MGT-415: Data Science in Practice

Project Notebook
May 9, 2019

Do Carmo Maria   Genc Murat   Nyambuu Lkham   Trichilo Giulio   Xi Fan

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

I Python Dependencies

```
In [1]: ! pip install missingno #missing data
        ! pip install inblearn #over/undersampling
```

```
Requirement already satisfied: missingno in
/home/zaratras/anaconda3/lib/python3.7/site-packages (0.4.1)
Requirement already satisfied: numpy in /home/zaratras/anaconda3/lib/python3.7/site-
packages (from missingno) (1.16.3)
Requirement already satisfied: scipy in /home/zaratras/anaconda3/lib/python3.7/site-
packages (from missingno) (1.2.1)
Requirement already satisfied: seaborn in /home/zaratras/anaconda3/lib/python3.7/site-
packages (from missingno) (0.9.0)
Requirement already satisfied: matplotlib in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from missingno) (3.0.3)
Requirement already satisfied: pandas>=0.15.2 in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from seaborn->missingno)
(0.24.2)
Requirement already satisfied: cycler>=0.10 in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from matplotlib->missingno)
(0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from matplotlib->missingno)
(1.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from matplotlib->missingno)
(2.3.1)
Requirement already satisfied: python-dateutil>=2.1 in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from matplotlib->missingno)
(2.8.0)
Requirement already satisfied: pytz>=2011k in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from
pandas>=0.15.2->seaborn->missingno) (2018.9)
Requirement already satisfied: six in /home/zaratras/anaconda3/lib/python3.7/site-
packages (from cycler>=0.10->matplotlib->missingno) (1.12.0)
Requirement already satisfied: setuptools in
/home/zaratras/anaconda3/lib/python3.7/site-packages (from
kiwisolver>=1.0.1->matplotlib->missingno) (40.8.0)
Collecting inblearn
  Could not find a version that satisfies the requirement inblearn (fromversions: )
No matching distribution found for inblearn
```

II Dataset Analysis

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

        ######################################

        import numpy as np
```

```
import scipy.stats as sc
import pandas as pd

import seaborn as sns
import matplotlib as mpl

import matplotlib.pyplot as plt
from IPython.display import display

import statsmodels.api as sm

from sklearn.model_selection import train_test_split
from statsmodels.stats import outliers_influence as oi

from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import log_loss
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import RFE
from sklearn import metrics


from sklearn.ensemble import IsolationForest
from sklearn.model_selection import cross_val_predict
#####################################################

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn import metrics

from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
#from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

######################################

#from sklearn.datasets import load_iris
```

```
from sklearn import preprocessing

from scipy import interp
from sklearn import datasets, neighbors
from sklearn.metrics import auc, roc_curve
from sklearn.model_selection import StratifiedKFold

from imblearn.over_sampling import ADASYN, SMOTE, RandomOverSampler, SMOTENC
from imblearn.combine import SMOTEENN
from imblearn.under_sampling import ClusterCentroids,
RandomUnderSampler,EditedNearestNeighbours


from imblearn.pipeline import make_pipeline
from imblearn import FunctionSampler
######################################
import warnings
warnings.filterwarnings("ignore")
pd.set_option('display.max_colwidth', 250)
sns.set()
```

II Basic Info

In [3]:
```
dsdata = pd.read_excel('Data/full_dataset.xlsx')
dsdata = dsdata.replace('unknown', np.nan)
dsdata = dsdata.replace('nonexistent', np.nan)

dsdata.head()
```

Out[3]:
```
   age        job  marital    education default housing loan    contact month  \
0   56  housemaid  married     basic.4y      no      no   no  telephone   may
1   57   services  married  high.school     NaN      no   no  telephone   may
2   37   services  married  high.school      no     yes   no  telephone   may
3   40     admin.  married     basic.6y      no      no   no  telephone   may
4   56   services  married  high.school      no      no  yes  telephone   may

  day_of_week  ...  campaign  pdays  previous  poutcome emp.var.rate  \
0         mon  ...         1    999         0       NaN          1.1
1         mon  ...         1    999         0       NaN          1.1
2         mon  ...         1    999         0       NaN          1.1
3         mon  ...         1    999         0       NaN          1.1
4         mon  ...         1    999         0       NaN          1.1

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0          93.994          -36.4      4.857       5191.0  no
1          93.994          -36.4      4.857       5191.0  no
2          93.994          -36.4      4.857       5191.0  no
3          93.994          -36.4      4.857       5191.0  no
4          93.994          -36.4      4.857       5191.0  no

[5 rows x 21 columns]
```

In [4]:
```
dsdata.dtypes
```

Out[4]:
```
age                    int64
job                   object
```

```
marital           object
education         object
default           object
housing           object
loan              object
contact           object
month             object
day_of_week       object
duration           int64
campaign           int64
pdays              int64
previous           int64
poutcome          object
emp.var.rate     float64
cons.price.idx   float64
cons.conf.idx    float64
euribor3m        float64
nr.employed      float64
y                 object
dtype: object
```

II Input variables:

a. bank client data: 1 - age (numeric)
2 - job : type of job (categorical: ‘admin.’,‘blue-collar’,‘entrepreneur’,‘housemaid’,‘management’,‘retired’,‘self-employed’,‘services’,‘student’,‘technician’,‘unemployed’,‘unknown’)
3 - marital : marital status (categorical: ‘divorced’,‘married’,‘single’,‘unknown’; note: ‘divorced’ means divorced or widowed)
4 - education (categorical: ‘basic.4y’,‘basic.6y’,‘basic.9y’,‘high.school’,‘illiterate’,‘professional.course’,‘university.degree’,‘unkn
5 - default: has credit in default? (categorical: ‘no’,‘yes’,‘unknown’)
6 - housing: has housing loan? (categorical: ‘no’,‘yes’,‘unknown’)
7 - loan: has personal loan? (categorical: ‘no’,‘yes’,‘unknown’)

1. related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: ‘cellular’,‘telephone’)
9 - month: last contact month of year (categorical: ‘jan’, ‘feb’, ‘mar’, . . . , ‘nov’, ‘dec’)
10 - day_of_week: last contact day of the week (categorical: ‘mon’,‘tue’,‘wed’,‘thu’,‘fri’)
11 - 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.

2. other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13 - 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)
14 - previous: number of contacts performed before this campaign and for this client (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: ‘failure’,‘nonexistent’,‘success’)

3. social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - nr.employed: number of employees - quarterly indicator (numeric)

4. Output variable (desired target):

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

II Variable Description

In [5]:
```
#describe the "Object" type elements

dsdata.describe(include=['O'])
```

Out[5]:

	job	marital	education	default	housing	loan	contact
count	40858	41108	39457	32591	40198	40198	41188
unique	11	3	7	2	2	2	2
top	admin.	married	university.degree	no	yes	no	cellular
freq	10422	24928	12168	32588	21576	33950	26144

	month	day_of_week	poutcome	y
count	41188	41188	5625	41188
unique	10	5	2	2
top	may	thu	failure	no
freq	13769	8623	4252	36548

In [6]:
```
#describe the "numerical" type elements

dsdata.describe()
```

Out[6]:

	age	duration	campaign	pdays	previous
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963
std	10.42125	259.279249	2.770014	186.910907	0.494901
min	17.00000	0.000000	1.000000	0.000000	0.000000
25%	32.00000	102.000000	1.000000	999.000000	0.000000
50%	38.00000	180.000000	2.000000	999.000000	0.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000
max	98.00000	4918.000000	56.000000	999.000000	7.000000

	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	0.081886	93.575664	-40.502600	3.621291	5167.035911
std	1.570960	0.578840	4.628198	1.734447	72.251528
min	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	1.400000	94.767000	-26.900000	5.045000	5228.100000

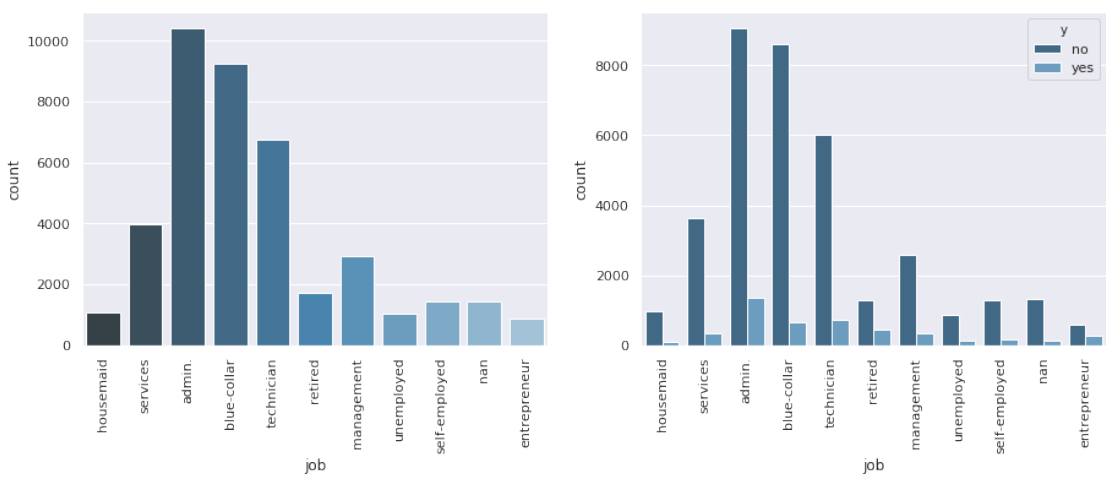
In [7]:
```
dsdata['y'].value_counts()
```

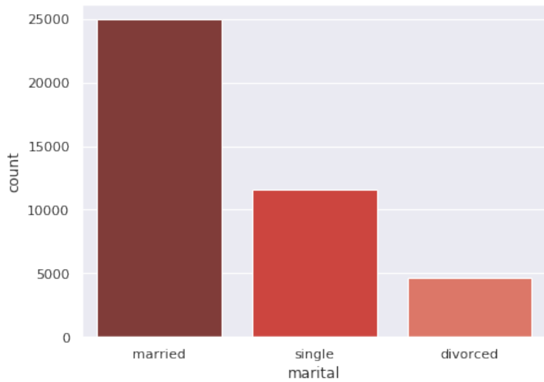
```
Out[7]: no      36548
        yes      4640
        Name: y, dtype: int64
```

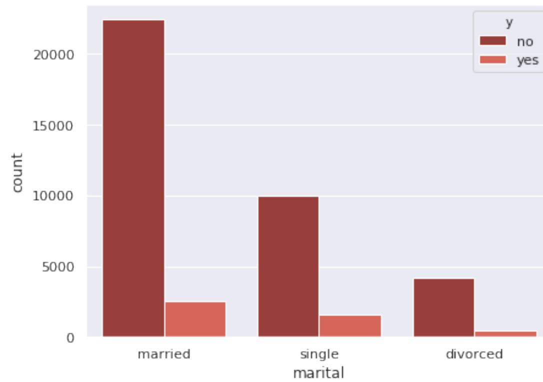
```
In [ ]:
```

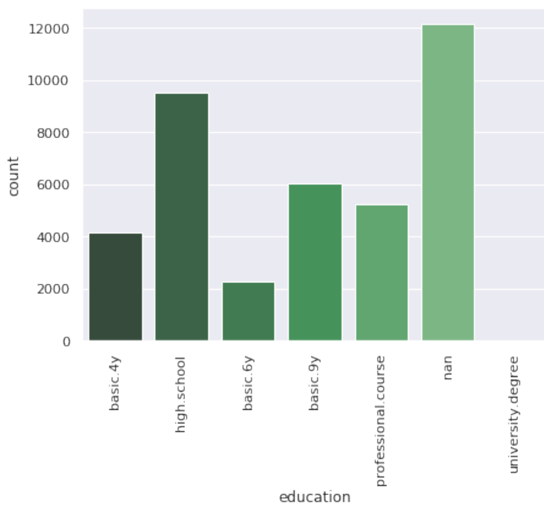
II Variable Distribution

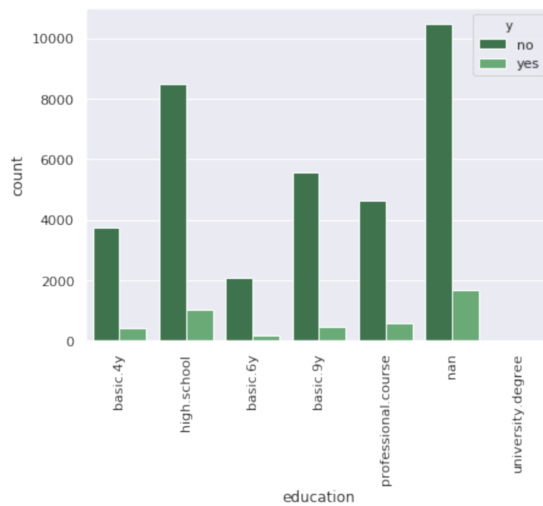
```
In [8]: # Barplots for categorical (object) variables
        column = "job"
        colors = ['Blues_d', 'Reds_d', 'Greens_d', 'Oranges_d', 'Blues_r', 'Reds_r', 'Greens_r',
        'Oranges_r', 'mako', 'Reds_d','BuPu_r']
        color_coef=0
        for column in dsdata.columns:
            if dsdata[column].dtype == "O":
                if column=='job' or column=='education':
                    fig,ax=plt.subplots(1,2,figsize=(15,5))
                    t = sns.countplot(x=column, data=dsdata, palette=colors[color_coef],
        ax=ax[0])
                    t = t.set_xticklabels(dsdata[column].unique(), rotation=90)
                    g = sns.countplot(x=dsdata[column], hue=dsdata['y'],
        palette=colors[color_coef], ax=ax[1])
                    t = g.set_xticklabels(dsdata[column].unique(), rotation=90)
                else:
                    fig,ax=plt.subplots(1,2,figsize=(15,5))
                    t = sns.countplot(x=column, data=dsdata, palette=colors[color_coef],
        ax=ax[0])
                    g = sns.countplot(x=dsdata[column], hue=dsdata['y'],
        palette=colors[color_coef], ax=ax[1])
                color_coef +=1
```

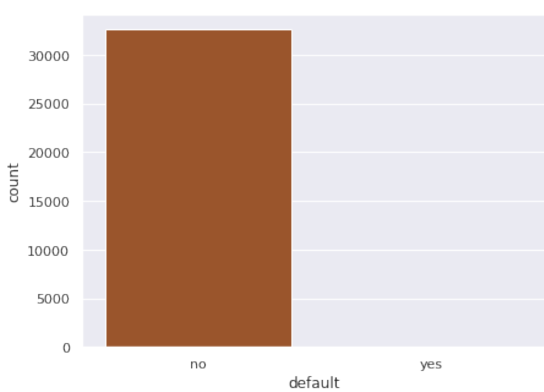


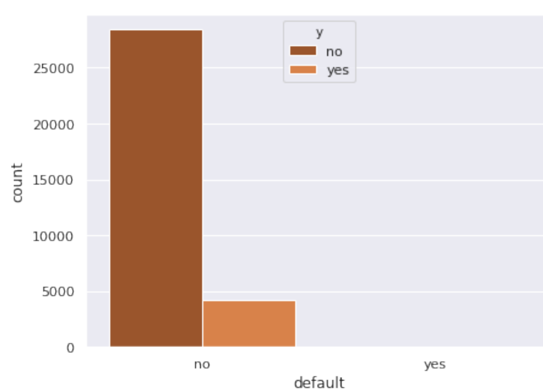

25000
20000
15000
10000
5000
0
count
married
single
divorced
marital

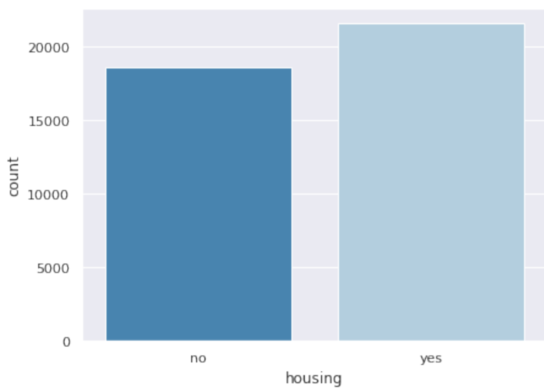

y
no
yes
20000
15000
10000
5000
0
count
married
single
divorced
marital

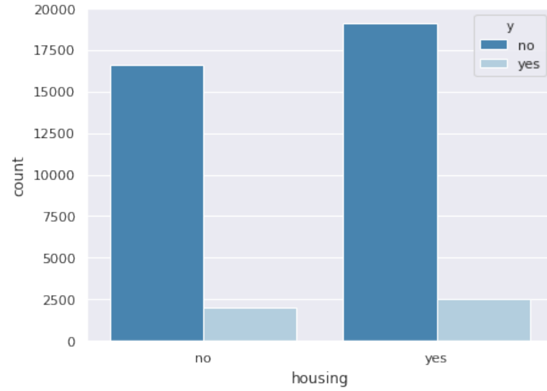

12000
10000
8000
6000
4000
2000
0
count
basic.4y
high.school
basic.6y
basic.9y
professional.course
nan
university.degree
education

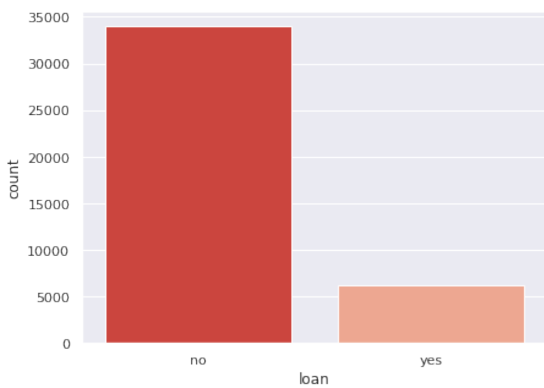

y
no
yes
10000
8000
6000
4000
2000
0
count
basic.4y
high.school
basic.6y
basic.9y
professional.course
nan
university.degree
education

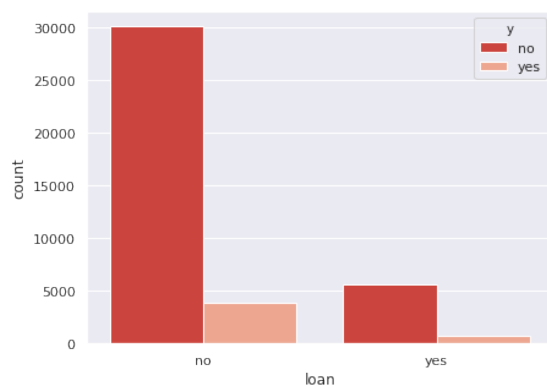

30000
25000
20000
15000
10000
5000
0
count
no
yes
default

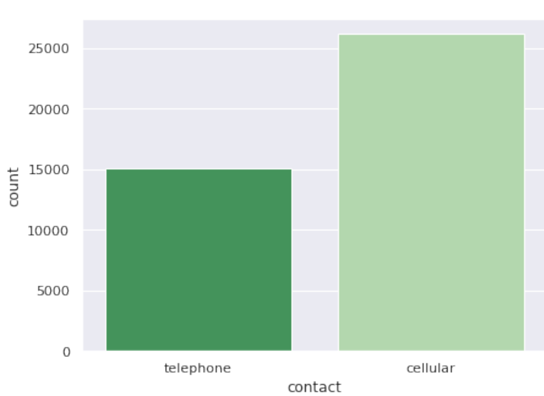

y
no
yes
25000
20000
15000
10000
5000
0
count
no
yes
default

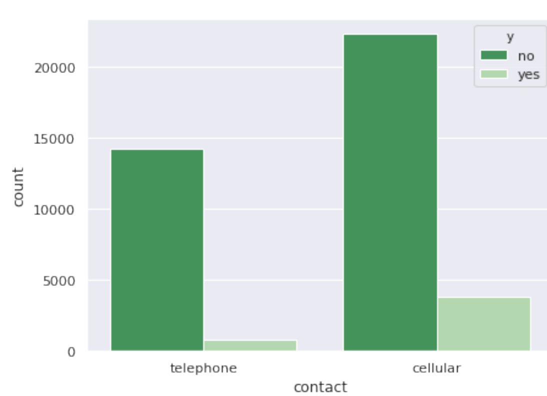

count
20000
15000
10000
5000
0
no
yes
housing

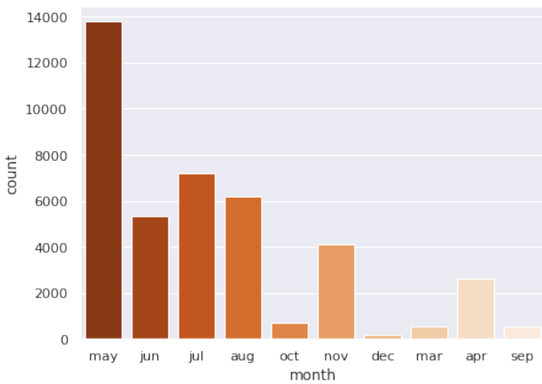

count
20000
17500
15000
12500
10000
7500
5000
2500
0
no
yes
housing
y
no
yes

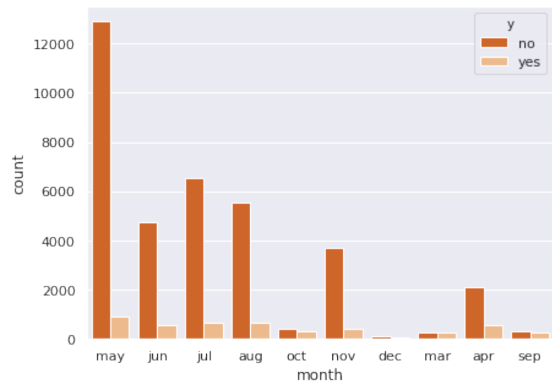

count
35000
30000
25000
20000
15000
10000
5000
0
no
yes
loan

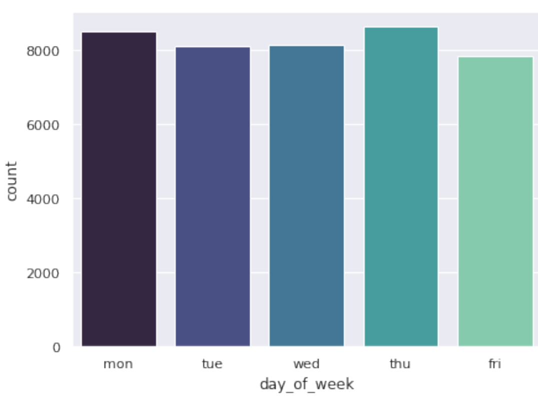

count
30000
25000
20000
15000
10000
5000
0
no
yes
loan
y
no
yes

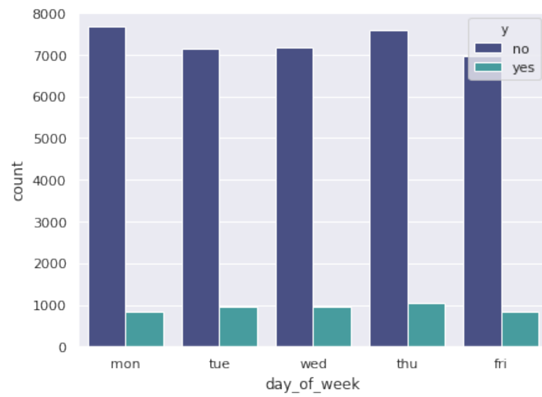

count
25000
20000
15000
10000
5000
0
telephone
cellular
contact

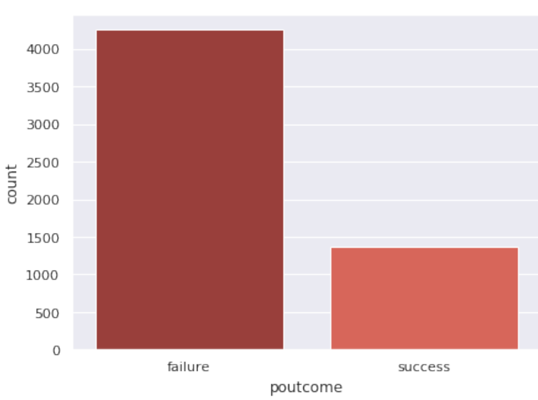

count
20000
15000
10000
5000
0
telephone
cellular
contact
y
no
yes

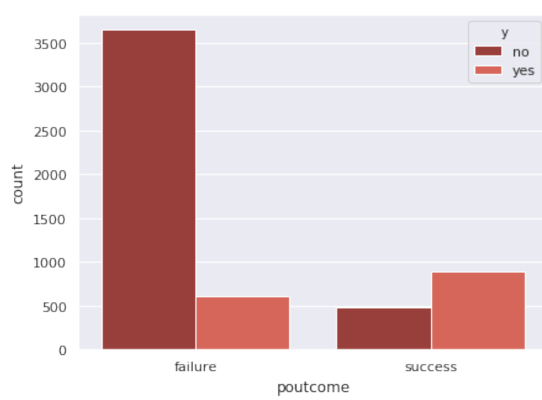

14000
12000
10000
8000
6000
4000
2000
0
count
may
jun
jul
aug
oct
nov
dec
mar
apr
sep
month

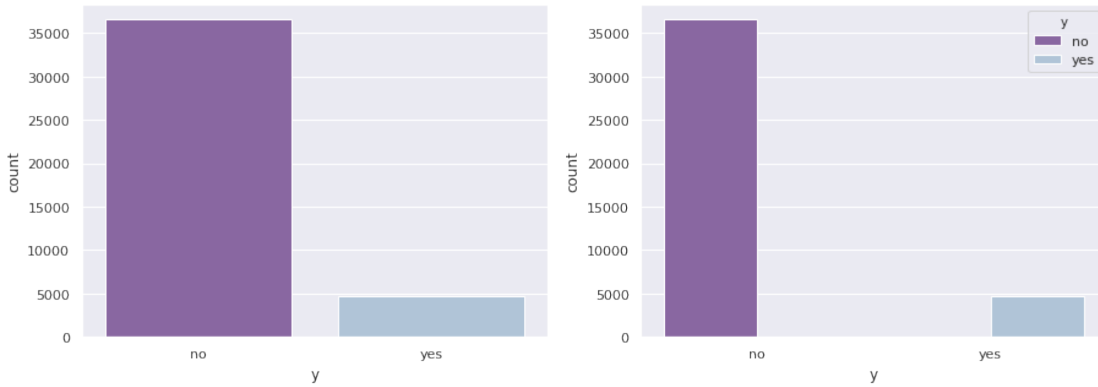

y
no
yes
12000
10000
8000
6000
4000
2000
0
count
may
jun
jul
aug
oct
nov
dec
mar
apr
sep
month


8000
6000
4000
2000
0
count
mon
tue
wed
thu
fri
day_of_week


y
no
yes
8000
7000
6000
5000
4000
3000
2000
1000
0
count
mon
tue
wed
thu
fri
day_of_week

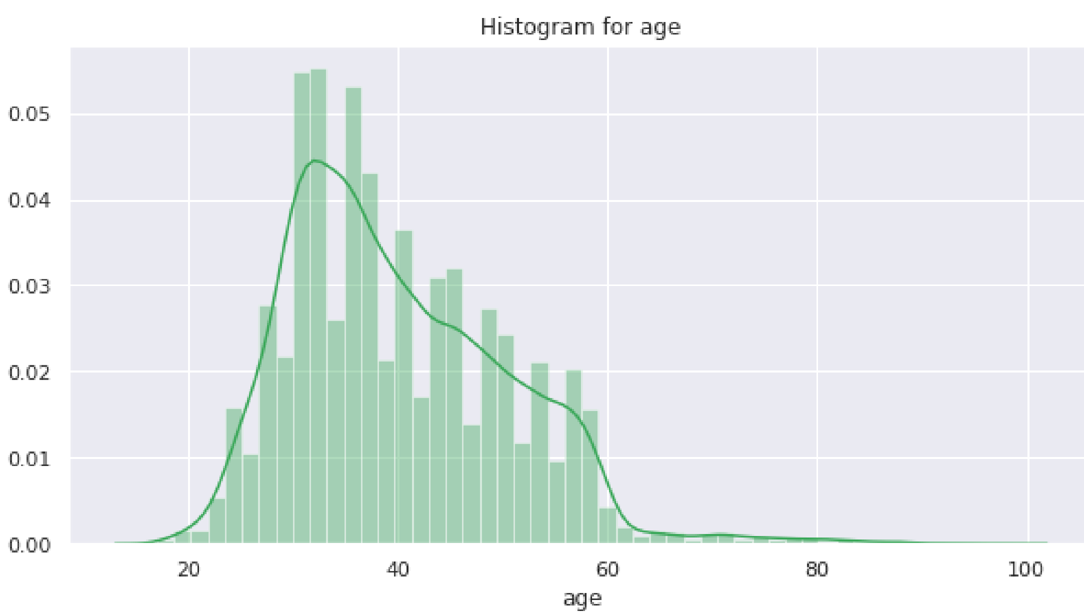

4000
3500
3000
2500
2000
1500
1000
500
0
count
failure
success
poutcome

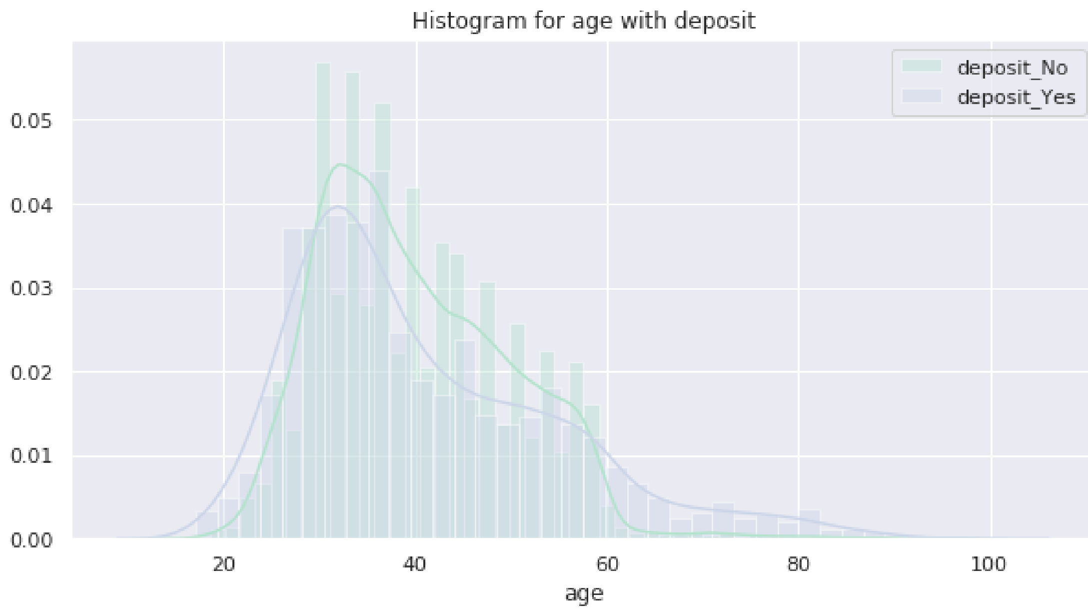

y
no
yes
3500
3000
2500
2000
1500
1000
500
0
count
failure
success
poutcome

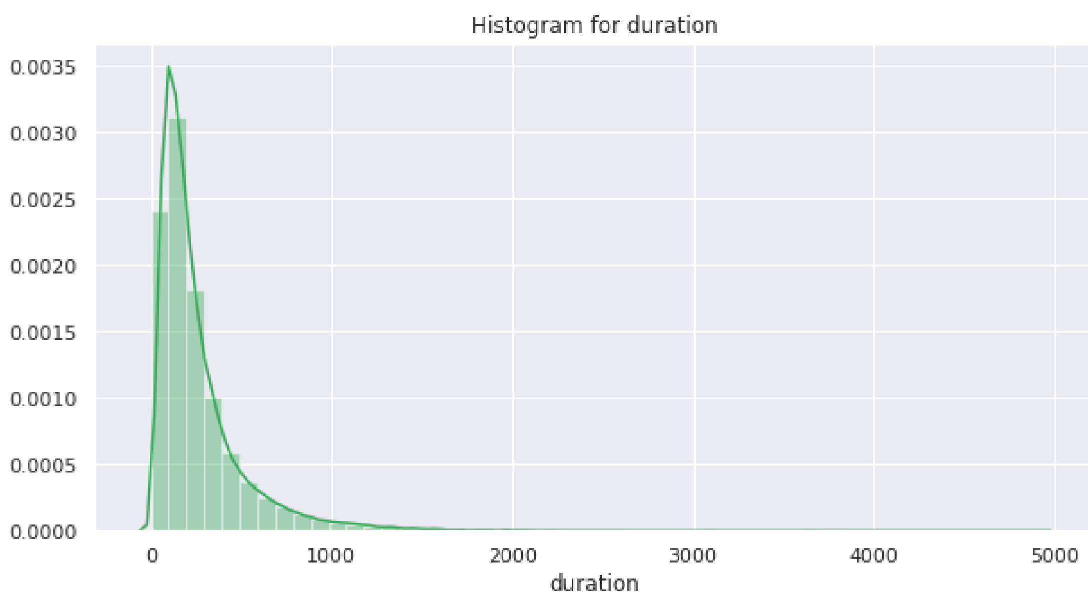


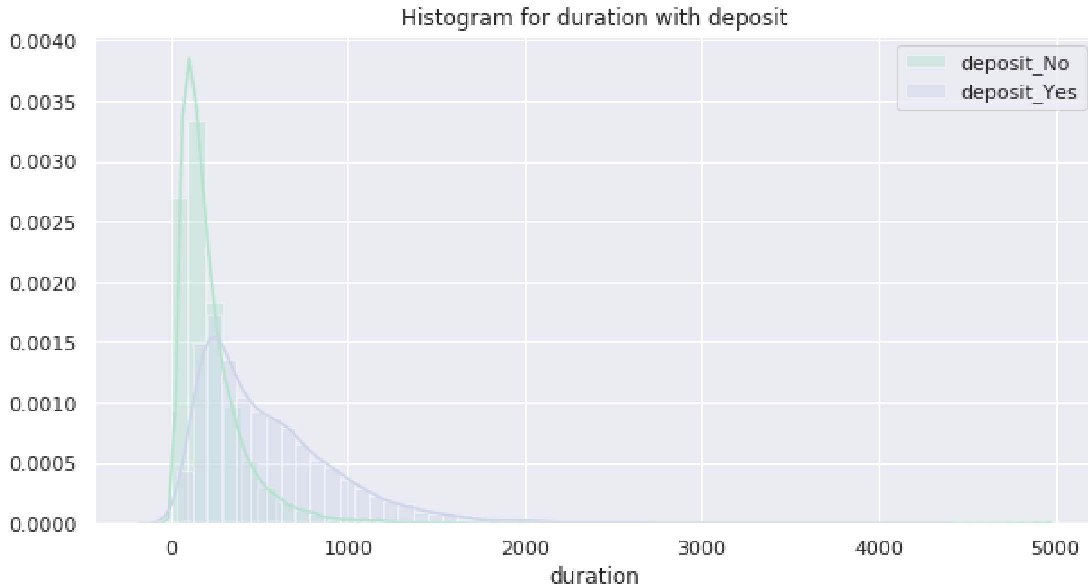

```
In [9]: def histogram(variable):
            plt.figure(figsize=(10, 5))
            plt.title("Histogram for {}".format(variable))
            ax = sns.distplot(dsdata[variable],color=sns.color_palette("RdYlGn_r")[0])

        def histogram_by_deposit(feature):
            plt.figure(figsize=(10, 5))
            plt.title("Histogram for {} with deposit".format(feature))
            ax0 = sns.distplot(dsdata[dsdata["y"]=="no"][feature],color=sns.color_palette("Paste
        l2")[0],label="deposit_No")
            ax1 = sns.distplot(dsdata[dsdata["y"]=="yes"][feature],color=sns.color_palette("Past
        el2")[2],label="deposit_Yes")
            plt.legend()
```

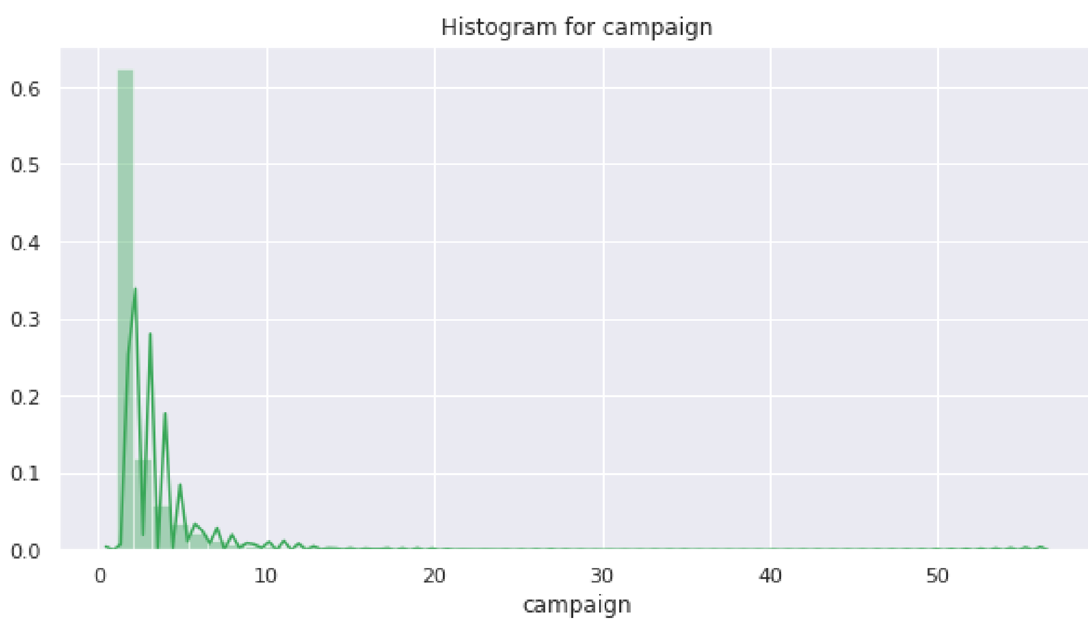
```
In [10]: # histogram for numerical variables
         column = "job"
         for column in dsdata.columns:
             if dsdata[column].dtype == "int64" or dsdata[column].dtype == "float64":
                 histogram(column)
                 histogram_by_deposit(column)
```

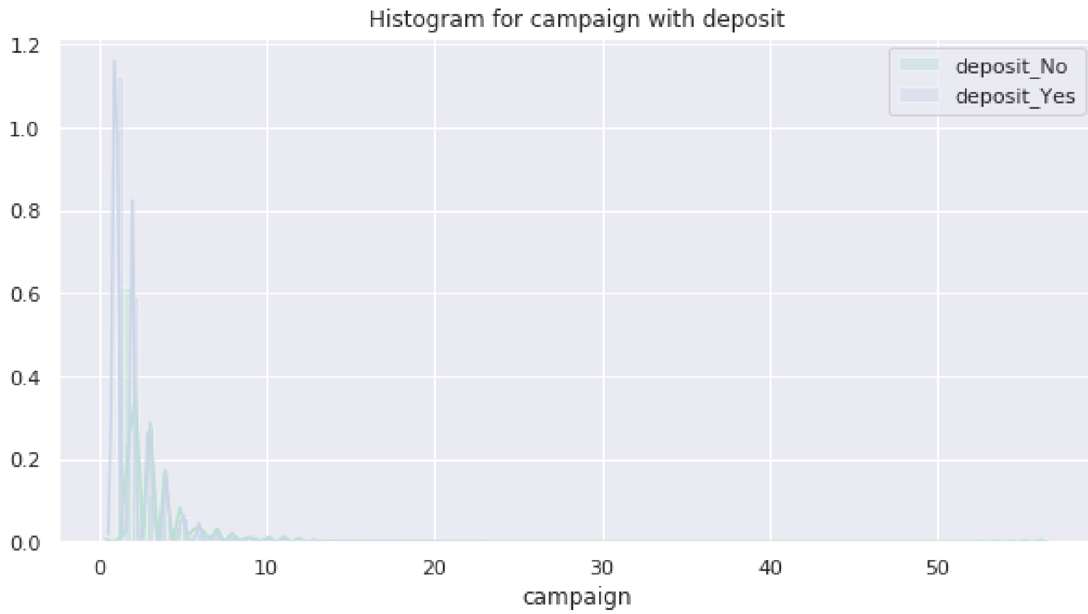


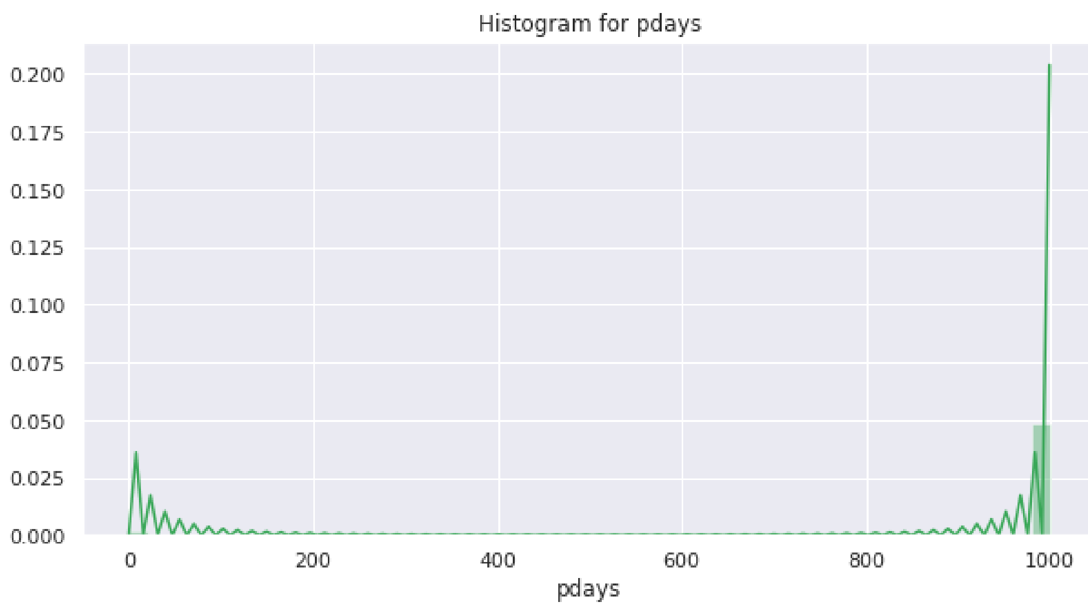

Histogram for age with deposit
deposit_No
deposit_Yes
0.00
0.01
0.02
0.03
0.04
0.05
20
40
60
80
100
age

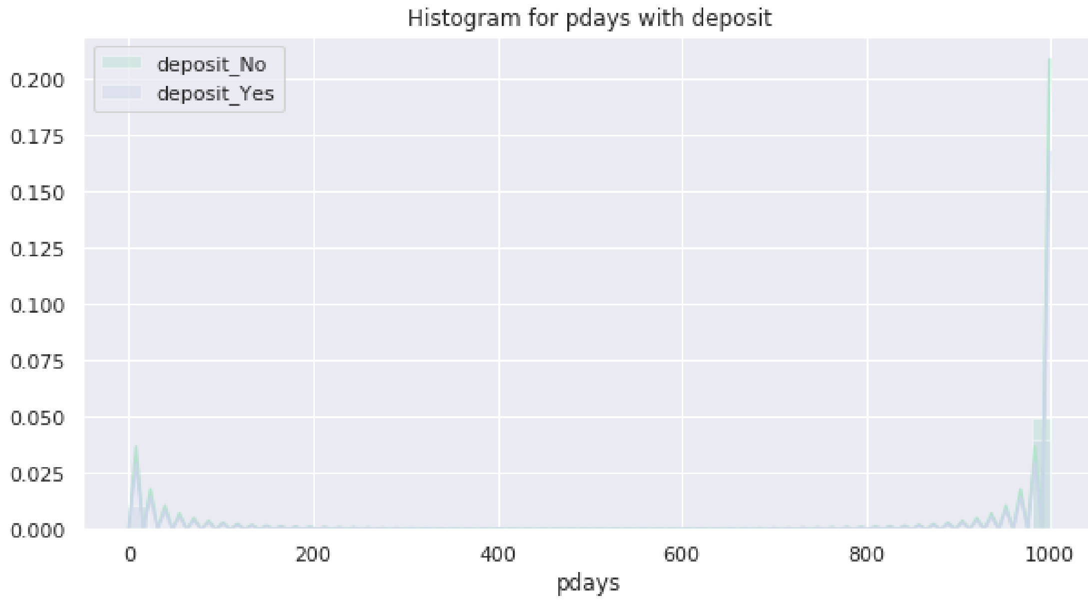

Histogram for duration
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
0
1000
2000
3000
4000
5000
duration

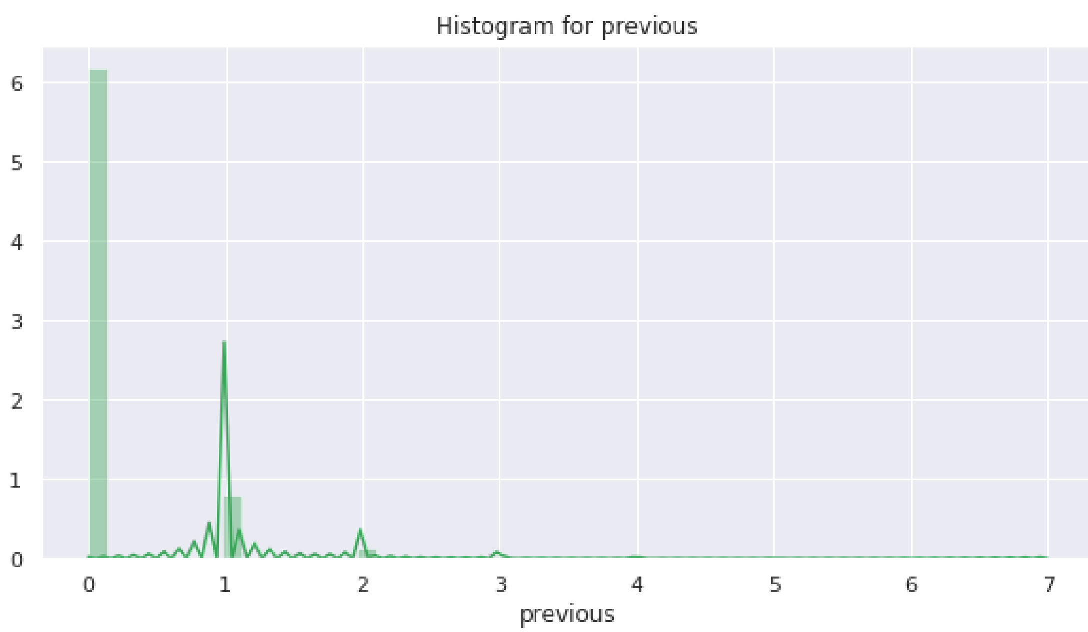

Histogram for duration with deposit
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
1000
2000
3000
4000
5000
duration
deposit_No
deposit_Yes

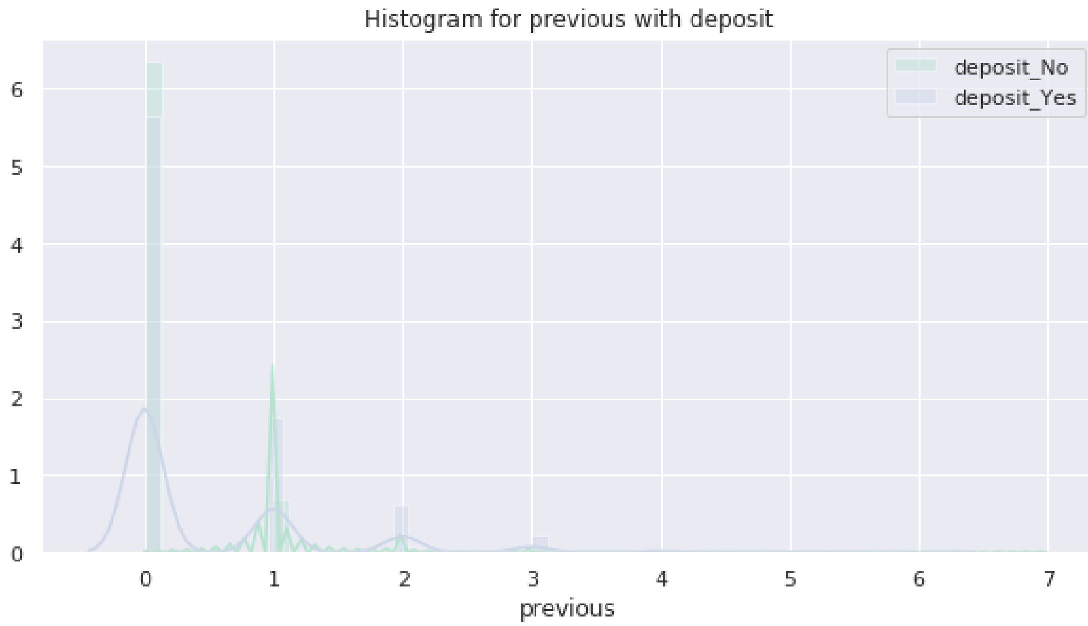

Histogram for campaign
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
10
20
30
40
50
campaign

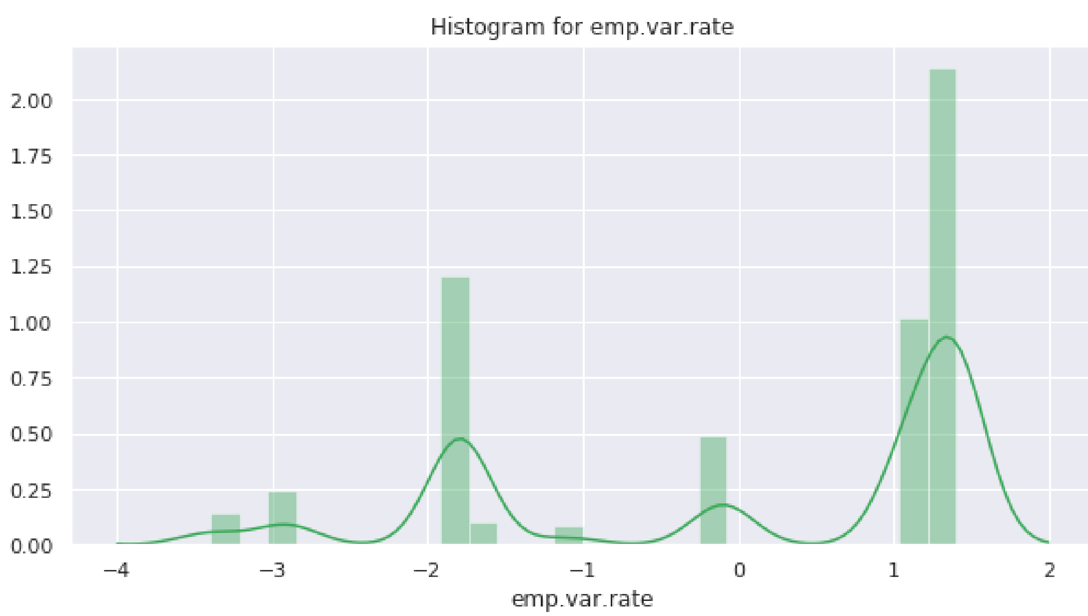

Histogram for campaign with deposit
deposit_No
deposit_Yes
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
campaign

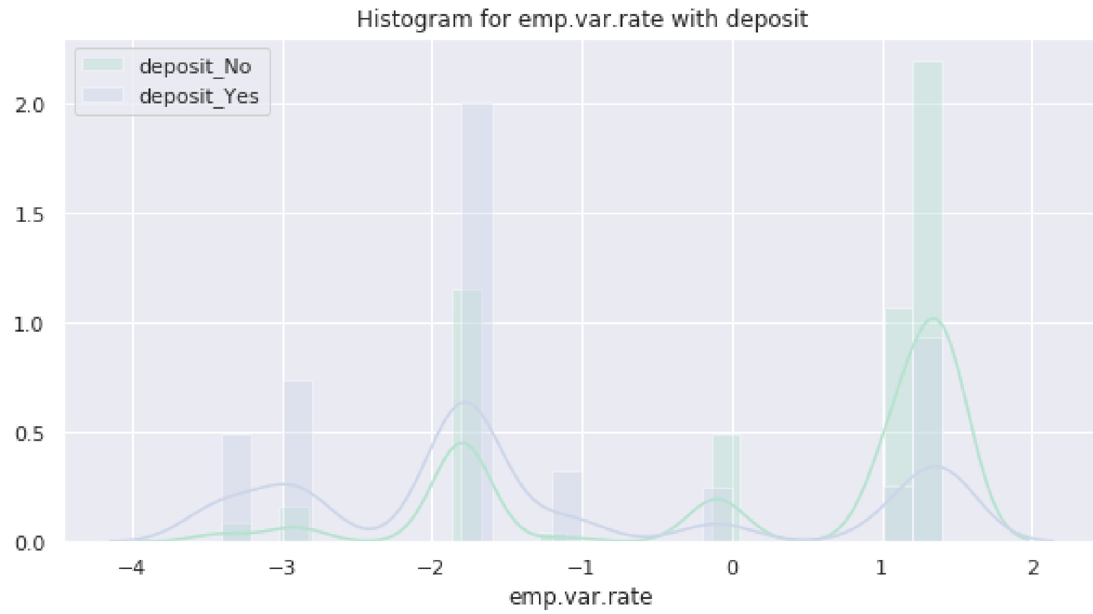

Histogram for pdays
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
200
400
600
800
1000
pdays

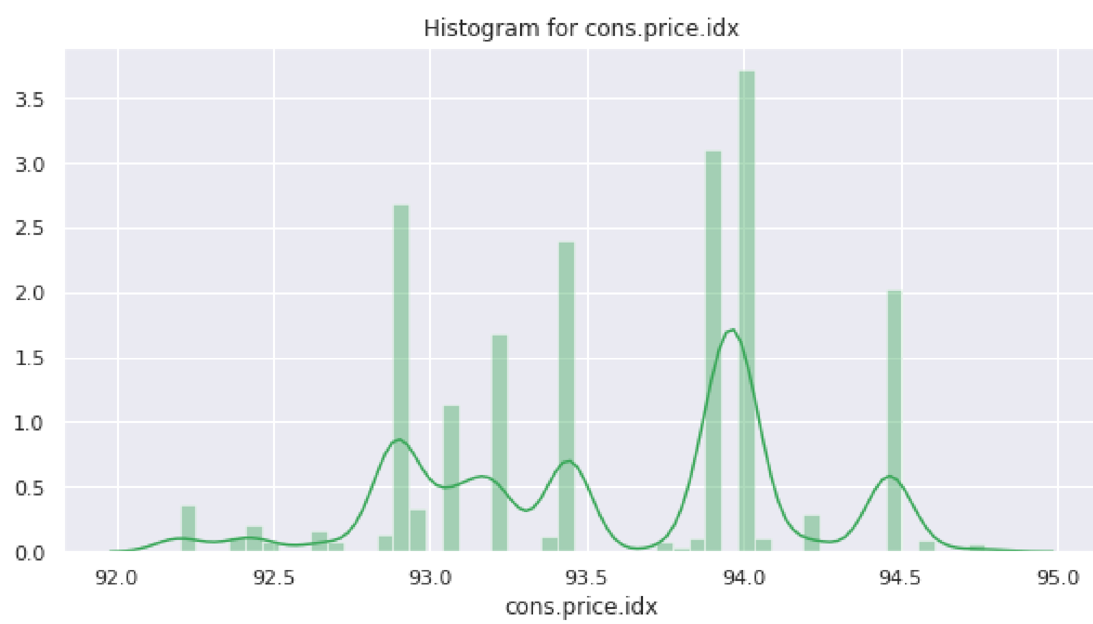

Histogram for pdays with deposit
deposit_No
deposit_Yes
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
200
400
600
800
1000
pdays

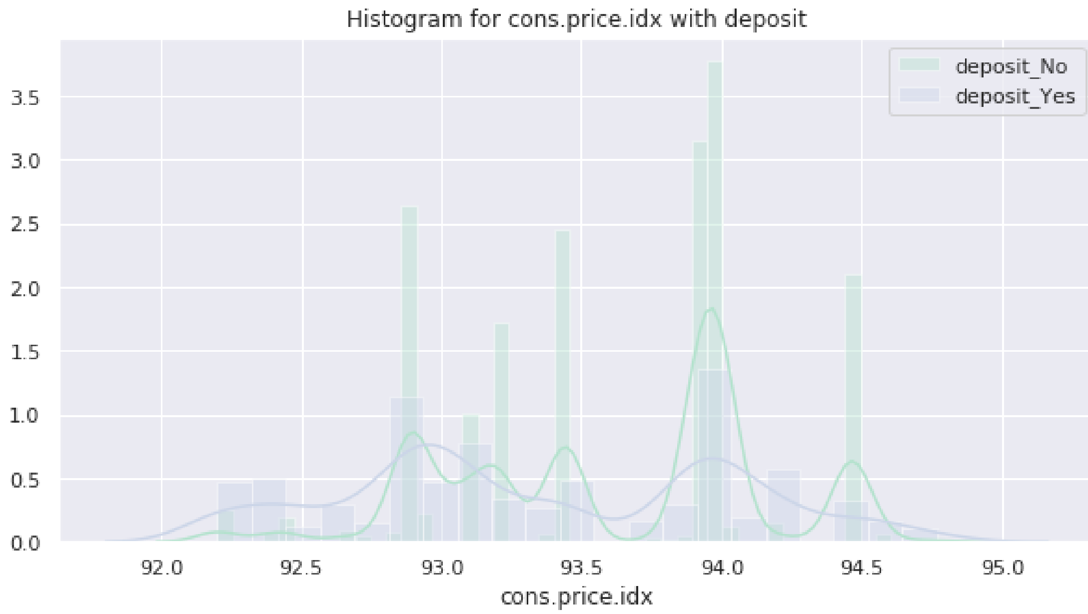

Histogram for previous
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
previous

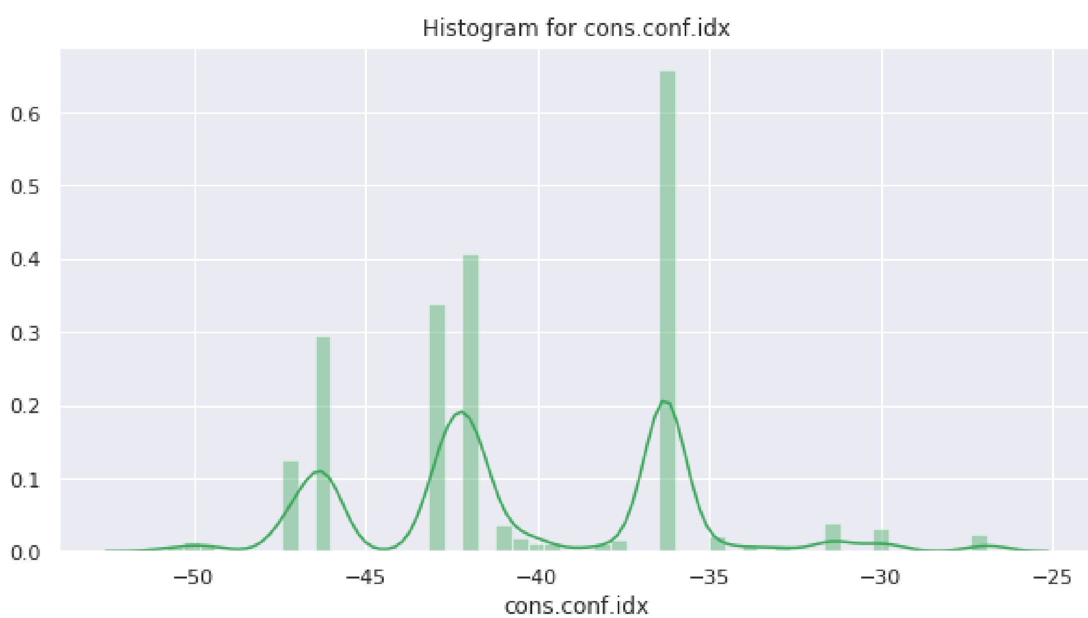

Histogram for previous with deposit
deposit_No
deposit_Yes
0
1
2
3
4
5
6
0
1
2
3
4
5
6
7
previous

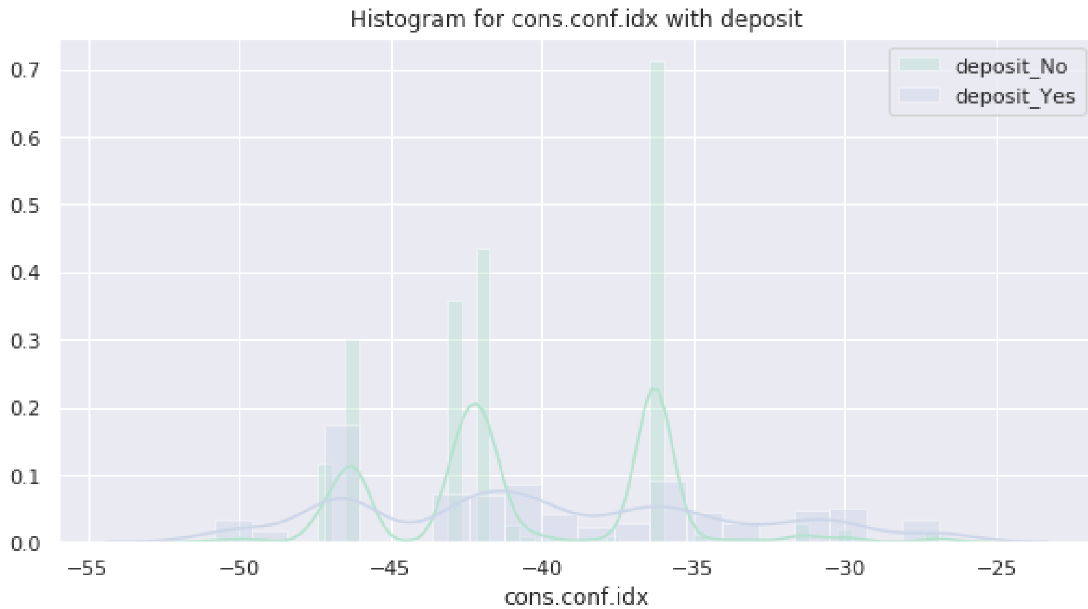

Histogram for emp.var.rate
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
−4
−3
−2
−1
0
1
2
emp.var.rate

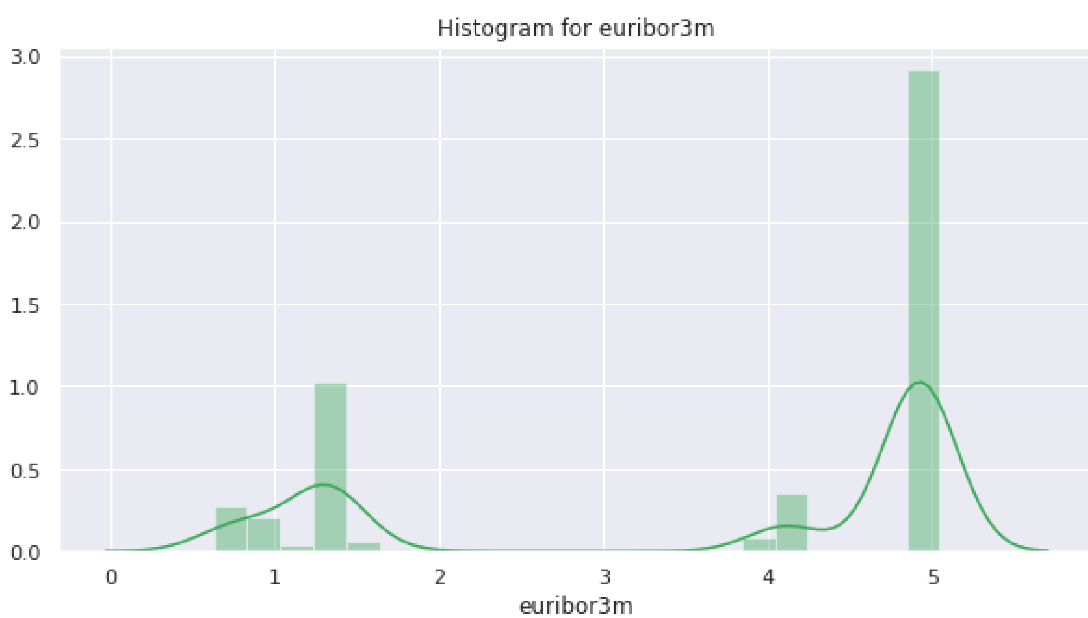

Histogram for emp.var.rate with deposit
deposit_No
deposit_Yes
2.0
1.5
1.0
0.5
0.0
−4
−3
−2
−1
0
1
2
emp.var.rate

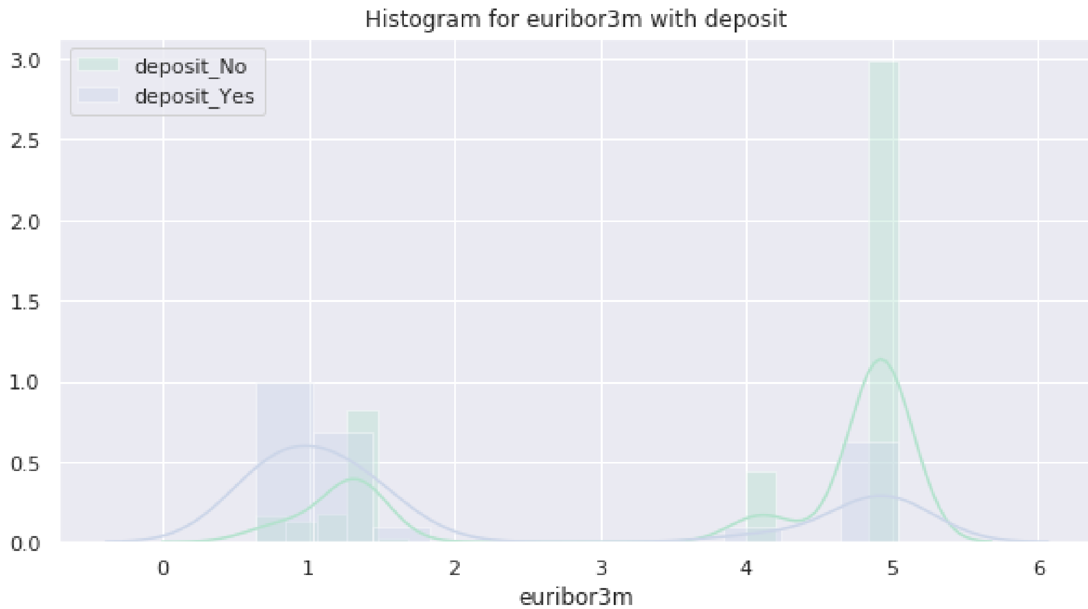

Histogram for cons.price.idx
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
92.0
92.5
93.0
93.5
94.0
94.5
95.0
cons.price.idx

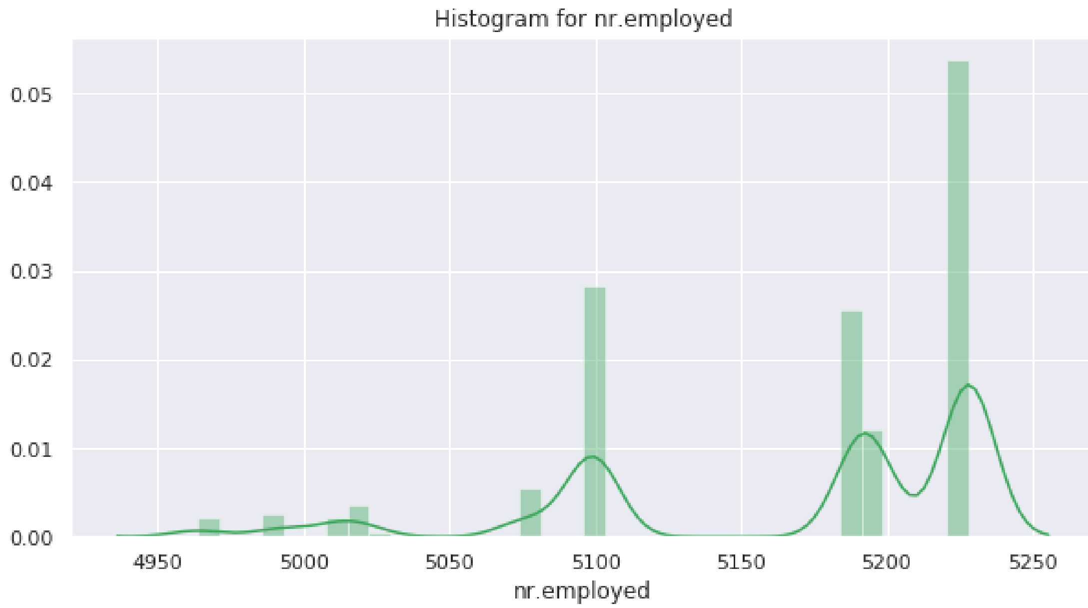

Histogram for cons.price.idx with deposit
deposit_No
deposit_Yes
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
92.0
92.5
93.0
93.5
94.0
94.5
95.0
cons.price.idx

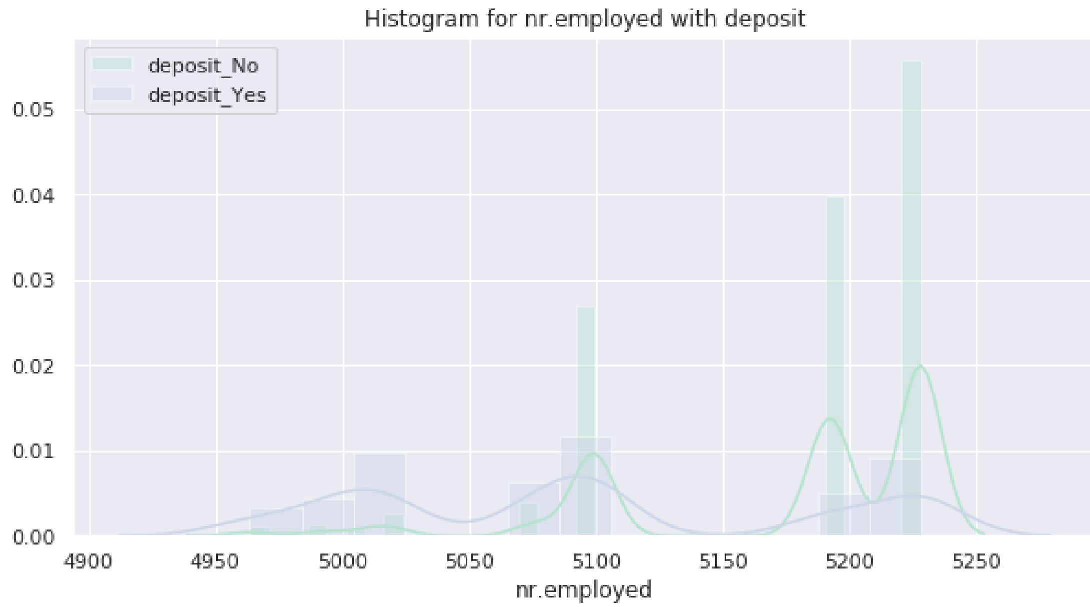

Histogram for cons.conf.idx
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−50
−45
−40
−35
−30
−25
cons.conf.idx


Histogram for cons.conf.idx with deposit
deposit_No
deposit_Yes
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−55
−50
−45
−40
−35
−30
−25
cons.conf.idx


Histogram for euribor3m
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
euribor3m

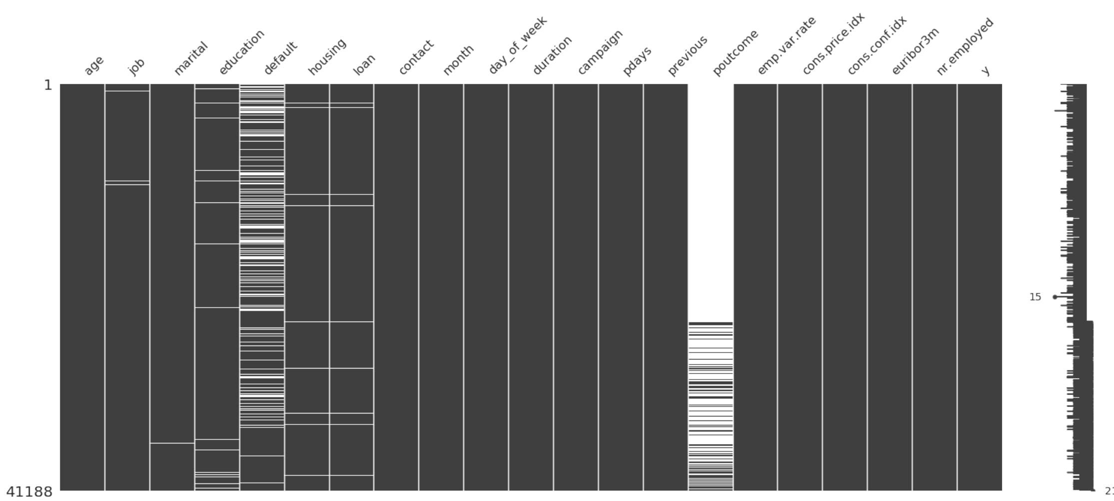

Histogram for euribor3m with deposit
deposit_No
deposit_Yes
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6
euribor3m


Histogram for nr.employed
0.05
0.04
0.03
0.02
0.01
0.00
4950
5000
5050
5100
5150
5200
5250
nr.employed

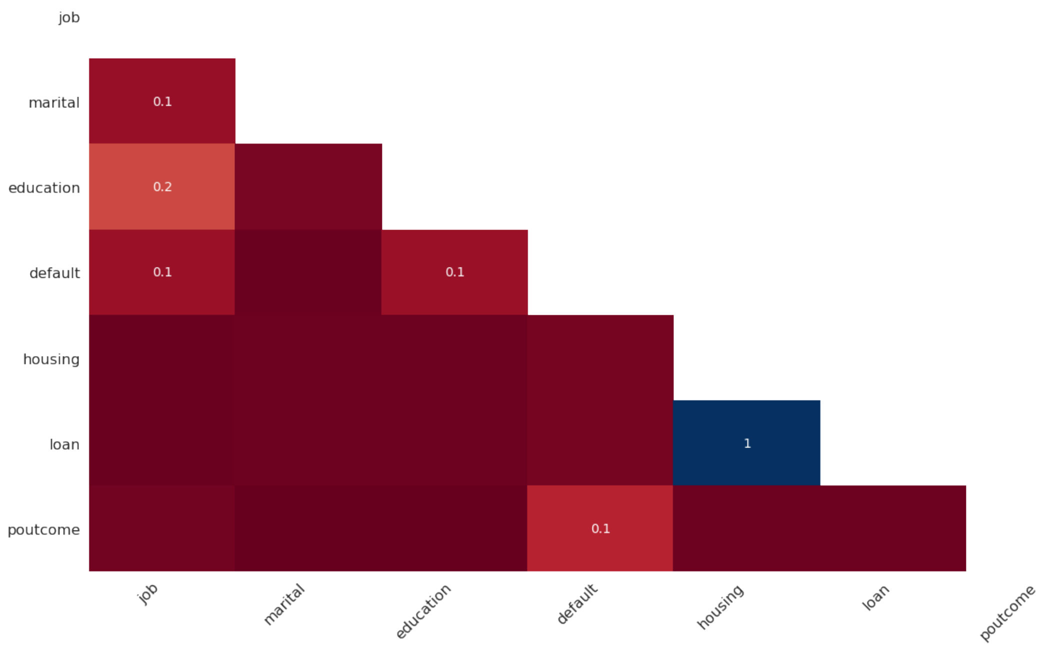


III Missing Data Handling

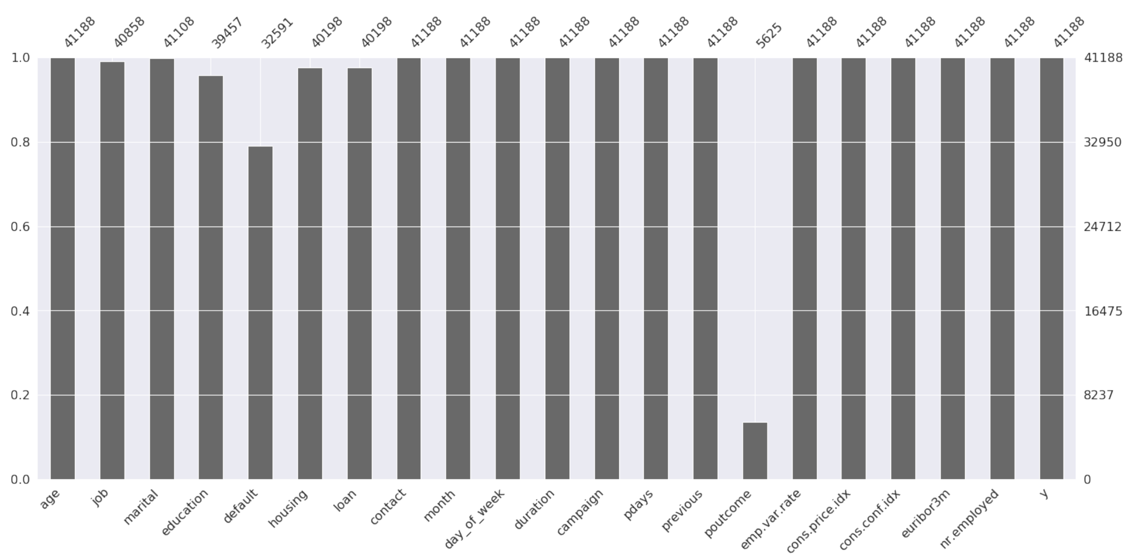
In [11]:
```
#visualizing missing data

msno.matrix(dsdata)
plt.show()
```



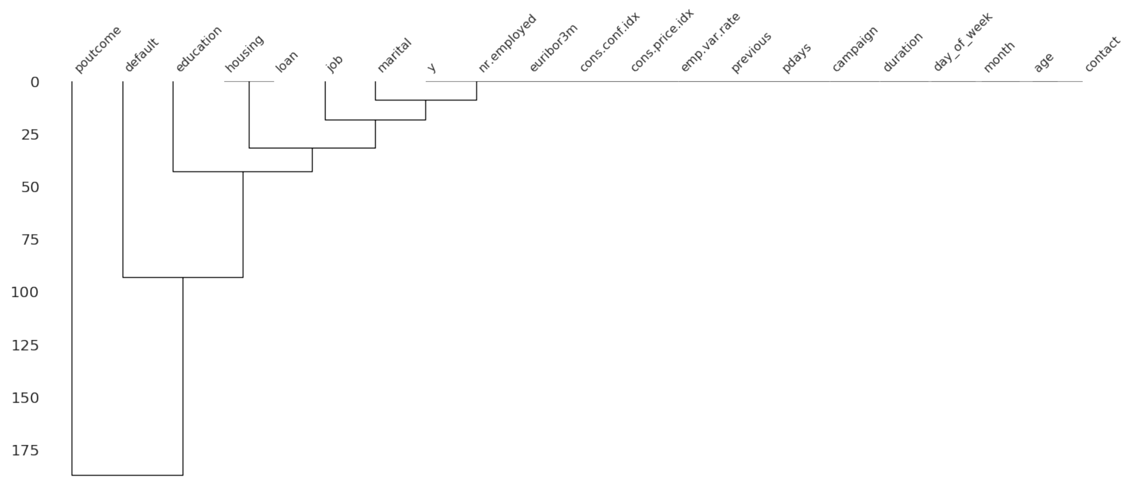
In [12]:
```
#nullity correlation: how strongly the presence or absence of one variable affects the
presence of another
msno.heatmap(dsdata)
plt.show()
missing = ['job','marital','education','default','housing','loan','poutcome']
ds_missing = dsdata[missing]
```




```
In [13]: msno.bar(dsdata)
         plt.show()
```




```
In [14]: msno.dendrogram(dsdata)
         plt.show()
```



In [15]: #ds_missing.head(25)

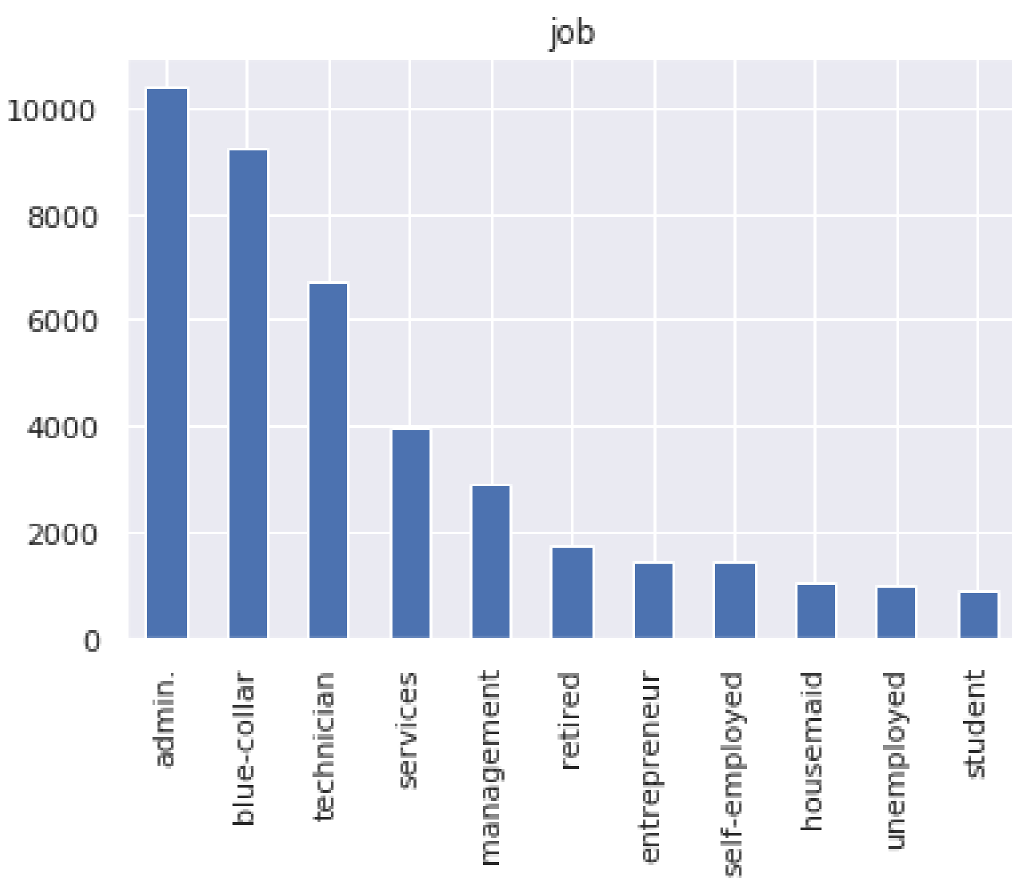
In [ ]:

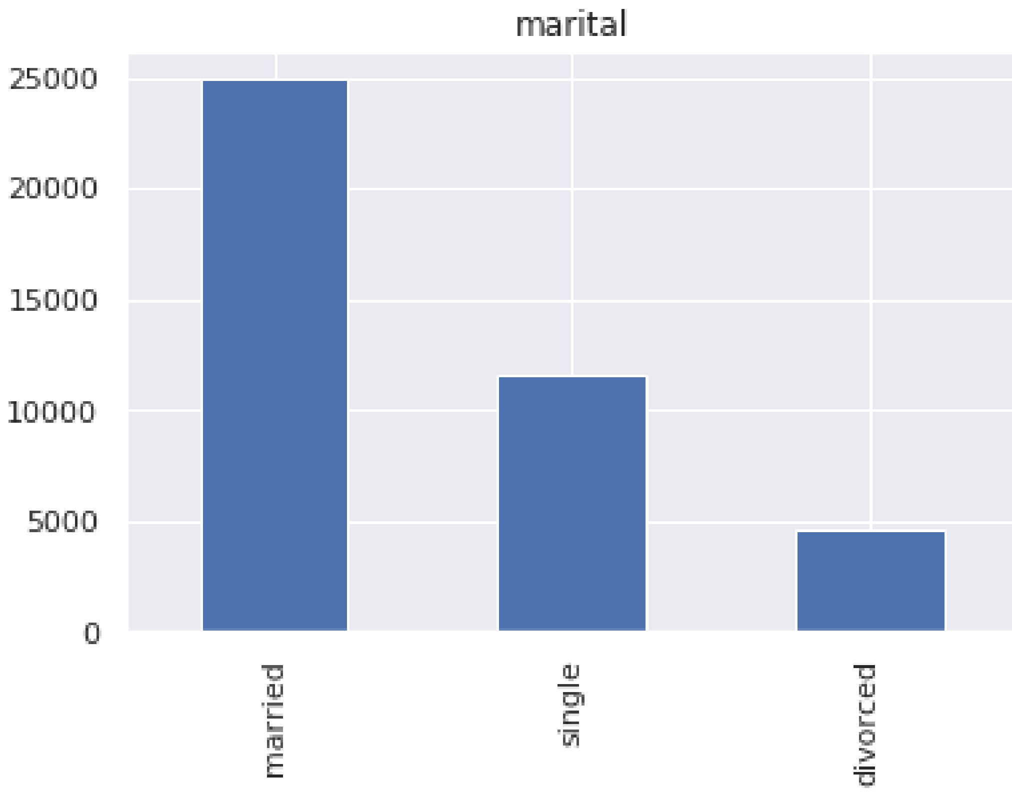
In [16]:
```
for col in list(ds_missing.columns):
    print(ds_missing[col].value_counts(),'\n')
```

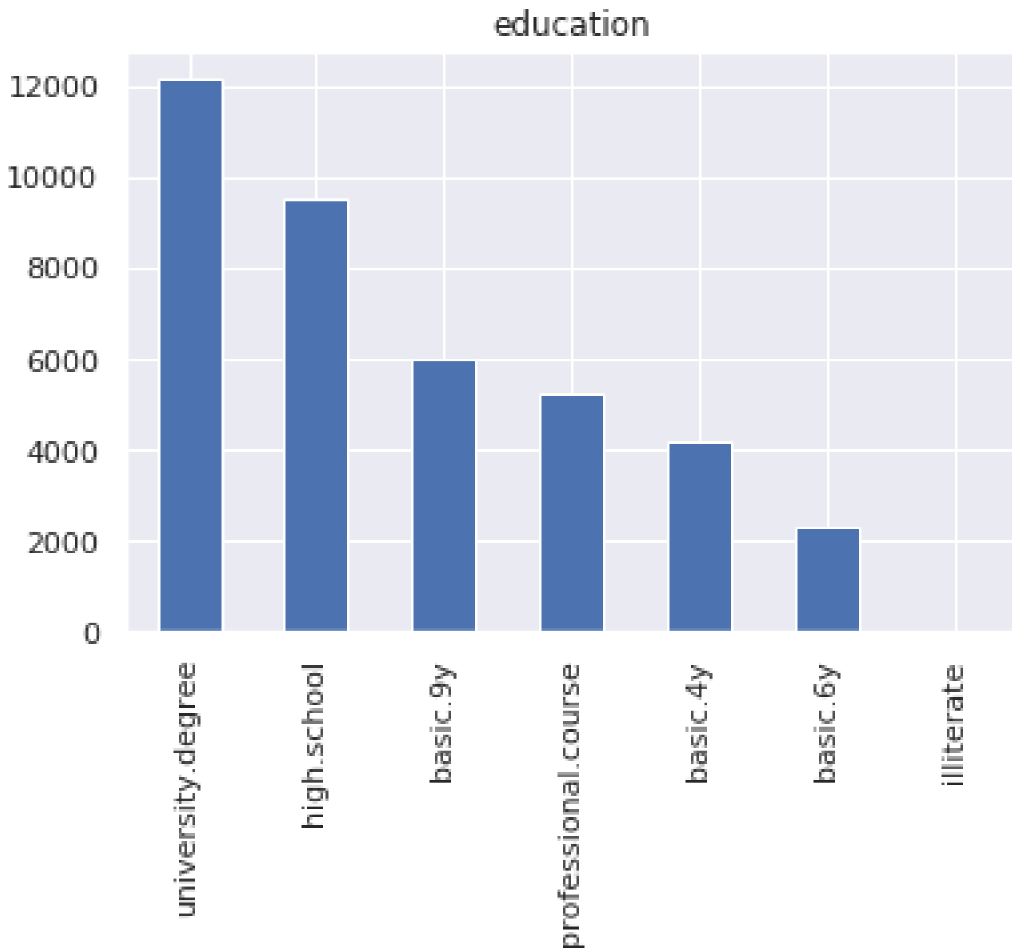
```
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
Name: job, dtype: int64

married     24928
single      11568
divorced     4612
Name: marital, dtype: int64

university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
illiterate                18
Name: education, dtype: int64

no     32588
yes        3
Name: default, dtype: int64
```

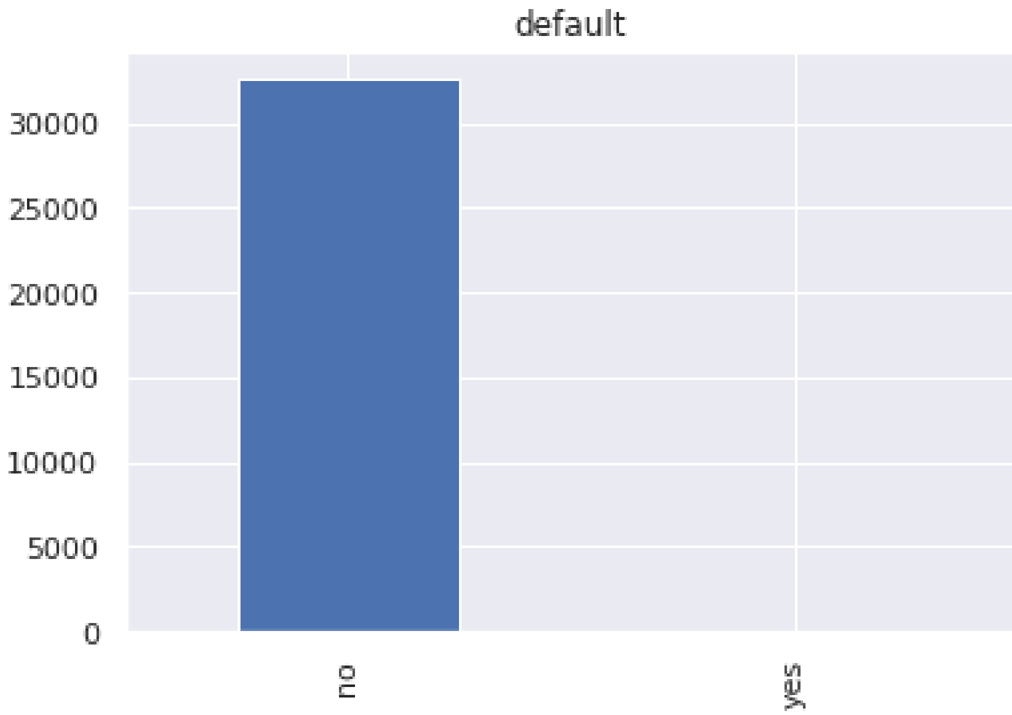
```
yes    21576
no     18622
Name: housing, dtype: int64

no     33950
yes     6248
Name: loan, dtype: int64

failure    4252
success    1373
Name: poutcome, dtype: int64
```

```
In [17]: for col in list(ds_missing.columns):
             if ds_missing[col].dtype == "O":
                 ds_missing[col].value_counts().plot(kind='bar')
                 plt.title(col)
                 plt.show()
```

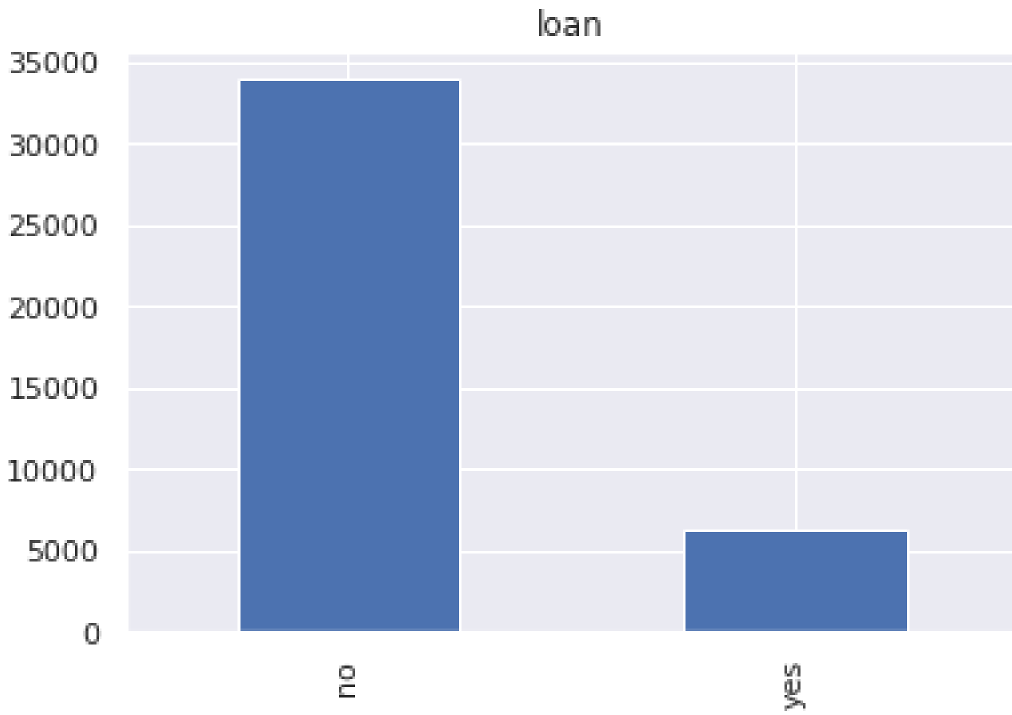


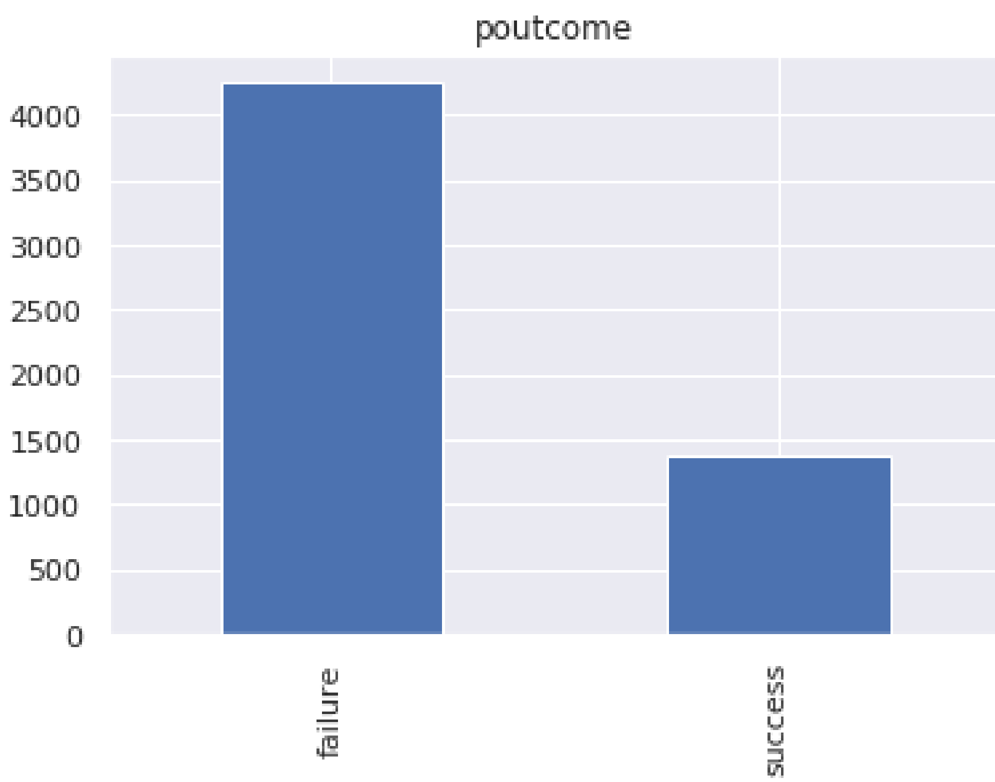

marital
25000
20000
15000
10000
5000
0
married
single
divorced


education
12000
10000
8000
6000
4000
2000
0
university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
illiterate


default
30000
25000
20000
15000
10000
5000
0
no
yes


housing
20000
15000
10000
5000
0
yes
no






```
In [18]: ds_missing.head(5)
```

```
Out[18]:          job  marital    education default housing loan poutcome
         0  housemaid  married     basic.4y      no      no   no      NaN
         1   services  married  high.school     NaN      no   no      NaN
         2   services  married  high.school      no     yes   no      NaN
         3     admin.  married     basic.6y      no      no   no      NaN
         4   services  married  high.school      no      no  yes      NaN
```

```
In [19]: missing2 = ['job', 'marital', 'education', 'housing', 'loan']
```

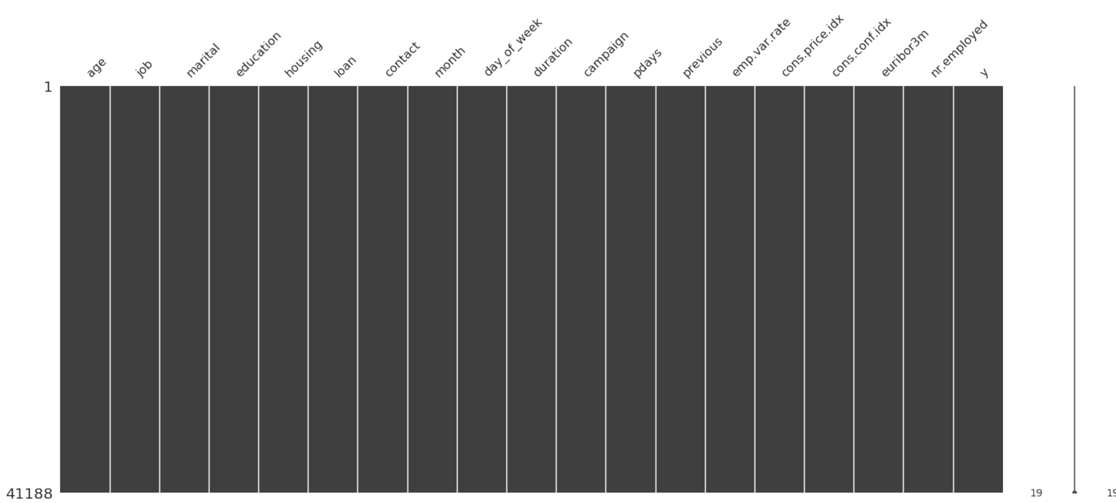
III Interpolation of categorical variables through empirical distributions

```
In [20]: dsdata2 = dsdata.copy()
         dsdata2 = dsdata2.drop(columns=['poutcome','default']) #too many missing values, and
         zero variance variable
         #dsdata2.shape
```

```
In [21]: #fill missing data at random from discrete distribution corresponding to histogram
         def dist_random_selection(col, num):
             arr = list(col.value_counts().index)
             prob = np.array(list(col.value_counts().values))
             p_norm = prob.sum()
             prob = prob/p_norm
             return np.random.choice(arr, num, replace=True, p=prob)

         def fill_missing(data,missing):
             for col in list(data[missing].columns):
                 count = len(data[col][data[col].isnull()])
                 data[col][data[col].isnull()] = dist_random_selection(data[col],count)
             return data
```

```
In [22]: dsdata3 = fill_missing(dsdata2, missing2)
```

```
In [23]: msno.matrix(dsdata3)
         plt.show()
```



IV Feature Selection

In [24]:
```
dsdata3.shape
```

Out[24]:
```
(41188, 19)
```

In [25]:
```
# Data preparation
dsn = dsdata3.copy()
dsn.describe(include=['O'])
# Drop missing value
#dsn2 = dsn.copy().dropna() HELL NO!!!!!!! XD,
# That drops about 10,000 columns...
```

Out[25]:
```
          job  marital          education housing   loan   contact  month  \
count   41188    41188              41188   41188  41188     41188  41188
unique     11        3                  7       2      2         2     10
top    admin.  married  university.degree     yes     no  cellular    may
freq    10499    24979              12664   22090  34778     26144  13769

       day_of_week      y
count        41188  41188
unique           5      2
top            thu     no
freq          8623  36548
```

In [26]:
```
# Create dummy
def make_dummies(dsn):
    numvar = ['age','campaign','pdays','previous','emp.var.rate','cons.price.idx','cons.
conf.idx','euribor3m','nr.employed']
    nonnumvar = ['job','marital','education','month','day_of_week'] #WE ONLY NEED THE
CATEGORICALS, DONT INCLUDE BINARIES!!

    for c,var in enumerate(nonnumvar):
        dummy = pd.get_dummies(dsn[var],drop_first=True)
        dsn = dsn.drop(columns=[var])
        dsn = pd.concat([dsn, dummy], axis=1)
        #print(var)
        #display(dummy.head(5))
    return dsn

def make_numeric(dsn):
    for c,var in enumerate(['housing','loan','contact']):
        dsn[var] = dsn[var].astype("category").cat.codes

    return dsn

dsn2 = make_dummies(dsn) #make dummies out of categoricals
dsn2 = make_numeric(dsn2) #make binaries out of yes/no
dsn2.info()
#dsn2.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 45 columns):
age                      41188 non-null int64
housing                  41188 non-null int8
loan                     41188 non-null int8
```

```
contact               41188 non-null int8
duration              41188 non-null int64
campaign              41188 non-null int64
pdays                 41188 non-null int64
previous              41188 non-null int64
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
blue-collar           41188 non-null uint8
entrepreneur          41188 non-null uint8
housemaid             41188 non-null uint8
management            41188 non-null uint8
retired               41188 non-null uint8
self-employed         41188 non-null uint8
services              41188 non-null uint8
student               41188 non-null uint8
technician            41188 non-null uint8
unemployed            41188 non-null uint8
married               41188 non-null uint8
single                41188 non-null uint8
basic.6y              41188 non-null uint8
basic.9y              41188 non-null uint8
high.school           41188 non-null uint8
illiterate            41188 non-null uint8
professional.course   41188 non-null uint8
university.degree     41188 non-null uint8
aug                   41188 non-null uint8
dec                   41188 non-null uint8
jul                   41188 non-null uint8
jun                   41188 non-null uint8
mar                   41188 non-null uint8
may                   41188 non-null uint8
nov                   41188 non-null uint8
oct                   41188 non-null uint8
sep                   41188 non-null uint8
mon                   41188 non-null uint8
thu                   41188 non-null uint8
tue                   41188 non-null uint8
wed                   41188 non-null uint8
dtypes: float64(5), int64(5), int8(3), object(1), uint8(31)
memory usage: 4.8+ MB
```

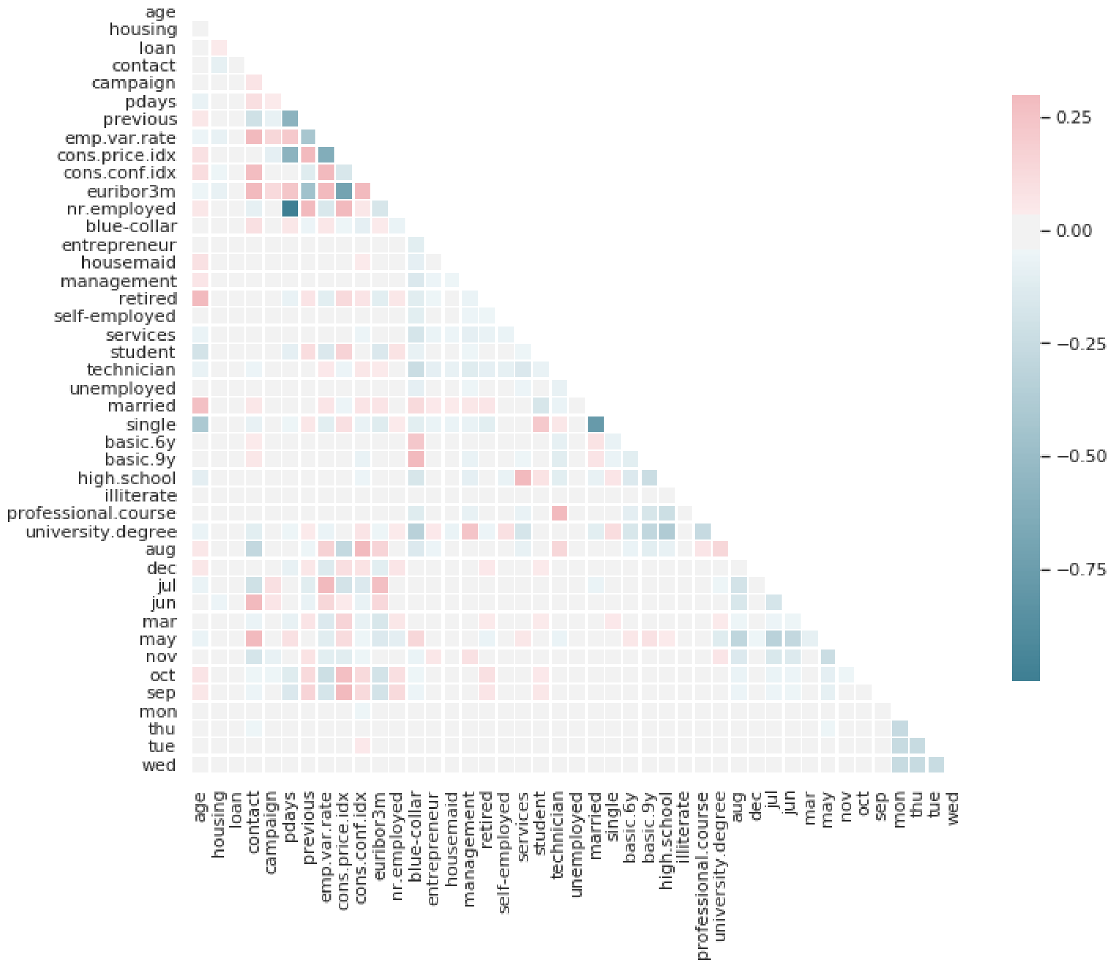
In [27]:
```
# Drop priori
dsn3 = dsn2.drop(columns=['duration']) #remove output and duration, which should not be
known a priori
```

In [28]:
```
# Data Normalization

# separate the data from the target attributes
X = dsn3.drop(columns=['y'])
#Y = pd.get_dummies(dsn3['y'],drop_first=True,dummy_na=True)

# normalize the data attributes
```

```
normalized_X = preprocessing.normalize(X)
```

In [29]:
```
# MAKE NORMALIZED DF
X_n = pd.DataFrame(normalized_X)
X_n.columns = X.columns
```

In [30]:
```
# Check the correlation

# NORMALIZE THE DATA !
corr = X_n.corr()
# Generate a mask for the upper triangle
sns.set(style="white")
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

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

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .76})
f.show()
```



IV Feature Selection

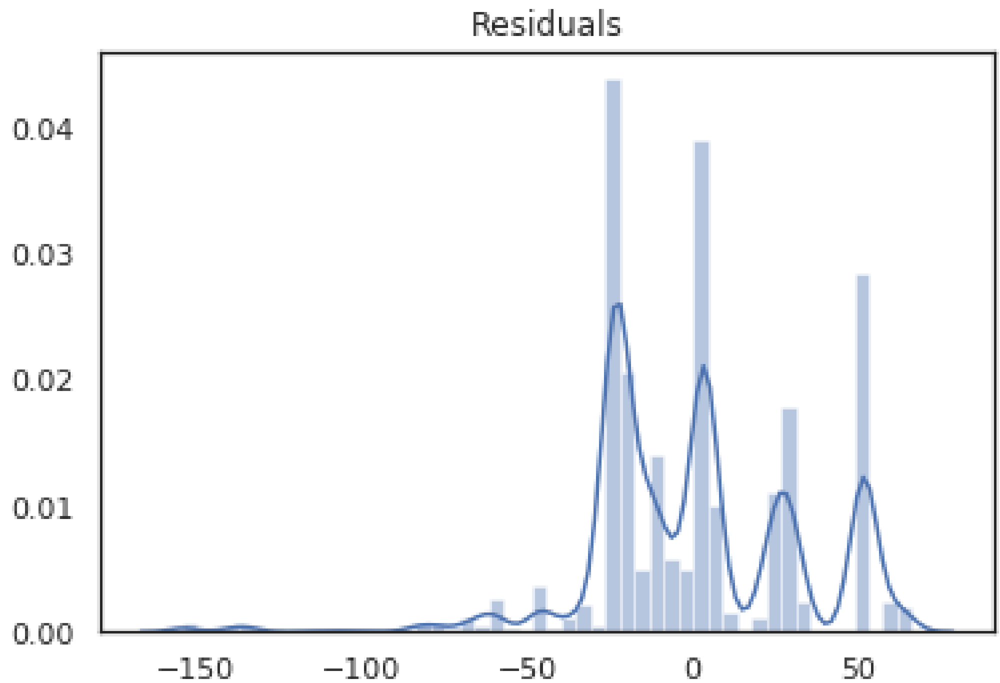
```
In [31]: def print_VIF(X):
             colnames = list(X.columns)
             for i in range(X.shape[1]):
                 print(oi.variance_inflation_factor(X.values, i), colnames[i])

         print("Variance Inflation Factors:")
         print_VIF(X)
```

```
Variance Inflation Factors:
25.681606550879618 age
2.1840846838967662 housing
1.1884269118600956 loan
5.169134506238248 contact
1.9403738005754756 campaign
45.31784765956443 pdays
2.0804529016210154 previous
93.69614375243889 emp.var.rate
62393.324630103554 cons.price.idx
```

```
389.9335927565134 cons.conf.idx
785.3183820323019 euribor3m
80118.64109558242 nr.employed
3.000196164065369 blue-collar
1.1794228557794595 entrepreneur
1.2074919210148327 housemaid
1.34132266030209 management
1.5435369807274977 retired
1.15945753421996 self-employed
1.5661286780219867 services
1.1946518543365574 student
2.0781095871717814 technician
1.1248546619346393 unemployed
6.550642053351103 married
4.03266944551772 single
1.6012107129188906 basic.6y
2.625822929245276 basic.9y
4.332027802671218 high.school
1.0051484048132502 illiterate
3.0152297601535163 professional.course
5.692430069185153 university.degree
8.13576981204706 aug
1.1411369408293892 dec
5.436567608186831 jul
3.782090835101747 jun
1.2544644335392852 mar
7.5886348027215975 may
3.932663973419781 nov
1.5992725497309754 oct
1.518952664229258 sep
2.0997274956465657 mon
2.119070550559964 thu
2.056185398273185 tue
2.0552461307481122 wed
```

1. intermediate regression on economic variables for VIF

```
In [32]: economic = ['emp.var.rate','cons.price.idx','cons.conf.idx','euribor3m']
         numeric = sm.OLS(X['nr.employed'].values,X[economic].values).fit()
         sns.distplot(numeric.resid)
         plt.title("Residuals")
         plt.show()
         display(numeric.summary())
```



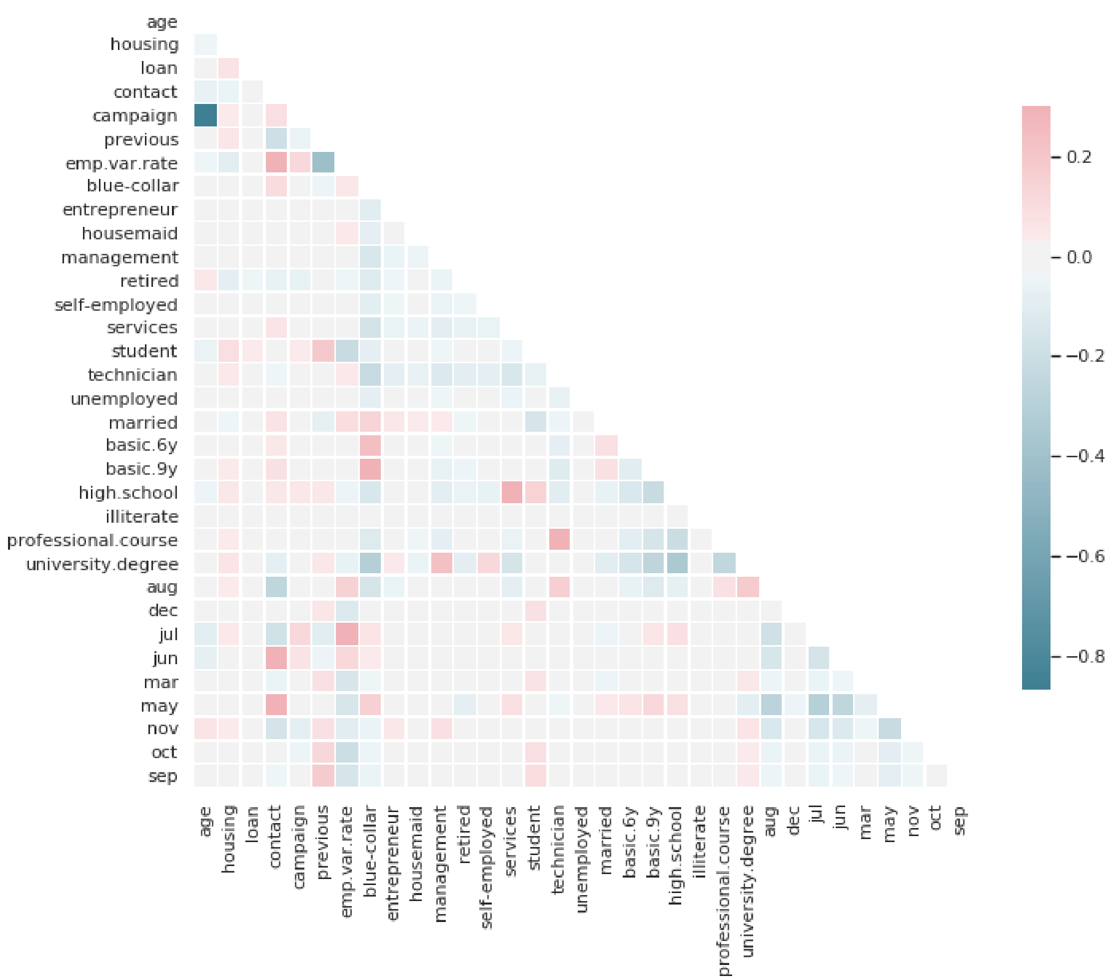

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 2.623e+08
Date:                Thu, 09 May 2019   Prob (F-statistic):               0.00
Time:                        22:03:38   Log-Likelihood:            -2.0167e+05
No. Observations:               41188   AIC:                         4.034e+05
Df Residuals:                   41184   BIC:                         4.034e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1           -63.3977      0.445   -142.626      0.000     -64.269     -62.526
x2            50.2830      0.027   1890.625      0.000      50.231      50.335
x3            -3.7858      0.038    -99.832      0.000      -3.860      -3.711
x4            86.5789      0.416    208.289      0.000      85.764      87.394
==============================================================================
Omnibus:                     3472.453   Durbin-Watson:                   0.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9896.607
Skew:                          -0.464   Prob(JB):                         0.00
Kurtosis:                       5.215   Cond. No.                         387.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

```
In [33]: dsn4 = X.copy()
         dsn4 = X.drop(columns=['nr.employed','euribor3m','cons.price.idx','cons.conf.idx'])

         print("Variance Inflation Factors:")
         print_VIF(dsn4)
```

```
Variance Inflation Factors:
18.80115670825449 age
2.1646083874537148 housing
1.1866246865988785 loan
3.7649497595290184 contact
1.9215194708841905 campaign
27.290000564335937 pdays
1.7336939576946881 previous
2.5126142325555105 emp.var.rate
2.8286185942884576 blue-collar
1.1730236609455233 entrepreneur
1.1917039329486732 housemaid
1.3381672712448212 management
1.5374238887447822 retired
1.1547841706846482 self-employed
1.5421419725346974 services
1.1675079710942462 student
2.046279789599359 technician
1.1168978674318881 unemployed
6.033846092902686 married
3.5392097508920575 single
1.5304324449169406 basic.6y
2.4107596377102323 basic.9y
3.76458134421943 high.school
1.0046407547199638 illiterate
2.7571835845860364 professional.course
4.864005877271467 university.degree
3.917570727343788 aug
1.0775324108601532 dec
4.4081745260225835 jul
3.3578106179599425 jun
1.1876113746312156 mar
6.3152035657393615 may
2.5110192350378955 nov
1.2740204607295855 oct
1.218884720090602 sep
2.0513989168836284 mon
2.0576022009794563 thu
2.013620242918694 tue
2.0105110975194673 wed
```

```
In [34]: dsn4 = dsn4.drop(columns=['pdays'])
         dsn4 = dsn4.drop(columns=['mon','thu','tue','wed'])
         dsn4 = dsn4.drop(columns=['single'])
         #dsn4 = dsn4.drop(columns=['campaign'])

         # NORMALIZE THE DATA !
         normalized_X = preprocessing.normalize(dsn4)

         X_n = pd.DataFrame(normalized_X)
```

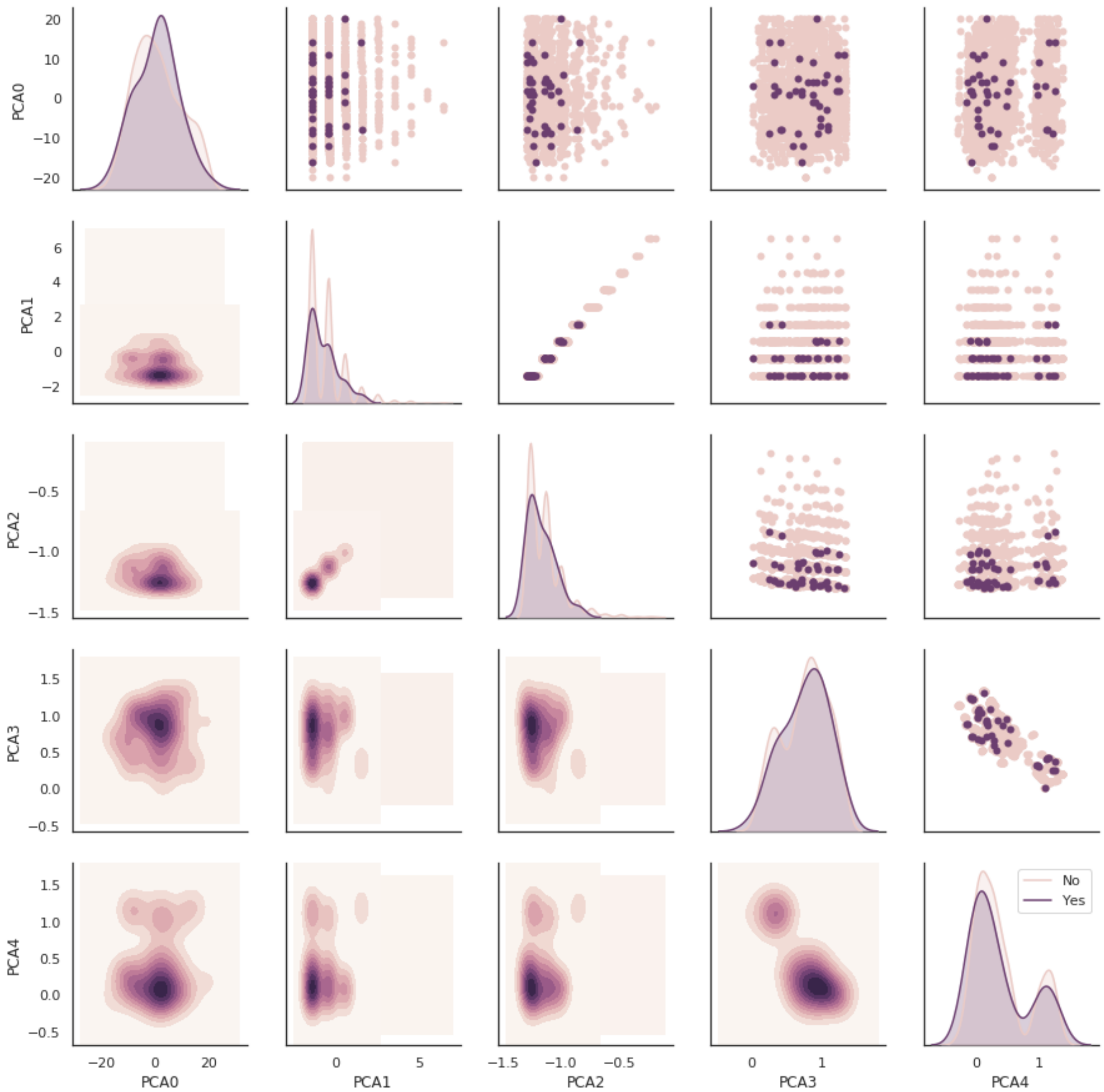
```
X_n.columns = dsn4.columns


#plt.figure(figsize=(12,8))
corr = X_n.corr()

# Generate a mask for the upper triangle
sns.set(style="white")
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

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

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .75})
plt.show()
```



IV Outlier Detection

1. PCA

```
In [35]: from sklearn.decomposition import PCA
         from sklearn.preprocessing import StandardScaler

In [ ]:

In [36]: X = dsn4#dsn4.drop(columns=['y'])

         #sns.pairplot(X)
         # Plot the data
         #fig = plt.figure(figsize=(12,8))
         #with plt.style.context(('ggplot')):
         #    plt.plot( X.T)
         #    plt.show()

In [ ]:

In [37]: pcaA = PCA()
         pcaX = pcaA.fit_transform(X)#.fit_transform(StandardScaler().fit_transform(X))
         # PCA & score
         print(pcaA.explained_variance_ratio_[:5])

[0.89099519 0.06367793 0.02046394 0.00326992 0.00232495]


In [38]: # Compute the euclidean distance ( 3 PC )
         euclidean = np.zeros(X.shape[0])
         for i in range(3):
             euclidean += (pcaX[:,i] - np.mean(pcaX[:,:3]))**2/np.var(pcaX[:,:3])

         #colors = [plt.cm.jet(float(i)/max(euclidean)) for i in euclidean]

In [39]: X_pca = pd.DataFrame(pcaX, columns=['PCA%i' % i for i in range(X.shape[1])],
         index=X.index)

In [40]: colors = ["blue", "blue"]
         pal = sns.xkcd_palette(colors)

         pal = [sns.cubehelix_palette(light=1)[1], sns.cubehelix_palette(light=1)[4]]

In [41]: sns.set_style('white')
         cmap = sns.cubehelix_palette(light=1, as_cmap=True)


         df = X_pca.copy()
         df['y']  = dsdata['y']
         df = df[['PCA0','PCA1','PCA2','PCA3','PCA4','y']]
         df = df.iloc[:2000]
         g = sns.PairGrid(df, diag_sharey=False, hue='y', palette=pal)
         g.map_lower(sns.kdeplot, cmap=cmap, shade=True)
         g.map_upper(sns.scatterplot, linewidth=0)
         g.map_diag(sns.kdeplot, shade=True)
         plt.legend(['No','Yes'])
         plt.show()
```

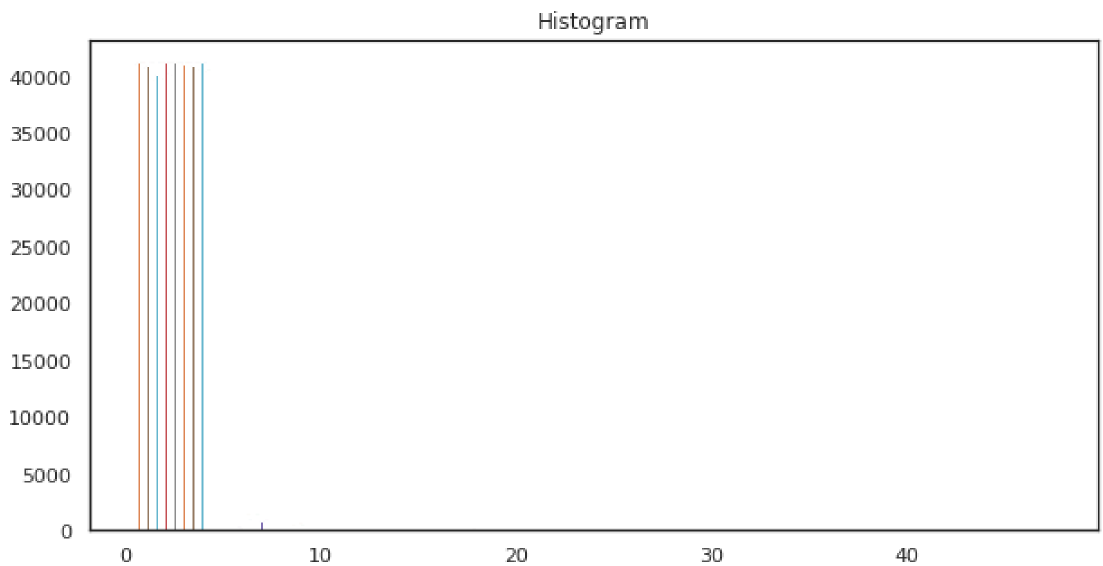


In [42]:
```
'''
colors = [plt.cm.jet(float(i)/max(euclidean)) for i in euclidean]
fig = plt.figure(figsize=(8,6))
with plt.style.context(('ggplot')):
    plt.scatter(pcaX[:, 0], pcaX[:, 1], c=colors, edgecolors='k')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title('Score Plot')
plt.show()
'''
a = 0
```

2. Z-score

In [43]:
```
#NORMALIZATION????
```

```
In [44]: from scipy import stats

         dsn5 = dsn4.copy()

         zX = np.abs(stats.zscore(X))

In [45]: X.shape

Out[45]: (41188, 33)

In [46]: def histogram(variable):
             plt.figure(figsize=(10, 5))
             plt.title("Histogram")
             ax = plt.hist(zX)

         histogram(zX)
```




```
In [47]: dsn5 = dsn5[(zX < 10).all(axis=1)]
         dsn5 = dsn5.drop(columns = ['dec','illiterate'])
         dsn5.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 40949 entries, 0 to 41187
Data columns (total 31 columns):
age                       40949 non-null int64
housing                   40949 non-null int8
loan                      40949 non-null int8
contact                   40949 non-null int8
campaign                  40949 non-null int64
previous                  40949 non-null int64
emp.var.rate              40949 non-null float64
blue-collar               40949 non-null uint8
entrepreneur              40949 non-null uint8
housemaid                 40949 non-null uint8
management                40949 non-null uint8
retired                   40949 non-null uint8
```

```
self-employed          40949 non-null uint8
services               40949 non-null uint8
student                40949 non-null uint8
technician             40949 non-null uint8
unemployed             40949 non-null uint8
married                40949 non-null uint8
basic.6y               40949 non-null uint8
basic.9y               40949 non-null uint8
high.school            40949 non-null uint8
professional.course    40949 non-null uint8
university.degree      40949 non-null uint8
aug                    40949 non-null uint8
jul                    40949 non-null uint8
jun                    40949 non-null uint8
mar                    40949 non-null uint8
may                    40949 non-null uint8
nov                    40949 non-null uint8
oct                    40949 non-null uint8
sep                    40949 non-null uint8
dtypes: float64(1), int64(3), int8(3), uint8(24)
memory usage: 2.6 MB
```

```
In [48]: X = dsn5#.drop(columns=['y'])
         Y = dsdata['y'][X.index].astype("category").cat.codes
         Y.value_counts()
```

```
Out[48]: 0    36405
         1     4544
         dtype: int64
```

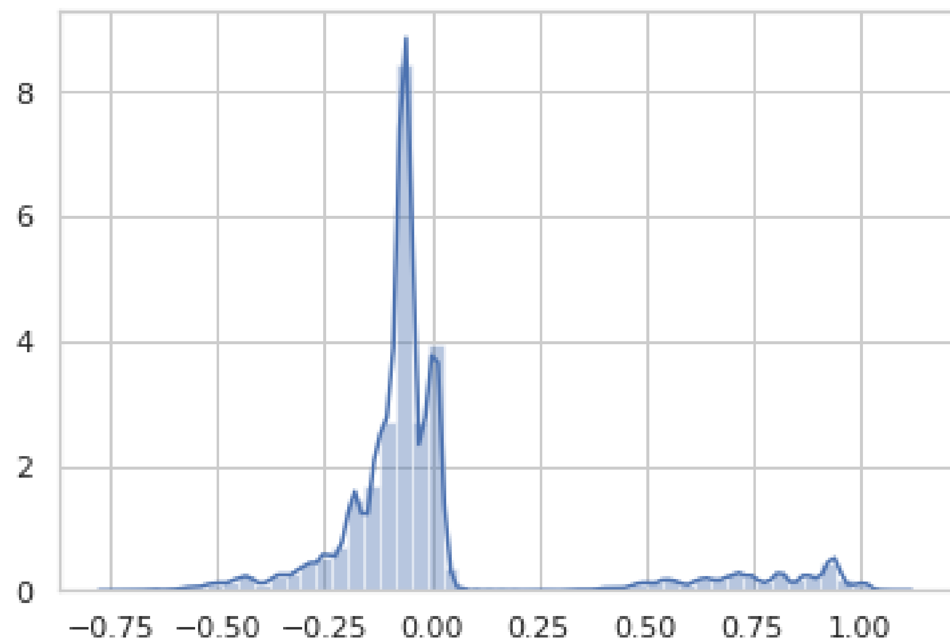
V Prediction

```
In [49]: sns.set_style('whitegrid')
```

```
In [ ]:
```

```
In [50]: X_ = sm.add_constant(X)
         model = sm.OLS(Y,X_).fit()

         print("Distribution of OLS residuals")
         sns.distplot(model.resid)
         plt.show()
         display(model.summary())
```

```
Distribution of OLS residuals
```



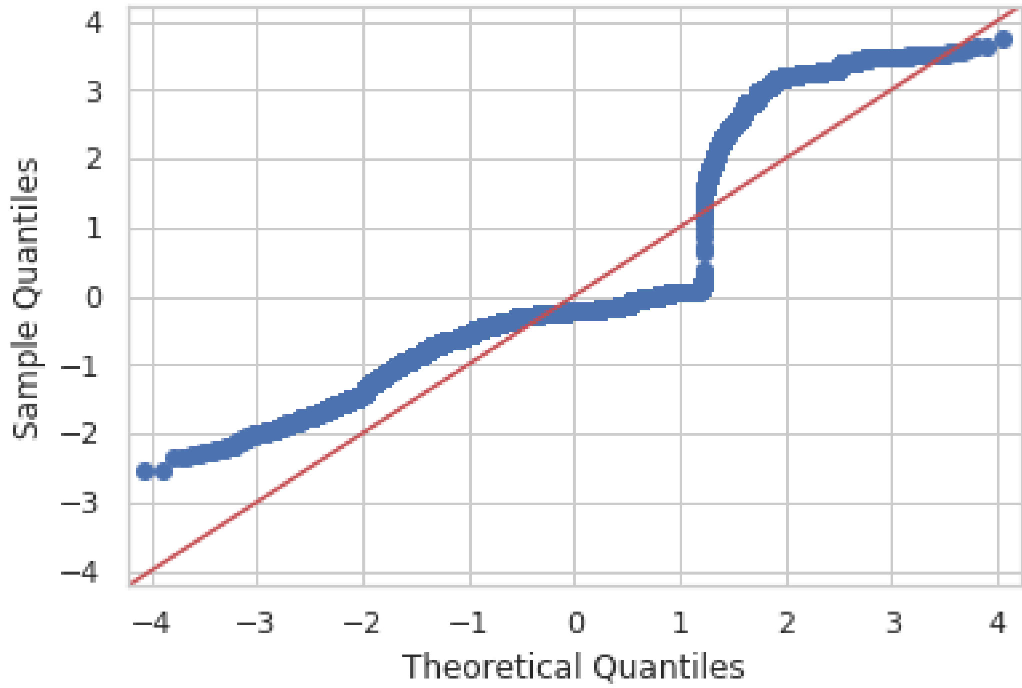

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.144
Model:                            OLS   Adj. R-squared:                  0.144
Method:                 Least Squares   F-statistic:                     222.7
Date:                Thu, 09 May 2019   Prob (F-statistic):               0.00
Time:                        22:03:58   Log-Likelihood:                -7490.6
No. Observations:               40949   AIC:                         1.505e+04
Df Residuals:                   40917   BIC:                         1.532e+04
Df Model:                          31
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                   0.0882      0.012      7.607      0.000       0.065       0.111
age                     0.0003      0.000      1.596      0.111   -6.1e-05       0.001
housing                -0.0040      0.003     -1.378      0.168      -0.010       0.002
loan                   -0.0021      0.004     -0.524      0.600      -0.010       0.006
contact                 0.0288      0.005      6.189      0.000       0.020       0.038
campaign               -0.0035      0.001     -6.270      0.000      -0.005      -0.002
previous                0.0686      0.003     20.841      0.000       0.062       0.075
emp.var.rate           -0.0529      0.001    -36.040      0.000      -0.056      -0.050
blue-collar            -0.0219      0.005     -4.195      0.000      -0.032      -0.012
entrepreneur           -0.0187      0.008     -2.266      0.023      -0.035      -0.003
housemaid              -0.0069      0.010     -0.704      0.481      -0.026       0.012
management             -0.0115      0.006     -1.846      0.065      -0.024       0.001
retired                 0.0458      0.009      5.258      0.000       0.029       0.063
self-employed          -0.0172      0.008     -2.074      0.038      -0.033      -0.001
services               -0.0192      0.006     -3.335      0.001      -0.030      -0.008
```

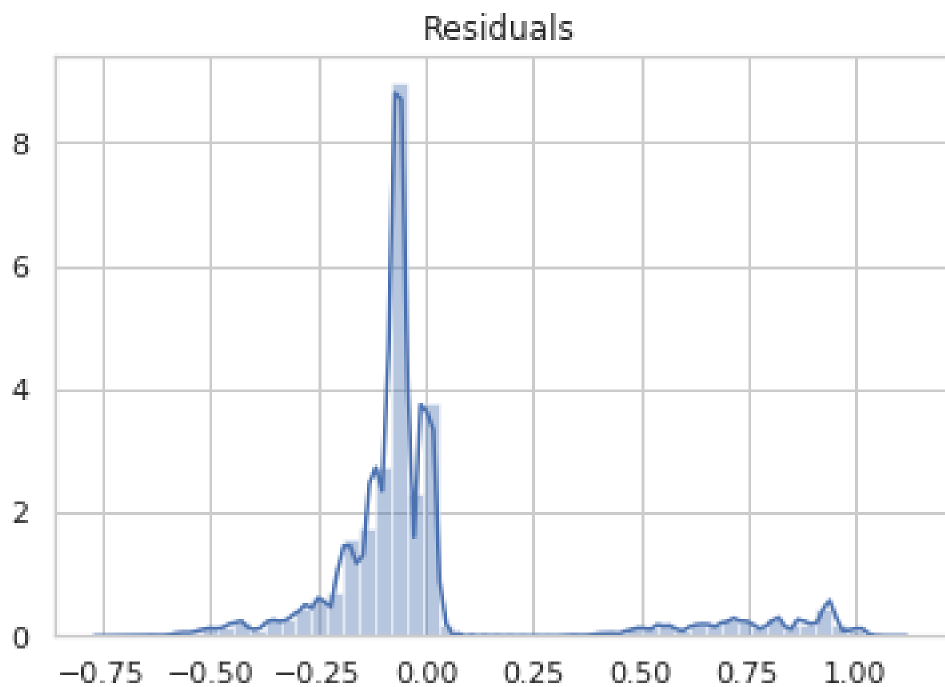
```
student                  0.0758      0.011      7.054      0.000       0.055       0.097
technician              -0.0079      0.005     -1.543      0.123      -0.018       0.002
unemployed               0.0019      0.010      0.195      0.846      -0.017       0.021
married                 -0.0035      0.003     -1.097      0.273      -0.010       0.003
basic.6y                 0.0021      0.008      0.280      0.780      -0.013       0.017
basic.9y                -0.0062      0.006     -1.041      0.298      -0.018       0.005
high.school             -0.0021      0.006     -0.353      0.724      -0.014       0.010
professional.course      0.0037      0.007      0.542      0.588      -0.010       0.017
university.degree        0.0095      0.006      1.530      0.126      -0.003       0.022
aug                      0.0497      0.008      6.399      0.000       0.034       0.065
jul                      0.0676      0.008      8.744      0.000       0.052       0.083
jun                      0.0343      0.008      4.469      0.000       0.019       0.049
mar                      0.2780      0.014     20.268      0.000       0.251       0.305
may                     -0.0496      0.007     -7.544      0.000      -0.062      -0.037
nov                     -0.0274      0.008     -3.635      0.000      -0.042      -0.013
oct                      0.1578      0.012     12.741      0.000       0.134       0.182
sep                      0.1712      0.014     12.607      0.000       0.145       0.198
==============================================================================
Omnibus:                    16009.804   Durbin-Watson:                   1.812
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            54083.506
Skew:                           2.043   Prob(JB):                         0.00
Kurtosis:                       6.873   Cond. No.                         570.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

In [51]:
```python
alpha = 0.05
a = model.pvalues < alpha

X2 = X_[X_.columns[a]]
X2 = sm.add_constant(X2)
print("Not Statistically significant regressors are:")
print(list(X_.columns[~a]))
```

```
Not Statistically significant regressors are:
['age', 'housing', 'loan', 'housemaid', 'management', 'technician', 'unemployed',
'married', 'basic.6y', 'basic.9y', 'high.school', 'professional.course',
'university.degree']
```

In [52]:
```python
model2 = sm.OLS(Y,X2).fit(cov_type='HC0')
sm.qqplot(model2.resid, sc.norm, fit=True, line='45')
plt.show()
sns.distplot(model2.resid)
plt.title('Residuals')
plt.show()
display(model2.summary())
print("Variance Inflation Factors:")
print_VIF(X2)
```






```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:                      y   R-squared:                       0.144
Model:                            OLS   Adj. R-squared:                  0.144
Method:                 Least Squares   F-statistic:                     188.6
Date:                Thu, 09 May 2019   Prob (F-statistic):               0.00
Time:                        22:03:58   Log-Likelihood:                -7501.5
No. Observations:               40949   AIC:                         1.504e+04
Df Residuals:                   40930   BIC:                         1.520e+04
Df Model:                          18
Covariance Type:                  HC0
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const              0.0937      0.009     10.452      0.000       0.076       0.111
contact            0.0294      0.006      5.288      0.000       0.018       0.040
campaign          -0.0035      0.000     -8.882      0.000      -0.004      -0.003
previous           0.0687      0.005     13.463      0.000       0.059       0.079
emp.var.rate      -0.0531      0.002    -25.670      0.000      -0.057      -0.049
blue-collar       -0.0238      0.003     -7.158      0.000      -0.030      -0.017
entrepreneur      -0.0151      0.007     -2.036      0.042      -0.030      -0.001
retired            0.0523      0.010      5.298      0.000       0.033       0.072
self-employed     -0.0127      0.008     -1.619      0.105      -0.028       0.003
services          -0.0204      0.005     -4.375      0.000      -0.030      -0.011
student            0.0746      0.015      4.945      0.000       0.045       0.104
aug                0.0515      0.011      4.811      0.000       0.031       0.073
jul                0.0678      0.010      6.512      0.000       0.047       0.088
jun                0.0344      0.010      3.419      0.001       0.015       0.054
mar                0.2794      0.023     12.311      0.000       0.235       0.324
may               -0.0504      0.009     -5.820      0.000      -0.067      -0.033
nov               -0.0271      0.010     -2.835      0.005      -0.046      -0.008
oct                0.1586      0.021      7.676      0.000       0.118       0.199
sep                0.1726      0.023      7.629      0.000       0.128       0.217
==============================================================================
Omnibus:                    16019.833   Durbin-Watson:                   1.810
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            54130.320
Skew:                           2.045   Prob(JB):                         0.00
Kurtosis:                       6.873   Cond. No.                         50.7
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
"""


Variance Inflation Factors:
20.972372500572607 const
2.4234688619455382 contact
1.0356172703733448 campaign
1.246284573034863 previous
2.5257756330545083 emp.var.rate
1.151683246873376 blue-collar
1.0371156669402068 entrepreneur
1.0559290697772752 retired
1.02854766753986 self-employed
1.0847262596966627 services
1.044883382599428 student
3.714266612071874 aug
4.159414719803091 jul
```

```
3.206044394229739 jun
1.1977713020110123 mar
4.659805935987449 may
2.4666988074595966 nov
1.2795283021381727 oct
1.2252868135788397 sep
```

```
In [53]: X_train, X_test, Y_train, Y_test = train_test_split(X2.values, Y.values, test_size=0.25)

         reg = LinearRegression()
         reg = reg.fit(X_train,Y_train)

         y_hat = reg.predict(X_test)

         print('Test accuracy:',np.round(reg.score(np.round(X_test), Y_test),3), ', MSE Loss
         is:', mean_squared_error(Y_test,y_hat))
```

```
Test accuracy: 0.151 , MSE Loss is: 0.0852462833821851
```

```
In [ ]:
```

VI Classification

```
In [54]: def plot_hist2(df,df2,df_col):
             df = df.dropna()
             df2 = df2.dropna()
             for d in df_col:
                 print("Empirical Distribution of Variable "+d)
                 fig, axes = plt.subplots(1,2,figsize=(15,9))
                 sns.distplot(df[d],ax=axes[0])
                 sns.distplot(df2[d],ax=axes[1])
                 axes[0].set_ylabel("Probability")
                 axes[1].set_ylabel("Probability")
                 plt.suptitle("Empirical Probability Distribution of Numerical Variable "+d)
                 plt.show()

         def get_num_cols(df):
             idx = df.select_dtypes(exclude='object').columns.values
             dF = df[idx].dropna() #remove NaNs or else it cant plot
             return dF.columns

         def calculate_metrics(y_test,y_hat):
             c = confusion_matrix(y_test, y_hat)
             print("Confusion matrix is:")
             print(c)
             print("We have",c[0][0]+c[1][1],"correct observations and",c[0][1]+c[1][0],
         "misclassifications.")
             print(classification_report(y_test, y_hat))
             plt.figure(figsize=(6,6))
             sns.heatmap(c,cmap="YlGnBu",annot=True,fmt='g')
             plt.show()

         def plot_ROC(y_test,X_test,classifier):
```

```
    roc = roc_auc_score(y_test, classifier.predict(X_test))
    fpr, tpr, _ = roc_curve(y_test, classifier.predict_proba(X_test)[:,1])
    plt.figure(figsize=(12,12))
    plt.plot(fpr, tpr, label='Classifier area ='+str(np.round(roc,2)))
    plt.plot([0, 1], [0, 1],'--')

    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.title('ROC curve')
    plt.legend()

    plt.xlim([-0.01, 1.0])
    plt.ylim([0.0, 1.01])
    plt.show()


def plot_ROC2(L_Y,L_X,L_YHAT,L_PROB,L_NAME):

    plt.figure(figsize=(12,12))
    for i in range(len(L_Y)):
        roc = roc_auc_score(L_Y[i], L_YHAT[i])
        fpr, tpr, _ = roc_curve(L_Y[i], L_PROB[i][:,1])
        plt.plot(fpr, tpr, label=L_NAME[i]+', area ='+str(np.round(roc,2)))
    plt.plot([0, 1], [0, 1],'--')

    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.title('ROC curve')
    plt.legend()

    plt.xlim([-0.01, 1.0])
    plt.ylim([0.0, 1.01])
    plt.show()



def MAE(y_test,y_hat):
    return np.abs(y_test-y_hat).sum()#/y_test.shape[0]
```

VI Logistic Regression with PCA

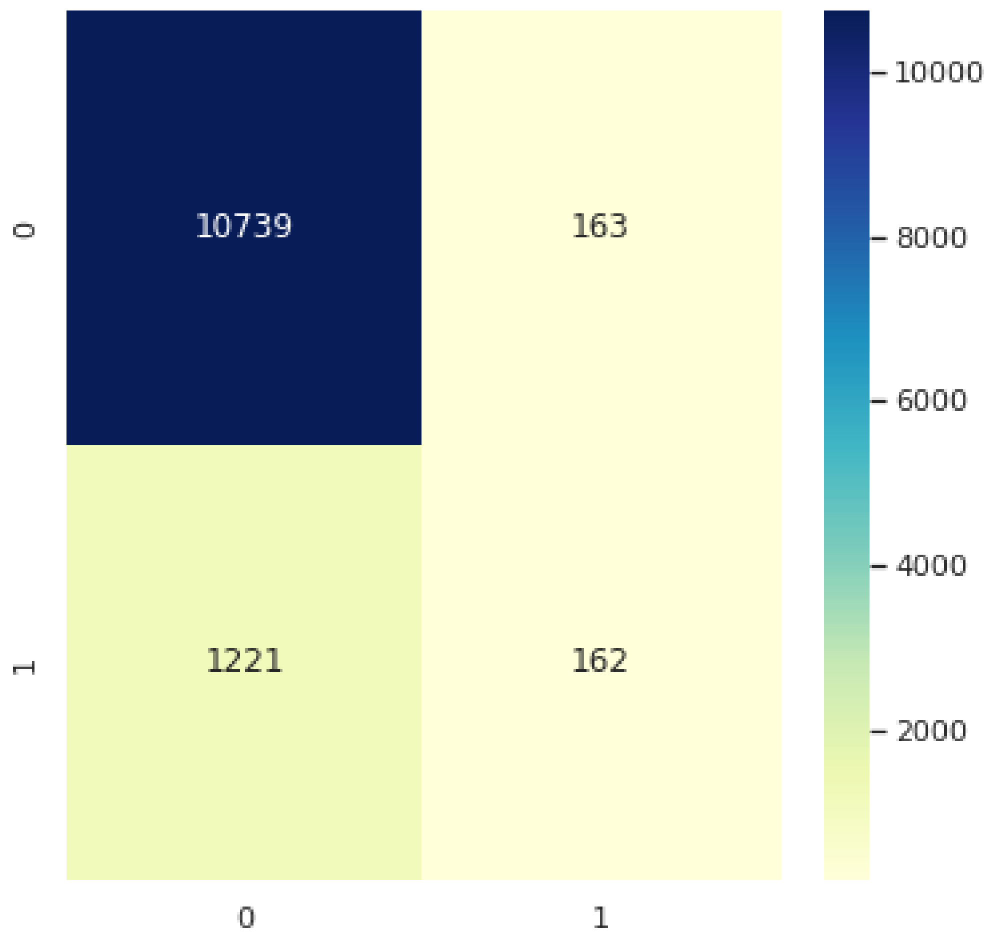
```
In [55]: pcaXmodel = PCA(n_components=5, whiten=True)
         pc = pcaXmodel.fit_transform(X)#.fit_transform(StandardScaler().fit_transform(X))
         print(pcaXmodel.explained_variance_ratio_[:5])

         proj = pcaXmodel.inverse_transform(pc)
         a = pd.DataFrame(proj)[list((np.ones((31,1))-1).cumsum().ravel())]

         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
         random_state=0)
         lr = LogisticRegression().fit(X_train,y_train)

         y_hat = lr.predict(X_test)
         print('Test accuracy:',np.round(lr.score(X_test, y_test),3), ', Cross Entropy Loss is:',
         log_loss(y_test,y_hat))
```

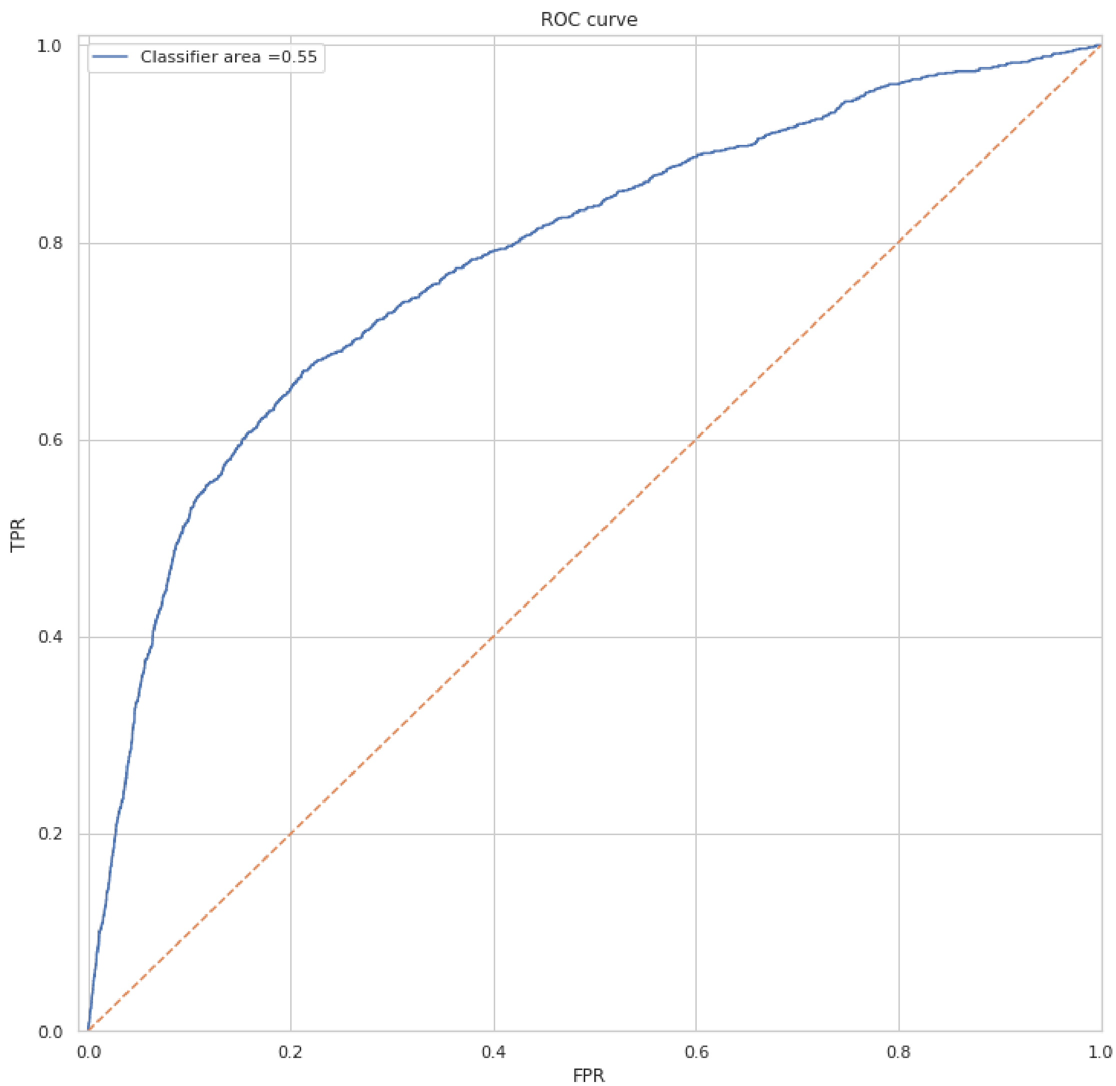
```
print(MAE(y_test,y_hat))

#pcaXmodel =
PCA(n_components=5).fit_transform(X)#.fit_transform(StandardScaler().fit_transform(X))
```

```
[0.89682146 0.05748207 0.02049585 0.00333144 0.00236537]
Test accuracy: 0.887 , Cross Entropy Loss is: 3.89107015587607
1384
```

```
In [56]: calculate_metrics(y_test,y_hat)
         plot_ROC(y_test,X_test,lr)
```

```
Confusion matrix is:
[[10739   163]
 [ 1221   162]]
We have 10901 correct observations and 1384 misclassifications.
              precision    recall  f1-score   support

           0       0.90      0.99      0.94     10902
           1       0.50      0.12      0.19      1383

   micro avg       0.89      0.89      0.89     12285
   macro avg       0.70      0.55      0.56     12285
weighted avg       0.85      0.89      0.86     12285
```


10739
163
1221
162
0
1
0
1
10000
8000
6000
4000
2000




```
In [57]: a.shape

Out[57]: (40949, 31)

In [58]: logit = sm.Logit(Y, X).fit()
         display(logit.summary())

Optimization terminated successfully.
         Current function value: 0.293629
         Iterations 7


<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                40949
Model:                          Logit   Df Residuals:                    40918
Method:                           MLE   Df Model:                           30
```

```
Date:                Thu, 09 May 2019   Pseudo R-squ.:                  0.1575
Time:                        22:03:59   Log-Likelihood:                -12024.
converged:                       True   LL-Null:                       -14272.
                                        LLR p-value:                     0.000
=========================================================================================
                          coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------
age                    -0.0212      0.001    -14.239      0.000      -0.024      -0.018
housing                -0.1281      0.034     -3.801      0.000      -0.194      -0.062
loan                   -0.0677      0.048     -1.425      0.154      -0.161       0.025
contact                -0.2701      0.051     -5.250      0.000      -0.371      -0.169
campaign               -0.0806      0.010     -8.281      0.000      -0.100      -0.061
previous                0.3963      0.026     15.057      0.000       0.345       0.448
emp.var.rate           -0.4202      0.014    -31.114      0.000      -0.447      -0.394
blue-collar            -0.6748      0.060    -11.169      0.000      -0.793      -0.556
entrepreneur           -0.3512      0.104     -3.379      0.001      -0.555      -0.148
housemaid              -0.3760      0.121     -3.114      0.002      -0.613      -0.139
management             -0.1945      0.072     -2.704      0.007      -0.336      -0.054
retired                 0.4768      0.089      5.338      0.000       0.302       0.652
self-employed          -0.3755      0.100     -3.763      0.000      -0.571      -0.180
services               -0.4582      0.071     -6.478      0.000      -0.597      -0.320
student                -0.0736      0.090     -0.822      0.411      -0.249       0.102
technician             -0.2791      0.058     -4.804      0.000      -0.393      -0.165
unemployed             -0.1971      0.106     -1.864      0.062      -0.404       0.010
married                -0.0664      0.037     -1.790      0.073      -0.139       0.006
basic.6y               -0.4553      0.091     -5.022      0.000      -0.633      -0.278
basic.9y               -0.6082      0.067     -9.122      0.000      -0.739      -0.478
high.school            -0.7014      0.058    -11.991      0.000      -0.816      -0.587
professional.course    -0.5467      0.071     -7.689      0.000      -0.686      -0.407
university.degree      -0.6419      0.056    -11.420      0.000      -0.752      -0.532
aug                    -0.2133      0.065     -3.273      0.001      -0.341      -0.086
jul                     0.0528      0.066      0.801      0.423      -0.076       0.182
jun                    -0.0471      0.069     -0.687      0.492      -0.181       0.087
mar                     0.9616      0.098      9.775      0.000       0.769       1.154
may                    -0.9755      0.056    -17.501      0.000      -1.085      -0.866
nov                    -0.6573      0.070     -9.441      0.000      -0.794      -0.521
oct                     0.2938      0.093      3.173      0.002       0.112       0.475
sep                     0.4102      0.100      4.114      0.000       0.215       0.606
=========================================================================================
"""
```

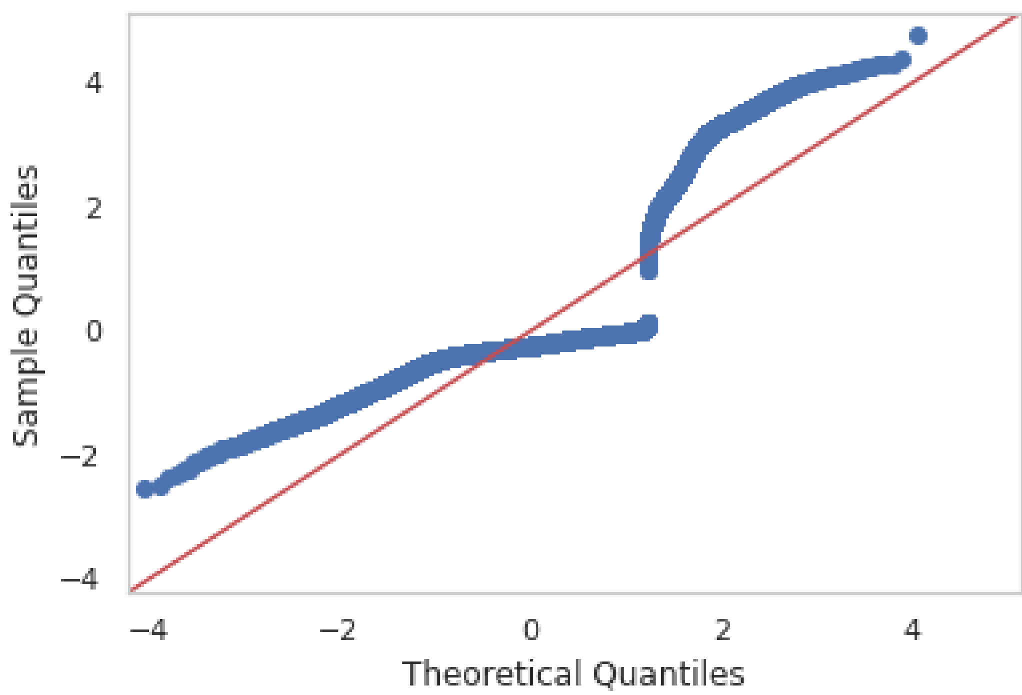
```
In [59]: sns.set_style('whitegrid')
         alpha = 0.05
         a = logit.pvalues < alpha

         X3 = X[X.columns[a]]
         print("Not Statistically significant regressors are:")
         print(list(X.columns[~a]))
```

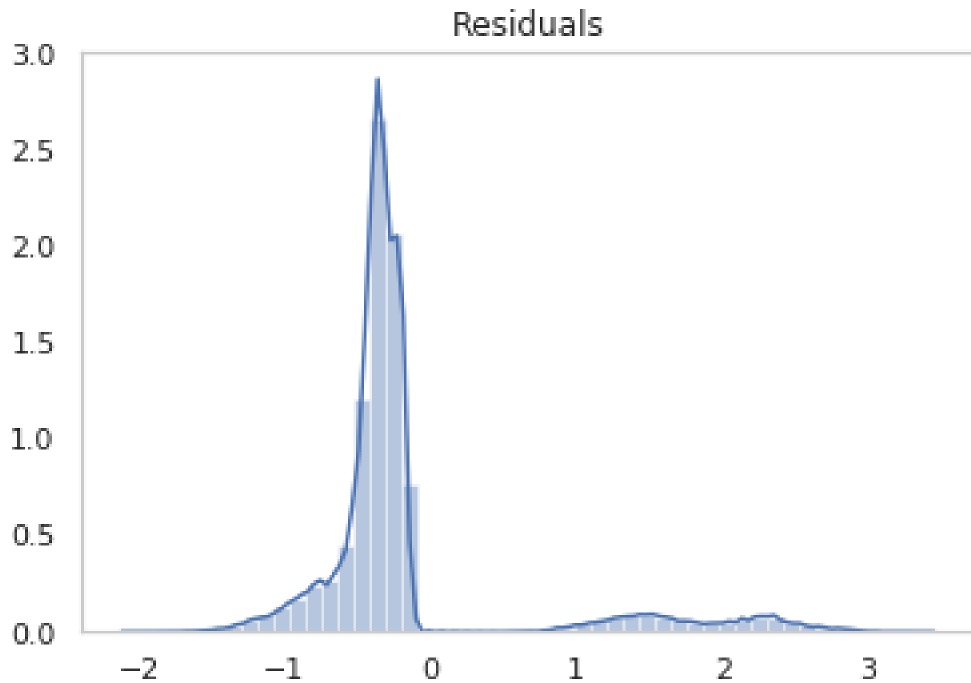
```
Not Statistically significant regressors are:
['loan', 'student', 'unemployed', 'married', 'jul', 'jun']
```

```
In [60]: model2 = sm.Logit(Y.values,X3.values).fit()
         sm.qqplot(model2.resid_dev, sc.norm, fit=True, line='45')
         plt.grid()
```

```
    plt.show()
    sns.distplot(model2.resid_dev)
    plt.title('Residuals')
    plt.grid()
    plt.show()
    display(model2.summary())
    print("Variance Inflation Factors:")
    print_VIF(X3)
```

```
Optimization terminated successfully.
         Current function value: 0.293766
         Iterations 7
```






```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                40949
Model:                          Logit   Df Residuals:                    40924
Method:                           MLE   Df Model:                           24
Date:                Thu, 09 May 2019   Pseudo R-squ.:                  0.1571
Time:                        22:04:00   Log-Likelihood:                -12029.
converged:                       True   LL-Null:                       -14272.
                                        LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.0224      0.001    -17.006      0.000      -0.025      -0.020
x2            -0.1317      0.034     -3.926      0.000      -0.198      -0.066
x3            -0.3043      0.047     -6.499      0.000      -0.396      -0.213
x4            -0.0808      0.010     -8.376      0.000      -0.100      -0.062
x5             0.3961      0.026     15.098      0.000       0.345       0.448
x6            -0.4128      0.012    -34.251      0.000      -0.436      -0.389
x7            -0.6594      0.058    -11.308      0.000      -0.774      -0.545
x8            -0.3403      0.103     -3.303      0.001      -0.542      -0.138
x9            -0.3535      0.120     -2.955      0.003      -0.588      -0.119
x10           -0.1865      0.071     -2.629      0.009      -0.326      -0.047
x11            0.5209      0.088      5.942      0.000       0.349       0.693
x12           -0.3589      0.099     -3.621      0.000      -0.553      -0.165
x13           -0.4394      0.069     -6.325      0.000      -0.576      -0.303
x14           -0.2568      0.057     -4.536      0.000      -0.368      -0.146
x15           -0.4666      0.089     -5.213      0.000      -0.642      -0.291
x16           -0.6203      0.065     -9.572      0.000      -0.747      -0.493
x17           -0.6994      0.055    -12.654      0.000      -0.808      -0.591
```

```
x18          -0.5535      0.069     -7.983      0.000      -0.689      -0.418
x19          -0.6333      0.053    -11.914      0.000      -0.737      -0.529
x20          -0.2323      0.054     -4.337      0.000      -0.337      -0.127
x21           0.9632      0.093     10.333      0.000       0.781       1.146
x22          -0.9765      0.045    -21.668      0.000      -1.065      -0.888
x23          -0.6694      0.061    -11.011      0.000      -0.789      -0.550
x24           0.2970      0.088      3.385      0.001       0.125       0.469
x25           0.4036      0.095      4.265      0.000       0.218       0.589
==============================================================================
"""


Variance Inflation Factors:
10.595228987761248 age
2.1162627291809826 housing
2.5787549377784673 contact
1.9650742218414552 campaign
1.3878420674037453 previous
1.8225837030888707 emp.var.rate
2.3498218720655597 blue-collar
1.141514821122065 entrepreneur
1.1505968256965058 housemaid
1.2955824062114378 management
1.4853298133639519 retired
1.1259569038839308 self-employed
1.4472162108565212 services
1.9011737412545888 technician
1.3973642634299883 basic.6y
2.0054267834052713 basic.9y
2.674200337054219 high.school
2.309779602262992 professional.course
3.3033756849895126 university.degree
1.639060932630886 aug
1.0728776210604138 mar
2.208910038097664 may
1.3446108077360526 nov
1.1415496233265712 oct
1.1068748593204247 sep
```

In [ ]:

In [61]:
```python
logit = sm.Logit(Y.values, pc).fit()
display(logit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.673033
         Iterations 4


<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                40949
Model:                          Logit   Df Residuals:                    40944
Method:                           MLE   Df Model:                            4
```

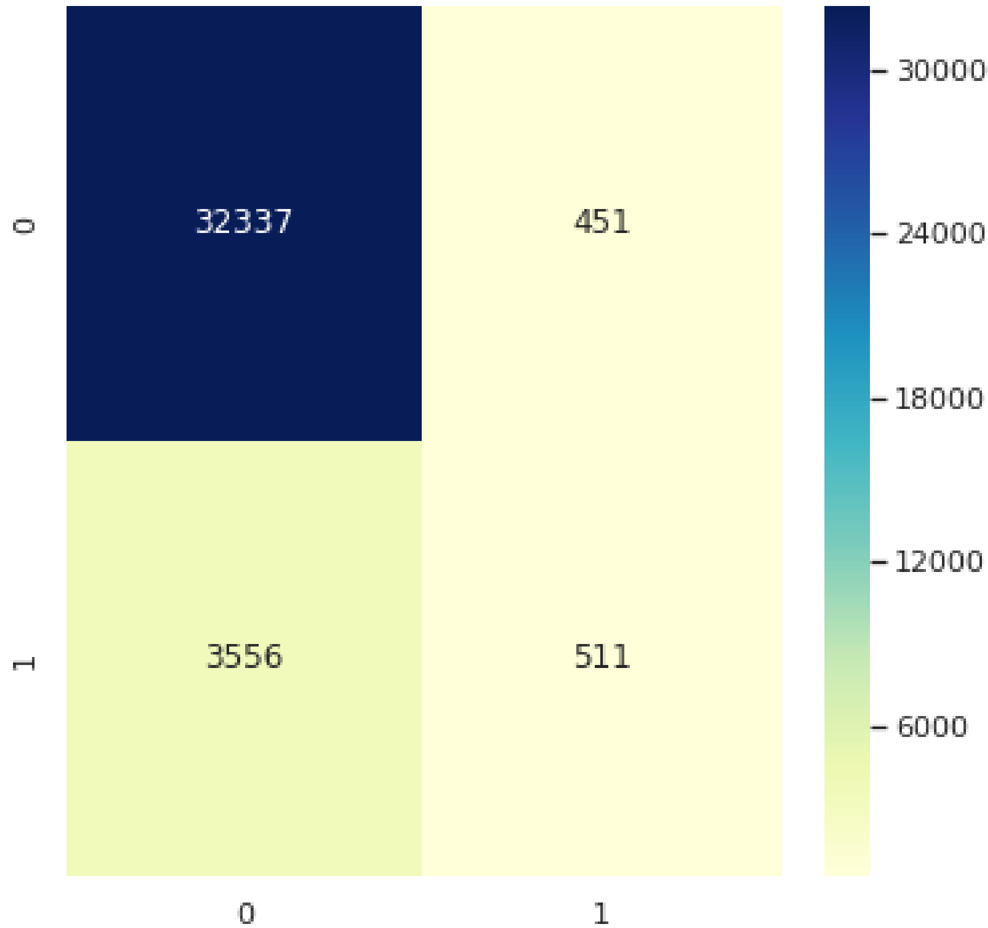
```
Date:                Thu, 09 May 2019   Pseudo R-squ.:                 -0.9310
Time:                        22:04:01   Log-Likelihood:                -27560.
converged:                       True   LL-Null:                       -14272.
                                        LLR p-value:                     1.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.0352      0.010      3.448      0.001       0.015       0.055
x2            -0.1188      0.010    -11.713      0.000      -0.139      -0.099
x3             0.3641      0.010     35.553      0.000       0.344       0.384
x4            -0.1405      0.010    -13.936      0.000      -0.160      -0.121
x5            -0.0247      0.010     -2.458      0.014      -0.044      -0.005
==============================================================================
"""
```

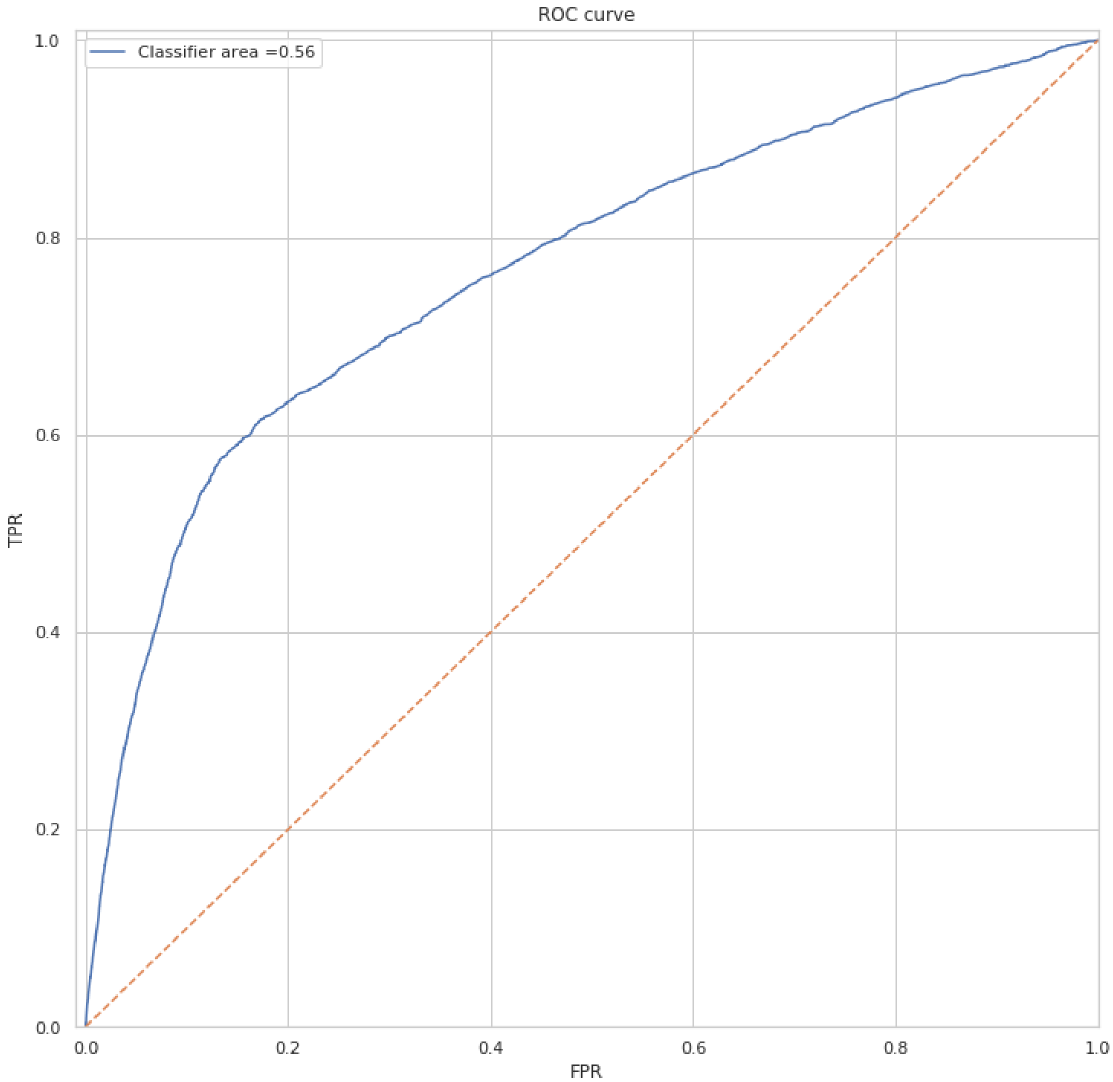
```
In [ ]:
```

1. AdaBoost Classifier

```
In [62]: X_train, X_test, Y_train, Y_test = train_test_split(X.values, Y.values, test_size=0.9)

         clf_rdfore = AdaBoostClassifier()
         clf_rdfore = clf_rdfore.fit(X_train,Y_train)

         y_hat = clf_rdfore.predict(X_test)


         print('Test accuracy:',np.round(clf_rdfore.score(X_test, Y_test),3), ', Cross Entropy
         Loss is:', log_loss(Y_test,y_hat))
         calculate_metrics(Y_test,y_hat)
         plot_ROC(Y_test,X_test,clf_rdfore)
         MAE(Y_test,y_hat)
```

```
Test accuracy: 0.891 , Cross Entropy Loss is: 3.7551821362867495
Confusion matrix is:
[[32337   451]
 [ 3556   511]]
We have 32848 correct observations and 4007 misclassifications.
              precision    recall  f1-score   support

           0       0.90      0.99      0.94     32788
           1       0.53      0.13      0.20      4067

   micro avg       0.89      0.89      0.89     36855
   macro avg       0.72      0.56      0.57     36855
weighted avg       0.86      0.89      0.86     36855
```


32337
451
3556
511
0
1
0
1
30000
24000
18000
12000
6000



Out[62]: 4007

2. SVC

VII Unbalanced Data Problem

```
In [63]: sns.set_style('whitegrid')

In [64]: def make_balanced(X,Y):
             DF = X.copy()
             DF['Y'] = Y
             DF_Yes = DF[DF.Y==1]#.info()
             DF_No = DF[DF.Y==0]#.info()
             DF_B = DF_No.sample(len(DF_Yes))
             D = pd.concat([DF_B,DF_Yes])
             D = D.reindex(np.random.permutation(D.index))
```

```
        D = D[:int(len(D)/2)]#, D[int(len(D)/2):]
        return D
```

```
In [65]: D = make_balanced(X,Y)
```

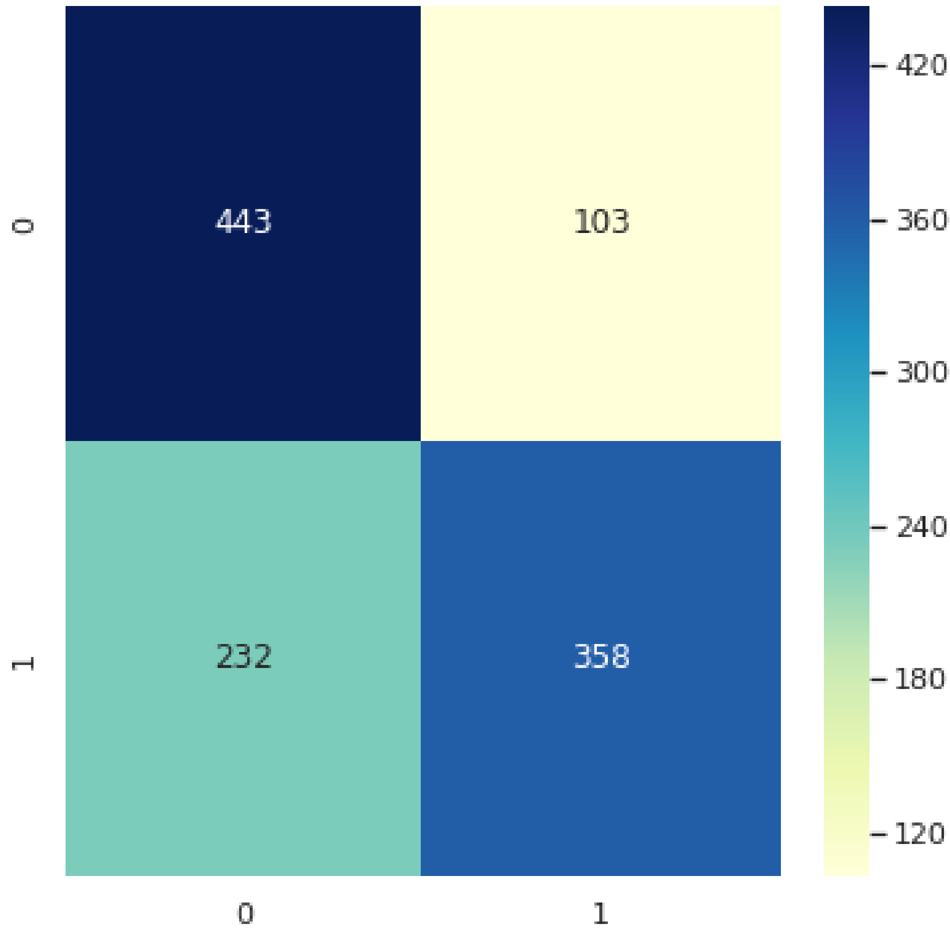
```
In [66]: D.shape
```

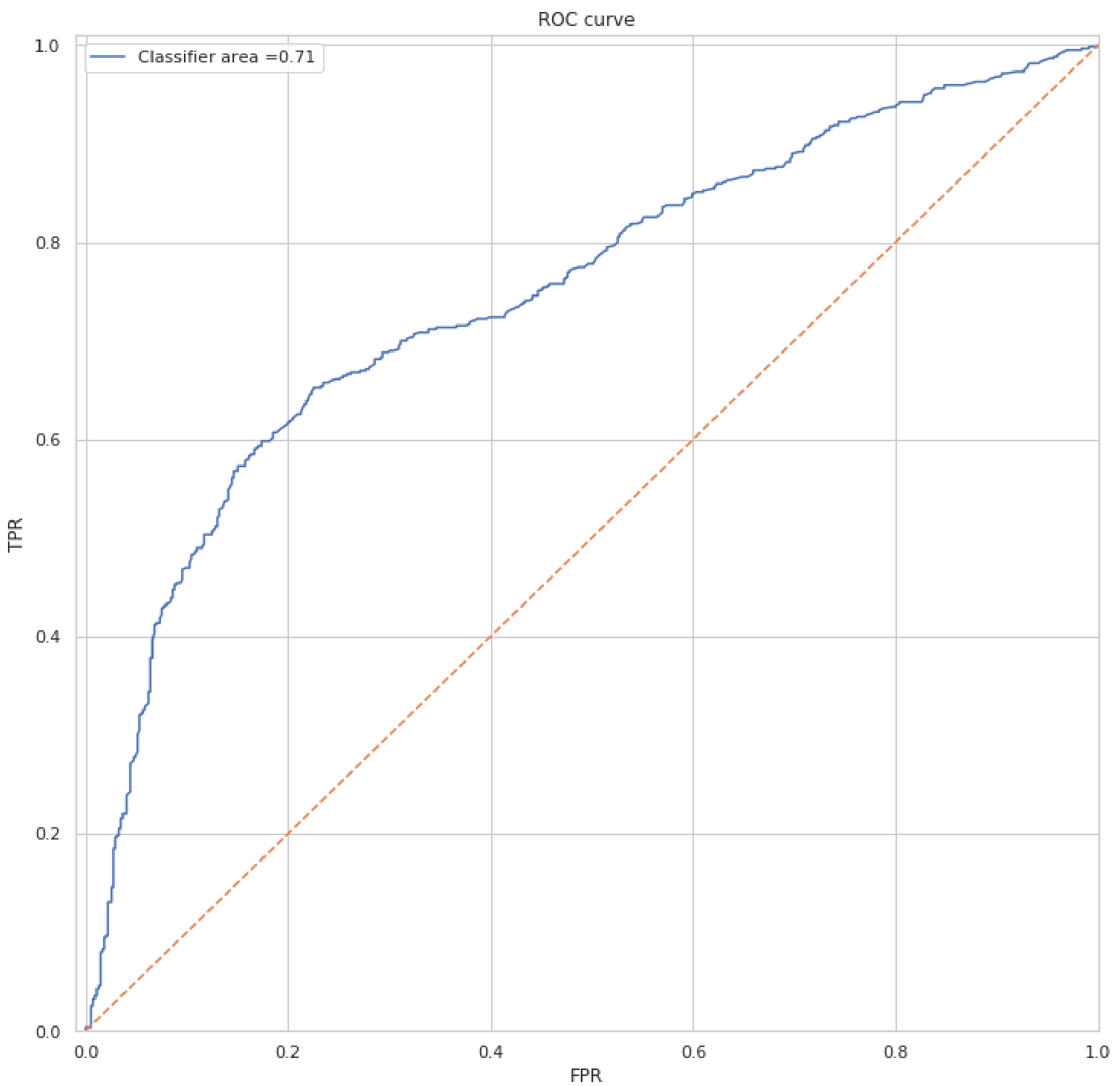
```
Out[66]: (4544, 32)
```

```
In [67]: X_B = D.drop(columns='Y')
         Y_B = D['Y']
```

```
In [68]: X_train, X_test, Y_train, Y_test = train_test_split(X_B.values, Y_B.values,
         test_size=0.25)
         clf_rdfore = AdaBoostClassifier()
         clf_rdfore = clf_rdfore.fit(X_train,Y_train)

         y_hat = clf_rdfore.predict(X_test)


         print('Test accuracy:',np.round(clf_rdfore.score(X_test, Y_test),3), ', Cross Entropy
         Loss is:', log_loss(Y_test,y_hat))
         calculate_metrics(Y_test,y_hat)
         plot_ROC(Y_test,X_test,clf_rdfore)
         MAE(Y_test,y_hat)
```

```
Test accuracy: 0.705 , Cross Entropy Loss is: 10.185363072914074
Confusion matrix is:
[[443 103]
 [232 358]]
We have 801 correct observations and 335 misclassifications.
              precision    recall  f1-score   support

           0       0.66      0.81      0.73       546
           1       0.78      0.61      0.68       590

   micro avg       0.71      0.71      0.71      1136
   macro avg       0.72      0.71      0.70      1136
weighted avg       0.72      0.71      0.70      1136
```


443
103
232
358
0
1
0
1
420
360
300
240
180
120




```
Out[68]: 335
```

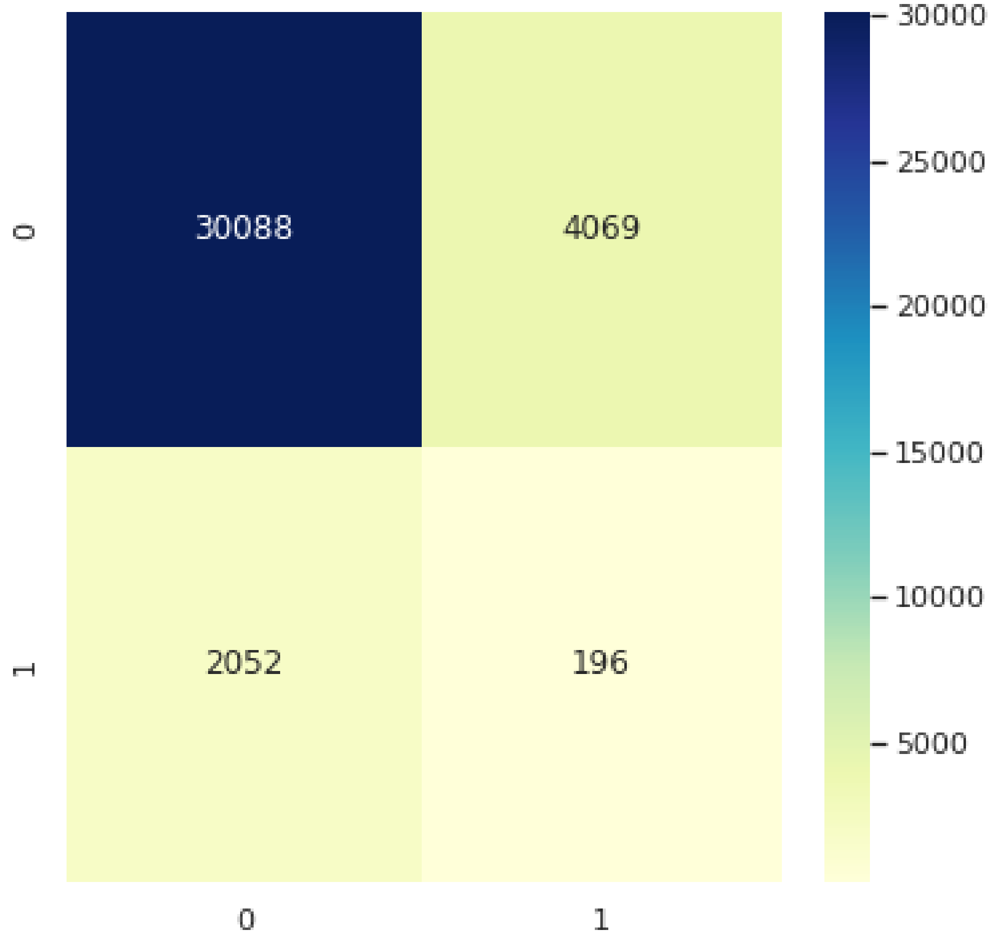
```
In [69]: X_R = X.drop(D.index)
         Y_R = Y.drop(D.index)
```

```
In [70]: C = [LogisticRegression(), AdaBoostClassifier(), KNeighborsClassifier(),
         RandomForestClassifier()]

         def calc_sampling_imb(X,Y):
             L_NAME = ['lr','ada','knn','rforest']
             L_X, L_Y, L_YHAT, L_PROB = [],[],[],[]

             for cl in C:
                 y_hat = cross_val_predict(cl, X, Y, cv=10, n_jobs=-1)
                 probs = cross_val_predict(cl, X, Y, cv=10, method='predict_proba',n_jobs=-1)

                 #print('Test accuracy:',np.round(c.score(X_R, Y_R),3), ', Cross Entropy Loss
         is:', log_loss(Y_R,y_hat))
```

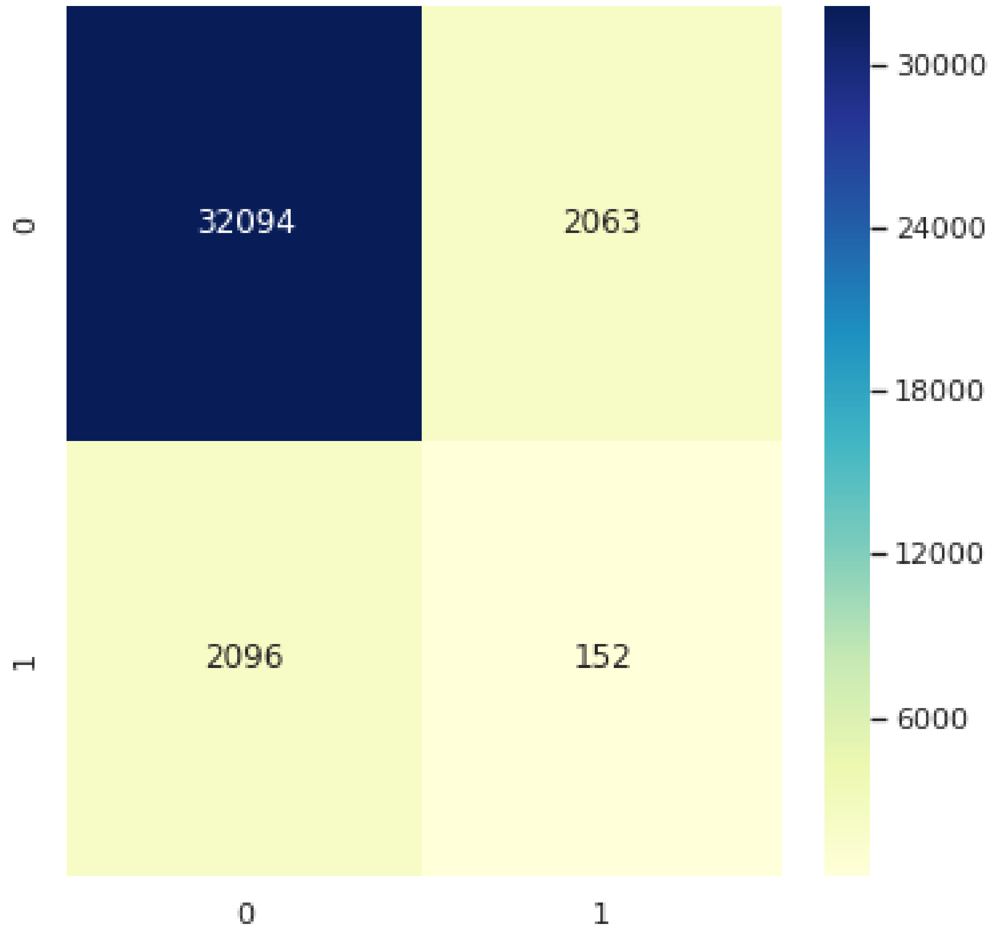
```
                calculate_metrics(Y,y_hat)
                #plot_ROC(Y_R,X,c)
                print('MAE:',MAE(Y,y_hat))

                                #L_NAME.append(name)
                L_X.append(X.values)
                L_Y.append(Y.values)
                L_YHAT.append(y_hat)
                L_PROB.append(probs)

            plot_ROC2(L_Y,L_X,L_YHAT,L_PROB,L_NAME)

            #plot_ROC3(L_Y,L_X,L_CLASS,L_NAME)
```

```
In [71]: calc_sampling_imb(X_R,Y_R)
```

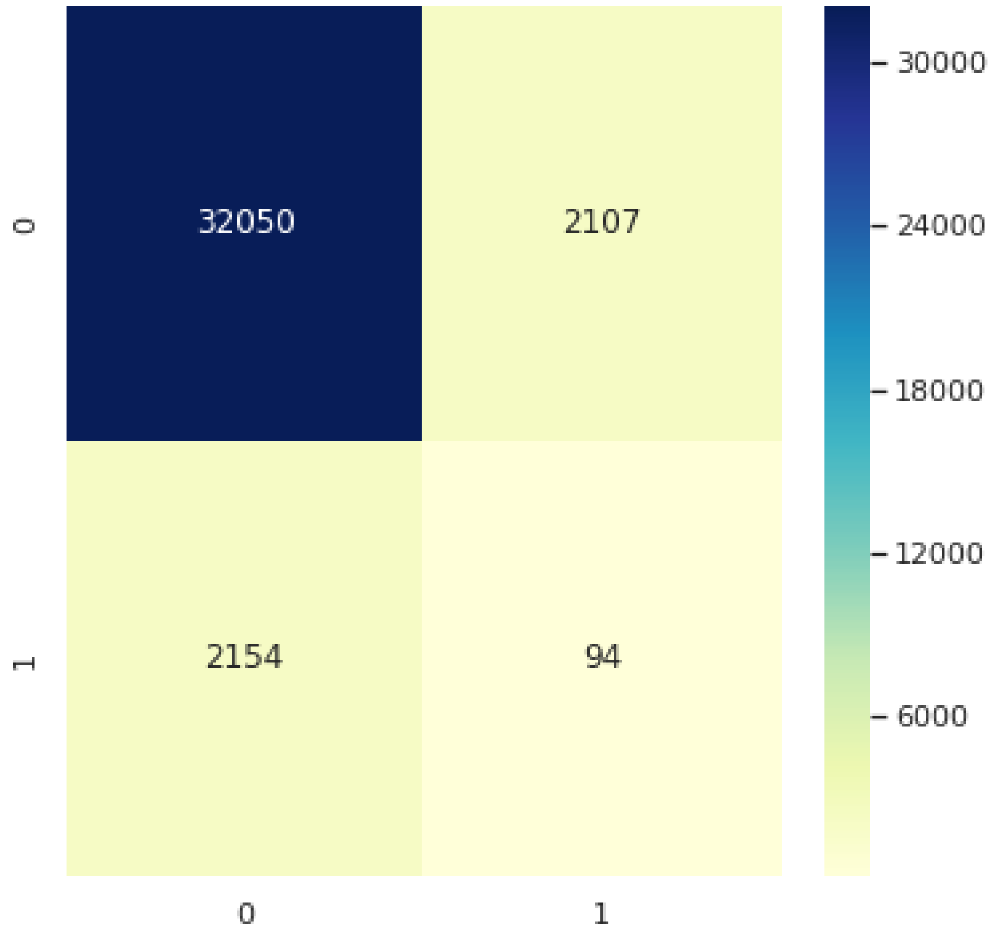
```
Confusion matrix is:
[[30088  4069]
 [ 2052   196]]
We have 30284 correct observations and 6121 misclassifications.
              precision    recall  f1-score   support

           0       0.94      0.88      0.91     34157
           1       0.05      0.09      0.06      2248

   micro avg       0.83      0.83      0.83     36405
   macro avg       0.49      0.48      0.48     36405
weighted avg       0.88      0.83      0.86     36405
```



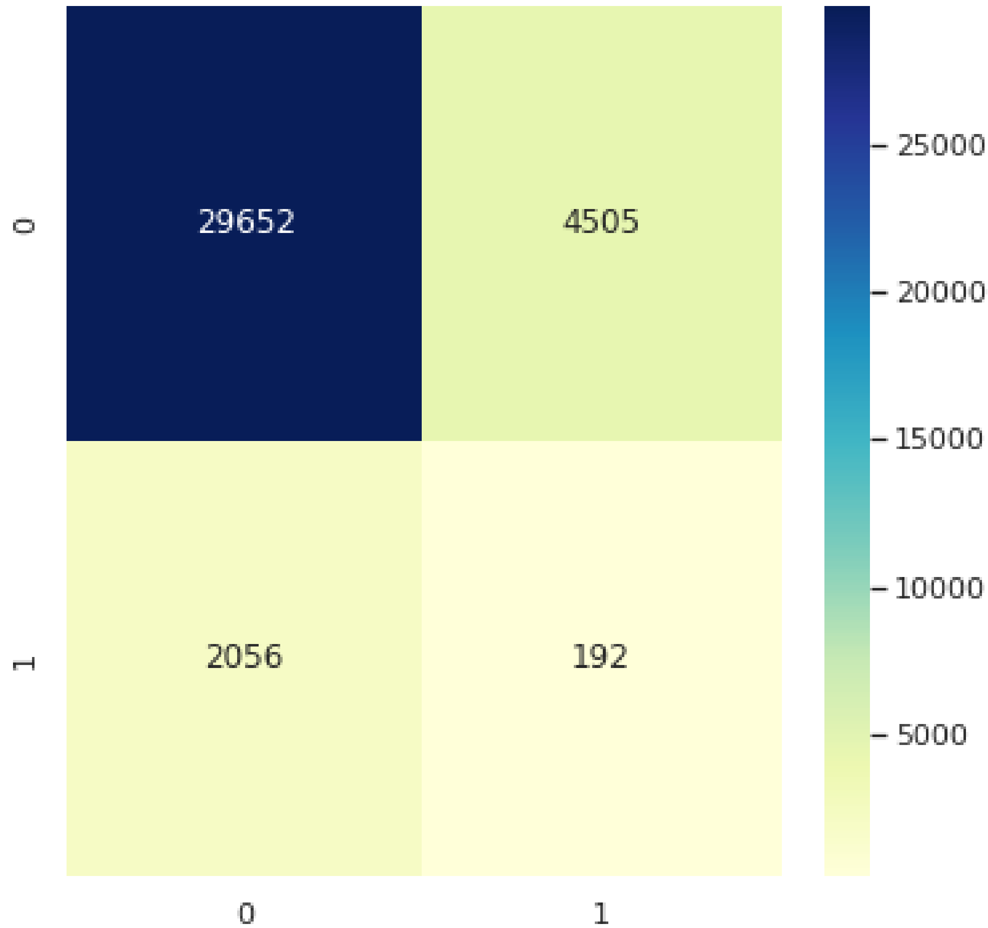

```
MAE: 6121
Confusion matrix is:
[[32094  2063]
 [ 2096   152]]
We have 32246 correct observations and 4159 misclassifications.
              precision    recall  f1-score   support

           0       0.94      0.94      0.94     34157
           1       0.07      0.07      0.07      2248

   micro avg       0.89      0.89      0.89     36405
   macro avg       0.50      0.50      0.50     36405
weighted avg       0.88      0.89      0.89     36405
```



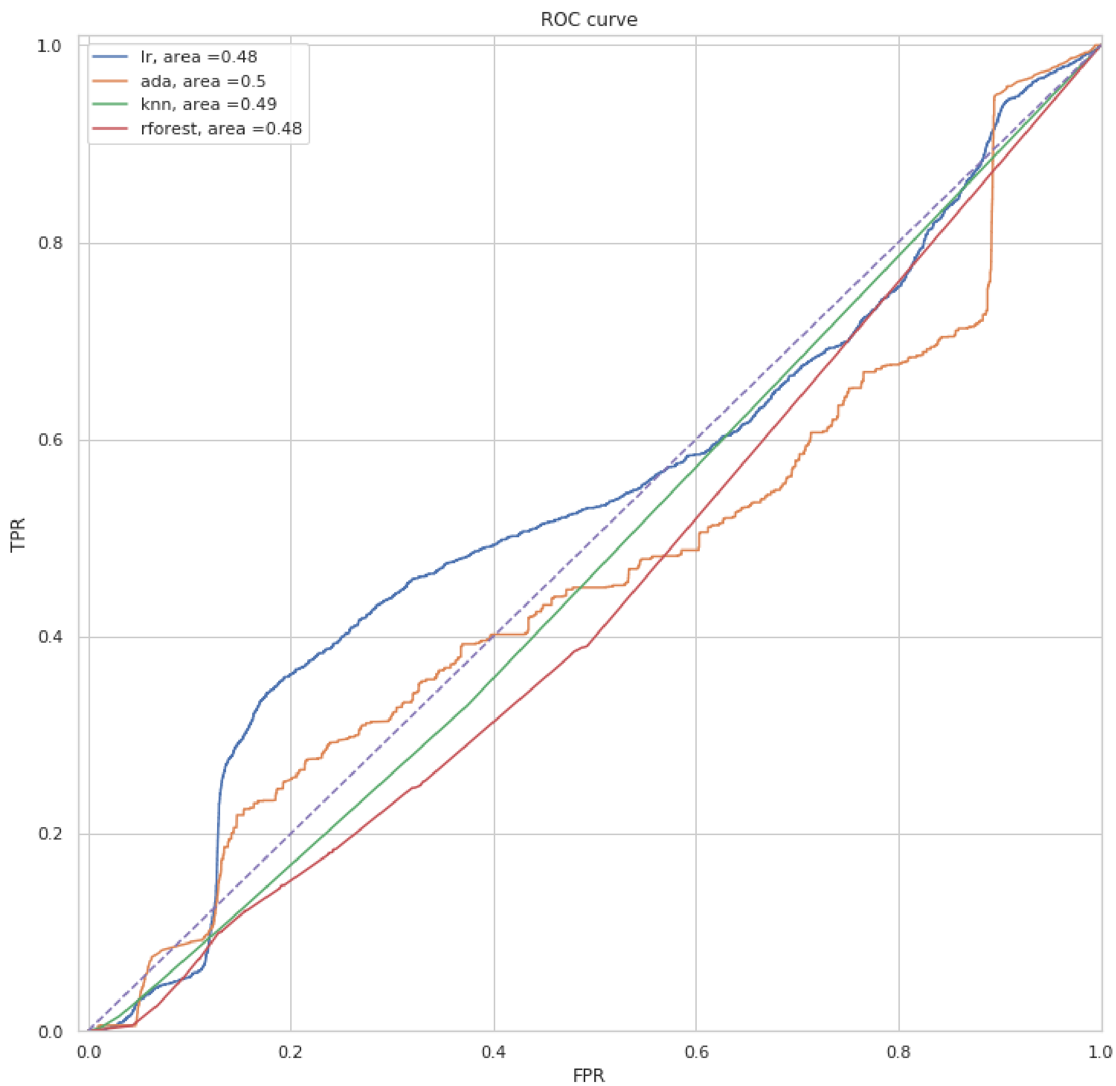

```
MAE: 4159
Confusion matrix is:
[[32050  2107]
 [ 2154    94]]
We have 32144 correct observations and 4261 misclassifications.
              precision    recall  f1-score   support

           0       0.94      0.94      0.94     34157
           1       0.04      0.04      0.04      2248

   micro avg       0.88      0.88      0.88     36405
   macro avg       0.49      0.49      0.49     36405
weighted avg       0.88      0.88      0.88     36405
```




```
MAE: 4261
Confusion matrix is:
[[29652  4505]
 [ 2056   192]]
We have 29844 correct observations and 6561 misclassifications.
              precision    recall  f1-score   support

           0       0.94      0.87      0.90     34157
           1       0.04      0.09      0.06      2248

   micro avg       0.82      0.82      0.82     36405
   macro avg       0.49      0.48      0.48     36405
weighted avg       0.88      0.82      0.85     36405
```




```
MAE: 6561
```

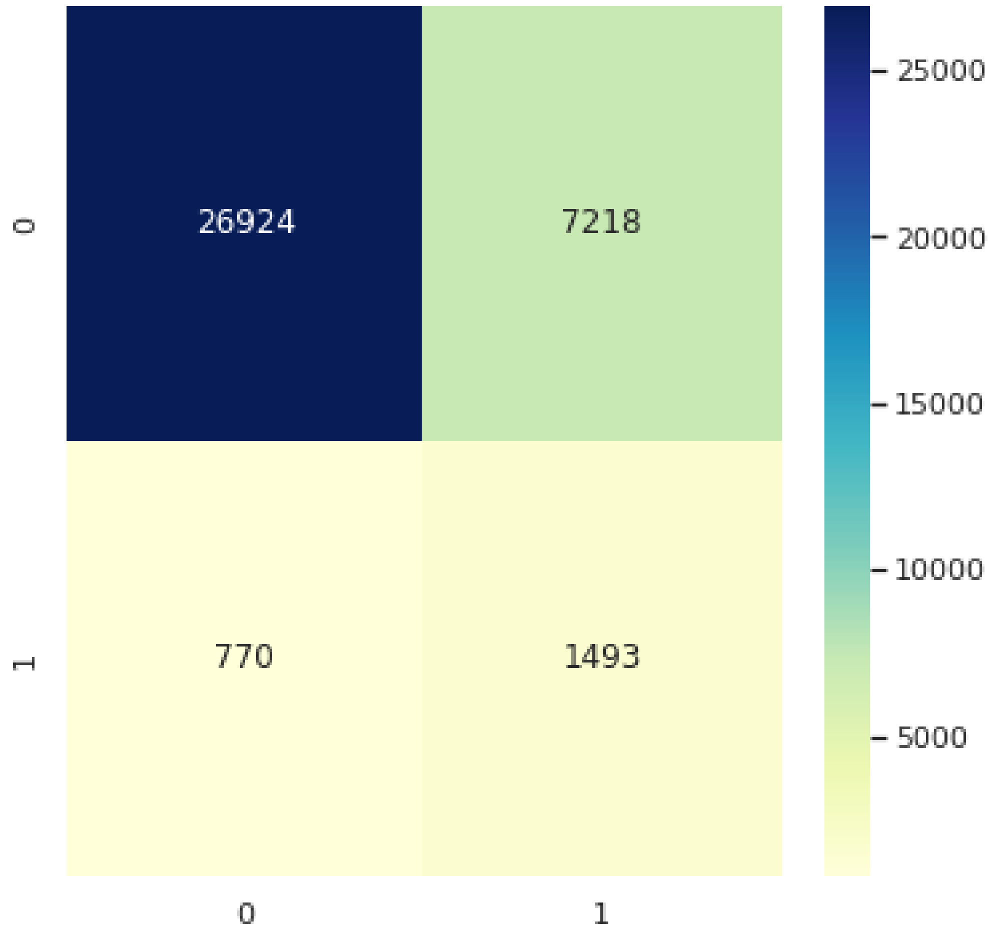



```
In [72]: def train_B_test(X,Y):
             L_NAME = ['lr','ada','knn','rforest']
             L_X, L_Y, L_YHAT, L_PROB = [],[],[],[]

             D = make_balanced(X,Y)
             X_B = D.drop(columns='Y')
             Y_B = D['Y']

             X_R = X.drop(D.index)
             Y_R = Y.drop(D.index)

             for i,cl in enumerate(C):

                 c = cl.fit(X_B,Y_B)
                 y_hat = c.predict(X_R)
                 probs = c.predict_proba(X_R)
                 print(L_NAME[i])
                 print('Test accuracy:',np.round(c.score(X_R, Y_R),3), ', Cross Entropy Loss
```

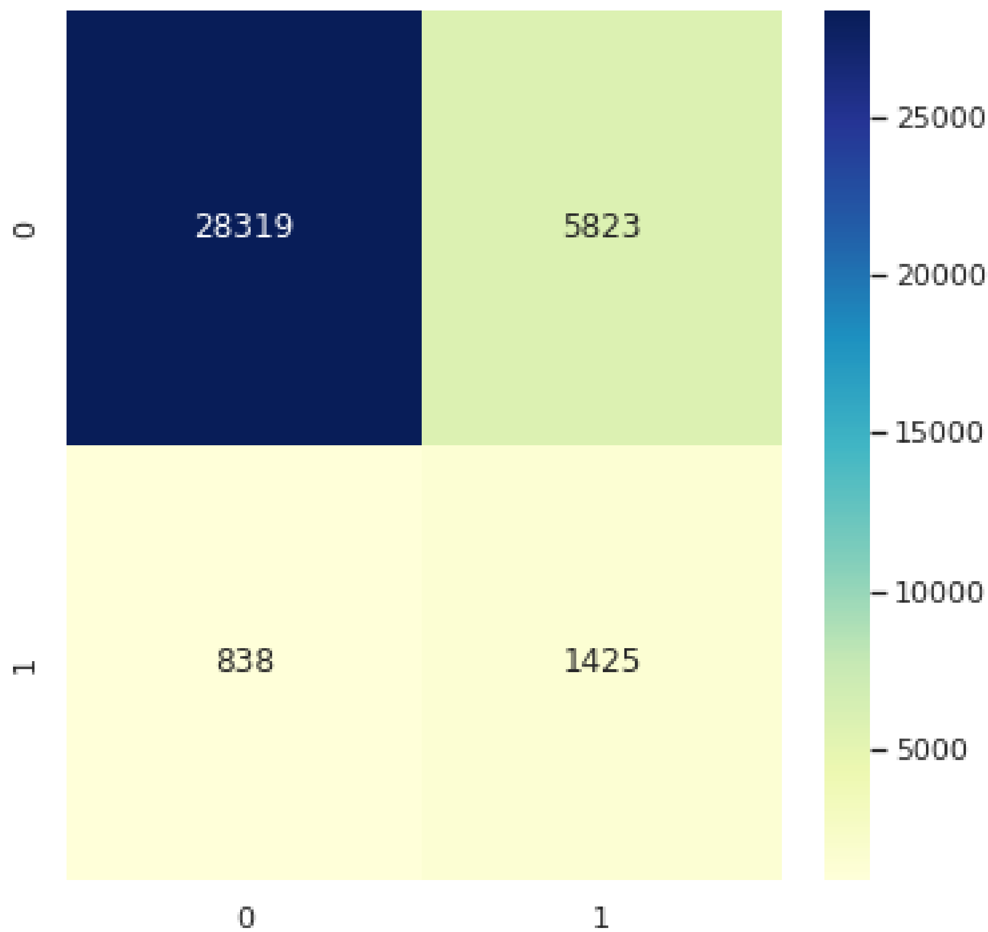
```
         is:', log_loss(Y_R,y_hat))
                 calculate_metrics(Y_R,y_hat)
                 MAE(Y_R,y_hat)

                 L_X.append(X_R.values)
                 L_Y.append(Y_R.values)
                 L_YHAT.append(y_hat)
                 L_PROB.append(probs)

             plot_ROC2(L_Y,L_X,L_YHAT,L_PROB,L_NAME)
```

```
In [73]: train_B_test(X,Y)
```

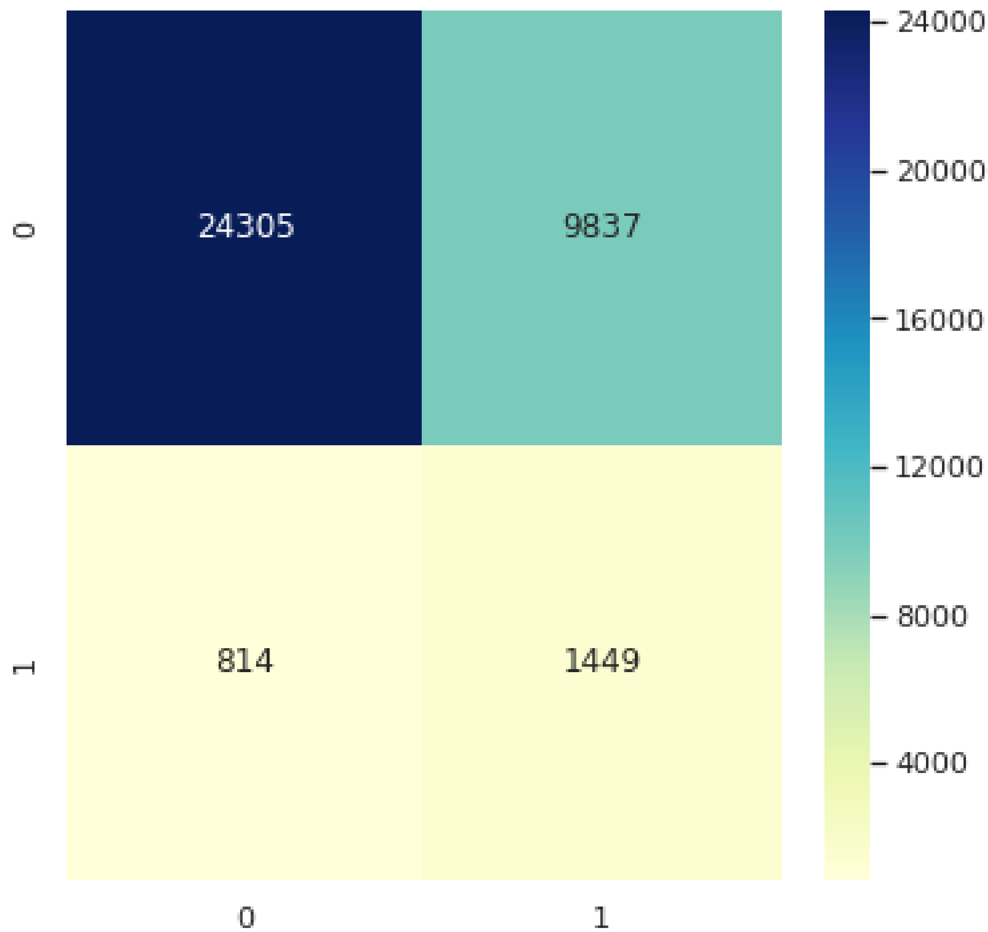
```
lr
Test accuracy: 0.781 , Cross Entropy Loss is: 7.578670988512503
Confusion matrix is:
[[26924  7218]
 [  770  1493]]
We have 28417 correct observations and 7988 misclassifications.
              precision    recall  f1-score   support

           0       0.97      0.79      0.87     34142
           1       0.17      0.66      0.27      2263

   micro avg       0.78      0.78      0.78     36405
   macro avg       0.57      0.72      0.57     36405
weighted avg       0.92      0.78      0.83     36405
```



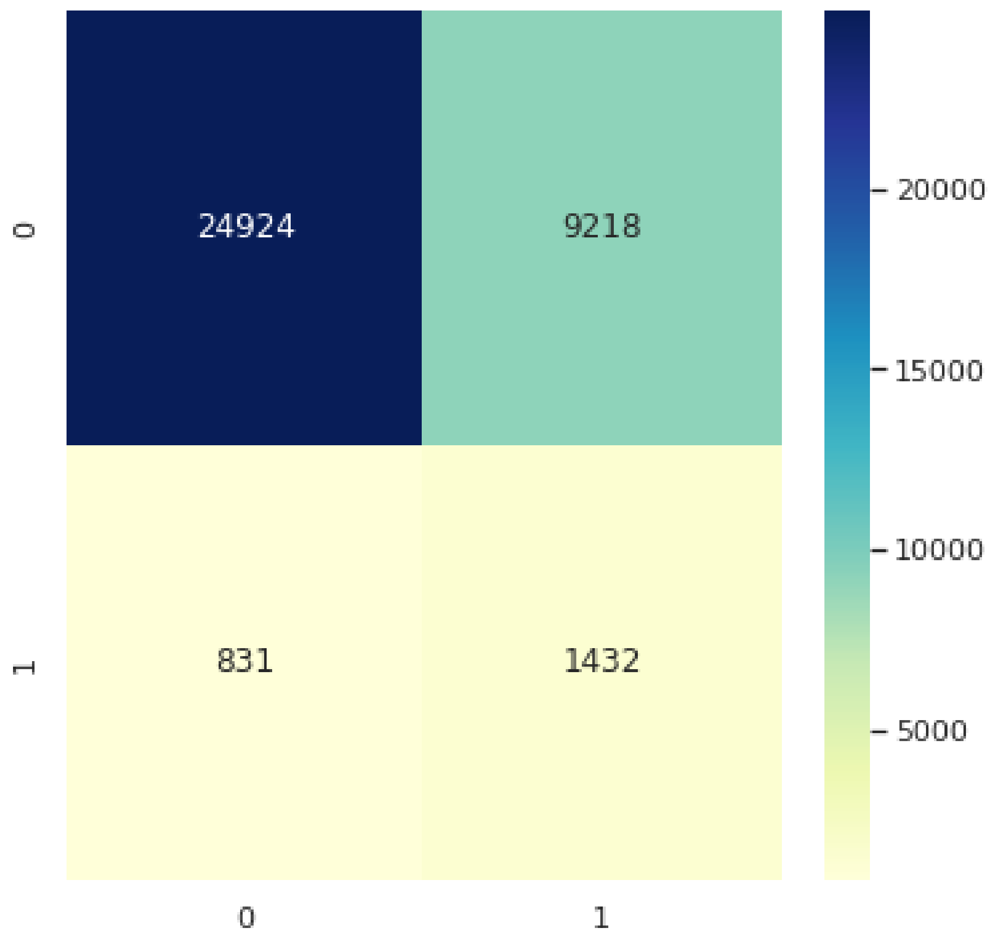

```
ada
Test accuracy: 0.817 , Cross Entropy Loss is: 6.319666134386378
Confusion matrix is:
[[28319  5823]
 [  838  1425]]
We have 29744 correct observations and 6661 misclassifications.
              precision    recall  f1-score   support

           0       0.97      0.83      0.89     34142
           1       0.20      0.63      0.30      2263

   micro avg       0.82      0.82      0.82     36405
   macro avg       0.58      0.73      0.60     36405
weighted avg       0.92      0.82      0.86     36405
```

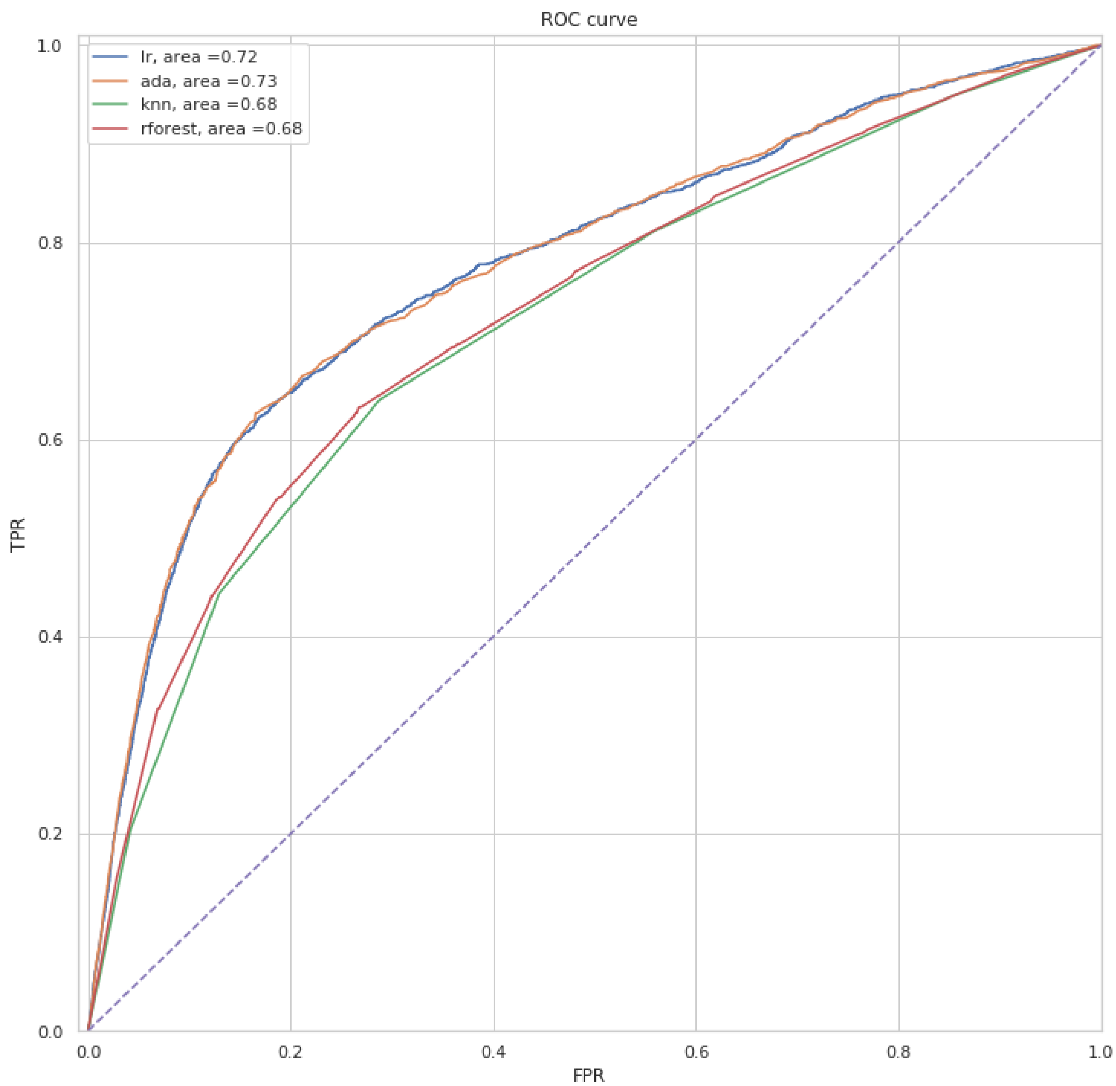



```
knn
Test accuracy: 0.707 , Cross Entropy Loss is: 10.105215575391142
Confusion matrix is:
[[24305  9837]
 [  814  1449]]
We have 25754 correct observations and 10651 misclassifications.
              precision    recall  f1-score   support

           0       0.97      0.71      0.82     34142
           1       0.13      0.64      0.21      2263

   micro avg       0.71      0.71      0.71     36405
   macro avg       0.55      0.68      0.52     36405
weighted avg       0.92      0.71      0.78     36405
```




```
rforest
Test accuracy: 0.724 , Cross Entropy Loss is: 9.534062207981568
Confusion matrix is:
[[24924  9218]
 [  831  1432]]
We have 26356 correct observations and 10049 misclassifications.
              precision    recall  f1-score   support

           0       0.97      0.73      0.83     34142
           1       0.13      0.63      0.22      2263

   micro avg       0.72      0.72      0.72     36405
   macro avg       0.55      0.68      0.53     36405
weighted avg       0.92      0.72      0.79     36405
```


24924
9218
831
1432
0
1
0
1
20000
15000
10000
5000

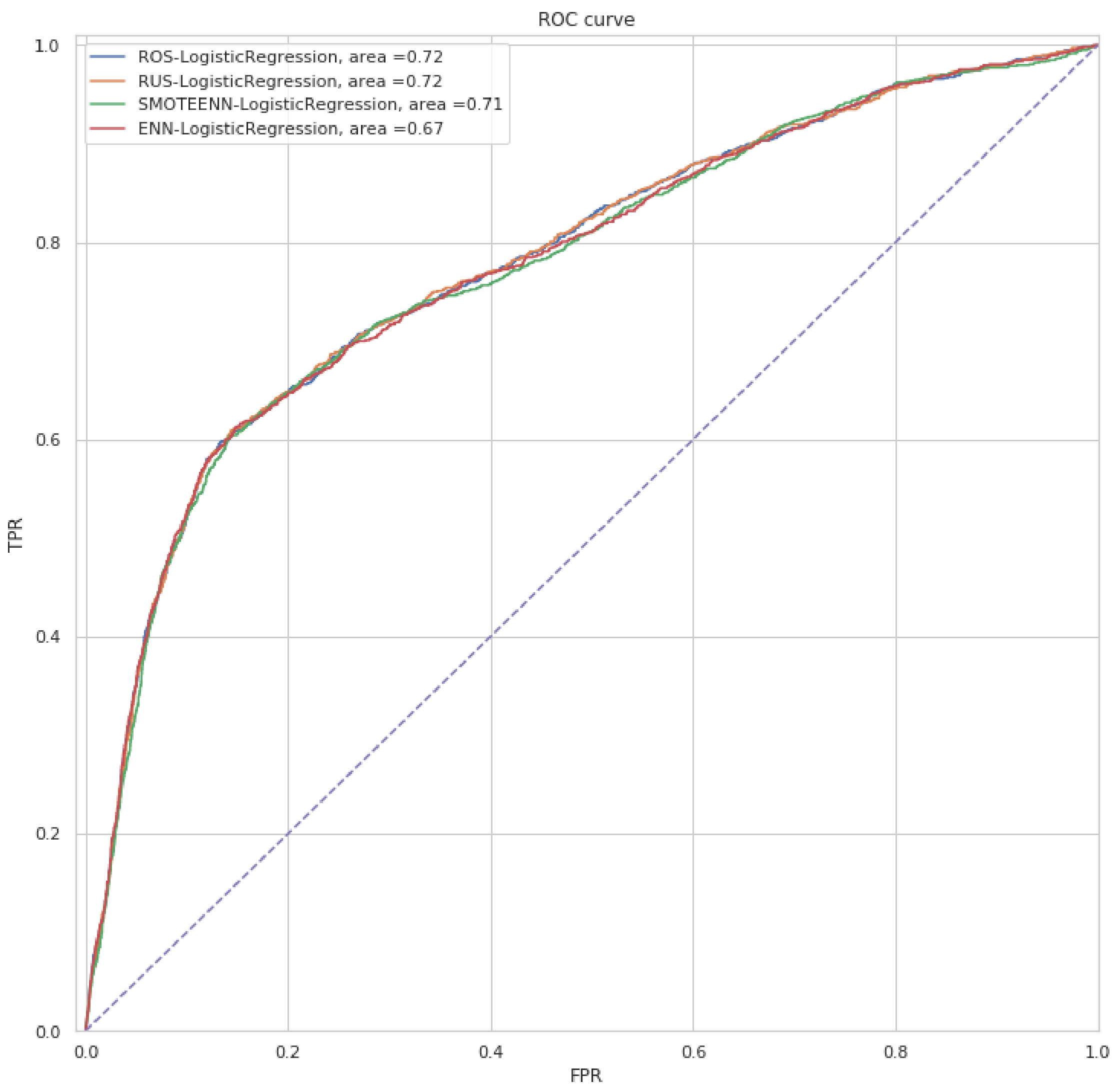


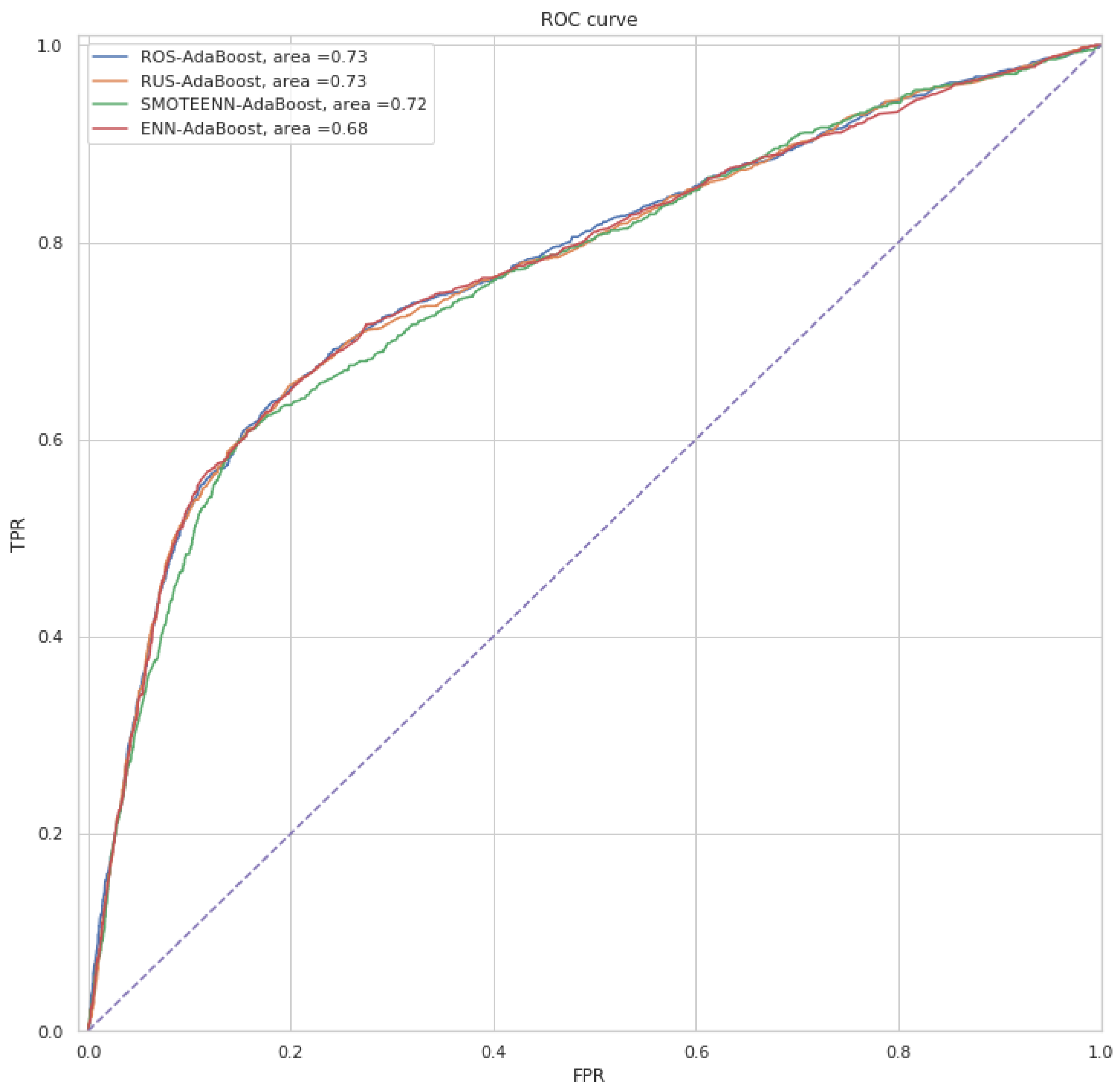
VII Test Over/Under Sampling Methods

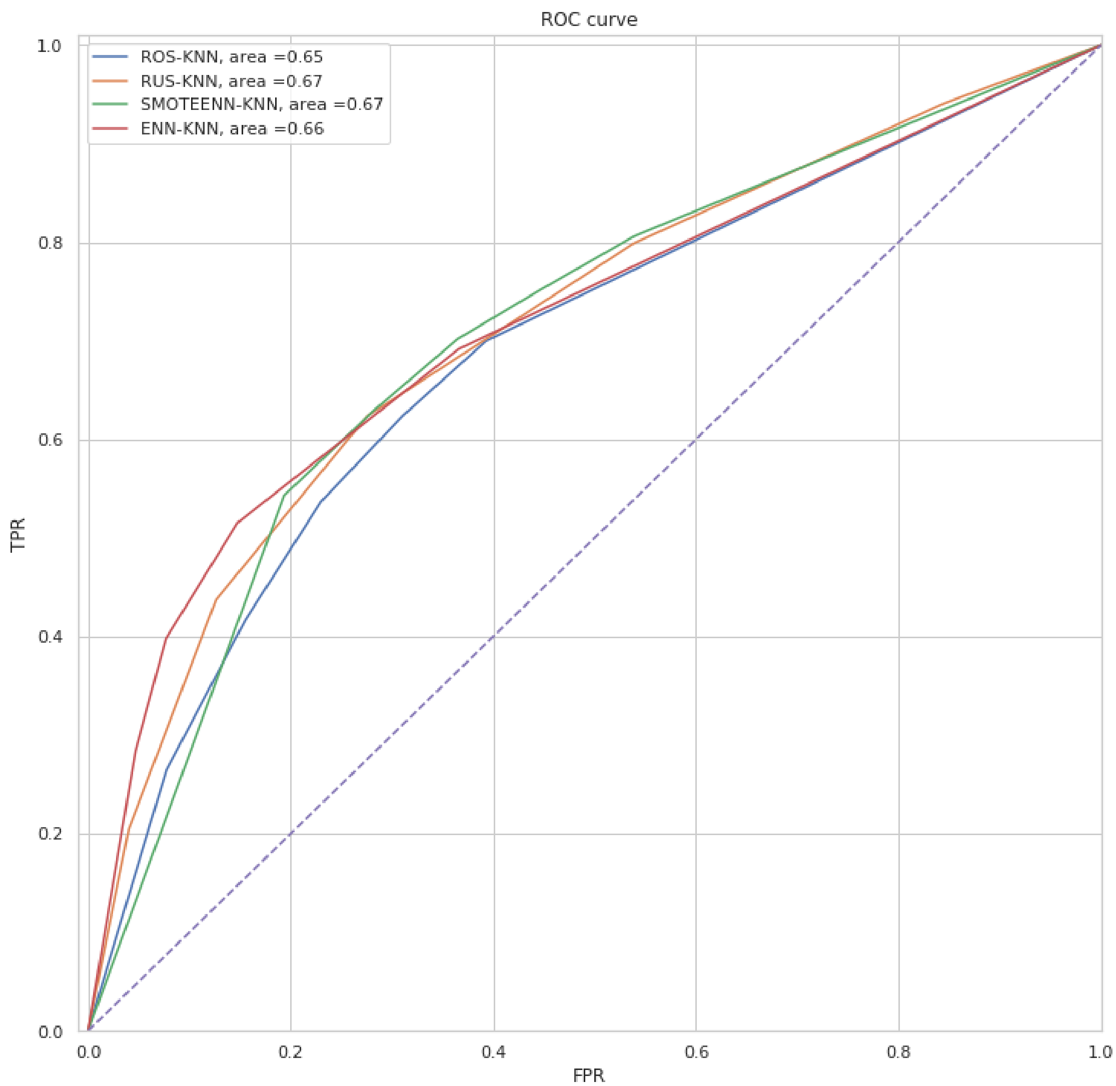
```
In [74]: sns.set_style('whitegrid')

In [75]: def calc_performance(c,name):

             classifier = [name, c]

             over_samplers = [
                 #['ADA-SYN', ADASYN()],
                 ['ROS', RandomOverSampler()],
                 ['RUS', RandomUnderSampler()],
                 #['SMOTENC', SMOTENC()],
                 ['SMOTEENN', SMOTEENN()],
                 ['ENN', EditedNearestNeighbours()],
             ]
```

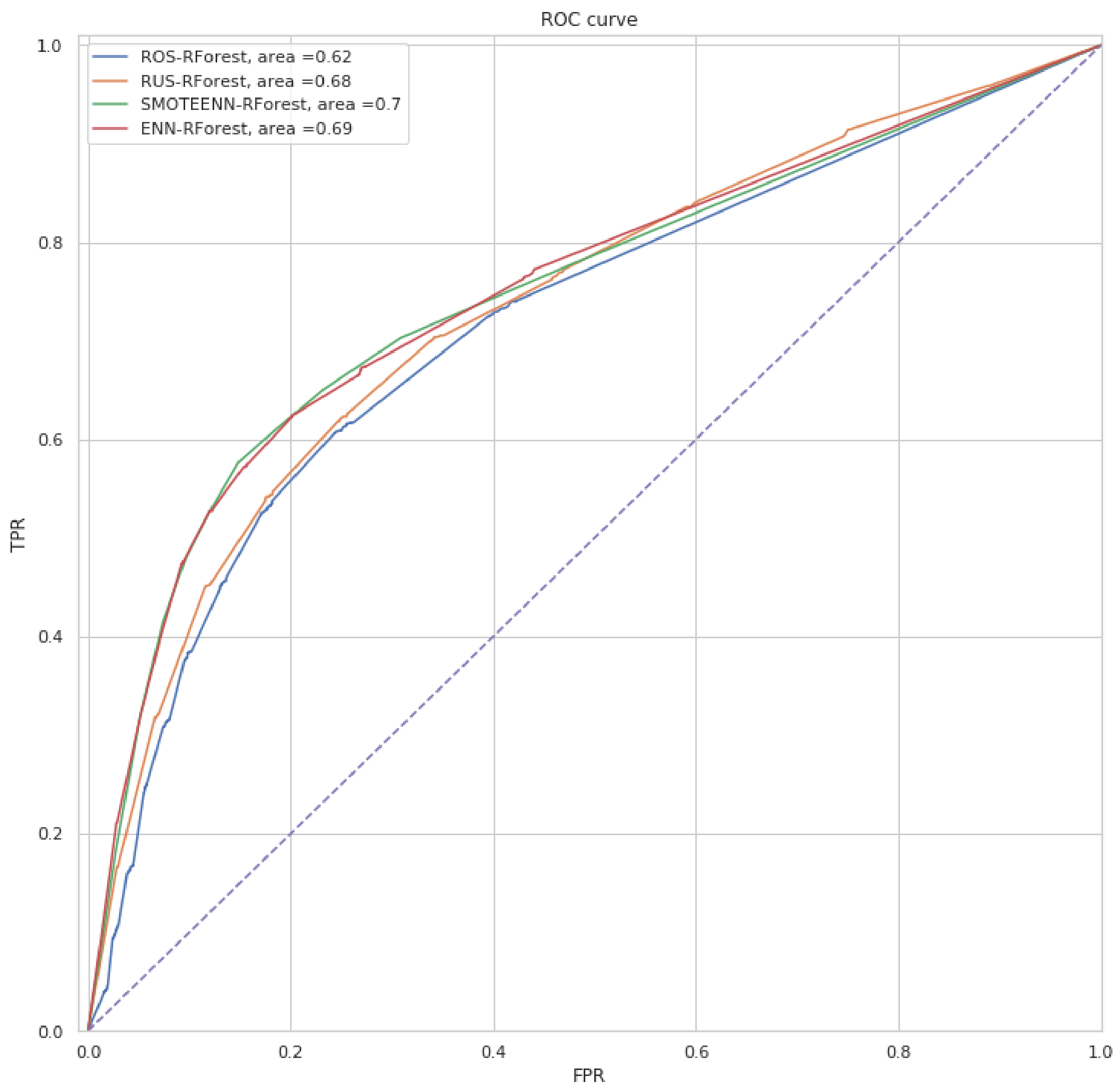
```
    pipelines = [
        ['{}-{}'.format(s[0], classifier[0]),
         make_pipeline(s[1], classifier[1])]
        for s in over_samplers
    ]


    calc_sampling(pipelines,X,Y)


def calc_sampling(pipelines,X,Y):
    L_NAME, L_X, L_Y, L_YHAT, L_PROB = [],[],[],[],[]

    X_train, X_test, Y_train, Y_test = train_test_split(X.values, Y.values,
test_size=0.25)
    for name, pipeline in pipelines:

        p = pipeline.fit(X_train,Y_train)
        y_hat = p.predict(X_test)
        probs = p.predict_proba(X_test)
        #calculate_metrics(Y_test,y_hat)

        L_NAME.append(name)
        L_X.append(X_test)
        L_Y.append(Y_test)
        L_YHAT.append(y_hat)
        L_PROB.append(probs)

    plot_ROC2(L_Y,L_X,L_YHAT,L_PROB,L_NAME)
```

```
In [76]: NAME = ['LogisticRegression','AdaBoost','KNN','RForest']
         C_dict = dict(zip(NAME,C))

         for name,c in C_dict.items():
             calc_performance(c,name)
```


ROC curve
ROS-LogisticRegression, area =0.72
RUS-LogisticRegression, area =0.72
SMOTEENN-LogisticRegression, area =0.71
ENN-LogisticRegression, area =0.67
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve
ROS-AdaBoost, area =0.73
RUS-AdaBoost, area =0.73
SMOTEENN-AdaBoost, area =0.72
ENN-AdaBoost, area =0.68
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve
ROS-KNN, area =0.65
RUS-KNN, area =0.67
SMOTEENN-KNN, area =0.67
ENN-KNN, area =0.66
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0



VII selection of relevant variables from balanced dataset

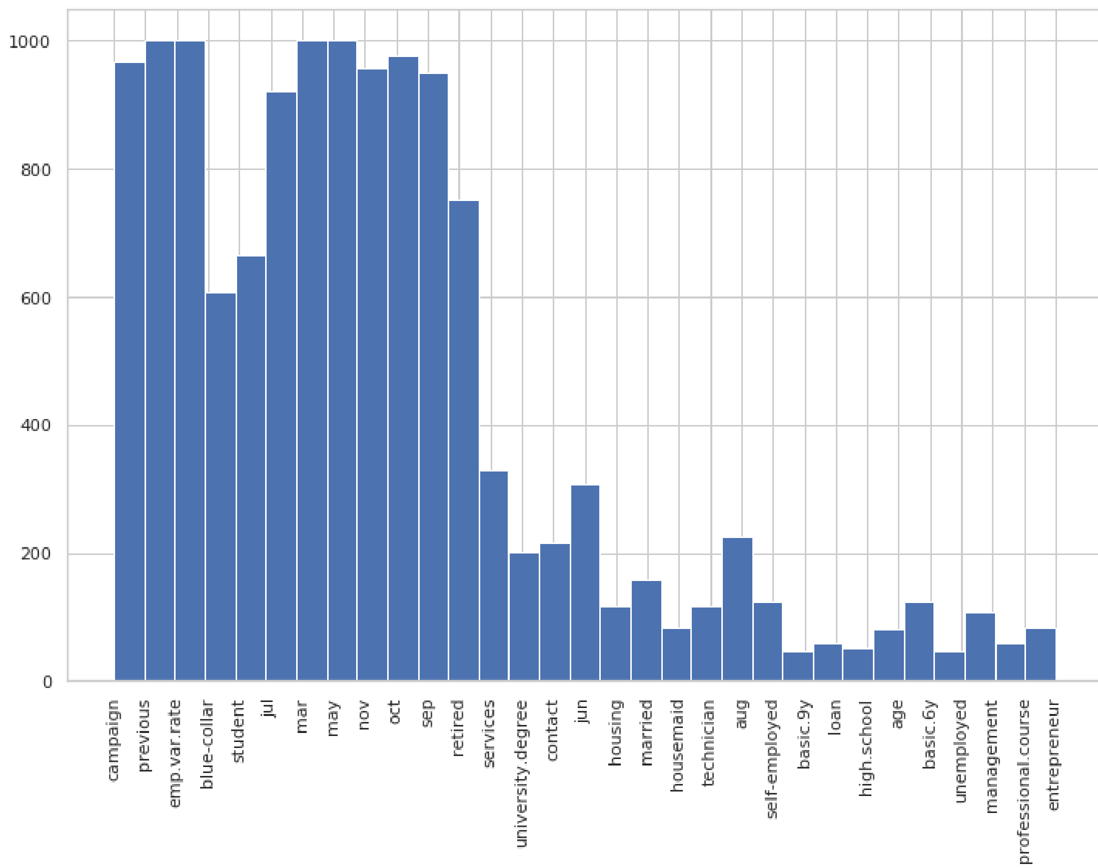
```
In [77]: L, Counts,Coeffs = [],[],[]
         for i in range(1000):
             D = make_balanced(X,Y)
             X_B = D.drop(columns='Y')
             Y_B = D['Y']

             logit = sm.Logit(Y_B, X_B).fit(disp=0)
             #display(logit.summary())

             alpha = 0.10
             a = logit.pvalues < alpha

             X4 = X_B[X_B.columns[a]]
             #print("Not Statistically significant regressors are:")
```

```
        temp = list(X_B.columns[a])
        L.append(temp)
        Counts.append(len(temp))
        Coeffs.append(logit.params.loc[temp])
```

In [ ]:

In [78]:
```
L = [y for x in L for y in x]
```

In [79]:
```
plt.figure(figsize=(12,8))
plt.hist(L,bins=len(set(L)))
plt.xticks(rotation=90)
plt.show()
```



In [80]:
```
from collections import Counter
d = Counter(L)

import operator
sorted_d = dict(sorted(d.items(), key=operator.itemgetter(1), reverse=True))
```

In [81]:
```
TopVars = []
print('Most frequently significant variables:')
for i, (k,v) in enumerate(sorted_d.items()):
    if v > 500:
        print(k, ':',v)
        TopVars.append(k)
```

```
Most frequently significant variables:
previous : 1000
emp.var.rate : 1000
mar : 1000
may : 1000
oct : 977
campaign : 968
nov : 957
sep : 950
jul : 921
retired : 751
student : 665
blue-collar : 606
```

```
In [82]: V = []
         for v in TopVars:
             summ = 0
             count = 0
             for c in Coeffs:
                 try:
                     summ += c.loc[v]
                     count +=1
                 except KeyError:
                     pass
             V.append(summ/count)
```

```
In [83]: list(np.round(np.array(V),3))
```

```
Out[83]: [0.349,
          -0.476,
          1.271,
          -0.654,
          0.842,
          -0.051,
          -0.447,
          0.847,
          0.435,
          0.509,
          0.55,
          -0.283]
```

```
In [84]: list(np.round(np.exp(np.array(V)),3))
```

```
Out[84]: [1.418,
          0.621,
          3.565,
          0.52,
          2.322,
          0.95,
          0.64,
          2.332,
          1.544,
          1.663,
          1.733,
          0.753]
```

```
In [ ]:
```