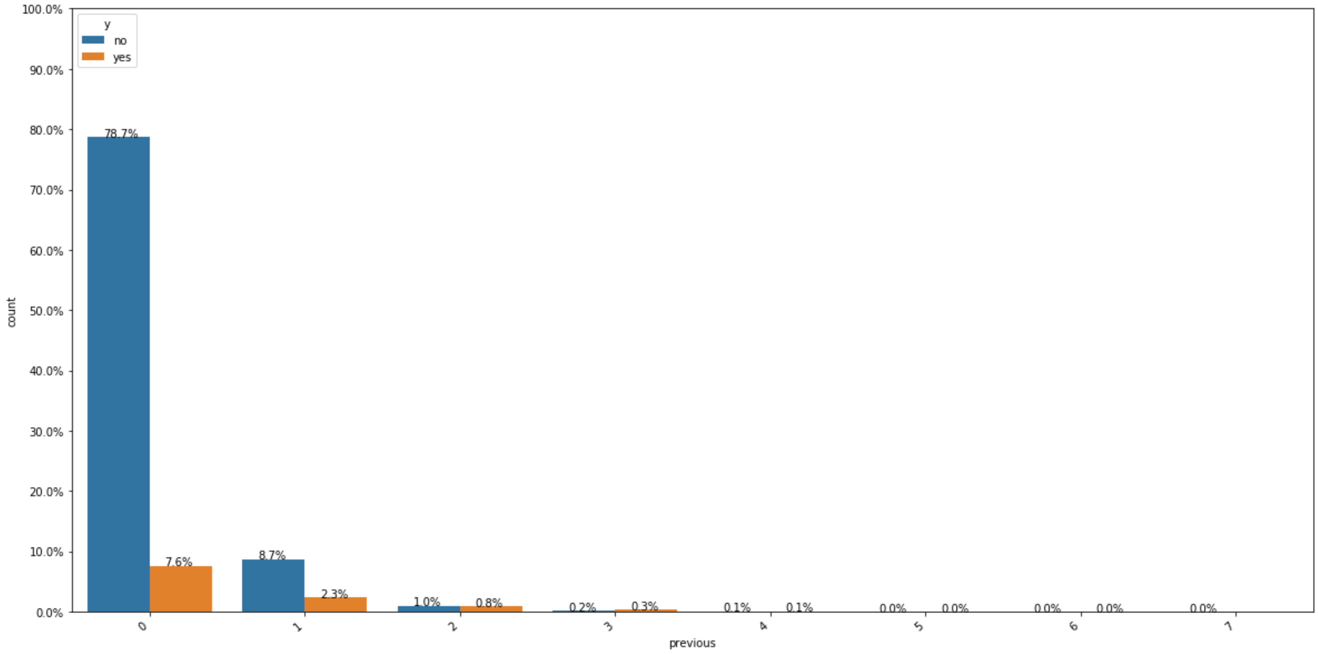
In [0]:

```
countplot("previous", data)
```



In [0]:

```
countplot_withY("previous", data)
```



emp.var.rate

In [0]:

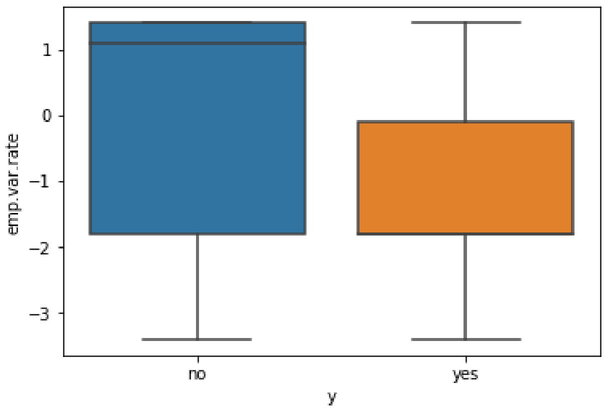
```
data["emp.var.rate"].value_counts()
```

Out[0]:

```
 1.4    16234
-1.8     9184
 1.1     7763
-0.1     3683
-2.9     1663
-3.4     1071
-1.7      773
-1.1      635
-3.0      172
-0.2       10
Name: emp.var.rate, dtype: int64
```

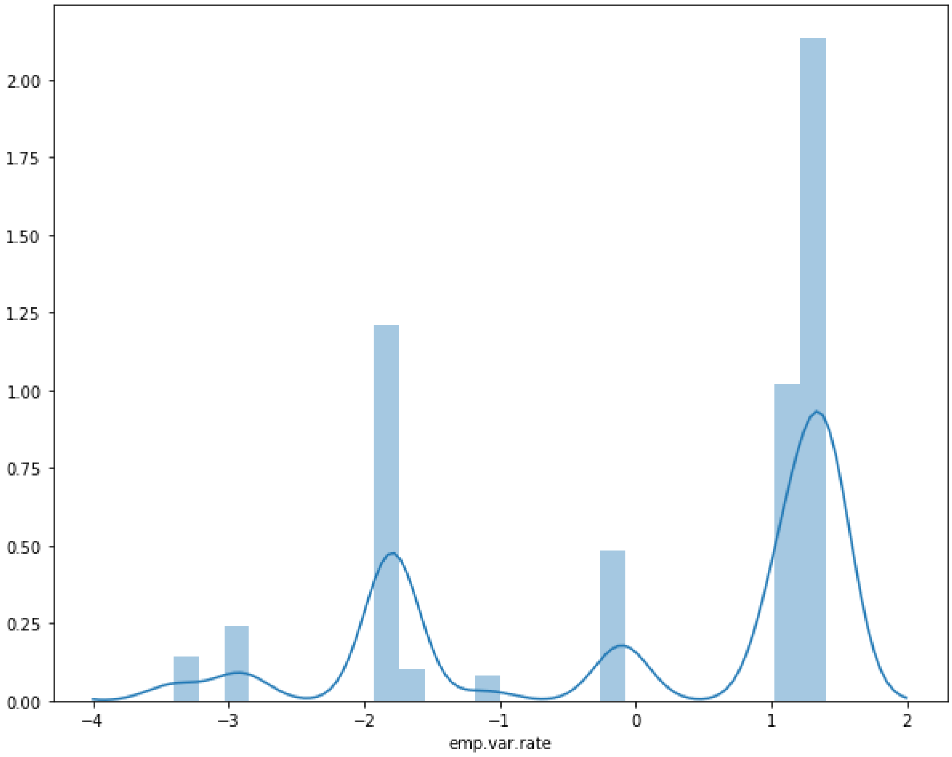
In [0]:

```
%matplotlib inline
sns.boxplot(data=data, x="y", y="emp.var.rate")
plt.show()
```



In [0]:

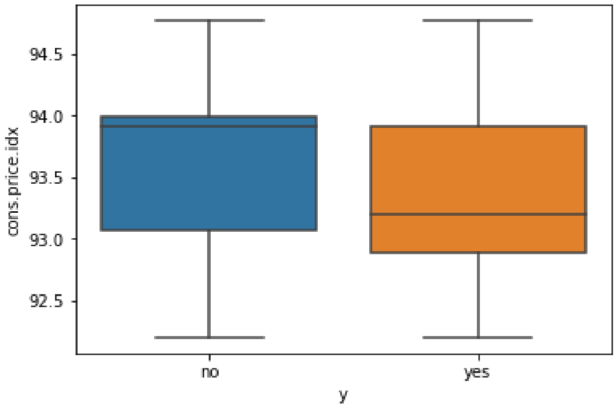
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["emp.var.rate"])
plt.show()
```



cons.price.idx

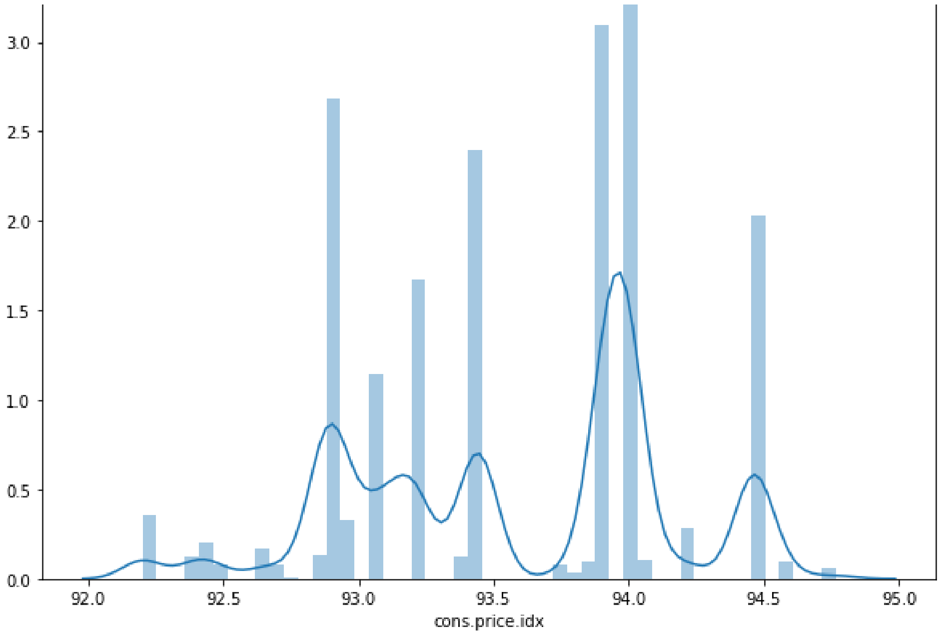
In [0]:

```
%matplotlib inline
sns.boxplot(data=data, x="y", y="cons.price.idx")
plt.show()
```



In [0]:

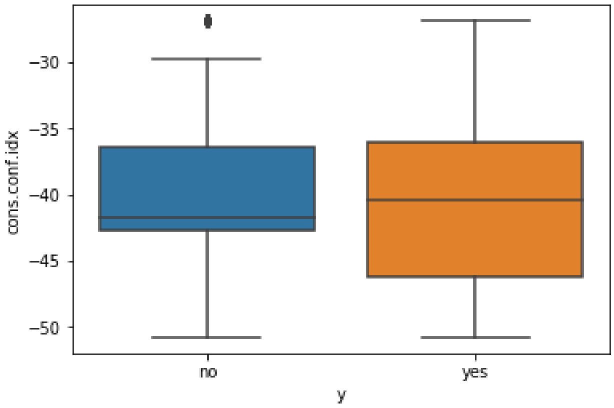
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["cons.price.idx"])
plt.show()
```



cons.conf.idx

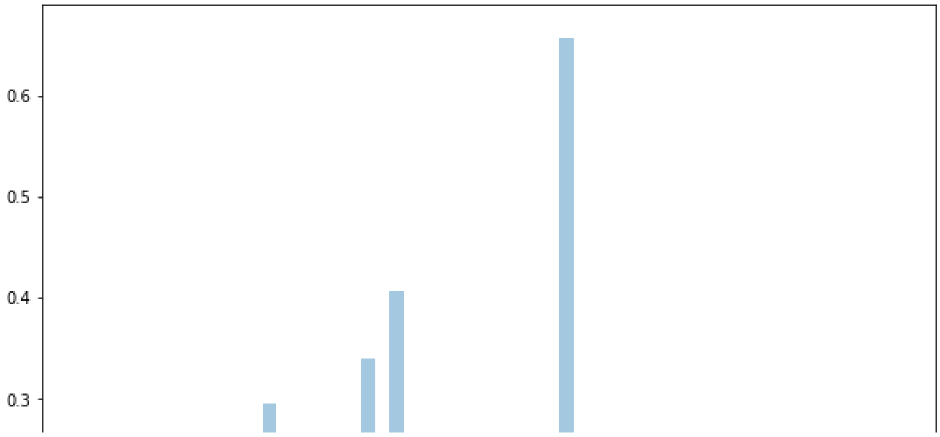
In [0]:

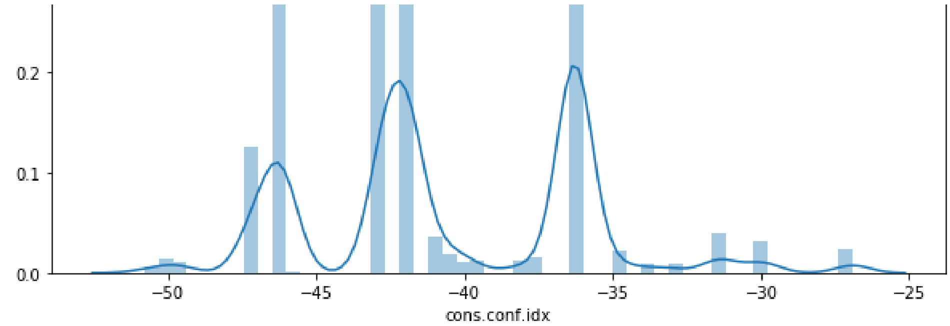
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="cons.conf.idx")
plt.show()
```



In [0]:

```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["cons.conf.idx"])
plt.show()
```

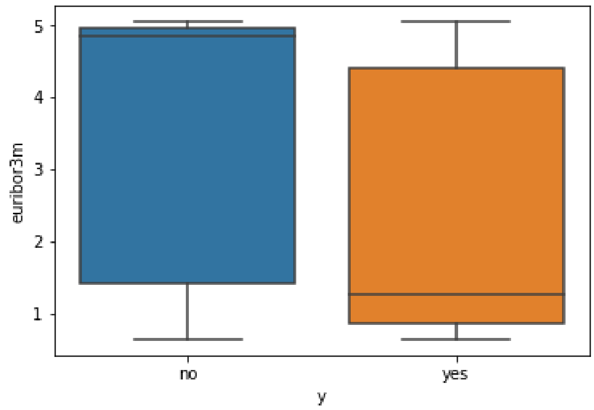




euribor3m

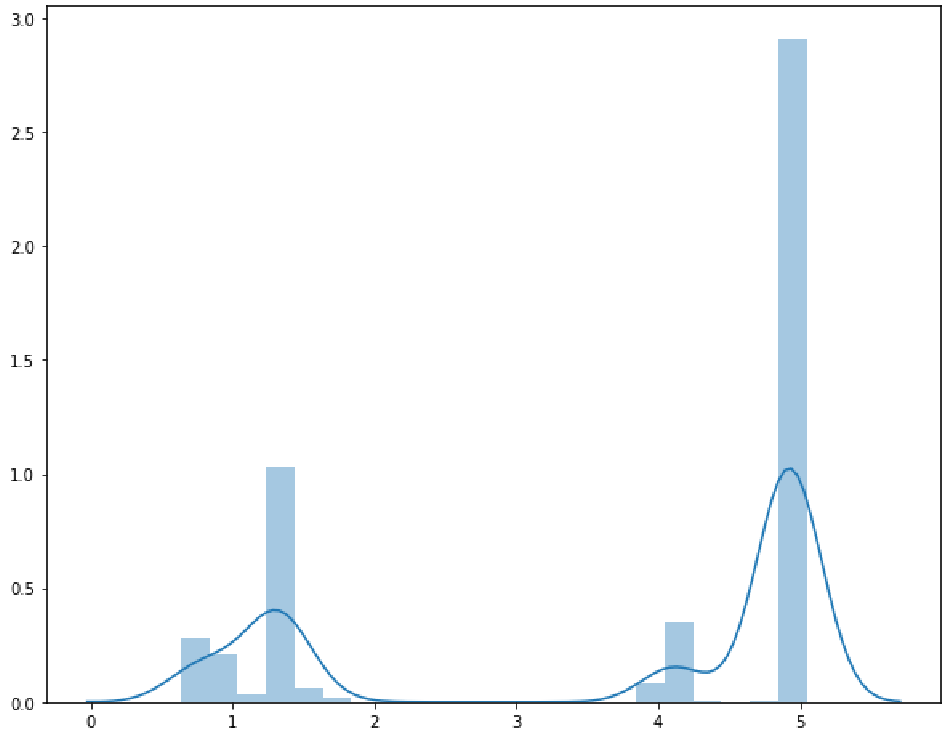
In [0]:

```
%matplotlib inline
sns.boxplot(data=data, x="y", y="euribor3m")
plt.show()
```



In [0]:

```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["euribor3m"])
plt.show()
```

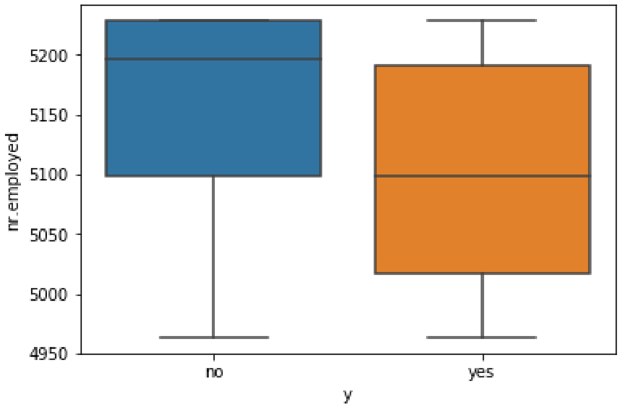


euribor3m

nr.employed

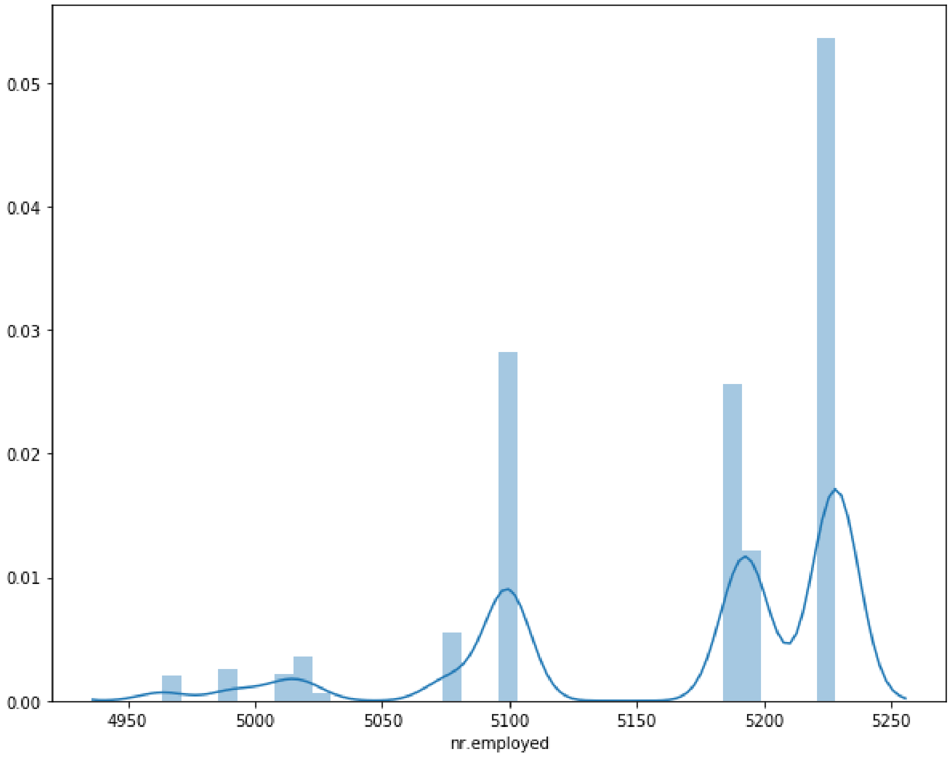
In [0]:

```
%matplotlib inline
sns.boxplot(data=data, x="y", y="nr.employed")
plt.show()
```



In [0]:

```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["nr.employed"])
plt.show()
```



Correlation matrix of numerical features

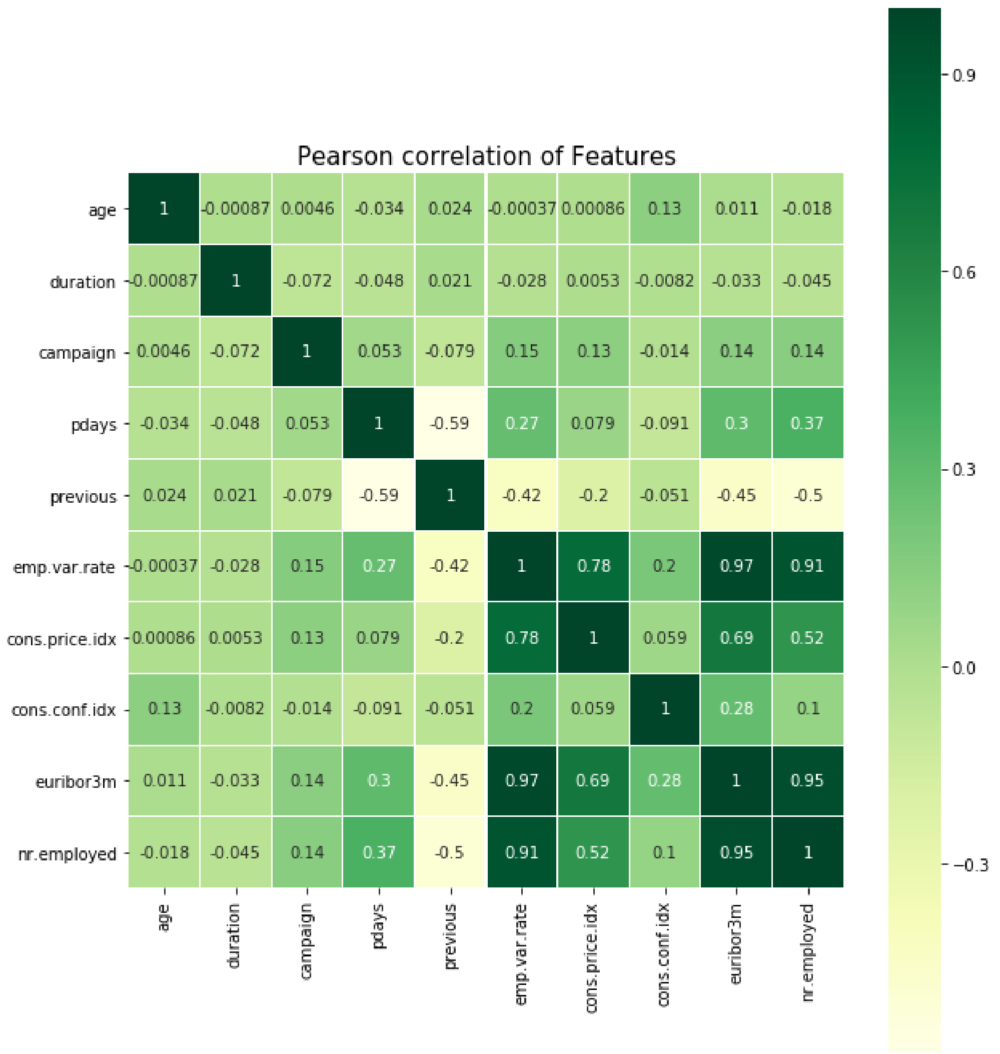
In [0]:

```
# Idea of correlation matrix of numerical feature:
https://medium.com/datadriveninvestor/introduction-to-exploratory-data-analysis-682eb64063ff
%matplotlib inline
```

```
corr = data.corr()

f, ax = plt.subplots(figsize=(10,12))

cmap = sns.diverging_palette(220, 10, as_cmap=True)

_ = sns.heatmap(corr, cmap="YlGn", square=True, ax=ax, annot=True, linewidth=0.1)

plt.title("Pearson correlation of Features", y=1.05, size=15)
```

Out[0]:

```
Text(0.5, 1.05, 'Pearson correlation of Features')
```



From the above heatmap we can see that there are some numerical features which share a high correlation between them, e.g nr.employed and euribor3m these features share a correlation value of 0.95, and euribor3m and emp.var.rate share a correlation of 0.97, which is very high compared to the other features that we see in the heatmap.

Data Preprocessing

In [0]:

```
# Import the libraries
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
```

```
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd

from sklearn.linear_model import SGDClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In [0]:

```
# Loading the dataset
data = pd.read_csv("/content/drive/My Drive/BankMarketing_CaseStudy/bank-additional/bank-additional-full.csv", sep=";")
# data = data.drop_duplicates()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
duration          41188 non-null int64
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
nr.employed       41188 non-null float64
y                 41188 non-null object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

Dealing with Missing data

From the above basic info of each feature, we know that there are no missing values in this dataset.

Dealing with duplicate data

In [0]:

```
data_dup = data[data.duplicated(keep="last")]
data_dup
```

Out[0]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pday
1265	39	blue-collar	married	basic.6y	no	no	no	telephone	may	thu	124	1	9
12260	36	retired	married	unknown	no	no	no	telephone	jul	thu	88	1	9
14155	27	technician	single	professional.course	no	no	no	cellular	jul	mon	331	2	9

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays
16819	47	technician	divorced	high.school	no	yes	no	cellular	jul	thu	43	3	99
18464	32	technician	single	professional.course	no	yes	no	cellular	jul	thu	128	1	99
20072	55	services	married	high.school	unknown	no	no	cellular	aug	mon	33	1	99
20531	41	technician	married	professional.course	no	yes	no	cellular	aug	tue	127	1	99
25183	39	admin.	married	university.degree	no	no	no	cellular	nov	tue	123	2	99
28476	24	services	single	high.school	no	yes	no	cellular	apr	tue	114	1	99
32505	35	admin.	married	university.degree	no	yes	no	cellular	may	fri	348	4	99
36950	45	admin.	married	university.degree	no	no	no	cellular	jul	thu	252	1	99
38255	71	retired	single	university.degree	no	no	no	telephone	oct	tue	120	1	99

In [0]:

```
data_dup.shape
```

Out[0]:

```
(12, 21)
```

So we have 12 rows which are duplicates. We will drop these duplicate rows before proceeding furthur.

In [0]:

```
data = data.drop_duplicates()
data.shape
```

Out[0]:

```
(41176, 21)
```

Separate inpedendent and target variables

In [0]:

```
data_x = data.iloc[:, :-1]
print("Shape of X:", data_x.shape)
data_y = data["y"]
print("Shape of Y:", data_y.shape)
```

```
Shape of X: (41176, 20)
Shape of Y: (41176,)
```

Train Test split

In [0]:

```
from sklearn.model_selection import train_test_split

X_rest, X_test, y_rest, y_test = train_test_split(data_x, data_y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_rest, y_rest, test_size=0.2)

print("X Train:", X_train.shape)
print("X CV:", X_cv.shape)
print("X Test:", X_test.shape)
print("Y Train:", y_train.shape)
print("Y CV:", y_cv.shape)
print("Y Test:", y_test.shape)
```

```
X Train: (26352, 20)
X CV: (6588, 20)
X Test: (8236, 20)
Y Train: (26352,)
Y CV: (6588,)
Y Test: (8236,)
```

Y Test: (8236,)

In [0]:

```
# Replace "no" with 0 and "yes" with 1

y_train.replace({"no":0, "yes":1}, inplace=True)
y_cv.replace({"no":0, "yes":1}, inplace=True)
y_test.replace({"no":0, "yes":1}, inplace=True)
```

Encoding Categorical Features

For this case study I will encode categorical features using two methods:

- One hot encoding
- Response coding

And compare the results to see which encoding method performed better for the data that we have.

One Hot Encoding Categorical features

The next big step for our data preprocessing is to encode all the categorical features so that we can apply models on the data.

In [0]:

```
# Categorical boolean mask
categorical_feature_mask = data_x.dtypes==object

# filter categorical columns using mask and turn it into a list
categorical_cols = data_x.columns[categorical_feature_mask].tolist()
```

In [0]:

```
categorical_cols
```

Out[0]:

```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'poutcome']
```

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer

def add_onehot_to_dataframe(sparse, df, vectorizer, name):
  '''
      This function will add the one hot encoded to the dataframe.

  '''
  for i, col in enumerate(vectorizer.get_feature_names()):
    colname = name+"_"+col
    # df[colname] = pd.SparseSeries(sparse[:, i].toarray().flatten(), fill_value=0)
    df[colname] = sparse[:, i].toarray().ravel().tolist()

  return df

def OneHotEncoder(categorical_cols, X_train, X_test, X_cv=None, include_cv=False):
  '''
    This function takes categorical column names as inputs. The objective
    of this function is to take the column names iteratively and encode the
    features using One hot Encoding mechanism and also adding the encoded feature
    to the respective dataframe
```

```
    to the respective dataframe.

    The include_cv parameter indicates whether we should include CV dataset or not.
    This is added specifically because when using GridSearchCV or RandomizedSearchCV,
    we only split the dataset into train and test to give more data to training purposes.
    This is done because GridSearchCV splits the data internally anyway.
  '''

  for i in categorical_cols:
    Vectorizer = CountVectorizer(token_pattern="[A-Za-z0-9-.]+")
    print("Encoding for feature: ", i)
    # Encoding training dataset
    temp_cols = Vectorizer.fit_transform(X_train[i])
    X_train = add_onehot_to_dataframe(temp_cols, X_train, Vectorizer, i)

    # Encoding Cross validation dataset
    if include_cv:
      temp_cols = Vectorizer.transform(X_cv[i])
      X_cv = add_onehot_to_dataframe(temp_cols, X_cv, Vectorizer, i)

    # Encoding Test dataset
    temp_cols = Vectorizer.transform(X_test[i])
    X_test = add_onehot_to_dataframe(temp_cols, X_test, Vectorizer, i)
```

In [0]:

```
OneHotEncoder(categorical_cols, X_train, X_test, X_cv, True)

# Drop the categorical features as the one hot encoded representation is present
X_train = X_train.drop(categorical_cols, axis=1)
X_cv = X_cv.drop(categorical_cols, axis=1)
X_test = X_test.drop(categorical_cols, axis=1)

print("Shape of train: ", X_train.shape)
print("Shape of CV: ", X_cv.shape)
print("Shape of test: ", X_test.shape)
```

```
Encoding for feature:  job
Encoding for feature:  marital
Encoding for feature:  education
Encoding for feature:  default
Encoding for feature:  housing
Encoding for feature:  loan
Encoding for feature:  contact
Encoding for feature:  month
Encoding for feature:  day_of_week
Encoding for feature:  poutcome
Shape of train:  (26352, 63)
Shape of CV:  (6588, 63)
Shape of test:  (8236, 63)
```

In [0]:

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26352 entries, 3466 to 12894
Data columns (total 63 columns):
age                              26352 non-null int64
duration                         26352 non-null int64
campaign                         26352 non-null int64
pdays                            26352 non-null int64
previous                         26352 non-null int64
emp.var.rate                     26352 non-null float64
cons.price.idx                   26352 non-null float64
cons.conf.idx                    26352 non-null float64
euribor3m                        26352 non-null float64
nr.employed                      26352 non-null float64
job_admin.                       26352 non-null int64
job_blue-collar                  26352 non-null int64
job_entrepreneur                 26352 non-null int64
job_housemaid                    26352 non-null int64
job_management                   26352 non-null int64
job_retired                      26352 non-null int64
```

```
job_self-employed             26352 non-null int64
job_services                  26352 non-null int64
job_student                   26352 non-null int64
job_technician                26352 non-null int64
job_unemployed                26352 non-null int64
job_unknown                   26352 non-null int64
marital_divorced              26352 non-null int64
marital_married               26352 non-null int64
marital_single                26352 non-null int64
marital_unknown               26352 non-null int64
education_basic.4y            26352 non-null int64
education_basic.6y            26352 non-null int64
education_basic.9y            26352 non-null int64
education_high.school         26352 non-null int64
education_illiterate          26352 non-null int64
education_professional.course 26352 non-null int64
education_university.degree   26352 non-null int64
education_unknown             26352 non-null int64
default_no                    26352 non-null int64
default_unknown               26352 non-null int64
default_yes                   26352 non-null int64
housing_no                    26352 non-null int64
housing_unknown               26352 non-null int64
housing_yes                   26352 non-null int64
loan_no                       26352 non-null int64
loan_unknown                  26352 non-null int64
loan_yes                      26352 non-null int64
contact_cellular              26352 non-null int64
contact_telephone             26352 non-null int64
month_apr                     26352 non-null int64
month_aug                     26352 non-null int64
month_dec                     26352 non-null int64
month_jul                     26352 non-null int64
month_jun                     26352 non-null int64
month_mar                     26352 non-null int64
month_may                     26352 non-null int64
month_nov                     26352 non-null int64
month_oct                     26352 non-null int64
month_sep                     26352 non-null int64
day_of_week_fri               26352 non-null int64
day_of_week_mon               26352 non-null int64
day_of_week_thu               26352 non-null int64
day_of_week_tue               26352 non-null int64
day_of_week_wed               26352 non-null int64
poutcome_failure              26352 non-null int64
poutcome_nonexistent          26352 non-null int64
poutcome_success              26352 non-null int64
dtypes: float64(5), int64(58)
memory usage: 12.9 MB
```

In [0]:

```
data_x.to_csv("encoded_data_x.csv")
data_y.to_csv("data_y.csv")
```

Benchmark model

We will create a simple LogisticRegression model without any hyper-parameter tuning and apply that to the data in two ways.

- Use "Duration" feature to see how the model performs with this feature. It will probably give very high AUC as the duration feature is very correlated with the target variable. But obviously we can't use the Duration feature for actual modelling.
- Next remove the "Duration" feature, and apply the same model to check how the model performs.

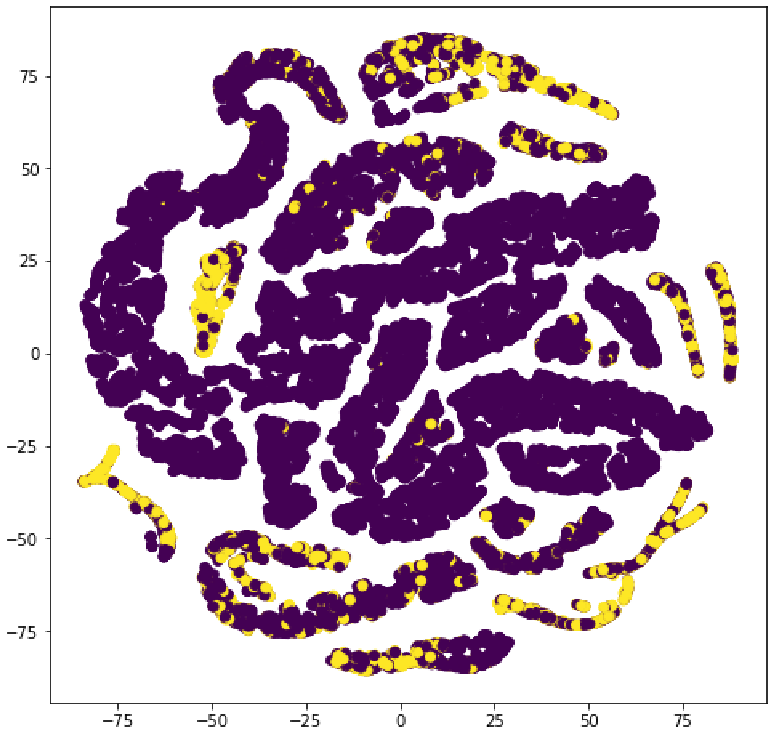
Visualize data with T-SNE plot

We will plot the t-sne plot for the dataset with "Duration" feature.

In [0]:

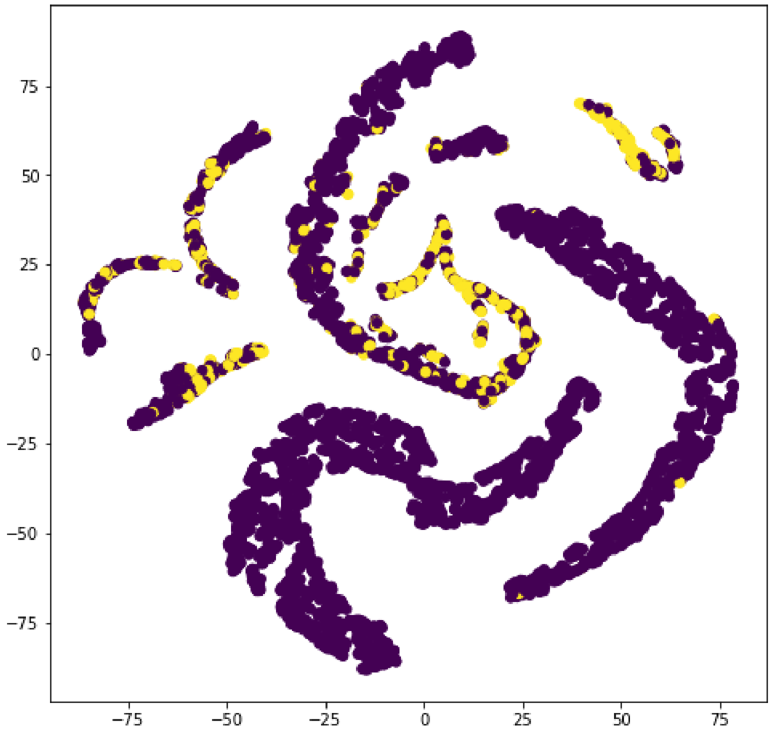
```
%matplotlib inline
```

```
# T-SNE plot for train dataset
model = TSNE(n_components=2, random_state=0, perplexity=30)
tsne_data = model.fit_transform(X_train)
plt.figure(figsize=(8,8))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=y_train.values)
plt.show()
```



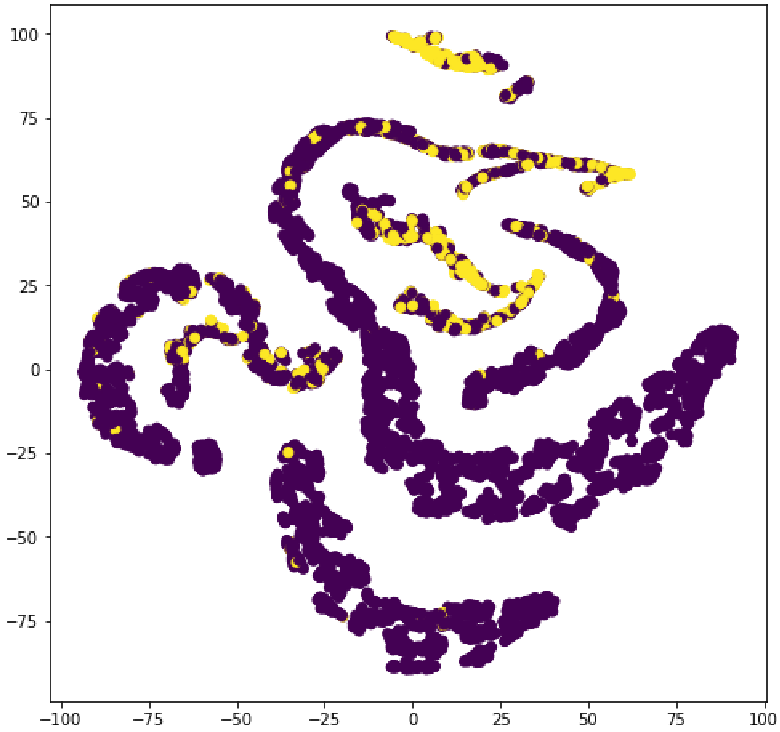
In [0]:

```
%matplotlib inline

# T-SNE plot for CV dataset
model = TSNE(n_components=2, random_state=0, perplexity=30)
tsne_data = model.fit_transform(X_cv)
plt.figure(figsize=(8,8))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=y_cv.values)
plt.show()
```



In [0]:

```
%matplotlib inline

# T-SNE plot for test dataset
model = TSNE(n_components=2, random_state=0, perplexity=30)
tsne_data = model.fit_transform(X_test)
plt.figure(figsize=(8,8))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=y_test.values)
plt.show()
```



Modelling with "Duration" Column

Seeing how the model performs with the "duration" feature. It is to be noted again that the duration feature can not be included in the final model as it is highly correlated with the target variable, and to build any reasonable predictive model, we cannot include this feature.

In [0]:

```
# with "duration" column
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(class_weight='balanced')
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)

print("AUC score with duration column: ", roc_auc_score(y_test, y_pred[:,1]))
```

```
AUC score with duration column:  0.9337494549248538
```

As we can see that, with duration column the AUC score is very good, 0.933. Which is a lot better than what the original paper achieved (AUC of 0.8). Now let's see how the same model does without the duration feature.

Removing "Duration" feature

duration: last contact duration, in seconds (numeric).

Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

In [0]:

```
# Removing duration feature

# From Train
X_train = X_train.drop("duration", axis=1)
print("The shape of the train dataset: ", X_train.shape)

# From CV
X_cv = X_cv.drop("duration", axis=1)
print("The shape of the cv dataset: ", X_cv.shape)

# From Test
X_test = X_test.drop("duration", axis=1)
print("The shape of the test dataset: ", X_test.shape)
```

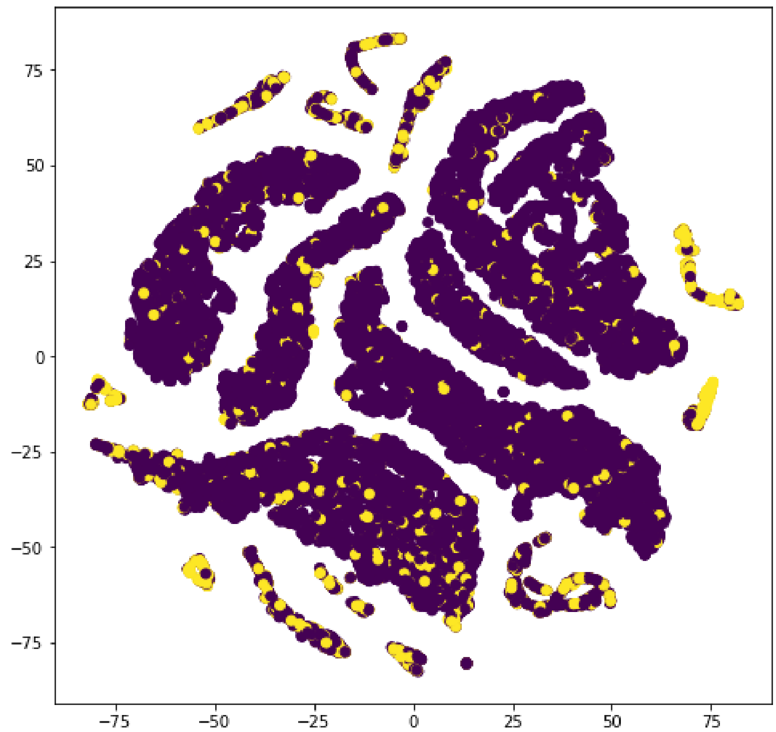
```
The shape of the train dataset:  (26352, 62)
The shape of the cv dataset:  (6588, 62)
The shape of the test dataset:  (8236, 62)
```

Visualize using T-SNE

Visualize the dataset with T-SNE plot but this time without the "Duration" column to see if there is any noticable change in the data.

In [0]:

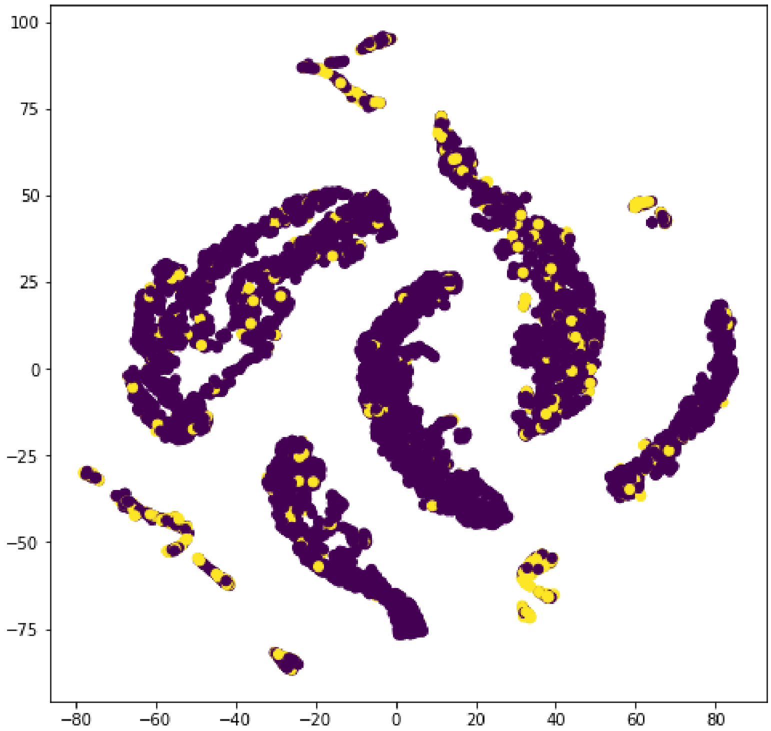
```
%matplotlib inline

# T-SNE plot for train dataset
model = TSNE(n_components=2, random_state=0, perplexity=30)
tsne_data = model.fit_transform(X_train)
plt.figure(figsize=(8,8))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=y_train.values)
plt.show()
```



In [0]:

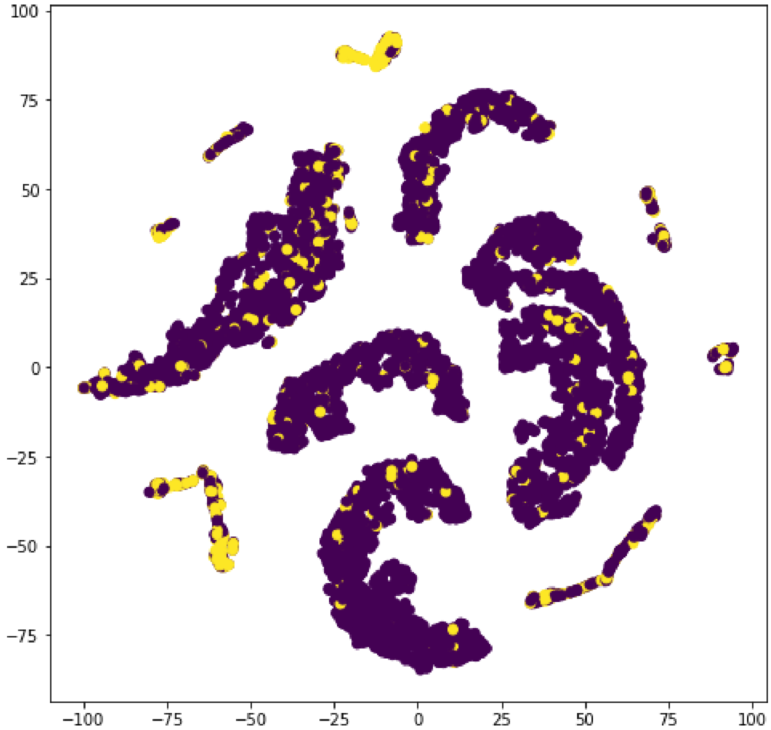
```
%matplotlib inline

# T-SNE plot for CV dataset
model = TSNE(n_components=2, random_state=0, perplexity=30)
```

```
tsne_data = model.fit_transform(X_cv)
plt.figure(figsize=(8,8))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=y_cv.values)
plt.show()
```



In [0]:

```
%matplotlib inline

# T-SNE plot for test dataset
model = TSNE(n_components=2, random_state=0, perplexity=30)
tsne_data = model.fit_transform(X_test)
plt.figure(figsize=(8,8))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=y_test.values)
plt.show()
```



Modelling Without "Duration" Column

Here we will just implement a LogisticRegression model without any hyperparameter tuning just to check how much the performance changed once we removed the "Duration" column from the dataset.

We will use two variations of the LogisticRegression model:

- With class balancing
- Without class balancing

This is to check the model is performing in both the scenarios.

In [0]:

```
# without "duration" column
# X_train = X_train.drop("duration", axis=1)
# X_test = X_test.drop("duration", axis=1)

# print(X_train.shape)
# print(X_test.shape)

model = LogisticRegression(class_weight='balanced')
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)

print("AUC score without duration column: ", roc_auc_score(y_test, y_pred[:,1]))
```

```
AUC score without duration column:  0.790764962074946
```

In [0]:

```
# without "duration" column and without class balancing
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)

print("AUC score without duration column and class balancing: ", roc_auc_score(y_test, y_pred[:,1]
))
```

```
AUC score without duration column and class balancing:  0.7896509920938419
```

KNN

In [0]:

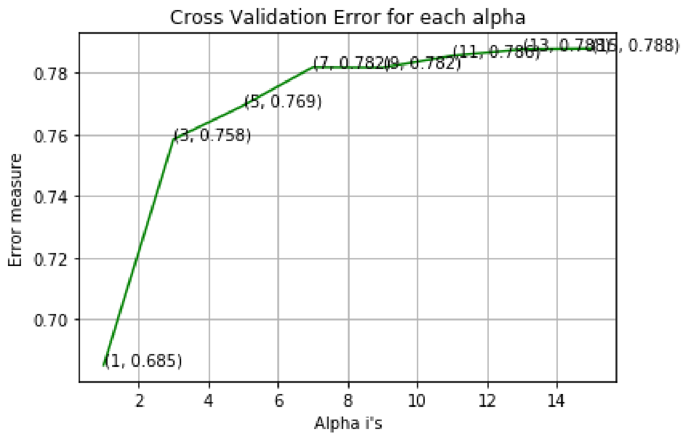
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [0]:

```
%matplotlib inline

alpha = [x for x in range(1, 17, 2)]
cv_auc_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for k = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
```

```
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for k =  1 is 0.6852384592221992
AUC for k =  3 is 0.7584454404373103
AUC for k =  5 is 0.7693689342876335
AUC for k =  7 is 0.781837375211359
AUC for k =  9 is 0.7816898292914554
AUC for k =  11 is 0.7855414263544345
AUC for k =  13 is 0.7876174924142404
AUC for k =  15 is 0.7878500451671182
```




```
For values of best alpha =  15 The train AUC is: 0.856258115532607
For values of best alpha =  15 The cross validation AUC is: 0.7878500451671182
For values of best alpha =  15 The test AUC is: 0.7728696243953527
```

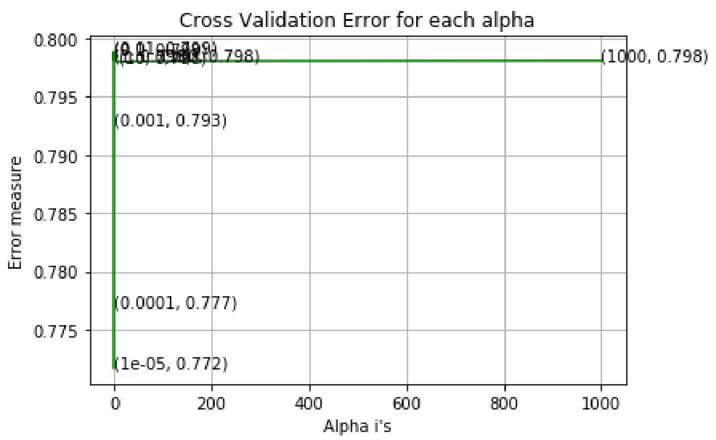
Logistic Regression

In [0]:

```
%matplotlib inline

alpha = [10 ** x for x in range(-5, 4)]
cv_auc_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for k = ',alpha[i],'is',cv_auc_array[i])
```

```
best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for k =  1e-05 is 0.7717236467236467
AUC for k =  0.0001 is 0.7769359321798347
AUC for k =  0.001 is 0.7926209668079587
AUC for k =  0.01 is 0.7988514812498552
AUC for k =  0.1 is 0.7987247816922614
AUC for k =  1 is 0.7981515067287425
AUC for k =  10 is 0.7979824195677854
AUC for k =  100 is 0.7980475065434416
AUC for k =  1000 is 0.7980880411368216
```




```
For values of best alpha =  0.01 The train AUC is: 0.79086052972031
For values of best alpha =  0.01 The cross validation AUC is: 0.7988514812498552
For values of best alpha =  0.01 The test AUC is: 0.7985460667145203
```

Linear SVM

In [0]:

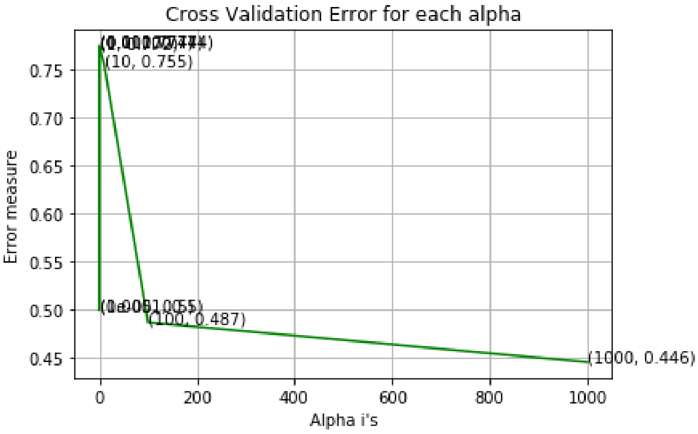
```
%matplotlib inline

alpha = [10 ** x for x in range(-5, 4)]
cv_auc_array=[]
for i in alpha:
    linearSVM = SGDClassifier(penalty='l2',alpha=i,class_weight='balanced')
    linearSVM.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(linearSVM, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
```

```
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for alpha = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

linearSVM = SGDClassifier(penalty='l2', alpha=alpha[best_alpha], class_weight='balanced')
linearSVM.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(linearSVM, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
```

```
AUC for alpha =  1e-05 is 0.5
AUC for alpha =  0.0001 is 0.5
AUC for alpha =  0.001 is 0.7738082829546244
AUC for alpha =  0.01 is 0.7743871169480925
AUC for alpha =  0.1 is 0.773978296620573
AUC for alpha =  1 is 0.772226738934056
AUC for alpha =  10 is 0.7545263243230723
AUC for alpha =  100 is 0.4869611794408542
AUC for alpha =  1000 is 0.4458929191856021
```




```
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
```

```
For values of best alpha =  0.01 The train AUC is: 0.7546448103852004
For values of best alpha =  0.01 The cross validation AUC is: 0.7742766312278507
For values of best alpha =  0.01 The test AUC is: 0.7781762429289234
```

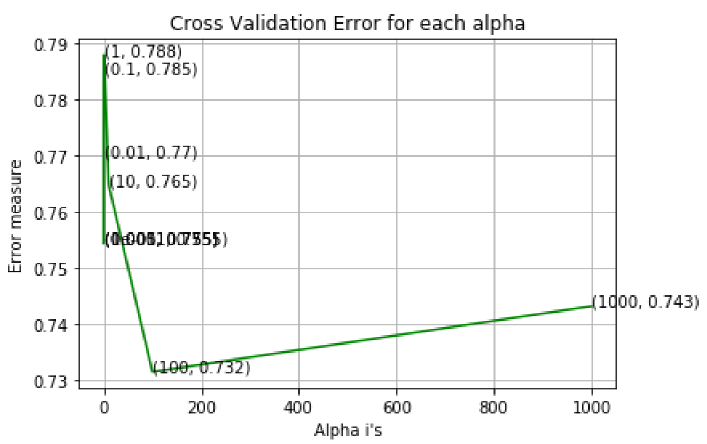
RBF Kernal SVM

In [0]:

```python
%matplotlib inline
from sklearn.svm import SVC

alpha = [10 ** x for x in range(-5, 4)]
cv_auc_array=[]
for i in alpha:
    SVM = SVC(C=i,class_weight='balanced')
    SVM.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(SVM, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for C = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
```

```
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

SVM = SVC(C=alpha[best_alpha], class_weight='balanced')
SVM.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(SVM, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for C =  1e-05 is 0.7545013086883005
AUC for C =  0.0001 is 0.7545013086883005
AUC for C =  0.001 is 0.7545013086883005
AUC for C =  0.01 is 0.7698435364695527
AUC for C =  0.1 is 0.7847331665624349
AUC for C =  1 is 0.7880278183123711
AUC for C =  10 is 0.7649124452782989
AUC for C =  100 is 0.7315849026011628
AUC for C =  1000 is 0.7432822597456744
```




```
For values of best alpha =  1 The train AUC is: 0.8545166434611486
For values of best alpha =  1 The cross validation AUC is: 0.7880278183123711
For values of best alpha =  1 The test AUC is: 0.7840642222878124
```

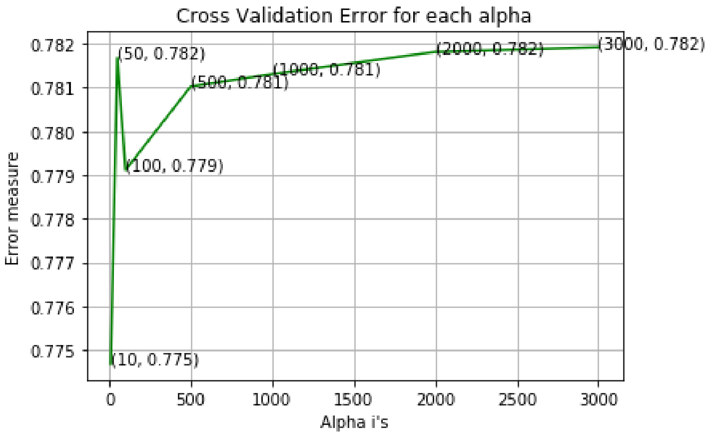
Random Forest

In [0]:

```
%matplotlib inline

alpha=[10,50,100,500,1000,2000,3000]
cv_auc_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for number of estimators = ',alpha[i],'is',cv_auc_array[i])
```

```
best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for number of estimators =  10 is 0.7746759548792068
AUC for number of estimators =  50 is 0.7816813749334074
AUC for number of estimators =  100 is 0.779111018460612
AUC for number of estimators =  500 is 0.7810240196419058
AUC for number of estimators =  1000 is 0.7813074143561949
AUC for number of estimators =  2000 is 0.7818161814096773
AUC for number of estimators =  3000 is 0.7819235401755726
```




```
For values of best alpha =  3000 The train AUC is: 0.9993202025886
For values of best alpha =  3000 The cross validation AUC is: 0.7819235401755726
For values of best alpha =  3000 The test AUC is: 0.7850592814848839
```

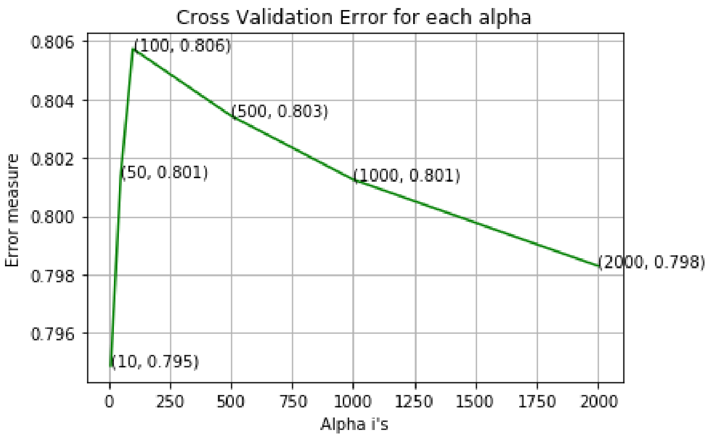
XGBoost

In [0]:

```
%matplotlib inline

alpha=[10,50,100,500,1000,2000]
cv_auc_array=[]
for i in alpha:
    x_cfl=x_cfl=XGBClassifier(n_estimators=i, tree_method="gpu_hist")
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))
```

```
for i in range(len(cv_auc_array)):
    print ('AUC for number of estimators = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=x_cfl=XGBClassifier(n_estimators=i, tree_method="gpu_hist")
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for number of estimators =  10 is 0.7948761957705047
AUC for number of estimators =  50 is 0.801374470154958
AUC for number of estimators =  100 is 0.8057312440645774
AUC for number of estimators =  500 is 0.8034416186042201
AUC for number of estimators =  1000 is 0.8012539086929331
AUC for number of estimators =  2000 is 0.7982958098811757
```




```
For values of best alpha =  100 The train AUC is: 0.7560302431582993
For values of best alpha =  100 The cross validation AUC is: 0.7654107891506265
For values of best alpha =  100 The test AUC is: 0.7647796927003376
```

XGBoost with RandomizedSearchCV hyper parameter tuning

In [0]:

```
# For RandomizedSearchCV I will use 80% of data for train and
# 20% of data for test. RandomizedSearchCV will internally split train data for Cross validation.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data_x, data_y, test_size=0.2)

print("X Train:", X_train.shape)
print("X Test:", X_test.shape)
print("Y Train:", y_train.shape)
```

```
print("Y Train:", y_test.shape)
```

```
X Train: (32940, 20)
X Test: (8236, 20)
Y Train: (32940,)
Y Train: (8236,)
```

In [0]:

```
OneHotEncoder(categorical_cols, X_train, X_test)
X_train = X_train.drop(categorical_cols, axis=1)
X_test = X_test.drop(categorical_cols, axis=1)

print("Shape of train: ", X_train.shape)
print("Shape of test: ", X_test.shape)
```

```
Encoding for feature:  job
Encoding for feature:  marital
Encoding for feature:  education
Encoding for feature:  default
Encoding for feature:  housing
Encoding for feature:  loan
Encoding for feature:  contact
Encoding for feature:  month
Encoding for feature:  day_of_week
Encoding for feature:  poutcome
Shape of train:  (32940, 63)
Shape of test:  (8236, 63)
```

In [0]:

```
x_cfl=XGBClassifier(tree_method='gpu_hist', max_bin=16)

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_iter=20, cv=10, scoring=
'roc_auc')
random_cfl.fit(X_train, y_train)
print (random_cfl.best_params_)
```

```
Fitting 10 folds for each of 20 candidates, totalling 200 fits
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.794, total=   0.5s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.5s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.795, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.9s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.797, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.4s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.778, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.8s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.815, total=   0.5s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.3s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.802, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    2.7s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.814, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    3.2s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.795, total=   0.5s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    3.6s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.806, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    4.1s remaining:    0.0s
```

```
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.1, colsample_bytree=0.3,
score=0.810, total=   0.4s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.751, total=   7.6s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.740, total=   7.6s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.748, total=   7.5s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.747, total=   7.5s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.759, total=   7.5s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.739, total=   7.6s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.750, total=   7.6s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.751, total=   7.5s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.761, total=   7.5s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.15, colsample_bytree=1,
score=0.750, total=   7.4s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
```

```
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.785, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.796, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.799, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.775, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.814, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.796, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.816, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.785, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.802, total=   0.3s
[CV] subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=100, max_depth=3, learning_rate=0.05, colsample_bytree=0.5,
score=0.807, total=   0.3s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.756, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.747, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.769, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.752, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.765, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.749, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.765, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.744, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.754, total=  11.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sc
ore=0.768, total=  11.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.746, total=  20.9s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.726, total=  21.2s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.728, total=  21.2s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.739, total=  21.2s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.759, total=  21.2s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.728, total=  21.2s
```

```
ore=0.720, total= 21.1s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.736, total=  21.2s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.725, total=  21.2s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.739, total=  21.1s
[CV] subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.3, sc
ore=0.736, total=  21.0s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.798, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.795, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.797, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.778, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.809, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.800, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.811, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.790, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.806, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.03, colsample_bytree=0.5, scor
e=0.812, total=   1.8s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.789, total=   0.6s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.795, total=   0.6s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.788, total=   0.6s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.777, total=   0.6s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.792, total=   0.6s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.790, total=   0.5s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.801, total=   0.6s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.778, total=   0.6s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.784, total=   0.5s
[CV] subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1
[CV]  subsample=0.1, n_estimators=200, max_depth=3, learning_rate=0.15, colsample_bytree=1,
score=0.793, total=   0.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.707, total=   6.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
```

```
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.700, total=   6.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.712, total=   6.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.705, total=   6.5s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.727, total=   6.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.685, total=   6.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.741, total=   6.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.734, total=   6.5s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.730, total=   6.6s
[CV] subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=5, learning_rate=0.2, colsample_bytree=0.5,
score=0.718, total=   6.6s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.776, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.772, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.772, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.769, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.793, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.773, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.788, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.776, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.782, total=   1.8s
[CV] subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=500, max_depth=5, learning_rate=0.15, colsample_bytree=0.3, scor
e=0.800, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1,
score=0.757, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1,
score=0.746, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1,
score=0.762, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1,
score=0.767, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1,
score=0.774, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1,
score=0.757, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1,
score=0.776, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
```

```
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1, score=0.762, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1, score=0.768, total=   1.8s
[CV] subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=500, max_depth=5, learning_rate=0.2, colsample_bytree=1, score=0.770, total=   1.8s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.792, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.798, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.786, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.778, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.803, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.792, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.808, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.790, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.792, total=   0.7s
[CV] subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.1, score=0.804, total=   0.7s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.790, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.802, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.798, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.774, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.811, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.799, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.810, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.793, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.806, total=   0.4s
[CV] subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=100, max_depth=5, learning_rate=0.2, colsample_bytree=0.1, score=0.812, total=   0.4s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5, score=0.799, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5, score=0.795, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5, score=0.799, total=   0.8s
```

```
score=0.799, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5,
score=0.785, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5,
score=0.815, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5,
score=0.798, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5,
score=0.812, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5,
score=0.791, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5,
score=0.805, total=   0.8s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.1, colsample_bytree=0.5,
score=0.817, total=   0.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.794, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.795, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.797, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.776, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.810, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.802, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.811, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.792, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.803, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.3, sco
re=0.810, total=   4.7s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.755, total=   6.4s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.735, total=   6.3s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.745, total=   6.4s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.745, total=   6.4s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.754, total=   6.2s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.736, total=   6.4s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.755, total=   6.3s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1,
score=0.741, total=   6.3s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
```

```
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1, score=0.763, total=   6.3s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.2, colsample_bytree=1, score=0.753, total=   6.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.794, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.800, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.795, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.776, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.810, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.798, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.812, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.791, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.804, total=   2.4s
[CV] subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.3, n_estimators=1000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.812, total=   2.4s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.716, total=  15.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.717, total=  15.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.712, total=  15.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.709, total=  15.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.723, total=  15.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.704, total=  15.1s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.728, total=  15.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.712, total=  15.0s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.711, total=  15.1s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.15, colsample_bytree=0.5, score=0.707, total=  15.0s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.771, total=   9.3s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.752, total=   9.3s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.771, total=   9.3s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.748, total=   9.3s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
```

```
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.776, total=   9.3s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.759, total=   9.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.774, total=   9.2s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.758, total=   9.1s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.771, total=   9.1s
[CV] subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.1, n_estimators=2000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.774, total=   9.1s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.791, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.789, total=   4.6s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.796, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.773, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.801, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.800, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.798, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.783, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.791, total=   4.7s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.1, colsample_bytree=0.1, score=0.811, total=   4.7s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.792, total=   1.3s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.800, total=   1.2s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.800, total=   1.2s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.776, total=   1.2s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.813, total=   1.2s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.799, total=   1.3s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.812, total=   1.2s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.793, total=   1.3s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, score=0.807, total=   1.2s
[CV] subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=500, max_depth=3, learning_rate=0.03, colsample_bytree=0.5, scor
```

```
e=0.813, total=   1.3s
```

```
[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed: 16.6min finished
```

```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 0.
5}
```

In [0]:

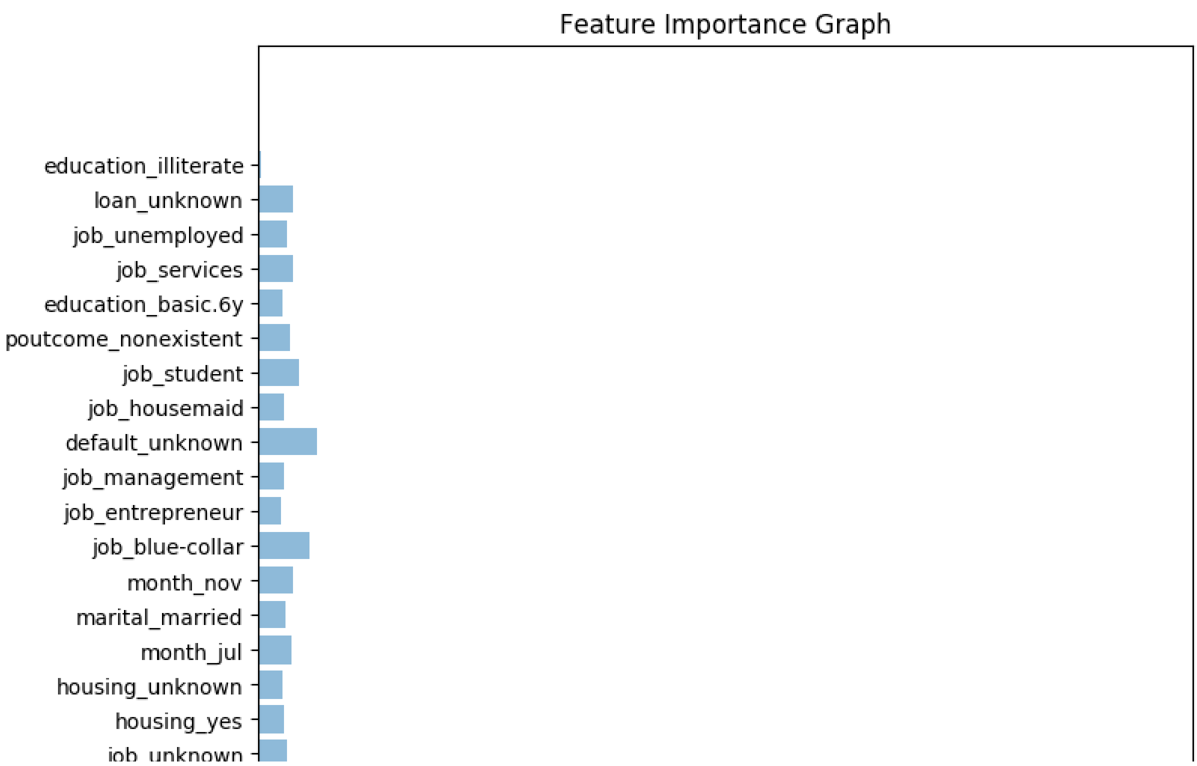
```
x_cfl=XGBClassifier(n_estimators=200,max_depth=5,learning_rate=0.1, \
                    colsample_bytree=0.5,subsample=1,tree_method='gpu_hist', max_bin=16)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("For values of best alpha = 200 The train AUC is:",roc_auc_score(y_train, predict_y[:, 1]))
predict_y = sig_clf.predict_proba(X_test)
print("For values of best alpha = 200 The test AUC is:",roc_auc_score(y_test, predict_y[:, 1]))
```

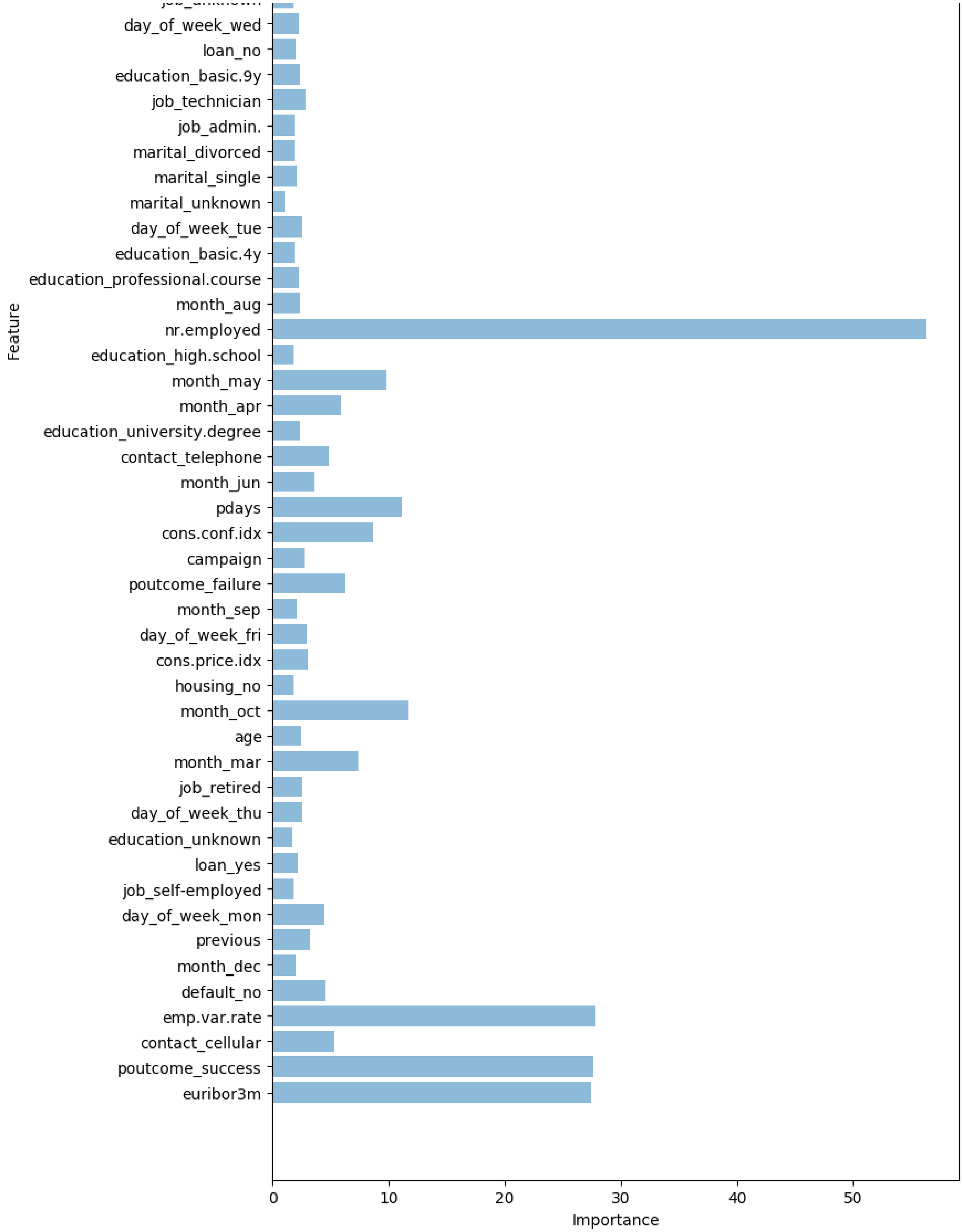
```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The
default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this
warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
For values of best alpha = 200 The train AUC is: 0.8611384014781621
For values of best alpha = 200 The test AUC is: 0.7925603920640862
```

In [0]:

```
import matplotlib.pyplot as plt; plt.rcdefaults()
feature_importance = x_cfl.get_booster().get_score(importance_type='gain')

objects = feature_importance.keys()
y_pos = np.arange(len(objects))
performance = feature_importance.values()
plt.figure(figsize=(8,20))
plt.barh(y_pos, performance, align='center', alpha=0.5)
plt.yticks(y_pos, objects)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance Graph')
plt.show()
```





From the above feature importance graph we can see that, the most relevent features are the following:

- nr.employed
- emp.var.rate
- poutcome_success
- euribor3m
- etc.

Response coding

Train test split

In [0]:

```
from sklearn.model_selection import train_test_split

X_rest, X_test, y_rest, y_test = train_test_split(data_x, data_y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_rest, y_rest, test_size=0.2)

print("X Train:", X_train.shape)
print("X CV:", X_cv.shape)
print("X Test:", X_test.shape)
print("Y Train:", y_train.shape)
print("Y CV:", y_cv.shape)
print("Y Test:", y_test.shape)
```

```
X Train: (26352, 20)
X CV: (6588, 20)
X Test: (8236, 20)
Y Train: (26352,)
Y CV: (6588,)
Y Test: (8236,)
```

In [0]:

```
y_train.replace({"no":0, "yes":1}, inplace=True)
y_cv.replace({"no":0, "yes":1}, inplace=True)
y_test.replace({"no":0, "yes":1}, inplace=True)
```

In [0]:

```
X_train.head()
```

Out[0]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays
37077	32	admin.	single	university.degree	no	yes	no	cellular	jul	thu	315	4	999
20127	39	technician	single	professional.course	no	no	no	cellular	aug	mon	71	1	999
20619	30	technician	married	professional.course	no	no	yes	cellular	aug	wed	100	1	999
29494	38	admin.	single	university.degree	no	yes	no	cellular	apr	mon	285	1	999
33592	32	admin.	single	university.degree	no	no	no	cellular	may	tue	144	5	999

In [0]:

```
# Categorical boolean mask
categorical_feature_mask = data_x.dtypes==object

# filter categorical columns using mask and turn it into a list
categorical_cols = data_x.columns[categorical_feature_mask].tolist()
```

In [0]:

```
categorical_cols
```

Out[0]:

```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'poutcome']
```

In [0]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: Categorical Features
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train dataframe
# build a vector (1*2) , the first element = (number of times it occured in class1 + 10*alpha / nu
mber of time it occurred in total data+20*alpha)
# feat_dict is like a look up table, for every categorical data it store a (1*2) representation of
it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'feat_dict' look up table to 'res_fea'
# if it is not there is train:
# we add [1/2, 1/2] to 'res_fea'
# return 'res_fea'
# ----------------------

# get_fea_dict: Get categorical data Feature Dict
def get_fea_dict(alpha, feature, train_df, train_df_y):
    # value_count: it contains a dict like
    value_count = train_df[feature].value_counts()

    # feat_dict : Categorical feature Dict, which contains the probability array for each
categorical variable
    feat_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of the particular
        # categorical feature belongs to particular class
        # vec is 2 diamensional vector
        vec = []
        for k in range(0, 2):
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df_y==k) & (train_df[feature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 20*alpha))

        # we are adding the categorical feature to the dict as key and vec as value
        feat_dict[i]=vec
    return feat_dict

# Get Response coded feature
def get_response_feature(alpha, feature, train_df, train_df_y):

    feat_dict = get_fea_dict(alpha, feature, train_df, train_df_y)
    # value_count is similar in get_fea_dict
    value_count = train_df[feature].value_counts()

    # res_fea: response coded feature, it will contain the response coded feature for each feature
value in the data
    res_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to res_fea
    # if not we will add [1/2, 1/2] to res_fea
    for index, row in train_df.iterrows():
        if row[feature] in dict(value_count).keys():
            res_fea.append(feat_dict[row[feature]])
        else:
            res_fea.append([1/2, 1/2])
    return res_fea
```

In [0]:

```
def ResponseEncoder(categorical_cols, x_df, y_df):

    """
```

```
    This function takes Categorical column names and X and Y dataframe.

    Returns the response coded dataframe
  """
  print("Encoding Train dataset")
  print("Shape of the train dataset before encoding: ", X_train.shape)
  for i in (categorical_cols):
    temp_response_coded_feature = np.array(get_response_feature(alpha=1, feature=i, train_df=x_df,
train_df_y=y_df))
    df_response = pd.DataFrame(temp_response_coded_feature, columns=[i+"_0", i+"_1"])
    x_df = pd.concat([x_df, df_response], axis=1)

  # Remove the categorical features as the response coded features are added
  x_df = x_df.drop(categorical_cols, axis=1)
  return x_df
```

In [0]:

```
# Reset index so that pd.concat works properly in ResponseEncoder function
X_train = X_train.reset_index().drop("index",axis=1)
X_test = X_test.reset_index().drop("index",axis=1)
X_cv = X_cv.reset_index().drop("index",axis=1)
```

In [0]:

```
X_train = ResponseEncoder(categorical_cols, X_train, y_train)
print("Shape of the train dataset after encoding: ", X_train.shape)

X_cv = ResponseEncoder(categorical_cols, X_cv, y_cv)
print("Shape of the cv dataset after encoding: ", X_cv.shape)

X_test = ResponseEncoder(categorical_cols, X_test, y_test)
print("Shape of the test dataset after encoding: ", X_test.shape)
```

```
Encoding Train dataset
Shape of the train dataset before encoding:  (26352, 20)
Shape of the train dataset after encoding:  (26352, 30)
Encoding Train dataset
Shape of the train dataset before encoding:  (26352, 30)
Shape of the cv dataset after encoding:  (6588, 30)
Encoding Train dataset
Shape of the train dataset before encoding:  (26352, 30)
Shape of the test dataset after encoding:  (8236, 30)
```

In [0]:

```
# Remove duration feature
X_train = X_train.drop("duration", axis=1)
X_cv = X_cv.drop("duration", axis=1)
X_test = X_test.drop("duration", axis=1)
```

In [0]:

```
X_train.to_csv("Response_coded_features_train.csv")
X_cv.to_csv("Response_coded_features_cv.csv")
X_test.to_csv("Response_coded_features_test.csv")
```

KNN

In [0]:

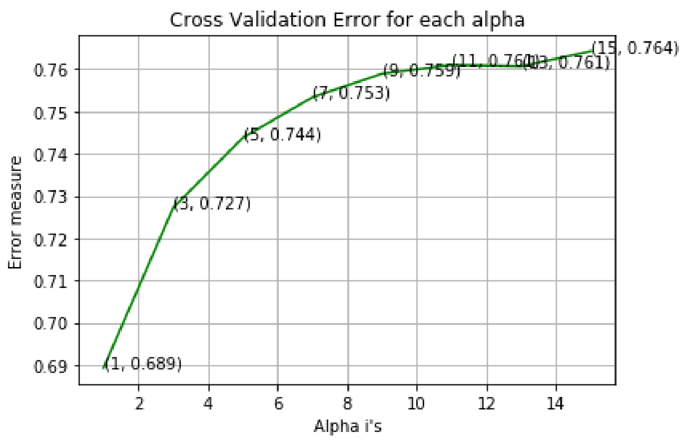
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [0]:

```
%matplotlib inline
```

```
alpha = [x for x in range(1, 17, 2)]
cv_auc_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for k = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for k =  1 is 0.6892692815811569
AUC for k =  3 is 0.7271750154586474
AUC for k =  5 is 0.7437369930303207
AUC for k =  7 is 0.7533350633870685
AUC for k =  9 is 0.758886708016394
AUC for k =  11 is 0.7609873043077136
AUC for k =  13 is 0.7606555236134297
AUC for k =  15 is 0.7642352436958662
```



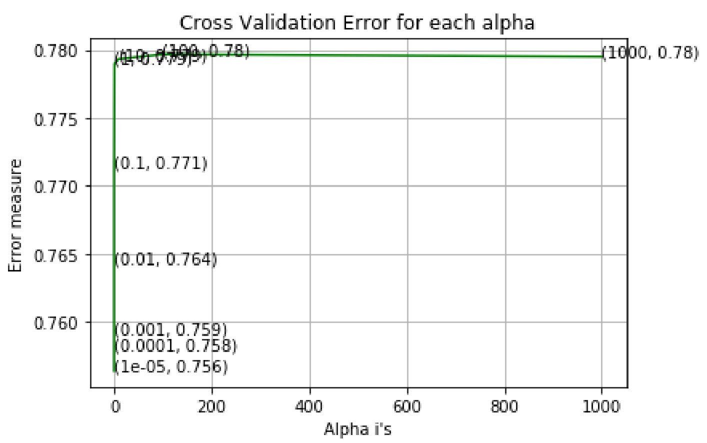

```
For values of best alpha =  15 The train AUC is: 0.8579027560908474
For values of best alpha =  15 The cross validation AUC is: 0.7642352436958662
For values of best alpha =  15 The test AUC is: 0.7588179261926234
```

Logistic Regression

In [0]:

```
%matplotlib inline

alpha = [10 ** x for x in range(-5, 4)]
cv_auc_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for k = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for k =  1e-05 is 0.7563957673489795
AUC for k =  0.0001 is 0.7580064489797294
AUC for k =  0.001 is 0.7591216842674381
AUC for k =  0.01 is 0.7643363772819093
AUC for k =  0.1 is 0.7713955256385064
AUC for k =  1 is 0.77901481697226
AUC for k =  10 is 0.7793269962699625
AUC for k =  100 is 0.7796918468051047
AUC for k =  1000 is 0.7795145924581042
```




```
For values of best alpha =  100 The train AUC is: 0.7932716928892307
For values of best alpha =  100 The cross validation AUC is: 0.7796918468051047
For values of best alpha =  100 The test AUC is: 0.7785456926327435
```

Linear SVM

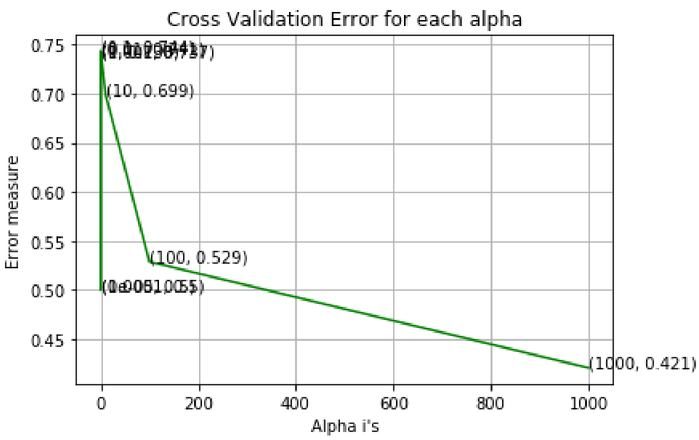
In [0]:

```
%matplotlib inline

alpha = [10 ** x for x in range(-5, 4)]
cv_auc_array=[]
for i in alpha:
    linearSVM = SGDClassifier(penalty='l2',alpha=i,class_weight='balanced')
    linearSVM.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(linearSVM, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for alpha = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

linearSVM = SGDClassifier(penalty='l2', alpha=alpha[best_alpha], class_weight='balanced')
linearSVM.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(linearSVM, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:453: RuntimeWarning: overflow
encountered in exp
  E = np.exp(AB[0] * F + AB[1])
/usr/local/lib/python3.6/dist-packages/sklearn/calibration.py:455: RuntimeWarning: invalid value e
ncountered in multiply
  TEP_minus_T1P = P * (T * E - T1)
```

```
AUC for alpha =  1e-05 is 0.5
AUC for alpha =  0.0001 is 0.5
AUC for alpha =  0.001 is 0.7372568002412763
AUC for alpha =  0.01 is 0.740947512136408
AUC for alpha =  0.1 is 0.7435116434288933
AUC for alpha =  1 is 0.7375577095315304
AUC for alpha =  10 is 0.6988044343419817
AUC for alpha =  100 is 0.5287883988035883
AUC for alpha =  1000 is 0.4207704238179307
```




```
For values of best alpha =  0.1 The train AUC is: 0.7530163400449308
For values of best alpha =  0.1 The cross validation AUC is: 0.744081679037255
For values of best alpha =  0.1 The test AUC is: 0.7576224490091846
```

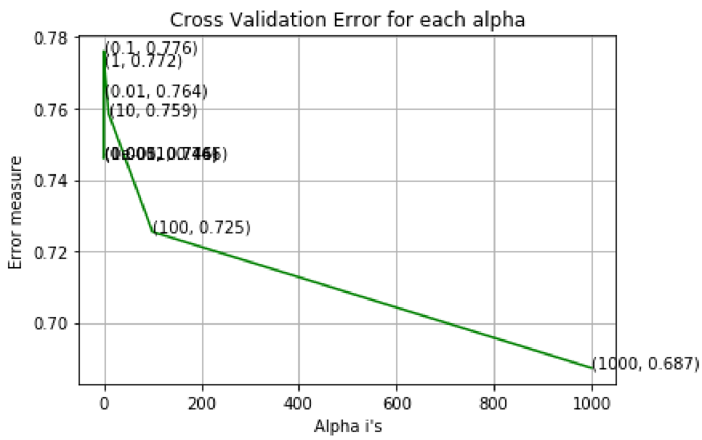
RBF Kernal SVM

In [0]:

```
%matplotlib inline
from sklearn.svm import SVC

alpha = [10 ** x for x in range(-5, 4)]
cv_auc_array=[]
for i in alpha:
    SVM = SVC(C=i,class_weight='balanced')
    SVM.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(SVM, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for C = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
SVM = SVC(C=alpha[best_alpha], class_weight='balanced')
SVM.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(SVM, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for C =  1e-05 is 0.7460862323061918
AUC for C =  0.0001 is 0.7460862323061918
AUC for C =  0.001 is 0.7460862323061918
AUC for C =  0.01 is 0.7636404656315556
AUC for C =  0.1 is 0.7760276747942293
AUC for C =  1 is 0.7722338394399492
AUC for C =  10 is 0.758527093933343
AUC for C =  100 is 0.7254473190043089
AUC for C =  1000 is 0.6873026616336615
```




```
For values of best alpha =  0.1 The train AUC is: 0.7865539927853393
For values of best alpha =  0.1 The cross validation AUC is: 0.7760276747942293
For values of best alpha =  0.1 The test AUC is: 0.7508529620979546
```

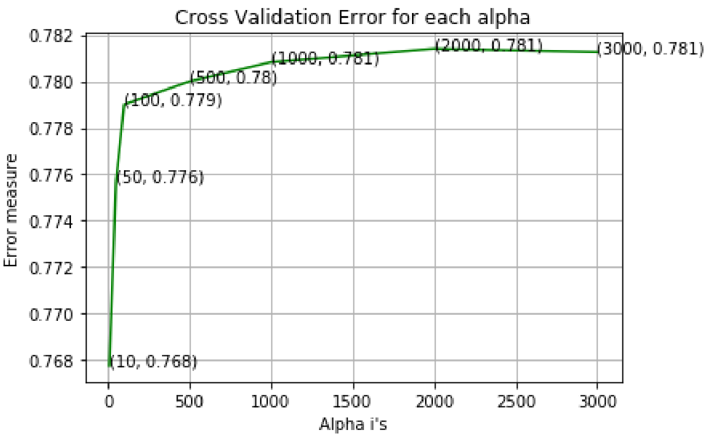
Random Forest

In [0]:

```
%matplotlib inline

alpha=[10,50,100,500,1000,2000,3000]
cv_auc_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for number of estimators = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
```

```
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for number of estimators =  10 is 0.7677506035902254
AUC for number of estimators =  50 is 0.7756808419171664
AUC for number of estimators =  100 is 0.7790068457401851
AUC for number of estimators =  500 is 0.7799931306886485
AUC for number of estimators =  1000 is 0.7808271010046929
AUC for number of estimators =  2000 is 0.78140459458155
AUC for number of estimators =  3000 is 0.7812622696339397
```




```
For values of best alpha =  2000 The train AUC is: 0.9991794771349454
For values of best alpha =  2000 The cross validation AUC is: 0.78140459458155
For values of best alpha =  2000 The test AUC is: 0.7759755415777504
```

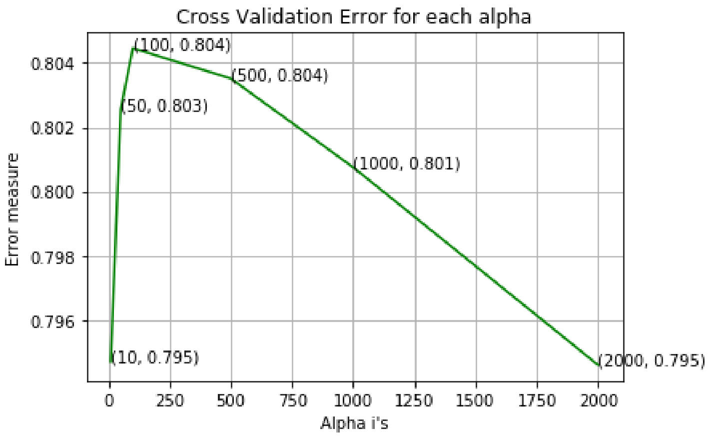
XGBoost

In [0]:

```
%matplotlib inline

alpha=[10,50,100,500,1000,2000]
cv_auc_array=[]
for i in alpha:
    x_cfl=x_cfl=XGBClassifier(n_estimators=i, tree_method="gpu_hist")
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_auc_array.append(roc_auc_score(y_cv, predict_y[:,1]))

for i in range(len(cv_auc_array)):
    print ('AUC for number of estimators = ',alpha[i],'is',cv_auc_array[i])

best_alpha = np.argmax(cv_auc_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_auc_array,c='g')
for i, txt in enumerate(np.round(cv_auc_array,3)):
```

```
for i, txt in enumerate(np.round(cv_auc_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_auc_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=x_cfl=XGBClassifier(n_estimators=i, tree_method="gpu_hist")
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train AUC is:",roc_auc_score(y_train
, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation AUC
is:",roc_auc_score(y_cv, predict_y[:,1]))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test AUC is:",roc_auc_score(y_test, p
redict_y[:,1]))
```

```
AUC for number of estimators =  10 is 0.7947058037823865
AUC for number of estimators =  50 is 0.802548828793639
AUC for number of estimators =  100 is 0.8044594401960223
AUC for number of estimators =  500 is 0.8035154711034336
AUC for number of estimators =  1000 is 0.8007511345026876
AUC for number of estimators =  2000 is 0.7946001825081741
```




```
For values of best alpha =  100 The train AUC is: 0.7911513565458436
For values of best alpha =  100 The cross validation AUC is: 0.7926794691773326
For values of best alpha =  100 The test AUC is: 0.7778055194167837
```

XGBoost with RandomizedSearchCV hyper parameter tuning

In [0]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data_x, data_y, test_size=0.2)

print("X Train:", X_train.shape)
print("X Test:", X_test.shape)
print("Y Train:", y_train.shape)
print("Y Test:", y_test.shape)
```

```
X Train: (32940, 20)
X Test: (8236, 20)
Y Train: (32940,)
Y Test: (8236,)
```

In [0]:

```
y_train.replace({"no":0, "yes":1}, inplace=True)
y_test.replace({"no":0, "yes":1}, inplace=True)
```

In [0]:

```
# Reset index so that pd.concat works properly in ResponseEncoder function
X_train = X_train.reset_index().drop("index",axis=1)
X_test = X_test.reset_index().drop("index",axis=1)
X_cv = X_cv.reset_index().drop("index",axis=1)
```

In [0]:

```
X_train = ResponseEncoder(categorical_cols, X_train, y_train)
print("Shape of the train dataset after encoding: ", X_train.shape)

X_test = ResponseEncoder(categorical_cols, X_test, y_test)
print("Shape of the test dataset after encoding: ", X_test.shape)
```

```
Encoding Train dataset
Shape of the train dataset before encoding:  (32940, 20)
Shape of the train dataset after encoding:  (32940, 30)
Encoding Train dataset
Shape of the train dataset before encoding:  (32940, 30)
Shape of the test dataset after encoding:  (8236, 30)
```

In [0]:

```
# Remove duration feature
X_train = X_train.drop("duration", axis=1)
X_test = X_test.drop("duration", axis=1)
```

In [0]:

```
x_cfl=XGBClassifier(tree_method='gpu_hist', max_bin=16)

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_iter=20, cv=5, scoring='
roc_auc')
random_cfl.fit(X_train, y_train)
print (random_cfl.best_params_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5,
score=0.772, total=   7.9s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    7.9s remaining:    0.0s
```

```
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5,
score=0.784, total=   7.9s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   15.8s remaining:    0.0s
```

```
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5,
score=0.760, total=   7.4s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   23.2s remaining:    0.0s
```

```
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5, score=0.785, total=   7.7s
[CV] subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:   30.9s remaining:    0.0s
```

```
[CV]  subsample=1, n_estimators=500, max_depth=10, learning_rate=0.03, colsample_bytree=0.5, score=0.755, total=   7.6s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   38.5s remaining:    0.0s
```

```
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.785, total=   5.6s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:   44.2s remaining:    0.0s
```

```
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.794, total=   5.6s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:   49.8s remaining:    0.0s
```

```
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.768, total=   5.6s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:   55.4s remaining:    0.0s
```

```
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.794, total=   5.6s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:  1.0min remaining:    0.0s
```

```
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.761, total=   5.5s
[CV] subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.794, total=   0.6s
[CV] subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.807, total=   0.6s
[CV] subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.777, total=   0.6s
[CV] subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.801, total=   0.6s
[CV] subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=100, max_depth=10, learning_rate=0.1, colsample_bytree=0.1, score=0.775, total=   0.6s
[CV] subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1, score=0.793, total=   0.9s
[CV] subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1, score=0.812, total=   0.9s
[CV] subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1, score=0.775, total=   0.9s
[CV] subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1, score=0.808, total=   0.9s
[CV] subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1
```

```
[CV]  subsample=0.3, n_estimators=200, max_depth=3, learning_rate=0.03, colsample_bytree=1,
score=0.779, total=   0.9s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.761, total=   8.4s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.761, total=   8.4s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.733, total=   8.3s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.761, total=   8.3s
[CV] subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.5, n_estimators=500, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.730, total=   8.4s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5,
score=0.797, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5,
score=0.811, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5,
score=0.781, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5,
score=0.807, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.01, colsample_bytree=0.5,
score=0.781, total=   8.8s
[CV] subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sco
re=0.795, total=   1.4s
[CV] subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sco
re=0.810, total=   1.4s
[CV] subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sco
re=0.779, total=   1.4s
[CV] subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sco
re=0.807, total=   1.3s
[CV] subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.5, n_estimators=100, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sco
re=0.778, total=   1.4s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.796, total=   1.0s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.811, total=   1.0s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.777, total=   1.0s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.805, total=   1.0s
[CV] subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=200, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.777, total=   1.0s
[CV] subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3, sc
ore=0.783, total=  14.9s
[CV] subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3, sc
ore=0.799, total=  14.9s
[CV] subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3, sc
ore=0.772, total=  14.9s
[CV] subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3, sc
ore=0.795, total=  14.8s
[CV] subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.3, sc
ore=0.764, total=  14.9s
```

```
[CV] subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, sco
re=0.796, total=   4.5s
[CV] subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, sco
re=0.812, total=   4.5s
[CV] subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, sco
re=0.778, total=   4.5s
[CV] subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, sco
re=0.809, total=   4.5s
[CV] subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=1000, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, sco
re=0.780, total=   4.5s
[CV] subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1,
score=0.796, total=   2.3s
[CV] subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1,
score=0.811, total=   2.2s
[CV] subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1,
score=0.776, total=   2.2s
[CV] subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1,
score=0.809, total=   2.2s
[CV] subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1
[CV]  subsample=1, n_estimators=500, max_depth=3, learning_rate=0.01, colsample_bytree=1,
score=0.777, total=   2.2s
[CV] subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.749, total=  13.8s
[CV] subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.754, total=  13.8s
[CV] subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.733, total=  13.8s
[CV] subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.747, total=  13.7s
[CV] subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=1,
score=0.723, total=  13.6s
[CV] subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, scor
e=0.792, total=   0.9s
[CV] subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, scor
e=0.806, total=   0.9s
[CV] subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, scor
e=0.773, total=   0.9s
[CV] subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, scor
e=0.805, total=   0.9s
[CV] subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3
[CV]  subsample=0.5, n_estimators=200, max_depth=3, learning_rate=0.01, colsample_bytree=0.3, scor
e=0.775, total=   0.9s
[CV] subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sco
re=0.795, total=   1.2s
[CV] subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sco
re=0.806, total=   1.2s
[CV] subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sco
re=0.775, total=   1.2s
[CV] subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sco
re=0.799, total=   1.2s
[CV] subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1
[CV]  subsample=0.3, n_estimators=200, max_depth=10, learning_rate=0.15, colsample_bytree=0.1, sco
re=0.775, total=   1.2s
[CV] subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1, sco
```

```
re=0.793, total=   1.2s
[CV] subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1, sco
re=0.810, total=   1.2s
[CV] subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1, sco
re=0.774, total=   1.2s
[CV] subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1, sco
re=0.801, total=   1.2s
[CV] subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1
[CV]  subsample=0.5, n_estimators=200, max_depth=10, learning_rate=0.03, colsample_bytree=0.1, sco
re=0.775, total=   1.2s
[CV] subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5,
score=0.746, total=  13.6s
[CV] subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5,
score=0.750, total=  13.6s
[CV] subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5,
score=0.723, total=  13.4s
[CV] subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5,
score=0.759, total=  13.3s
[CV] subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=1000, max_depth=10, learning_rate=0.05, colsample_bytree=0.5,
score=0.732, total=  13.5s
[CV] subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.797, total=   5.1s
[CV] subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.809, total=   5.1s
[CV] subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.778, total=   5.1s
[CV] subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.806, total=   5.1s
[CV] subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1
[CV]  subsample=1, n_estimators=1000, max_depth=5, learning_rate=0.05, colsample_bytree=0.1,
score=0.773, total=   5.1s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sc
ore=0.785, total=   9.3s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sc
ore=0.798, total=   9.3s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sc
ore=0.774, total=   9.3s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sc
ore=0.798, total=   9.3s
[CV] subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5
[CV]  subsample=0.1, n_estimators=1000, max_depth=10, learning_rate=0.01, colsample_bytree=0.5, sc
ore=0.764, total=   9.3s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5,
score=0.794, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5,
score=0.807, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5,
score=0.783, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5,
score=0.805, total=   8.9s
[CV] subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5
[CV]  subsample=1, n_estimators=2000, max_depth=3, learning_rate=0.03, colsample_bytree=0.5,
score=0.774, total=   8.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1,
score=0.752, total=   8.9s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1
```

```
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1,
score=0.777, total=   8.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1,
score=0.741, total=   8.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1,
score=0.763, total=   8.8s
[CV] subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1
[CV]  subsample=0.3, n_estimators=2000, max_depth=3, learning_rate=0.2, colsample_bytree=1,
score=0.736, total=   8.8s
```

```
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  9.9min finished
```

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.01, 'colsample_bytree':
0.5}
```

In [0]:

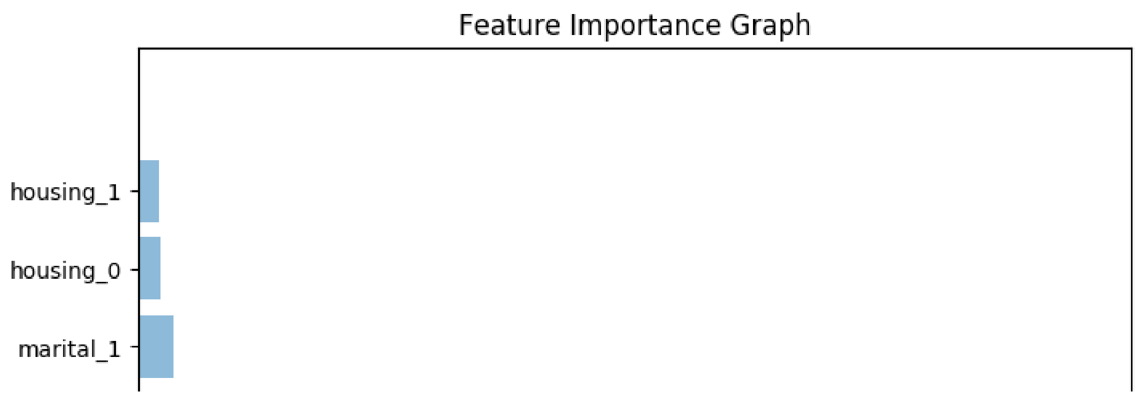
```
x_cfl=XGBClassifier(n_estimators=2000,max_depth=3,learning_rate=0.01, \
                    colsample_bytree=0.5,subsample=1,tree_method='gpu_hist', max_bin=16)
x_cfl.fit(X_train,y_train,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ("For values of best alpha = 2000 The train AUC is:",roc_auc_score(y_train, predict_y[:, 1])
)
predict_y = sig_clf.predict_proba(X_test)
print("For values of best alpha = 2000 The test AUC is:",roc_auc_score(y_test, predict_y[:, 1]))
```

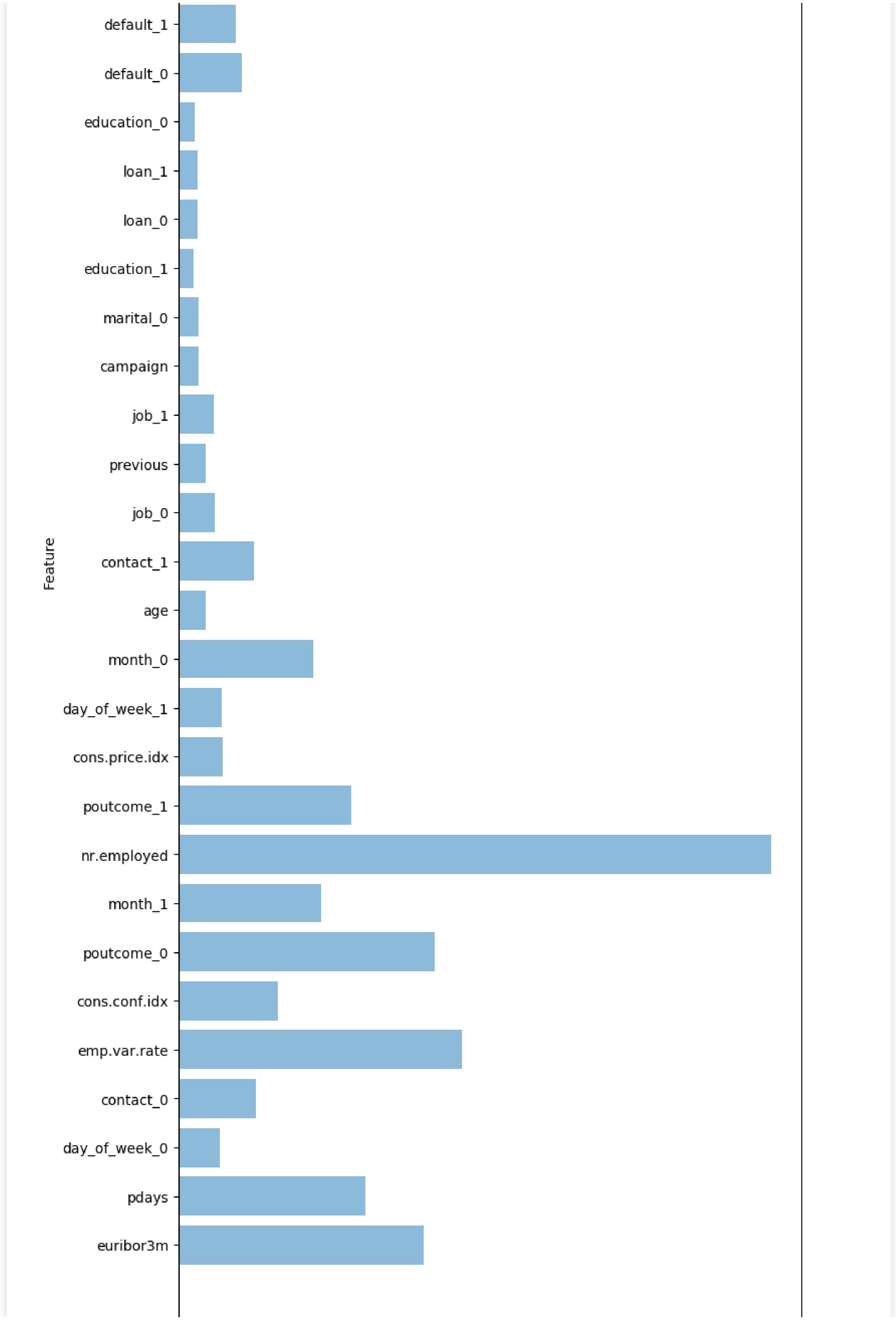
```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The
default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this
warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

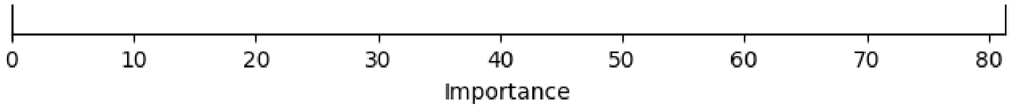
```
For values of best alpha = 2000 The train AUC is: 0.8250426975361628
For values of best alpha = 2000 The test AUC is: 0.803683987322812
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt; plt.rcdefaults()
feature_importance = x_cfl.get_booster().get_score(importance_type='gain')

objects = feature_importance.keys()
y_pos = np.arange(len(objects))
performance = feature_importance.values()
plt.figure(figsize=(8,20))
plt.barh(y_pos, performance, align='center', alpha=0.5)
plt.yticks(y_pos, objects)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance Graph')
plt.show()
```




Feature
default_1
default_0
education_0
loan_1
loan_0
education_1
marital_0
campaign
job_1
previous
job_0
contact_1
age
month_0
day_of_week_1
cons.price.idx
poutcome_1
nr.employed
month_1
poutcome_0
cons.conf.idx
emp.var.rate
contact_0
day_of_week_0
pdays
euribor3m



Conclusion

Encoding	Model	Train AUC	Test AUC
One hot encoding	Knn	0.856	0.772
	Logistic Regression	0.790	0.798
	Linear SVM	0.754	0.778
	RBF kernal SVM	0.854	0.784
	Random Forest	0.999	0.785
	XGBoost	0.756	0.764
	XGBoost + Search	0.861	0.792
Response Coding	Knn	0.857	0.758
	Logistic Regression	0.793	0.778
	Linear SVM	0.753	0.757
	RBF kernal SVM	0.786	0.750
	Random Forest	0.999	0.775
	XGBoost	0.791	0.777
	XGBoost + Search	0.825	0.803

Steps followed to acheive 0.80 AUC

- The objective of this case study was not predict whether a customer will subscribe a term deposit or not given the data of the customer.
- There were a lot of categorical variables and some numerical variables which capture various information about the customer and the bank-customer relationship.
- First we did EDA and figured out that there is no null values for the data, and the data is imbalanced, where "no" is the majority class.
- After doing univariate analysis of we figured that day_of_week and month features does not help very much when it comes to predicting the target variable. But on the other hand, some numerical features tend to predict the target variable much better.
- After doing data visualization with T-SNE, it was clear that the data does not have a very well separation between the classes, as the two classes were highly overlapping even after trying different parameters.
- After basic data preprocessing, I encoded the categorical data into both One hot encoding method and also the response coding method.
- After implementing all the models, we saw that the model that gave the best performance was XGBoost with RandomizedSearchCV and the test AUC was 0.803 which was similar to the results the researchers got in the relevent research paper.
- The most important features or key attributes in predicting whether any customer will submit a term deposit are:
  - nr.employed
  - emp.var.rate
  - poutcome_success
  - euribor3m
  - and so on.