NAME : PIYUSHA PARDESHI

BATCH NO : PGA21

INSTITUTE : IMARTICUS

Machine Learning using Python

Exam – Paper 1

[Time: 4 hrs]

[Total Marks: 50]

Part I: Supervised Learning [Total Marks - 30]

Given is the ‘Portugal Bank Marketing’ dataset:

Bank client data:

1) age (numeric)

2) job: type of job(categorical:"admin.","bluecollar","entrepreneur","housemaid","management","retired","selfemployed","services","student","technician","unempl

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

4) education: education of individual (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","u nknown")

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")

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) dayofweek: 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.

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") 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) concavepoints_se: standard error for number of concave portions of the contour

20) euribor3m: euribor 3 month rate - daily indicator (numeric)

21) nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

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

Perform the following tasks: Marks

Q1. What does the primary analysis of several categorical features reveal? [5]

Q2. Perform the following Exploratory Data Analysis tasks: a. Missing Value Analysis

b. Label Encoding wherever required

c. Selecting important features based on Random Forest

d. Handling unbalanced data using SMOTE

e. Standardize the data using the anyone of the scalers provided by sklearn [10]

Q3. Build the following Supervised Learning models: a. Logistic Regression

b. AdaBoost

c. Naïve Bayes

d. KNN

e. SVM [10]

Q4. Tabulate the performance metrics of all the above models and tell which model performs better in predicting if the client will subscribe to term deposit or not

PREDICTING TERM DEPOSIT SUBSCRIPTION BY CLIENT

IMPORT LIBRARIES

In [1]:
```
import pandas as pd
```

```
import numpy as np

from sklearn.model_selection import train_test_split,cross_val_score,KFold,GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,precision_recall_curve,recall_score,roc_curve,f1_score,confusic
from scipy.stats import skew
from sklearn import neighbors
from sklearn import metrics
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing, svm

from sklearn import tree
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
import statsmodels.api as smapi
from sklearn.feature_selection import f_classif
from sklearn.linear_model import LogisticRegression

# feature selection
# RFE (recursive feature elimination)
from sklearn.feature_selection import RFE

# visualisation
import seaborn as sns
import matplotlib.pyplot as plt
import pylab

import warnings
warnings.filterwarnings('ignore')
```

READ DATA

In [2]:
```
bank = pd.read_csv(r"C:\Users\Admin\Sriraman sir exams\Final Machine Learning Exam\Final Machine Learning Exam ir
```

In [3]:
```
bank.head()
```

Out[3]:

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

5 rows × 21 columns

In [4]:
```
bank.tail()
```

Out[4]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutc
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	999	0	nonexis
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexis
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	...	2	999	0	nonexis
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexis
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	3	999	1	fa

5 rows × 21 columns

EXTRACT DATA INFORMATION

In [5]:
```
bank.columns
```

Out[5]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
```

```
 'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
dtype='object')
```

In [6]:

```
len(bank.columns)
```

Out[6]:

```
21
```

In [7]:

```
bank.shape
```

Out[7]:

```
(41188, 21)
```

- We have 41188 rows and 21 columns

In [8]:

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [9]:

```
bank.describe()
```

Out[9]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.emp
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.0
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.0
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.2
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.6
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.1
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.0
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.1
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.1

CHECK FOR DATA TYPES

In [10]:

```
bank.dtypes
```

Out[10]:

```
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
```

PRINT UNIQUE VALUES OF DATA WITH DATA TYPES

In [11]:

```
def checkuniquevalues(data,cols):
    for c in cols:
        print("Column name:",c,data[c].dtypes)
        print(pd.unique(data[c]))
        print("---------------------------------------------")
```

In [12]:

```
checkuniquevalues(bank,bank.columns)
```

```
Column name: age int64
[56 57 37 40 45 59 41 24 25 29 35 54 46 50 39 30 55 49 34 52 58 32 38 44
 42 60 53 47 51 48 33 31 43 36 28 27 26 22 23 20 21 61 19 18 70 66 76 67
 73 88 95 77 68 75 63 80 62 65 72 82 64 71 69 78 85 79 83 81 74 17 87 91
 86 98 94 84 92 89]
---------------------------------------------
Column name: job object
['housemaid' 'services' 'admin.' 'blue-collar' 'technician' 'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']
---------------------------------------------
Column name: marital object
['married' 'single' 'divorced' 'unknown']
---------------------------------------------
Column name: education object
['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'professional.course'
 'unknown' 'university.degree' 'illiterate']
---------------------------------------------
Column name: default object
['no' 'unknown' 'yes']
---------------------------------------------
Column name: housing object
['no' 'yes' 'unknown']
---------------------------------------------
Column name: loan object
['no' 'yes' 'unknown']
---------------------------------------------
Column name: contact object
['telephone' 'cellular']
---------------------------------------------
Column name: month object
['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']
---------------------------------------------
Column name: day_of_week object
['mon' 'tue' 'wed' 'thu' 'fri']
---------------------------------------------
Column name: duration int64
[ 261  149  226 ... 1246 1556 1868]
---------------------------------------------
Column name: campaign int64
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 19 18 23 14 22 25 16 17 15 20 56
 39 35 42 28 26 27 32 21 24 29 31 30 41 37 40 33 34 43]
---------------------------------------------
Column name: pdays int64
[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]
---------------------------------------------
Column name: previous int64
[0 1 2 3 4 5 6 7]
---------------------------------------------
Column name: poutcome object
['nonexistent' 'failure' 'success']
---------------------------------------------
Column name: emp.var.rate float64
[ 1.1  1.4 -0.1 -0.2 -1.8 -2.9 -3.4 -3.  -1.7 -1.1]
---------------------------------------------
Column name: cons.price.idx float64
[93.994 94.465 93.918 93.444 93.798 93.2   92.756 92.843 93.075 92.893
 92.963 92.469 92.201 92.379 92.431 92.649 92.713 93.369 93.749 93.876
 94.055 94.215 94.027 94.199 94.601 94.767]
---------------------------------------------
Column name: cons.conf.idx float64
[-36.4 -41.8 -42.7 -36.1 -40.4 -42.  -45.9 -50.  -47.1 -46.2 -40.8 -33.6
 -31.4 -29.8 -26.9 -30.1 -33.  -34.8 -34.6 -40.  -39.8 -40.3 -38.3 -37.5
 -49.5 -50.8]
---------------------------------------------
Column name: euribor3m float64
[4.857 4.856 4.855 4.859 4.86  4.858 4.864 4.865 4.866 4.967 4.961 4.959
 4.958 4.96  4.962 4.955 4.947 4.956 4.966 4.963 4.957 4.968 4.97  4.965
 4.964 5.045 5.    4.936 4.921 4.918 4.912 4.827 4.794 4.76  4.733 4.7
 4.663 4.592 4.474 4.406 4.343 4.286 4.245 4.223 4.191 4.153 4.12  4.076
 4.021 3.901 3.879 3.853 3.816 3.743 3.669 3.563 3.488 3.428 3.329 3.282
 3.053 1.811 1.799 1.778 1.757 1.726 1.703 1.687 1.663 1.65  1.64  1.629
 1.614 1.602 1.584 1.574 1.56  1.556 1.548 1.538 1.531 1.52  1.51  1.498
 1.483 1.479 1.466 1.453 1.445 1.435 1.423 1.415 1.41  1.405 1.406 1.4
 1.392 1.384 1.372 1.365 1.354 1.344 1.334 1.327 1.313 1.299 1.291 1.281
 1.266 1.25  1.244 1.259 1.264 1.27  1.262 1.26  1.268 1.286 1.252 1.235
 1.224 1.215 1.206 1.099 1.085 1.072 1.059 1.048 1.044 1.029 1.018 1.007
 0.996 0.979 0.969 0.944 0.937 0.933 0.927 0.921 0.914 0.908 0.903 0.899
 0.884 0.883 0.881 0.879 0.873 0.869 0.861 0.859 0.854 0.851 0.849 0.843
 0.838 0.834 0.829 0.825 0.821 0.819 0.813 0.809 0.803 0.797 0.788 0.781
 0.778 0.773 0.771 0.77  0.768 0.766 0.762 0.755 0.749 0.743 0.741 0.739
 0.75  0.753 0.754 0.752 0.744 0.74  0.742 0.737 0.735 0.733 0.73  0.731
 0.728 0.724 0.722 0.72  0.719 0.716 0.715 0.714 0.718 0.721 0.717 0.712
 0.71  0.709 0.708 0.706 0.707 0.7   0.655 0.654 0.653 0.652 0.651 0.65
 0.649 0.646 0.644 0.643 0.639 0.637 0.635 0.636 0.634 0.638 0.64  0.642
 0.645 0.659 0.663 0.668 0.672 0.677 0.682 0.683 0.684 0.685 0.688 0.69
 0.692 0.695 0.697 0.699 0.701 0.702 0.704 0.711 0.713 0.723 0.727 0.729
 0.732 0.748 0.761 0.767 0.782 0.79  0.793 0.802 0.81  0.822 0.827 0.835
 0.84  0.846 0.87  0.876 0.885 0.889 0.893 0.896 0.898 0.9   0.904 0.905
 0.895 0.894 0.891 0.89  0.888 0.886 0.882 0.88  0.878 0.877 0.942 0.953
 0.956 0.959 0.965 0.972 0.977 0.982 0.985 0.987 0.993 1.    1.008 1.016
 1.025 1.032 1.037 1.043 1.045 1.047 1.05  1.049 1.046 1.041 1.04  1.039
 1.035 1.03  1.031 1.028]
---------------------------------------------
Column name: nr.employed float64
[5191.  5228.1 5195.8 5176.3 5099.1 5076.2 5017.5 5023.5 5008.7 4991.6
 4963.6]
---------------------------------------------
Column name: y object
['no' 'yes']
---------------------------------------------
```

- Observed that no inappropriate data types found in above data.

- Also no inappropriate symbols found in data.

CHECK FOR MISSING VALUES

In [13]:

```
bank.isnull().sum()
```

Out[13]:

```
age            0
job            0
marital        0
education      0
default        0
housing        0
loan           0
contact        0
month          0
day_of_week    0
duration       0
```

```
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

SPLIT COLUMNS

In [14]:
```
def splitcols(data):
    nc=data.select_dtypes(exclude='object').columns.values
    fc=data.select_dtypes(include='object').columns.values

    return(nc,fc)
```

In [15]:
```
numeric_cols,factor_cols=splitcols(bank)
```

In [16]:
```
numeric_cols
```

Out[16]:
```
array(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed'],
      dtype=object)
```

In [17]:
```
factor_cols
```

Out[17]:
```
array(['job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'poutcome', 'y'], dtype=object)
```

CHECK FOR ZEROS

In [18]:
```
bank[numeric_cols][bank[numeric_cols]==0].count()
```

Out[18]:
```
age                   0
duration              4
campaign              0
pdays                15
previous          35563
emp.var.rate          0
cons.price.idx        0
cons.conf.idx         0
euribor3m             0
nr.employed           0
dtype: int64
```

* Duration has zeros but it indicates that 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 mbenchmark purposes and should be discarded if the intention is to have a realistic predictive model.

* We will not delete the zeros or neither replace those in this column duration

* pdays column has zero values but those denote that its been 0 days until the client has been contacted so we decide not to delete or replace these zeros in this column pdays

* For previous column the zero indicates those many contacts have been performed before the campaign here more count of 0 contacts performed is been observed which is helpful data for prediction so lets keep zeros in this data as it is without deleting or replacing it

CHECK FOR SINGULARITY AND DATA IMBALANCE IN DATA

In [19]:

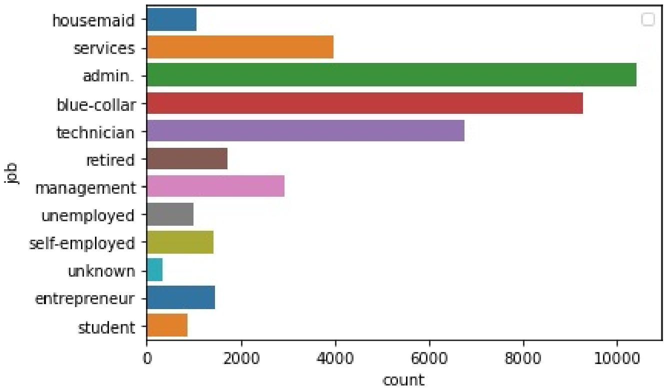
```
def checksingularity(data,factor_cols):
    for c in factor_cols:
        print("Column name :",c)
        counts = (data[c].value_counts()/len(data[c])*100)
        print("\n",counts)
        sns.countplot(y=data[c], data=bank, palette="tab10")
        plt.legend()
        plt.show()
        print("------------------------------------------------")
```

In [20]:

```
checksingularity(bank,factor_cols)
```

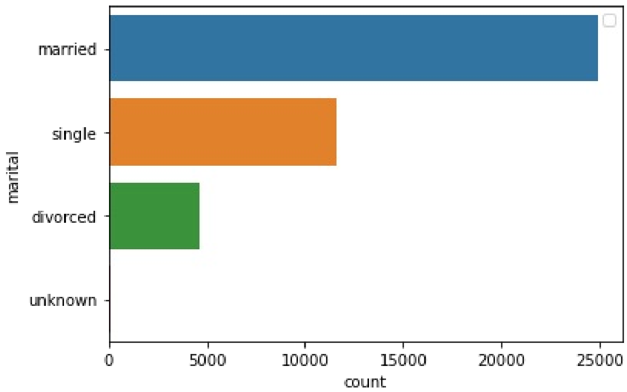
```
No handles with labels found to put in legend.
```

```
Column name : job

 admin.           25.303486
blue-collar      22.467709
technician       16.371273
services          9.636302
management        7.099155
retired           4.175974
entrepreneur      3.535010
self-employed     3.450034
housemaid         2.573565
unemployed        2.461882
student           2.124405
unknown           0.801204
Name: job, dtype: float64
```



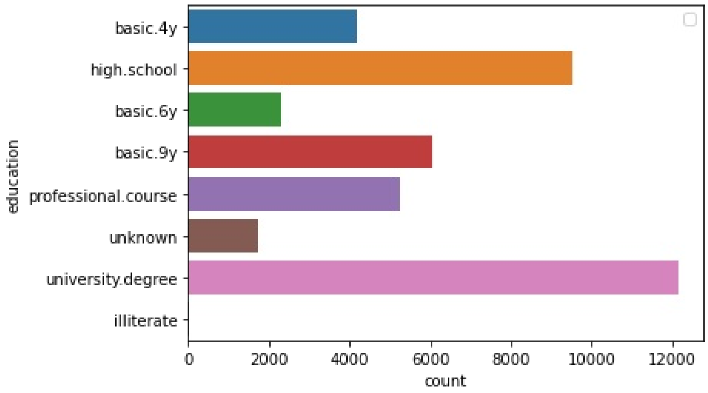

```
No handles with labels found to put in legend.
```

```
------------------------------------------------
Column name : marital

 married     60.522482
single      28.085850
divorced    11.197436
unknown      0.194231
Name: marital, dtype: float64
```

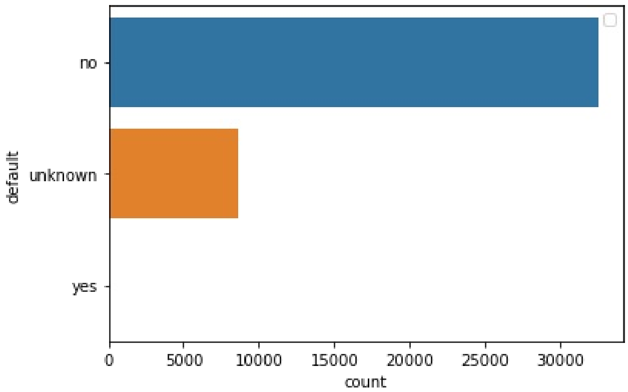



```
No handles with labels found to put in legend.
```

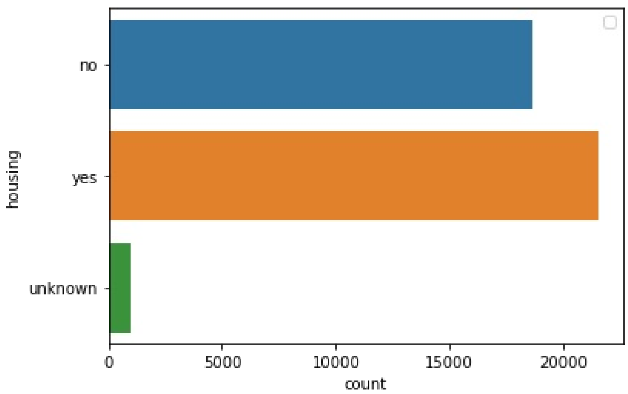
```
-----------------------------------------------
Column name : education

 university.degree      29.542585
high.school            23.101389
basic.9y               14.676605
professional.course    12.729436
basic.4y               10.138875
basic.6y                5.564728
unknown                 4.202680
illiterate              0.043702
Name: education, dtype: float64
```




```
No handles with labels found to put in legend.
-----------------------------------------------
Column name : default

 no         79.120132
unknown    20.872584
yes         0.007284
Name: default, dtype: float64
```



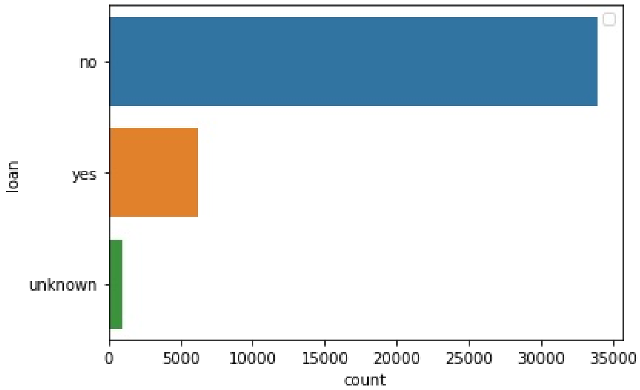

```
No handles with labels found to put in legend.

-----------------------------------------------
Column name : housing

 yes        52.384190
no         45.212198
unknown     2.403613
Name: housing, dtype: float64
```




```
-----------------------------------------------
```

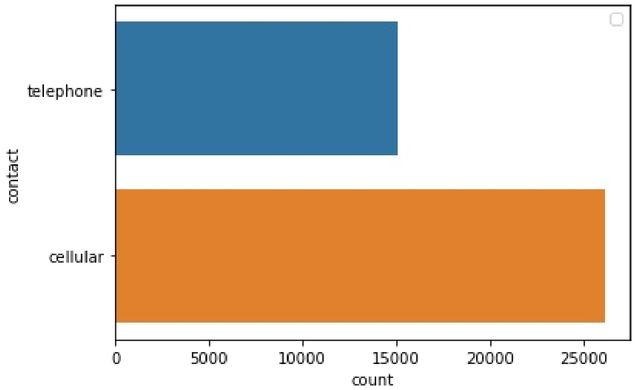
```
Column name : loan

 no         82.426920
yes        15.169467
unknown     2.403613
Name: loan, dtype: float64
```

```
No handles with labels found to put in legend.
```



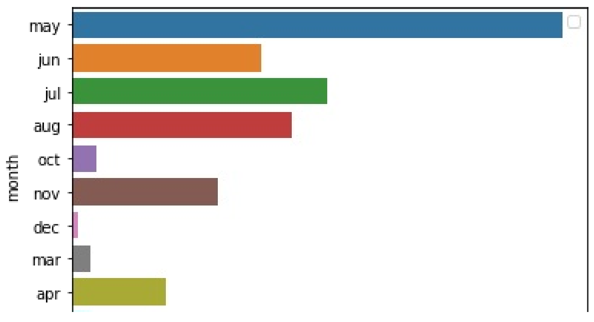

```
No handles with labels found to put in legend.
```

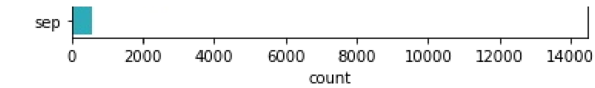
```
------------------------------------------------
Column name : contact

 cellular     63.474798
telephone    36.525202
Name: contact, dtype: float64
```




```
No handles with labels found to put in legend.
```

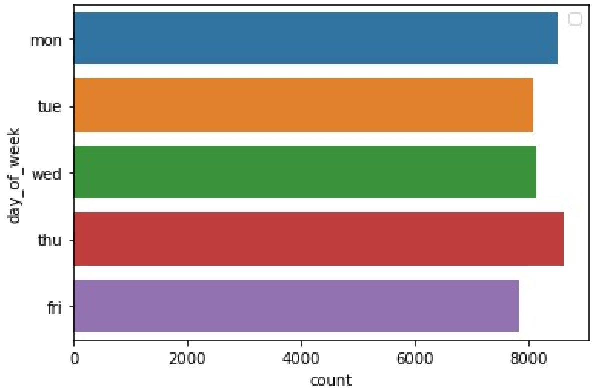
```
------------------------------------------------
Column name : month

 may    33.429640
jul    17.417694
aug    14.999514
jun    12.911528
nov     9.956784
apr     6.390211
oct     1.743226
sep     1.383898
mar     1.325629
dec     0.441876
Name: month, dtype: float64
```





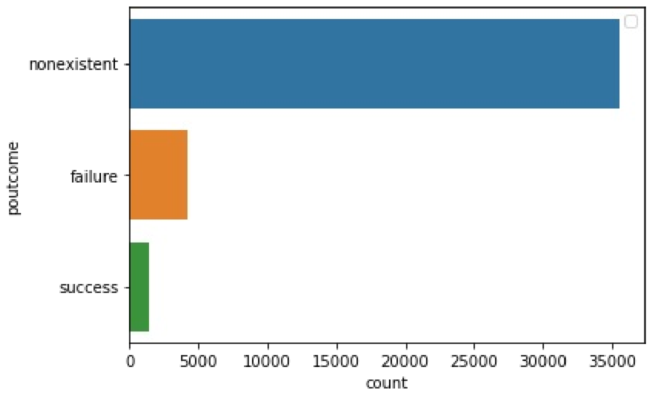

```
No handles with labels found to put in legend.
```

```
-------------------------------------------------
Column name : day_of_week

 thu    20.935709
mon    20.671069
wed    19.748470
tue    19.641643
fri    19.003108
Name: day_of_week, dtype: float64
```



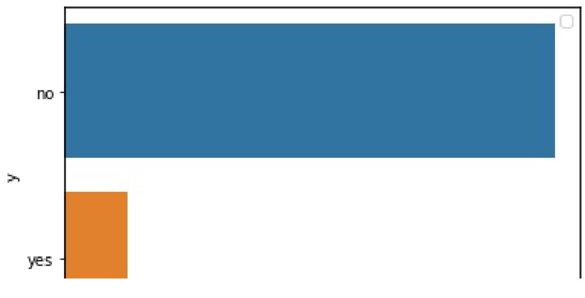

```
No handles with labels found to put in legend.
```

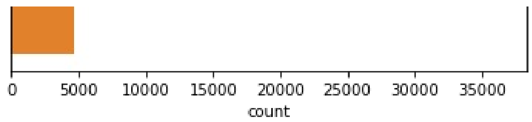
```
-------------------------------------------------
Column name : poutcome

 nonexistent    86.343110
failure        10.323395
success         3.333495
Name: poutcome, dtype: float64
```




```
No handles with labels found to put in legend.
```

```
-------------------------------------------------
Column name : y

 no     88.734583
yes    11.265417
Name: y, dtype: float64
```





----

* Looking to the percentage of contribution of every attribute to the column there are multiple atrributes that need to be deleted and has no importance in the overall data

* Also target column data imbalance is observed as the number of clients not subscribing to term deposit are more in number than those subscribing.

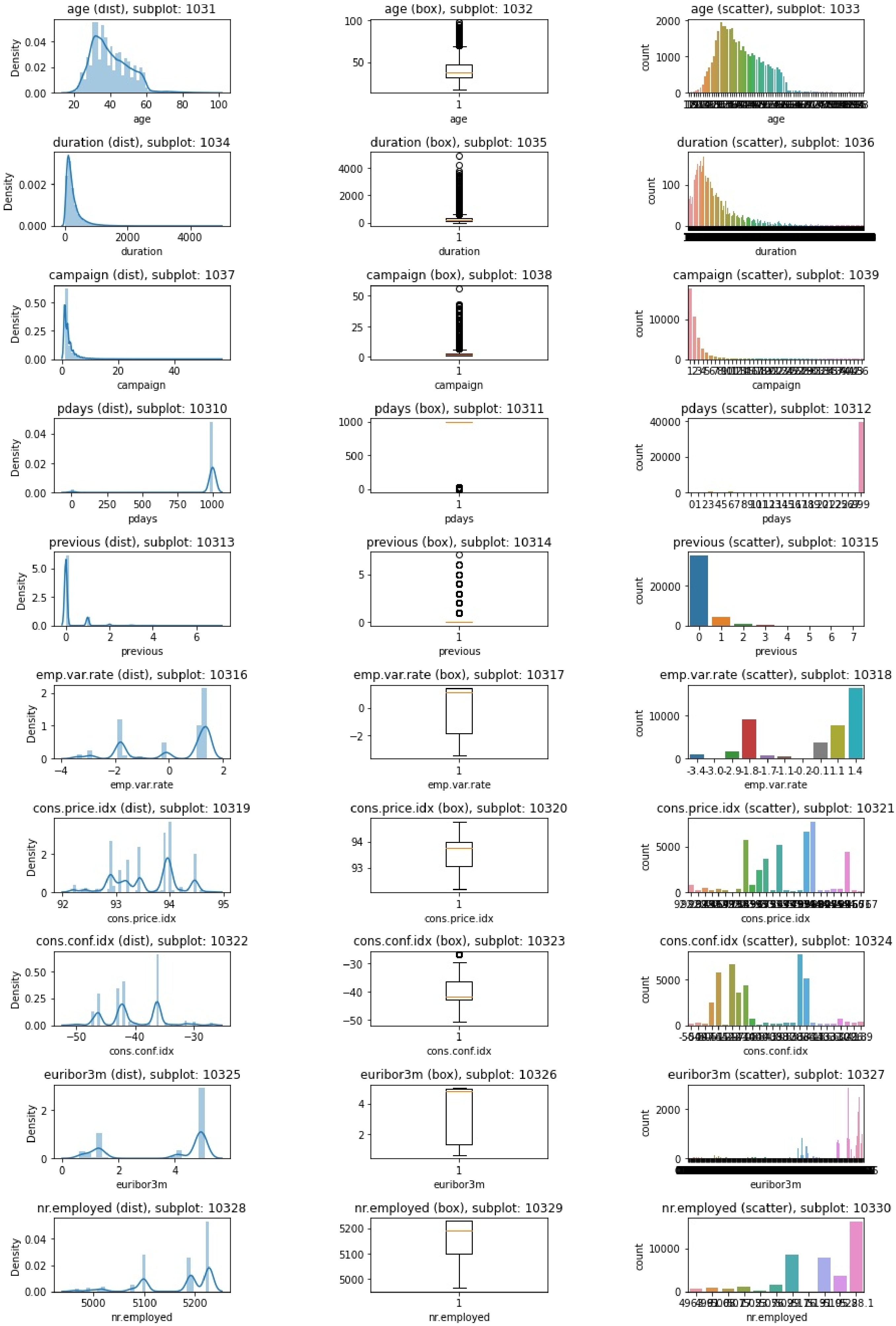
EXPLORATORY DATA ANALYSIS

LETS ANALYSE NUMERIC DATA

In [21]:
```
len(numeric_cols)
```

Out[21]: 10

In [22]:
```
#  Numerical Data
features_num = numeric_cols

#  plot Numerical Data
a = 10  # number of rows
b = 3  # number of columns
c = 1  # initialize plot counter

fig = plt.figure(figsize=(14,22))

for i in features_num:
    plt.subplot(a, b, c)
    plt.title('{} (dist), subplot: {}{}{}'.format(i, a, b, c))
    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0.8,
                        hspace=0.8)
    plt.xlabel(i)
    sns.distplot(bank[i])
    c = c + 1

    plt.subplot(a, b, c)
    plt.title('{} (box), subplot: {}{}{}'.format(i, a, b, c))
    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0.8,
                        hspace=0.8)
    plt.xlabel(i)
    plt.boxplot(x = bank[i])
    c = c + 1

    plt.subplot(a, b, c)
    plt.title('{} (scatter), subplot: {}{}{}'.format(i, a, b, c))
    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0.8,
                        hspace=0.8)
    plt.xlabel(i)
    sns.countplot(data = bank, x = bank[i])
    c = c + 1

plt.show()
```



- Few outliers and skewness observed in above columns

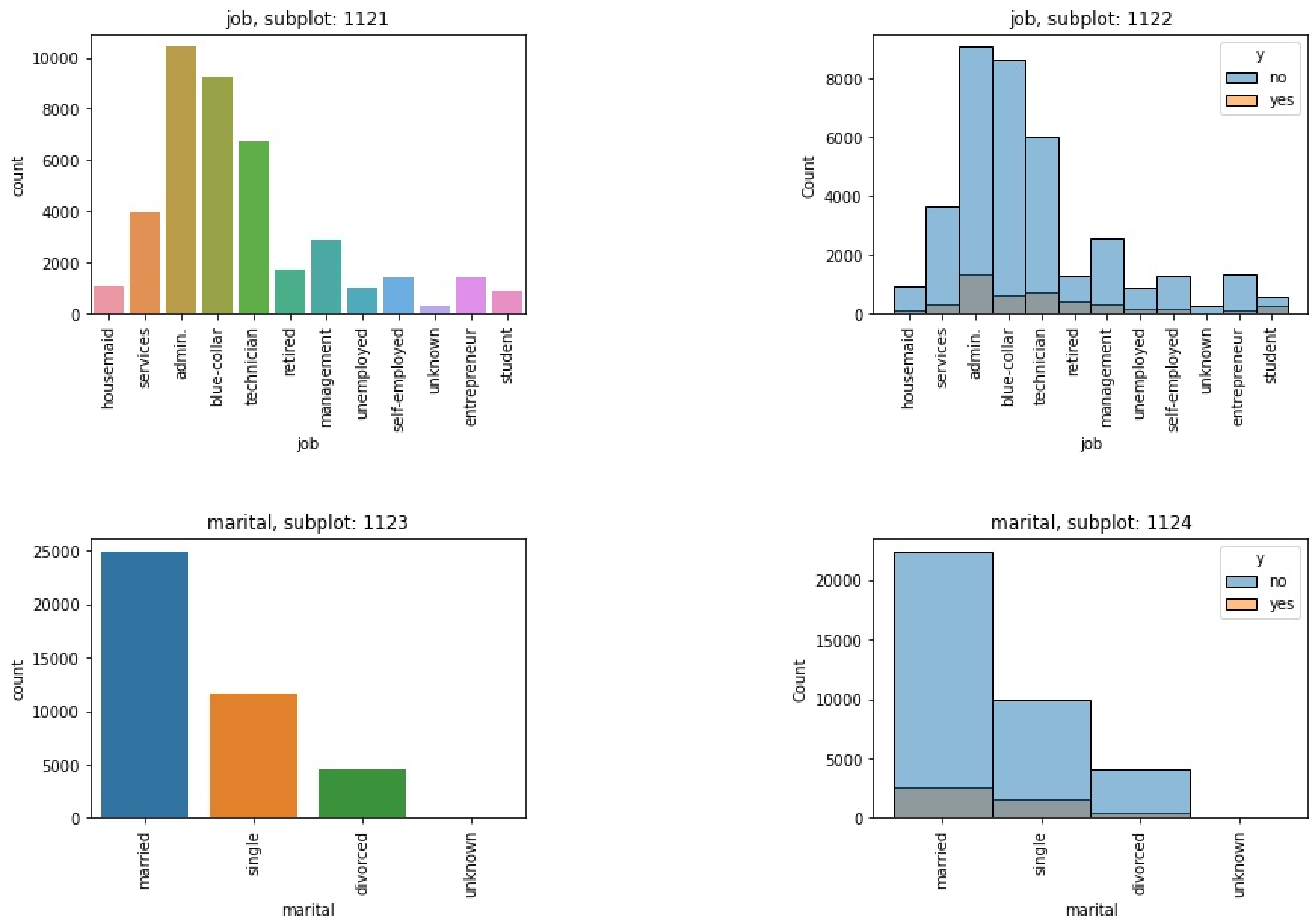
LETS ANALYSE CATEGORICAL DATA

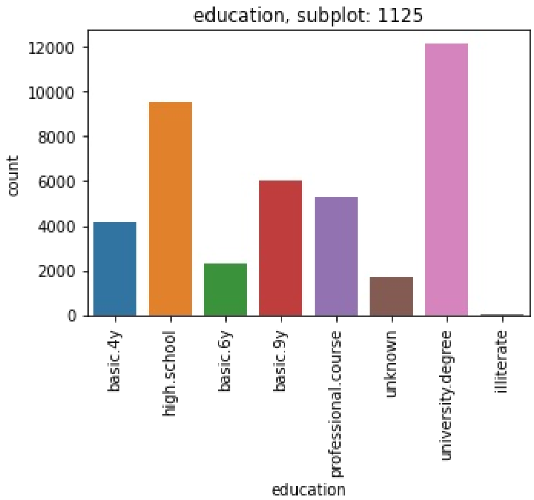
LETS ANALYSE CATEGORICAL DATA

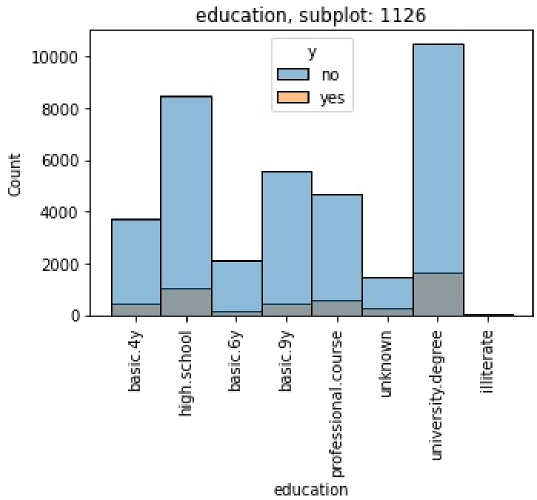
In [23]:
```
len(factor_cols)
```

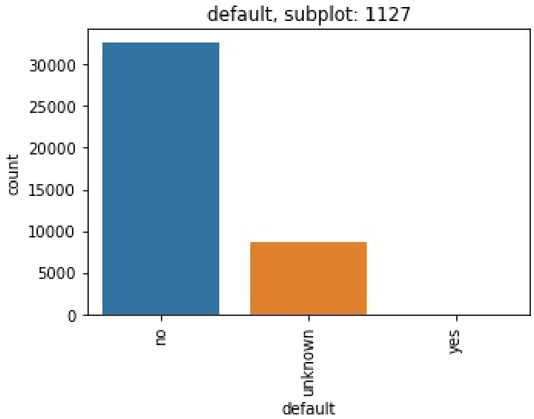
Out[23]: 11

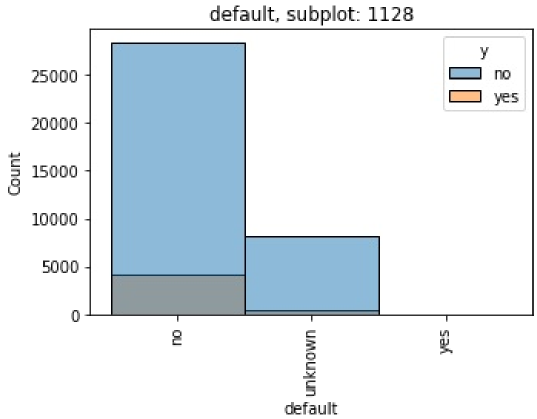
In [24]:
```
#  Categorical Data
features_cat = factor_cols

#  Categorical Data
a = 11  # number of rows
b = 2  # number of columns
c = 1  # initialize plot counter

fig = plt.figure(figsize=(14,10))

for i in features_cat:
    plt.subplot(a, b, c)
    plt.title('{}, subplot: {}{}{}'.format(i, a, b, c))
    plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=5,
                    wspace=0.8,
                    hspace=0.8)
    plt.xticks(rotation=90,size=10)
    plt.xlabel(i)
    sns.countplot(bank[i])
    c = c + 1

    plt.subplot(a, b, c)
    plt.title('{}, subplot: {}{}{}'.format(i, a, b, c))
    plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=5,
                    wspace=0.8,
                    hspace=0.8)
    plt.xticks(rotation=90,size=10)
    plt.xlabel(i)
    sns.histplot(x=bank[i],hue=bank['y'])
    c = c + 1

plt.show()
```

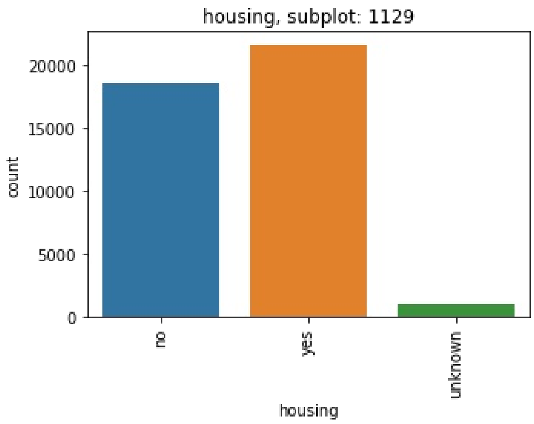


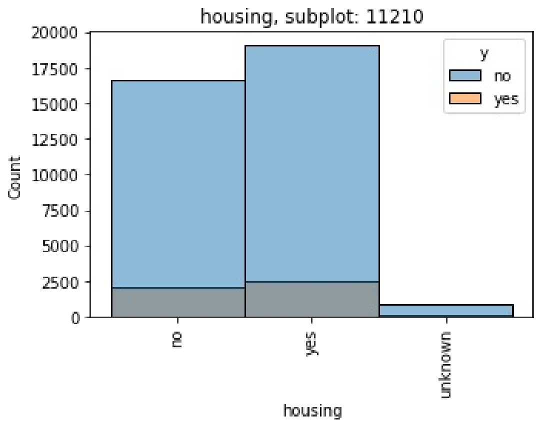

education, subplot: 1125
count
0
2000
4000
6000
8000
10000
12000
basic.4y
high.school
basic.6y
basic.9y
professional.course
unknown
university.degree
illiterate
education

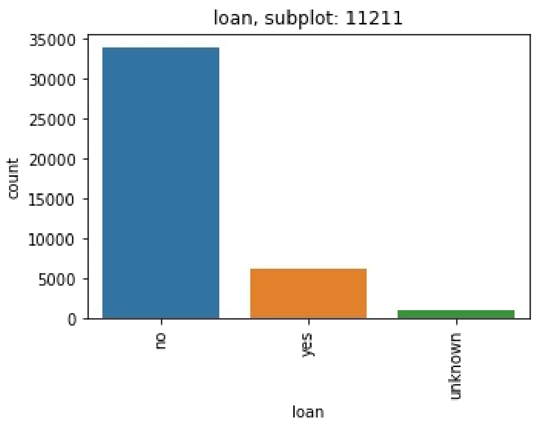

education, subplot: 1126
y
no
yes
Count
0
2000
4000
6000
8000
10000
basic.4y
high.school
basic.6y
basic.9y
professional.course
unknown
university.degree
illiterate
education

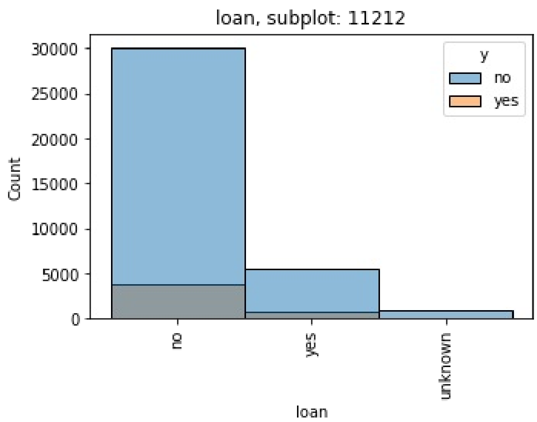

default, subplot: 1127
count
0
5000
10000
15000
20000
25000
30000
no
unknown
yes
default

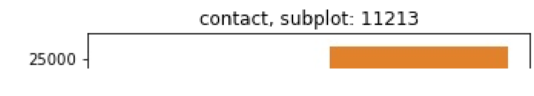

default, subplot: 1128
y
no
yes
Count
0
5000
10000
15000
20000
25000
no
unknown
yes
default


housing, subplot: 1129
count
0
5000
10000
15000
20000
no
yes
unknown
housing

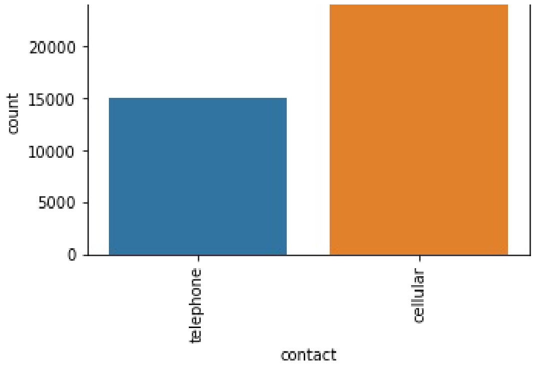

housing, subplot: 11210
y
no
yes
Count
0
2500
5000
7500
10000
12500
15000
17500
20000
no
yes
unknown
housing

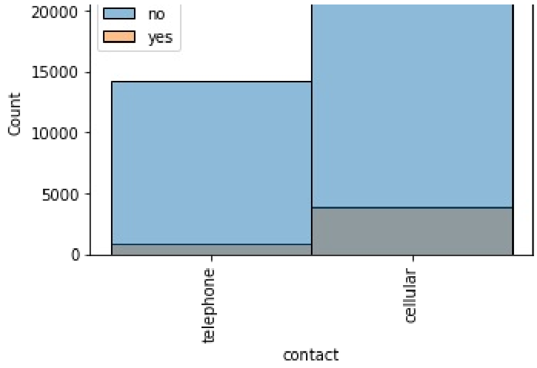

loan, subplot: 11211
count
0
5000
10000
15000
20000
25000
30000
35000
no
yes
unknown
loan

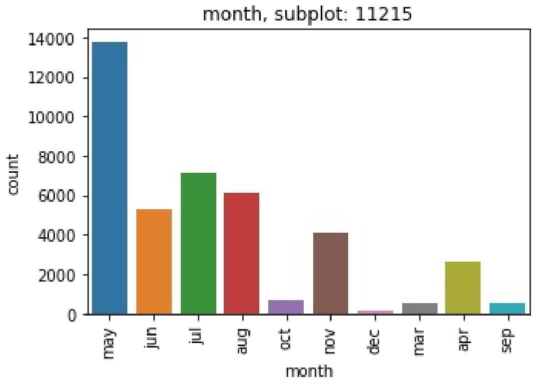

loan, subplot: 11212
y
no
yes
Count
0
5000
10000
15000
20000
25000
30000
no
yes
unknown
loan

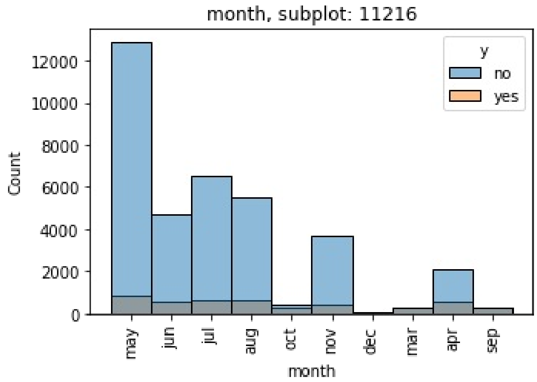

contact, subplot: 11213
25000

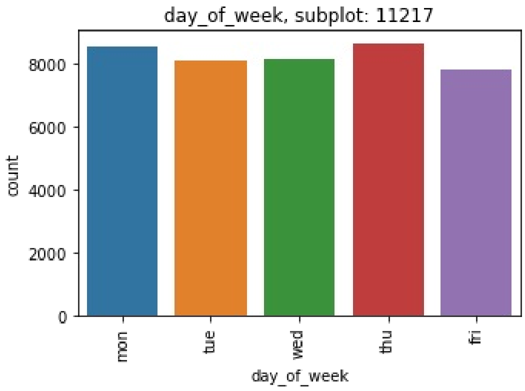

contact, subplot: 11214
y

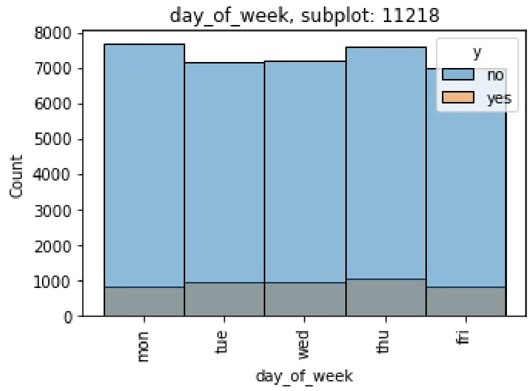

count
telephone
cellular
contact

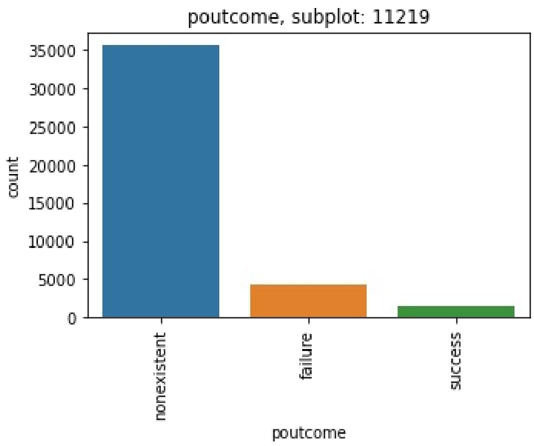

no
yes
Count
telephone
cellular
contact

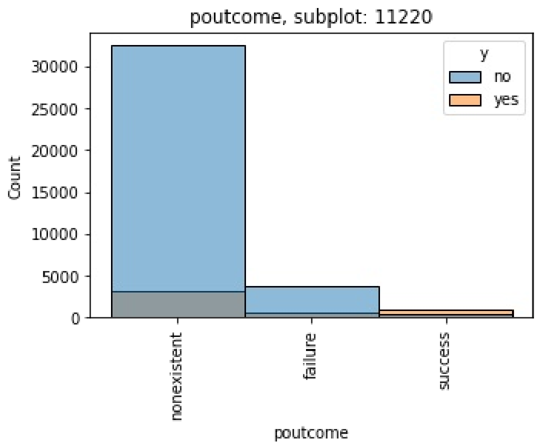

month, subplot: 11215
count
may
jun
jul
aug
oct
nov
dec
mar
apr
sep
month

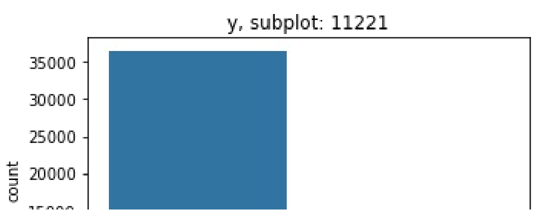

month, subplot: 11216
y
no
yes
Count
may
jun
jul
aug
oct
nov
dec
mar
apr
sep
month

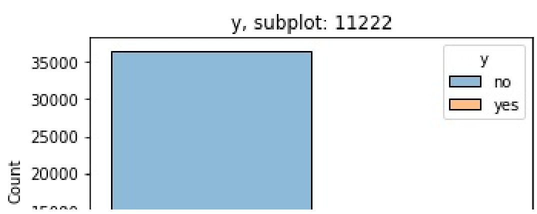

day_of_week, subplot: 11217
count
mon
tue
wed
thu
fri
day_of_week

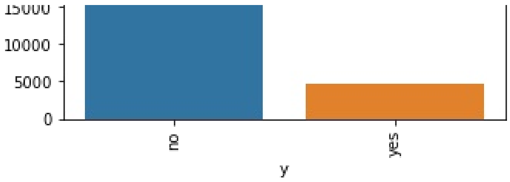

day_of_week, subplot: 11218
y
no
yes
Count
mon
tue
wed
thu
fri
day_of_week

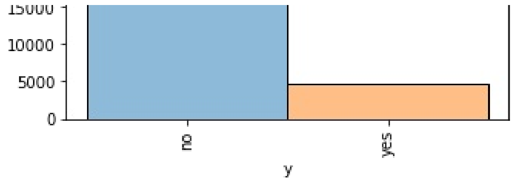

poutcome, subplot: 11219
count
nonexistent
failure
success
poutcome


poutcome, subplot: 11220
y
no
yes
Count
nonexistent
failure
success
poutcome


y, subplot: 11221
count

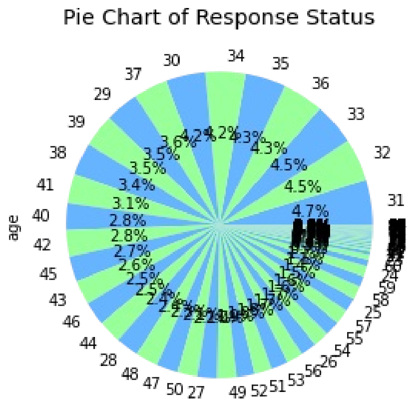

y, subplot: 11222
y
no
yes
Count

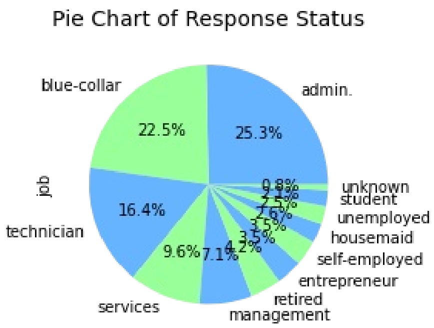


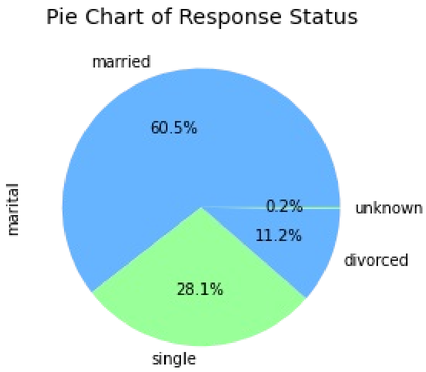


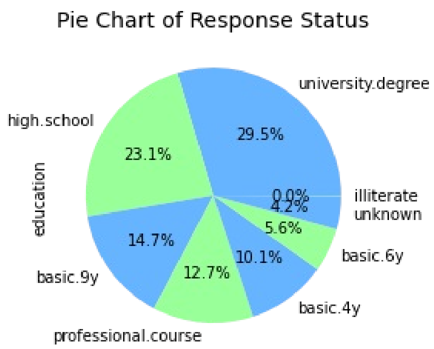
In [25]:

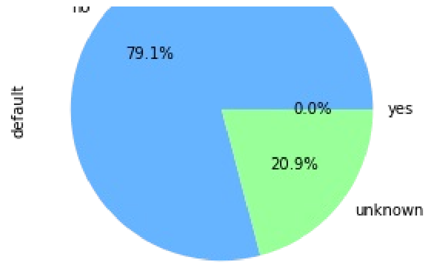
```python
for c in bank.columns:
    plt.figure(figsize=(4,4))
    bank[c].value_counts().plot.pie(autopct='%1.1f%%', colors = ['#66b3ff','#99ff99'])
    plt.title("Pie Chart of Response Status", fontdict={'fontsize': 14})
    plt.tight_layout()
```

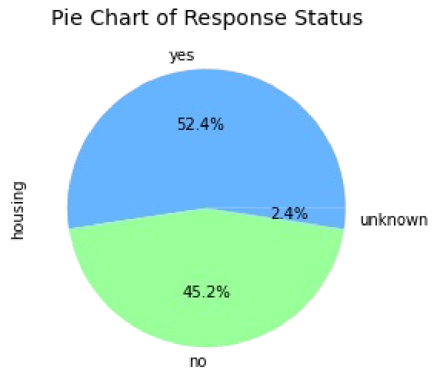


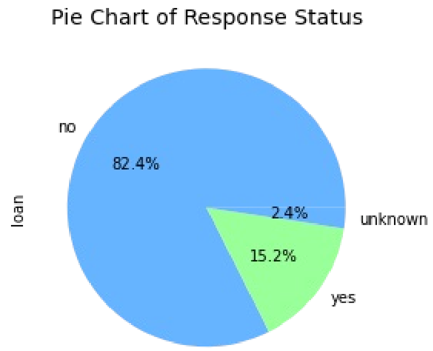


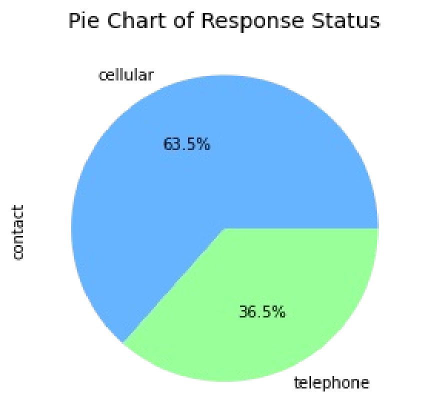


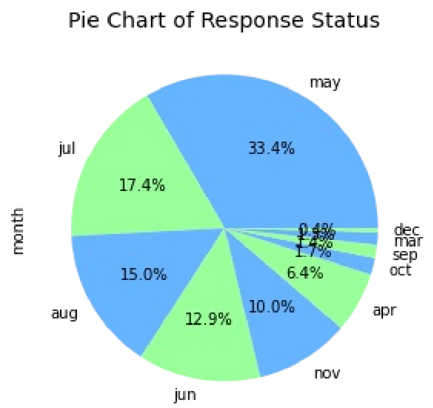


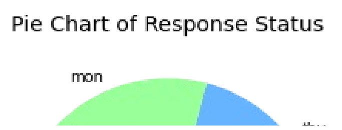

default
79.1%
0.0%
yes
20.9%
unknown

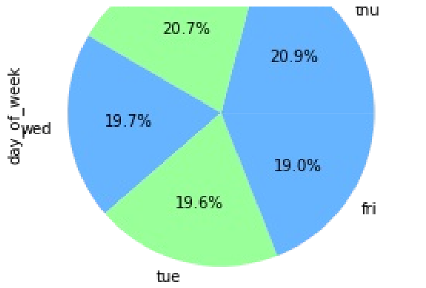

Pie Chart of Response Status
yes
52.4%
housing
2.4%
unknown
45.2%
no


Pie Chart of Response Status
no
82.4%
loan
2.4%
unknown
15.2%
yes

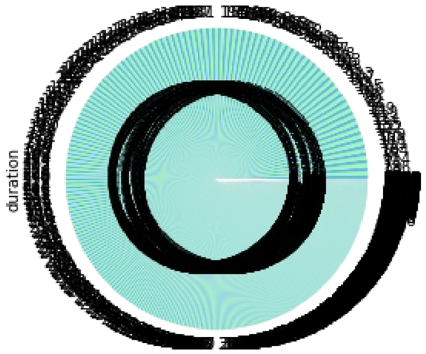

Pie Chart of Response Status
cellular
63.5%
contact
36.5%
telephone


Pie Chart of Response Status
may
33.4%
jul
17.4%
month
dec
mar
sep
1.7%
oct
6.4%
15.0%
aug
12.9%
10.0%
apr
nov
jun

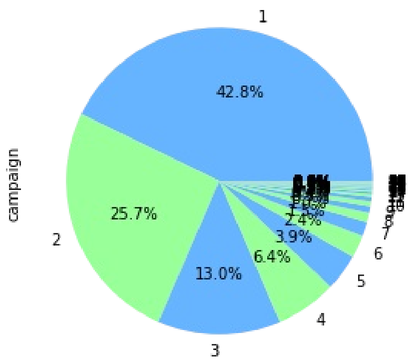

Pie Chart of Response Status
mon



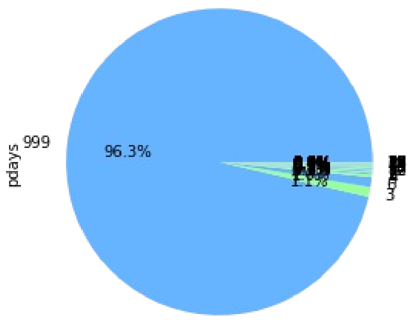
Pie Chart of Response Status



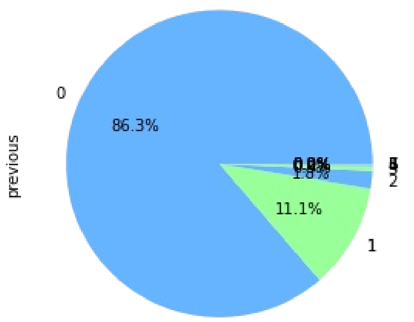
Pie Chart of Response Status



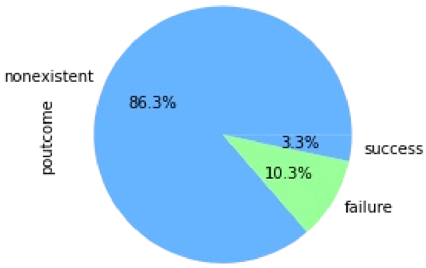
Pie Chart of Response Status



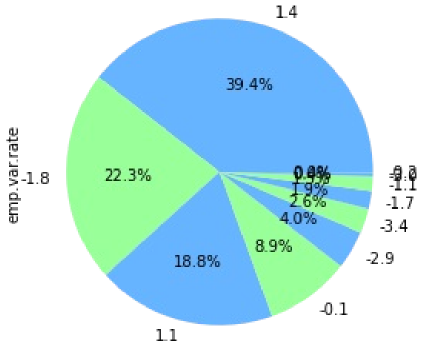
Pie Chart of Response Status



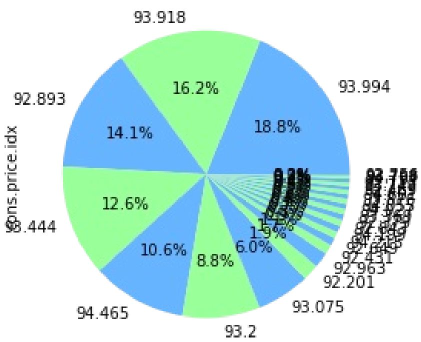
Pie Chart of Response Status



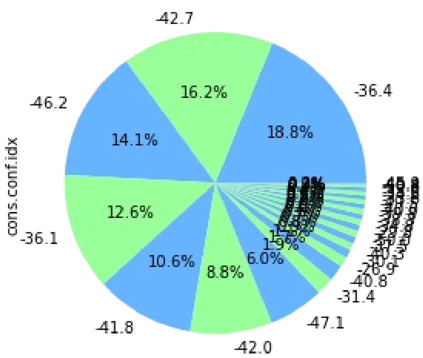
Pie Chart of Response Status



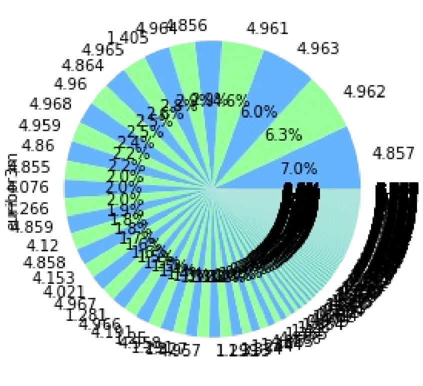
Pie Chart of Response Status



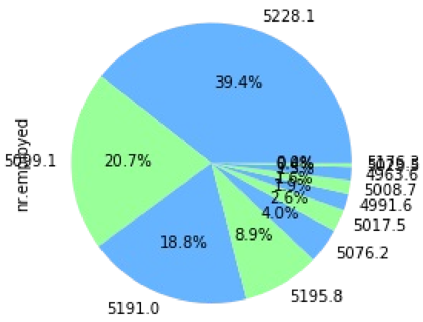
Pie Chart of Response Status

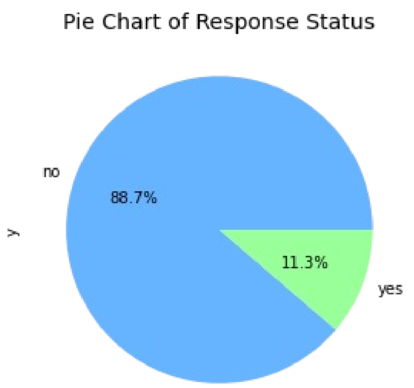


Pie Chart of Response Status



Pie Chart of Response Status





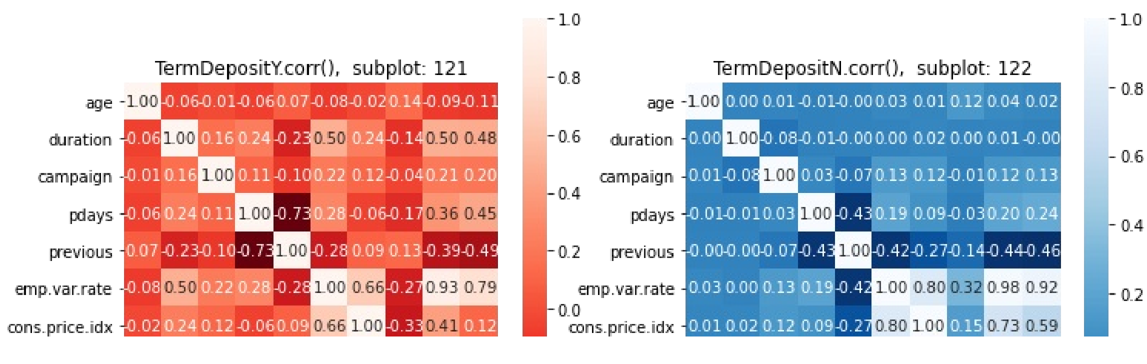
- Maximum people subscribe to term deposit who have admin job, who are married,educated from university college,who are non defaulters,who have house ,who have not taken loan.¶

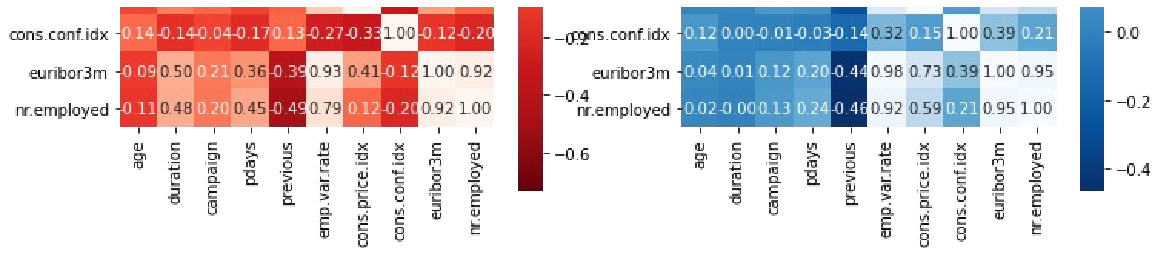
- Also subscribers are from the people who are contacted via cells,in month of may,mostly on thursday with nonexistant poutcome

CHECK MULTICOLINARITY

In [26]:

```
# select few attributes
df_bank = bank[['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed','y']]

# correlation - target yes
dfTermDepositY = df_bank[(df_bank['y'] == 'yes')]
dfTermDepositYCorr = dfTermDepositY.drop(["y"], axis=1).corr()
# correlation - target no
dfTermDepositN   = df_bank[(df_bank['y'] == 'no')]
dfTermDepositNCorr = dfTermDepositN.drop(["y"], axis=1).corr()


#  SUBPLOTS
fig = plt.figure(figsize=(12,6))

plt.subplot(121)   #  subplot 1 - target yes
plt.title('TermDepositY.corr(),  subplot: 121')
sns.heatmap(dfTermDepositYCorr, annot=True, fmt='.2f', square=True, cmap = 'Reds_r')

plt.subplot(122)   #  subplot 2 - target no
plt.title('TermDepositN.corr(),  subplot: 122')
sns.heatmap(dfTermDepositNCorr, annot=True, fmt='.2f', square=True, cmap = 'Blues_r')

plt.show()
```





In [27]:

```
plt.figure(figsize=(10,5))
sns.heatmap(bank.corr(),annot = True,cmap='RdYlGn')
```

Out[27]: <AxesSubplot:>

* Highly correlated columns are emp.var.rate and cons.price.idx , euribor3m and emp.var.rate, nr.employed and emp.var.rate, nr.employed and euribor3m, cons.price.idx and euribor3m

* We can say overall columns haing multicolinearity to be removed are euribor3m,nr.employed,cons.price.idx

In [28]:

```
cor_matrix = bank.corr().abs()
print(cor_matrix)
```

```
                     age  duration  campaign     pdays  previous  \
age             1.000000  0.000866  0.004594  0.034369  0.024365
duration        0.000866  1.000000  0.071699  0.047577  0.020640
campaign        0.004594  0.071699  1.000000  0.052584  0.079141
pdays           0.034369  0.047577  0.052584  1.000000  0.587514
previous        0.024365  0.020640  0.079141  0.587514  1.000000
emp.var.rate    0.000371  0.027968  0.150754  0.271004  0.420489
cons.price.idx  0.000857  0.005312  0.127836  0.078889  0.203130
cons.conf.idx   0.129372  0.008173  0.013733  0.091342  0.050936
euribor3m       0.010767  0.032897  0.135133  0.296899  0.454494
nr.employed     0.017725  0.044703  0.144095  0.372605  0.501333

                emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
age                 0.000371        0.000857       0.129372   0.010767
duration            0.027968        0.005312       0.008173   0.032897
campaign            0.150754        0.127836       0.013733   0.135133
pdays               0.271004        0.078889       0.091342   0.296899
previous            0.420489        0.203130       0.050936   0.454494
emp.var.rate        1.000000        0.775334       0.196041   0.972245
cons.price.idx      0.775334        1.000000       0.058986   0.688230
cons.conf.idx       0.196041        0.058986       1.000000   0.277686
euribor3m           0.972245        0.688230       0.277686   1.000000
nr.employed         0.906970        0.522034       0.100513   0.945154

                nr.employed
age                0.017725
duration           0.044703
campaign           0.144095
pdays              0.372605
previous           0.501333
```

```
emp.var.rate       0.906970
cons.price.idx     0.522034
cons.conf.idx      0.100513
euribor3m          0.945154
nr.employed        1.000000
```

In [29]:
```
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(np.bool))
print(upper_tri)
```

```
                age  duration  campaign     pdays  previous  emp.var.rate  \
age             NaN  0.000866  0.004594  0.034369  0.024365      0.000371
duration        NaN       NaN  0.071699  0.047577  0.020640      0.027968
campaign        NaN       NaN       NaN  0.052584  0.079141      0.150754
pdays           NaN       NaN       NaN       NaN  0.587514      0.271004
previous        NaN       NaN       NaN       NaN       NaN      0.420489
emp.var.rate    NaN       NaN       NaN       NaN       NaN           NaN
cons.price.idx  NaN       NaN       NaN       NaN       NaN           NaN
cons.conf.idx   NaN       NaN       NaN       NaN       NaN           NaN
euribor3m       NaN       NaN       NaN       NaN       NaN           NaN
nr.employed     NaN       NaN       NaN       NaN       NaN           NaN

                cons.price.idx  cons.conf.idx  euribor3m  nr.employed
age                   0.000857       0.129372   0.010767     0.017725
duration              0.005312       0.008173   0.032897     0.044703
campaign              0.127836       0.013733   0.135133     0.144095
pdays                 0.078889       0.091342   0.296899     0.372605
previous              0.203130       0.050936   0.454494     0.501333
emp.var.rate          0.775334       0.196041   0.972245     0.906970
cons.price.idx             NaN       0.058986   0.688230     0.522034
cons.conf.idx              NaN            NaN   0.277686     0.100513
euribor3m                  NaN            NaN        NaN     0.945154
nr.employed                NaN            NaN        NaN          NaN
```

In [30]:
```
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.60)]
print(to_drop)
```

```
['cons.price.idx', 'euribor3m', 'nr.employed']
```

DROPPED COLUMNS WITH MULTICOLINEARITY DATASET

In [31]:
```
bank.drop(columns=['nr.employed','euribor3m','cons.price.idx'],inplace=True)
```

HANDLING OUTLIERS

In [32]:
```
def removeoutliers(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    LL=Q1-1.5*(IQR)

    UL=Q3+1.5*(IQR)

    data = data[~((data < (Q1 - 1.5 * IQR)) |(data > (Q3 + 1.5 * IQR))).any(axis=1)]
    return data
```

In [33]:
```
bank=removeoutliers(bank)
```

- REMOVED ALL THE OUTLIERS IN DATA

In [34]:
```
bank
```

Out[34]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previo
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	

2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999
...	...	...	...	...	...	...	...	...	...	...	...	...	...
41180	36	admin.	married	university.degree	no	no	no	cellular	nov	fri	254	2	999
41181	37	admin.	married	university.degree	no	yes	no	cellular	nov	fri	281	1	999
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	383	1	999
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	189	2	999
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	442	1	999

30360 rows × 18 columns

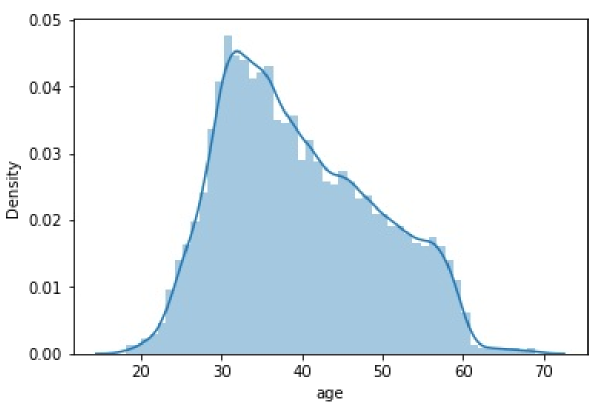
AFTER DROPPING COLUMNS AGAIN SPLIT THE DATA

In [35]:
```
numeric_cols,factor_cols=splitcols(bank)
```

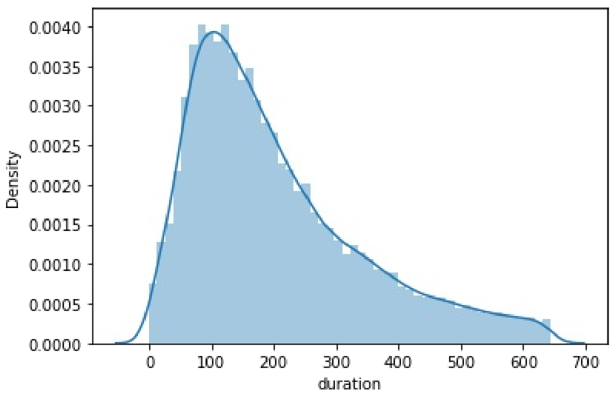
HANDLING SKEWNESS IN DATA

In [36]:
```
for col in numeric_cols:
    print(col)
    print(bank[col].skew())
    plt.figure()
    sns.distplot(bank[col])
    plt.show()
```

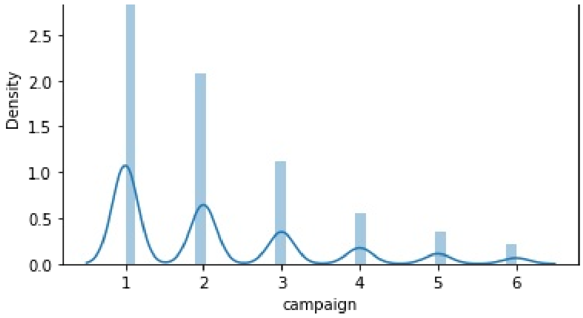
```
age
0.40393336854725415
```



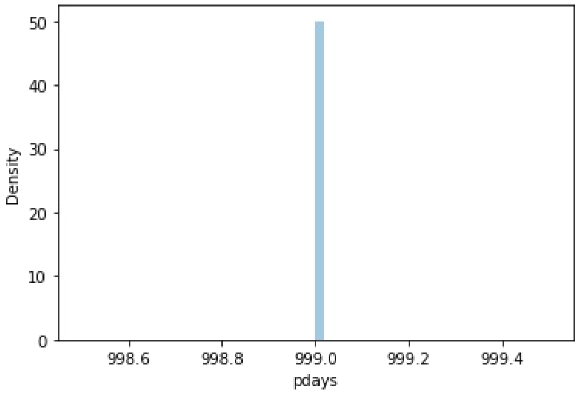

```
duration
1.0504633333893845
```



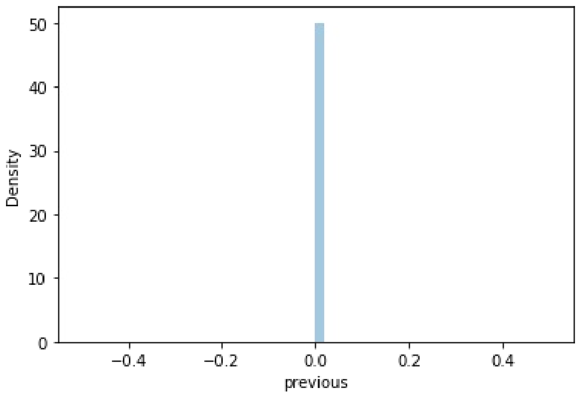

```
campaign
1.2494328717767667
```



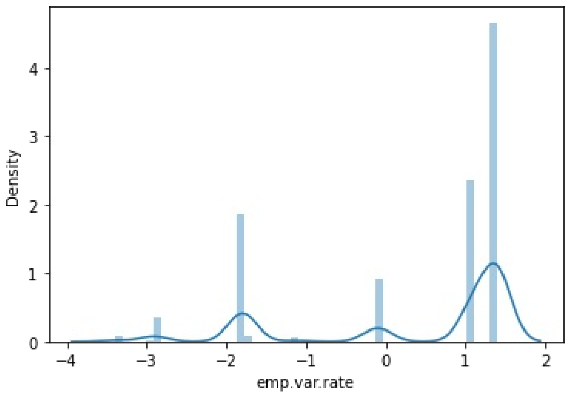

```
pdays
0
```



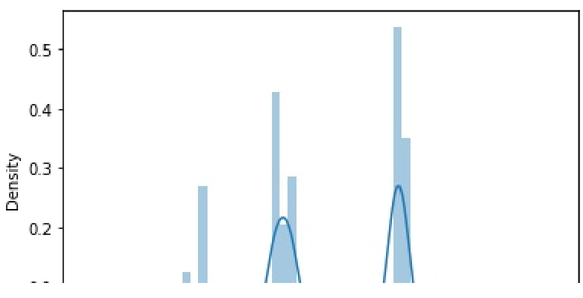

```
previous
0
```

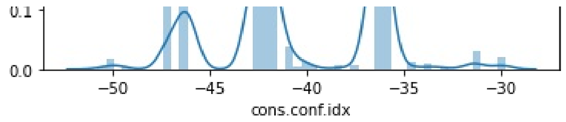



```
emp.var.rate
-1.1131460454175437
```




```
cons.conf.idx
-0.019265689273849376
```





In [37]:
```
bank[numeric_cols].skew()
```

Out[37]:
```
age              0.403933
duration         1.050463
campaign         1.249433
pdays            0.000000
previous         0.000000
emp.var.rate    -1.113146
cons.conf.idx   -0.019266
dtype: float64
```

In [38]:
```
skew_cols = bank[numeric_cols].apply(lambda x: skew(x)).sort_values(ascending=False)
skew_cols = abs(skew_cols)
high_skew = skew_cols[skew_cols > 0.5]
print(high_skew)
skew_index = high_skew.index
# Normalize skewed feature
#bank[skew_index] = np.log1p(bank[skew_index])
```

```
campaign        1.249371
duration        1.050411
emp.var.rate    1.113091
dtype: float64
```

In [39]:
```
#bank[numeric_cols].skew()
```

- SUCCESFULLY REMOVED SKEWNESS IN DATA

In [40]:
```
bank
```

Out[40]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previou
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41180	36	admin.	married	university.degree	no	no	no	cellular	nov	fri	254	2	999	
41181	37	admin.	married	university.degree	no	yes	no	cellular	nov	fri	281	1	999	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	383	1	999	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	189	2	999	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	442	1	999	

30360 rows × 18 columns

CONVERTING DATA TO DUMMY VARIABLES (ONE HOT ENCODING) AND LABEL ENCODING

LABEL ENCODING ON TARGET AND EDUCATION COLUMN

In [41]:
```
label_encoder = preprocessing.LabelEncoder()
```

```
# Encode labels in column 'species'.
bank['y']= label_encoder.fit_transform(bank['y'])

bank['y'].unique()

bank['education']= label_encoder.fit_transform(bank['education'])

bank['education'].unique()
```

Out[41]:
```
array([0, 3, 1, 2, 5, 7, 6, 4])
```

In [42]:
```
bank
```

Out[42]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	pou
0	56	housemaid	married	0	no	no	no	telephone	may	mon	261	1	999	0	none
1	57	services	married	3	unknown	no	no	telephone	may	mon	149	1	999	0	none
2	37	services	married	3	no	yes	no	telephone	may	mon	226	1	999	0	none
3	40	admin.	married	1	no	no	no	telephone	may	mon	151	1	999	0	none
4	56	services	married	3	no	no	yes	telephone	may	mon	307	1	999	0	none
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41180	36	admin.	married	6	no	no	no	cellular	nov	fri	254	2	999	0	none
41181	37	admin.	married	6	no	yes	no	cellular	nov	fri	281	1	999	0	none
41184	46	blue-collar	married	5	no	no	no	cellular	nov	fri	383	1	999	0	none
41185	56	retired	married	6	no	yes	no	cellular	nov	fri	189	2	999	0	none
41186	44	technician	married	5	no	no	no	cellular	nov	fri	442	1	999	0	none

30360 rows × 18 columns

* Here label encoding is applied for these columns for assigning numerical values according to specific order to the data.

DUMMIES (ONE HOT ENCODING) ON REST ALL COLUMNS

In [43]:
```
bank_dummies_cat=pd.get_dummies(bank[factor_cols].drop('y',1),drop_first=True)
```

In [44]:
```
bank_dummies_cat.head()
```

Out[44]:

	education	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student	job_technician	...
0	0	0	0	1	0	0	0	0	0	0	...
1	3	0	0	0	0	0	0	1	0	0	...
2	3	0	0	0	0	0	0	1	0	0	...
3	1	0	0	0	0	0	0	0	0	0	...
4	3	0	0	0	0	0	0	1	0	0	...

5 rows × 35 columns

In [45]:
```
bank_data = pd.concat([bank[numeric_cols],bank_dummies_cat,bank['y']],axis =1)
bank_data.head()
```

Out[45]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.conf.idx	education	job_blue-collar	job_entrepreneur	...	month_mar	month_may
0	56	261	1	999	0	1.1	-36.4	0	0	0	...	0	1
1	57	149	1	999	0	1.1	-36.4	3	0	0	...	0	1
2	37	226	1	999	0	1.1	-36.4	3	0	0	...	0	1
3	40	151	1	999	0	1.1	-36.4	1	0	0	...	0	1
4	56	307	1	999	0	1.1	-36.4	3	0	0	...	0	1

5 rows × 43 columns

In [46]:
```
bank_data.columns
```

Out[46]:
```
Index(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.conf.idx', 'education', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired', 'job_self-employed',
       'job_services', 'job_student', 'job_technician', 'job_unemployed',
       'job_unknown', 'marital_married', 'marital_single', 'marital_unknown',
       'default_unknown', 'default_yes', 'housing_unknown', 'housing_yes',
       'loan_unknown', 'loan_yes', 'contact_telephone', 'month_aug',
       'month_dec', 'month_jul', 'month_jun', 'month_mar', 'month_may',
       'month_nov', 'month_oct', 'month_sep', 'day_of_week_mon',
       'day_of_week_thu', 'day_of_week_tue', 'day_of_week_wed', 'y'],
      dtype='object')
```

TAKE BACKUP OF ORIGINAL DATA

In [47]:
```
data = bank_data.copy()
```

In [48]:
```
data = data.sample(frac=1)
```

In [49]:
```
data.reset_index(drop=True,inplace = True)
```

In [50]:
```
data.dropna(inplace=True)
```

In [51]:
```
data
```

Out[51]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.conf.idx	education	job_blue-collar	job_entrepreneur	...	month_mar	month_m
0	33	200	1	999	0	-0.1	-42.0	6	0	0	...	0	
1	43	190	2	999	0	1.4	-41.8	2	1	0	...	0	
2	46	282	1	999	0	-2.9	-33.6	3	0	0	...	0	
3	51	133	5	999	0	1.4	-41.8	3	0	0	...	0	
4	24	242	1	999	0	1.4	-42.7	3	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
30355	52	114	6	999	0	1.4	-36.1	3	0	0	...	0	
30356	34	133	2	999	0	-0.1	-42.0	2	1	0	...	0	
30357	49	197	1	999	0	-0.1	-42.0	6	0	0	...	0	
30358	29	290	1	999	0	-1.8	-46.2	3	0	0	...	0	
30359	43	26	3	999	0	-1.8	-47.1	3	0	0	...	0	

30360 rows × 43 columns

SMOTE - OVERSAMPLING TECHNIQUE

In [52]:
```
sm=SMOTE()
smX,smY = sm.fit_resample(data.drop('y',1),data.y)

# create the new dataset
data_smote = smX.join(smY)

# compare the 2 datasets (original / oversampled)
len(data), len(data_smote)

# compare distribution of classes (original / oversampled)
data.y.value_counts(), data_smote.y.value_counts()
```

Out[52]:
```
(0    28705
 1     1655
```

```
Name: y, dtype: int64,
0    28705
1    28705
Name: y, dtype: int64)
```

In [53]:
```
data_smote
```

Out[53]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.conf.idx	education	job_blue-collar	job_entrepreneur	...	month_mar	month_m
0	33	200	1	999	0	-0.100000	-42.000000	6	0	0	...	0	
1	43	190	2	999	0	1.400000	-41.800000	2	1	0	...	0	
2	46	282	1	999	0	-2.900000	-33.600000	3	0	0	...	0	
3	51	133	5	999	0	1.400000	-41.800000	3	0	0	...	0	
4	24	242	1	999	0	1.400000	-42.700000	3	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
57405	38	271	1	999	0	-0.551629	-40.293734	2	0	0	...	0	
57406	25	242	2	999	0	-1.800000	-49.328979	6	0	0	...	0	
57407	45	442	2	999	0	-2.900000	-35.334845	6	0	0	...	0	
57408	20	211	1	999	0	-2.419975	-40.599990	4	0	0	...	0	
57409	38	237	1	999	0	-1.759258	-47.315349	5	0	0	...	0	

57410 rows × 43 columns

Data before oversampling

In [54]:
```
data["y"].value_counts()
```

Out[54]:
```
0    28705
1     1655
Name: y, dtype: int64
```

Data after oversampling

In [55]:
```
data_smote["y"].value_counts()
```

Out[55]:
```
0    28705
1    28705
Name: y, dtype: int64
```

- FOR IMBALANCED DATA WE NEED TO OVERSAMPLE OR UNDERSAMPLE THE DATA I PREFER TO OVERSAMPLE DATA USING SMOTE

STANDARDIZE THE DATA TO USE FOR MODELS

In [56]:
```
def stdData(data,y,std):
    D = data.copy()
    if std == "ss":
        tr = preprocessing.StandardScaler()
    elif std == "minmax":
        tr = preprocessing.MinMaxScaler()
    else:
        return("Invalid type specifies")

    D.iloc[:,:]=tr.fit_transform(D.iloc[:,:])
    D[y] = data[y]
    return (D)
```

In [57]:
```
df_ss = stdData(data_smote,"y","ss")
```

In [58]:

```python
df_mm = stdData(data_smote,"y","minmax")
```

SPLIT DATA INTO TRAIN AND TEST

In [59]:

```python
def splitdata(data,y,ratio=0.3):

    trainx,testx,trainy,testy = train_test_split(data.drop(y,1),
                                                 data[y],
                                                 test_size = ratio )

    return(trainx,testx,trainy,testy)
```

In [60]:

```python
#trainx,testx,trainy,testy = splitdata(data_smote,'y')
#print(trainx.shape, trainy.shape, testx.shape, testy.shape)
```

In [61]:

```python
trainx,testx,trainy,testy = splitdata(df_ss,'y')
print(trainx.shape, trainy.shape, testx.shape, testy.shape)
```

```
(40187, 42) (40187,) (17223, 42) (17223,)
```

In [62]:

```python
def cm(actual,predicted):
    # method 1
    print("Confusion matrix:\n",confusion_matrix(actual,predicted))

    print("\n")

    # method 2: using cross tab to print confusion matrix
    df = pd.DataFrame({'actual':actual,'predicted':predicted})
    print("Cross tab:\n",pd.crosstab(df.actual,df.predicted,margins=True))

    print("\n")

    # print the classification report
    print("Classification report:\n",classification_report(actual,predicted))
```

RANDOM FOREST

In [63]:

```python
m_rf = RandomForestClassifier().fit(trainx,trainy)
m_rf.estimators_

y_train_pred = m_rf.predict(trainx)
y_test_pred = m_rf.predict(testx)

# confusion matrix,cross tab,classification report
print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')

accuracy_score_rf_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_rf_train)
accuracy_score_rf_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_rf_test)
print('----------------------------------------------------------------')
print("----------------------------------------------------------------")
cv_rf_train=np.mean(cross_val_score(m_rf,trainx,trainy,scoring='f1',cv=5))
cv_rf_test=np.mean(cross_val_score(m_rf,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_rf_train)
print('Test cross val score: %.3f' % cv_rf_test)
print('----------------------------------------------------------------')
print("----------------------------------------------------------------")

recall_score_rf_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_rf_train)
recall_score_rf_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_rf_test)
print('----------------------------------------------------------------')
print("----------------------------------------------------------------")
f1_score_rf_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_rf_train)
f1_score_rf_test=f1_score(testy,y_test_pred)
```

```
print('Test f1 score:',f1_score_rf_test)
print('-----------------------------------------------------------------')
print("-----------------------------------------------------------------")
precision_score_rf_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_rf_train)
precision_score_rf_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_rf_test)
print('-----------------------------------------------------------------')
print("-----------------------------------------------------------------")

df_rf=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_rf)
print("Cross Tab:",pd.crosstab(df_rf.actual,df_rf.predicted,margins=True))
```

```
Train Matrices:

Confusion matrix:
 [[20096     0]
 [    0 20091]]


Cross tab:
 predicted      0      1    All
actual
0          20096      0  20096
1              0  20091  20091
All        20096  20091  40187


Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     20096
           1       1.00      1.00      1.00     20091

    accuracy                           1.00     40187
   macro avg       1.00      1.00      1.00     40187
weighted avg       1.00      1.00      1.00     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8388  221]
 [ 244 8370]]


Cross tab:
 predicted     0     1    All
actual
0          8388   221   8609
1           244  8370   8614
All        8632  8591  17223


Classification report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97      8609
           1       0.97      0.97      0.97      8614

    accuracy                           0.97     17223
   macro avg       0.97      0.97      0.97     17223
weighted avg       0.97      0.97      0.97     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 1.0
Test accuracy_score: 0.9730012192997736
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.970
Test cross val score: 0.967
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 1.0
Test recall score: 0.971674019038774
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 1.0
Test f1 score: 0.9729729729729729
```

```
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 1.0
Test Precision: 0.9742754044930741
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
Cross Tab: predicted     0     1    All
actual
0           8388   221   8609
1            244  8370   8614
All         8632  8591  17223
```

HYPERTUNING - RANDOM FOREST

In [64]:

```python
rf_clf = RandomForestClassifier(n_estimators=500,max_features=2,class_weight = 'balanced')
rf_clf.fit(trainx,trainy)


y_train_pred = rf_clf.predict(trainx)
y_test_pred = rf_clf.predict(testx)


# confusion matrix,cross tab,classification report
print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

accuracy_score_rf_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_rf_train)
accuracy_score_rf_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_rf_test)
print('-----------------------------------------------------------------')
cv_rf_train=np.mean(cross_val_score(rf_clf,trainx,trainy,scoring='f1',cv=5))
cv_rf_test=np.mean(cross_val_score(rf_clf,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_rf_train)
print('Test cross val score: %.3f' % cv_rf_test)
print('-----------------------------------------------------------------')
print("-----------------------------------------------------------------")

recall_score_rf_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_rf_train)
recall_score_rf_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_rf_test)
print('-----------------------------------------------------------------')
print("-----------------------------------------------------------------")
f1_score_rf_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_rf_train)
f1_score_rf_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_rf_test)
print('-----------------------------------------------------------------')
print("-----------------------------------------------------------------")
precision_score_rf_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_rf_train)
precision_score_rf_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_rf_test)
print('-----------------------------------------------------------------')
print("-----------------------------------------------------------------")

df_rf=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_rf)
print("-----------------------------------------------------------------")
print("-----------------------------------------------------------------")
```

```
Train Matrices:
```

```
Train Matrices:

Confusion matrix:
 [[20096     0]
 [    0 20091]]


Cross tab:
 predicted      0      1    All
actual
0          20096      0  20096
1              0  20091  20091
All        20096  20091  40187


Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     20096
           1       1.00      1.00      1.00     20091

    accuracy                           1.00     40187
   macro avg       1.00      1.00      1.00     40187
weighted avg       1.00      1.00      1.00     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8378  231]
 [ 272 8342]]


Cross tab:
 predicted     0     1    All
actual
0          8378   231   8609
1           272  8342   8614
All        8650  8573  17223


Classification report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97      8609
           1       0.97      0.97      0.97      8614

    accuracy                           0.97     17223
   macro avg       0.97      0.97      0.97     17223
weighted avg       0.97      0.97      0.97     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 1.0
Test accuracy_score: 0.9707948673285722
---------------------------------------------------------------
Train cross val score: 0.970
Test cross val score: 0.966
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 1.0
Test recall score: 0.9684234966333876
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 1.0
Test f1 score: 0.9707336940711003
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 1.0
Test Precision: 0.9730549399276799
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
```

```
2447       0          0
18946      0          0
15607      0          0

[17223 rows x 2 columns]
----------------------------------------------------------------
----------------------------------------------------------------
```

IMPORTANT FEATURES USING RANDOM FOREST MODEL

In [65]:

```
# Important features
# method 1
feat = pd.DataFrame({'feature':trainx.columns,
                     'score':rf_clf.feature_importances_})

feat = feat.sort_values('score',ascending=False)
print(feat.head(10))
```

```
              feature     score
1            duration  0.159452
5        emp.var.rate  0.155449
6       cons.conf.idx  0.078324
28  contact_telephone  0.062667
34          month_may  0.053920
0                 age  0.043641
22    default_unknown  0.034101
8     job_blue-collar  0.033201
31          month_jul  0.031408
7           education  0.028898
```

In [66]:

```
# method 2 : RFE
rfe = RFE(m_rf,n_features_to_select=10).fit(trainx,trainy)

feat = pd.DataFrame({'feature':trainx.columns,
                     'support':rfe.support_,
                     'rank':rfe.ranking_ })
feat = feat.sort_values('rank')
print(feat)
```

```
              feature  support  rank
0                 age     True     1
1            duration     True     1
5        emp.var.rate     True     1
6       cons.conf.idx     True     1
7           education     True     1
8     job_blue-collar     True     1
35          month_nov     True     1
34          month_may     True     1
28  contact_telephone     True     1
22    default_unknown     True     1
2            campaign    False     2
31          month_jul    False     3
29          month_aug    False     4
38    day_of_week_mon    False     5
19    marital_married    False     6
39    day_of_week_thu    False     7
41    day_of_week_wed    False     8
40    day_of_week_tue    False     9
16     job_technician    False    10
27           loan_yes    False    11
25        housing_yes    False    12
14       job_services    False    13
20     marital_single    False    14
11     job_management    False    15
32          month_jun    False    16
13  job_self-employed    False    17
17     job_unemployed    False    18
9    job_entrepreneur    False    19
15        job_student    False    20
26       loan_unknown    False    21
12        job_retired    False    22
10      job_housemaid    False    23
36          month_oct    False    24
33          month_mar    False    25
37          month_sep    False    26
24    housing_unknown    False    27
30          month_dec    False    28
```

```
18        job_unknown    False    29
21    marital_unknown    False    30
23        default_yes    False    31
3               pdays    False    32
4            previous    False    33
```

BUILD MULTIPLE ML MODELS -

LOGISTIC REGRESSION MODEL (M1)

In [67]:

```
def buildModel(trainx,trainy):
    model = smapi.Logit(trainy,trainx).fit()
    return(model)
```

In [68]:

```
def predictClass(probs,cutoff):
    if (0<=cutoff<=1):
        P = probs.copy()
        P[P < cutoff] = 0
        P[P > cutoff] = 1

        return(P.astype(int))
```

In [69]:

```
def bestFeatures(trainx,trainy):
    features = trainx.columns

    fscore,pval = f_classif(trainx,trainy)

    df = pd.DataFrame({'feature':features, 'fscore':fscore,'pval':pval})
    df = df.sort_values('fscore',ascending=False)
    return(df)
```

In [70]:

```
'''# build model M1
m_lr = buildModel(trainx,trainy)

# summarise the model
print(m_lr.summary())

# predict on the test data and convert predictions into classes
p_lr_train = m_lr.predict(trainx)
p_lr_test = m_lr.predict(testx)
cutoff = 0.50
y_train_pred = predictClass(p_lr_train,cutoff)
y_test_pred = predictClass(p_lr_test,cutoff)


# confusion matrix,cross tab,classification report
print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('-------------------------------------------------------------------')
print('-------------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('-------------------------------------------------------------------')
print('-------------------------------------------------------------------')
# select the best features
print("Best Features:\n",bestFeatures(trainx,trainy).head(10))
print('-------------------------------------------------------------------')
print('-------------------------------------------------------------------')

accuracy_score_lr_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_lr_train)
accuracy_score_lr_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_lr_test)
print('-------------------------------------------------------------------')
print('-------------------------------------------------------------------')


recall_score_lr_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_lr_train)
recall_score_lr_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_lr_test)
print('-------------------------------------------------------------------')
print('-------------------------------------------------------------------')
f1_score_lr_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_lr_train)
f1_score_lr_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_lr_test)
```

```
print('---------------------------------------------------------------')
print('---------------------------------------------------------------')
precision_score_lr_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_lr_train)
precision_score_lr_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_lr_test)
print('---------------------------------------------------------------')
print('---------------------------------------------------------------')
#cv_lr_train=np.mean(cross_val_score(m_lr,trainx,trainy,scoring='f1',cv=5))
#cv_lr_test=np.mean(cross_val_score(m_lr,testx,testy,scoring='f1',cv=5))
#print('Train cross val score: %.3f' % cv_lr_train)
#print('Test cross val score: %.3f' % cv_lr_test)
#print('---------------------------------------------------------------')
print('---------------------------------------------------------------')
df_lr=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_lr)'''
```

Out[70]:

```
'# build model M1\nm_lr = buildModel(trainx,trainy)\n\n# summarise the model\nprint(m_lr.summary())\n\n# predict
on the test data and convert predictions into classes\np_lr_train = m_lr.predict(trainx)\np_lr_test = m_lr.predic
t(testx)\ncutoff = 0.50\ny_train_pred = predictClass(p_lr_train,cutoff)\ny_test_pred = predictClass(p_lr_test,cut
off)\n\n\n# confusion matrix,cross tab,classification report\nprint("Train Matrices:\n")\nprint(cm(trainy,y_train
_pred))\nprint(\'------------------------------------------------------------\')\nprint(\'--------------------
-----------------------------------------\')\nprint("Test Matrices:\n")\nprint(cm(testy,y_test_pred))\nprint(\'
------------------------------------------------------------\')\nprint(\'-------------------------------------
-------------------------\')\n# select the best features\nprint("Best Features:\n",bestFeatures(trainx,trainy).h
ead(10))\nprint(\'------------------------------------------------------------\')\nprint(\'-------------------
------------------------------------------\')\n\naccuracy_score_lr_train=accuracy_score(trainy,y_train_pred)\np
rint(\'Train accuracy_score:\',accuracy_score_lr_train)\naccuracy_score_lr_test=accuracy_score(testy,y_test_pred)
\nprint(\'Test accuracy_score:\',accuracy_score_lr_test)\nprint(\'---------------------------------------------
---------------\')\nprint(\'------------------------------------------------------------\')\n\n\nrecall_score
_lr_train=recall_score(trainy, y_train_pred)\nprint(\'Train recall score:\',recall_score_lr_train)\nrecall_score_
lr_test=recall_score(testy, y_test_pred)\nprint(\'Test recall score:\',recall_score_lr_test)\nprint(\'----------
--------------------------------------------------\')\nprint(\'----------------------------------------------
--------------\')\nf1_score_lr_train=f1_score(trainy,y_train_pred)\nprint(\'Train f1 score:\',f1_score_lr_train)
\nf1_score_lr_test=f1_score(testy,y_test_pred)\nprint(\'Test f1 score:\',f1_score_lr_test)\nprint(\'---------
-----------------------------------------------\')\nprint(\'------------------------------------------------
------------\')\nprecision_score_lr_train=precision_score(trainy, y_train_pred)\nprint(\'Train Precision:\',prec
ision_score_lr_train)\nprecision_score_lr_test=precision_score(testy, y_test_pred)\nprint(\'Test Precision:\',pre
cision_score_lr_test)\nprint(\'------------------------------------------------------------\')\nprint(\'------
------------------------------------------------------\')\n#cv_lr_train=np.mean(cross_val_score(m_lr,trainx,tr
ainy,scoring=\'f1\',cv=5))\n#cv_lr_test=np.mean(cross_val_score(m_lr,testx,testy,scoring=\'f1\',cv=5))\n#print(\'
Train cross val score: %.3f\' % cv_lr_train)\n#print(\'Test cross val score: %.3f\' % cv_lr_test)\n#print(\'-----
-------------------------------------------------------\')\nprint(\'----------------------------------------
--------------------\')\ndf_lr=pd.DataFrame({\'actual\':testy,\'predicted\':y_test_pred})\nprint(df_lr)'
```

LOGISTIC REGRESSION USING SKLEARN (M1)

In [71]:

```python
sk_lr = LogisticRegression()
sk_lr.fit(trainx,trainy)
y_train_pred = sk_lr.predict(trainx)
y_train_proba = sk_lr.predict_proba(trainx)[:,1]
y_test_pred =sk_lr.predict(testx) # Predicted class
y_test_proba= sk_lr.predict_proba(testx)[:,1] # probability of class

print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('---------------------------------------------------------------')
print('---------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('---------------------------------------------------------------')
print('---------------------------------------------------------------')

accuracy_score_sklr_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_sklr_train)
accuracy_score_sklr_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_sklr_test)
print('---------------------------------------------------------------')
print('---------------------------------------------------------------')
cv_sklr_train=np.mean(cross_val_score(sk_lr,trainx,trainy,scoring='f1',cv=5))
cv_sklr_test=np.mean(cross_val_score(sk_lr,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_sklr_train)
print('Test cross val score: %.3f' % cv_sklr_test)
print('---------------------------------------------------------------')
print('---------------------------------------------------------------')

recall_score_sklr_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_sklr_train)
recall_score_sklr_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_sklr_test)
print('---------------------------------------------------------------')
```

```python
print('----------------------------------------------------------------')
f1_score_sklr_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_sklr_train)
f1_score_sklr_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_sklr_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
precision_score_sklr_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_sklr_train)
precision_score_sklr_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_sklr_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
# select the best features
print("Best Features:\n",bestFeatures(trainx,trainy).head(10))
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')

df_sk_lr=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_sk_lr)
```

```
Train Matrices:

Confusion matrix:
 [[19109   987]
 [  865 19226]]


Cross tab:
 predicted      0      1    All
actual
0          19109    987  20096
1            865  19226  20091
All        19974  20213  40187


Classification report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.95     20096
           1       0.95      0.96      0.95     20091

    accuracy                           0.95     40187
   macro avg       0.95      0.95      0.95     40187
weighted avg       0.95      0.95      0.95     40187

None
----------------------------------------------------------------
----------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8172  437]
 [ 362 8252]]


Cross tab:
 predicted     0     1    All
actual
0          8172   437   8609
1           362  8252   8614
All        8534  8689  17223


Classification report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.95      8609
           1       0.95      0.96      0.95      8614

    accuracy                           0.95     17223
   macro avg       0.95      0.95      0.95     17223
weighted avg       0.95      0.95      0.95     17223

None
----------------------------------------------------------------
----------------------------------------------------------------
Train accuracy_score: 0.953915445293254
Test accuracy_score: 0.9536085467107938
----------------------------------------------------------------
----------------------------------------------------------------
Train cross val score: 0.954
Test cross val score: 0.953
----------------------------------------------------------------
```

```
----------------------------------------------------------------
Train recall score: 0.9569458961724155
Test recall score: 0.9579753889017878
----------------------------------------------------------------
----------------------------------------------------------------
Train f1 score: 0.9540492258832871
Test f1 score: 0.9538230364676645
----------------------------------------------------------------
----------------------------------------------------------------
Train Precision: 0.951170039083758
Test Precision: 0.9497065254920014
----------------------------------------------------------------
----------------------------------------------------------------
Best Features:
              feature        fscore  pval
5        emp.var.rate  20502.444349   0.0
1            duration  11903.371506   0.0
28  contact_telephone  10285.887873   0.0
34          month_may   6146.743667   0.0
22    default_unknown   5209.208656   0.0
8     job_blue-collar   4126.647028   0.0
19    marital_married   3451.079208   0.0
31          month_jul   2995.438932   0.0
2            campaign   2554.498791   0.0
38    day_of_week_mon   2421.365088   0.0
----------------------------------------------------------------
----------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

HYPERTUNING - LOGISTIC REGRESSION USING SKLEARN (M1.1)

In [72]:

```
sk_lr_ht = LogisticRegression(C=1,class_weight='balanced')
sk_lr_ht.fit(trainx,trainy)
y_train_pred =sk_lr_ht.predict(trainx)
y_test_pred = sk_lr_ht.predict(testx)
cross_val = np.mean(cross_val_score(sk_lr_ht,trainx,trainy,cv=5,scoring='f1'))

print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')

accuracy_score_sklrht_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_sklrht_train)
accuracy_score_sklrht_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_sklrht_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
cv_sklrht_train=np.mean(cross_val_score(sk_lr_ht,trainx,trainy,scoring='f1',cv=5))
cv_sklrht_test=np.mean(cross_val_score(sk_lr_ht,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_sklrht_train)
print('Test cross val score: %.3f' % cv_sklrht_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')

recall_score_sklrht_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_sklrht_train)
recall_score_sklrht_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_sklrht_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
f1_score_sklrht_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_sklrht_train)
f1_score_sklrht_test=f1_score(testy,y_test_pred)
```

```
print('Test f1 score:',f1_score_sklrht_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
precision_score_sklrht_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_sklrht_train)
precision_score_sklrht_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_sklrht_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

df_sk_lr_ht=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_sk_lr_ht)
```

```
Train Matrices:

Confusion matrix:
 [[19109   987]
 [  865 19226]]


Cross tab:
 predicted      0      1    All
actual
0          19109    987  20096
1            865  19226  20091
All        19974  20213  40187


Classification report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.95     20096
           1       0.95      0.96      0.95     20091

    accuracy                           0.95     40187
   macro avg       0.95      0.95      0.95     40187
weighted avg       0.95      0.95      0.95     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8172  437]
 [ 362 8252]]


Cross tab:
 predicted     0     1    All
actual
0          8172   437   8609
1           362  8252   8614
All        8534  8689  17223


Classification report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.95      8609
           1       0.95      0.96      0.95      8614

    accuracy                           0.95     17223
   macro avg       0.95      0.95      0.95     17223
weighted avg       0.95      0.95      0.95     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 0.953915445293254
Test accuracy_score: 0.9536085467107938
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.954
Test cross val score: 0.953
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 0.9569458961724155
Test recall score: 0.9579753889017878
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 0.9540492258832871
Test f1 score: 0.9538230364676645
---------------------------------------------------------------
```

```
-----------------------------------------------------------------
Train Precision: 0.951170039083758
Test Precision: 0.9497065254920014
-----------------------------------------------------------------
-----------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

DECISION TREE

In [73]:

```
dt_clf = DecisionTreeClassifier()
dt_clf.fit(trainx,trainy)
y_train_pred = dt_clf.predict(trainx)
y_test_pred = dt_clf.predict(testx)

print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

accuracy_score_dt_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_dt_train)
accuracy_score_dt_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_dt_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
cv_dt_train=np.mean(cross_val_score(dt_clf,trainx,trainy,scoring='f1',cv=5))
cv_dt_test=np.mean(cross_val_score(dt_clf,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_dt_train)
print('Test cross val score: %.3f' % cv_dt_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

recall_score_dt_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_dt_train)
recall_score_dt_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_dt_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
f1_score_dt_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_dt_train)
f1_score_dt_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_dt_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
precision_score_dt_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_dt_train)
precision_score_dt_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_dt_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
df_dt=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_dt)
```

```
Train Matrices:

Confusion matrix:
 [[20096     0]
 [    0 20091]]


Cross tab:
 predicted      0      1    All
actual
0          20096      0  20096
```

```
1              0  20091  20091
All        20096  20091  40187


Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     20096
           1       1.00      1.00      1.00     20091

    accuracy                           1.00     40187
   macro avg       1.00      1.00      1.00     40187
weighted avg       1.00      1.00      1.00     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8232  377]
 [ 284 8330]]


Cross tab:
 predicted     0     1    All
actual
0          8232   377   8609
1           284  8330   8614
All        8516  8707  17223


Classification report:
               precision    recall  f1-score   support

           0       0.97      0.96      0.96      8609
           1       0.96      0.97      0.96      8614

    accuracy                           0.96     17223
   macro avg       0.96      0.96      0.96     17223
weighted avg       0.96      0.96      0.96     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 1.0
Test accuracy_score: 0.9616210880798932
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.958
Test cross val score: 0.951
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 1.0
Test recall score: 0.9670304156025076
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 1.0
Test f1 score: 0.9618382310490157
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 1.0
Test Precision: 0.9567015045365798
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

HYPERTUNING - DECISION TREE

In [74]:

```
dt_clf_ht = DecisionTreeClassifier()
param_grid ={
    'max_depth':[2,3,4,5,6,7,8,9],
    'criterion':['gini','entropy'],
     'class_weight':['balanced',{1:2,2:1}]

}
grid_clf=GridSearchCV(dt_clf_ht,param_grid=param_grid,cv= 5,scoring='f1')
grid_clf.fit(trainx,trainy)
y_train_prob = grid_clf.predict_proba(trainx)[:,0]
y_test_prob = grid_clf.predict_proba(testx)[:,0]

print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

accuracy_score_dtht_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_dtht_train)
accuracy_score_dtht_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_dtht_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
cv_dtht_train=np.mean(cross_val_score(dt_clf_ht,trainx,trainy,scoring='f1',cv=5))
cv_dtht_test=np.mean(cross_val_score(dt_clf_ht,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_dtht_train)
print('Test cross val score: %.3f' % cv_dtht_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

recall_score_dtht_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_dtht_train)
recall_score_dtht_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_dtht_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
f1_score_dtht_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_dtht_train)
f1_score_dtht_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_dtht_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
precision_score_dtht_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_dtht_train)
precision_score_dtht_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_dtht_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
df_dt_ht=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_dt_ht)
```

```
Train Matrices:

Confusion matrix:
 [[20096     0]
 [    0 20091]]



Cross tab:
 predicted      0      1    All
actual
0          20096      0  20096
1              0  20091  20091
All        20096  20091  40187



Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     20096
           1       1.00      1.00      1.00     20091

    accuracy                           1.00     40187
   macro avg       1.00      1.00      1.00     40187
weighted avg       1.00      1.00      1.00     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:
```

```
Confusion matrix:
 [[8232  377]
 [ 284 8330]]


Cross tab:
 predicted     0     1    All
actual
0          8232   377   8609
1           284  8330   8614
All        8516  8707  17223


Classification report:
               precision    recall  f1-score   support

           0       0.97      0.96      0.96      8609
           1       0.96      0.97      0.96      8614

    accuracy                           0.96     17223
   macro avg       0.96      0.96      0.96     17223
weighted avg       0.96      0.96      0.96     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 1.0
Test accuracy_score: 0.9616210880798932
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.957
Test cross val score: 0.951
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 1.0
Test recall score: 0.9670304156025076
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 1.0
Test f1 score: 0.9618382310490157
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 1.0
Test Precision: 0.9567015045365798
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

In [75]:

```
grid_clf.best_params_
```

Out[75]:

```
{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 9}
```

In [76]:

```
for k,v in grid_clf.best_params_.items():
    if k=="max_depth":
        max_dpt=v
    if k=="class_weight":
        class_wt=v
    if k=="criterion":
        crit=v
```

In [77]:

```
grid_clf.best_estimator_
```

Out[77]:

```
DecisionTreeClassifier(class_weight='balanced', max_depth=9)
```

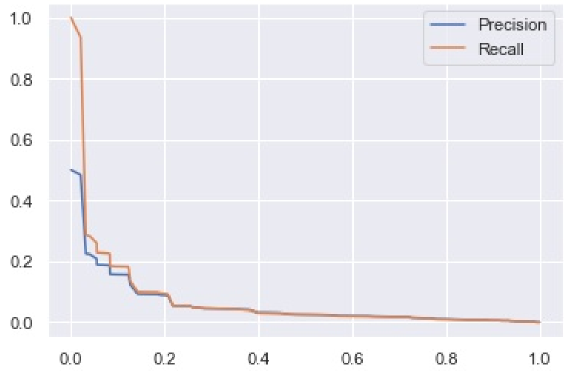
Out[77]: DecisionTreeClassifier(class_weight='balanced', max_depth=5)

In [78]:
```python
grid_clf.best_score_
```

Out[78]: 0.9483606597899161

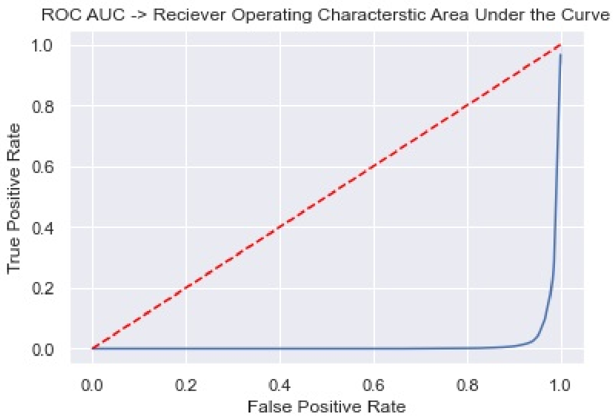
In [79]:
```python
precision, recall, thresholds = precision_recall_curve(trainy, y_train_prob,pos_label=1)
```

In [80]:
```python
sns.set()
sns.lineplot(thresholds,precision[:-1],label='Precision')
sns.lineplot(thresholds,recall[:-1],label ='Recall')
plt.legend()
plt.show()
```



In [81]:
```python
fpr, tpr, thresholds = roc_curve(trainy, y_train_prob, pos_label=1)
```

In [82]:
```python
plt.figure
sns.lineplot(fpr,tpr)
sns.lineplot(x = [0,1],y=[0,1],linestyle = '--',color ='red')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC AUC -> Reciever Operating Characterstic Area Under the Curve')
plt.show()
```



ADABOOST MODEL (M2)

In [83]:
```python
trees = 100

m_ab = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),n_estimators=trees).fit(trainx,trainy)

p_ab = m_ab.predict(testx)
p_ab

def cm_adaboost(actual,pred):

    # accuracy score
    print("Model Accuracy = {}".format(accuracy_score(actual,pred)))
```

```
    print("\n")

    # confusion matrix
    df = pd.DataFrame({'actual':actual, 'predicted':pred})
    print(pd.crosstab(df.actual, df.predicted, margins=True))
    print("\n")

    # classification report
    print(classification_report(actual,pred))

# model 1 evaluation
cm_adaboost(testy,p_ab)
```

```
Model Accuracy = 0.9626081402775358


predicted     0     1    All
actual
0          8257   352   8609
1           292  8322   8614
All        8549  8674  17223


              precision    recall  f1-score   support

           0       0.97      0.96      0.96      8609
           1       0.96      0.97      0.96      8614

    accuracy                           0.96     17223
   macro avg       0.96      0.96      0.96     17223
weighted avg       0.96      0.96      0.96     17223
```

ADABOOST MODEL USING BASE MODEL AS DECISION TREE (M2.1)

In [84]:

```
base_estimator = DecisionTreeClassifier(max_depth=max_dpt,class_weight=class_wt,criterion=crit)
ad_dt_clf = AdaBoostClassifier(base_estimator=base_estimator,n_estimators=trees,learning_rate=0.1)
ad_dt_clf.fit(trainx,trainy)
y_train_pred =ad_dt_clf.predict(trainx)
y_test_pred = ad_dt_clf.predict(testx)

print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')

accuracy_score_addt_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_addt_train)
accuracy_score_addt_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_addt_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
cv_addt_train=np.mean(cross_val_score(dt_clf,trainx,trainy,scoring='f1',cv=5))
cv_addt_test=np.mean(cross_val_score(dt_clf,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_addt_train)
print('Test cross val score: %.3f' % cv_addt_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')

recall_score_addt_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_addt_train)
recall_score_addt_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_addt_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
f1_score_addt_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_addt_train)
f1_score_addt_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_addt_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
precision_score_addt_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_addt_train)
precision_score_addt_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_addt_test)
print('----------------------------------------------------------------')
print('----------------------------------------------------------------')
df_dt_ad=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
```

```
print(df_dt_ad)
```

```
Train Matrices:

Confusion matrix:
 [[20096     0]
 [    0 20091]]


Cross tab:
 predicted      0      1    All
actual
0          20096      0  20096
1              0  20091  20091
All        20096  20091  40187


Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     20096
           1       1.00      1.00      1.00     20091

    accuracy                           1.00     40187
   macro avg       1.00      1.00      1.00     40187
weighted avg       1.00      1.00      1.00     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8376  233]
 [ 231 8383]]


Cross tab:
 predicted     0     1    All
actual
0          8376   233   8609
1           231  8383   8614
All        8607  8616  17223


Classification report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97      8609
           1       0.97      0.97      0.97      8614

    accuracy                           0.97     17223
   macro avg       0.97      0.97      0.97     17223
weighted avg       0.97      0.97      0.97     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 1.0
Test accuracy_score: 0.9730592811937525
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.957
Test cross val score: 0.952
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 1.0
Test recall score: 0.9731831901555608
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 1.0
Test f1 score: 0.9730702263493906
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 1.0
Test Precision: 0.9729572887650882
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
```

```
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

ADABOOST MODEL USING BASE MODEL AS LOGISTIC REGRESSION (M2.2)

In [85]:

```
base_estimator = LogisticRegression(C=2)
ad_lr_clf = AdaBoostClassifier(base_estimator=base_estimator,n_estimators=trees,learning_rate=1)
ad_lr_clf.fit(trainx,trainy)
y_train_pred =ad_lr_clf.predict(trainx)
y_test_pred = ad_lr_clf.predict(testx)


print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

accuracy_score_adlr_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_adlr_train)
accuracy_score_adlr_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_adlr_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
cv_adlr_train=np.mean(cross_val_score(dt_clf,trainx,trainy,scoring='f1',cv=5))
cv_adlr_test=np.mean(cross_val_score(dt_clf,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_adlr_train)
print('Test cross val score: %.3f' % cv_adlr_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

recall_score_adlr_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_adlr_train)
recall_score_adlr_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_adlr_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
f1_score_adlr_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_adlr_train)
f1_score_adlr_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_adlr_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
precision_score_adlr_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_adlr_train)
precision_score_adlr_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_adlr_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
df_lr_ad=pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_lr_ad)
```

```
Train Matrices:

Confusion matrix:
 [[19076  1020]
 [  892 19199]]


Cross tab:
 predicted      0      1    All
actual
0          19076   1020  20096
1            892  19199  20091
All        19968  20219  40187


Classification report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.95     20096
           1       0.95      0.96      0.95     20091
```

```
    accuracy                           0.95     40187
   macro avg       0.95      0.95      0.95     40187
weighted avg       0.95      0.95      0.95     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8158  451]
 [ 366 8248]]


Cross tab:
 predicted     0     1    All
actual
0          8158   451   8609
1           366  8248   8614
All        8524  8699  17223


Classification report:
               precision    recall  f1-score   support

           0       0.96      0.95      0.95      8609
           1       0.95      0.96      0.95      8614

    accuracy                           0.95     17223
   macro avg       0.95      0.95      0.95     17223
weighted avg       0.95      0.95      0.95     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 0.952422425162366
Test accuracy_score: 0.9525634326191721
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.957
Test cross val score: 0.951
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 0.9556020108506297
Test recall score: 0.9575110285581612
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 0.9525676010915407
Test f1 score: 0.9528100271472304
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 0.9495524012067857
Test Precision: 0.948154960340269
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

NAIVE BAYES (M3)

In [86]:

```
# build the NaiveBayes classifier model
m_nb = GaussianNB().fit(trainx,trainy)

# predict on test data
y_train_pred =m_nb.predict(trainx)
y_test_pred = m_nb.predict(testx)
```

```
print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

accuracy_score_nb_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_nb_train)
accuracy_score_nb_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_nb_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
cv_nb_train=np.mean(cross_val_score(m_nb,trainx,trainy,scoring='f1',cv=5))
cv_nb_test=np.mean(cross_val_score(m_nb,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_nb_train)
print('Test cross val score: %.3f' % cv_nb_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

recall_score_nb_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_nb_train)
recall_score_nb_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_nb_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
f1_score_nb_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_nb_train)
f1_score_nb_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_nb_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
precision_score_nb_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_nb_train)
precision_score_nb_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_nb_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
df_nb = pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_nb)
```

```
Train Matrices:

Confusion matrix:
 [[14283  5813]
 [  978 19113]]


Cross tab:
 predicted      0      1    All
actual
0          14283   5813  20096
1            978  19113  20091
All        15261  24926  40187


Classification report:
               precision    recall  f1-score   support

           0       0.94      0.71      0.81     20096
           1       0.77      0.95      0.85     20091

    accuracy                           0.83     40187
   macro avg       0.85      0.83      0.83     40187
weighted avg       0.85      0.83      0.83     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[6097 2512]
 [ 416 8198]]


Cross tab:
 predicted     0      1    All
actual
0          6097   2512   8609
1           416   8198   8614
All        6513  10710  17223
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.94      0.71      0.81      8609
           1       0.77      0.95      0.85      8614

    accuracy                           0.83     17223
   macro avg       0.85      0.83      0.83     17223
weighted avg       0.85      0.83      0.83     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 0.8310150048523154
Test accuracy_score: 0.8299947744295418
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.860
Test cross val score: 0.812
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 0.9513214872330894
Test recall score: 0.951706524262828
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 0.8491458782237821
Test f1 score: 0.8484785758642103
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 0.7667896975046137
Test Precision: 0.7654528478057889
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          1
49600       1          1
2447        0          1
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

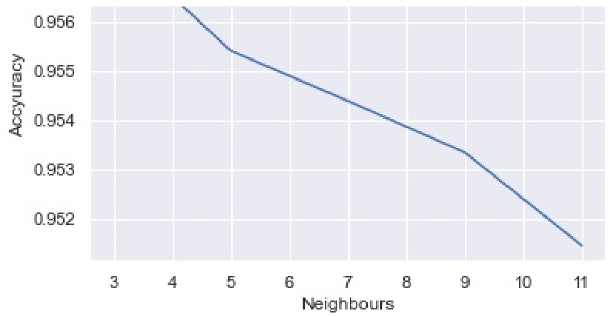
KNN MODEL (M4)

In [87]:

```python
# cross-validation to determine the best K
cv_accuracy = []

n_list = np.arange(3,12,2); n_list

for n in n_list:
    model = neighbors.KNeighborsClassifier(n_neighbors=n)
    scores = cross_val_score(model,trainx,trainy,cv=10,scoring='accuracy')
    cv_accuracy.append(scores.mean() )

print(cv_accuracy)

bestK = n_list[cv_accuracy.index(max(cv_accuracy))]
print("best K = ", bestK)

# plot the Accuracy vs Neighbours to determine the best K
plt.plot(n_list,cv_accuracy)
plt.xlabel("Neighbours")
plt.ylabel("Accyuracy")
plt.title("Accuracy - Neighbours")
```

```
[0.957623104588694, 0.9554084190191166, 0.9543881347075756, 0.9533430490882594, 0.9514518084890696]
best K =  3
```

Out[87]: `Text(0.5, 1.0, 'Accuracy - Neighbours')`



In [88]:

```
# build the model using the best K
m_knn = neighbors.KNeighborsClassifier(n_neighbors=bestK,metric = "manhattan").fit(trainx,trainy)
# metric = "manhattan"

# predict on test data
y_train_pred =m_knn.predict(trainx)
y_test_pred = m_knn.predict(testx)

print("Train Matrices:\n")
print(cm(trainy,y_train_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy,y_test_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

accuracy_score_knn_train=accuracy_score(trainy,y_train_pred)
print('Train accuracy_score:',accuracy_score_knn_train)
accuracy_score_knn_test=accuracy_score(testy,y_test_pred)
print('Test accuracy_score:',accuracy_score_knn_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
cv_knn_train=np.mean(cross_val_score(m_knn,trainx,trainy,scoring='f1',cv=5))
cv_knn_test=np.mean(cross_val_score(m_knn,testx,testy,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_knn_train)
print('Test cross val score: %.3f' % cv_knn_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

recall_score_knn_train=recall_score(trainy, y_train_pred)
print('Train recall score:',recall_score_knn_train)
recall_score_knn_test=recall_score(testy, y_test_pred)
print('Test recall score:',recall_score_knn_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
f1_score_knn_train=f1_score(trainy,y_train_pred)
print('Train f1 score:',f1_score_knn_train)
f1_score_knn_test=f1_score(testy,y_test_pred)
print('Test f1 score:',f1_score_knn_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
precision_score_knn_train=precision_score(trainy, y_train_pred)
print('Train Precision:',precision_score_knn_train)
precision_score_knn_test=precision_score(testy, y_test_pred)
print('Test Precision:',precision_score_knn_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
df_knn = pd.DataFrame({'actual':testy,'predicted':y_test_pred})
print(df_knn)
```

```
Train Matrices:

Confusion matrix:
 [[19621   475]
 [  533 19558]]


Cross tab:
 predicted      0      1    All
actual
0          19621    475  20096
1            533  19558  20091
All        20154  20033  40187


Classification report:
               precision    recall  f1-score   support
```

```
           0       0.97      0.98      0.97     20096
           1       0.98      0.97      0.97     20091

    accuracy                           0.97     40187
   macro avg       0.97      0.97      0.97     40187
weighted avg       0.97      0.97      0.97     40187

None
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[8237  372]
 [ 304 8310]]


Cross tab:
 predicted     0     1    All
actual
0          8237   372   8609
1           304  8310   8614
All        8541  8682  17223


Classification report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96      8609
           1       0.96      0.96      0.96      8614

    accuracy                           0.96     17223
   macro avg       0.96      0.96      0.96     17223
weighted avg       0.96      0.96      0.96     17223

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 0.97491726180108
Test accuracy_score: 0.9607501596702085
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.960
Test cross val score: 0.953
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 0.9734707082773381
Test recall score: 0.9647086138843742
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 0.9748778785764131
Test f1 score: 0.9609158186864013
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 0.9762891229471372
Test Precision: 0.9571527297857636
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
6051        0          0
39970       1          1
46367       1          1
34028       1          1
13824       0          0
...       ...        ...
28336       0          0
49600       1          1
2447        0          0
18946       0          0
15607       0          0

[17223 rows x 2 columns]
```

FOR SVM MODEL IT TAKES LOT OF TIME SO FOR EXAM PURPOSE TAKING SAMPLE DATA

In [89]:

```
data_sample_svm= df_ss.sample(frac=0.20)
```

SVM (M5)

In [90]:

```
data_sample_svm
```

Out[90]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.conf.idx	education	job_blue-collar	job_entrepreneur	...	month_mar	mo
11566	0.695786	0.590688	0.162676	0.0	0.0	0.947865	0.873803	1.068250	-0.399102	-0.143288	...	-0.120375	
32523	2.453387	-0.488783	-0.709090	0.0	0.0	-0.603921	-1.548960	-1.459269	-0.399102	-0.143288	...	-0.120375	
22640	-0.378303	-1.472831	-0.709090	0.0	0.0	1.130429	-0.225598	0.562746	-0.399102	-0.143288	...	-0.120375	
54224	-0.085370	0.859064	0.162676	0.0	0.0	-0.816920	-0.807719	0.562746	-0.399102	-0.143288	...	-0.120375	
28569	0.891075	0.566832	0.162676	0.0	0.0	1.130429	0.934881	1.068250	-0.399102	-0.143288	...	-0.120375	
...	...	...	...	...	...	...	...	...	...	...	...	...	
28733	-0.085370	-1.025537	-0.709090	0.0	0.0	0.947865	0.873803	1.068250	-0.399102	-0.143288	...	-0.120375	
4137	-0.964170	0.047970	1.034442	0.0	0.0	1.130429	-0.225598	-0.448261	-0.399102	6.978942	...	-0.120375	
5760	-0.671237	-0.178659	-0.709090	0.0	0.0	1.130429	0.934881	1.068250	-0.399102	-0.143288	...	-0.120375	
39901	0.500497	0.787497	-0.709090	0.0	0.0	-1.790595	2.205623	-0.448261	-0.399102	-0.143288	...	-0.120375	
28051	0.109919	0.924668	-0.709090	0.0	0.0	1.130429	-0.408831	0.562746	2.505622	-0.143288	...	-0.120375	

11482 rows × 43 columns

In [91]:

```
trainx_svm,testx_svm,trainy_svm,testy_svm = splitdata(data_sample_svm,'y')
print(trainx_svm.shape, trainy_svm.shape, testx_svm.shape, testy_svm.shape)
```

```
(8037, 42) (8037,) (3445, 42) (3445,)
```

In [92]:

```
# SVM specific parameters

# list of values for C and gamma
lim=10
lov_c = np.logspace(-5,4,lim)
lov_g = np.random.random(lim)
```

In [93]:

```
'''
kernels in SVM
linear -> C
sigmoid -> C,gamma
poly -> C,gamma
rbf(radial basis function) -> C,gamma
'''
```

Out[93]:

```
'\nkernels in SVM\nlinear -> C\nsigmoid -> C,gamma\npoly -> C,gamma\nrbf(radial basis function) -> C,gamma\n'
```

In [94]:

```
# build the parameters
params = [{ 'kernel':['linear'], 'C':lov_c,
            'kernel':['sigmoid'],'C':lov_c,'gamma':lov_g,
            'kernel':['poly'],'C':lov_c,'gamma':lov_g,
            'kernel':['rbf'],'C':lov_c,'gamma':lov_g}]
```

In [95]:

```
# perform Grid Search
model = svm.SVC()

# grid CV on stdscaler data
grid = GridSearchCV(model,param_grid=params,
                    scoring="accuracy",cv=3,
                    n_jobs=-1).fit(trainx_svm,trainy_svm)

# best parameters
bp = grid.best_params_
bp1 = bp.copy()
bp1
```

Out[95]:

```
{'C': 10.0, 'gamma': 0.20384418248375802, 'kernel': 'rbf'}
```

In [96]:

```python
# build the model with the best parameters
m_svm = svm.SVC(kernel=bp['kernel'], C=bp['C'], gamma=bp['gamma']).fit(trainx_svm,trainy_svm)


def cm_svm(actual,pred):
    # model accuracy
    print("Model Accuracy = {}".format(accuracy_score(actual,pred)))
    print("\n")

    # confusion matrix
    df = pd.DataFrame({'actual':actual,'pred':pred})
    print(pd.crosstab(df.actual,df.pred,margins=True))
    print("\n")

    # classification report
    print(classification_report(actual,pred))

    return(1)


# predict on test data
y_train_pred =m_svm.predict(trainx_svm)
y_test_pred = m_svm.predict(testx_svm)

print("Train Matrices:\n")
print(cm_svm(trainy_svm,y_train_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
print("Test Matrices:\n")
print(cm(testy_svm,y_test_pred))
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

accuracy_score_svm_train=accuracy_score(trainy_svm,y_train_pred)
print('Train accuracy_score:',accuracy_score_svm_train)
accuracy_score_svm_test=accuracy_score(testy_svm,y_test_pred)
print('Test accuracy_score:',accuracy_score_svm_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
cv_svm_train=np.mean(cross_val_score(m_svm,trainx_svm,trainy_svm,scoring='f1',cv=5))
cv_svm_test=np.mean(cross_val_score(m_svm,testx_svm,testy_svm,scoring='f1',cv=5))
print('Train cross val score: %.3f' % cv_svm_train)
print('Test cross val score: %.3f' % cv_svm_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')

recall_score_svm_train=recall_score(trainy_svm, y_train_pred)
print('Train recall score:',recall_score_svm_train)
recall_score_svm_test=recall_score(testy_svm, y_test_pred)
print('Test recall score:',recall_score_svm_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
f1_score_svm_train=f1_score(trainy_svm,y_train_pred)
print('Train f1 score:',f1_score_svm_train)
f1_score_svm_test=f1_score(testy_svm,y_test_pred)
print('Test f1 score:',f1_score_svm_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
precision_score_svm_train=precision_score(trainy_svm, y_train_pred)
print('Train Precision:',precision_score_svm_train)
precision_score_svm_test=precision_score(testy_svm, y_test_pred)
print('Test Precision:',precision_score_svm_test)
print('-----------------------------------------------------------------')
print('-----------------------------------------------------------------')
df_svm = pd.DataFrame({'actual':testy_svm,'predicted':y_test_pred})
print(df_svm)
```

```
Train Matrices:

Model Accuracy = 0.9990046037078512


pred       0     1   All
actual
0       4000     7  4007
1          1  4029  4030
All     4001  4036  8037


              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4007
           1       1.00      1.00      1.00      4030

    accuracy                           1.00      8037
   macro avg       1.00      1.00      1.00      8037
```

```
weighted avg       1.00      1.00      1.00      8037

1
---------------------------------------------------------------
---------------------------------------------------------------
Test Matrices:

Confusion matrix:
 [[1629   34]
 [ 126 1656]]


Cross tab:
 predicted     0     1   All
actual
0          1629    34  1663
1           126  1656  1782
All        1755  1690  3445


Classification report:
               precision    recall  f1-score   support

           0       0.93      0.98      0.95      1663
           1       0.98      0.93      0.95      1782

    accuracy                           0.95      3445
   macro avg       0.95      0.95      0.95      3445
weighted avg       0.95      0.95      0.95      3445

None
---------------------------------------------------------------
---------------------------------------------------------------
Train accuracy_score: 0.9990046037078512
Test accuracy_score: 0.95355587808418
---------------------------------------------------------------
---------------------------------------------------------------
Train cross val score: 0.945
Test cross val score: 0.930
---------------------------------------------------------------
---------------------------------------------------------------
Train recall score: 0.9997518610421836
Test recall score: 0.9292929292929293
---------------------------------------------------------------
---------------------------------------------------------------
Train f1 score: 0.999008182494421
Test f1 score: 0.9539170506912442
---------------------------------------------------------------
---------------------------------------------------------------
Train Precision: 0.9982656095143707
Test Precision: 0.9798816568047337
---------------------------------------------------------------
---------------------------------------------------------------
       actual  predicted
7347        0          0
9000        0          0
55900       1          1
52250       1          1
49858       1          1
...       ...        ...
13387       0          0
28110       0          0
3036        0          0
1398        0          0
11692       0          0

[3445 rows x 2 columns]
```

In [97]:

```
result = [["Logistic Regression",round(recall_score_sklrht_test,2),round(precision_score_sklrht_test,2)],
          ["Decision Tree",round(recall_score_dtht_test,2),round(precision_score_dtht_test,2)],
        ["Adaboost using Logistic Regression",round(recall_score_adlr_test,2),
round(precision_score_adlr_test,2)],
["Adaboost using Decision Tree",round(recall_score_addt_test,2),
round(precision_score_addt_test,2)],
['Naive Bayes',
round(recall_score_nb_test,2),
round(precision_score_nb_test,2)],
["KNN",round(recall_score_knn_test,2),
round(precision_score_knn_test,2)],
["SVM",
round(recall_score_svm_test,2),
```

```
round(precision_score_svm_test,2)]]
```

In [98]:
```
results = pd.DataFrame(result,columns=['Model','Recall','Precision'])
```

In [99]:
```
results
```

Out[99]:

	Model	Recall	Precision
0	Logistic Regression	0.96	0.95
1	Decision Tree	0.97	0.96
2	Adaboost using Logistic Regression	0.96	0.95
3	Adaboost using Decision Tree	0.97	0.97
4	Naive Bayes	0.95	0.77
5	KNN	0.96	0.96
6	SVM	0.93	0.98

CONCLUSION

- ALL THE MODELS ARE BUILD ON CLEAN DATA ALSO DROPPED MULTICULINEAR FEATURES AND REMOVED ALL

OUTLIERS IN DATA ALSO STANDARDIZED THE DATA USING STANDARD SCALER AS MINMAX SCALER AND STANDARD

SCALER DIDNT AFFECT THE ACCURACY MUCH

- USED RANDOM FOREST FOR GETTING TOP FEATURES

- TRAINED ON MULTIPLE MODELS LIKE LOGISTIC REGRESSION, ADABOOST,DECISION TREE ,NAIVE BAYES,KNN ,SVM

- PERFORMED EDA TO HAVE BETTER ANALYSIS ON DATA

- PLOTTED GRAPHS AND MATRICES TO GET THE ACCURACY OF DATA FOR TERM DEPOSIT SUBSCRIPTION

- AMONG THE ABOVE MODELS DECISION TREE AND ADABOOST WITH DECISION TREE HAS PERMORED WELL WITH RECALL RATE AND PRECISION RATE BOTH HIGH.

- SVM HAS BEEN PERFORMED ON SAMPLE DATA SO PROBABILTY OF GETTING HIGHER SVM ACCURACY ON ACTUAL DATA IS LESS