MH6151 Data Mining Project Source Code

1. Team Members

> **HON JIA YU SAMUEL**
>
> **JIANG LEI**
>
> **LEONG YEW WING**
>
> **LI FENGZHI**

2. Source Code Structure

PART 1: Data Exploration

PART 2: Summary for Categorical data

PART 3: Categorical Attributes Exploration

PART 4: Summary for Continous data

PART 5: Continous Attributes Exploration

PART 6: Categorical: Feature Engineering

PART 7: Continuous: Feature Engineering

PART 8: Unknow/Missing/Outlier Data Handling

PART 9: Data Normalization/Transformation

PART 10: Imbalance Data Handling (with Under Sampling)

PART 11: Modelling: Random Forest Classifier

PART 12: Modelling: KNN

PART 13: Modelling: Logistic Regression

PART 14: Modelling: Neural Network MLP Classifier

3. Important Note

It takes about 15 minutes to run through all the models, the Random Forest Classifier and Neural Network MLPClassifier are very time consuming.

In [48]:

```
######################################################################################
## PART 1: Data Description
######################################################################################

#Import packages
import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels
import scipy
import matplotlib.pyplot as plt
import math
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error
import datawig
from scipy.stats import chisquare

#Import Data, store into data frame df_original
df_original = pd.read_csv('bank-full.csv',sep=";")

# Check top 5 attributes
df_original.head()
```

Out[48]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	nc
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	nc
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	nc
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	nc
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	nc

In [49]:
```
# Summary of data frame
# 45211 entries, 17 features including 1 class feature
# Besides, all these 16 features are either categorical or integer types

df_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
age          45211 non-null int64
job          45211 non-null object
marital      45211 non-null object
education    45211 non-null object
default      45211 non-null object
balance      45211 non-null int64
housing      45211 non-null object
loan         45211 non-null object
contact      45211 non-null object
day          45211 non-null int64
month        45211 non-null object
duration     45211 non-null int64
campaign     45211 non-null int64
pdays        45211 non-null int64
previous     45211 non-null int64
poutcome     45211 non-null object
y            45211 non-null object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In [50]:
```
# checking NA data
missing_data = df_original.isnull().mean()*100

# 0.0 ,no missing data
missing_data.sum()

# no empty/NA data exists, however, we will examine the "unknown" values in the following sections
```

Out[50]: 0.0

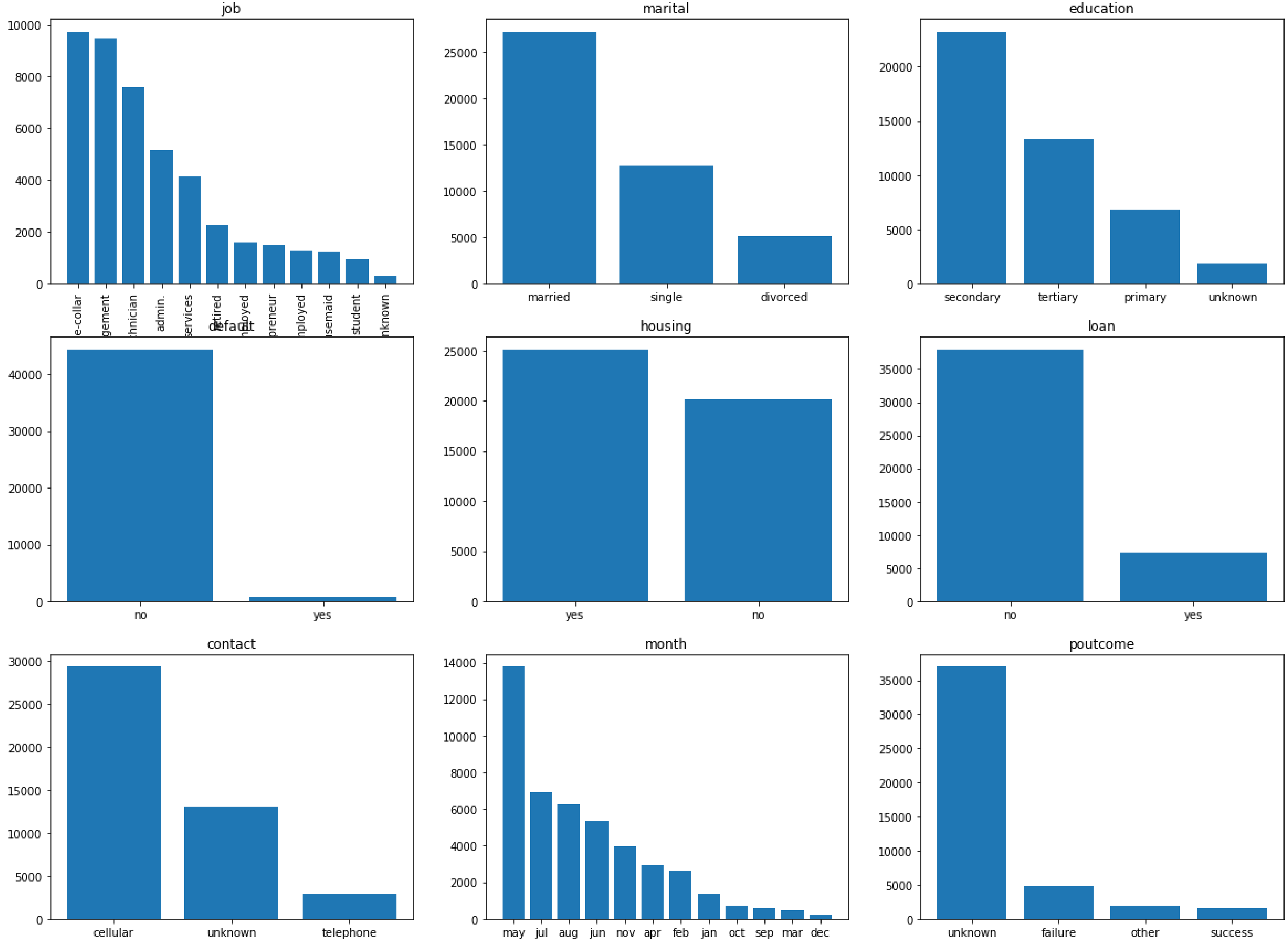
In [51]:

```
##########################################################################################
## PART 2: Summary for categorical data
##########################################################################################
for column in df_original.select_dtypes(include='object').columns:
    df_original[column] = df_original[column].astype('category')
    print(column)
    print(df_original[column].unique())

def PrintDataframeCategoricalSummary(df):
    for column in df.dtypes[df.dtypes == 'category'].index:
        print(df[column].name, df[column].unique())

# Check categorical features with "unknown" value
for column in df_original.dtypes[df_original.dtypes == 'category'].index:
    num_of_unknown = df_original[column].str.contains('unknown').sum()
    print(df_original[column].name, 'unknown: ', num_of_unknown)

### we can see that [job] [education] [contact] [poutcome] these four features have unknown values
### we will deal with the unknown data handling in the following sections
```

```
job
[management, technician, entrepreneur, blue-collar, unknown, ..., services, self-employed, unemployed, housem
aid, student]
Length: 12
Categories (12, object): [management, technician, entrepreneur, blue-collar, ..., self-employed, unemployed,
housemaid, student]
marital
[married, single, divorced]
Categories (3, object): [married, single, divorced]
education
[tertiary, secondary, unknown, primary]
Categories (4, object): [tertiary, secondary, unknown, primary]
default
[no, yes]
Categories (2, object): [no, yes]
housing
[yes, no]
Categories (2, object): [yes, no]
loan
[no, yes]
Categories (2, object): [no, yes]
```

```
contact
[unknown, cellular, telephone]
Categories (3, object): [unknown, cellular, telephone]
month
[may, jun, jul, aug, oct, ..., jan, feb, mar, apr, sep]
Length: 12
Categories (12, object): [may, jun, jul, aug, ..., feb, mar, apr, sep]
poutcome
[unknown, failure, other, success]
Categories (4, object): [unknown, failure, other, success]
y
[no, yes]
Categories (2, object): [no, yes]
job unknown:  288
marital unknown:  0
education unknown:  1857
default unknown:  0
housing unknown:  0
loan unknown:  0
contact unknown:  13020
month unknown:  0
poutcome unknown:  36959
y unknown:  0
```

In [52]:
```
######################################################################################
## PART 3: Categorical Attributes Exploration
######################################################################################

# only for categorical data and excluding class feature
cat_columns = df_original.columns[df_original.dtypes == 'category']
cat_columns = cat_columns.drop(['y'])

fig, axs = plt.subplots(3, 3, sharex=False, sharey=False, figsize=(20, 15))

counter = 0
for column in cat_columns:
    value_counts = df_original[column].value_counts()
    trace_x = counter // 3
    trace_y = counter % 3
    x_pos = np.arange(0, len(value_counts))

    axs[trace_x, trace_y].bar(x_pos, value_counts.values, tick_label = value_counts.index)
    axs[trace_x, trace_y].set_title(column)

    for tick in axs[0, 0].get_xticklabels():
        tick.set_rotation(90)

    counter += 1

plt.show()

# For now,nothing Special for the categorical attributes, we will engineer these features one by one in the fol
```

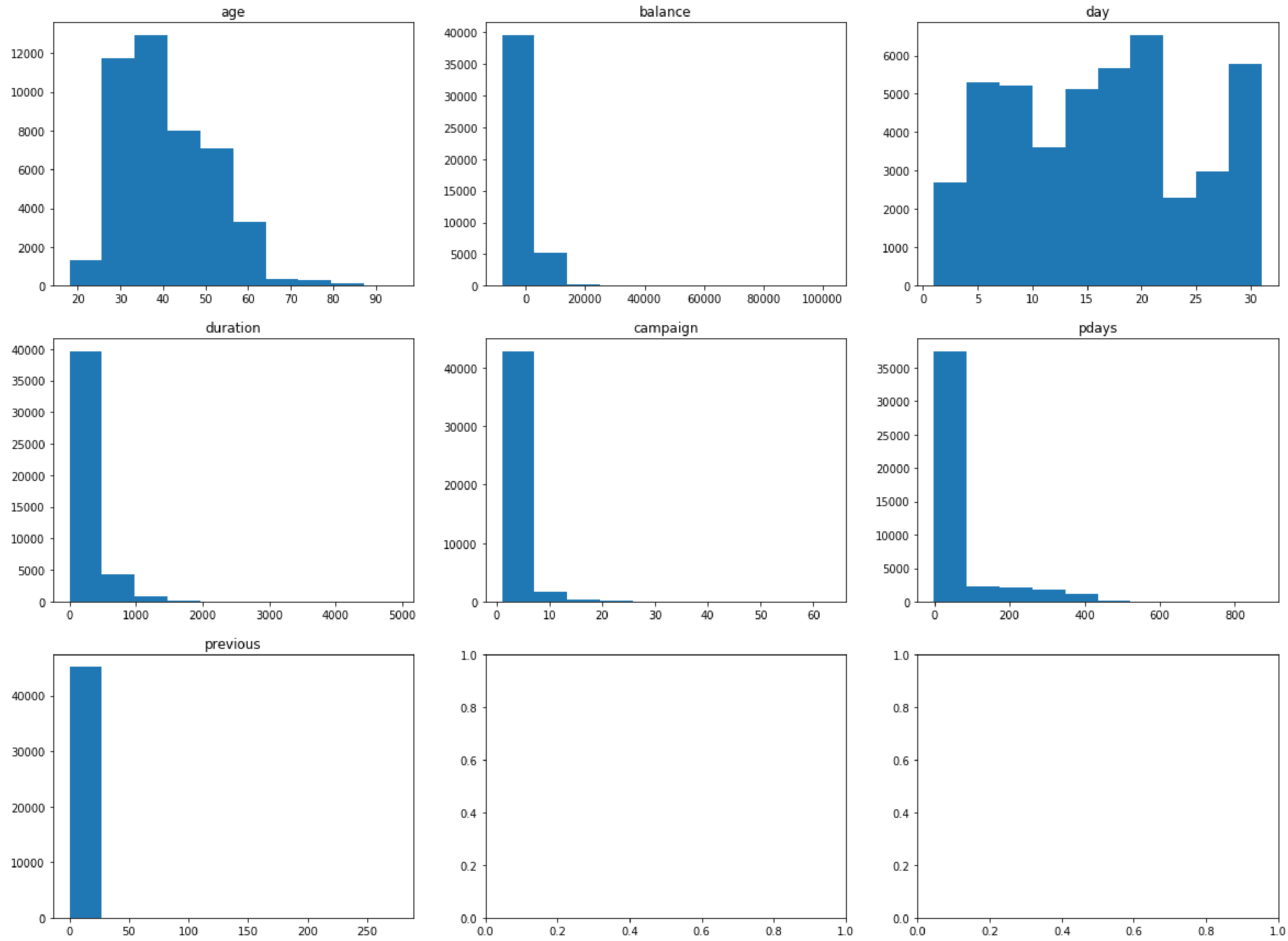

job
e-collar
gement
chnician
admin.
services
etired
ployed
preneur
ployed
semaid
student
nknown
marital
married
single
divorced
education
secondary
tertiary
primary
unknown
default
no
yes
housing
yes
no
loan
no
yes
contact
cellular
unknown
telephone
month
may
jul
aug
jun
nov
apr
feb
jan
oct
sep
mar
dec
poutcome
unknown
failure
other
success

In [53]:
```
########################################################################################
## PART 4: Summary for Continous data
########################################################################################
df_original.describe()
```

Out[53]:

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

In [54]:
```
###############################################################################
## PART 5: Continous Attributes Exploration
###############################################################################
def NumericalHistPlot(df):
    num_columns = df.columns[~(df.dtypes == 'category')]

    fig, axs = plt.subplots(3, 3, sharex=False, sharey=False, figsize=(20, 15))

    counter = 0
    for num_column in num_columns:
        trace_x = counter // 3
        trace_y = counter % 3

        axs[trace_x, trace_y].hist(df[num_column])
        axs[trace_x, trace_y].set_title(num_column)

        counter += 1

    plt.show()

NumericalHistPlot(df_original)

#zoom in the graphs, we noticed that previous and campagin seems have outliers. we will handling the outliers
#in following sections
```


age
balance
day
duration
campaign
pdays
previous

In [55]:

```
##########################################################################################
## PART 6: Categorical: Feature Engineering
##########################################################################################

### [NOTE] We have performed a lot of data exploration using [Tableau] software, these [Tableau]
### explorations are not covered here.

## Feature [poutcome]:
## It has 36959 out of 45211 unknown values, we decide to drop the feature
df_clean = df_original.drop(columns=['poutcome'], errors='ignore')

## Feature [job]:
## It has many categories and some of which are simply duplicate

for jobcat in df_original.job.unique():
    print (jobcat)

## management
## technician
## entrepreneur
## retired
## admin.
## services
## blue-collar
## self-employed
## unemployed
## housemaid
## student

## Below is what we did to reduce the number of categories for [job]

# merge entrepreneur into self_employed
df_clean.job.replace(['entrepreneur', 'self-employed'], 'self-employed', inplace=True)

# merge admin. into management
df_clean.job.replace(['admin.', 'management'], 'management', inplace=True)

# merge technician into blue-collar
df_clean.job.replace(['blue-collar', 'technician'], 'blue-collar', inplace=True)

# merge housemaid into services
```

```
df_clean.job.replace(['services', 'housemaid'], 'services', inplace=True)


## After the merging, we have reduced the number of job categories to 7

## management
## blue-collar
## self-employed
## unemployed
## retired
## services
## student

PrintDataframeCategoricalSummary(df_clean)
```

```
management
technician
entrepreneur
blue-collar
unknown
retired
admin.
services
self-employed
unemployed
housemaid
student
job [management, blue-collar, self-employed, unknown, retired, services, unemployed, student]
Categories (8, object): [management, blue-collar, self-employed, unknown, retired, services, unemployed, stu
dent]
marital [married, single, divorced]
Categories (3, object): [married, single, divorced]
education [tertiary, secondary, unknown, primary]
Categories (4, object): [tertiary, secondary, unknown, primary]
default [no, yes]
Categories (2, object): [no, yes]
housing [yes, no]
Categories (2, object): [yes, no]
loan [no, yes]
Categories (2, object): [no, yes]
```

```
contact [unknown, cellular, telephone]
Categories (3, object): [unknown, cellular, telephone]
month [may, jun, jul, aug, oct, ..., jan, feb, mar, apr, sep]
Length: 12
Categories (12, object): [may, jun, jul, aug, ..., feb, mar, apr, sep]
y [no, yes]
Categories (2, object): [no, yes]
```

In [56]:
```
####################################################################################
## PART 7: Numberical/Continuous: Feature Engineering
####################################################################################

### [NOTE] We have performed a lot of data exploration using [Tableau] software, these [Tableau]
### explorations are not covered here.

# Numerical correlation matrix among numerical features
correlations = df_clean.corr()

print(correlations)

plt.figure(figsize=(15,10))

ax = sns.heatmap(
    correlations,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 320, n=200),
    square=True,
    linewidths=1,
    annot=True
)

ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```

```
               age   balance       day  duration  campaign     pdays  previous
age       1.000000  0.097783 -0.009120 -0.004648  0.004760 -0.023758  0.001288
balance   0.097783  1.000000  0.004503  0.021560 -0.014578  0.003435  0.016674
day      -0.009120  0.004503  1.000000 -0.030206  0.162490 -0.093044 -0.051710
duration -0.004648  0.021560 -0.030206  1.000000 -0.084570 -0.001565  0.001203
campaign  0.004760 -0.014578  0.162490 -0.084570  1.000000 -0.088628 -0.032855
pdays    -0.023758  0.003435 -0.093044 -0.001565 -0.088628  1.000000  0.454820
previous  0.001288  0.016674 -0.051710  0.001203 -0.032855  0.454820  1.000000
```


age
balance
day
duration
campaign
pdays
previous
1
0.098
-0.0091
-0.0046
0.0048
-0.024
0.0013
0.0045
0.022
-0.015
0.0034
0.017
-0.03
0.16
-0.093
-0.052
-0.085
-0.0016
0.0012
-0.089
-0.033
0.45
0.8
0.4
0.0
−0.4
−0.8

In [57]:
```
###############################################################################
## PART 7: Numberical/Continuous: Feature Engineering Continues...
###############################################################################

## Creating new categorical feature [balanceGroup] from [balance] feature which will be removed
df_clean.loc[df_clean['balance'] < 0,'balanceGroup'] = 'negative'
df_clean.loc[(df_clean['balance'] >= 0) & (df_clean['balance'] < 1000),'balanceGroup'] = 'low'
df_clean.loc[(df_clean['balance'] >= 1000) & (df_clean['balance'] < 5000),'balanceGroup'] = 'medium'
df_clean.loc[df_clean['balance'] >= 5000,'balanceGroup'] = 'high'
df_clean['balanceGroup'] = df_clean['balanceGroup'].astype('category')
df_clean = df_clean.drop(columns=['balance'], errors='ignore')

PrintDataframeCategoricalSummary(df_clean)
```

```
job [management, blue-collar, self-employed, unknown, retired, services, unemployed, student]
Categories (8, object): [management, blue-collar, self-employed, unknown, retired, services, unemployed, stud
ent]
marital [married, single, divorced]
Categories (3, object): [married, single, divorced]
education [tertiary, secondary, unknown, primary]
Categories (4, object): [tertiary, secondary, unknown, primary]
default [no, yes]
Categories (2, object): [no, yes]
housing [yes, no]
Categories (2, object): [yes, no]
loan [no, yes]
Categories (2, object): [no, yes]
contact [unknown, cellular, telephone]
Categories (3, object): [unknown, cellular, telephone]
month [may, jun, jul, aug, oct, ..., jan, feb, mar, apr, sep]
Length: 12
Categories (12, object): [may, jun, jul, aug, ..., feb, mar, apr, sep]
y [no, yes]
Categories (2, object): [no, yes]
balanceGroup [medium, low, negative, high]
Categories (4, object): [medium, low, negative, high]
```

In [58]:
```
##########################################################################################
## PART 8: Unknow/Missing/Outlier Data Handling
##########################################################################################
from scipy.stats import chisquare

## We are using Chi-Square testing statistics to measure the importance of [unknown] data below
## The Null Hypothesis H0: Known/Unknow is statistically significant to Response y
## 1) Reject H0:
##         if the Chi-Square P-Value is < threshold P-Value of [0.01],
##         which indicates the unknow is indeed statistically significant to the Response y
## 2) Fail to reject H0: if the Chi-Square P-Value is >= threshold P-Value of [0.01],
##         in which case, we can simply drop these unknown records

## Below is to determine [education]'s unknown's significance to Response y
#          y       n
# known    5037    38317
# unkown   252     1605

known = df_clean[(df_clean['education'] != "unknown")]
unknown = df_clean[(df_clean['education'] == "unknown")]

matrics = [ [ len(known[known['y']=='yes']), len(known[known['y']=='no'])],
       [ len(unknown[unknown['y']=='yes']), len(unknown[unknown['y']=='no']) ] ]
chi2, p, ddof, expected = scipy.stats.chi2_contingency( matrics )
msg = "Test Statistic: {}\np-value: {}\nDegrees of Freedom: {}\n"
print( msg.format( chi2, p, ddof ) )
print( expected )

## Based on Chi-Squared distribution table, p-value = 0.01,
## We fail to reject H0, hence we will have to impute the unknown status of [education].
```

```
Test Statistic: 6.38058198299012
p-value: 0.011537559276897627
Degrees of Freedom: 1

[[ 5071.75921789 38282.24078211]
 [  217.24078211  1639.75921789]]
```

In [59]:
```
## Below is to determine [Job]'s unknown's significance to Response y
#           y       n
# known     5255    39668
# unknown   34      254

known = df_clean[(df_clean['job'] != "unknown")]
unknown = df_clean[(df_clean['job'] == "unknown")]

matrics = [ [ len(known[known['y']=='yes']), len(known[known['y']=='no'])],
       [ len(unknown[unknown['y']=='yes']), len(unknown[unknown['y']=='no']) ] ]
chi2, p, ddof, expected = scipy.stats.chi2_contingency( matrics )
msg = "Test Statistic: {}\np-value: {}\nDegrees of Freedom: {}\n"
print( msg.format( chi2, p, ddof ) )
print( expected )

## Based on Chi-Squared distribution table, The Statistic = 0.001 < 0.01,
## We can reject H0, [job]'s [unknown] records is statistically insignificant.
```

```
Test Statistic: 0.0012421778371059784
p-value: 0.9718847438072241
Degrees of Freedom: 1

[[5.25530838e+03 3.96676916e+04]
 [3.36916237e+01 2.54308376e+02]]
```

In [60]:
```
## Below is to determine [contact]'s unknown's significance to Response y
#            y        n
# known      4759     27432
# unknown    530       12490

known = df_clean[(df_clean['contact'] != "unknown")]
unknown = df_clean[(df_clean['contact'] == "unknown")]

matrics = [ [ len(known[known['y']=='yes']), len(known[known['y']=='no'])],
       [ len(unknown[unknown['y']=='yes']), len(unknown[unknown['y']=='no']) ] ]
chi2, p, ddof, expected = scipy.stats.chi2_contingency( matrics )
msg = "Test Statistic: {}\np-value: {}\nDegrees of Freedom: {}\n"
print( msg.format( chi2, p, ddof ) )
print( expected )

# Based on Chi-Squared distribution table, The Statistic = 1028.93 > 0.01,
## We fail to reject H0, hence we will have to impute the unknown status of [contact].
```

```
Test Statistic: 1028.9314916038188
p-value: 9.24040995646625e-226
Degrees of Freedom: 1

[[ 3765.85784433 28425.14215567]
 [ 1523.14215567 11496.85784433]]
```

In [61]:
```
## Drop Job with 'unknown' based on Chi-Square test conclusion above

df_clean.drop(df_clean[df_clean['job'] == "unknown"].index , inplace=True)
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 44923 entries, 0 to 45210
Data columns (total 16 columns):
age             44923 non-null int64
job             44923 non-null category
marital         44923 non-null category
education       44923 non-null category
default         44923 non-null category
housing         44923 non-null category
loan            44923 non-null category
contact         44923 non-null category
day             44923 non-null int64
month           44923 non-null category
duration        44923 non-null int64
campaign        44923 non-null int64
pdays           44923 non-null int64
previous        44923 non-null int64
y               44923 non-null category
balanceGroup    44923 non-null category
dtypes: category(10), int64(6)
memory usage: 2.8 MB
```

In [62]:
```
## Dealling with Outliers
## [NOTE] We observed the possible outliers for [pdays] [previous] [campaign] from Tableau software

from collections import Counter

def detect_outliers(df, n, features):

    outlier_indices = []

    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col], 75)

        IQR = Q3 - Q1

        outlier_step = 1.5 * IQR

        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step)].index

        outlier_indices.extend(outlier_list_col)

    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n)

    return multiple_outliers

Outliers_to_drop = detect_outliers(df_clean, 2, ["pdays", "previous", "campaign"])
```

In [63]:
```
## Drop outlier records from the result

df_clean = df_clean.copy().drop(Outliers_to_drop, axis=0).reset_index(drop=True)
```

In [64]:
```
## Impute unknown data from [education] and [contact] by using 'most_frequent'

from sklearn.impute import SimpleImputer

imp = SimpleImputer(missing_values="unknown",strategy="most_frequent")
df_clean["education"]=imp.fit_transform(df_clean[["education"]])
df_clean["contact"]=imp.fit_transform(df_clean[["contact"]])
```

In [65]:
```
## Check the education result after imputation
df_clean["education"].value_counts()
```

Out[65]:
```
secondary    24728
tertiary     13202
primary       6784
Name: education, dtype: int64
```

In [66]:
```
#Check the contact result after imputation
df_clean["contact"].value_counts()
```

Out[66]:
```
cellular     41890
telephone     2824
Name: contact, dtype: int64
```

In [67]:
```
###########################################################################################
## PART 9: Data Normalization/Transformation
###########################################################################################

## 9.1: [job] Normalization

# to map each categorical value to a numeric value
labels = df_clean['job'].astype('category').cat.categories.tolist()

replace_map_comp = {'job' : {k: v for k,v in zip(labels,list(range(1,len(labels)+1)))}}

df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[67]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	married	tertiary	no	yes	no	cellular	5	may	261	1	-1	0	no	medium
1	44	2	single	secondary	no	yes	no	cellular	5	may	151	1	-1	0	no	low
2	33	7	married	secondary	no	yes	yes	cellular	5	may	76	1	-1	0	no	low
3	47	2	married	secondary	no	yes	no	cellular	5	may	92	1	-1	0	no	medium
4	35	5	married	tertiary	no	yes	no	cellular	5	may	139	1	-1	0	no	low

In [68]:

```
## 9.2: [marital] Normalization

replace_map_comp = {'marital': {'divorced': 0, 'single': 1, 'married': 2}}
df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[68]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	tertiary	no	yes	no	cellular	5	may	261	1	-1	0	no	medium
1	44	2	1	secondary	no	yes	no	cellular	5	may	151	1	-1	0	no	low
2	33	7	2	secondary	no	yes	yes	cellular	5	may	76	1	-1	0	no	low
3	47	2	2	secondary	no	yes	no	cellular	5	may	92	1	-1	0	no	medium
4	35	5	2	tertiary	no	yes	no	cellular	5	may	139	1	-1	0	no	low

In [69]:

```
## 9.3: [education] Normalization

replace_map_comp = {'education': {'unknown': 0, 'primary': 1, 'secondary': 2, 'tertiary': 3}}
df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[69]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	3	no	yes	no	cellular	5	may	261	1	-1	0	no	medium
1	44	2	1	2	no	yes	no	cellular	5	may	151	1	-1	0	no	low
2	33	7	2	2	no	yes	yes	cellular	5	may	76	1	-1	0	no	low
3	47	2	2	2	no	yes	no	cellular	5	may	92	1	-1	0	no	medium
4	35	5	2	3	no	yes	no	cellular	5	may	139	1	-1	0	no	low

In [70]:
```
## 9.4: [default] Normalization

replace_map_comp = {'default': {'no': 0, 'yes': 1}}
df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[70]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	3	0	yes	no	cellular	5	may	261	1	-1	0	no	medium
1	44	2	1	2	0	yes	no	cellular	5	may	151	1	-1	0	no	low
2	33	7	2	2	0	yes	yes	cellular	5	may	76	1	-1	0	no	low
3	47	2	2	2	0	yes	no	cellular	5	may	92	1	-1	0	no	medium
4	35	5	2	3	0	yes	no	cellular	5	may	139	1	-1	0	no	low

In [71]:
```
## 9.5: [housing] Normalization

replace_map_comp = {'housing': {'no': 0, 'yes': 1}}
df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[71]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	3	0	1	no	cellular	5	may	261	1	-1	0	no	medium
1	44	2	1	2	0	1	no	cellular	5	may	151	1	-1	0	no	low
2	33	7	2	2	0	1	yes	cellular	5	may	76	1	-1	0	no	low
3	47	2	2	2	0	1	no	cellular	5	may	92	1	-1	0	no	medium
4	35	5	2	3	0	1	no	cellular	5	may	139	1	-1	0	no	low

In [72]:

```
## 9.6: [loan] Normalization

replace_map_comp = {'loan': {'no': 0, 'yes': 1}}
df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[72]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	3	0	1	0	cellular	5	may	261	1	-1	0	no	medium
1	44	2	1	2	0	1	0	cellular	5	may	151	1	-1	0	no	low
2	33	7	2	2	0	1	1	cellular	5	may	76	1	-1	0	no	low
3	47	2	2	2	0	1	0	cellular	5	may	92	1	-1	0	no	medium
4	35	5	2	3	0	1	0	cellular	5	may	139	1	-1	0	no	low

In [73]:

```
## 9.7: [contact] Normalization

replace_map_comp = {'contact': {'unknown': 0, 'telephone': 1, 'cellular': 2}}
df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[73]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	3	0	1	0	2	5	may	261	1	-1	0	no	medium
1	44	2	1	2	0	1	0	2	5	may	151	1	-1	0	no	low
2	33	7	2	2	0	1	1	2	5	may	76	1	-1	0	no	low
3	47	2	2	2	0	1	0	2	5	may	92	1	-1	0	no	medium
4	35	5	2	3	0	1	0	2	5	may	139	1	-1	0	no	low

In [74]:
```
## 8. Month Normalization
replace_map_comp = {'month': {'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4, 'may': 5, 'jun': 6, 'jul': 7, 'aug': 8, '
df_clean.replace(replace_map_comp, inplace=True)
# # data.info()

df_clean.head()
```

Out[74]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	3	0	1	0	2	5	5	261	1	-1	0	no	medium
1	44	2	1	2	0	1	0	2	5	5	151	1	-1	0	no	low
2	33	7	2	2	0	1	1	2	5	5	76	1	-1	0	no	low
3	47	2	2	2	0	1	0	2	5	5	92	1	-1	0	no	medium
4	35	5	2	3	0	1	0	2	5	5	139	1	-1	0	no	low

In [75]:
```
## 9.8: [balanceGroup] Normalization

replace_map_comp = {'balanceGroup': {'negative': 0, 'low': 1, 'medium': 2, 'high': 3}}
df_clean.replace(replace_map_comp, inplace=True)

df_clean.head()
```

Out[75]:

	age	job	marital	education	default	housing	loan	contact	day	month	duration	campaign	pdays	previous	y	balanceGroup
0	58	5	2	3	0	1	0	2	5	5	261	1	-1	0	no	2
1	44	2	1	2	0	1	0	2	5	5	151	1	-1	0	no	1
2	33	7	2	2	0	1	1	2	5	5	76	1	-1	0	no	1
3	47	2	2	2	0	1	0	2	5	5	92	1	-1	0	no	2
4	35	5	2	3	0	1	0	2	5	5	139	1	-1	0	no	1

In [76]:
```
###############################################################################
## PART 10: Imbalance Data Handling (with Under Sampling)
###############################################################################

# Class count
count_class_0, count_class_1 = df_clean['y'].value_counts()
count_class_0, count_class_1
```

Out[76]:
```
(39472, 5242)
```

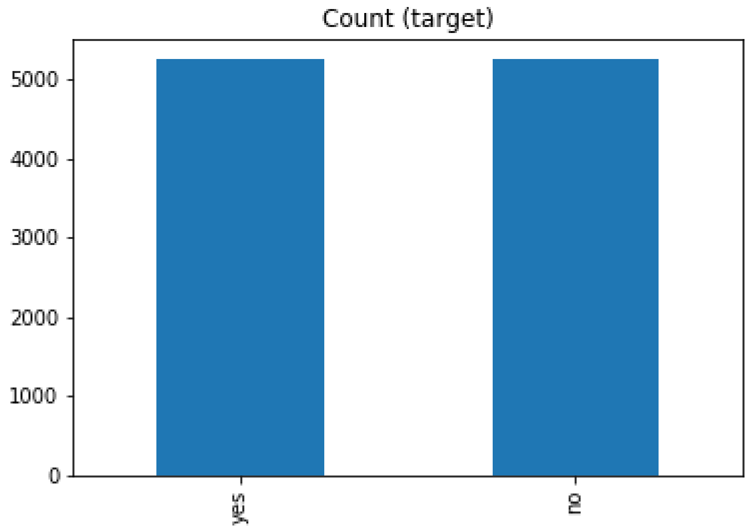
In [77]:
```
## The responses in the data are 90% “no” and 10% “yes”, obviously a significantly imbalanced dataset.
## Procced to Resampling

## Split by class label first
df_class_0 = df_clean[df_clean['y'] == 'no']
df_class_1 = df_clean[df_clean['y'] == 'yes']
```

In [78]:
```
## Perform Random Under-sampling on class 0

df_class_0_under = df_class_0.sample(count_class_1)
df_test_under = pd.concat([df_class_0_under, df_class_1], axis=0)
```

In [79]:
```
## Check after under sampling of minorities to confirm the results

print('After Random under-sampling:')
print(df_test_under['y'].value_counts())
```

```
After Random under-sampling:
yes    5242
no     5242
Name: y, dtype: int64
```

In [80]:
```
df_test_under['y'].value_counts().plot(kind='bar', title='Count (target)');
```



In [81]:
```
########Seperate Data to Training and Testing
from sklearn.model_selection import train_test_split

#seperate data, get training and testing data
data_under = df_test_under.drop('y', axis=1)
labels_under = df_test_under["y"].copy()
train, test, train_labels, test_labels = train_test_split(data_under, labels_under, test_size=0.2, random_state

## checking training data
len(labels_under)
```

Out[81]: 10484

In [102]:
```
######################################################################################
## PART 11: Modelling: [Random Forest Classifier]
######################################################################################

from sklearn.ensemble import RandomForestClassifier

## Ramdom FOREST Model
clf = RandomForestClassifier(
    n_estimators=200,
    criterion='gini',
    max_depth=5,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features='auto',
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=-1,
    random_state=0,
    verbose=0,
    warm_start=False,
    class_weight='balanced'
)
```

In [103]:
```
from sklearn.model_selection import GridSearchCV
## 5 Fold cross validation to get the best model

## Compute grid search to find best paramters for pipeline
tuned_parameter = [{'max_depth':range(3,7), 'n_estimators':[50,100,200,300]}]
grid_search = GridSearchCV(clf, param_grid=tuned_parameter, cv=5).fit(train, train_labels)

grid_search.best_score_
```

Out[103]: 0.8265172290449505

In [104]:
```
from sklearn.model_selection import cross_val_score

print ('on train set after resampling')
scores = cross_val_score(grid_search.best_estimator_, train, train_labels,cv=5,scoring='accuracy')
print (scores.mean(),scores)
```

```
on train set after resampling
0.8265177574744398 [0.82359952 0.83015495 0.81644815 0.84078712 0.82159905]
```

In [105]:
```
## Predication On Testig Data

## To Use the best estimator to predict Test Set after Sampling
print(classification_report(test_labels, grid_search.best_estimator_.predict(test)))
```

```
              precision    recall  f1-score   support

          no       0.83      0.78      0.80      1034
         yes       0.80      0.85      0.82      1063

    accuracy                           0.81      2097
   macro avg       0.81      0.81      0.81      2097
weighted avg       0.81      0.81      0.81      2097
```

In [106]:
```
## Confusion Matrix
test_pred = grid_search.best_estimator_.predict(test)
confusion_matrix(test_labels, test_pred)
```

Out[106]:
```
array([[803, 231],
       [163, 900]])
```

In [107]:
```
## Accuracy
from sklearn.metrics import accuracy_score,f1_score
accuracy_score(test_labels, test_pred)
```

Out[107]:
```
0.8121125417262757
```

In [108]:
```
## Classification Report
print ('Classification Report:\n', classification_report(test_labels, test_pred))
```

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

          no       0.83      0.78      0.80      1034
         yes       0.80      0.85      0.82      1063

    accuracy                           0.81      2097
   macro avg       0.81      0.81      0.81      2097
weighted avg       0.81      0.81      0.81      2097
```

In [109]:
```
## F1
print ('F1 score:', f1_score(test_labels, test_pred,average="macro"))
```

```
F1 score: 0.8117096627164995
```

In [110]:
```
## MCC
from sklearn.metrics import matthews_corrcoef
mcc = matthews_corrcoef(test_labels, test_pred)
mcc
```

Out[110]:
```
0.625134692890111
```

In [111]:
```
################################################################################
## PART 12: Modelling: [KNN]
################################################################################

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

## Neighbors
clf = KNeighborsClassifier(n_neighbors =25, weights='uniform', p=2, metric='euclidean')
```

In [112]:
```
## Compute grid search to find best paramters for pipeline
tuned_parameter = [{'n_neighbors':[10,20,30,40,50]}]

grid_search = GridSearchCV(clf, param_grid=tuned_parameter, cv=5).fit(train, train_labels)
grid_search.best_score_, grid_search.best_params_
```

Out[112]:
```
(0.7807320853702158, {'n_neighbors': 40})
```

In [113]:
```
## Predication On Testig Data
## Confusion Matrix
test_pred = grid_search.best_estimator_.predict(test)
confusion_matrix(test_labels, test_pred)
```

Out[113]:
```
array([[864, 170],
       [287, 776]])
```

In [114]:
```
## Accuracy
from sklearn.metrics import accuracy_score,f1_score
accuracy_score(test_labels, test_pred)
```

Out[114]:
```
0.78206962327134
```

In [115]:
```
## Classification Report
print ('Classification Report:\n', classification_report(test_labels, test_pred))
```

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

          no       0.75      0.84      0.79      1034
         yes       0.82      0.73      0.77      1063

    accuracy                           0.78      2097
   macro avg       0.79      0.78      0.78      2097
weighted avg       0.79      0.78      0.78      2097
```

In [116]:
```
## F1
print ('F1 score:', f1_score(test_labels, test_pred,average="macro"))
```

```
F1 score: 0.7816851627629899
```

In [117]:
```
## MCC
mcc = matthews_corrcoef(test_labels, test_pred)
mcc
```

Out[117]:
```
0.5682671780175735
```

In [125]:
```
##########################################################################################
## PART 13: Modelling: [Logistic Regression]
##########################################################################################

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.metrics import matthews_corrcoef, roc_curve, roc_auc_score, auc, accuracy_score, confusion_matrix,

import collections

## define the function
def Run_LogisticRegression(train_x, train_y):
    five_fold = KFold(n_splits=5, random_state=1337)

    params=[{'penalty':['l1','l2']}]

    logitRegression = GridSearchCV(LogisticRegression(tol=1e-4), params, cv=5)
    gscv_lr = logitRegression.fit(train, train_labels)

    myscore = cross_val_score(gscv_lr.best_estimator_, train_x, train_y, cv=five_fold, scoring='accuracy')
    print(myscore)

    pred = logitRegression.predict(test)
    print('accuracy = ', accuracy_score(test_labels, pred))
    print('confusion_matrix =\n ', confusion_matrix(test_labels, pred))
    print('MCC = ', matthews_corrcoef(test_labels, pred))
```

In [126]:
```
## Run logistic regression
Run_LogisticRegression(train, train_labels)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

```
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver w
ill be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
[0.78188319 0.80393325 0.78950507 0.80918306 0.78115683]
accuracy =  0.7758702908917501
confusion_matrix =
  [[820 214]
 [256 807]]
MCC =  0.5523461110751036
```

In [127]:
```
###############################################################################
## PART 14: Modelling: [Neural Network]
###############################################################################

###############################################################################
# !!!!!!!! [NOTE] This model takes a lot of time to run
###############################################################################

from sklearn.neural_network import MLPClassifier

## define the function to run the model
def Run_NeuralNetwork(train_x, train_y):

    five_fold = KFold(n_splits=5, random_state=1337)

    ###########################################################################
    ## The following params for GridSearchCV has been commented
    ## Because it takes TOOOOOOOOOOOO long in my laptop to find the best params
    ###########################################################################
    # params = {'solver': ['lbfgs', 'sgd', 'adam'], 'max_iter': [500, 700, 900, 1100, 1300,1500],
    #           'alpha': 10.0 ** -np.arange(1, 5), 'hidden_layer_sizes':np.arange(2, 10),
    #           'activation':['logistic','tanh']}
    ###########################################################################

    ## [NOTE] I chose the following setup because it is solvable by my Laptop's computation power
    params = {'hidden_layer_sizes':np.arange(2, 10), 'max_iter': [500]}
    nn_clf = GridSearchCV(MLPClassifier(), params, cv=5)
    gscv_nn_clf = nn_clf.fit(train_x, train_y)

    myscore = cross_val_score(gscv_nn_clf.best_estimator_, train_x, train_y, cv=five_fold, scoring='accuracy')
    print(myscore)

    pred = nn_clf.predict(test)

    print('accuracy = ', accuracy_score(test_labels, pred))
    print('confusion_matrix =\n ', confusion_matrix(test_labels, pred))
    print('MCC = ', matthews_corrcoef(test_labels, pred))
```

In [128]:
```
## Run the Model
Run_NeuralNetwork(train, train_labels)
```

```
[0.78247914 0.80333731 0.7954681  0.80799046 0.7757901 ]
accuracy =  0.7749165474487363
confusion_matrix =
  [[763 271]
 [201 862]]
MCC =  0.5505619050916242
```

--The End ---