AN ANALYSIS OF PORTUGUESE BANK MARKETING DATA

The George Washington University (DATS 6103: An Introduction to Data Mining)

TEAM 11: Anjali Mudgal , Guoshan Yu and Medhasweta Sen

December 20, 2022

INTRODUCTION

Bank marketing is the practice of attracting and acquiring new customers through traditional media and digital media strategies. The use of these media strategies helps determine what kind of customer is attracted to a certain institutions. This also includes different banking institutions purposefully using different strategies to attract the type of customer they want to do business with.

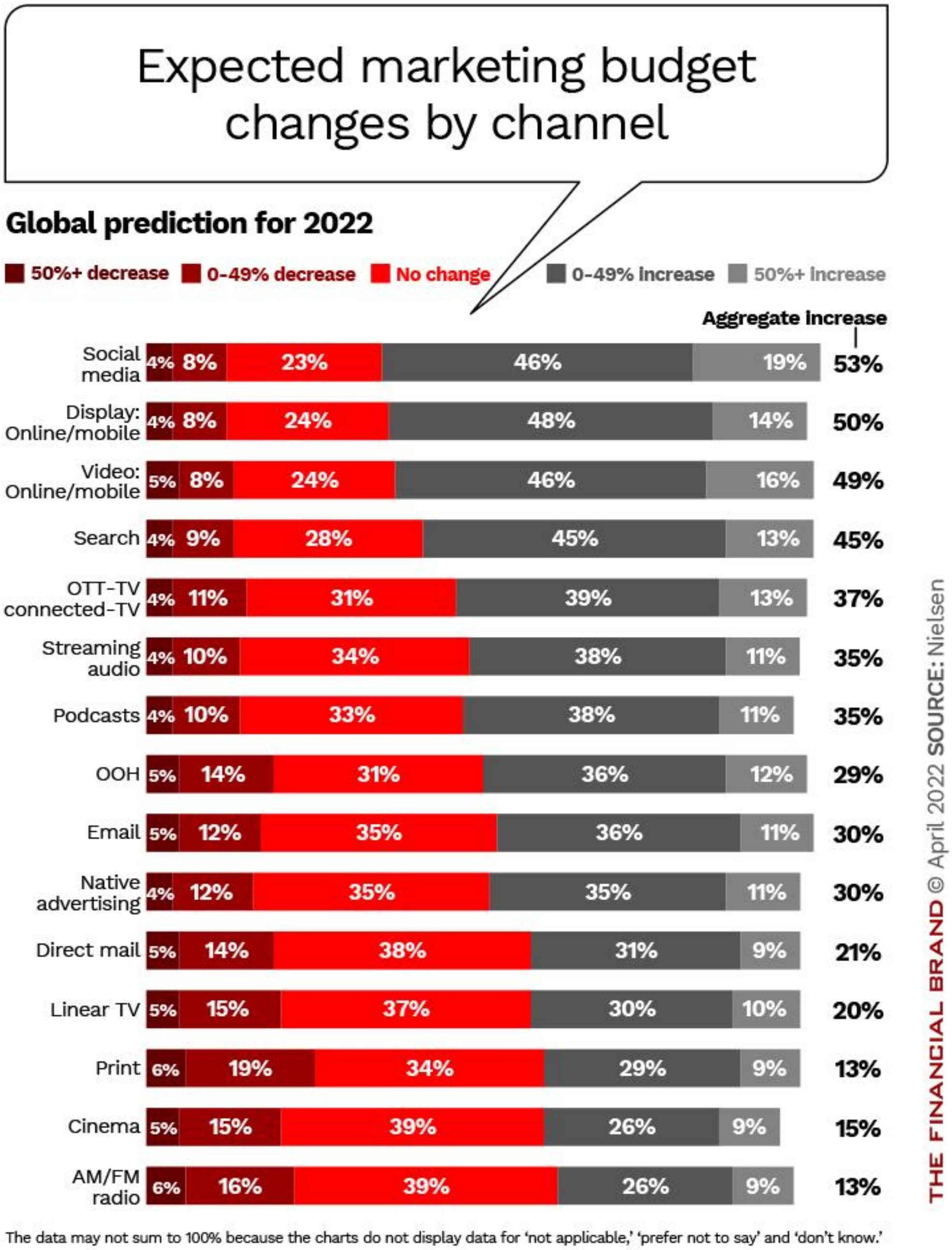
As a discipline, marketing has evolved over the past few decades to become what it is today. Earlier, marketing strategies were primarily a means of spreading brand awareness. Today, marketing has been reinvented to fit a much bigger role. Creating both value and revenue to the institution. It is a big step up from its previous communication role, no doubt. One that was necessitated by the evolution of three factors: the consumer, the technology, and data analytics.

Marketing has evolved from a communication role to a revenue generating role. The consumer has evolved from being a passive recipient of marketing messages to an active participant in the marketing process. Technology has evolved from being a means of communication to a means of data collection and analysis. Data analytics has evolved from being a means of understanding the consumer to a means of understanding the consumer and the institution.

Bank marketing strategy is increasingly focused on digital channels, including social media, video, search and connected TV. As bank and credit union marketers strive to promote brand awareness, they need a new way to assess channel ROI and more accurate data to enable personalized offers. Add to that the growing importance of purpose-driven marketing.

The relentless pace of digitization is disrupting not only the established order in banking, but bank marketing strategies. Marketers at both traditional institutions and digital disruptors are feeling the pressure.

Just as bank marketers begin to master one channel, consumers move to another. Many now toggle between devices on a seemingly infinite number of platforms, making it harder than ever for marketers to pin down the right consumers at the right time in the right place.



The Data Set

The data set used in this analysis is from a Portuguese bank. The data set contains 41,188 observations and 21 variables. The variables include the following:

1.

- age (numeric)

2.

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

3.

- marital : marital status (categorical: ‘divorced’,‘married’,‘single’,‘unknown’; note: ‘divorced’ means divorced or widowed)

4.

- education (categorical: ‘basic.4y’,‘basic.6y’,‘basic.9y’,‘high.school’,‘illiterate’,‘professional.course’,‘university.degree’,‘unknown’)

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’)

8.

- contact: contact communication type (categorical: ‘cellular’,‘telephone’)

9.

- month: last contact month of year (categorical: ‘jan’, ‘feb’, ‘mar’, …, ‘nov’, ‘dec’)

10.

- day_of_week: last contact day of the week (categorical: ‘mon’,‘tue’,‘wed’,‘thu’,‘fri’)

11.

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

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')

16.

- emp.var.rate: employment variation rate - quarterly indicator (numeric)

17.

- cons.price.idx: consumer price index - monthly indicator (numeric)

18.

- cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19.

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

20.

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

21.

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

22.

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

The SMART Questions

The SMART questions are as follows:

1.Relationship between subscribing the term deposit and how much the customer is contacted (last contact, Campaign, Pdays, Previous Number of contacts) 2.Since the dataset is imbalanced, will down sampling/up sampling or other techniques improve upon the accuracy of models. 3.Marital status, age, job, and loan to find out the financially stable population?Will that affect the outcome? 4.Effect of dimensionality reduction on accuracy of the model. 5.The optimal cut off value for classification of our imbalance dataset. 6.Modeling to estimate the potential population who would subscribe to termdeposit. 7. How are the likelihood of subscriptions affected by social and economic factors?

As per the comments, 2 and 6 are more of analysis than comments so they would be covered in our analysis.

Throughout the paper we would try to answer the questions

Importing the libraries

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
%matplotlib inline
import warnings

from scipy.stats import zscore
import seaborn as sns
import scipy.stats as stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
import statsmodels.stats.multicomp as mc
import statsmodels.stats.outliers_influence as influence
import statsmodels.stats.diagnostic as diag
import statsmodels.stats.stattools as stattools
import statsmodels.stats.anova as anova
import statsmodels.stats.weightstats as weightstats
import statsmodels.stats.libqsturng as libqsturng
import statsmodels.stats.power as power
import statsmodels.stats.proportion as proportion
import statsmodels.stats.contingency_tables as contingency_tables
import statsmodels.stats.multitest as multitest
import statsmodels.stats.diagnostic as diagnostic
import statsmodels.stats.correlation_tools as correlation_tools
from statsmodels.formula.api import ols
import researchpy as rp
import scipy.stats as stats
import seaborn as sns
# Import label encoder
from sklearn import preprocessing
warnings.filterwarnings('ignore')
```

```
sns.set_theme(style="whitegrid")

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans

from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import precision_recall_curve
```

Importing the dataset

```
inputFile = "Dataset/primary.csv"
df = pd.read_csv(inputFile)
```

Basic Information about the data

```
print(f"Shape of dataset is : {df.shape}")
print(f"Columns in dataset \n {df.info()}")

Shape of dataset is : (45211, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 23 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             45211 non-null  int64
 1   job             45211 non-null  object
 2   marital         45211 non-null  object
 3   education       45211 non-null  object
 4   default         45211 non-null  object
 5   balance         45211 non-null  int64
 6   housing         45211 non-null  object
 7   loan            45211 non-null  object
```

```
 8   contact          45211 non-null  object
 9   day              45211 non-null  int64
 10  month            45211 non-null  object
 11  duration         45211 non-null  int64
 12  campaign         45211 non-null  int64
 13  pdays            45211 non-null  int64
 14  previous         45211 non-null  int64
 15  poutcome         45211 non-null  object
 16  y                45211 non-null  int64
 17  month_int        45211 non-null  int64
 18  cons.conf.idx    45211 non-null  float64
 19  emp.var.rate     45211 non-null  float64
 20  euribor3m        45211 non-null  float64
 21  nr.employed      45211 non-null  float64
 22  cons.price.idx   45211 non-null  float64
dtypes: float64(5), int64(9), object(9)
memory usage: 7.9+ MB
Columns in dataset
 None
```

Here, we have 45211 variables and 23 columns.

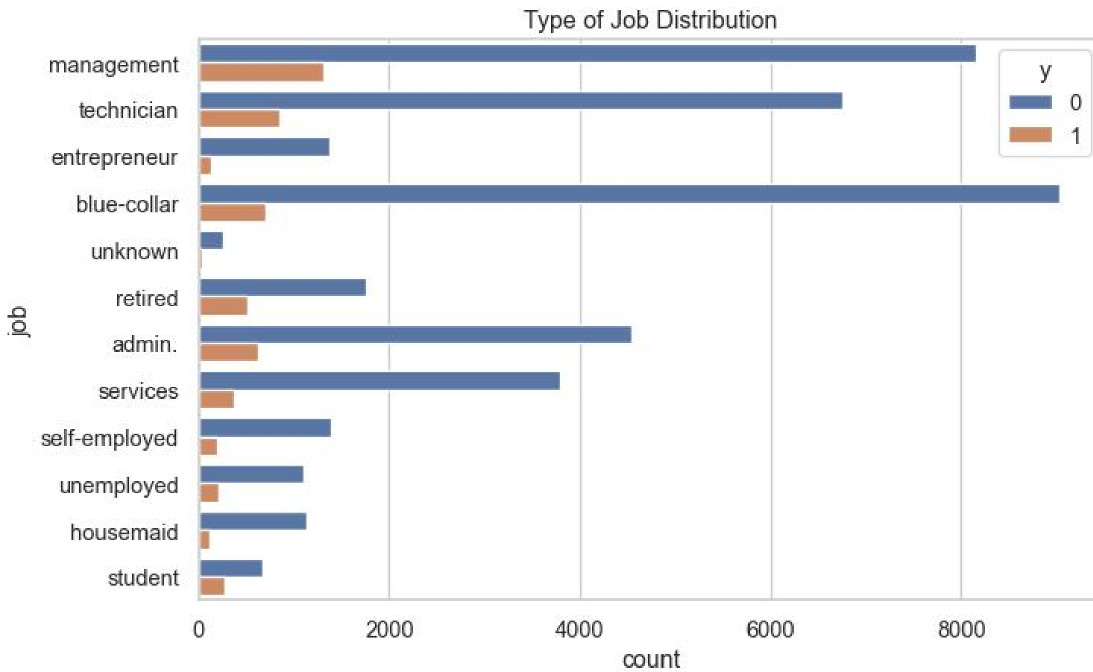
Exploratory Data Analysis (EDA)

Here we would explore the variables which might be important for subscription of term deposits.

Analysing the variables

Job Description

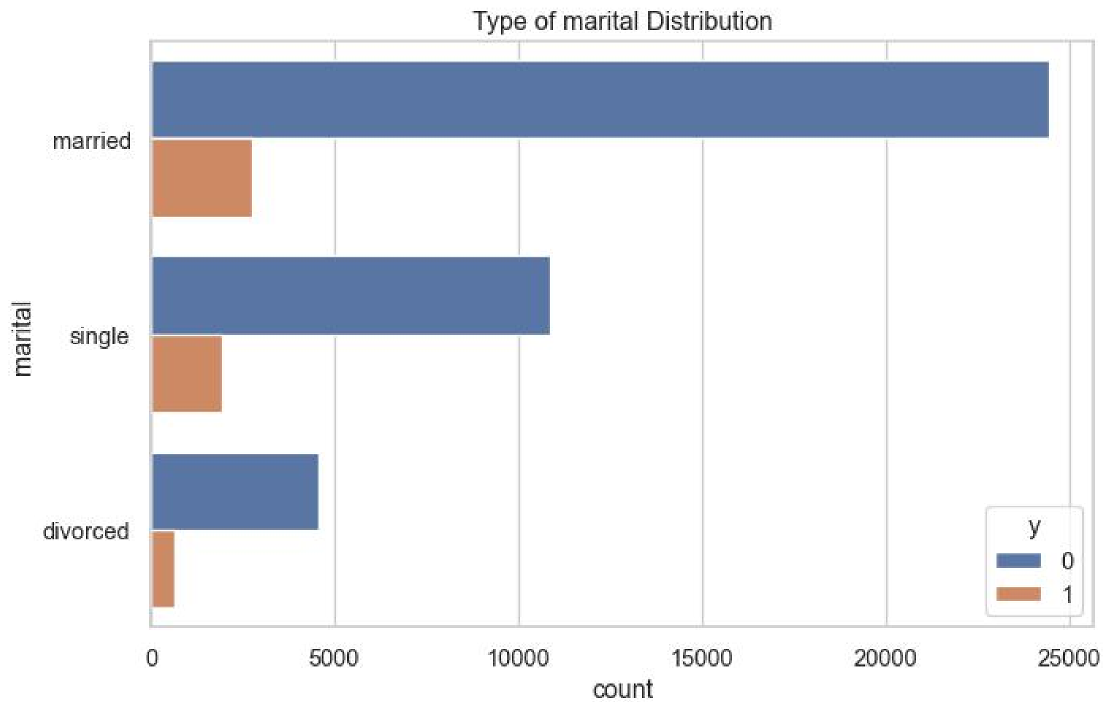
```
# JOB
plt.figure(figsize = (8, 5))
sns.countplot(data=df,y='job',hue='y')
plt.title("Type of Job Distribution")

Text(0.5, 1.0, 'Type of Job Distribution')
```



People in management, technical are more likely to subscibe to the term deposit
So we will explore them later.

Marital

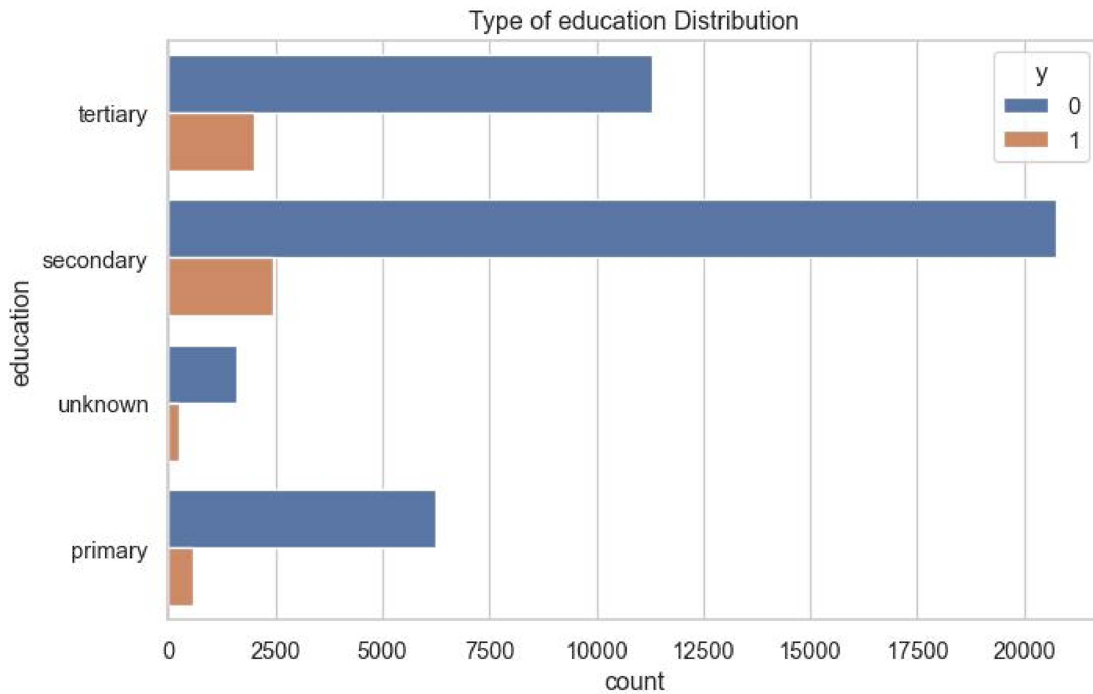
```
# MARITAL
plt.figure(figsize = (8, 5))
sns.countplot(data=df,y='marital',hue='y')
plt.title("Type of marital Distribution")

Text(0.5, 1.0, 'Type of marital Distribution')
```



Married and Single are more likely to subscribe for term deposits rather than divorced. But this might also be because of less number of people being divorced in total.

```
# EDUCATION
plt.figure(figsize = (8, 5))
sns.countplot(data=df,y='education',hue='y')
plt.title("Type of education Distribution")
```

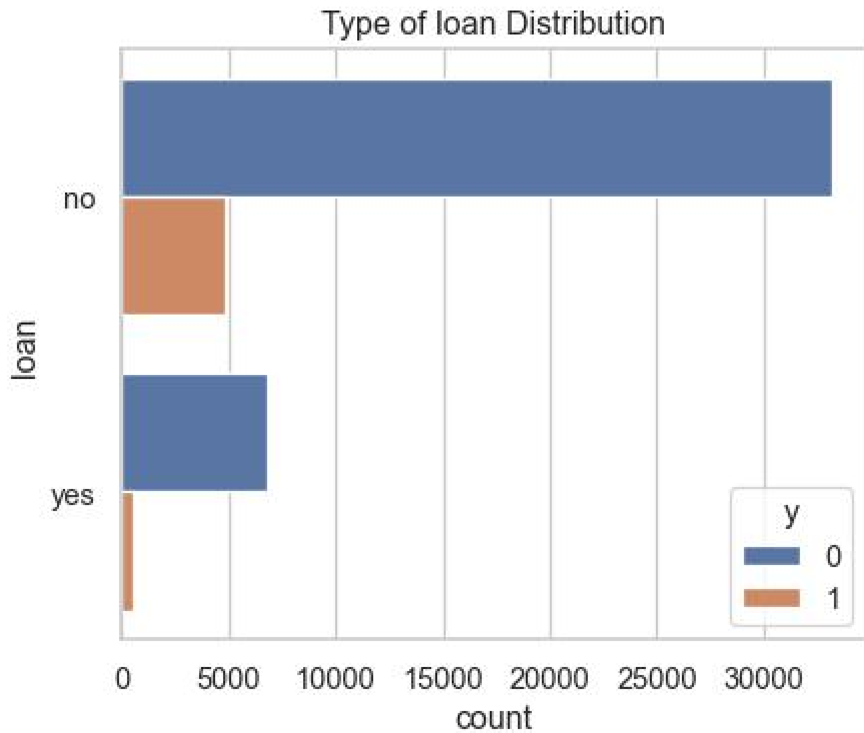
```
Text(0.5, 1.0, 'Type of education Distribution')
```



There are unknown values in education that we need to get rid of.

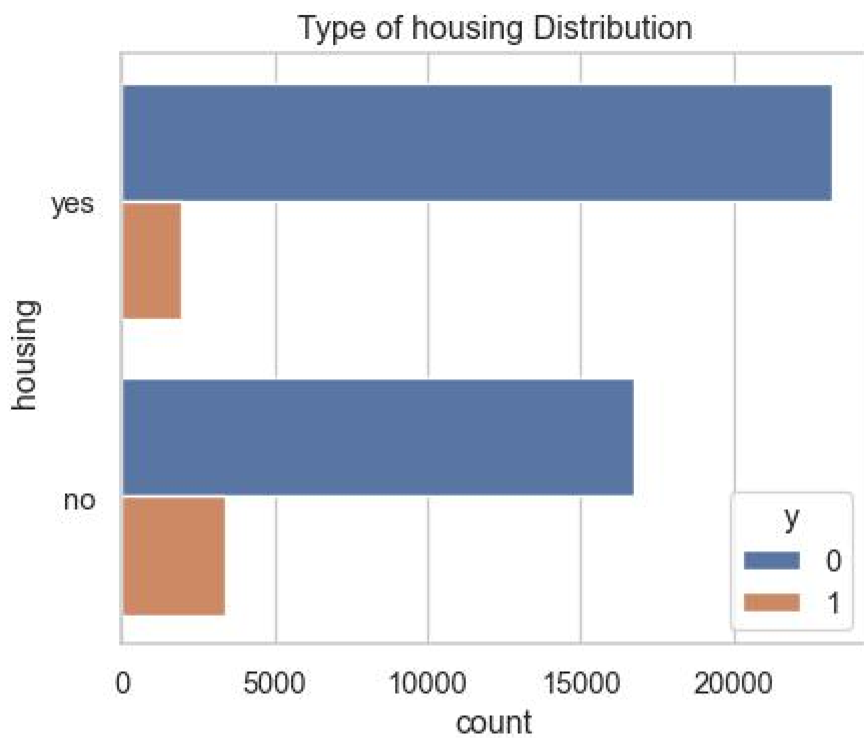
Loan

```
# Loan
sns.countplot(data=df,y='loan',hue='y')
plt.title("Type of loan Distribution")

Text(0.5, 1.0, 'Type of loan Distribution')
```



People with personal loans are less likely to subscribe to term deposit but the difference here is not huge.

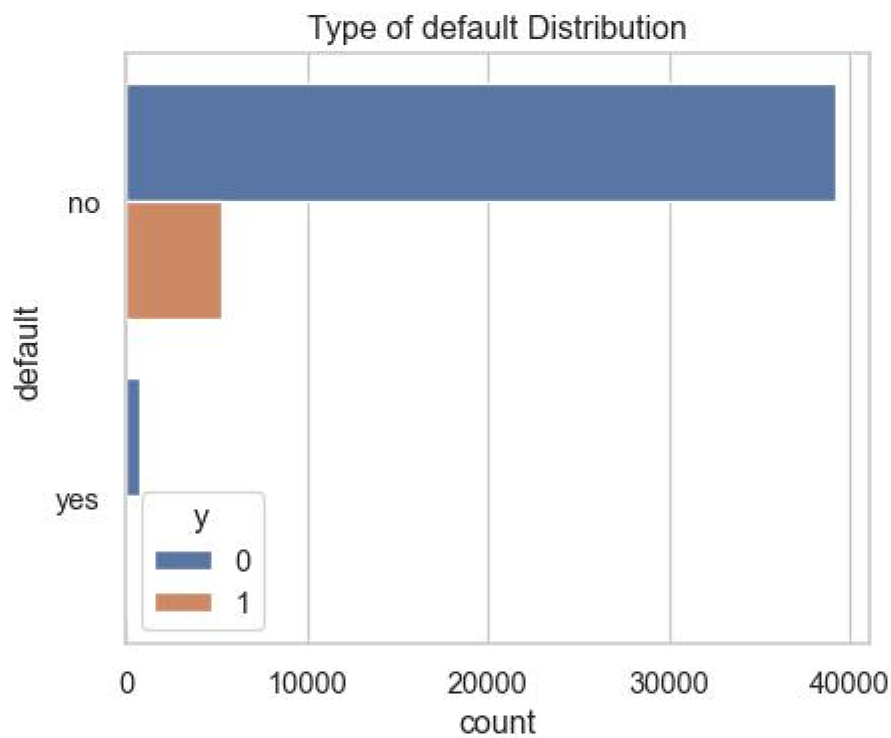
```
# Housing Loan
sns.countplot(data=df,y='housing',hue='y')
plt.title("Type of housing Distribution")

Text(0.5, 1.0, 'Type of housing Distribution')
```



People with housing loans are less likely to subscribe to term deposit but the difference here is not huge.

```
# DEFAULT
sns.countplot(data=df,y='default',hue='y')
plt.title("Type of default Distribution")
```

```
Text(0.5, 1.0, 'Type of default Distribution')
```

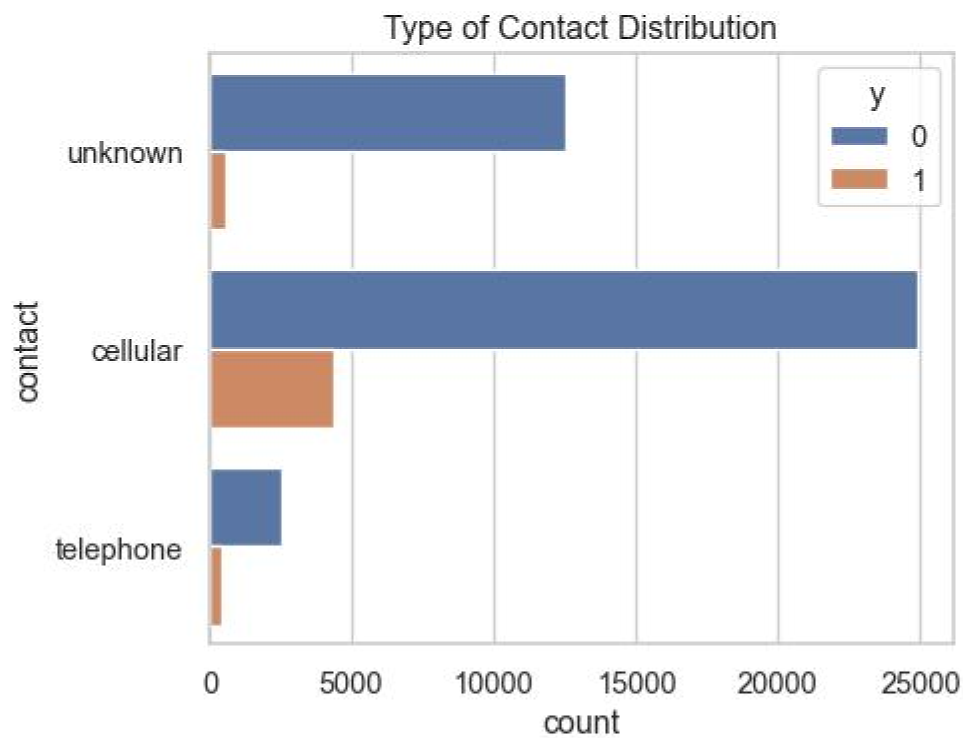


So people who have not paid back there loans and have credits, have not subcribed to the term deposit.

- people who have loans are subscribing to term deposit less.

Contact

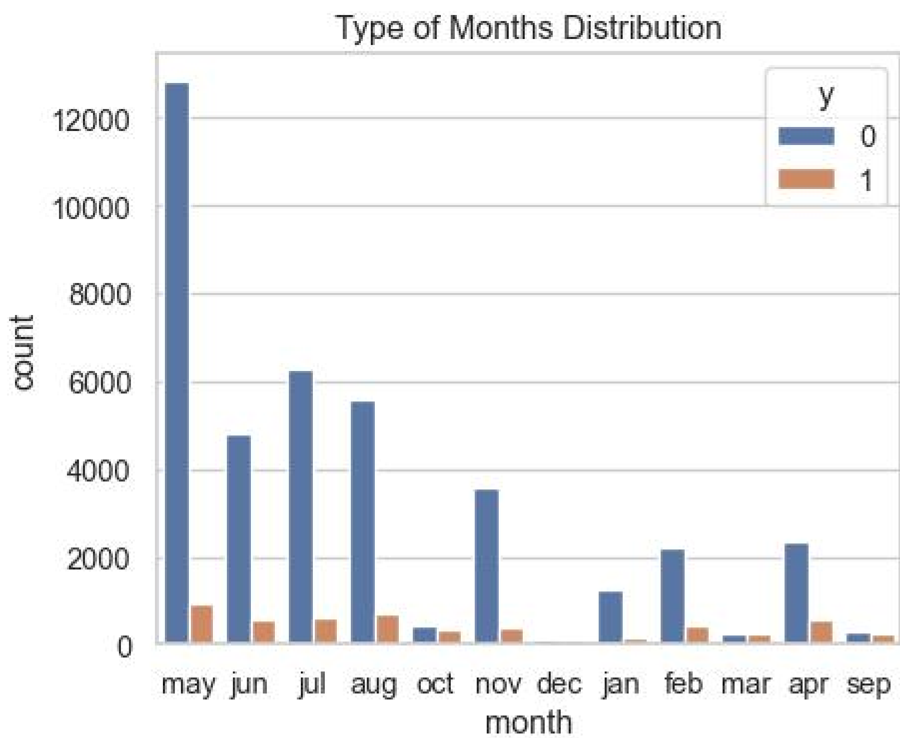
```
# CONTACT
sns.countplot(data=df,y='contact',hue='y')
plt.title("Type of Contact Distribution")

Text(0.5, 1.0, 'Type of Contact Distribution')
```



- since the type of communication(cellular and telephone) is not really a good indicator of subcription, we drop this variable.

Month

```
# MONTH
sns.countplot(x ='month',hue='y', data = df)
plt.title("Type of Months Distribution")

Text(0.5, 1.0, 'Type of Months Distribution')
```



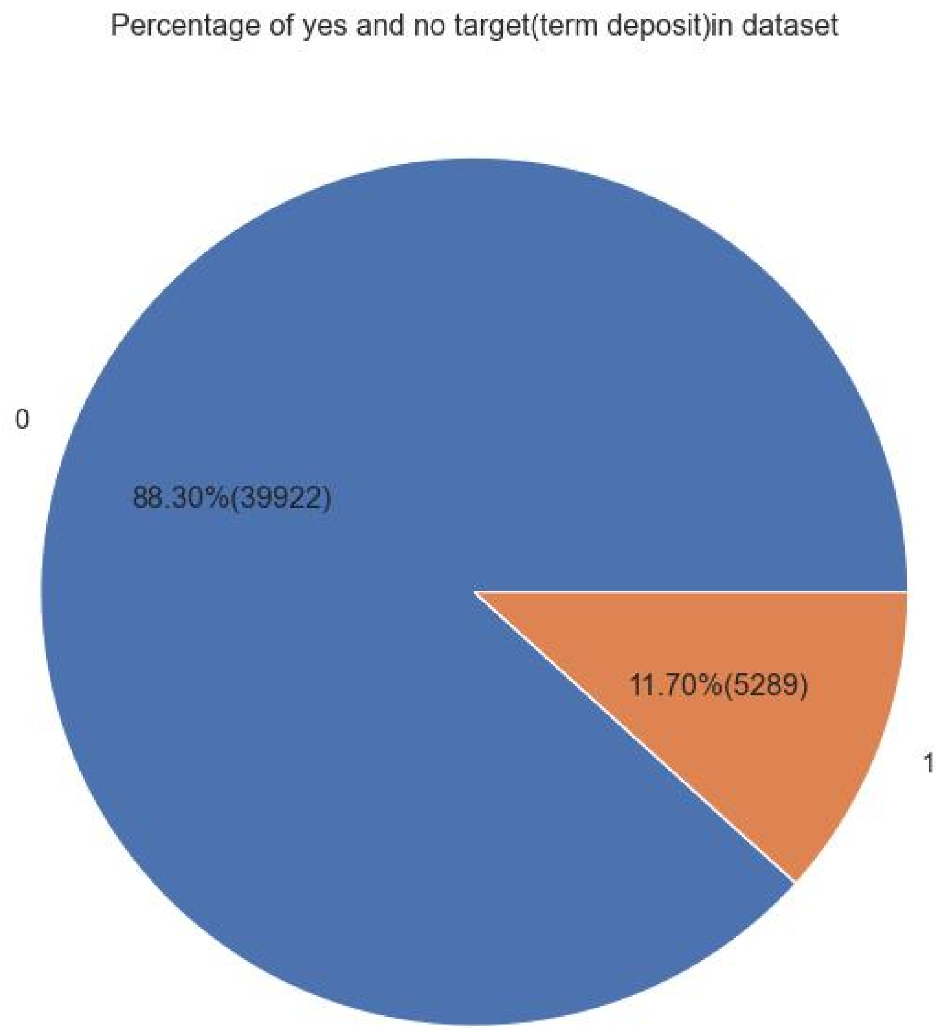
Here,we see that most number of term deposit subscription is in the month of may while the least in the month of december.

```
def pieChart(x_var,title):
    yesNo = df.groupby(x_var).size()
    yesNo.plot(kind='pie', title=title, figsize=[8,8],
          autopct=lambda p: '{:.2f}%({:.0f})'.format(p,(p/100)*yesNo.su
m()))
    plt.show()
```

Term Deposit

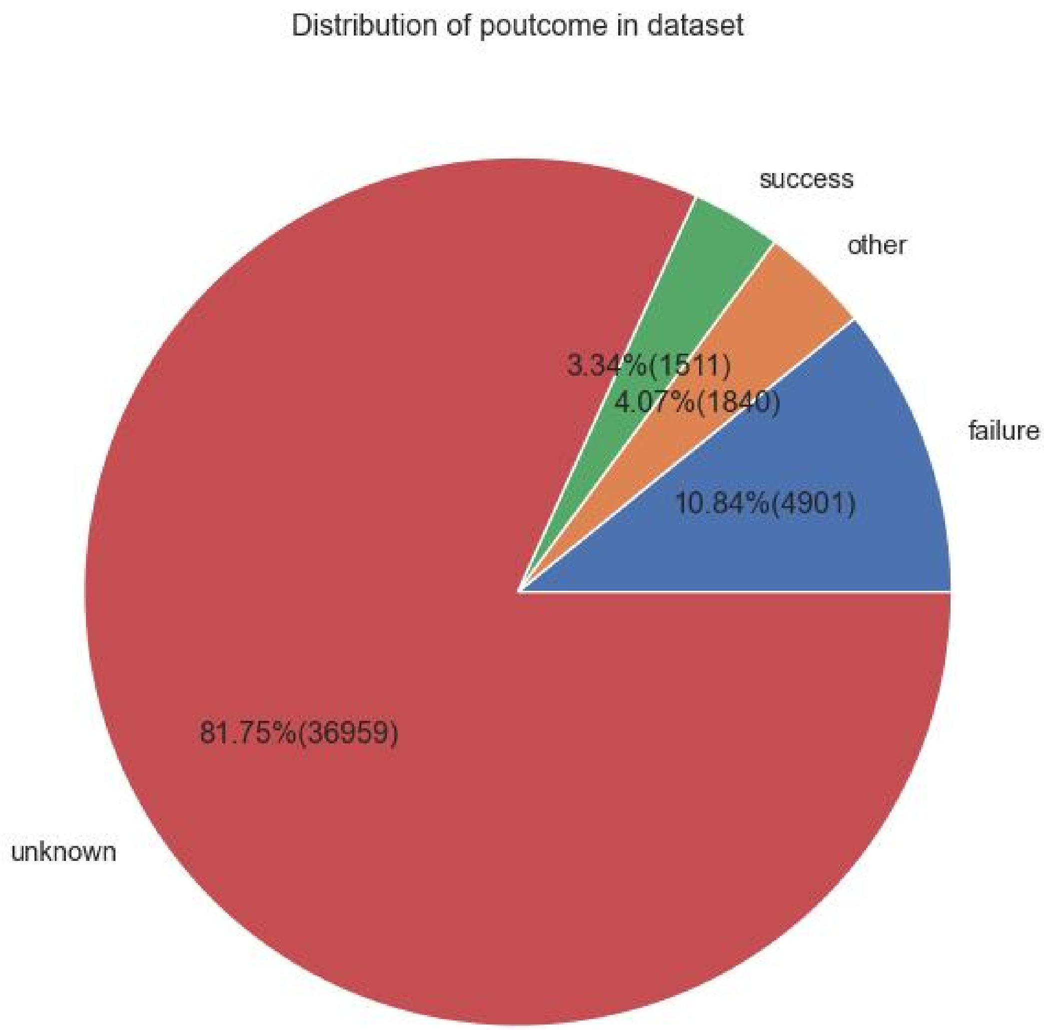
Distribution of y(target) variable

```
pieChart('y','Percentage of yes and no target(term deposit)in dataset')
```



only 11.7% of enteries are for y=1, so our dataset is unbalanced.

```
# POUTCOME
pieChart('poutcome','Distribution of poutcome in dataset')
df.poutcome.value_counts()
df.groupby('poutcome').size()
```



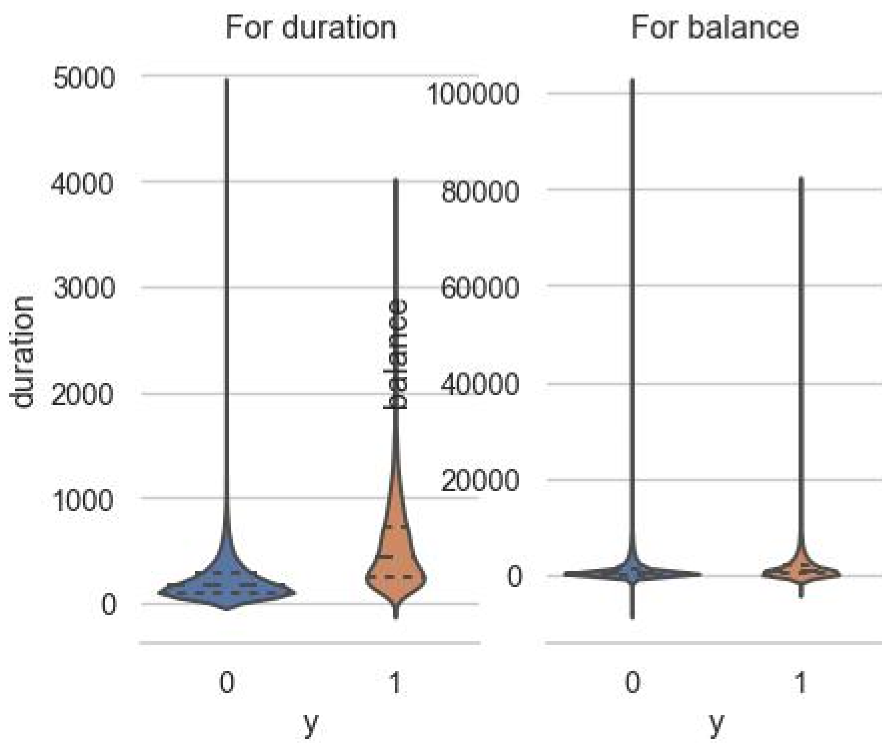

```
poutcome
failure     4901
other       1840
success     1511
unknown    36959
dtype: int64
```

There are *36959 unknown* values and 1840 values with other category. Since, 82% of entries are unknown, 4.07% other, we will directly drop this column.

Age, duration and balance

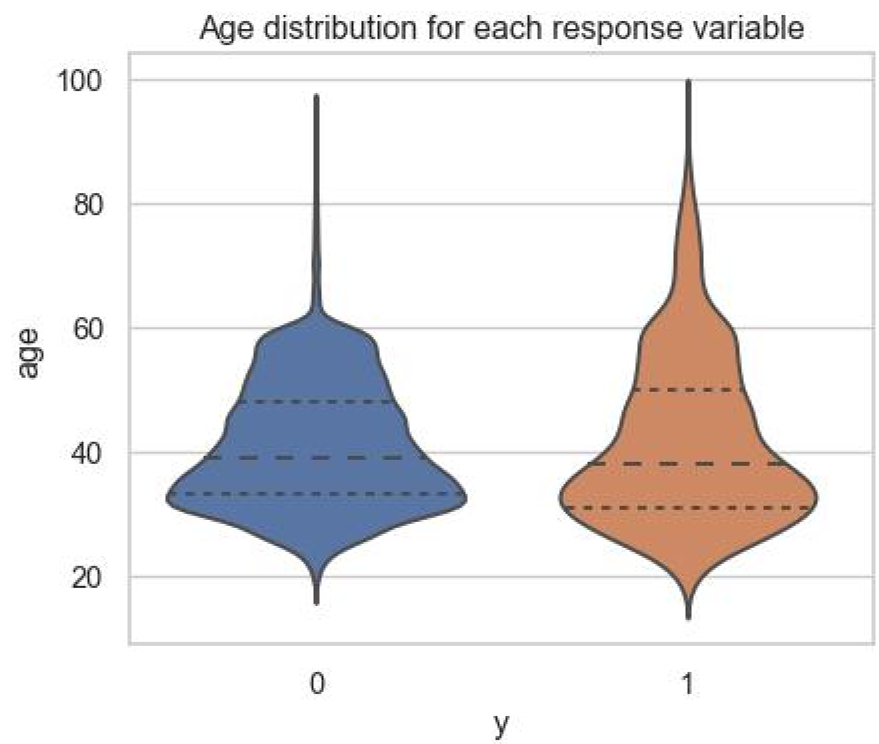
```
# plotting violen plot for duration and balance

f, axes = plt.subplots(1, 2,sharex=True)
axes[0].set_title('For duration')
```

```
sns.violinplot( x='y',y='duration',  split=True, inner="quart", ax=axes
[0], data=df)
axes[1].set_title('For balance')
sns.violinplot( x='y',y='balance',  split=True, inner="quart", ax=axes
[1], data=df)
sns.despine(left=True)
plt.show()
```



- There are outliers in duration and balance so we need to get rid of them.
- people who have a high balance, are more likely to subscribe to term deposit.

```
sns.violinplot( x='y',y='age',  split=True, inner="quart", data=df)
plt.title('Age distribution for each response variable')
plt.show()
```



- No outliers
- People who are old are more likely to subscribe to term deposit.

Summary

Data Cleaning

- Contact is not useful so we drop it.
- In poutcome, we have a lot of missing values so we drop it.

- Day is not giving any relevant infomation so we drop it.
- Removing the unknowns
- Remove the outliers from balance and duration.

Data Visualization

Data Cleaning

Dropping the column

```
clean_data = df.drop(['contact','poutcome','day'],axis=1)
```

As most of the values of poutcome is unknown making the column unimportant in subsequent analysis. More over the day of the week has no significant relation to the subcription of term deposit neither cantact type. ## Removing unknown from job and education

```
for i in clean_data.columns:
    if clean_data[i].dtype == np.int64:
        pass
    else:
        # printing names and count using loop.
        for idx, name in enumerate(clean_data[i].value_counts().index.t
olist()):
            if name == 'unknown' or name == 'other':
                print(f"for {i}")
                print(f"{name} : {clean_data[i].value_counts()[idx]}")
                if clean_data[i].value_counts()[idx] < 15000:
                    print(f"dropping rows with value as {name} in {i}")
                    clean_data = clean_data[clean_data[i] != name]

for job
unknown : 288
dropping rows with value as unknown in job
for education
unknown : 1730
dropping rows with value as unknown in education
```

Dropping the rows

Dropping the rows where values are 3SD away

Balance - Outliers

```
standard_deviation = clean_data[['balance']].std()
mean = clean_data[['balance']].mean()
clean_data['balance_outliers'] = clean_data['balance']
clean_data['balance_outliers']= zscore(clean_data['balance_outliers'])
print(f"removing entries before {mean - 3*standard_deviation } and afte
r {mean + 3*standard_deviation }")
three_SD = (clean_data['balance_outliers']>3) | (clean_data['balance_ou
tliers']<-3 )
clean_data = clean_data.drop(clean_data[three_SD].index, axis = 0, inpl
ace = False)
clean_data = clean_data.drop('balance_outliers', axis=1)

removing entries before balance   -7772.283533
dtype: float64 and after balance    10480.338218
dtype: float64
```

Duration - Outliers

Dropping rows where the duration of calls is less than 5sec since that is irrelevant

```
less_5 = (clean_data['duration']<5)
clean_data = clean_data.drop(clean_data[less_5].index, axis = 0, inplac
e = False)
```

Changing unit of duration from seconds to minutes to make more sense

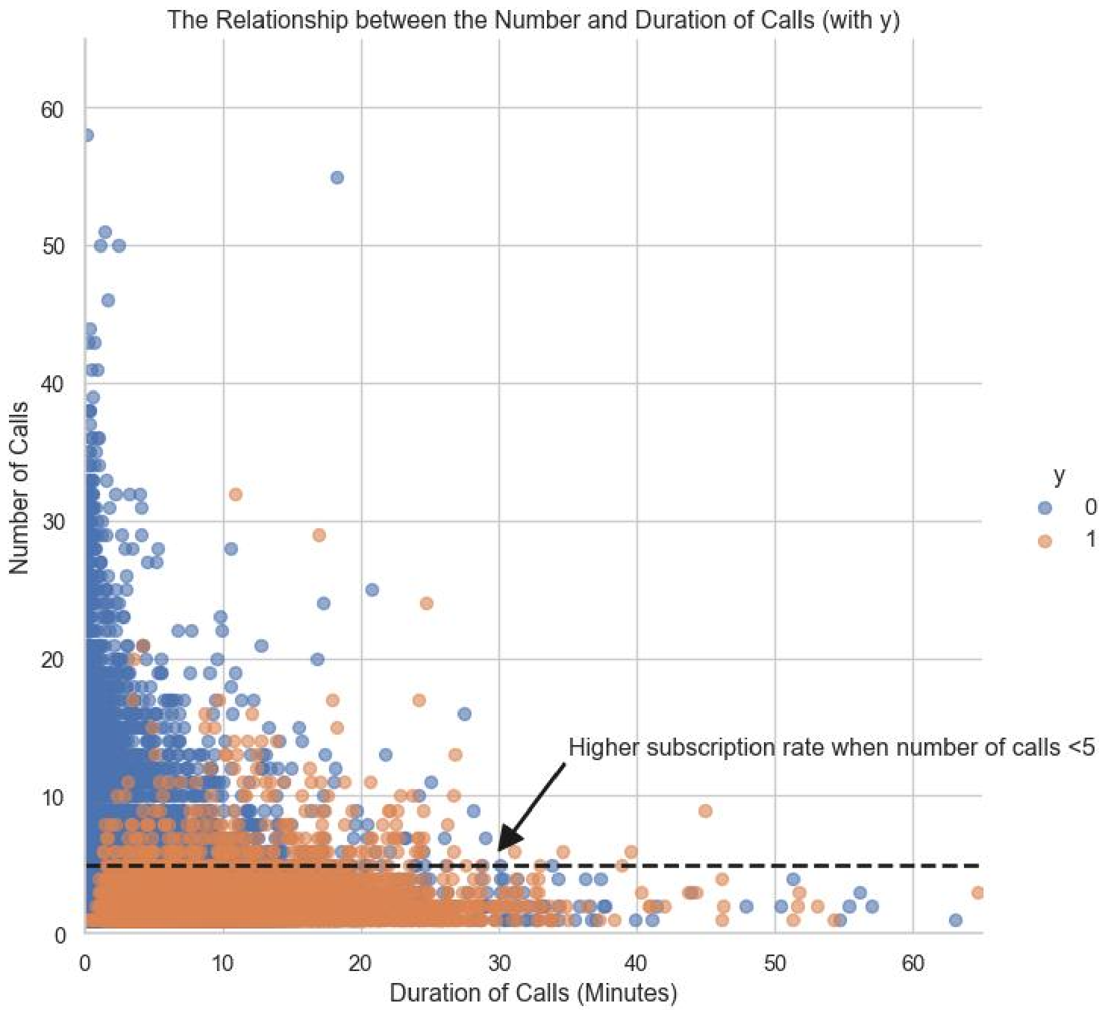
```
clean_data['duration'] = clean_data['duration'].apply(lambda n:n/60).ro
und(2)
```

Data Visualization

Contact versus Subscription month wise

Number of calls versus Duration and affect on subscription

```
import seaborn as sns
dur_cam = sns.lmplot(x='duration', y='campaign',data = clean_data,
                     hue = 'y',
                     fit_reg = False,
                     scatter_kws={'alpha':0.6}, height =7)

plt.axis([0,65,0,65])
plt.ylabel('Number of Calls')
plt.xlabel('Duration of Calls (Minutes)')
plt.title('The Relationship between the Number and Duration of Calls (w
ith y)')

# Annotation
plt.axhline(y=5, linewidth=2, color="k", linestyle='--')
plt.annotate('Higher subscription rate when number of calls <5 ',xytext
 = (35,13),
             arrowprops=dict(color = 'k', width=1),xy=(30,6))
plt.show()
```



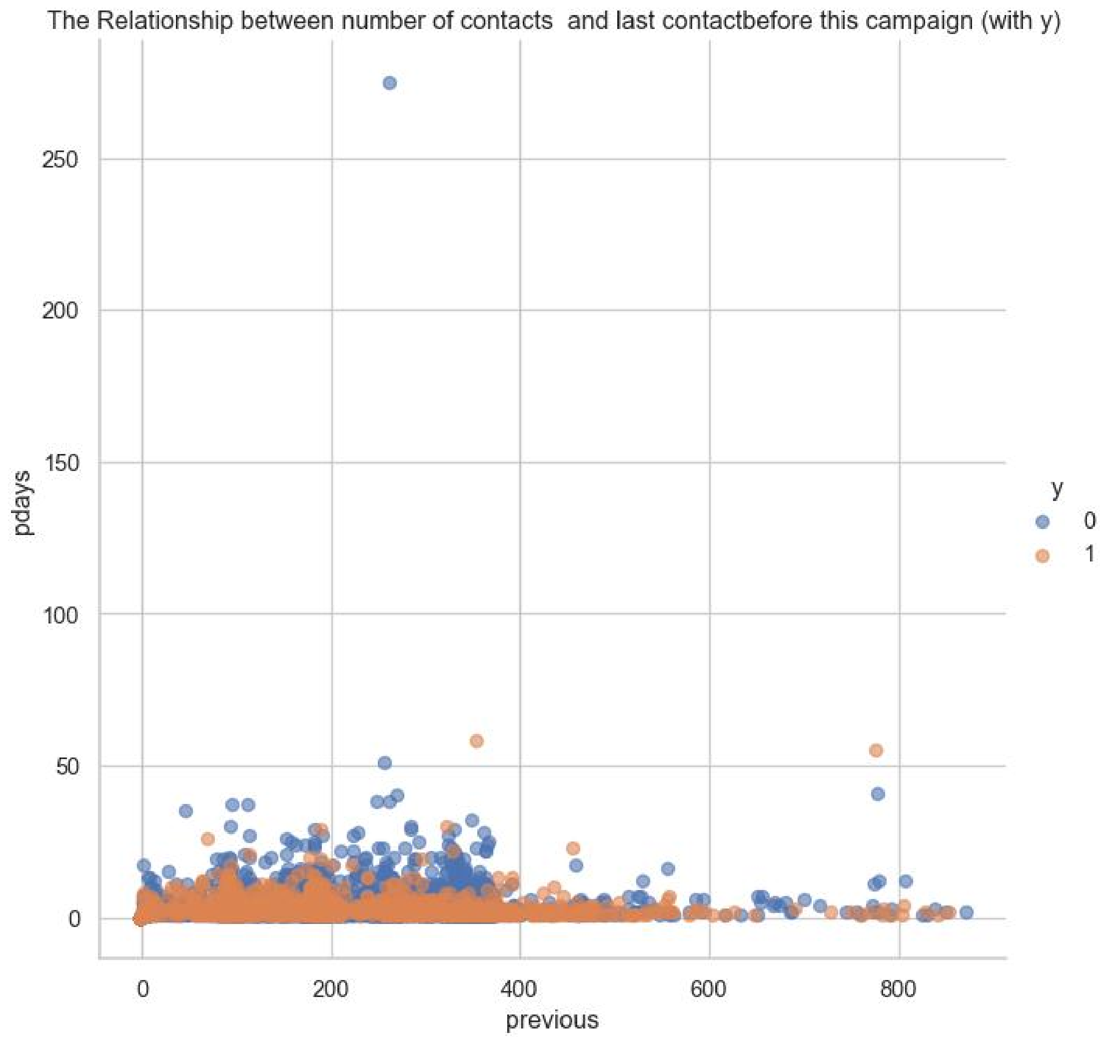
Checking between pdays and previous as well

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)

```
import seaborn as sns
dur_cam = sns.lmplot(x='pdays', y='previous',data = clean_data,
                     hue = 'y',
                     fit_reg = False,
                     scatter_kws={'alpha':0.6}, height =7)

# plt.axis([0,65,0,65])
plt.ylabel('pdays')
plt.xlabel('previous')
```

```
plt.title('The Relationship between number of contacts  and last contac
tbefore this campaign (with y)')

plt.show()
```



Smart Question

Based on last contact info only number of contacts performed during this campaign is contributing a lot towards subscription rates.

Month wise subscription

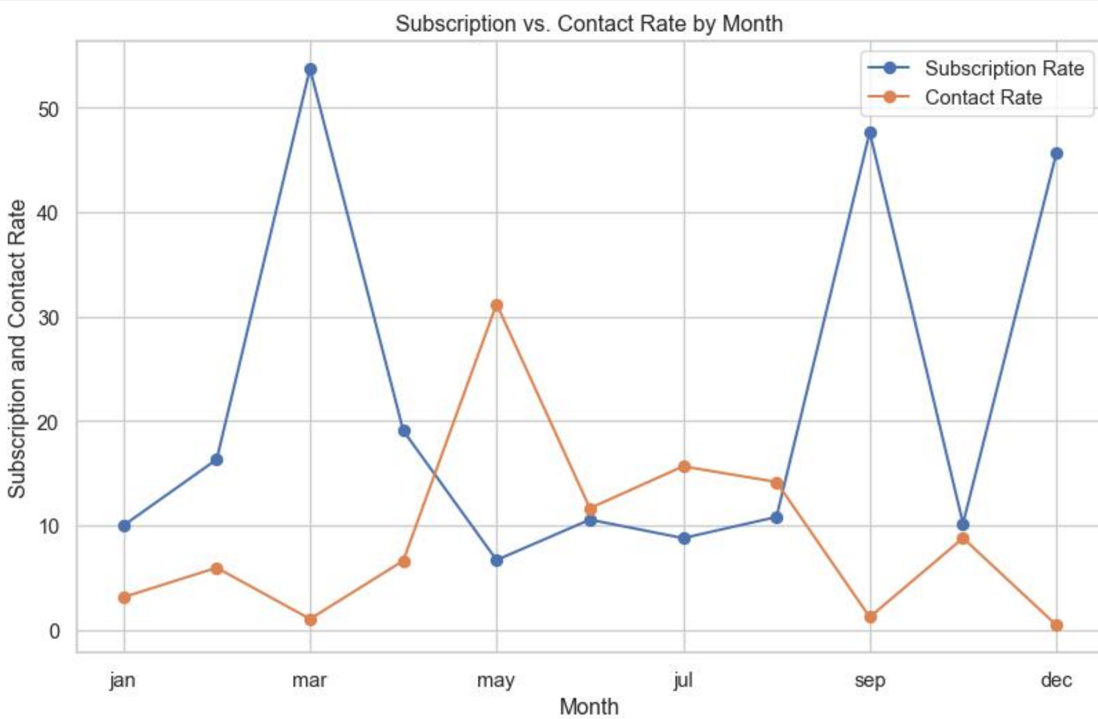
```
#converting y values
# bankdata['y'] = bankdata['y'].apply(lambda x: 'no' if x == 'yes' else
 1)
# bankdata['y'] = bankdata['y'].astype('category')

#value count for each month
month = clean_data['month'].value_counts().rename_axis('month').reset_i
ndex(name='counts')
```

```
#for sequencing the month
m1_list=['jan','feb','mar','apr','may','jun','jul','aug','sep','nov','d
ec']
m1=pd.DataFrame(m1_list,columns=['month'])
#now the dataset is sequeced
month = m1.merge(month)
#month - counts
#% of people contacted in that month
month['Contact Rate'] = month['counts']*100/month['counts'].sum()
#percentage of people contacted in that month
# y response
month_y = pd.crosstab(clean_data['y'],clean_data['month']).apply(lambda
 x: x/x.sum() * 100)
#% of 0 and 1 for each month
month_y = month_y.transpose()
month_y.rename(columns = {'y':'month',0:'no', 1:'yes'}, inplace = True)

# month_y
# y | no% | yes%

#month = month.merge(month_y)
month['yes'] = " "
month['no'] = " "
#to make it in sequence
def addingCrossTab():
    for i, val in enumerate(m1_list):
        #print (i, ",",val)
        month['yes'].iloc[i]=month_y.loc[val].loc['yes']
        #print(month_y.loc[val].loc['yes'])
        month['no'].iloc[i]=month_y.loc[val].loc['no']

addingCrossTab()
#print(month)
#print(month_y)
# month['Subscription Rate'] = month_y['yes']
# month['% NotSubscription'] = month_y['no']
month.rename(columns = {'yes':'Subscription Rate','no':'NotSubscribed R
ate'}, inplace = True)
#month.drop('month_int',axis = 1,inplace = True)
print(month)

   month  counts  Contact Rate Subscription Rate NotSubscribed Rate
0    jan    1310      3.134046              10.0               90.0
1    feb    2492      5.961865         16.332263          83.667737
2    mar     439      1.050264         53.758542          46.241458
3    apr    2772      6.631738         19.083694          80.916306
4    may   13050     31.220843          6.697318          93.302682
5    jun    4874     11.660566          10.56627           89.43373
6    jul    6550     15.670231          8.793893          91.206107
7    aug    5924     14.172588         10.820392          89.179608
```

```
8     sep     514      1.229694        47.66537          52.33463
9     nov    3679      8.801646        10.192987         89.807013
10    dec     195      0.466518        45.641026         54.358974
```

```
plot_month = month[['month','Subscription Rate','Contact Rate']].plot(x
='month',kind ='line',
                                                          figsize = (10,
6),
                                                          marker = 'o')

plt.title('Subscription vs. Contact Rate by Month')
plt.ylabel('Subscription and Contact Rate')
plt.xlabel('Month')
```

```
Text(0.5, 0, 'Month')
```



Maximum percentage of people have subscribed in the month of March but bank is contacting people more in the month of May. So it's better to contact customer's based on the subcription rate plot.

Social and economic Factors in month

```
month_social_economic = clean_data[['month','cons.conf.idx','emp.var.ra
te','euribor3m','nr.employed']].groupby(['month']).count().reset_index()
month_list= ['jan','feb','mar','apr','may','jun','jul','aug','sep','oct
','nov','dec']
month_pd = pd.DataFrame(month_list,columns=['month'])
month_pd = month_pd.merge(month_social_economic,on='month')
print(month_pd)
```

```
   month  cons.conf.idx  emp.var.rate  euribor3m  nr.employed
0    jan           1310          1310       1310         1310
1    feb           2492          2492       2492         2492
2    mar            439           439        439          439
3    apr           2772          2772       2772         2772
4    may          13050         13050      13050        13050
5    jun           4874          4874       4874         4874
6    jul           6550          6550       6550         6550
7    aug           5924          5924       5924         5924
8    sep            514           514        514          514
9    oct            661           661        661          661
10   nov           3679          3679       3679         3679
11   dec            195           195        195          195
```

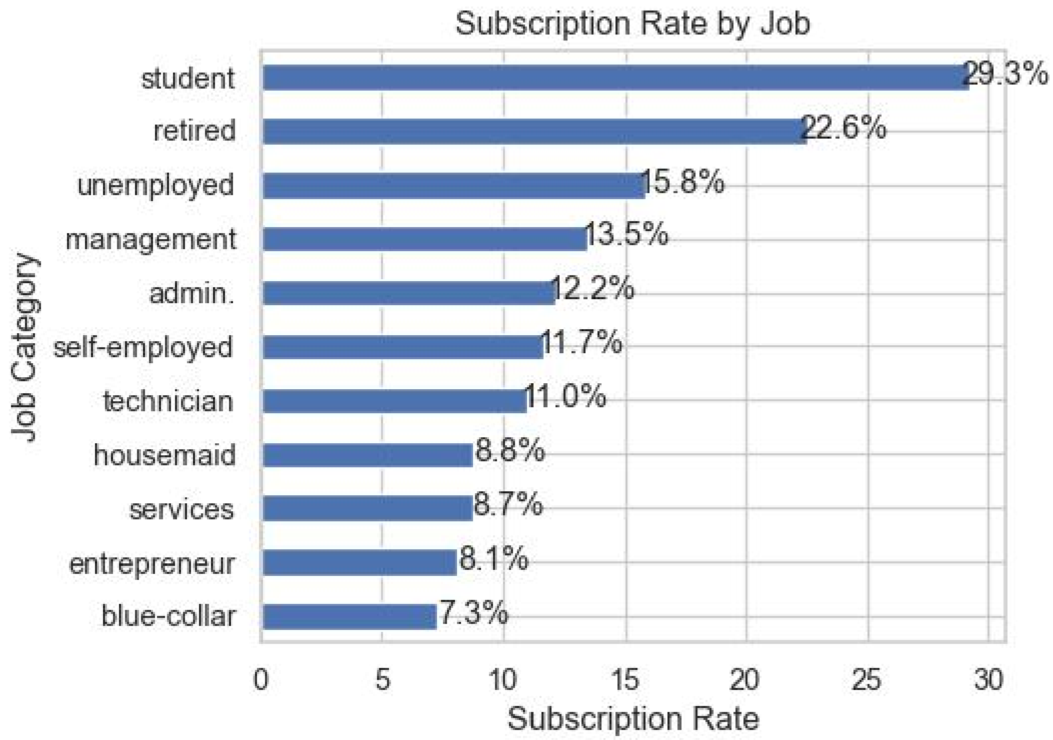
Based on the above table we can see that there is no distinguishable difference in the month of march or may from rest of all the month, so social and economic factor **do not have major influence** on the outcome.

Checking the Financially stable population

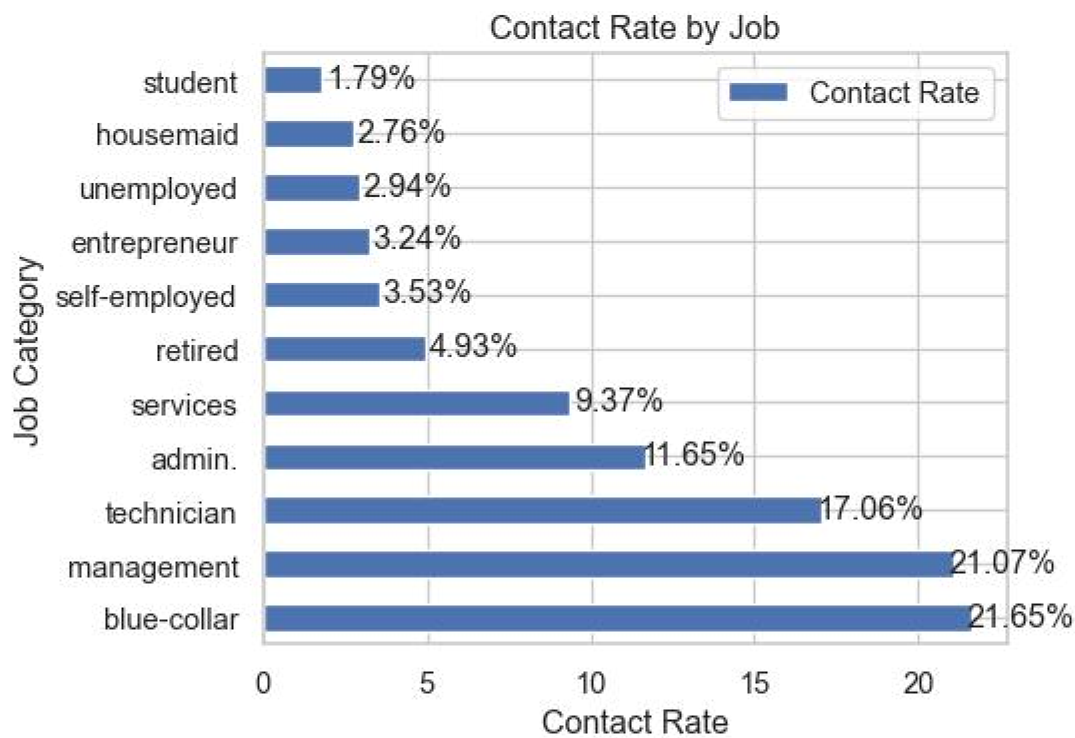
```
data_vis = clean_data.copy()
```

Job

```
y_job = pd.crosstab(data_vis['y'],data_vis['job']).apply(lambda x: x/x.
sum() * 100)
y_job = y_job.transpose()

y_job.rename(columns = {'y':'job',0:'no', 1:'yes'}, inplace = True)
jobs_sub = y_job['yes'].sort_values(ascending = True).plot(kind ='barh')


plt.title('Subscription Rate by Job')
plt.xlabel('Subscription Rate')
plt.ylabel('Job Category')
# Label each bar
for patch_i, label in zip(jobs_sub.patches,
                      y_job['yes'].sort_values(ascending = True).round
(1).astype(str)):
    jobs_sub.text(patch_i.get_width()+1.5,
                 patch_i.get_y()+ patch_i.get_height()-0.5,
                 label+'%',
                 ha = 'center',
                 va='bottom')
```




```
job_contact=  data_vis['job'].value_counts().rename_axis('job').reset_i
ndex(name='counts')
job_contact['Contact Rate']= job_contact['counts']*100/job_contact['cou
nts'].sum()
job_contact['Contact Rate'] = job_contact['Contact Rate'].round(2)
job_contact=job_contact.drop(['counts'],axis=1)

# job_contact['Contact Rate']= job_contact['Contact Rate'].sort_values
(ascending = False)
job_contact_plot = job_contact.plot(x='job',kind ='barh')
#.plot(kind ='barh')
plt.title('Contact Rate by Job')
plt.xlabel('Contact Rate')
plt.ylabel('Job Category')
# Label each bar
for patch_i, label in zip(job_contact_plot.patches,
                      job_contact['Contact Rate'].astype(str)):
    job_contact_plot.text(patch_i.get_width()+1.5,
                  patch_i.get_y()+ patch_i.get_height()-0.5,
                  label+'%',
                  ha = 'center',
                  va='bottom')
```



People in blue color and managemnet jobs are contacted more, which should not be the case.

Balance

```
#max = 10399
#min = -6847
def balance_group(bal):
    balGroup = 'Negative' if bal < 0 else 'low balance' if bal < 1000 e
lse 'moderate balance' if bal < 2500 else 'high balance'
    return balGroup
data_vis['balGroup'] = data_vis['balance'].apply(balance_group)
```

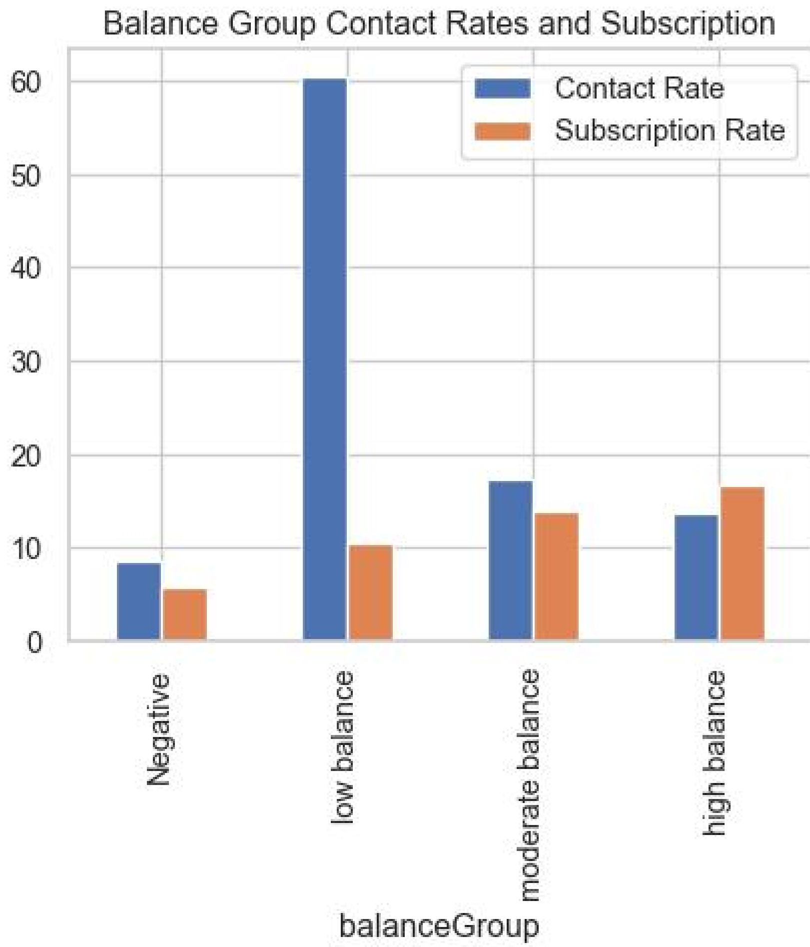
checking the subscription based on y value

```
y_balance = pd.crosstab(data_vis['y'],data_vis['balGroup']).apply(lambd
a x: x/x.sum() * 100)
y_balance = y_balance.transpose()
```

Checking the subscriptions in each balance groups

```
bal = pd.DataFrame(data_vis['balGroup'].value_counts().rename_axis('bal
Group').reset_index(name='counts'))
bal_y = bal.merge(y_balance,on='balGroup')

bal_y['% Contacted'] = bal_y['counts']*100/bal_y['counts'].sum()
bal_y['% Subscription'] = bal_y[1]
bal_y.rename(columns = {'y':'month',0:'no', 1:'yes'}, inplace = True)
```

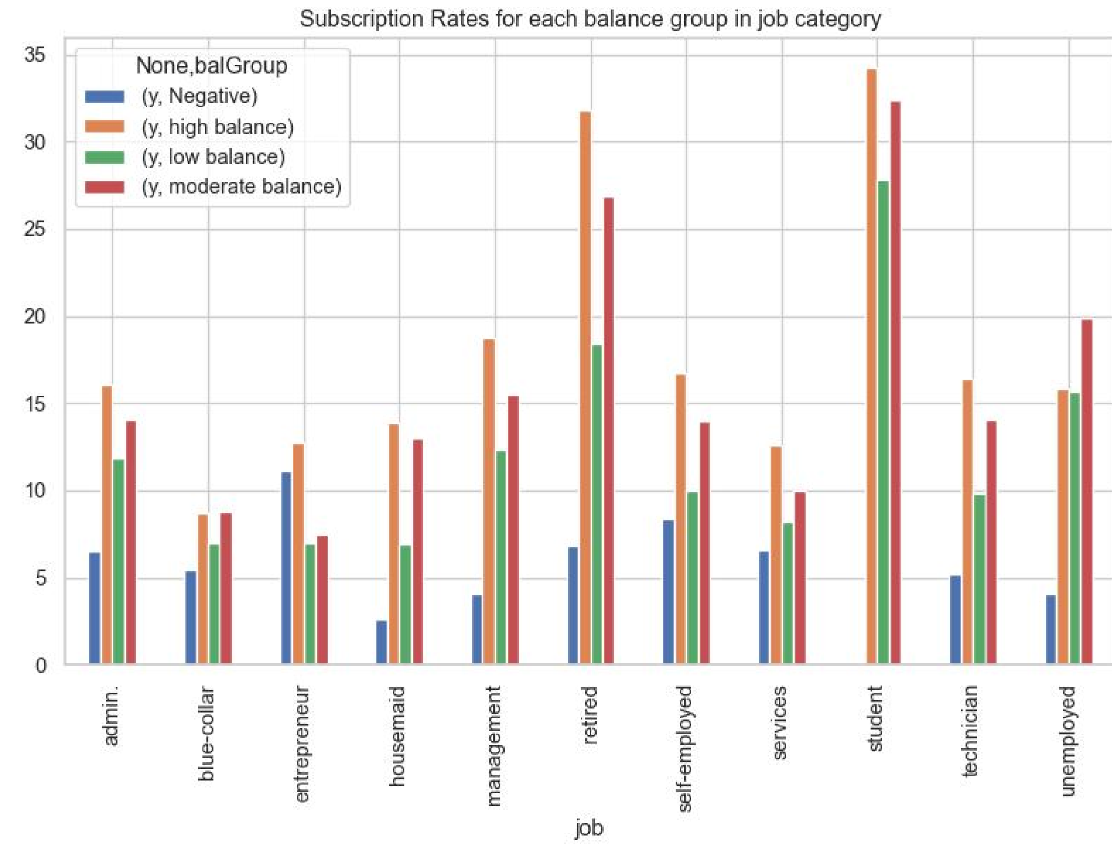
```
bal_y = bal_y.drop(['counts','no','yes'],axis=1)
print(bal_y)

bal_list = ['Negative','low balance', 'moderate balance','high balance']
balanceGroupInfo =pd.DataFrame(bal_list,columns=['balanceGroup'])
balanceGroupInfo['Contact Rate'] = " "
balanceGroupInfo['Subscription Rate'] = " "
bal_y = bal_y.set_index(['balGroup'])


for i,val in enumerate(bal_list):
     balanceGroupInfo['Contact Rate'].iloc[i]=bal_y.loc[val].loc['% Con
tacted']
     balanceGroupInfo['Subscription Rate'].iloc[i]=bal_y.loc[val].loc
['% Subscription']
print(balanceGroupInfo)
#bal['bal'] = [1,2,0,3]
#bal = bal.sort_values('bal',ascending = True)

           balGroup  % Contacted  % Subscription
0       low balance    60.339143       10.503513
1  moderate balance    17.399906       14.036275
2      high balance    13.709374       16.715341
3          Negative     8.551578        5.700909
       balanceGroup Contact Rate Subscription Rate
0          Negative     8.551578          5.700909
1       low balance    60.339143         10.503513
2  moderate balance    17.399906         14.036275
3      high balance    13.709374         16.715341

balanceGroupInfo.plot(x='balanceGroup', kind='bar', stacked=False,
        title='Balance Group Contact Rates and Subscription')
plt.show()
```



People with moderate to high balance, are contacted less but they have high subscription rates so bank should target them more.

Balance Group versus Job

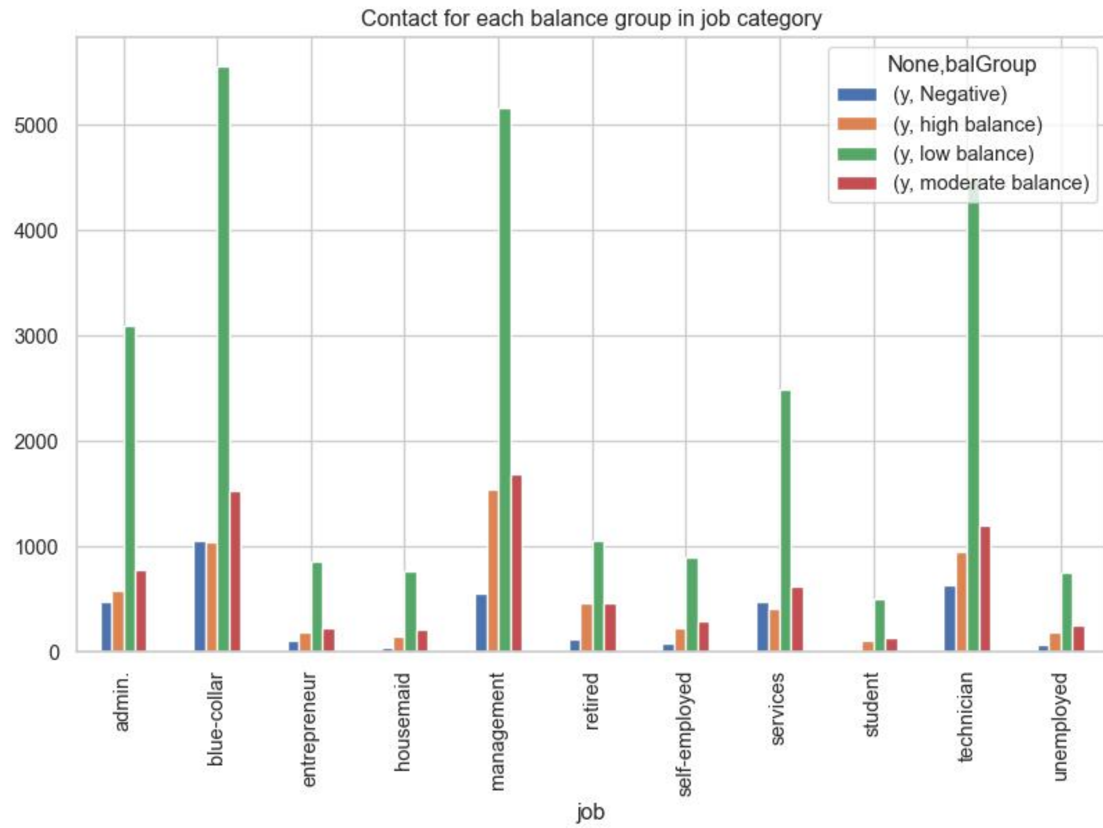
```
# add the values for 1
job_balance = pd.DataFrame(data_vis.groupby(['job','balGroup'])['y'].su
m())
# total number of values
job_balance_count = pd.DataFrame(data_vis.groupby(['job','balGroup'])['
y'].count())

job_balance['y'] = (job_balance['y']/job_balance_count['y'])*100
job_balance = job_balance.unstack()
job_balance = job_balance.plot(kind='bar',figsize = (10,6))
plt.title('Subscription Rates for each balance group in job category')

Text(0.5, 1.0, 'Subscription Rates for each balance group in job catego
ry')
```



Student and Retired are more likely to subscribe and usually have moderate to high balance.

```
job_balance_count1 = job_balance_count.unstack()
job_balance_count1 = job_balance_count1.plot(kind='bar',figsize = (10,
6))
plt.title('Contact for each balance group in job category')
```

```
Text(0.5, 1.0, 'Contact for each balance group in job category')
```



Loan

covered loan in initial EDA

```
data_encode = data_vis.copy()
```

Getting Data Ready for Modelling

Encoding

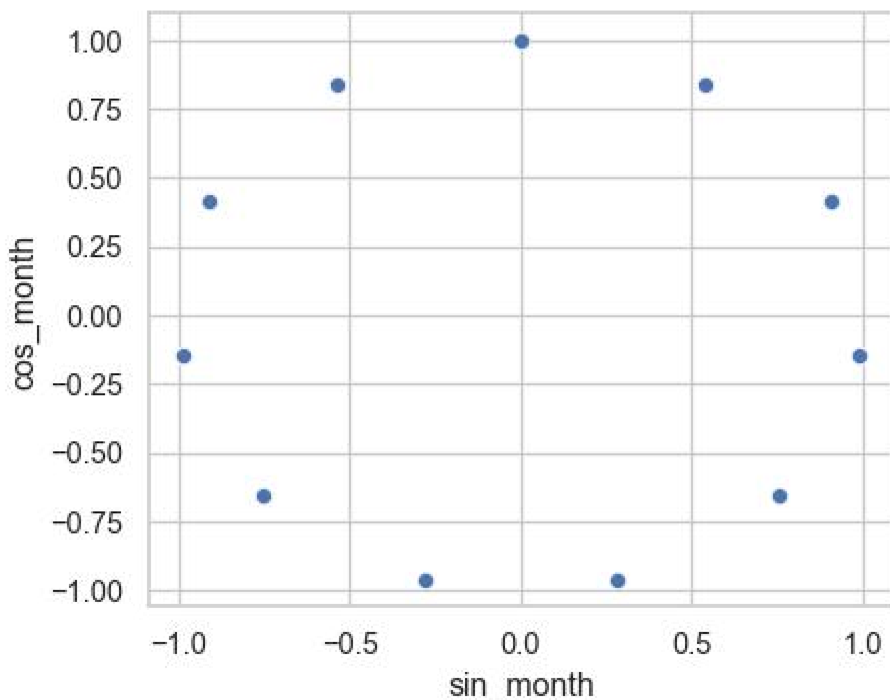
One Hot Encoding

```
data_encode = pd.get_dummies(data_encode, columns = ['housing'])
data_encode = pd.get_dummies(data_encode, columns = ['loan'])
data_encode = pd.get_dummies(data_encode, columns = ['default'])
data_encode = pd.get_dummies(data_encode, columns = ['job'])
data_encode = pd.get_dummies(data_encode, columns = ['education'])
data_encode = pd.get_dummies(data_encode, columns = ['marital'])
```

Sin - Cos encoding

```
import math
from math import pi
def sin_transformation(x):
```

```
    x=x-1
    sin_x = math.sin((2*pi*x)/11)
    return sin_x
def cos_transformation(x):
    x=x-1
    cos_x = math.cos((2*pi*x)/11)
    return cos_x
data_encode['sin_month'] = data_encode['month_int'].apply(sin_transform
ation)
data_encode['cos_month'] = data_encode['month_int'].apply(cos_transform
ation)

sns.scatterplot(data=data_encode,x='sin_month',y='cos_month')

<AxesSubplot: xlabel='sin_month', ylabel='cos_month'>
```



Label Encoding

```
data_encode= data_encode.drop(['month'],axis=1)
#data_encode= data_encode.drop(['month_int'],axis=1)
data_encode = data_encode.drop(['balGroup'],axis=1)
data_encode = data_encode.drop(['pdays'],axis=1)
```

Checkpoint

```
#data_encode.to_csv('Dataset/final_encoded.csv',index=False)
#data_encode =  pd.read_csv('Dataset/final_encoded.csv')

data_model = data_encode.copy()
```

Dropping the unecessary varibles for modelling

```
data_model=data_model.drop(['cons.conf.idx', 'emp.var.rate', 'euribor3m
', 'nr.employed',
       'cons.price.idx'],axis=1)
```

Splitting our Dataset

```
#dropping y to extract x variables
x = data_model.drop(['y'],axis=1)
#y variables
y = data_model['y']
#splitting the dataset
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```

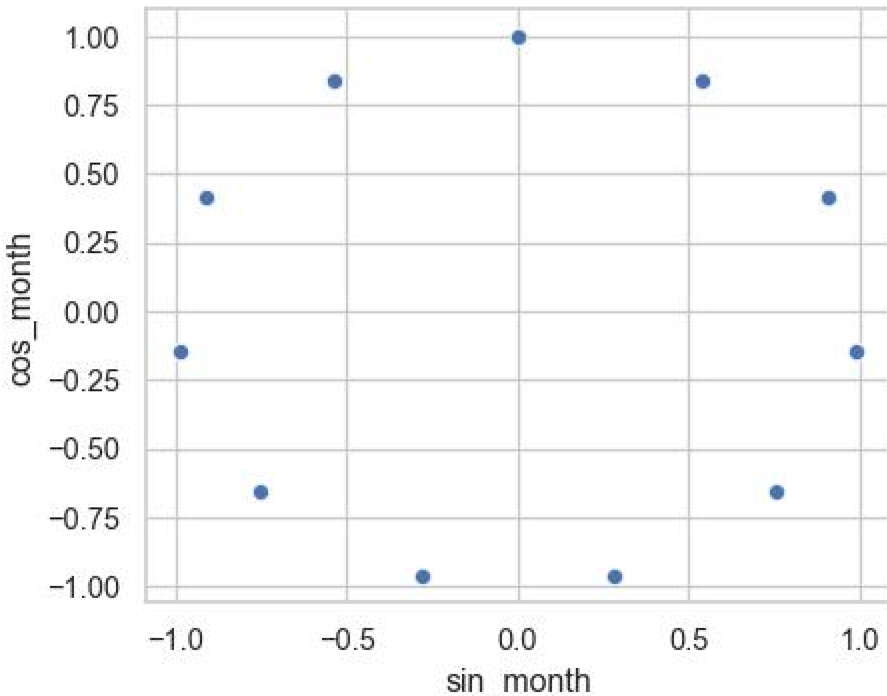
Balancing Our Dataset

```
sm = SMOTE(random_state=42)
train_sx, train_sy = sm.fit_resample(x_train, y_train)
test_sx, test_sy = sm.fit_resample(x_test, y_test)
#printing x and y values
np.bincount(train_sy)
```

```
array([30046, 30046], dtype=int64)
```

```
train_sx['sin_month'] = train_sx['month_int'].apply(sin_transformation)

train_sx['cos_month'] = train_sx['month_int'].apply(cos_transformation)


sns.scatterplot(data=train_sx,x='sin_month',y='cos_month')
```

```
<AxesSubplot: xlabel='sin_month', ylabel='cos_month'>
```




```
train_sx= train_sx.drop(['month_int'],axis=1)
test_sx=test_sx.drop(['month_int'],axis=1)

x_train= x_train.drop(['month_int'],axis=1)
x_test=x_test.drop(['month_int'],axis=1)
```

Checkpoint 2

```
train_balanced = pd.concat([train_sx, train_sy], axis=1)
train_unbalanced = pd.concat([x_train, y_train], axis=1)

test_unbalanced = pd.concat([x_test, y_test], axis=1)
test_balanced = pd.concat([test_sx, test_sy], axis=1)

# train_balanced.to_csv('Dataset/train_balanced.csv',index=False)
# train_unbalanced.to_csv('Dataset/train_unbalanced.csv',index=False)
# test_unbalanced.to_csv('Dataset/test_unbalanced.csv',index=False)
# test_balanced.to_csv('Dataset/test_balanced.csv',index=False)
# print("Before Smote")
# print(f"for training : {np.bincount(y_train)}")
# print(f"for testing : {np.bincount(y_test)}")
# print("After smote")
# print(f"for training : {np.bincount(y_res)}")
# print(f"for testing : {np.bincount(test_sy)}")

balanced_train= pd.read_csv('Dataset/train_balanced.csv')
balanced_test= pd.read_csv('Dataset/test.csv')
unbalanced_train= pd.read_csv('Dataset/train_unbalanced.csv')
unbalanced_test= pd.read_csv('Dataset/test.csv')
```

```
from sklearn.preprocessing import StandardScaler
# define standard scaler
scaler = StandardScaler()
# transform data
balanced_train[['age','balance','duration']]= scaler.fit_transform(bala
nced_train[['age','balance','duration']])

balanced_test[['age','balance','duration']]= scaler.fit_transform(balan
ced_test[['age','balance','duration']])

unbalanced_train[['age','balance','duration']]= scaler.fit_transform(un
balanced_train[['age','balance','duration']])

unbalanced_test[['age','balance','duration']]= scaler.fit_transform(unb
alanced_test[['age','balance','duration']])

x_train = unbalanced_train.drop(['y'],axis=1)
x_test = unbalanced_test.drop(['y'],axis=1)
y_train = unbalanced_train['y']
y_test = unbalanced_test['y']

bx_train = balanced_train.drop(['y'],axis=1)
bx_test = balanced_test.drop(['y'],axis=1)
by_train = balanced_train['y']
by_test = balanced_test['y']
```

Logistic Regression

Performing Logistic Regression on both balanced and unbalanced dataset. RFE is used in selecting the most important features ## Unbalanced Dataset

```
rfe_model = RFE(LogisticRegression(solver='lbfgs', max_iter=1000), step
= 25)
rfe_model = rfe_model.fit(x_train,y_train)

# feature selection
#print(rfe_model.support_)
#print(rfe_model.ranking_)

selected_columns = x_train.columns[rfe_model.support_]
list_column= selected_columns.tolist()
list_column.append('age')
list_column.append('balance')
#list_column.append('sin_month')
print(f"Columns selected by RE {list_column}")


X_train_final = x_train[list_column]
X_test_final = x_test[list_column]
```

```
#X_train_final['balance','age'] = x_train['balance','age']
#X_test_final['balance','age'] = x_test['balance','age']
```

```
Columns selected by RE ['duration', 'euribor3m', 'cons.price.idx', 'job
_blue-collar', 'job_retired', 'job_student', 'education_primary', 'educ
ation_tertiary', 'marital_single', 'housing_no', 'housing_yes', 'loan_n
o', 'loan_yes', 'poutcome_failure', 'poutcome_success', 'month_apr', 'm
onth_aug', 'month_feb', 'month_jan', 'month_jul', 'month_jun', 'month_m
ar', 'month_nov', 'month_oct', 'age', 'balance']
```

As we can see from RFE, the most relevant features are :

- Duration
- Housing
- Loan
- Job
- Education
- cos_month

From other features selection techniques and EDA, we can see that ‘age’ and ‘balance’ also contrubuted to the subscrption, so we added up these variables as well.

Applying model with selected features

```
lr = LogisticRegression(random_state=123)
lr.fit(X_train_final, y_train)
y_pred = lr.predict(X_test_final)
print(f"Accuracy for training set {accuracy_score(y_train, lr.predict(X
_train_final))}")
print(f"Accuracy for testing set {accuracy_score(y_test, y_pred)}")
print(f"Confusion matrix \n{confusion_matrix(y_test, y_pred)}")
print(f"{classification_report(y_test, y_pred)}")
```

```
Accuracy for training set 0.8717311715481172
Accuracy for testing set 0.8713389121338913
Confusion matrix
[[4759  148]
 [ 590  239]]
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      4907
           1       0.62      0.29      0.39       829

    accuracy                           0.87      5736
   macro avg       0.75      0.63      0.66      5736
weighted avg       0.85      0.87      0.85      5736
```

Here, the accuracy is 89% but the precision(0.59) and recall rate value(0.20) is low. And we also check on the balanced dataset since the low recall rate might be caused because of the less number of y = 1 value.

Balanced Dataset

```
rfe_model = RFE(LogisticRegression(solver='lbfgs', max_iter=1000), step
= 25)
rfe_model = rfe_model.fit(bx_train,by_train)

# feature selection
#print(rfe_model.support_)
#print(rfe_model.ranking_)

selected_columns = bx_train.columns[rfe_model.support_]
print(f"Columns selected by RE {selected_columns.tolist()}")

list_column= selected_columns.tolist()
list_column.append('age')
list_column.append('balance')
#list_column.append('sin_month')
list_column.append('duration')
#list_column.append('cos_month')

#balanced dataset
bX_train_final = bx_train[list_column]
bX_test_final = bx_test[list_column]
#unbalanced test dataset
ubx_test_final = x_test[list_column]

lr_b = LogisticRegression(random_state=123)
lr_b.fit(bX_train_final, by_train)
by_pred = lr_b.predict(ubx_test_final)
print(f"Accuracy for training set {accuracy_score(by_train, lr_b.predic
t(bX_train_final))}")
print(f"Accuracy for testing set {accuracy_score(y_test, by_pred)}")
print(f"Confusion matrix \n{confusion_matrix(y_test, by_pred)}")
print(f"{classification_report(y_test, by_pred)}")
```

```
Columns selected by RE ['duration', 'cons.price.idx', 'job_admin.', 'jo
b_blue-collar', 'job_management', 'job_self-employed', 'job_services',
'job_technician', 'job_unemployed', 'education_primary', 'education_sec
ondary', 'education_tertiary', 'marital_divorced', 'marital_married', '
marital_single', 'housing_no', 'housing_yes', 'loan_yes', 'poutcome_fai
lure', 'month_apr', 'month_aug', 'month_jul', 'month_may', 'month_nov']

Accuracy for training set 0.9064744536702155
Accuracy for testing set 0.8516387726638772
Confusion matrix
```

```
[[4440  467]
 [ 384  445]]
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      4907
           1       0.49      0.54      0.51       829

    accuracy                           0.85      5736
   macro avg       0.70      0.72      0.71      5736
weighted avg       0.86      0.85      0.85      5736
```

Here, important features are * Housing * Loan * Job * Education * Marital Status

We also added the important features from unbalaced dataset * Duration * Age * Month * Balance

Here even though the precision and recall have improved, and accuracy has dropped down, but the important relationships are lost since the training data now is artificially generated datapoints. We will try to find the optimal cut-off value for original dataset and compare it with the model for balanced data.

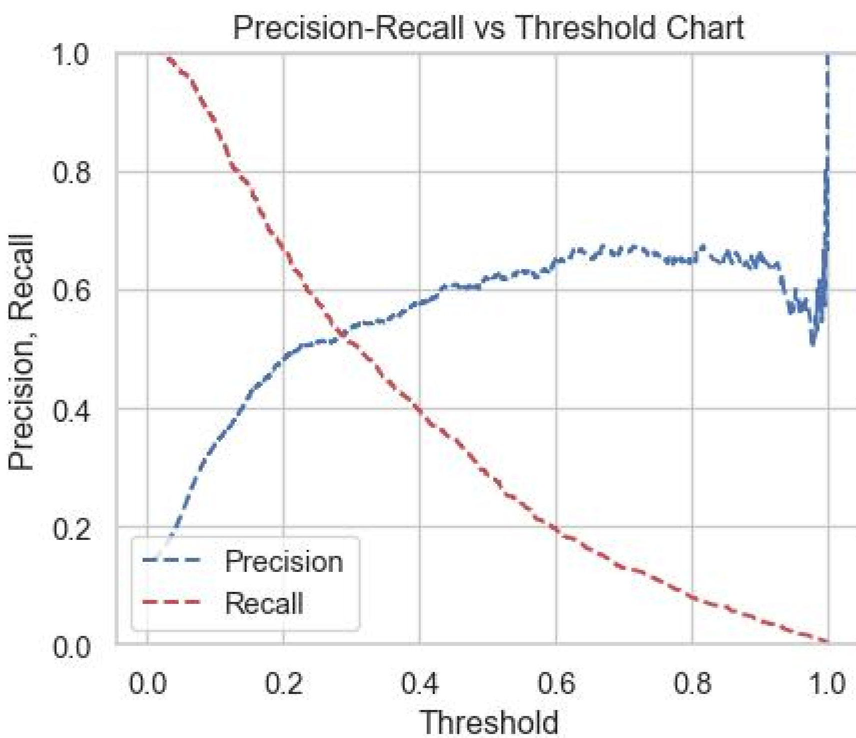
Deciding cut off value for logistic regression - Unbalance

But to have good values for cut-off we would try to find a cutoff where the precision and recall values are decent

```
# Precision-Recall vs Threshold
#y_pred=logit.predict(x_test)
y_pred_probs=lr.predict_proba(X_test_final)
# probs_y is a 2-D array of probability of being labeled as 0 (first
# column of array) vs 1 (2nd column in array)

precision, recall, thresholds = precision_recall_curve(y_test, y_pred_p
robs[:, 1])
#retrieve probability of being 1(in second column of probs_y)
pr_auc = metrics.auc(recall, precision)

plt.title("Precision-Recall vs Threshold Chart")
plt.plot(thresholds, precision[: -1], "b--", label="Precision")
plt.plot(thresholds, recall[: -1], "r--", label="Recall")
plt.ylabel("Precision, Recall")
plt.xlabel("Threshold")
plt.legend(loc="lower left")
plt.ylim([0,1])

print("\nBased on plot we would choose 0.25 as cut off ")


thres = 0.25
y_pred = np.where(y_pred_probs[:,1]>thres,1,0)
```

```
print(f"Accuracy for testing set {accuracy_score(y_test, y_pred)}")
print(f"Confusion matrix \n{confusion_matrix(y_test, y_pred)}")
print(f"{classification_report(y_test, y_pred)}")
```

```
Based on plot we would choose 0.25 as cut off
Accuracy for testing set 0.8589609483960948
Confusion matrix
[[4447  460]
 [ 349  480]]
              precision    recall  f1-score   support

           0       0.93      0.91      0.92      4907
           1       0.51      0.58      0.54       829

    accuracy                           0.86      5736
   macro avg       0.72      0.74      0.73      5736
weighted avg       0.87      0.86      0.86      5736
```



Optimal Cutoff at 0.25

Here as after applying feature selection, finding optimized cut-off, we are able to achieve higher accuracy with optimal precision and recall. Resulting from the comparison, we would continue our modellings with unbalance dataset.

Smart Question 5: The optimal cut off value for classification of our imbalance dataset.

Answer: The optimal cut off value for our imbalance dataset is 0.25 as the precision-recall chart indicated.

SMART Question 2: Since the dataset is imbalanced, will down sampling/up sampling or other techniques improve upon the accuracy of models.

Answer: As observed from above there is a slight improvement in accuracy, precision and recall after we apply SMOTE, but that improvement can also be acheived by adjusting the cut off value as well. So, we should always try adjusting cut-off first, before upsampling.

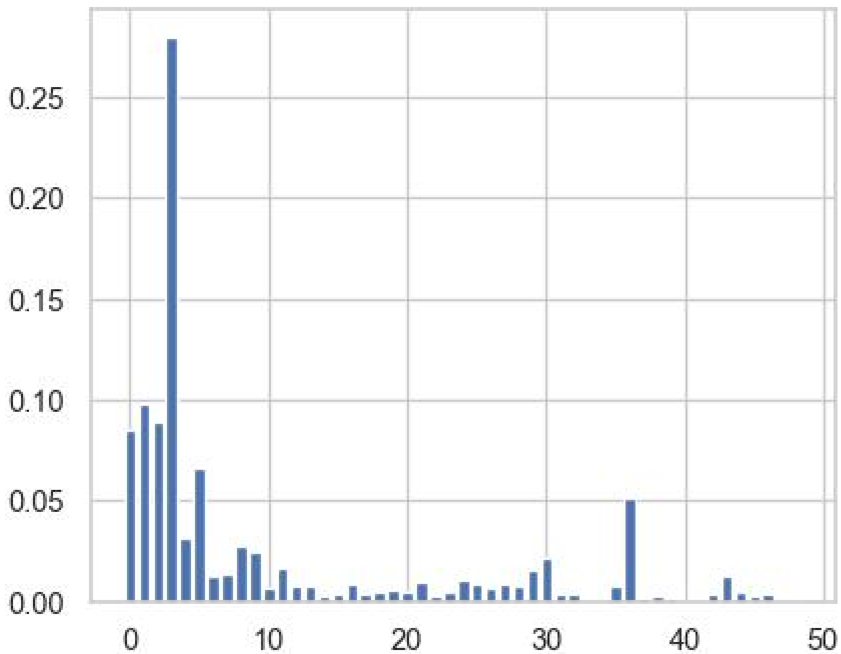
For ROC - AUC curve refer (Figure 1).
For precision recall curve refer(Figure 2).

Decision Tree

Feature Selection

```
# feature selection
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
importance = dtc.feature_importances_
features = []
imp = []
for i,v in enumerate(importance):
    if v >0.01:
        print(f"Feature {i} variable {x_train.columns[i]} score {v:.2f}
")
        features.append(x_train.columns[i])
        imp.append(v)
print(f"Important features from decision treee are : \n{features}")
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
x_train_dt = x_train[features]
x_test_dt = x_test[features]

Feature 0 variable age score 0.09
Feature 1 variable balance score 0.10
Feature 2 variable day score 0.09
Feature 3 variable duration score 0.28
Feature 4 variable campaign score 0.03
Feature 5 variable pdays score 0.07
Feature 6 variable previous score 0.01
Feature 7 variable cons.conf.idx score 0.01
Feature 8 variable emp.var.rate score 0.03
Feature 9 variable euribor3m score 0.02
Feature 11 variable cons.price.idx score 0.02
Feature 24 variable education_secondary score 0.01
```

```
Feature 29 variable housing_no score 0.02
Feature 30 variable housing_yes score 0.02
Feature 36 variable poutcome_success score 0.05
Feature 43 variable month_jun score 0.01
Important features from decision treee are :
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous',
'cons.conf.idx', 'emp.var.rate', 'euribor3m', 'cons.price.idx', 'educat
ion_secondary', 'housing_no', 'housing_yes', 'poutcome_success', 'month
_jun']
```



Features selected from this algorithm are

- Age
- Balance
- Duration
- Campaign
- Previous
- Housing
- Job
- Education
- Marital
- Month - Sin,cos

We have all the important features from EDA here

Hyperparameter tuning

For tuning the hyperparameter's we will use GridSearch CV.

```
# Creating a dictionary of parameters to use in GridSearchCV

params = {
    'criterion':  ['gini', 'entropy'],
    'max_depth':  [None, 2, 4, 6, 8, 10],
    'max_features': [None, 'sqrt', 'log2', 0.2, 0.4, 0.6, 0.8],
    'splitter': ['best', 'random']
}

clf = GridSearchCV(
    estimator=DecisionTreeClassifier(),
    param_grid=params,
    cv=5,
    n_jobs=5,
    verbose=1,
)

clf.fit(x_train_dt, y_train)
print(f"Best parameters from Grid Search CV : \n{clf.best_params_}")
```

```
Fitting 5 folds for each of 168 candidates, totalling 840 fits

Best parameters from Grid Search CV :
{'criterion': 'gini', 'max_depth': 6, 'max_features': None, 'splitter':
 'best'}
```

Training model based on the parameters we got from Grid SearchCV.

```
dtc = DecisionTreeClassifier(criterion='entropy',max_depth=8,max_featur
es= 0.8,splitter='best')
dtc.fit(x_train_dt,y_train )
dtcprediction = dtc.predict(x_test_dt)
print(accuracy_score(y_test, dtcprediction))
print(confusion_matrix(y_test, dtcprediction))
print(classification_report(y_test, dtcprediction))
```

```
0.8795327754532776
[[4631  276]
 [ 415  414]]
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      4907
           1       0.60      0.50      0.55       829

    accuracy                           0.88      5736
   macro avg       0.76      0.72      0.74      5736
weighted avg       0.87      0.88      0.87      5736
```

From the decision tree we have better precision, recall, accuracy and thus better f1 score. Hence, decision tree is performing better than logistic regression.

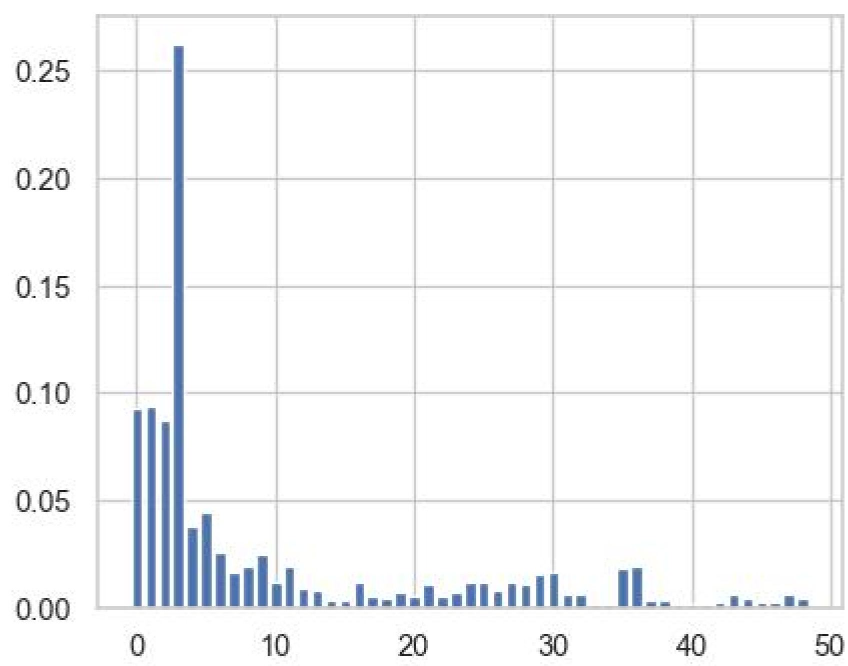
AUC Curve : Figure 1
Precision Recall Curve : Figure 2

Random Forest

Feature Selection

```
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)
importance_rfc = rfc.feature_importances_
features_rfc = []
for i,v in enumerate(importance_rfc):
    if v >0.01:
        #print(f"Feature {i} variable {balanced_train.columns[i]} score
 {v}")
        features_rfc.append(balanced_train.columns[i])
print(f"Important features from random forest :\n{features_rfc}")
pyplot.bar([x for x in range(len(importance_rfc))], importance_rfc)
pyplot.show()
#selecting important features
x_train_rf = x_train[features_rfc]
x_test_rf = x_test[features_rfc]
```

```
Important features from random forest :
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous',
'cons.conf.idx', 'emp.var.rate', 'euribor3m', 'nr.employed', 'cons.pric
e.idx', 'job_management', 'job_technician', 'education_secondary', 'edu
cation_tertiary', 'marital_married', 'marital_single', 'housing_no', 'h
ousing_yes', 'poutcome_failure', 'poutcome_success']
```



Hyperparameter Tuning

```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'n_estimators': [100, 200, 300, 1000]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid =param_grid, cv =
 3, n_jobs = -1, verbose = 2)
grid_search.fit(x_train_rf, y_train)
grid_search.best_params_

Fitting 3 folds for each of 32 candidates, totalling 96 fits

{'bootstrap': True, 'max_depth': 110, 'max_features': 3, 'n_estimators':
 1000}

rfc = RandomForestClassifier(bootstrap=True,max_depth=80,max_features=3,
n_estimators=200)
rfc.fit(x_train_rf, y_train)
rfcpredictions = rfc.predict(x_test_rf)
print(f"Training accuracy {accuracy_score(y_train, rfc.predict(x_train_
rf))}")
print(f"Testing set accuracy {accuracy_score(y_test, rfcpredictions )}")
print(confusion_matrix(y_test, rfcpredictions ))
print(classification_report(y_test, rfcpredictions ))

Training accuracy 1.0
Testing set accuracy 0.8877266387726639
[[4719  188]
 [ 456  373]]
              precision    recall  f1-score   support

           0       0.91      0.96      0.94      4907
           1       0.66      0.45      0.54       829

    accuracy                           0.89      5736
   macro avg       0.79      0.71      0.74      5736
weighted avg       0.88      0.89      0.88      5736
```

We are getting best performance from Random Forest, so we would also use cross validation to make our model more credible.

```
# from sklearn.model_selection import cross_val_score
# scores = cross_val_score(rfc, x_train_rf, y_train, cv=5)
# print(scores)
```

```python
# K-Fold Cross-Validation

def cross_validation(model, _X, _y, _cv=5):
      '''Function to perform 5 Folds Cross-Validation
       Parameters
       ----------
      model: Python Class, default=None
              This is the machine learning algorithm to be used for tra
ining.
      _X: array
           This is the matrix of features.
      _y: array
           This is the target variable.
      _cv: int, default=5
          Determines the number of folds for cross-validation.
       Returns
       -------
       The function returns a dictionary containing the metrics 'accura
cy', 'precision',
       'recall', 'f1' for both training set and validation set.
      '''
      _scoring = ['accuracy', 'precision', 'recall', 'f1']
      results = cross_validate(estimator=model,
                               X=_X,
                               y=_y,
                               cv=_cv,
                               scoring=_scoring,
                               return_train_score=True)

      return {"Training Accuracy scores": results['train_accuracy'],
              "Mean Training Accuracy": results['train_accuracy'].mean()
*100,
              "Training Precision scores": results['train_precision'],
              "Mean Training Precision": results['train_precision'].mea
n(),
              "Training Recall scores": results['train_recall'],
              "Mean Training Recall": results['train_recall'].mean(),
              "Training F1 scores": results['train_f1'],
              "Mean Training F1 Score": results['train_f1'].mean(),
              "Validation Accuracy scores": results['test_accuracy'],
              "Mean Validation Accuracy": results['test_accuracy'].mean
()*100,
              "Validation Precision scores": results['test_precision'],
              "Mean Validation Precision": results['test_precision'].me
an(),
              "Validation Recall scores": results['test_recall'],
              "Mean Validation Recall": results['test_recall'].mean(),
              "Validation F1 scores": results['test_f1'],
              "Mean Validation F1 Score": results['test_f1'].mean()
```

```
              }
cross_validation(rfc, x_train_rf, y_train, _cv=5)

{'Training Accuracy scores': array([1., 1., 1., 1., 1.]),
 'Mean Training Accuracy': 100.0,
 'Training Precision scores': array([1., 1., 1., 1., 1.]),
 'Mean Training Precision': 1.0,
 'Training Recall scores': array([1., 1., 1., 1., 1.]),
 'Mean Training Recall': 1.0,
 'Training F1 scores': array([1., 1., 1., 1., 1.]),
 'Mean Training F1 Score': 1.0,
 'Validation Accuracy scores': array([0.88603182, 0.88145565, 0.8893005
 , 0.88603182, 0.89210985]),
 'Mean Validation Accuracy': 88.6985927646624,
 'Validation Precision scores': array([0.64631579, 0.64619165, 0.674943
57, 0.65555556, 0.68351648]),
 'Mean Validation Precision': 0.6613046082657583,
 'Validation Recall scores': array([0.46374622, 0.39668175, 0.45098039,
 0.44494721, 0.46978852]),
 'Mean Validation Recall': 0.4452288189270595,
 'Validation F1 scores': array([0.54001759, 0.49158879, 0.54068716, 0.5
3009883, 0.5568487 ]),
 'Mean Validation F1 Score': 0.5318482140004461}
```

After applying cross validation, we are getting some what real estimates.

AUC Curve : Figure 1
Precision Recall Curve : Figure 2

Linear SVC

Finding a linear hyperplane that tries to separate two classes.

```
svc_linear = LinearSVC(C=1.0, class_weight=None, dual=True, fit_interce
pt=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=123, tol=0.0001,
          verbose=0)
svc_linear.fit(x_train,y_train)
svc_linear_predictions = svc_linear.predict(x_test)
print(accuracy_score(y_test, svc_linear_predictions))
print(confusion_matrix(y_test, svc_linear_predictions))
print(classification_report(y_test, svc_linear_predictions))

0.8559972105997211
[[4898    9]
 [ 817   12]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      4907
```

```
           1       0.57      0.01      0.03       829

    accuracy                           0.86      5736
   macro avg       0.71      0.51      0.48      5736
weighted avg       0.82      0.86      0.79      5736
```

SVC

Finding a complex hyperplane that tries to separate the classes.

```
#  SVM - Support Vector Machines balance check on unbalance test
svc= SVC(kernel='poly', random_state=123)
svc.fit(x_train,y_train)
svcpredictions = svc.predict(x_test)
print(accuracy_score(y_test, svcpredictions))
print(confusion_matrix(y_test, svcpredictions))
print(classification_report(y_test, svcpredictions))

0.8554741980474198
[[4907    0]
 [ 829    0]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      4907
           1       0.00      0.00      0.00       829

    accuracy                           0.86      5736
   macro avg       0.43      0.50      0.46      5736
weighted avg       0.73      0.86      0.79      5736
```

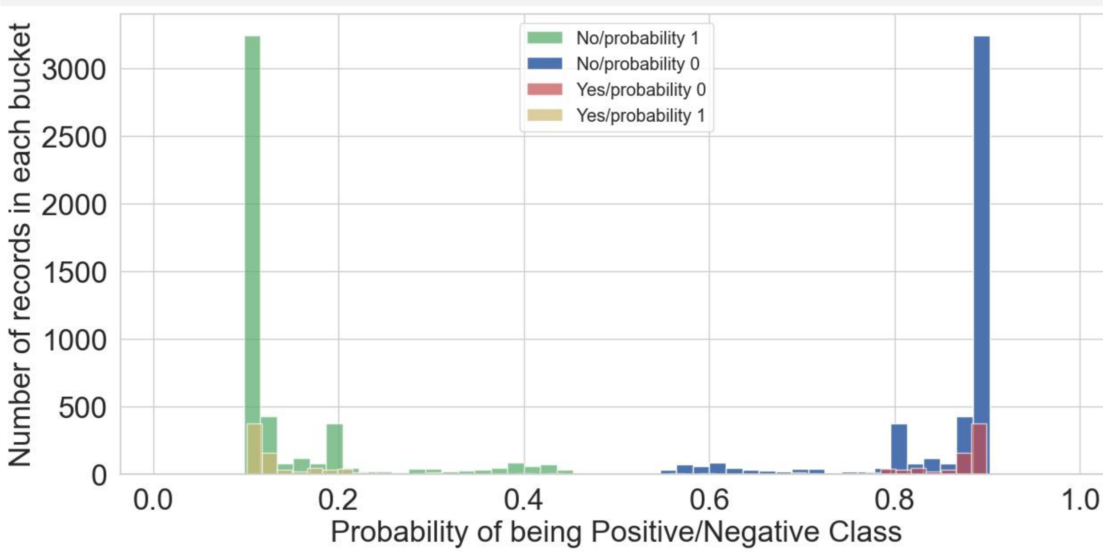
Naive Bayes

Naive Bayes a naive assumption that all the features are independent of each other and thus by reducing the complexity of computing conditional probabilities it evaluates the probability of 0 and 1.

```
param_grid_nb = {
    'var_smoothing': np.logspace(0,-9, num=100)
}
nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_gr
id_nb, verbose=1, cv=10, n_jobs=-1)
nbModel_grid.fit(x_train, y_train)
print(nbModel_grid.best_estimator_)

from sklearn.naive_bayes import GaussianNB
modelNB = GaussianNB(var_smoothing=0.04328761281083057)
modelNB.fit(x_train,y_train)
print(f"Model score is {modelNB.score(x_test,y_test)}")
```

```
def modelProbability(prediction0,prediction1,y):
    plt.figure(figsize=(15,7))
    plt.hist(prediction1[y==0], bins=50, label='No/probability 1', alph
a=0.7, color='g')
    plt.hist(prediction0[y==0], bins=50, label='No/probability 0')
    plt.hist(prediction0[y==1], bins=50, label='Yes/probability 0', alp
ha=0.7, color='r')
    plt.hist(prediction1[y==1], bins=50, label='Yes/probability 1', alp
ha=0.7, color='y')
    plt.xlabel('Probability of being Positive/Negative Class', fontsize
=25)
    plt.ylabel('Number of records in each bucket', fontsize=25)
    plt.legend(fontsize=15)
    plt.tick_params(axis='both', labelsize=25, pad=5)
    plt.show()
pred1=modelNB.predict_proba(x_test)[:,0]
pred2 = modelNB.predict_proba(x_test)[:,1]
modelProbability(pred1,pred2,y_test)

#modelling
def modelEvaluation(model,x,y):
    print('test set evaluation: ')
    y_pred = model.predict(x)
    print(accuracy_score(y, y_pred))
    print(confusion_matrix(y, y_pred))
    print(classification_report(y, y_pred))

modelEvaluation(modelNB,x_test,y_test)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits

GaussianNB(var_smoothing=0.0657933224657568)
Model score is 0.8561715481171548
```



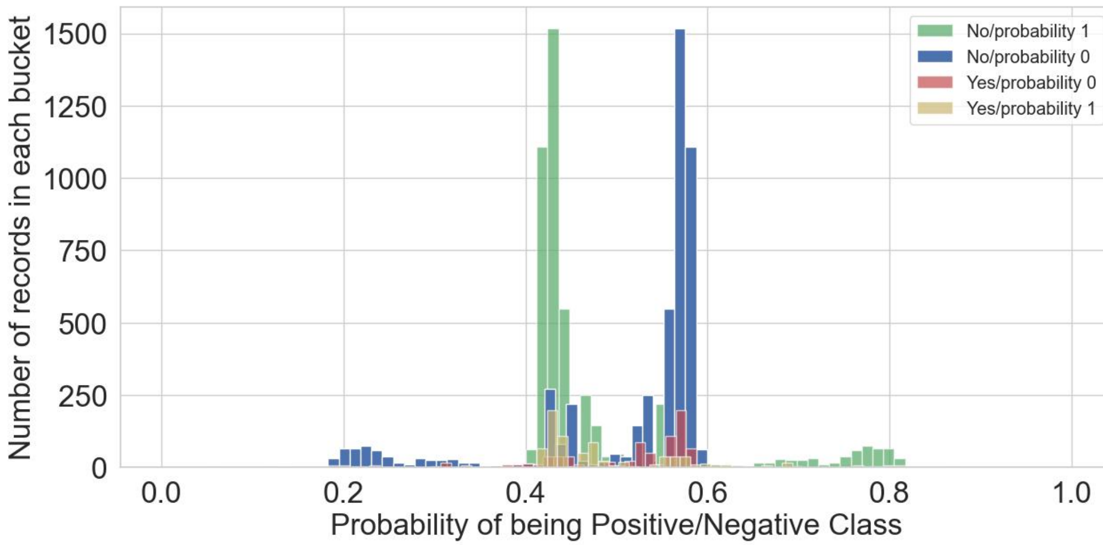
```
test set evaluation:
0.8561715481171548
[[4892   15]
 [ 810   19]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      4907
           1       0.56      0.02      0.04       829

    accuracy                           0.86      5736
   macro avg       0.71      0.51      0.48      5736
weighted avg       0.81      0.86      0.80      5736
```

For balanced

For balanced dataset, as we can see there is a slight improvement in performance. The f1 score has improved and also, the yellow bars are now slightly shifted towards right side.

```
param_grid_nb = {
    'var_smoothing': np.logspace(0,-9, num=100)
}

from sklearn.naive_bayes import GaussianNB
modelNB = GaussianNB(var_smoothing=0.04328761281083057)
modelNB.fit(bx_train,by_train)
print(f"Model score is {modelNB.score(x_test,y_test)}")
def modelProbability(prediction0,prediction1,y):
    plt.figure(figsize=(15,7))
    plt.hist(prediction1[y==0], bins=50, label='No/probability 1', alph
a=0.7, color='g')
    plt.hist(prediction0[y==0], bins=50, label='No/probability 0')
    plt.hist(prediction0[y==1], bins=50, label='Yes/probability 0', alp
ha=0.7, color='r')
    plt.hist(prediction1[y==1], bins=50, label='Yes/probability 1', alp
ha=0.7, color='y')
    plt.xlabel('Probability of being Positive/Negative Class', fontsize
=25)
    plt.ylabel('Number of records in each bucket', fontsize=25)
    plt.legend(fontsize=15)
    plt.tick_params(axis='both', labelsize=25, pad=5)
    plt.show()
pred1=modelNB.predict_proba(x_test)[:,0]
pred2 = modelNB.predict_proba(x_test)[:,1]
modelProbability(pred1,pred2,y_test)

#modelling
def modelEvaluation(model,x,y):
    print('test set evaluation: ')
    y_pred = model.predict(x)
```

```
    print(accuracy_score(y, y_pred))
    print(confusion_matrix(y, y_pred))
    print(classification_report(y, y_pred))

modelEvaluation(modelNB,x_test,y_test)

Model score is 0.693863319386332
```




```
test set evaluation:
0.693863319386332
[[3690 1217]
 [ 539  290]]
              precision    recall  f1-score   support

           0       0.87      0.75      0.81      4907
           1       0.19      0.35      0.25       829

    accuracy                           0.69      5736
   macro avg       0.53      0.55      0.53      5736
weighted avg       0.77      0.69      0.73      5736
```

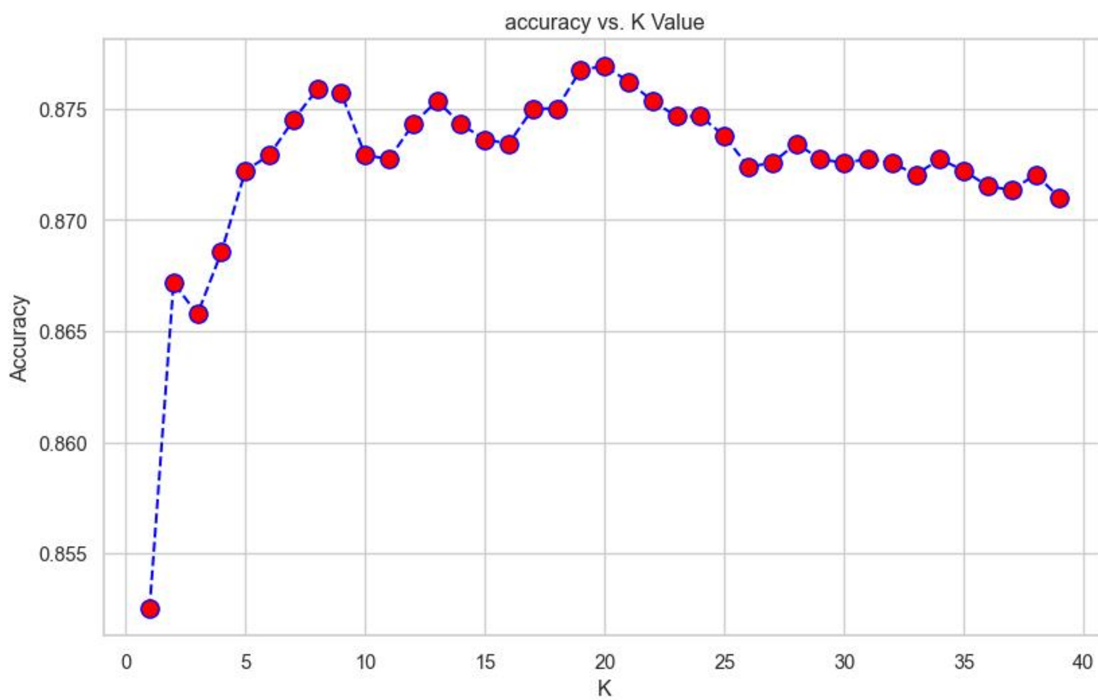
As we can see from the graph for the red and yellow bars for yes(1 term deposit) are coming on the opposite sides which is not expected.

AUC Curve : Figure 1
Precision Recall Curve : Figure 2

KNN

Using the k - nearest neighbours we try to predict the testing dataset. Now to find the optimal k value we will look into precision and accuracy curve for different k values.

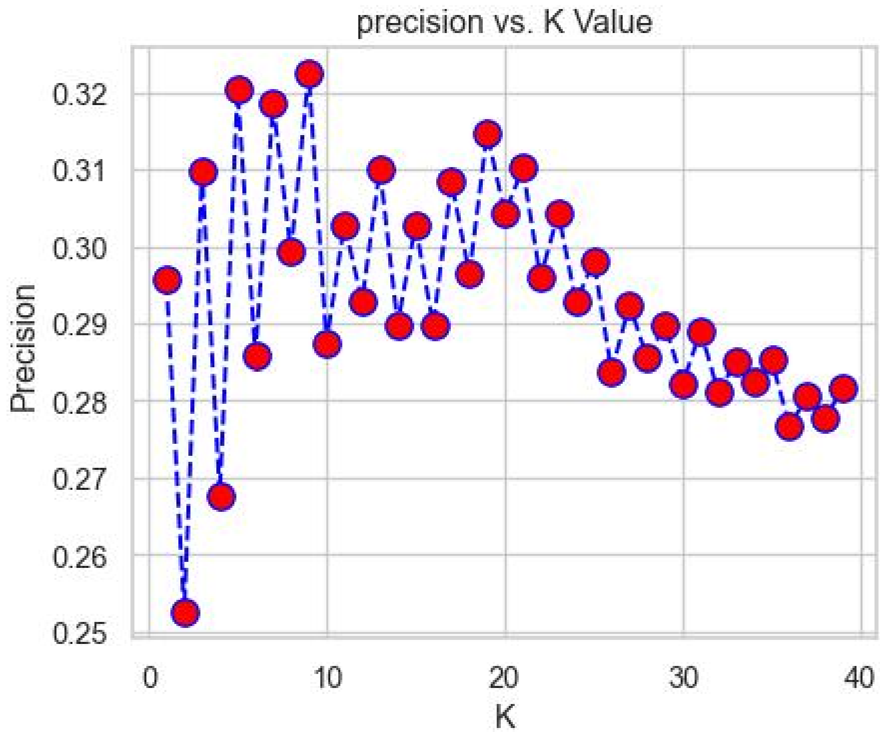
```
acc = []
prec = []
# Will take some time
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(x_train,y_train)
    y_pred = neigh.predict(x_test)
    acc.append(metrics.accuracy_score(y_test, y_pred))
    prec.append((metrics.average_precision_score(y_test, y_pred)))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red',  markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))

Maximum accuracy:- 0.8769177126917713 at K = 19
```



Accuracy curve for different k values

```
#plt.figure(figsize=(10,6))
plt.plot(range(1,40),prec,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('precision vs. K Value')
plt.xlabel('K')
plt.ylabel('Precision')
print("Maximum Precision:-",max(prec),"at K =",prec.index(max(prec)))
```

```
Maximum Precision:- 0.32247407633937064 at K = 8
```



Precision curve for different k values

Based on the above plot, optimal k value is 3, with maximum f1 score of 0.64.

```
mrroger = 3
knn = KNeighborsClassifier(n_neighbors=mrroger) # instantiate with n va
lue given
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
#y_pred = knn.predict_proba(x_test)
print(f"Train set accuracy {accuracy_score(y_train, knn.predict(x_trai
n))}")
print(f"Test set accuracy {accuracy_score(y_test, y_pred)}")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

Train set accuracy 0.9197611576011158
Test set accuracy 0.8657601115760112
[[4626  281]
 [ 489  340]]
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      4907
           1       0.55      0.41      0.47       829
```

```
    accuracy                           0.87      5736
   macro avg       0.73      0.68      0.70      5736
weighted avg       0.85      0.87      0.86      5736
```

AUC Curve : Figure 1
Precision Recall Curve : Figure 2

ROC -AUC Curve

```
from sklearn.metrics import roc_auc_score, roc_curve

# Instantiate the classfiers and make a list
classifiers = [LogisticRegression(random_state=123),
        DecisionTreeClassifier(criterion='entropy',max_depth=8,max_feat
ures= 0.8,splitter='best'),
        RandomForestClassifier(bootstrap=True,max_depth=80,max_features
=3,n_estimators=200),
        #SVC(kernel='poly', random_state=123),
        GaussianNB(var_smoothing=0.04328761281083057),
        KNeighborsClassifier(n_neighbors=mrroger)]

# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])
X_train = x_train
X_test = x_test
# Train the models and record the results
for cls in classifiers:
    model = cls.fit(X_train, y_train)
    yproba = model.predict_proba(X_test)[::,1]

    fpr, tpr, _ = roc_curve(y_test,  yproba)
    auc = roc_auc_score(y_test, yproba)

    result_table = result_table.append({'classifiers':cls.__class__.__n
ame__,
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)

# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)

fig = plt.figure(figsize=(8,6))

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc
']))
```

```
plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("Flase Positive Rate", fontsize=15)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.show()
```

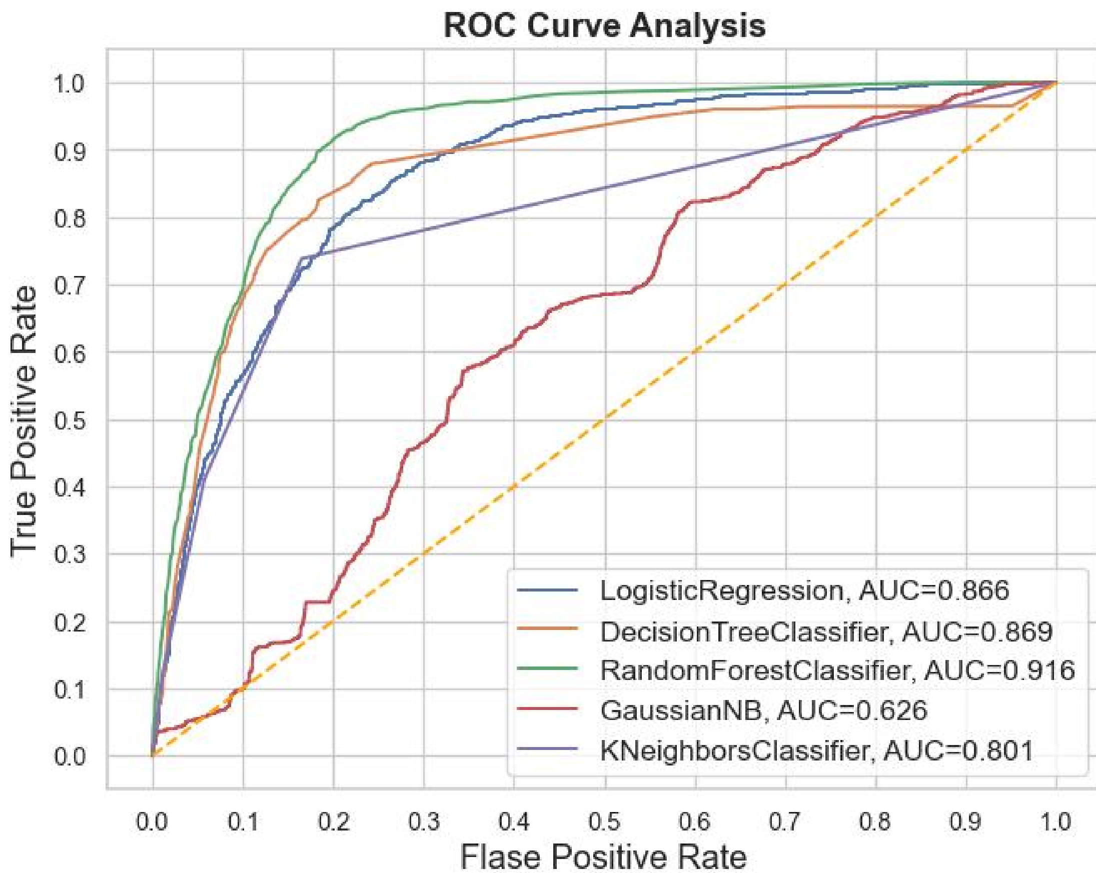


Figure 1: AUC ROC Curve for all Models

Precision Recall Curve

In imbalance problem since we have a high number of Negatives, this makes the False Posiitve Rate as low, resulting in the shift of ROC AUC Curve towards left, which is slightly misleading.

So in imbalance problem we usually make sure to look at precision recall curve as well.

```
#Logistic Regression
lr_probs = lr.predict_proba(X_test_final)
lr_probs = lr_probs[:, 1]
yhat = lr.predict(X_test_final)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Ski
ll')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic')
# axis labels

#Decision Tree Classifier
dtc_probs = dtc.predict_proba(x_test_dt)
dtc_probs = dtc_probs[:, 1]
yhat = dtc.predict(x_test_dt)
dtc_precision, dtc_recall, _ = precision_recall_curve(y_test, dtc_probs)
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot(dtc_recall, dtc_precision, marker='.', label='Decision Tree
')

#Random Forest Classifier
rfc_probs = rfc.predict_proba(x_test_rf)
rfc_probs = rfc_probs[:, 1]
yhat = rfc.predict(x_test_rf)
rfc_precision, rfc_recall, _ = precision_recall_curve(y_test, rfc_probs)
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot(rfc_recall, rfc_precision, marker='.', label='Random Forest
')

#Gaussian Model
nb_probs = modelNB.predict_proba(x_test)
# keep probabilities for the positive outcome only
nb_probs = nb_probs[:, 1]
# predict class values
yhat = modelNB.predict(x_test)
nb_precision, nb_recall, _ = precision_recall_curve(y_test, nb_probs)
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot(nb_recall, nb_precision, marker='.', label='Naive Bayes')

#knn
knn_probs = knn.predict_proba(x_test)
knn_probs = knn_probs[:, 1]
yhat = knn.predict(x_test)
knn_precision, knn_recall, _ = precision_recall_curve(y_test, knn_probs)
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot(knn_recall, knn_precision, marker='.', label='KNN')
```

```
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

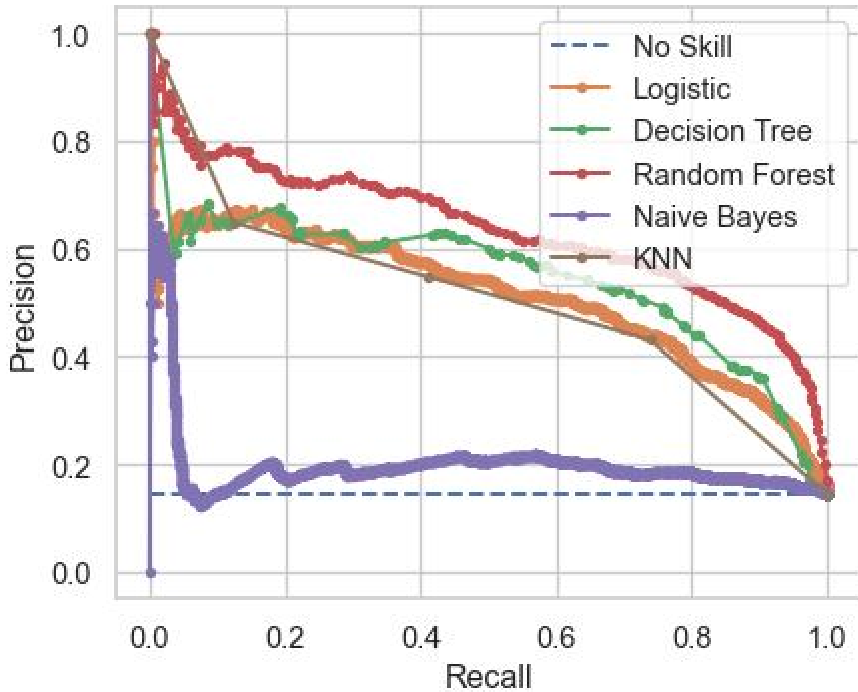


Figure 2: Precision Recall Curve for all Models

As per the ROC Curve and Precision Recall curve, KNN is performing best. But after combining these results with precision recall curve, we suggest using Random Forest for our problem.

Summary

Table 1: Summary of Models

Model	Accuracy	Precision	Recall	AUC
Logistic(Cutoff=0.25)	0.88	0.51	0.58	0.872
Logistic (Balanced-Train)	0.85	0.49	0.54	
Decision Tree	0.91	0.66	0.47	0.923
Random Forest	0.88	0.66	0.46	0.913
SVC	0.89	0.75	0.15	

Model	Accuracy	Precision	Recall	AUC
Linear SVC	0.89	0.62	0.16	
Gaussian Bayes	0.88	0.50	0.25	0.841
KNN	0.92	0.78	0.54	0.965
Naive Bayes	0.85	0.56	0.02	
Naive Bayes (Balanced-Train)	0.69	0.19	0.35	

See Table 1.