Machine Learning Classification Project

Link : http://archive.ics.uci.edu/ml/datasets/Bank+Marketing (http://archive.ics.uci.edu/ml/datasets/Bank+Marketing)

```
Attribute Information:

Input variables:
# bank client data:
1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemai
d','management','retired','self-employed','services','student','technician','unempl
oyed','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','illiter
ate','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')
# related with the last contact of the current campaign:
8 - contact: contact communication type (categorical: 'cellular','telephone')
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'no
v', 'dec')
10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','th
u','fri')
11 - duration: last contact duration, in seconds (numeric). Important note: this at
tribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, th
e 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 purpos
es and should be discarded if the intention is to have a realistic predictive mode
l.
# other attributes:
12 - campaign: number of contacts performed during this campaign and for this clien
t (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 clien
t (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failur
e','nonexistent','success')
# social and economic context attributes
16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):
21 - y - has the client subscribed a term deposit? (binary: 'yes','no')
```

In [304]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [305]:
```
df=pd.read_csv("Classification_dataset.csv")
```

In [306]:
```
df.head()
```

Out[306]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
0	51	blue-collar	married	basic.9y	no	yes	yes	telephone	may	wed
1	31	admin.	divorced	high.school	no	no	no	telephone	may	mon
2	32	blue-collar	divorced	basic.9y	no	no	no	telephone	may	wed
3	36	admin.	married	university.degree	no	yes	no	cellular	aug	wed
4	43	admin.	married	high.school	no	yes	no	cellular	jul	mon

5 rows × 21 columns

In [307]:
```
df.shape
```

Out[307]:
```
(20594, 21)
```

In [308]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20594 entries, 0 to 20593
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             20594 non-null  int64  
 1   job             20233 non-null  object 
 2   marital         20225 non-null  object 
 3   education       20594 non-null  object 
 4   default         20594 non-null  object 
 5   housing         20594 non-null  object 
 6   loan            20211 non-null  object 
 7   contact         20594 non-null  object 
 8   month           20594 non-null  object 
 9   day_of_week     20594 non-null  object 
 10  duration        20594 non-null  int64  
 11  campaign        20594 non-null  int64  
 12  pdays           20594 non-null  int64  
 13  previous        20594 non-null  int64  
 14  poutcome        20594 non-null  object 
 15  emp.var.rate    20594 non-null  float64
 16  cons.price.idx  20594 non-null  float64
 17  cons.conf.idx   20594 non-null  float64
 18  euribor3m       20195 non-null  float64
 19  nr.employed     20594 non-null  float64
 20  y               20594 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 3.3+ MB
```

In [309]:
```
# looking for missing values
df.isnull().any()
```

Out[309]:
```
age              False
job               True
marital           True
education        False
default          False
housing          False
loan              True
contact          False
month            False
day_of_week      False
duration         False
campaign         False
pdays            False
previous         False
poutcome         False
emp.var.rate     False
cons.price.idx   False
cons.conf.idx    False
euribor3m         True
nr.employed      False
y                False
dtype: bool
```

In [310]:
```
# looks like job,marital,loan and euribor3m has missing values
# lets check num of missing values in each column
a= ['job','marital','loan','euribor3m']
for values in a:
    print("Missing values in %s is equal to %s" % (values,df[values].isnull().
sum()))
```

```
Missing values in job is equal to 361
Missing values in marital is equal to 369
Missing values in loan is equal to 383
Missing values in euribor3m is equal to 399
```

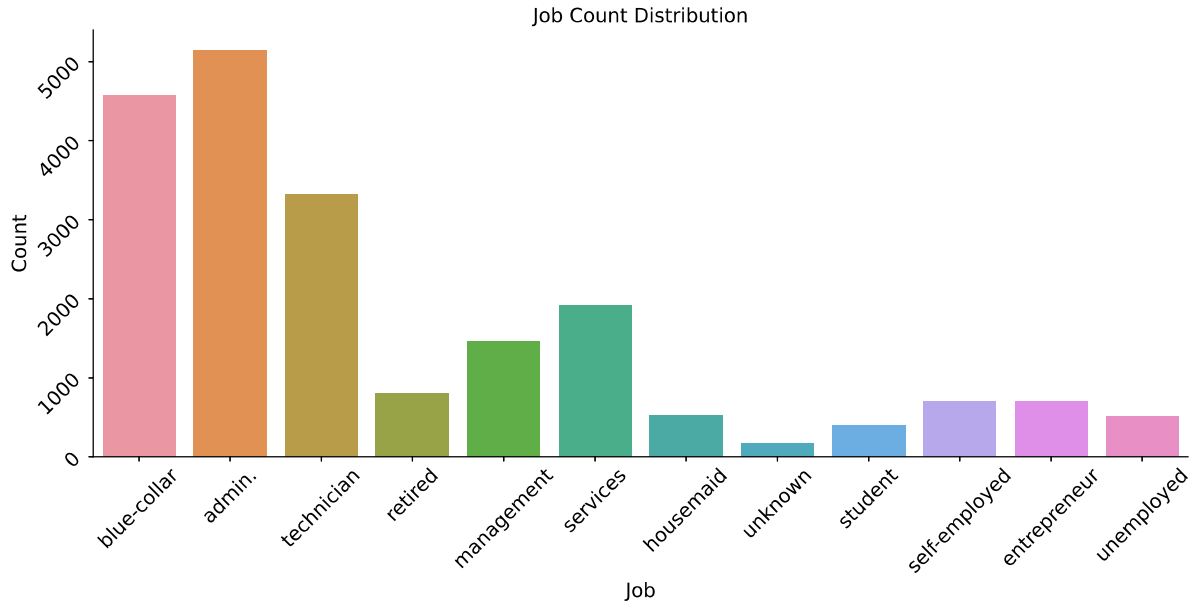
In [311]:
```
# to handle missing values in numerical column we will replace the value with
 median of the column
# Replace using median
median = df['euribor3m'].median()
df['euribor3m'].fillna(median, inplace=True)
df['euribor3m'].isnull().sum()
```

Out[311]:
```
0
```

To handle missing values in each categorical column we are usign the frequency substitution thereby substituting the values with max frequency value

In [312]:
```
# so  to check frequency we do some seasborn plots
```

In [313]:
```
#we have 3 columns of type object and another column of type float which has m
issing values
# What kind of jobs clients of this bank have
fig, ax = plt.subplots()
fig.set_size_inches(15, 6)
sns.countplot(x = 'job', data = df)
ax.set_xlabel('Job', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Job Count Distribution', fontsize=15)
ax.tick_params(labelsize=15,rotation=45)
sns.despine()
```

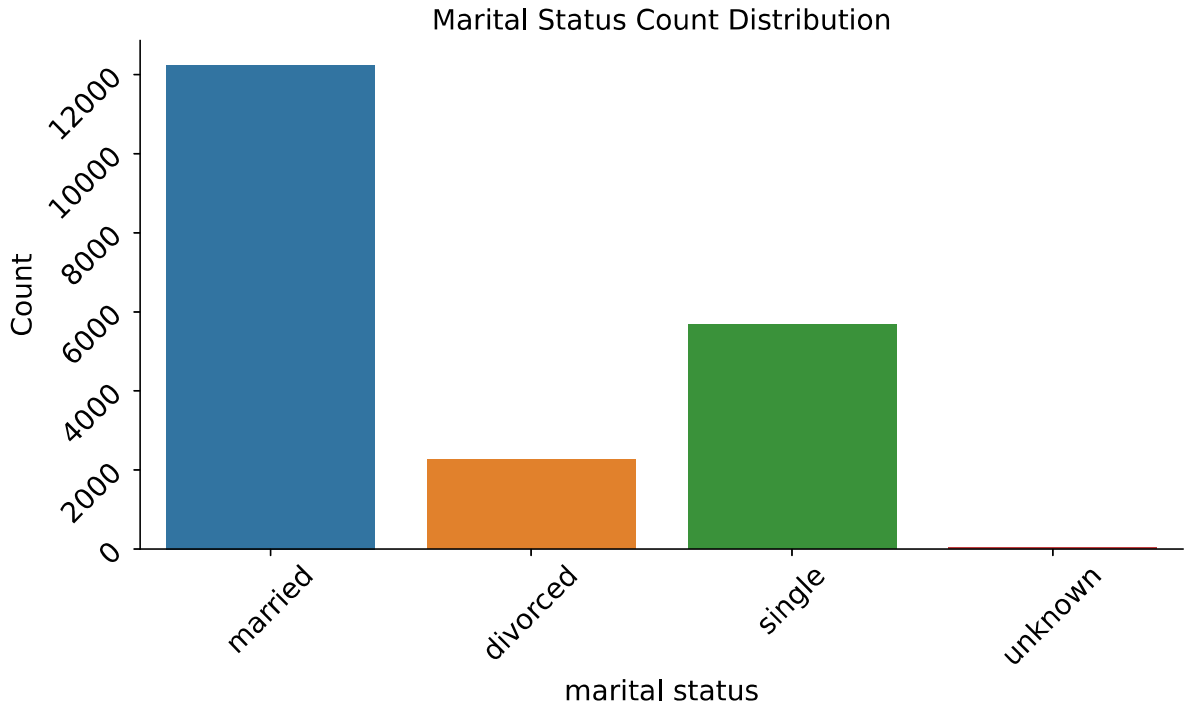


In [314]:
```
# replacing missing values by the admin as it has the highest frequency in thi
s column
df['job'].fillna('admin.', inplace=True)
df['job'].isnull().sum()
```

Out[314]: 0

also to check the number of unknowns in marital status column with respect to other categorical values

In [315]:
```
fig, ax = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'marital', data = df)
ax.set_xlabel('marital status', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Marital Status Count Distribution', fontsize=15)
ax.tick_params(labelsize=15,rotation=45)
sns.despine()
```

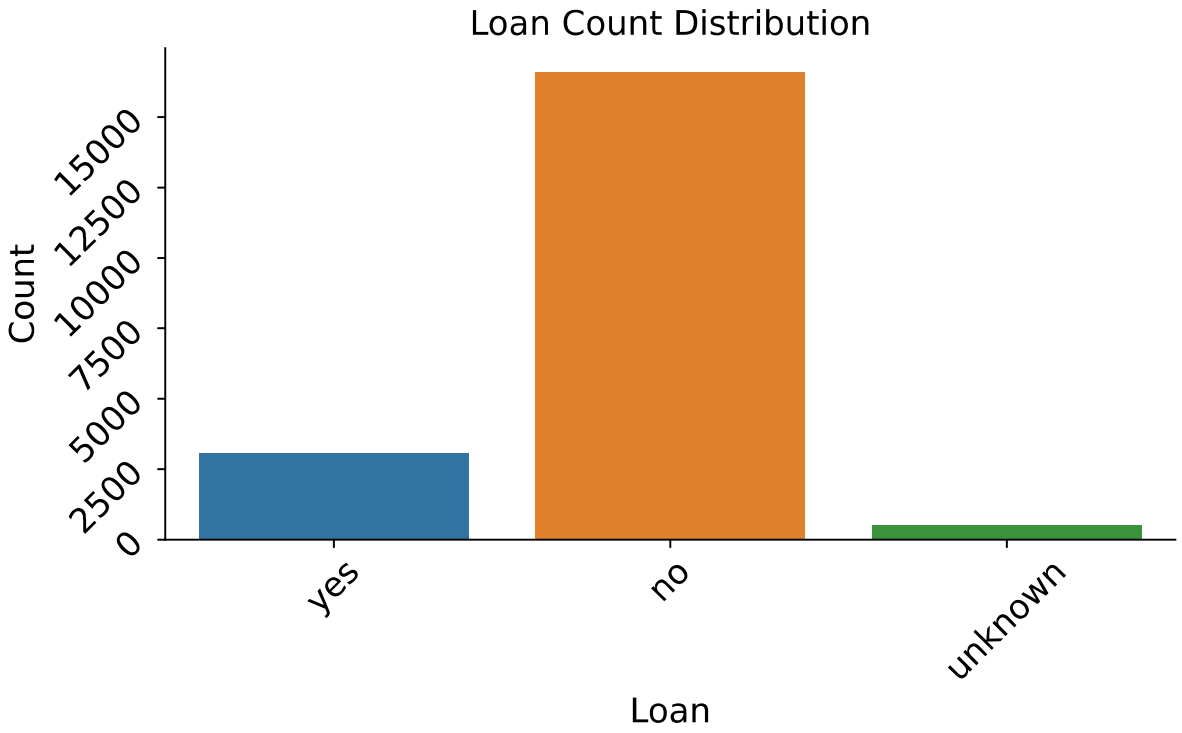


In [316]:
```
# replacing missing values by the married as it has the highest frequency in t
his column
df['marital'].fillna('married', inplace=True)
df['marital'].isnull().sum()
```

Out[316]: 0

to check the number of unknowns in Loan column with respect to other categorical values

In [317]:
```
fig, ax = plt.subplots()
fig.set_size_inches(8, 4)
sns.countplot(x = 'loan', data = df)
ax.set_xlabel('Loan', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Loan Count Distribution', fontsize=15)
ax.tick_params(labelsize=15,rotation=45)
sns.despine()
```



In [318]:
```
# replacing missing values by the 'no' as it has the highest frequency in this column
df['loan'].fillna('no', inplace=True)
df['loan'].isnull().sum()
```

Out[318]: 0

We have handled all the missing values

bifurcating datasets to seperate different types of attributes to help and understand them better

In [319]:
```
#bank client data set
df_client=df.iloc[:,0:7]
df_client.head()
```

Out[319]:

	age	job	marital	education	default	housing	loan
0	51	blue-collar	married	basic.9y	no	yes	yes
1	31	admin.	divorced	high.school	no	no	no
2	32	blue-collar	divorced	basic.9y	no	no	no
3	36	admin.	married	university.degree	no	yes	no
4	43	admin.	married	high.school	no	yes	no

In [320]:
```
#checking how many unique values each column related to client has
for r in df_client.columns:
    print("type of %s -->"%r,df_client[r].unique(),'\n')
```

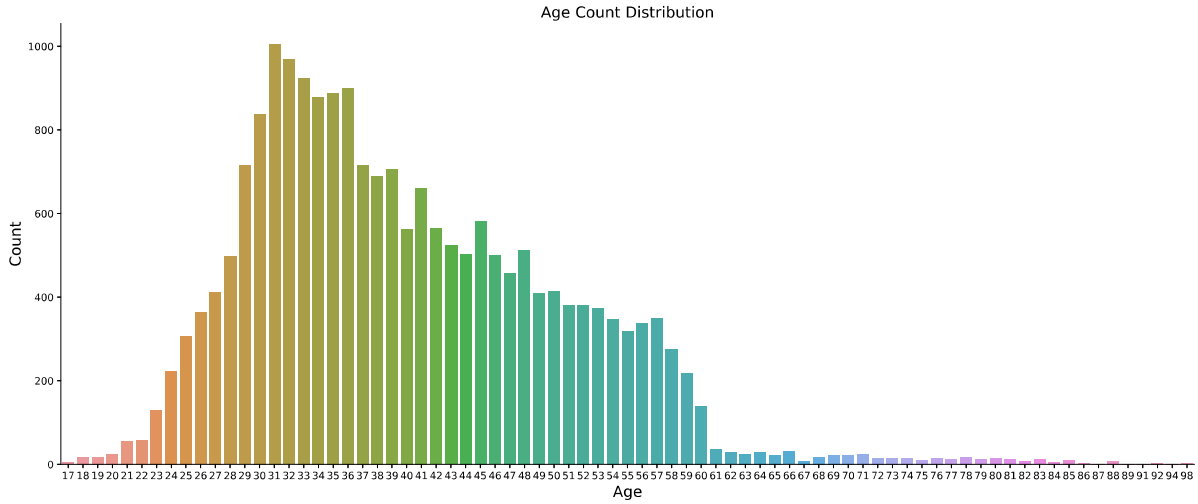
```
type of age --> [51 31 32 36 43 38 29 53 37 28 49 35 52 79 48 41 34 40 92 45
25 57 33 42
 27 56 30 50 39 65 47 19 59 69 44 46 24 26 54 22 58 78 60 66 72 23 61 80
 98 71 21 70 55 76 20 63 77 64 68 81 75 73 82 88 85 18 62 84 74 83 67 17
 87 91 94 86 89]

type of job --> ['blue-collar' 'admin.' 'technician' 'retired' 'management'
'services'
 'housemaid' 'unknown' 'student' 'self-employed' 'entrepreneur'
 'unemployed']

type of marital --> ['married' 'divorced' 'single' 'unknown']

type of education --> ['basic.9y' 'high.school' 'university.degree' 'professi
onal.course'
 'basic.4y' 'unknown' 'basic.6y' 'illiterate']

type of default --> ['no' 'unknown' 'yes']

type of housing --> ['yes' 'no' 'unknown']

type of loan --> ['yes' 'no' 'unknown']
```

Feature : AGE

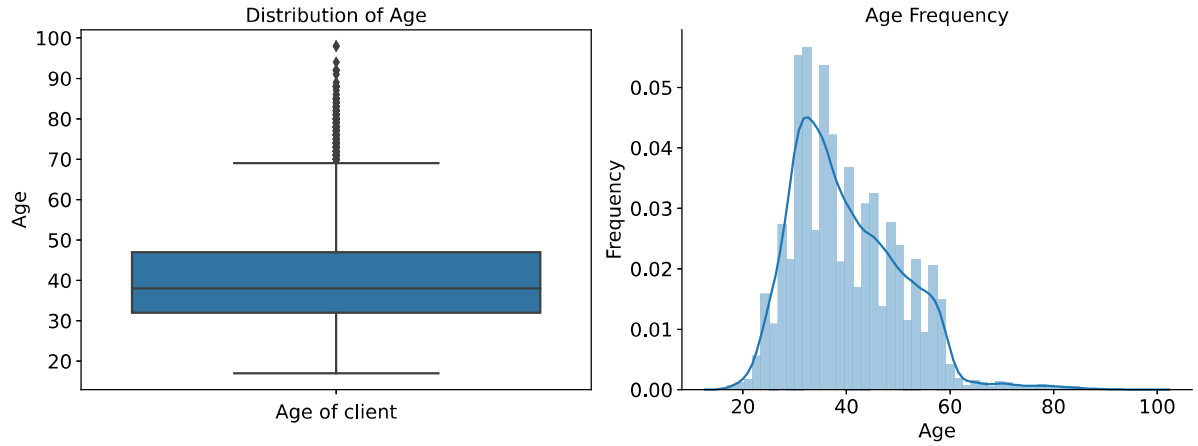
In [321]:
```
# Inspecting Age column in the client dataframe
print('Min age: ', df_client['age'].max())
print('Max age: ', df_client['age'].min())
print('Mean age: ',round(df_client['age'].mean(),2))
```

```
Min age:  98
Max age:  17
Mean age:  39.98
```

In [322]:
```
fig, ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'age', data = df_client)
ax.set_xlabel('Age', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
sns.despine()
```



In [323]:
```
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 5))
sns.boxplot(x = 'age', data = df_client, orient = 'v', ax = ax1)
ax1.set_xlabel('Age of client', fontsize=15)
ax1.set_ylabel('Age', fontsize=15)
ax1.set_title('Distribution of Age', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(df_client['age'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('Age', fontsize=15)
ax2.set_ylabel('Frequency', fontsize=15)
ax2.set_title('Age Frequency', fontsize=15)
ax2.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```



In [324]:
```
#Calculate the outliers:
  # Interquartile range, IQR = Q3 - Q1
  # lower 1.5*IQR whisker = Q1 - 1.5 * IQR
  # Upper 1.5*IQR whisker = Q3 + 1.5 * IQR

print('Age of client above : ', df_client['age'].quantile(q = 0.75) +
                      1.5*(df_client['age'].quantile(q = 0.75) - df_client['ag
e'].quantile(q = 0.25)), 'are outliers')

print('Numerber of outliers: ', df_client[df_client['age'] > 69.6]['age'].coun
t())
print('Number of clients: ', len(df_client))
#Outliers in %
print('Percentage of Outliers in Age column is :', round(df_client[df_client[
'age'] > 69.6]['age'].count()*100/len(df_client),2), '%')
```

```
Age of client above :  69.5 are outliers
Numerber of outliers:  234
Number of clients:  20594
Percentage of Outliers in Age column is : 1.14 %
```

In [325]:
```
df.shape
```

Out[325]:
```
(20594, 21)
```

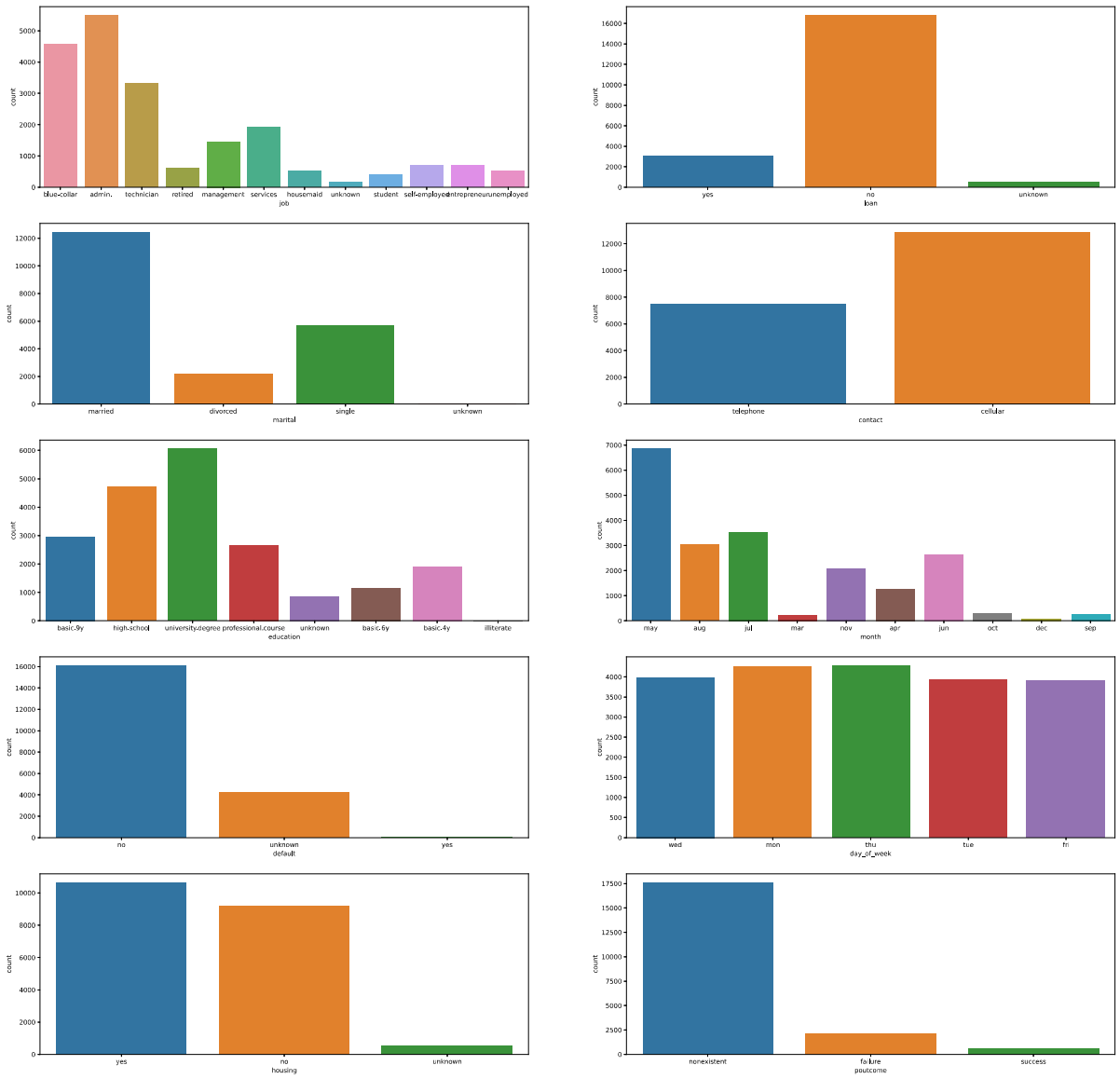
In [326]:
```
#droppping rows having outliers
ind=(df_client[df_client['age'] > 69.6]['age']).index
df.drop(index=ind,inplace=True)
df.shape
```

Out[326]:
```
(20360, 21)
```

count plot of All Features possible

In [327]:
```
# we do count plot of all the object variable to have a better understanding a
bout the nature of the client and its effect on term depostie decision
f, axes = plt.subplots(5, 2)
fig = plt.gcf()
fig.set_size_inches(30,30)

sns.countplot(x='job', data=df, ax=axes[0,0])
sns.countplot(x='marital', data=df, ax=axes[1,0])
sns.countplot(x='education', data=df, ax=axes[2,0])
sns.countplot(x='default', data=df, ax=axes[3,0])
sns.countplot(x='housing', data=df, ax=axes[4,0])
sns.countplot(x='loan', data=df, ax=axes[0,1])
sns.countplot(x='contact', data=df, ax=axes[1,1])
sns.countplot(x='month', data=df, ax=axes[2,1])
sns.countplot(x='day_of_week', data=df, ax=axes[3,1])
sns.countplot(x='poutcome', data=df, ax=axes[4,1])
```

Out[327]: <AxesSubplot:xlabel='poutcome', ylabel='count'>



we can see that by poutcome that most of the clients in this data set were not part of previous marketing campaigns

housing is equally divided between clients in this data set

most of the client who were target never defaulted

day_of_week doesnt look like playing important role here as it is fairly equally divided hence we need to analyze this more w.r.t to target variable=

having loan, job and marital status of the client may play more important roles

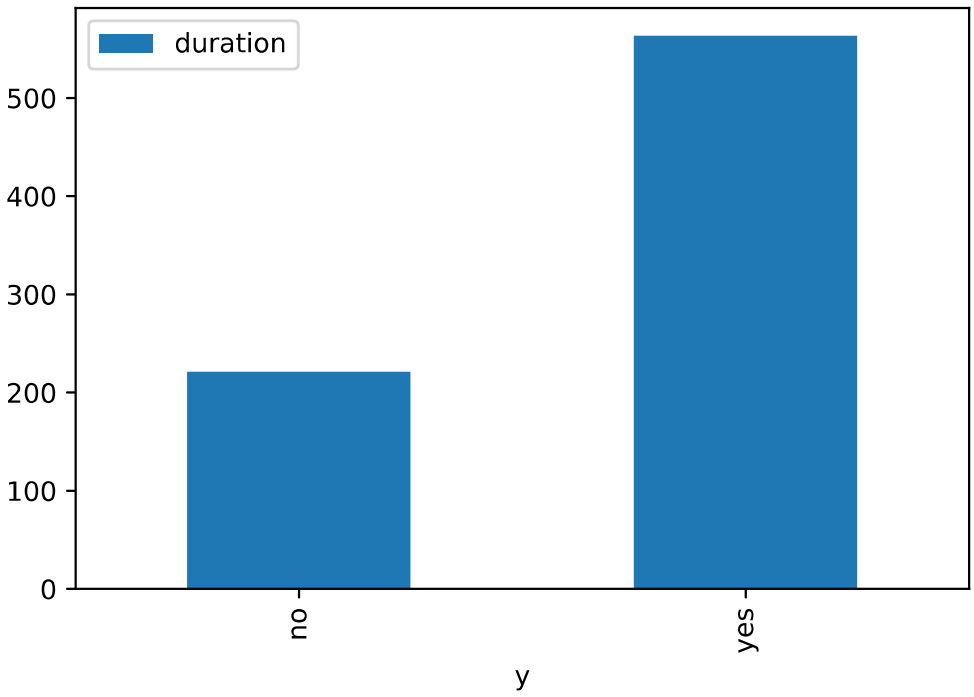
feature : Duration

In [328]:
```
df['duration'].describe()
```

Out[328]:
```
count    20360.000000
mean       257.929568
std        259.393665
min          0.000000
25%        102.000000
50%        179.000000
75%        317.000000
max       4199.000000
Name: duration, dtype: float64
```

In [329]:
```
# here we want to check that for increase in duration of last contact do the c
hances of saying yes increases or not.

df[['duration','y']].groupby('y').mean().plot(kind='bar')
#difference in mean shows that for high duration there will be mostly a positi
ve response
```

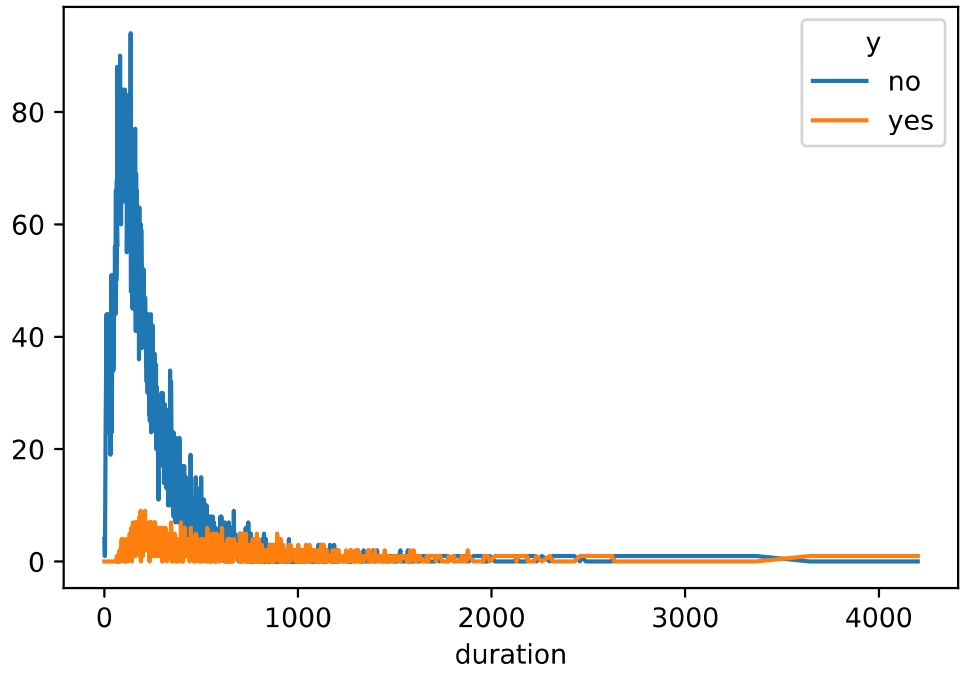
Out[329]: <AxesSubplot:xlabel='y'>



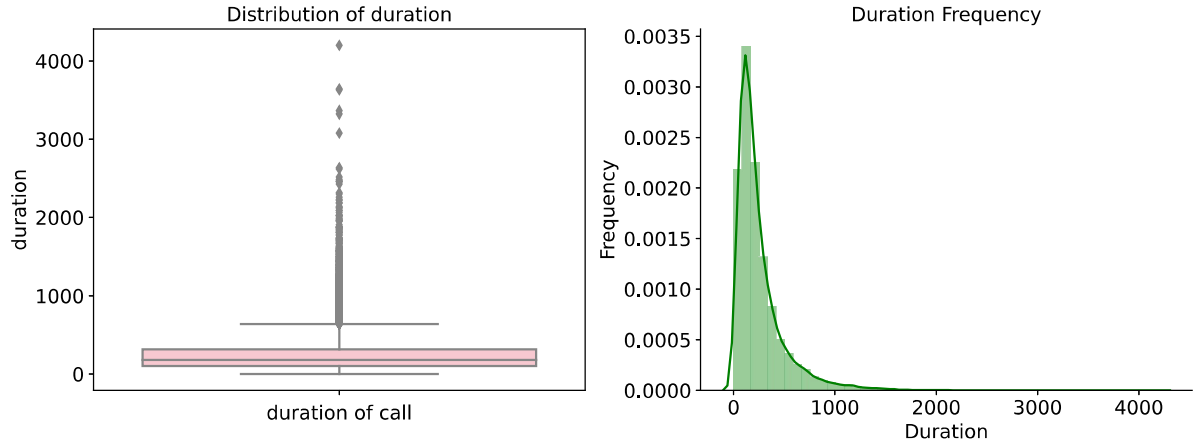
In [330]:
```
plt.scatter(df['y'],df['duration'],color='green',marker='2',s=500)
plt.show()
```

In [331]:
```
pd.crosstab(df['duration'],df['y']).plot(kind='line')
```

Out[331]: <AxesSubplot:xlabel='duration'>



In [332]:
```
fig, (ax3, ax4) = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 5))
sns.boxplot(x = 'duration', data = df, orient = 'v', ax = ax3,color='pink')
ax3.set_xlabel('duration of call', fontsize=15)
ax3.set_ylabel('duration', fontsize=15)
ax3.set_title('Distribution of duration', fontsize=15)
ax3.tick_params(labelsize=15)

sns.distplot(df['duration'], ax = ax4, color ='green')
sns.despine(ax = ax4)
ax4.set_xlabel('Duration', fontsize=15)
ax4.set_ylabel('Frequency', fontsize=15)
ax4.set_title('Duration Frequency', fontsize=15)
ax4.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```



feature : marital status

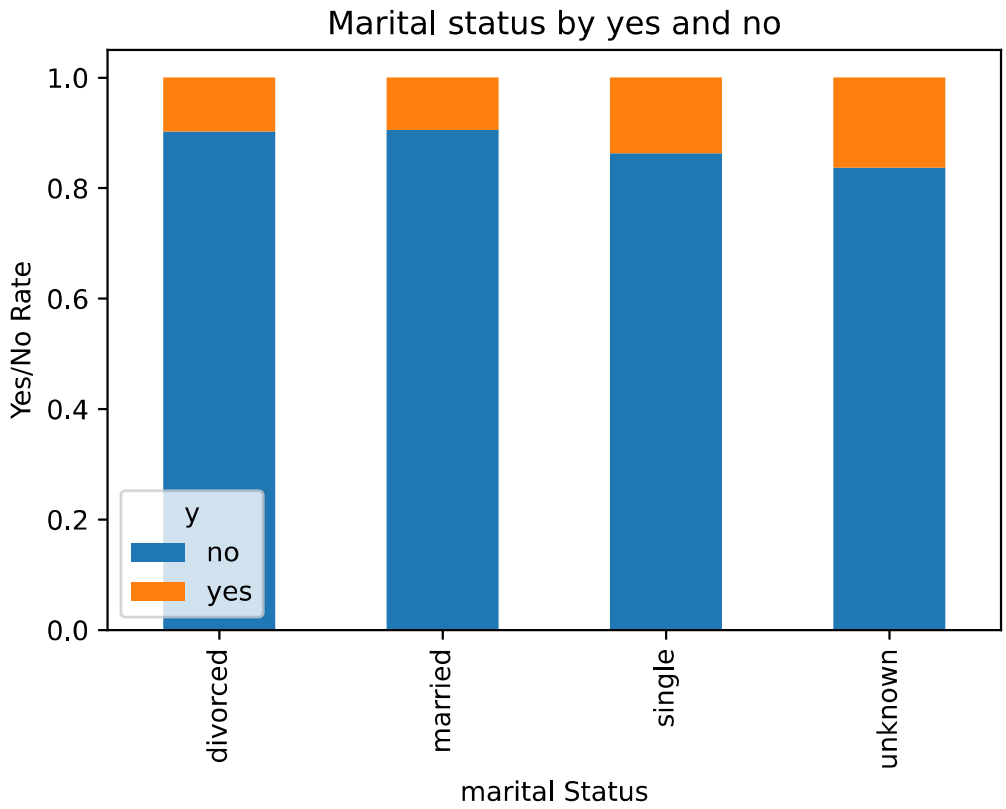
In [333]:
```
marital_s = pd.crosstab(df['marital'], df['y'])
marital_s
```

Out[333]:

y	no	yes
marital		
divorced	1983	215
married	11272	1179
single	4891	777
unknown	36	7

In [334]:
```
# Normalize the cross tab to sum to 1:
ms_t = marital_s.div(marital_s.sum(1).astype(float), axis=0)

ms_t.plot(kind='bar',
                  stacked=True,
                  title='Marital status by yes and no')
plt.xlabel('marital Status')
plt.ylabel('Yes/No Rate')
```

Out[334]: Text(0, 0.5, 'Yes/No Rate')



Target variable : y

In [335]:
```
plt.hist(df['y'])
```

Out[335]:
```
(array([18182.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0.,  2178.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <BarContainer object of 10 artists>)
```

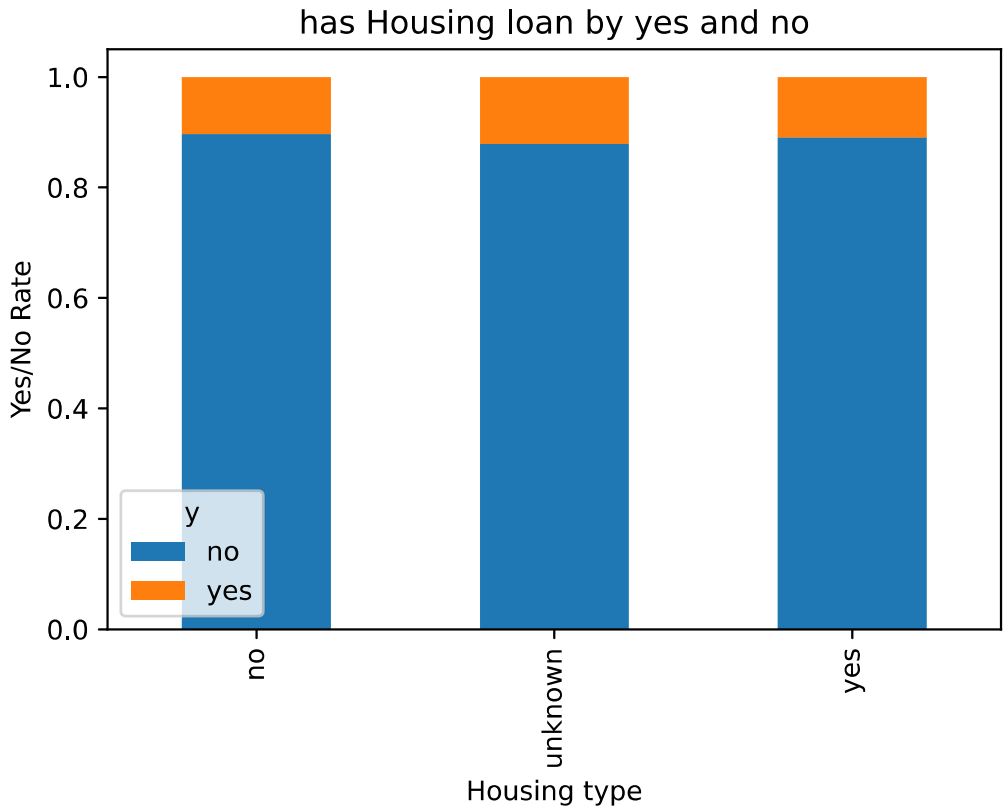
By the plot above we can see how imbalanced the data set is for training the model: so we definetely take this into account that our model is trained in highly imbalanced dataset which may be the business case

feature : Housing

In [336]:
```
## Its looks fairley evenly distributed in all categories then we can drop it
 before modelling
```

In [337]:
```
housing  = pd.crosstab(df['housing'], df['y'])
# Normalize the cross tab to sum to 1:
housing_t = housing.div(housing.sum(1).astype(float), axis=0)

housing_t.plot(kind='bar',
                   stacked=True,
                   title='has Housing loan by yes and no')
plt.xlabel('Housing type')
plt.ylabel('Yes/No Rate')
```

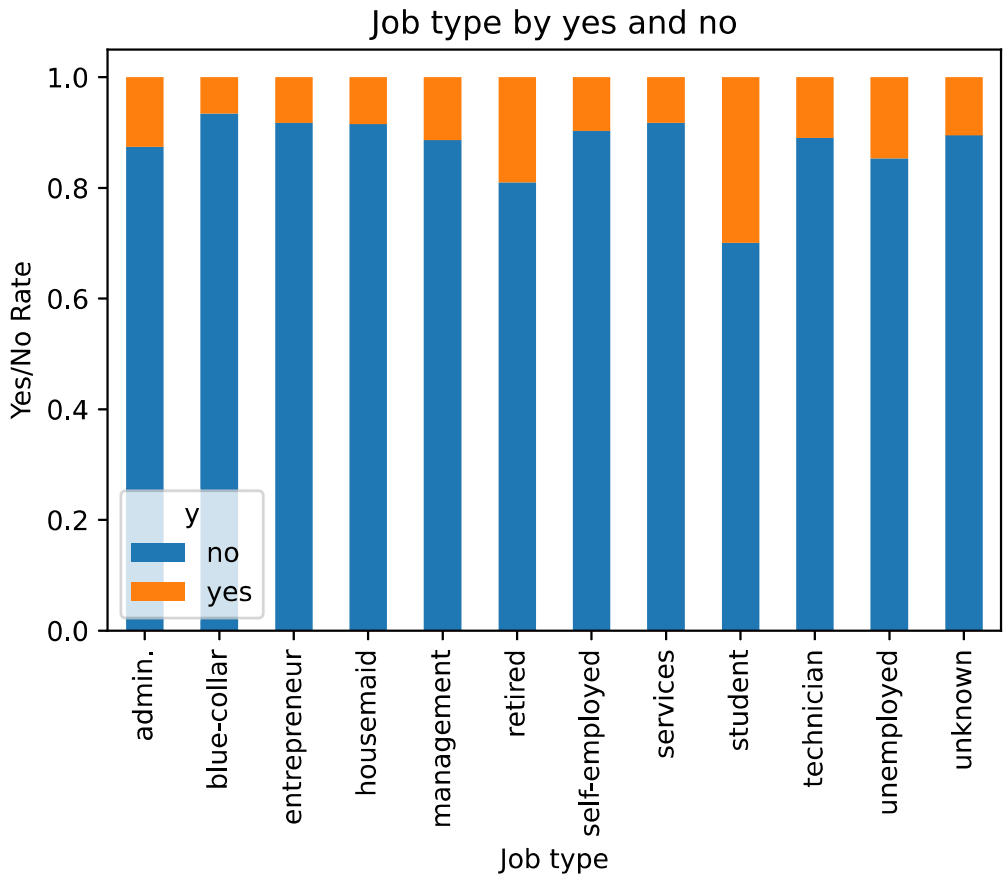
Out[337]: Text(0, 0.5, 'Yes/No Rate')



feature : Job

In [338]:
```
job = pd.crosstab(df['job'], df['y'])
# Normalize the cross tab to sum to 1:
job_t = job.div(job.sum(1).astype(float), axis=0)

job_t.plot(kind='bar',
                  stacked=True,
                  title='Job type by yes and no')
plt.xlabel('Job type')
plt.ylabel('Yes/No Rate')
```

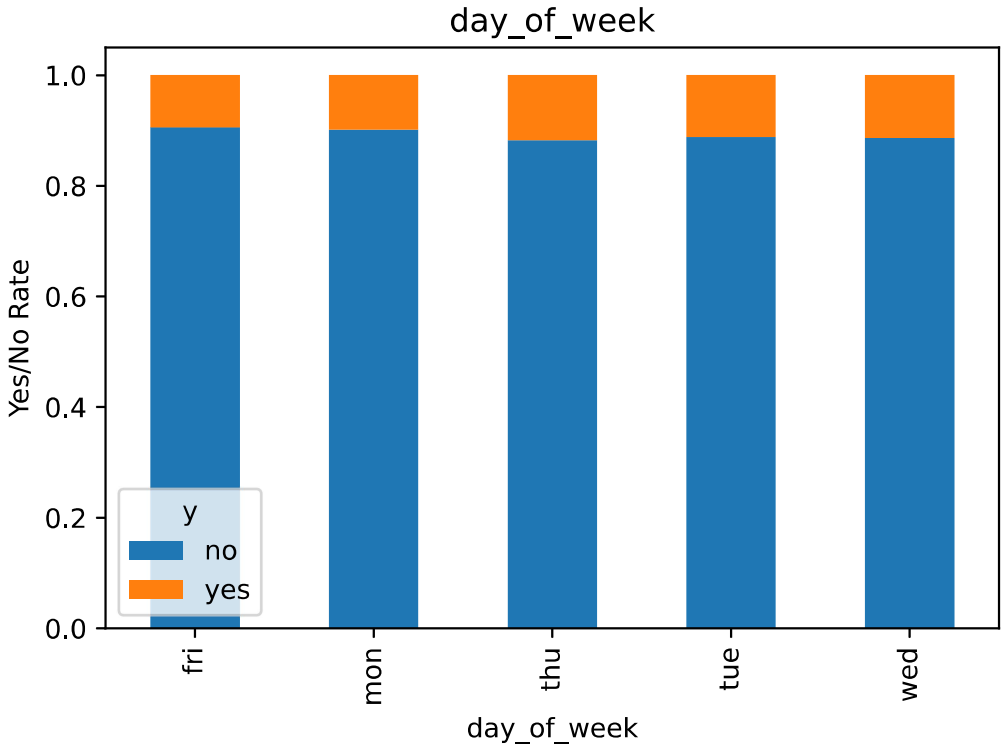
Out[338]: Text(0, 0.5, 'Yes/No Rate')



feature : day of week

In [339]:
```
dow = pd.crosstab(df['day_of_week'], df['y'])
# Normalize the cross tab to sum to 1:
dow_t = dow.div(dow.sum(1).astype(float), axis=0)

dow_t.plot(kind='bar',
                   stacked=True,
                   title='day_of_week')
plt.xlabel('day_of_week')
plt.ylabel('Yes/No Rate')

#we can drop this column as this is not adding much to the model
```

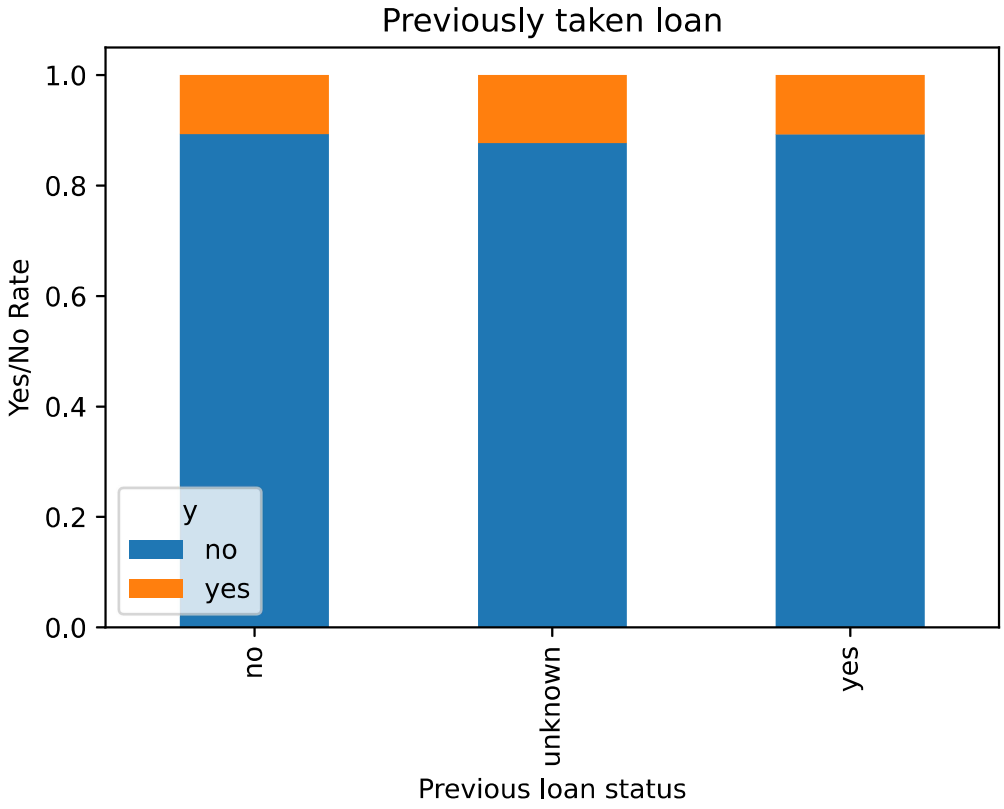
Out[339]: Text(0, 0.5, 'Yes/No Rate')



feature : Loan

In [340]:
```
loan = pd.crosstab(df['loan'], df['y'])
# Normalize the cross tab to sum to 1:
loan_t = loan.div(loan.sum(1).astype(float), axis=0)

loan_t.plot(kind='bar',
                   stacked=True,
                   title='Previously taken loan')
plt.xlabel('Previous loan status')
plt.ylabel('Yes/No Rate')

#we can drop this also as this is not generating enough to be added to model
```

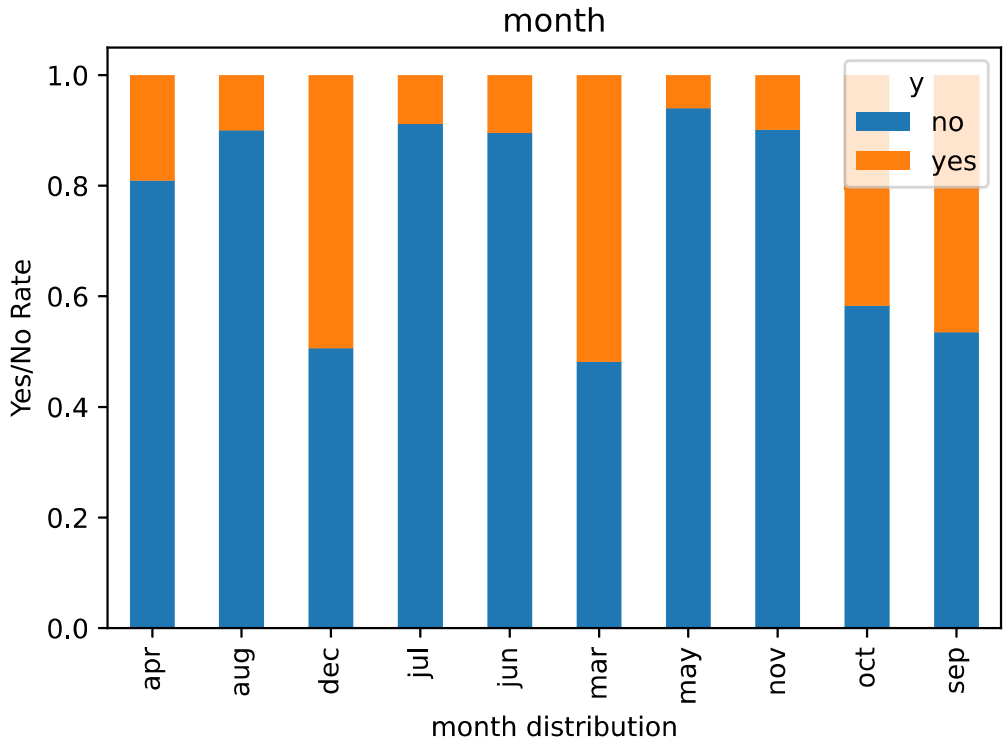
Out[340]: Text(0, 0.5, 'Yes/No Rate')



feature : Month

In [341]:
```
month = pd.crosstab(df['month'], df['y'])
# Normalize the cross tab to sum to 1:
month_t = month.div(month.sum(1).astype(float), axis=0)

month_t.plot(kind='bar',
                  stacked=True,
                  title='month')
plt.xlabel('month distribution')
plt.ylabel('Yes/No Rate')
```

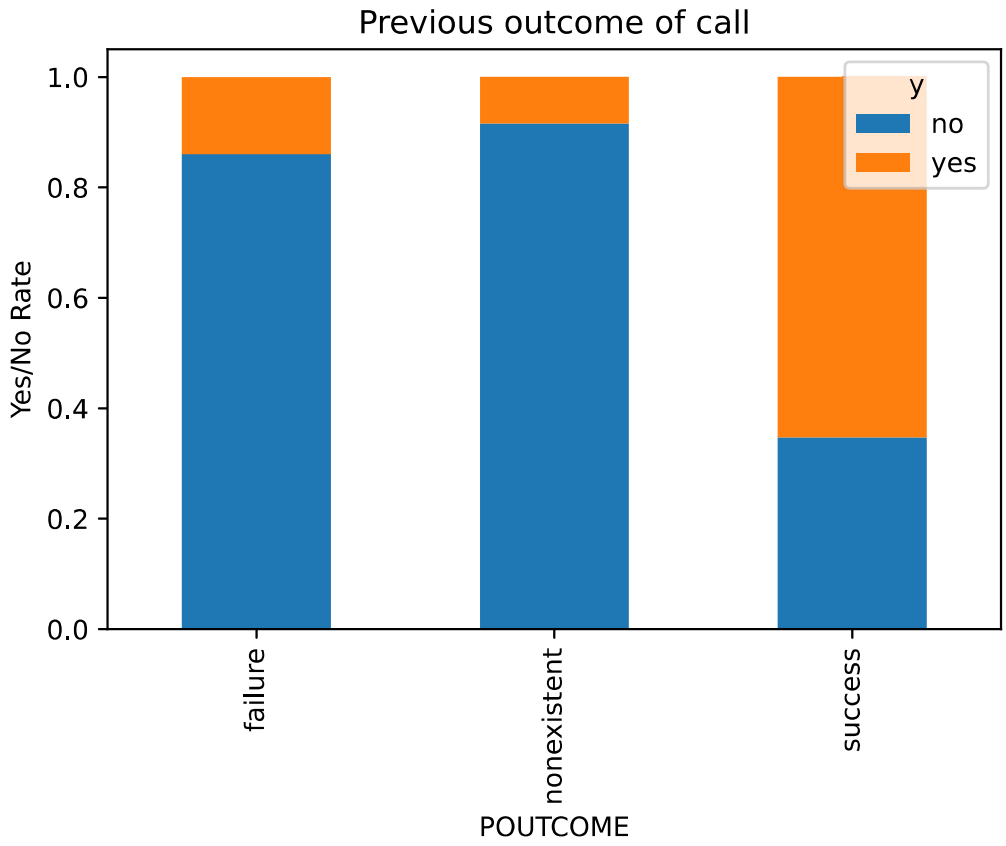
Out[341]: Text(0, 0.5, 'Yes/No Rate')



feature : poutcome

In [342]:
```
po = pd.crosstab(df['poutcome'], df['y'])
# Normalize the cross tab to sum to 1:
po_t = po.div(po.sum(1).astype(float), axis=0)

po_t.plot(kind='bar',
                  stacked=True,
                  title='Previous outcome of call')
plt.xlabel('POUTCOME')
plt.ylabel('Yes/No Rate')
```

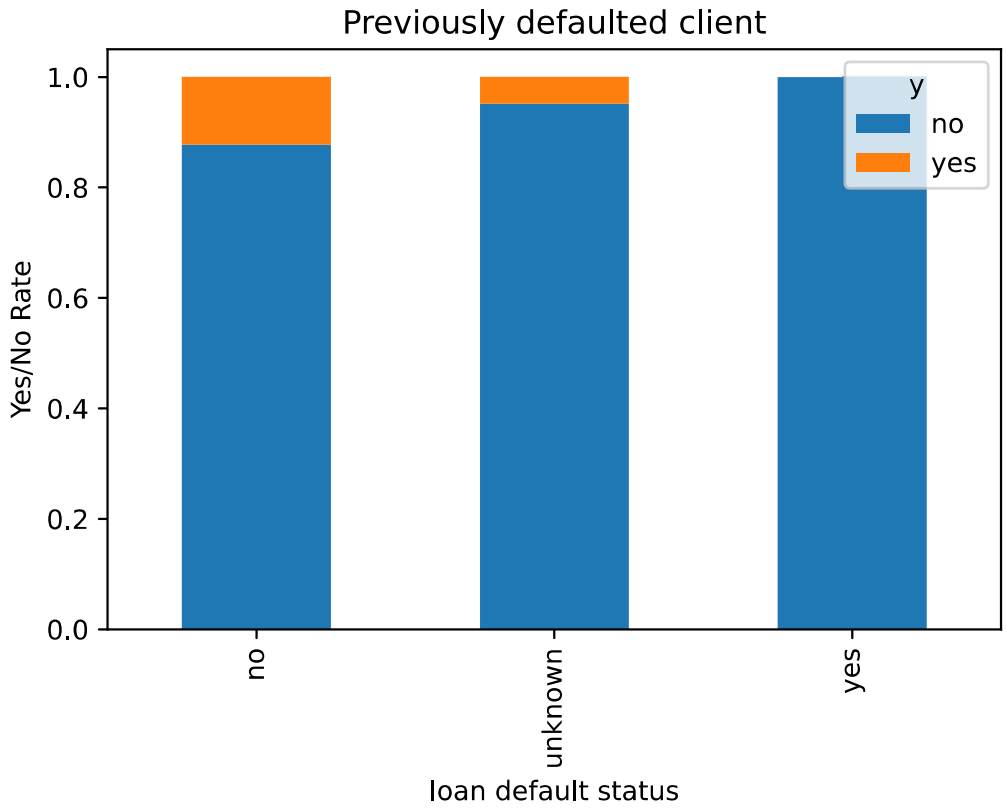
Out[342]: Text(0, 0.5, 'Yes/No Rate')



feature : default

In [343]:
```
default = pd.crosstab(df['default'], df['y'])
# Normalize the cross tab to sum to 1:
default_t = default.div(default.sum(1).astype(float), axis=0)

default_t.plot(kind='bar',
                   stacked=True,
                   title='Previously defaulted client')
plt.xlabel('loan default status')
plt.ylabel('Yes/No Rate')
```

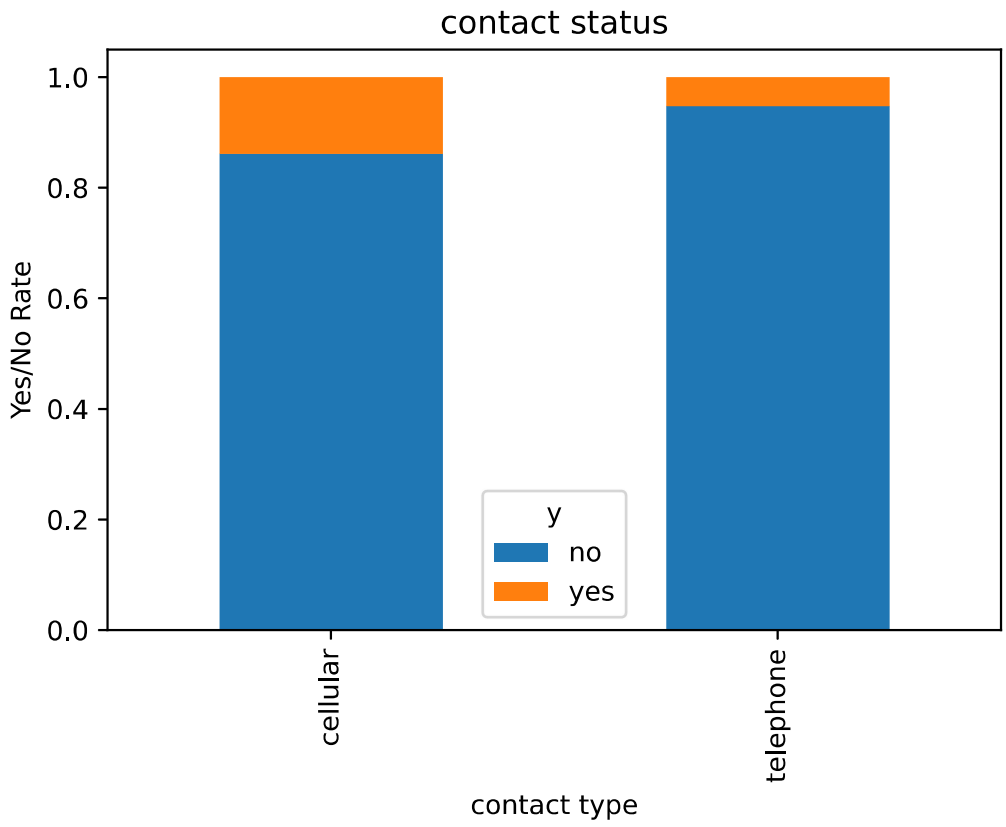
Out[343]: Text(0, 0.5, 'Yes/No Rate')



feature: contact

In [344]:
```
con = pd.crosstab(df['contact'], df['y'])
# Normalize the cross tab to sum to 1:
con_t = con.div(con.sum(1).astype(float), axis=0)

con_t.plot(kind='bar',
                   stacked=True,
                   title='contact status')
plt.xlabel('contact type')
plt.ylabel('Yes/No Rate')
```

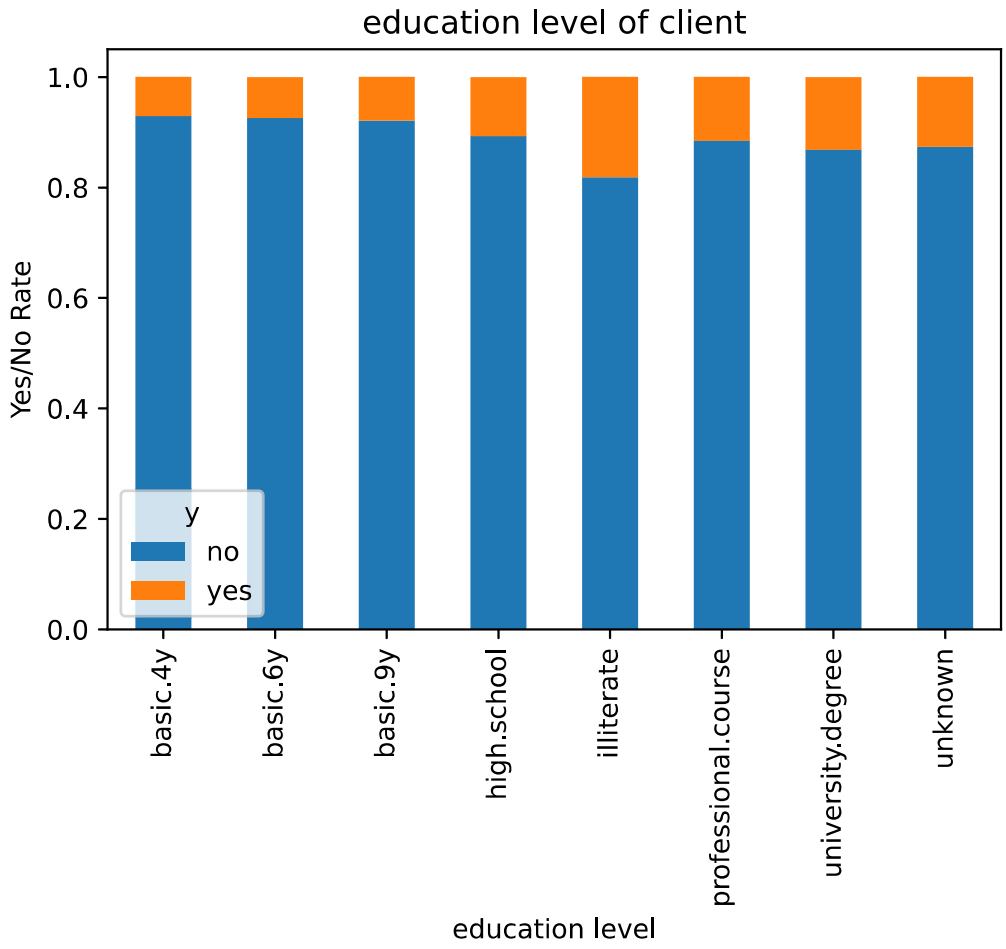
Out[344]: Text(0, 0.5, 'Yes/No Rate')



feature : education

In [345]:
```
edu = pd.crosstab(df['education'], df['y'])
# Normalize the cross tab to sum to 1:
edu_t = edu.div(edu.sum(1).astype(float), axis=0)

edu_t.plot(kind='bar',
                   stacked=True,
                   title='education level of client')
plt.xlabel('education level')
plt.ylabel('Yes/No Rate')
```

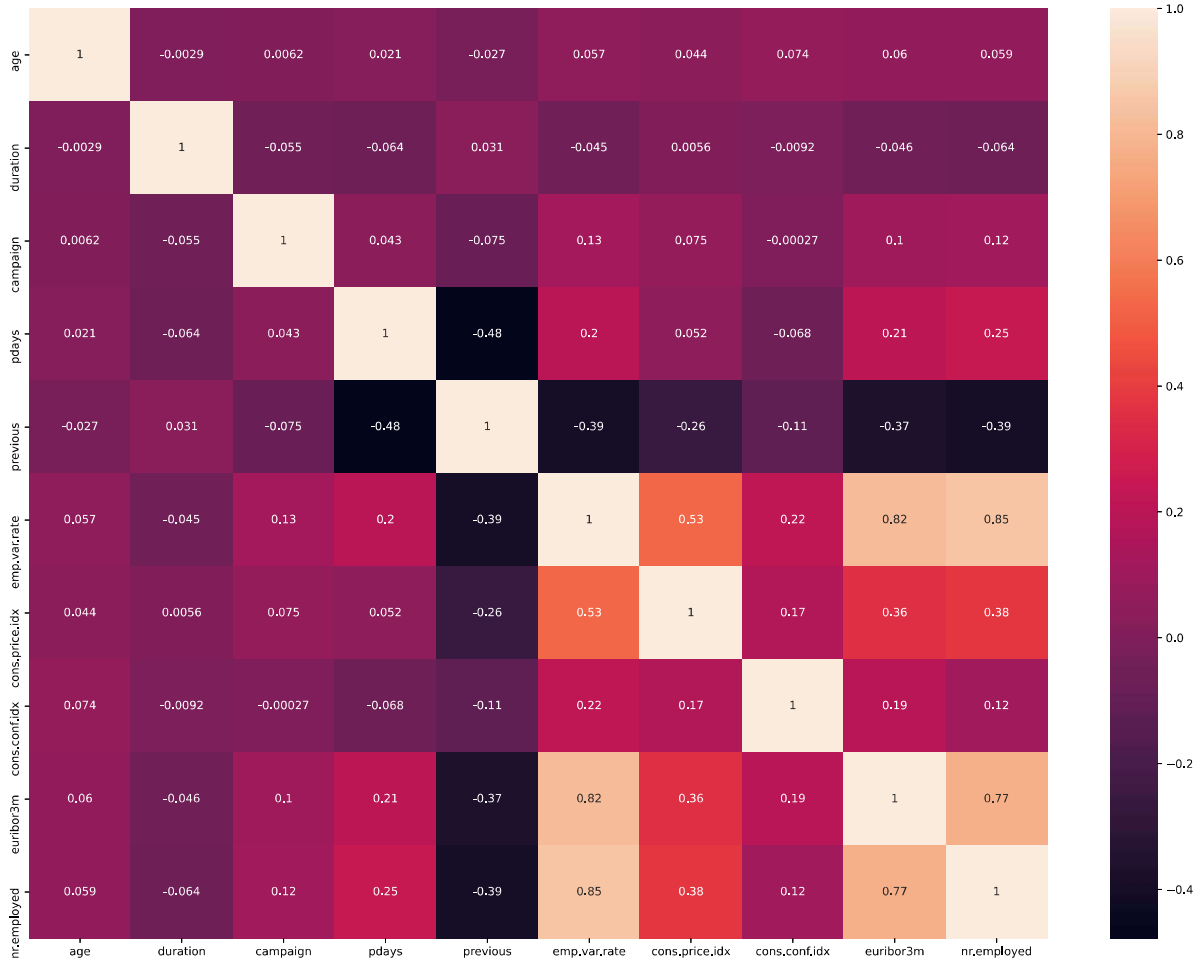
Out[345]: Text(0, 0.5, 'Yes/No Rate')



Corelation matrix : All numerical features

In [346]:
```
corr = df[["age","duration","campaign","pdays","previous","emp.var.rate","con
s.price.idx","cons.conf.idx","euribor3m","nr.employed"]].corr(method='kendall'
)
plt.figure(figsize=(20,15))
sns.heatmap(corr, annot=True)
df.columns
```

Out[346]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```



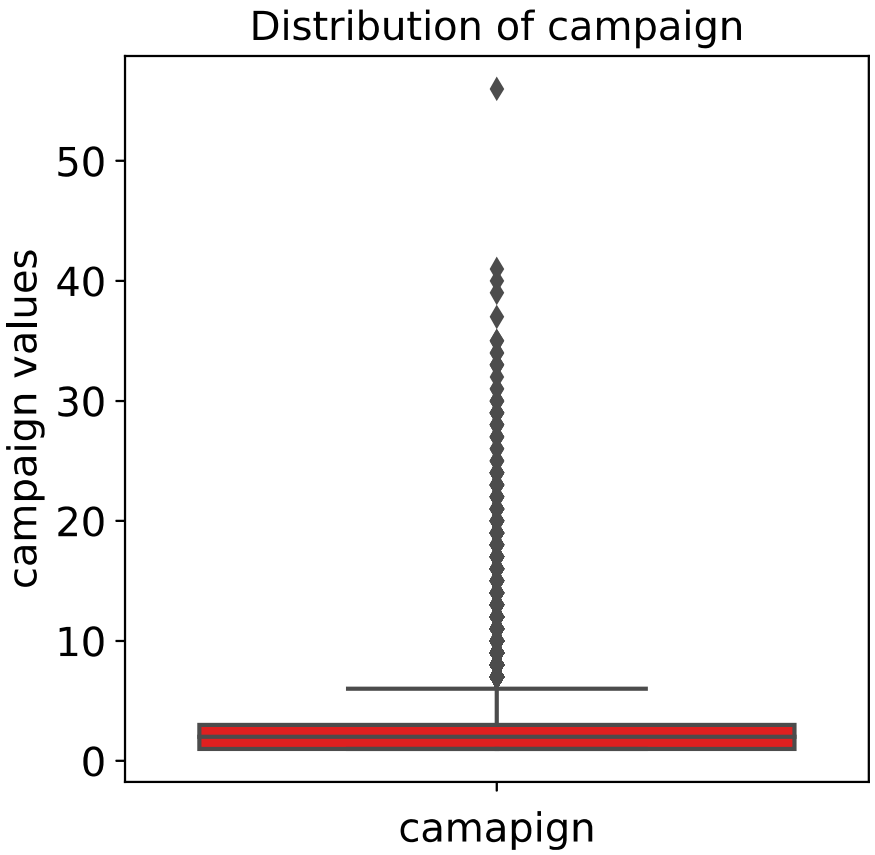
we can see some features are hihgliy positively correlated with each other like nr.employed and emp.var.rate and

feature:campaign

In [347]:
```
df['campaign'].describe()
```

Out[347]:
```
count    20360.000000
mean         2.588949
std          2.858372
min          1.000000
25%          1.000000
50%          2.000000
75%          3.000000
max         56.000000
Name: campaign, dtype: float64
```

In [348]:
```
### we can see above that there are some outliers in the campaign field
fig, (ax1) = plt.subplots(nrows = 1, ncols = 1, figsize = (5, 5))
sns.boxplot(x = 'campaign', data = df, orient = 'v', ax = ax1,color='red')
ax1.set_xlabel('camapign', fontsize=15)
ax1.set_ylabel('campaign values', fontsize=15)
ax1.set_title('Distribution of campaign', fontsize=15)
ax1.tick_params(labelsize=15)
```



feature : pdays

In [349]:
```
df['pdays'].describe()
```

Out[349]:
```
count    20360.000000
mean       965.991601
std        177.987746
min          0.000000
25%        999.000000
50%        999.000000
75%        999.000000
max        999.000000
Name: pdays, dtype: float64
```

In [350]:
```
df['previous'].describe()
```

Out[350]:
```
count    20360.000000
mean         0.168222
std          0.484387
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          6.000000
Name: previous, dtype: float64
```

In [351]:
```
df['emp.var.rate'].describe()
```

Out[351]:
```
count    20360.000000
mean         0.109479
std          1.554051
min         -3.400000
25%         -1.800000
50%          1.100000
75%          1.400000
max          1.400000
Name: emp.var.rate, dtype: float64
```

In [352]:
```
df['cons.price.idx'].describe()
```

Out[352]:
```
count    20360.000000
mean        93.579891
std          0.574459
min         92.201000
25%         93.075000
50%         93.749000
75%         93.994000
max         94.767000
Name: cons.price.idx, dtype: float64
```

In [353]:
```
df['cons.conf.idx'].describe()
```

Out[353]:
```
count    20360.000000
mean       -40.557642
std          4.563615
min        -50.800000
25%        -42.700000
50%        -41.800000
75%        -36.400000
max        -26.900000
Name: cons.conf.idx, dtype: float64
```

In [354]:
```
df['euribor3m'].describe()
```

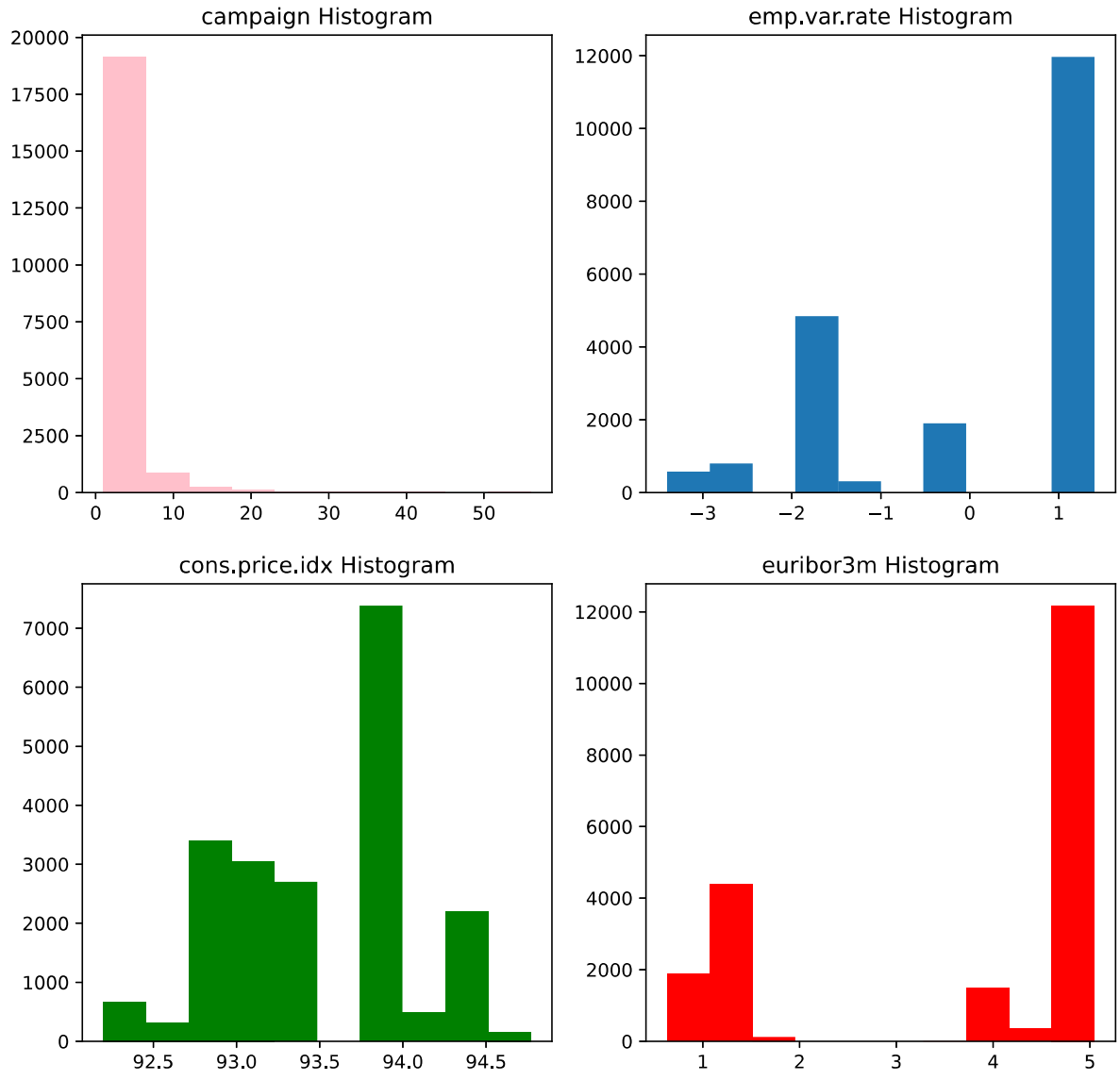
Out[354]:
```
count    20360.000000
mean         3.678060
std          1.708043
min          0.634000
25%          1.365000
50%          4.857000
75%          4.961000
max          5.045000
Name: euribor3m, dtype: float64
```

In [355]:
```
df['nr.employed'].describe()
```

Out[355]:
```
count    20360.000000
mean      5168.727417
std         70.908803
min       4963.600000
25%       5099.100000
50%       5191.000000
75%       5228.100000
max       5228.100000
Name: nr.employed, dtype: float64
```

In [356]:
```
# Set up a grid of plots
fig = plt.figure(figsize=(10,10))
fig_dims = (2, 2)

#plt.xticks(rotation=0)

# Plot the Price histogram
plt.subplot2grid(fig_dims, (0, 0))
df['campaign'].hist(grid = False,color='pink')
plt.title('campaign Histogram')

# Plot sqaure feet histogram
plt.subplot2grid(fig_dims, (0, 1))
df['emp.var.rate'].hist(grid = False)
plt.title('emp.var.rate Histogram')

# Plot $/square feet histogram
plt.subplot2grid(fig_dims, (1, 0))
df['cons.price.idx'].hist(grid = False,color='green')
plt.title('cons.price.idx Histogram')

# Plot days on market histogram
plt.subplot2grid(fig_dims, (1, 1))
df['euribor3m'].hist(grid = False,color= 'Red')
plt.title('euribor3m Histogram')
```

Out[356]: Text(0.5, 1.0, 'euribor3m Histogram')



final data preparation for MODELS

there are 9 categorical columns and we have process them accordingly for coverting them to numerical values using One hot Encoding or direct Mapping

based on analysis of data performed previously we will drop some columns

In [357]:
```
# Dropping duration for the modelling purpose as duration is knows after the c
all only
# we can drop emp.var.rate as it is highly correlated with nr.employeed
# Housing , day_of_week and loan attributes can be removed

# First dropping columns from our main data set
ls=['day_of_week','housing','loan','emp.var.rate']
for j in ls:
    df.drop(j,inplace=True,axis=1)
```

In [358]:
```
# let check the unique values in each categorical column
cat_col=['job', 'marital', 'education', 'default',
       'contact', 'month','poutcome']
for r in cat_col:
    print("type of %s -->"%r,df[r].unique(),'\n')
```

```
type of job --> ['blue-collar' 'admin.' 'technician' 'retired' 'management'
'services'
 'housemaid' 'unknown' 'student' 'self-employed' 'entrepreneur'
 'unemployed']

type of marital --> ['married' 'divorced' 'single' 'unknown']

type of education --> ['basic.9y' 'high.school' 'university.degree' 'professi
onal.course'
 'unknown' 'basic.6y' 'basic.4y' 'illiterate']

type of default --> ['no' 'unknown' 'yes']

type of contact --> ['telephone' 'cellular']

type of month --> ['may' 'aug' 'jul' 'mar' 'nov' 'apr' 'jun' 'oct' 'dec' 'se
p']

type of poutcome --> ['nonexistent' 'failure' 'success']
```

JOB

In [359]:
```
df['job'].value_counts()
```

Out[359]:
```
admin.           5496
blue-collar      4574
technician       3317
services         1921
management       1460
entrepreneur      704
self-employed     702
retired           604
unemployed        511
housemaid         508
student           401
unknown           162
Name: job, dtype: int64
```

In [360]:
```
#using one hot enconding with Job as they are not related to each other
cols = pd.get_dummies(df['job'], prefix= 'job')
df[cols.columns] = cols
df.drop('job', axis = 1, inplace = True)
```

MARITAL STATUS

In [361]:
```
df['marital'].value_counts()
```

Out[361]:
```
married     12451
single       5668
divorced     2198
unknown        43
Name: marital, dtype: int64
```

In [362]:
```
#using one hot enconding with marital status as they are not related to each o
ther
cols = pd.get_dummies(df['marital'], prefix= 'MS')
df[cols.columns] = cols
df.drop('marital', axis = 1, inplace = True)
```

Education level

In [363]:
```
df['education'].value_counts()
```

Out[363]:
```
university.degree      6055
high.school            4708
basic.9y               2982
professional.course    2677
basic.4y               1906
basic.6y               1161
unknown                 860
illiterate               11
Name: education, dtype: int64
```

In [364]:
```
# we can map them directly as they are related to each other in levels
df['education'] = df['education'].map({'basic.9y':4 ,'high.school':5 ,'univers
ity.degree':7, 'professional.course':6,
 'unknown':0 ,'basic.6y':3 ,'basic.4y':2 ,'illiterate':1}).astype(int)
```

DEFAULT

In [365]:
```
df['default'].value_counts()
```

Out[365]:
```
no         16097
unknown     4260
yes            3
Name: default, dtype: int64
```

In [366]:
```
#using one hot enconding with deault laon column as they are not related to ea
ch other
cols = pd.get_dummies(df['default'], prefix= 'df')
df[cols.columns] = cols
df.drop('default', axis = 1, inplace = True)
```

type of contact information available

In [367]:
```
df['contact'].value_counts()
```

Out[367]:
```
cellular     12881
telephone     7479
Name: contact, dtype: int64
```

In [368]:
```
# we can map them directly as they are related to each other in levels
df['contact'] = df['contact'].map({'telephone':0 ,'cellular':1}).astype(int)
```

MONTH COLUMN

In [369]:
```
df['month'].value_counts()
```

Out[369]:
```
may    6854
jul    3545
aug    3032
jun    2661
nov    2090
apr    1267
oct     312
sep     271
mar     245
dec      83
Name: month, dtype: int64
```

In [370]:
```
#using one hot enconding with month column as they are not related to each oth
er
cols = pd.get_dummies(df['month'], prefix= 'month')
df[cols.columns] = cols
df.drop('month', axis = 1, inplace = True)
```

previous outcome

In [371]:
```
df['poutcome'].value_counts()
```

Out[371]:
```
nonexistent    17623
failure         2132
success          605
Name: poutcome, dtype: int64
```

In [372]:
```
#using one hot enconding with poutcome column as they are not related to each
 other
cols = pd.get_dummies(df['poutcome'], prefix= 'po')
df[cols.columns] = cols
df.drop('poutcome', axis = 1, inplace = True)
```

In [373]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20360 entries, 0 to 20593
Data columns (total 44 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   age                 20360 non-null  int64  
 1   education           20360 non-null  int32  
 2   contact             20360 non-null  int32  
 3   duration            20360 non-null  int64  
 4   campaign            20360 non-null  int64  
 5   pdays               20360 non-null  int64  
 6   previous            20360 non-null  int64  
 7   cons.price.idx      20360 non-null  float64
 8   cons.conf.idx       20360 non-null  float64
 9   euribor3m           20360 non-null  float64
 10  nr.employed         20360 non-null  float64
 11  y                   20360 non-null  object 
 12  job_admin.          20360 non-null  uint8  
 13  job_blue-collar     20360 non-null  uint8  
 14  job_entrepreneur    20360 non-null  uint8  
 15  job_housemaid       20360 non-null  uint8  
 16  job_management      20360 non-null  uint8  
 17  job_retired         20360 non-null  uint8  
 18  job_self-employed   20360 non-null  uint8  
 19  job_services        20360 non-null  uint8  
 20  job_student         20360 non-null  uint8  
 21  job_technician      20360 non-null  uint8  
 22  job_unemployed      20360 non-null  uint8  
 23  job_unknown         20360 non-null  uint8  
 24  MS_divorced         20360 non-null  uint8  
 25  MS_married          20360 non-null  uint8  
 26  MS_single           20360 non-null  uint8  
 27  MS_unknown          20360 non-null  uint8  
 28  df_no               20360 non-null  uint8  
 29  df_unknown          20360 non-null  uint8  
 30  df_yes              20360 non-null  uint8  
 31  month_apr           20360 non-null  uint8  
 32  month_aug           20360 non-null  uint8  
 33  month_dec           20360 non-null  uint8  
 34  month_jul           20360 non-null  uint8  
 35  month_jun           20360 non-null  uint8  
 36  month_mar           20360 non-null  uint8  
 37  month_may           20360 non-null  uint8  
 38  month_nov           20360 non-null  uint8  
 39  month_oct           20360 non-null  uint8  
 40  month_sep           20360 non-null  uint8  
 41  po_failure          20360 non-null  uint8  
 42  po_nonexistent      20360 non-null  uint8  
 43  po_success          20360 non-null  uint8  
dtypes: float64(4), int32(2), int64(5), object(1), uint8(32)
memory usage: 3.1+ MB
```

TARGET VARIABLE

In [374]:
```
df['y'].value_counts()
```

Out[374]:
```
no     18182
yes     2178
Name: y, dtype: int64
```

In [375]:
```
# we can map them directly as they are related to each other in levels
df['y'] = df['y'].map({'no':0 ,'yes':1}).astype(int)
```

In [376]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20360 entries, 0 to 20593
Data columns (total 44 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   age                 20360 non-null  int64  
 1   education           20360 non-null  int32  
 2   contact             20360 non-null  int32  
 3   duration            20360 non-null  int64  
 4   campaign            20360 non-null  int64  
 5   pdays               20360 non-null  int64  
 6   previous            20360 non-null  int64  
 7   cons.price.idx      20360 non-null  float64
 8   cons.conf.idx       20360 non-null  float64
 9   euribor3m           20360 non-null  float64
 10  nr.employed         20360 non-null  float64
 11  y                   20360 non-null  int32  
 12  job_admin.          20360 non-null  uint8  
 13  job_blue-collar     20360 non-null  uint8  
 14  job_entrepreneur    20360 non-null  uint8  
 15  job_housemaid       20360 non-null  uint8  
 16  job_management      20360 non-null  uint8  
 17  job_retired         20360 non-null  uint8  
 18  job_self-employed   20360 non-null  uint8  
 19  job_services        20360 non-null  uint8  
 20  job_student         20360 non-null  uint8  
 21  job_technician      20360 non-null  uint8  
 22  job_unemployed      20360 non-null  uint8  
 23  job_unknown         20360 non-null  uint8  
 24  MS_divorced         20360 non-null  uint8  
 25  MS_married          20360 non-null  uint8  
 26  MS_single           20360 non-null  uint8  
 27  MS_unknown          20360 non-null  uint8  
 28  df_no               20360 non-null  uint8  
 29  df_unknown          20360 non-null  uint8  
 30  df_yes              20360 non-null  uint8  
 31  month_apr           20360 non-null  uint8  
 32  month_aug           20360 non-null  uint8  
 33  month_dec           20360 non-null  uint8  
 34  month_jul           20360 non-null  uint8  
 35  month_jun           20360 non-null  uint8  
 36  month_mar           20360 non-null  uint8  
 37  month_may           20360 non-null  uint8  
 38  month_nov           20360 non-null  uint8  
 39  month_oct           20360 non-null  uint8  
 40  month_sep           20360 non-null  uint8  
 41  po_failure          20360 non-null  uint8  
 42  po_nonexistent      20360 non-null  uint8  
 43  po_success          20360 non-null  uint8  
dtypes: float64(4), int32(3), int64(5), uint8(32)
memory usage: 3.0 MB
```

MODELLING

y is the target variable

In [377]:
```
X=df.drop(['y'],axis=1)
y=df['y']
```

In [378]:
```
X.shape
```

Out[378]: (20360, 43)

In [379]:
```
y.shape
```

Out[379]: (20360,)

SPLITING THE DATA

In [380]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score


X_train_1, X_test_1, y_train, y_test = train_test_split(X, y, random_state = 0
, test_size = 0.3)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train_1)
X_test = scaler.transform(X_test_1)

print("Training dataset size: "+str(len(X_train)))
print("Test dataset size: "+str(len(X_test)))
```

```
Training dataset size: 14252
Test dataset size: 6108
```

Models

Importing modeules from sklearn

In [121]:
```
import sklearn

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

In [122]:
```
# Building and evaluating classification models basic models
model_list =[]
models_list.append(('LogR', LogisticRegression()))
models_list.append(('KNN', KNeighborsClassifier()))
models_list.append(('L_SVC', LinearSVC()))
models_list.append(('SVM_linear', SVC(kernel='linear')))
models_list.append(('SVM_rbf', SVC(kernel='rbf')))
models_list.append(('SVM_poly', SVC(kernel='poly')))
models_list.append(('DTC', DecisionTreeClassifier()))

r_array=[]
n_array=[]
for name_str, model_str in models_list:
    K = KFold(n_splits=8, random_state=7)
    crv = model_selection.cross_val_score(model_str, X_train, y_train, cv=kfol
d, scoring = 'accuracy')
    r_array.append(crv)
    n_array.append(name_str)
    msg = "%s: %f (%f)" % (name_str, crv.mean(), crv.std())
    print(msg)
```

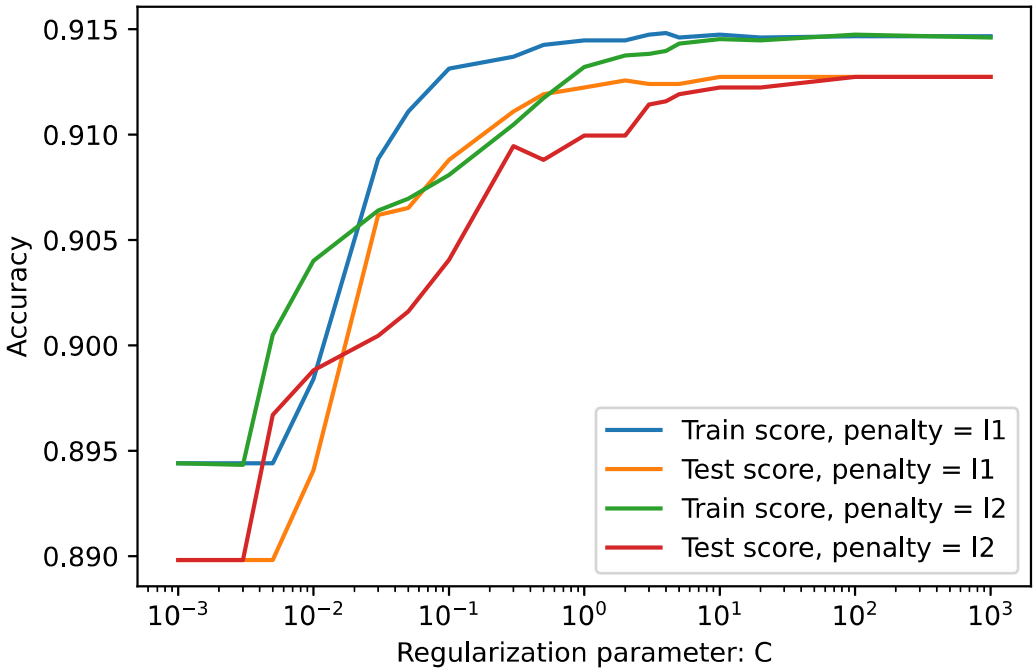
```
LogR: 0.911731 (0.006862)
KNN: 0.899031 (0.008032)
L_SVC: 0.911941 (0.007660)
SVM_linear: 0.904504 (0.008549)
SVM_rbf: 0.905486 (0.006932)
SVM_poly: 0.906188 (0.006940)
DTC: 0.893067 (0.008270)
```

Logistic Regression

In [82]:
```
logicR = LogisticRegression()
logicR.fit(X_train, y_train)
pred = logicR.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, pred)))
print("AUC: "+ str(roc_auc_score(y_test, logicR.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, pred)))
print(classification_report(y_test, pred))
```

```
Accuracy: 0.9099541584806811
AUC: 0.9276895254056109
Confusion Matrix: [[5327  108]
 [ 442  231]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      5435
           1       0.68      0.34      0.46       673

    accuracy                           0.91      6108
   macro avg       0.80      0.66      0.70      6108
weighted avg       0.90      0.91      0.90      6108
```

In [123]:
```
import matplotlib.pyplot as plt
%matplotlib inline

Range = [0.001,0.003,0.005,0.01,0.03,0.05,0.1,0.3,0.5,1,2,3,3,4,5,10,20,100,10
00]
trs_list1 = []
trs_list2 = []
ts_list1 = []
ts_list2 = []

for d in Range:
    logicR1 = LogisticRegression(penalty = 'l1', C = d,solver='liblinear')
    logicR2 = LogisticRegression(penalty = 'l2', C = d,solver='lbfgs')
    logicR1.fit(X_train, y_train)
    logicR2.fit(X_train, y_train)
    trs_list1.append(logicR1.score(X_train, y_train))
    trs_list2.append(logicR2.score(X_train, y_train))
    ts_list1.append(logicR1.score(X_test, y_test))
    ts_list2.append(logicR2.score(X_test, y_test))


plt.plot(Range, trs_list1, label = 'Train score, penalty = l1')
plt.plot(Range, ts_list1, label = 'Test score, penalty = l1')
plt.plot(Range, trs_list2, label = 'Train score, penalty = l2')
plt.plot(Range, ts_list2, label = 'Test score, penalty = l2')
plt.legend()
plt.xlabel('Regularization parameter: C')
plt.ylabel('Accuracy')
plt.xscale('log')
```



In [84]:
```
# applying gridsearch to find the best parameter
```

In [124]:
```
logicR = LogisticRegression(class_weight='balanced')
p1 = {'C':[0.001,0.003,0.005,0.01,0.03,0.05,0.1,0.3,0.5,1,2,3,3,4,5,10,20,100,
1000]}
clss_1 = GridSearchCV(logicR,p1,scoring='roc_auc',refit=True,cv=6)
clss_1.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss_1.best_score_, clss_1
.best_params_))
```

```
Best roc_auc: 0.9356, with best C: {'C': 1000}
```

In [86]:
```
predict_y = clss_1.predict(X_test)

print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.8652586771447283
[[4706  729]
 [  94  579]]
              precision    recall  f1-score   support

           0       0.98      0.87      0.92      5435
           1       0.44      0.86      0.58       673

    accuracy                           0.87      6108
   macro avg       0.71      0.86      0.75      6108
weighted avg       0.92      0.87      0.88      6108
```

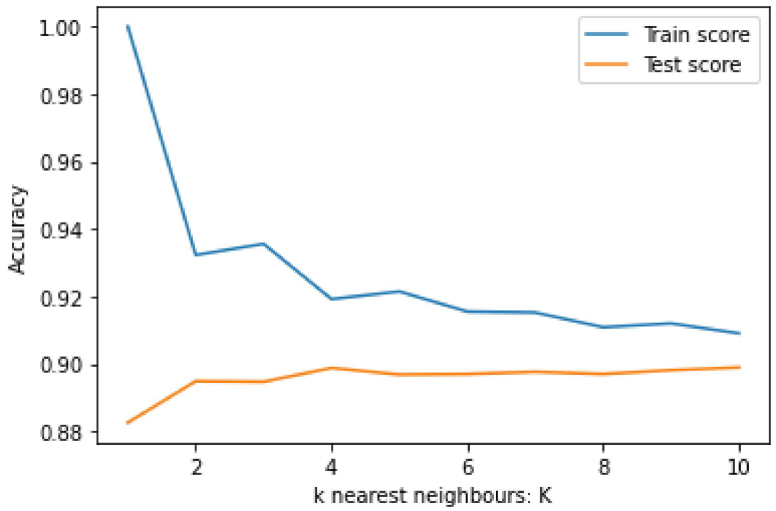
KNN classifier

In [87]:
```
Knn_d = KNeighborsClassifier()
Knn_d.fit(X_train, y_train)
predict_y = Knn_d.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predict_y)))
print("AUC: "+ str(roc_auc_score(y_test, Knn_d.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_y)))
print(classification_report(y_test, predict_y))
```

```
Accuracy: 0.8968565815324165
AUC: 0.789564910717093
Confusion Matrix: [[5300  135]
 [ 495  178]]
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      5435
           1       0.57      0.26      0.36       673

    accuracy                           0.90      6108
   macro avg       0.74      0.62      0.65      6108
weighted avg       0.88      0.90      0.88      6108
```

In [88]:
```
r = list(range(1, 11))
trs_knn_list = []
ts_knn_list = []

for k in r:
    md = KNeighborsClassifier(n_neighbors=k)
    md.fit(X_train, y_train)
    trs_knn_list.append(md.score(X_train, y_train))
    ts_knn_list.append(md.score(X_test, y_test))

plt.plot(k_range, trs_knn_list, label = 'Train score')
plt.plot(k_range, ts_knn_list, label = 'Test score')
plt.legend()
plt.xlabel('k nearest neighbours: K')
plt.ylabel('Accuracy')
```

Out[88]: Text(0, 0.5, 'Accuracy')



In [89]:
```
r = list(range(1, 11))
Knn_c = KNeighborsClassifier()
p = dict( n_neighbors = r)
clss_2 = GridSearchCV(Knn_c, p,'roc_auc',refit=True,cv=3)
clss_2.fit(X_train,y_train)
print('roc_auc: {:.4}, with best C: {}'.format(clss_2.best_score_, clss_2.best
_params_))
```

```
roc_auc: 0.7952, with best C: {'n_neighbors': 10}
```

In [90]:
```
predict_y = clss_2.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.8989849377865095
[[5370   65]
 [ 552  121]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.65      0.18      0.28       673

    accuracy                           0.90      6108
   macro avg       0.78      0.58      0.61      6108
weighted avg       0.88      0.90      0.87      6108
```

Linear SVM

In [91]:
```
lsmv = LinearSVC()
lsmv.fit(X_train, y_train)
predict_y_test = lsmv.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predict_y_test)))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_y_test)))
print(classification_report(y_test, predict_y_test))
```

```
Accuracy: 0.9076620825147348
Confusion Matrix: [[5326  109]
 [ 455  218]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      5435
           1       0.67      0.32      0.44       673

    accuracy                           0.91      6108
   macro avg       0.79      0.65      0.69      6108
weighted avg       0.89      0.91      0.89      6108
```

In [92]:
```
p = {"C": [0.01, 0.1, 1.0]}
lsmv = LinearSVC()
clss = GridSearchCV(lsmv, p,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.9318, with best C: {'C': 1.0}
```

In [93]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.9076620825147348
[[5326  109]
 [ 455  218]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      5435
           1       0.67      0.32      0.44       673

    accuracy                           0.91      6108
   macro avg       0.79      0.65      0.69      6108
weighted avg       0.89      0.91      0.89      6108
```

SVM with Linear Kernel

In [94]:
```
Svc_l = SVC(kernel='linear', probability=True)

Svc_l.fit(X_train, y_train)
predict_y_test = Svc_l.predict(X_test)

print("Accuracy: "+ str(accuracy_score(y_test, predict_y_test)))
print("AUC: "+ str(roc_auc_score(y_test, Svc_l.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_y_test)))
print(classification_report(y_test, predict_y_test))
```

```
Accuracy: 0.9022593320235757
AUC: 0.9290070001954751
Confusion Matrix: [[5366   69]
 [ 528  145]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.68      0.22      0.33       673

    accuracy                           0.90      6108
   macro avg       0.79      0.60      0.64      6108
weighted avg       0.88      0.90      0.88      6108
```

In [95]:
```
p3 = {'C':[0.1, 1, 10],'gamma':[0.1, 1, 10], 'kernel':['linear']}
gs_svc = SVC()
clss = GridSearchCV(gs_svc, p3,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.9305, with best C: {'C': 10, 'gamma': 0.1, 'kernel': 'linea
r'}
```

In [96]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.9034053700065487
[[5345   90]
 [ 500  173]]
              precision    recall  f1-score   support

           0       0.91      0.98      0.95      5435
           1       0.66      0.26      0.37       673

    accuracy                           0.90      6108
   macro avg       0.79      0.62      0.66      6108
weighted avg       0.89      0.90      0.88      6108
```

SVM with RBF kernel

In [97]:
```
rbf = SVC(kernel='rbf', probability=True)
rbf.fit(X_train, y_train)
predict_y_test = rbf.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predict_y_test)))
print("AUC: "+ str(roc_auc_score(y_test, rbf.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_y_test)))
print(classification_report(y_test, predict_y_test))
```

```
Accuracy: 0.9017681728880157
AUC: 0.9225568688990925
Confusion Matrix: [[5393   42]
 [ 558  115]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.73      0.17      0.28       673

    accuracy                           0.90      6108
   macro avg       0.82      0.58      0.61      6108
weighted avg       0.89      0.90      0.87      6108
```

In [98]:
```
p_rbf = {'C':[0.1, 1, 10],'gamma':[0.1, 1, 10], 'kernel':['rbf']}
rbf = SVC()
clss_rb = GridSearchCV(rbf, p_rbf,'roc_auc',refit=True,cv=3)
clss_rb.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss_rb.best_score_, clss_
rb.best_params_))
```

```
Best roc_auc: 0.9165, with best C: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

In [99]:
```
predict_y = clss_rb.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.9004584151931893
[[5391   44]
 [ 564  109]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.71      0.16      0.26       673

    accuracy                           0.90      6108
   macro avg       0.81      0.58      0.61      6108
weighted avg       0.88      0.90      0.87      6108
```

SVM with Poly Kernel

In [100]:
```
poly = SVC(kernel='poly', probability=True)
poly.fit(X_train, y_train)

predict_y = poly.predict(X_test)


print("Accuracy: "+ str(accuracy_score(y_test, predict_y)))
print("AUC: "+ str(roc_auc_score(y_test, poly.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_y)))
print(classification_report(y_test, predict_y))
```

```
Accuracy: 0.9029142108709889
AUC: 0.9229699638166033
Confusion Matrix: [[5360   75]
 [ 518  155]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.67      0.23      0.34       673

    accuracy                           0.90      6108
   macro avg       0.79      0.61      0.65      6108
weighted avg       0.89      0.90      0.88      6108
```

In [101]:
```
p_poly = {'C':[0.1,1],'gamma':[0.1, 1], 'kernel':['poly']}
poly = SVC()
clss = GridSearchCV(poly, p_poly,'roc_auc',refit=True,cv=2)

clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.9125, with best C: {'C': 1, 'gamma': 0.1, 'kernel': 'poly'}
```

In [102]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.9011132940406025
[[5394   41]
 [ 563  110]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.73      0.16      0.27       673

    accuracy                           0.90      6108
   macro avg       0.82      0.58      0.61      6108
weighted avg       0.89      0.90      0.87      6108
```

decisionTreeClassifier()

In [103]:
```
dtree = DecisionTreeClassifier()

dtree.fit(X_train, y_train)
predict_y_test = dtree.predict(X_test)


print("Accuracy: "+ str(accuracy_score(y_test, predict_y_test)))
print("AUC: "+ str(roc_auc_score(y_test, dtree.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_y_test)))
print(classification_report(y_test, predict_y_test))
```

```
Accuracy: 0.8903077930582842
AUC: 0.7105314598708771
Confusion Matrix: [[5115  320]
 [ 350  323]]
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      5435
           1       0.50      0.48      0.49       673

    accuracy                           0.89      6108
   macro avg       0.72      0.71      0.71      6108
weighted avg       0.89      0.89      0.89      6108
```

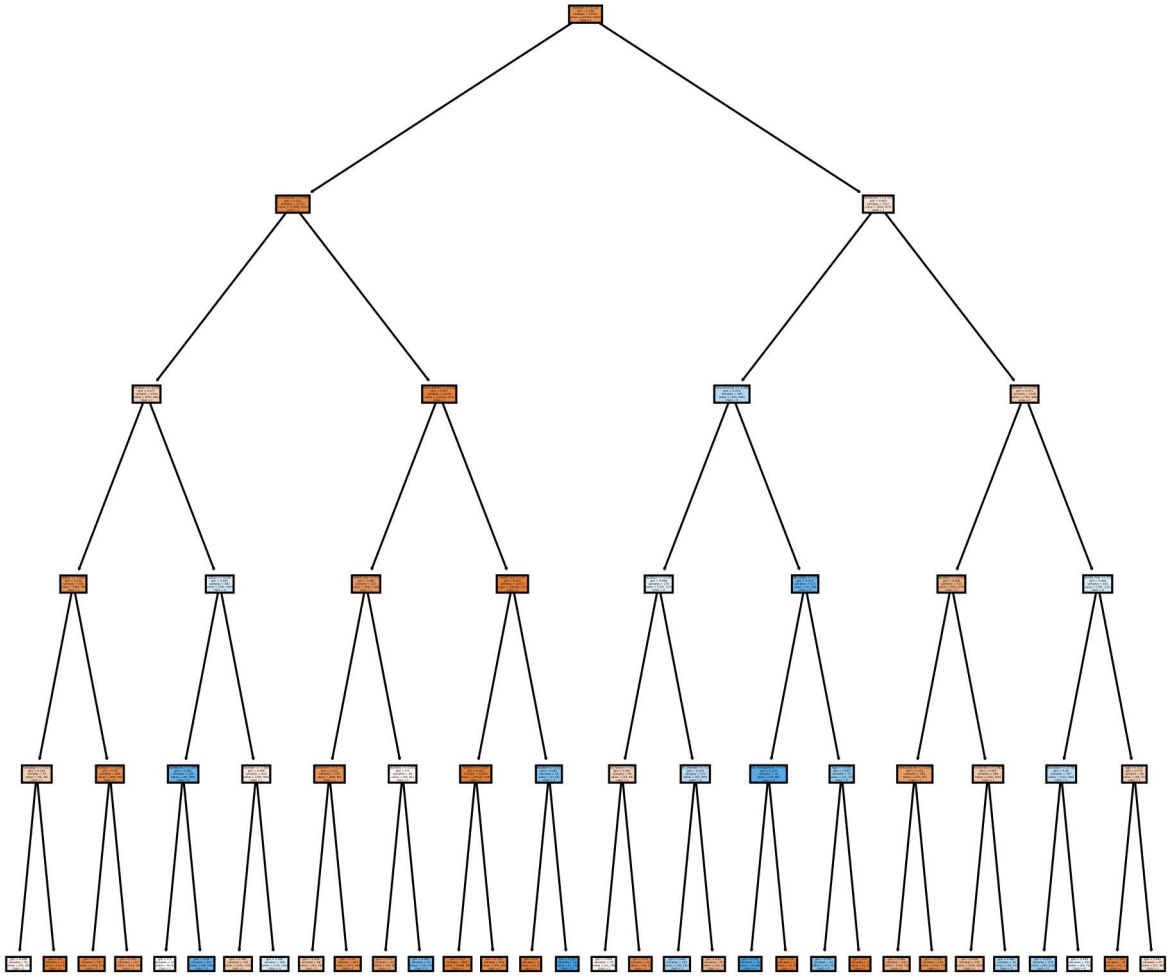
In [104]:
```
p5 = {'max_depth': [1,3, 5, 7, 10]}
clss = GridSearchCV(DecisionTreeClassifier(), p5,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.9091, with best C: {'max_depth': 5}
```

In [105]:
```
import graphviz
from sklearn import tree
dtree1=DecisionTreeClassifier(max_depth=5)
dtree1.fit(X=X_train, y=y_train)
```

Out[105]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [106]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (10,10), dpi=1080)
tree.plot_tree(dct1,
               feature_names =X.columns,class_names=str(y.unique()),
               filled = True);
```



In [107]:
```
predict_y= clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.9096267190569745
[[5255  180]
 [ 372  301]]
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      5435
           1       0.63      0.45      0.52       673

    accuracy                           0.91      6108
   macro avg       0.78      0.71      0.74      6108
weighted avg       0.90      0.91      0.90      6108
```

Conclusion

We have used AUC for evaluation rather Accuracy because - When we use accuracy, we assign equal cost to false positives and false negatives. When that data set is imbalanced - say it has 99% of instances in one class and only 1 % in the other - there is a great way to lower the cost. But as our data set is highly imbalaced so models should not be evaluated based on accuracy here. AUC has been used for model evaluation as it considers trade-off between True positive and true negatives. Models with high AUC can detect a large amount of true positives without losing its ability to detect true negatives and vice-versa.

After comparing the performance of all models in training and testing datasets, we see that basic Logistic regression model gave better performance w.r.t all the other models as it has the highest AUC value.

Logistic regularization with Inverse regularization parameter (C = 20) improved the results for classifying the people with salary (>50K).

Overall for this dataset the models that took less time for building, predicting and model which is easy to explain (Logistic regression and cross validated Decision tree) performed far more superior compared to the complex models (SVM with kernels).

Best Parameters for every model

Model	Params
Logistic regression	C = 1000
KNN Classification	n = 10
Linear SVC	C = 1.0
SVM Linear Kernel	C = 10, gamma = 0.1
SVM RBF Kernel	C = 1, gamma = 0.1
SVM Poly Kernel	C = 1, gamma = 0.1
Decision Tree	max_depth = 5

Performance in testing Dataset

Model	Logistic Regression	Logistic Regression	KNN Classification	KNN	Linear SVC	Linear SVC	SVM Linear	SVM Linear	SVM RBF	SVM RI
Params	default	C = 1000	default	n=10	default	C = 1.0	default	C=10, gamma=0.1	default	C=1,gamma=(
Accuracy	0.91	0.87	0.89	0.90	0.91	0.91	0.90	0.90	0.90	0.
F1-score(0)	0.95	0.92	0.94	0.95	0.95	0.95	0.95	0.95	0.95	0.
F1-score(1)	0.46	0.58	0.36	0.28	0.44	0.44	0.33	0.37	0.28	0.
AUC	0.91	0.93	0.789	0.795	--	0.931	0.929	0.93	0.922	0.9

In [108]:
```
## Logistic Regression with best parameters - This is selected to be the best
 model
```

In [381]:
```
logicR = LogisticRegression(C=1000)

logicR.fit(X_train, y_train)

predict_y_test = logicR.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predict_y_test)))
print("AUC: ", (roc_auc_score(y_test, logicR.predict_proba(X_test)[:,1])))
print("Confusion Matrix: ", confusion_matrix(y_test, predict_y_test), sep = '
\n')
print(classification_report(y_test, predict_y_test))
```

```
Accuracy: 0.9127373935821873
AUC:  0.9304319179387357
Confusion Matrix:
[[5308  127]
 [ 406  267]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      5435
           1       0.68      0.40      0.50       673

    accuracy                           0.91      6108
   macro avg       0.80      0.69      0.73      6108
weighted avg       0.90      0.91      0.90      6108
```

PART 2 of Banking Dataset Classification Project

In [125]:
```
#df was our final dataframe after the preprocessing in the previous part so we
will be using the same dataset to check which is already pre-processed to appl
y our final classification requriements
```

In [126]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20360 entries, 0 to 20593
Data columns (total 44 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   age                 20360 non-null  int64  
 1   education           20360 non-null  int32  
 2   contact             20360 non-null  int32  
 3   duration            20360 non-null  int64  
 4   campaign            20360 non-null  int64  
 5   pdays               20360 non-null  int64  
 6   previous            20360 non-null  int64  
 7   cons.price.idx      20360 non-null  float64
 8   cons.conf.idx       20360 non-null  float64
 9   euribor3m           20360 non-null  float64
 10  nr.employed         20360 non-null  float64
 11  y                   20360 non-null  int32  
 12  job_admin.          20360 non-null  uint8  
 13  job_blue-collar     20360 non-null  uint8  
 14  job_entrepreneur    20360 non-null  uint8  
 15  job_housemaid       20360 non-null  uint8  
 16  job_management      20360 non-null  uint8  
 17  job_retired         20360 non-null  uint8  
 18  job_self-employed   20360 non-null  uint8  
 19  job_services        20360 non-null  uint8  
 20  job_student         20360 non-null  uint8  
 21  job_technician      20360 non-null  uint8  
 22  job_unemployed      20360 non-null  uint8  
 23  job_unknown         20360 non-null  uint8  
 24  MS_divorced         20360 non-null  uint8  
 25  MS_married          20360 non-null  uint8  
 26  MS_single           20360 non-null  uint8  
 27  MS_unknown          20360 non-null  uint8  
 28  df_no               20360 non-null  uint8  
 29  df_unknown          20360 non-null  uint8  
 30  df_yes              20360 non-null  uint8  
 31  month_apr           20360 non-null  uint8  
 32  month_aug           20360 non-null  uint8  
 33  month_dec           20360 non-null  uint8  
 34  month_jul           20360 non-null  uint8  
 35  month_jun           20360 non-null  uint8  
 36  month_mar           20360 non-null  uint8  
 37  month_may           20360 non-null  uint8  
 38  month_nov           20360 non-null  uint8  
 39  month_oct           20360 non-null  uint8  
 40  month_sep           20360 non-null  uint8  
 41  po_failure          20360 non-null  uint8  
 42  po_nonexistent      20360 non-null  uint8  
 43  po_success          20360 non-null  uint8  
dtypes: float64(4), int32(3), int64(5), uint8(32)
memory usage: 3.0 MB
```

In [112]:
```
## just to check again for the null values
```

In [127]:
```
df.isnull().sum()
```

Out[127]:
```
age                  0
education            0
contact              0
duration             0
campaign             0
pdays                0
previous             0
cons.price.idx       0
cons.conf.idx        0
euribor3m            0
nr.employed          0
y                    0
job_admin.           0
job_blue-collar      0
job_entrepreneur     0
job_housemaid        0
job_management       0
job_retired          0
job_self-employed    0
job_services         0
job_student          0
job_technician       0
job_unemployed       0
job_unknown          0
MS_divorced          0
MS_married           0
MS_single            0
MS_unknown           0
df_no                0
df_unknown           0
df_yes               0
month_apr            0
month_aug            0
month_dec            0
month_jul            0
month_jun            0
month_mar            0
month_may            0
month_nov            0
month_oct            0
month_sep            0
po_failure           0
po_nonexistent       0
po_success           0
dtype: int64
```

In [114]:
```
# Apply two voting classifiers - one with hard voting and one with soft voting
```

In [115]:
```
# We will do all the imports here
```

In [128]:
```
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')
import glob
```

In [117]:
```
#invoking fit method on the voting Classifier
```

Voting Classifier

In [ ]:
```
# Importing Voting classifier
```

In [118]:
```
from sklearn.ensemble import VotingClassifier
```

In [ ]:
```
#fitting different modals
```

In [119]:
```
logicR= LogisticRegression(C=1000)
#using X_train and y_train from previous project part 1
logicR.fit(X_train, y_train)
```

Out[119]:
```
LogisticRegression(C=1000, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [120]:
```
dtree = DecisionTreeClassifier(max_depth = 5)
dtree.fit(X_train, y_train)
```

Out[120]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [121]:
```
Knn_c = KNeighborsClassifier(n_neighbors=10)
Knn_c.fit(X_train, y_train)
```

Out[121]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                     weights='uniform')
```

In [122]:
```
svc_l = LinearSVC(C=1.0)
svc_l.fit(X_train, y_train)
```

Out[122]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

In [123]:
```
L_svm = SVC(C = 10, kernel = 'linear', gamma=0.1, probability = True)
L_svm.fit(X_train, y_train)

r_svm = SVC(C = 1, kernel = 'rbf', gamma=0.1, probability = True)
r_svm .fit(X_train, y_train)

p_svm = SVC(C = 1, kernel = 'poly', gamma=0.1, probability = True)
p_svm.fit(X_train, y_train)
```

Out[123]:
```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='poly',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.0
01,
    verbose=False)
```

Hard Voting

In [124]:
```
hc = VotingClassifier(estimators=[('logicR', logicR),
                                  ('Knn_c',Knn_c),
                                  ('svc_l',svc_l),
                                  ('L_svm',L_svm),
                                  ('r_svm',r_svm),
                                  ('p_svm',p_svm),
                                  ('dtree', dtree)],
                      voting='hard')
hc.fit(X_train, y_train)

predict_y = hc.predict(X_test)
print('Accuracy Score',accuracy_score(y_test, predict_y))
print('Confusion Matrix',confusion_matrix(y_test, predict_y))
print('Classification Report',classification_report(y_test, predict_y))
print('ROC AUC score',roc_auc_score(y_test, predict_y))
```

```
Accuracy Score 0.9027504911591355
Confusion Matrix [[5353   82]
 [ 512  161]]
Classification Report               precision    recall  f1-score   support

           0       0.91      0.98      0.95      5435
           1       0.66      0.24      0.35       673

    accuracy                           0.90      6108
   macro avg       0.79      0.61      0.65      6108
weighted avg       0.89      0.90      0.88      6108

ROC AUC score 0.6120699718816598
```

In [ ]:
```
## While we can see that our classification model evaluation score when we do
 hard voting is very bad when compared to the accuracy score
```

Soft Voting

In [125]:
```
sc = VotingClassifier(estimators=[('logicR', logicR),
                                  ('Knn_c',Knn_c),
                                  ('L_svm',L_svm),
                                  ('r_svm',r_svm),
                                  ('p_svm',p_svm),
                                  ('dtree', dtree)],
                      voting='soft')
sc.fit(X_train, y_train)

# we dont use lsvc here as for soft voting you need a attribute called ''predi
ct_proba'' which is not present in lsvc
#link to documentation here - http://scikit-learn.org/stable/modules/generate
d/sklearn.svm.LinearSVC.html


predict_y_s = sc.predict(X_test)
print('Accuracy Score',accuracy_score(y_test, predict_y_s))
print('Confusion Matrix',confusion_matrix(y_test, predict_y_s))
print('Classification Report',classification_report(y_test, predict_y_s))
print('ROC AUC Scoring:',roc_auc_score(y_test, predict_y_s))
```

```
Accuracy Score 0.9027504911591355
Confusion Matrix [[5362   73]
 [ 521  152]]
Classification Report               precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.68      0.23      0.34       673

    accuracy                           0.90      6108
   macro avg       0.79      0.61      0.64      6108
weighted avg       0.89      0.90      0.88      6108

ROC AUC Scoring: 0.606211460308304
```

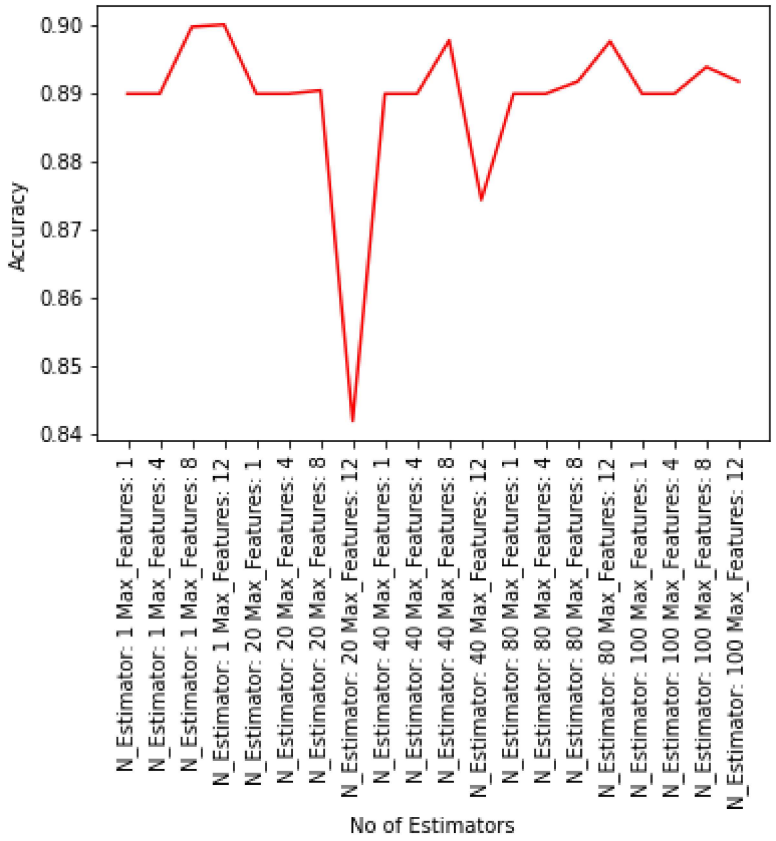
For soft voting also we are getting a score of Area under curve very low when compared to Accuracy!!

In [126]:
```
# Apply any two models with bagging and any two models with pasting.
```

Bagging for Logistic Regression

In [127]:
```
# import
from sklearn.ensemble import BaggingClassifier
```

In [146]:
```
e = [1,20,40,80,100]
mfeatures = [1,4,8,12]
pb=[]
accuracy = []

for estimator in e:
    for features in mfeatures:
        clss = BaggingClassifier(LogisticRegression(random_state=10, C=1000),
 \
                                 max_samples=0.5, \
                                 max_features=features, \
                                 n_estimators=estimator, \
                                bootstrap=True, random_state=10, oob_score=True
)
        clss.fit(X, y)
        accuracy_s = clss.score(X_test, y_test)

        pb.append("N_Estimator: " +str(estimator)+" Max_Features: "+str(featur
es))

        accuracy.append(accuracy_s)

plt.plot(pb, accuracy,color='red')
plt.ylabel("Accuracy")
plt.xlabel("No of Estimators")

plt.xticks(rotation=90)
plt.show()
```

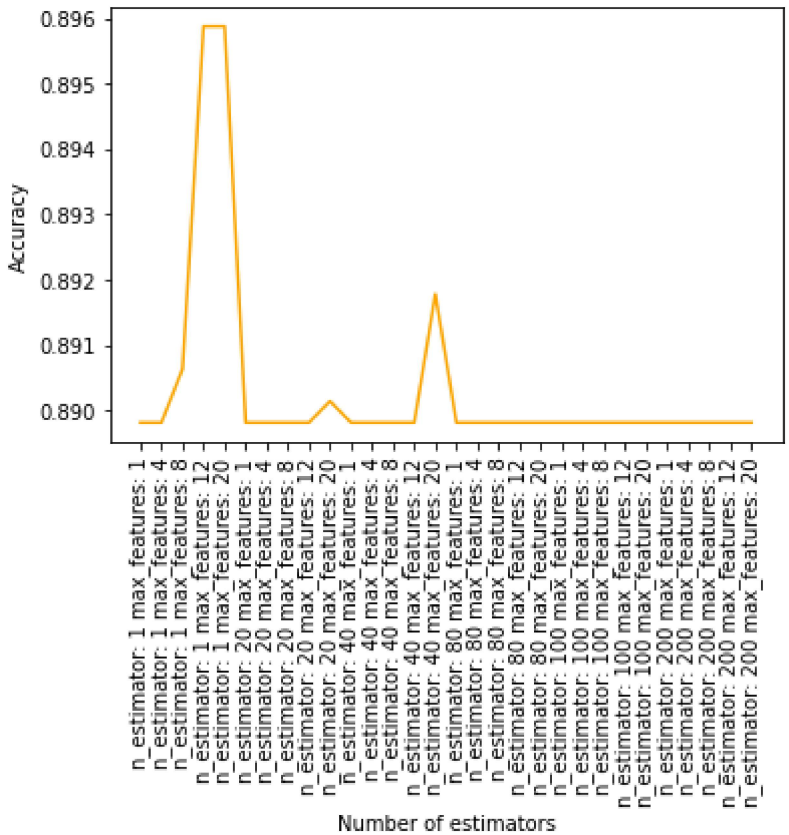


In [147]:
```
pp = {"n_estimators": [1,20,40,80,100],
                    "max_features":[1,4,8,12],
                    "max_samples": [0.5,0.1],
                    }
logicR= LogisticRegression(random_state=10, C=1000)
logicR_gs = GridSearchCV(BaggingClassifier(logicR, bootstrap=True, random_stat
e=10, oob_score=True),pp)
logicR_md = logicR_gs.fit(X_train,y_train)

print('Best out of ROC AUC: {:.4}, with best estimators {}'.format(logicR_md.b
est_score_, logicR_md.best_params_))

predict_y = logicR_md.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test,predict_y))
```

```
Best out of ROC AUC: 0.9038, with best estimators {'max_features': 12, 'max_s
amples': 0.5, 'n_estimators': 100}
0.8988212180746562
[[5414   21]
 [ 597   76]]
              precision    recall  f1-score   support

           0       0.90      1.00      0.95      5435
           1       0.78      0.11      0.20       673

    accuracy                           0.90      6108
   macro avg       0.84      0.55      0.57      6108
weighted avg       0.89      0.90      0.86      6108
```

Bagging for Decision Tree Classifier

In [132]:
```
n_e = [1,20,40,80,100]
mf = [1,4,8,12,20]
param_d=[]
accuracy_d = []

for estimator in n_e:
    for features in mf:
        clss = BaggingClassifier(DecisionTreeClassifier(random_state=10, max_d
epth = 5), \
                                 max_samples=0.5, \
                                 max_features=features, \
                                 n_estimators=estimator, \
                                bootstrap=True, random_state=0, oob_score=True)
        clss.fit(X, y)
        acc = clss.score(X_test, y_test)
        param_d.append("n_estimator: "+str(estimator)+" max_features: "+str(fe
atures))
        accuracy_d.append(acc)

plt.plot(param_d, accuracy_d,color='orange')
plt.xlabel("Number of estimators")
plt.xticks(rotation=90)
plt.ylabel("Accuracy")
plt.show()
```

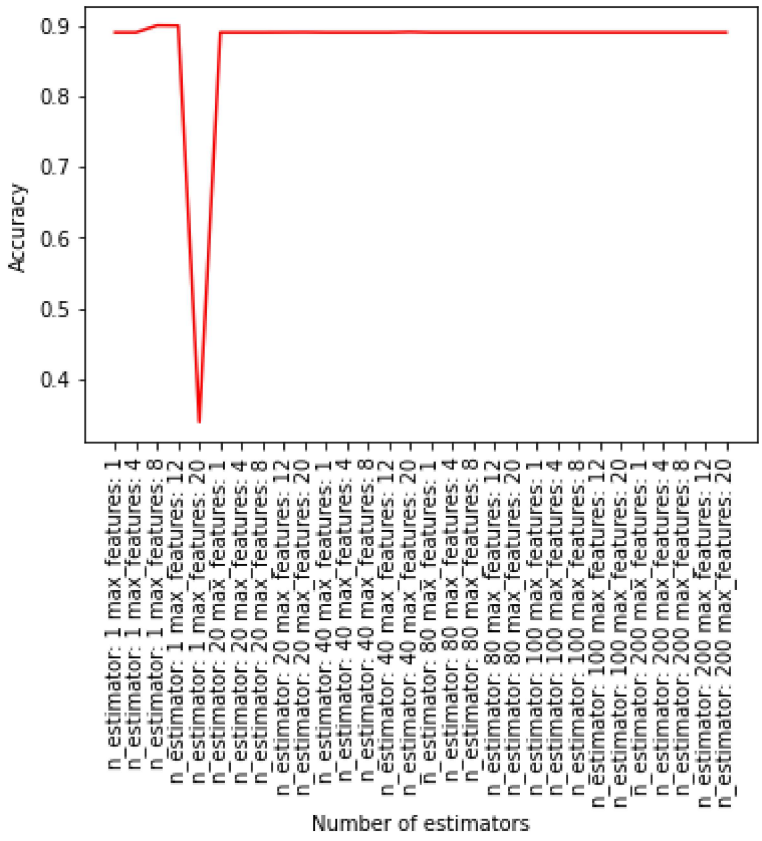


In [136]:
```
pp = {"n_estimators": [1,20,40,80,100],
                    "max_features":[1,4,8,12,20],
                    "max_samples": [0.5,0.1],
                    }
dtree = DecisionTreeClassifier(random_state=0, max_depth = 5)
dtree_gs = GridSearchCV(BaggingClassifier(dtree, bootstrap=True, random_state=
0, oob_score=True),pp)
dtree_md = dtree_gs.fit(X_train,y_train)

print('Best roc_auc: {:.4}, with best estimators {}'.format(dtree_md.best_scor
e_, dtree_md.best_params_))

Predict_y = dtree_md.predict(X_test)
print(accuracy_score(y_test, Predict_y))
print(confusion_matrix(y_test, Predict_y))
print(classification_report(y_test, Predict_y))
```

```
Best roc_auc: 0.9092, with best estimators {'max_features': 20, 'max_sample
s': 0.5, 'n_estimators': 80}
0.9043876882776686
[[5401   34]
 [ 550  123]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.78      0.18      0.30       673

    accuracy                           0.90      6108
   macro avg       0.85      0.59      0.62      6108
weighted avg       0.89      0.90      0.88      6108
```

Pasting Decision Tree Classifier

In [138]:
```
n_ep = [1,20,40,80,100]
mf_p = [1,4,8,12,20]
params_dtree_p=[]
accuracy_dtree_p = []

for estimator in n_ep:
    for features in mf_p:
        clss = BaggingClassifier(DecisionTreeClassifier(random_state=0, max_de
pth = 5), \
                                 bootstrap=False, random_state=10, \
                                 max_samples=0.5, \
                                 max_features=features, \
                                 n_estimators=estimator)
        clss.fit(X, y)
        acc = clss.score(X_test, y_test)
        params_dtree_p.append("n_estimator: "+str(estimator)+" max_features: "
+str(features))
        accuracy_dtree_p.append(acc)

plt.plot(params_dtree_p, accuracy_dtree_p,color='red')
plt.xlabel("Number of estimators")
plt.xticks(rotation=90)
plt.ylabel("Accuracy")
plt.show()
```

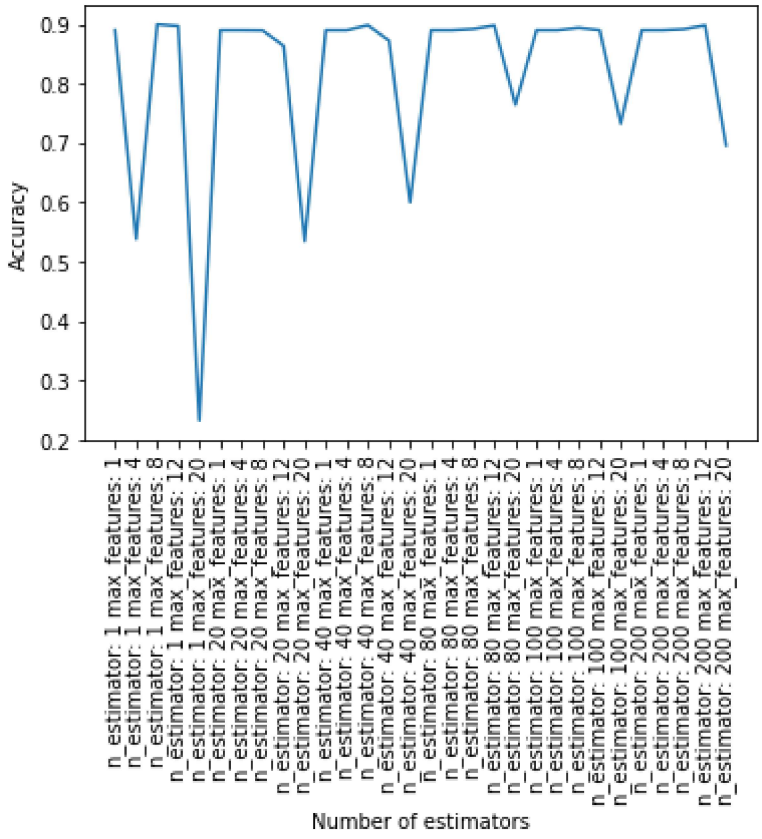


In [140]:
```
pp = {"n_estimators": [50,100,200,500],
                    "max_features":[1,2,4,6,8],
                    "max_samples": [0.5,0.1],
                    }
dtree_p = DecisionTreeClassifier(random_state=0, max_depth = 5)
dt_paste_gs = GridSearchCV(BaggingClassifier(dtree, bootstrap=False, random_st
ate=0),pp)
dt_paste_gs_model = dt_paste_gs.fit(X_train,y_train)

print('Best roc_auc: {:.4}, with best estimators {}'.format(dt_paste_gs_model.
best_score_, dt_paste_gs_model.best_params_))

predict_y = dt_paste_gs_model.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
Best roc_auc: 0.9042, with best estimators {'max_features': 8, 'max_samples':
0.5, 'n_estimators': 50}
0.9006221349050426
[[5418   17]
 [ 590   83]]
              precision    recall  f1-score   support

           0       0.90      1.00      0.95      5435
           1       0.83      0.12      0.21       673

    accuracy                           0.90      6108
   macro avg       0.87      0.56      0.58      6108
weighted avg       0.89      0.90      0.87      6108
```

Pasting for Logistic Regression

In [143]:
```
n = [1,20,40,80,100,200]
mf = [1,4,8,12,20]
a1=[]
a2 = []

for estimator in n:
    for features in mf:
        clss = BaggingClassifier(LogisticRegression(random_state=0, C=1000), \
                                 bootstrap=False, random_state=0, \
                                 max_samples=0.5, \
                                 max_features=features, \
                                 n_estimators=estimator)
        clss.fit(X, y)
        acc = clss.score(X_test, y_test)
        a1.append("n_estimator: "+str(estimator)+" max_features: "+str(feature
s))
        a2.append(acc)

plt.plot(a1, a2)
plt.xlabel("Number of estimators")
plt.xticks(rotation=90)
plt.ylabel("Accuracy")
plt.show()
```



In [148]:
```
pp = {"n_estimators": [1,20,40,80,100],
                    "max_features": [1,4,8,12],
                    "max_samples": [0.5,0.1],
                    }
logr = LogisticRegression(random_state=0, C=1000)
logr_pg = GridSearchCV(BaggingClassifier(logr, bootstrap=False, random_state=0
),pp)
logr_pg_model = logr_pg.fit(X_train,y_train)

print('Best roc_auc: {:.4}, with best estimators {}'.format(logr_pg_model.best
_score_, logr_pg_model.best_params_))

predict_y = logr_pg_model.predict(X_test)
print(accuracy_score(y_test,predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

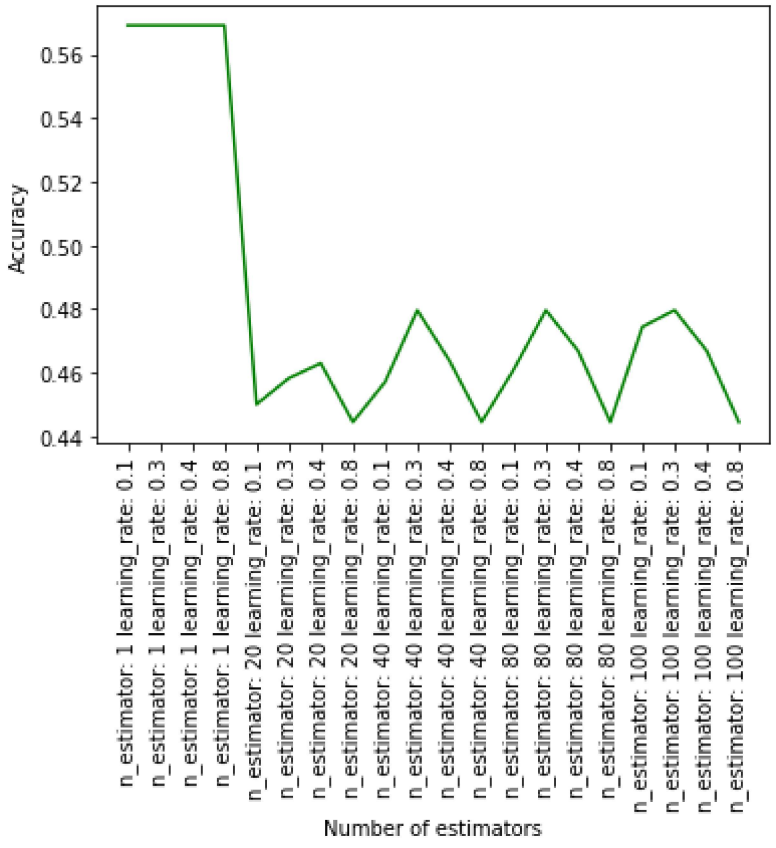
```
Best roc_auc: 0.9037, with best estimators {'max_features': 8, 'max_samples':
0.5, 'n_estimators': 1}
0.8996398166339227
[[5393   42]
 [ 571  102]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.95      5435
           1       0.71      0.15      0.25       673

    accuracy                           0.90      6108
   macro avg       0.81      0.57      0.60      6108
weighted avg       0.88      0.90      0.87      6108
```

ADABOOSTING

In [151]:
```
## ADABoosting
from sklearn.ensemble import AdaBoostClassifier
```

AdaBoosting for Logistic Regression

In [153]:
```
k = [1,20,40,80,100]
lr_ada = [0.1,0.3,0.4,0.8]
a1=[]
a2 = []

for estimator in k:
    for l in lr_ada:
        clss = AdaBoostClassifier(LogisticRegression(random_state=0, C=1000),
 \
                                  random_state=0, \
                                  n_estimators = estimator, \
                                  learning_rate = l, \
                                  algorithm = "SAMME.R"
                                 )
        clss.fit(X, y)
        acc = clss.score(X_test, y_test)
        a1.append("n_estimator: "+str(estimator)+" learning_rate: "+str(l))
        a2.append(acc)

plt.plot(a1, a2,color='green')
plt.xlabel("Number of estimators")
plt.xticks(rotation=90)
plt.ylabel("Accuracy")
plt.show()
```

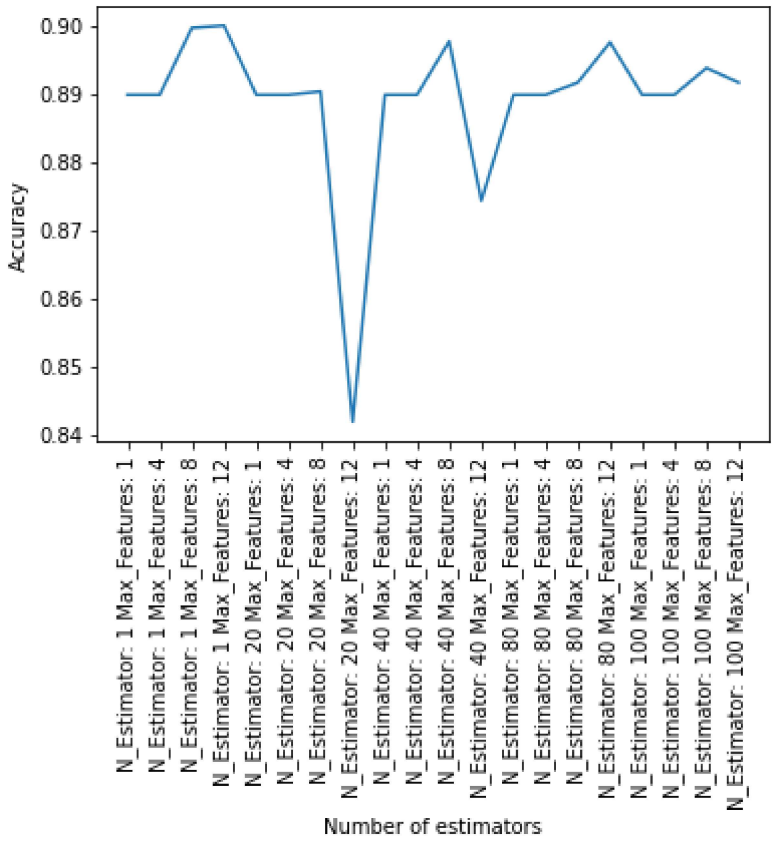


In [155]:
```
pp= {"n_estimators": [50,100,200],
                    "learning_rate":[0.4,0.5,0.6,0.7,0.8],
                    "algorithm":["SAMME.R","SAMME"],
                    }
LogicR = LogisticRegression(random_state=0, C=1000)
logr_ada_grid = GridSearchCV(AdaBoostClassifier(LogicR, random_state=0),pp)
logr_ada_grid_model = logr_ada_grid.fit(X_train,y_train)

print('Best roc_auc: {:.4}, with best estimators {}'.format(logr_ada_grid_mode
l.best_score_, logr_ada_grid_model.best_params_))

predict_y = logr_ada_grid_model.predict(X_test)
print('Accuracy Score', accuracy_score(y_test, predict_y))
print('Confusion Matrix', confusion_matrix(y_test, predict_y))
print('Classification Report', classification_report(y_test, predict_y))
```

```
Best roc_auc: 0.9105, with best estimators {'algorithm': 'SAMME.R', 'learning
_rate': 0.4, 'n_estimators': 50}
Accuracy Score 0.9071709233791748
Confusion Matrix [[5336   99]
 [ 468  205]]
Classification Report              precision    recall  f1-score   support

           0       0.92      0.98      0.95      5435
           1       0.67      0.30      0.42       673

    accuracy                           0.91      6108
   macro avg       0.80      0.64      0.68      6108
weighted avg       0.89      0.91      0.89      6108
```

AdABoosting for Decision Tree

In [156]:
```
n_e = [1,20,40,80,100,200]
lr = [0.1,0.2,0.3,0.4,0.8]
a1=[]
a2 = []

for estimator in n_e:
    for l in lr:
        clss = AdaBoostClassifier(DecisionTreeClassifier(random_state=0, max_d
epth = 5), \
                                  random_state=0, \
                                  n_estimators = estimator, \
                                  learning_rate = l, \
                                  algorithm = "SAMME"
                                 )
        clss.fit(X, y)
        acc = clss.score(X_test, y_test)
        a1.append("n_estimator: "+str(estimator)+" learning_rate: "+str(l))
        a2.append(acc)

plt.plot(params, accuracy)
plt.xlabel("Number of estimators")
plt.xticks(rotation=90)
plt.ylabel("Accuracy")
plt.show()
```



In [157]:
```
pp = {"n_estimators": [50,100,200],
                    "learning_rate":[0.4,0.5,0.6,0.7,0.8],
                    "algorithm":["SAMME.R","SAMME"],
                    }
dtree = DecisionTreeClassifier(random_state=0, max_depth = 5)
dtree_grid = GridSearchCV(AdaBoostClassifier(dtree, random_state=0),pp)
dtree_grid_model = dtree_grid.fit(X_train,y_train)

print('Best roc_auc: {:.4}, with best estimators {}'.format(dtree_grid_model.b
est_score_, dtree_grid_model.best_params_))

predict_y = dtree_grid_model.predict(X_test)
print('Accuracy Score', accuracy_score(y_test,predict_y))
print('Confusion Matrix', confusion_matrix(y_test, predict_y))
print('Classification Matrix', classification_report(y_test, predict_y))
```

```
Best roc_auc: 0.912, with best estimators {'algorithm': 'SAMME', 'learning_ra
te': 0.4, 'n_estimators': 100}
Accuracy Score 0.9112639161755075
Confusion Matrix [[5251  184]
 [ 358  315]]
Classification Matrix               precision    recall  f1-score   support

           0       0.94      0.97      0.95      5435
           1       0.63      0.47      0.54       673

    accuracy                           0.91      6108
   macro avg       0.78      0.72      0.74      6108
weighted avg       0.90      0.91      0.91      6108
```

Gradient Boosting

In [129]:
```
from sklearn.ensemble import GradientBoostingClassifier
pp = {"max_depth": [3,4,5],
                    "max_features":['log2', 'sqrt','auto'],
                    "learning_rate":[0.05,0.1,0.5,0.8,1],
                    "n_estimators": [100,200]}

gbg = GridSearchCV(GradientBoostingClassifier(random_state=0),tp)
gbg_model = gbg.fit(X_train,y_train)

print('Best roc_auc: {:.4}, with best estimators {}'.format(gbg_model.best_sco
re_, gbg_model.best_params_))

predict_y = gbg_model.predict(X_test)
print('Accuracy Score', accuracy_score(y_test,predict_y))
print('Confusion Matrix', confusion_matrix(y_test,predict_y))
print('Classification Report',classification_report(y_test, predict_y)
```

```
Best roc_auc: 0.9185, with best estimators {'learning_rate': 0.05, 'max_dept
h': 4, 'max_features': 'auto', 'n_estimators': 100}
Accuracy Score 0.9135559921414538
Confusion Matrix [[5265  170]
 [ 358  315]]
Classification Report               precision    recall  f1-score   support

           0       0.94      0.97      0.95      5435
           1       0.65      0.47      0.54       673

    accuracy                           0.91      6108
   macro avg       0.79      0.72      0.75      6108
weighted avg       0.90      0.91      0.91      6108
```

PCA

In [130]:
```
from sklearn.decomposition import PCA
```

In [133]:
```
p98 = PCA(.98,random_state=0)
p98.fit(X_train)
p99.n_components_
```

Out[133]: 26

In [132]:
```
p95 = PCA(.95,random_state=0)
p95.fit(X_train)
p95.n_components_
```

Out[132]: 21

In [ ]:
```
## creating a dataset set which is transformed and naming it X_train and X_tes
t
```

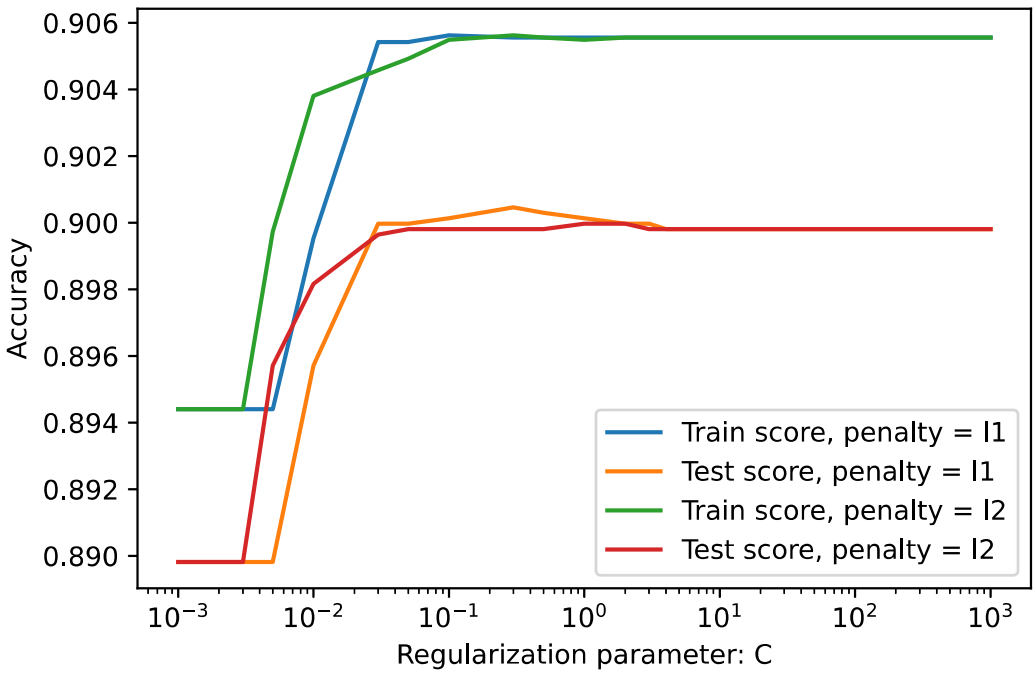
In [196]:
```
X_train_1= X_train
X_test_1= X_test
X_train = p95.transform(X_train)
X_test = p95.transform(X_test)
```

Logicstic Regression - PCA

In [135]:
```
logicR = LogisticRegression()
logicR.fit(X_train, y_train)
predict_test = logicR.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predict_test)))
print("AUC: "+ str(roc_auc_score(y_test, logicR.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_test)))
print(classification_report(y_test, predict_test))
```

```
Accuracy: 0.8999672560576293
AUC: 0.7718772853840676
Confusion Matrix: [[5393   42]
 [ 569  104]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.95      5435
           1       0.71      0.15      0.25       673

    accuracy                           0.90      6108
   macro avg       0.81      0.57      0.60      6108
weighted avg       0.88      0.90      0.87      6108
```

In [137]:
```
import matplotlib.pyplot as plt
%matplotlib inline

r = [0.001,0.003,0.01,0.03,0.05,0.1,0.3,0.5,1,5,10,20,100]
trs_l1 = []
trs_l2 = []
ts_l1 = []
ts_l2 = []

for c in r:
    logicR_1 = LogisticRegression(penalty = 'l1', C = c,solver='liblinear')
    logicR_2 = LogisticRegression(penalty = 'l2', C = c)
    logicR_1.fit(X_train, y_train)
    logicR_2.fit(X_train, y_train)
    trs_l1.append(log_l1.score(X_train, y_train))
    trs_l2.append(log_l2.score(X_train, y_train))
    ts_l1.append(log_l1.score(X_test, y_test))
    ts_l2.append(log_l2.score(X_test, y_test))


plt.plot(r,trs_l1, label = 'Train score, penalty = l1')
plt.plot(r, ts_l1, label = 'Test score, penalty = l1')
plt.plot(r, trs_l2, label = 'Train score, penalty = l2')
plt.plot(r, ts_l2, label = 'Test score, penalty = l2')
plt.legend()


plt.xlabel('Regularization parameter: C')
plt.ylabel('Accuracy')
plt.xscale('log')
```



In [139]:
```
logicR = LogisticRegression(class_weight='balanced')
pp = {'C':[0.001,0.003,0.01,0.03,0.05,0.1,0.3,0.5,1,5,10,20,100]}
clss = GridSearchCV(logicR,pp,scoring='roc_auc',refit=True,cv=4)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.2}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.778, with best C: {'C': 0.03}
```

In [140]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.806483300589391
[[4509  926]
 [ 256  417]]
              precision    recall  f1-score   support

           0       0.95      0.83      0.88      5435
           1       0.31      0.62      0.41       673

    accuracy                           0.81      6108
   macro avg       0.63      0.72      0.65      6108
weighted avg       0.88      0.81      0.83      6108
```

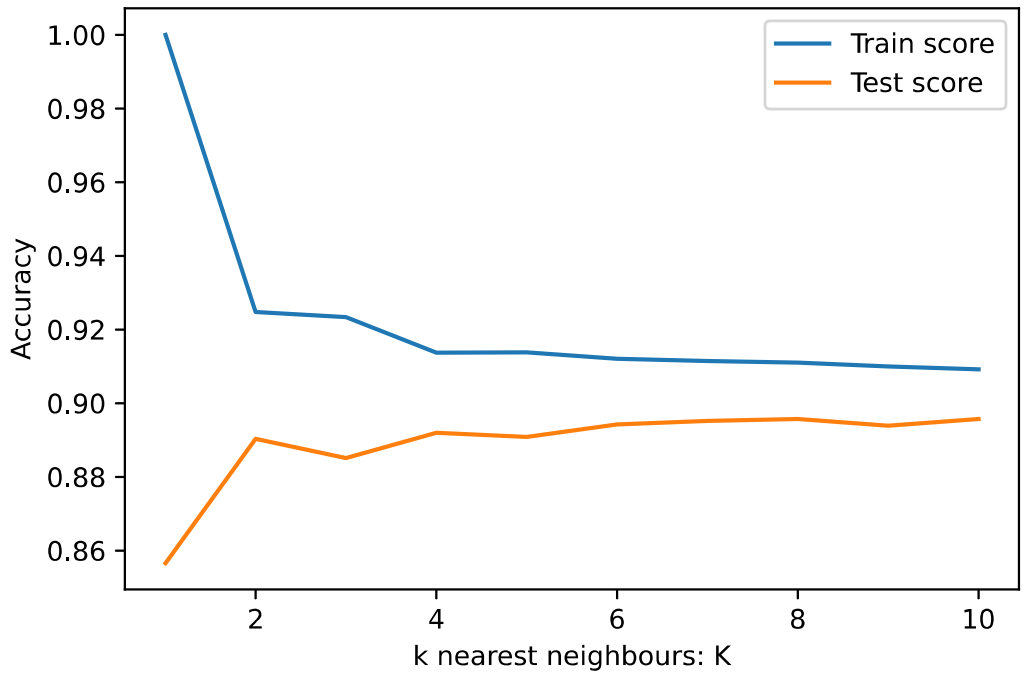
KNN - PCA

In [141]:
```
k = KNeighborsClassifier()
k.fit(X_train, y_train)

predict_test = k.predict(X_test)

print("Accuracy: "+ str(accuracy_score(y_test, predict_test)))
print("AUC: "+ str(roc_auc_score(y_test, knn.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_test)))

print(classification_report(y_test, predict_test))
```

```
Accuracy: 0.8907989521938441
AUC: 0.7076708254106685
Confusion Matrix: [[5273  162]
 [ 505  168]]
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      5435
           1       0.51      0.25      0.33       673

    accuracy                           0.89      6108
   macro avg       0.71      0.61      0.64      6108
weighted avg       0.87      0.89      0.87      6108
```

In [142]:
```
r = list(range(1, 11))
trs_knn = []
ts_knn = []

for k in r:
    clss = KNeighborsClassifier(n_neighbors=k)
    clss.fit(X_train, y_train)
    trsknn.append(clss.score(X_train, y_train))
    ts_knn.append(clss.score(X_test, y_test))

plt.plot(r, trs_knn, label = 'Train score')
plt.plot(r, ts_knn, label = 'Test score')
plt.legend()
plt.xlabel('k nearest neighbours: K')
plt.ylabel('Accuracy')
```

Out[142]:
```
Text(0, 0.5, 'Accuracy')
```



In [143]:
```
r1 = list(range(1, 11))
k = KNeighborsClassifier()
p2 = dict(n_neighbors=r)
clss = GridSearchCV(k, p2,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.7301, with best C: {'n_neighbors': 10}
```

In [144]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.8957105435494433
[[5349   86]
 [ 551  122]]
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      5435
           1       0.59      0.18      0.28       673

    accuracy                           0.90      6108
   macro avg       0.75      0.58      0.61      6108
weighted avg       0.87      0.90      0.87      6108
```

LINEAR SVM

In [145]:
```
svm_l = LinearSVC()

svm_l.fit(X_train, y_train)

predict_test = svm_l.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predict_test)))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_test)))
print(classification_report(y_test, predict_test))
```

```
Accuracy: 0.9007858546168959
Confusion Matrix: [[5392   43]
 [ 563  110]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.72      0.16      0.27       673

    accuracy                           0.90      6108
   macro avg       0.81      0.58      0.61      6108
weighted avg       0.88      0.90      0.87      6108
```

In [147]:
```
p = {"C": [0.01, 0.1, 1.0]}
svm_l = LinearSVC()
clss = GridSearchCV(svm_l, p,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.7769, with best C: {'C': 0.01}
```

In [148]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.900294695481336
[[5392   43]
 [ 566  107]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.71      0.16      0.26       673

    accuracy                           0.90      6108
   macro avg       0.81      0.58      0.60      6108
weighted avg       0.88      0.90      0.87      6108
```

Linear Kernel

In [150]:
```
l_svc = SVC(kernel='linear', probability=True)

l_svc.fit(X_train, y_train)

predict_test = l_svc.predict(X_test)


print("Accuracy: "+ str(accuracy_score(y_test, predict_test)))
print("AUC: "+ str(roc_auc_score(y_test, l_svc.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_test)))
print(classification_report(y_test, predict_test))
```

```
Accuracy: 0.8996398166339227
AUC: 0.7528839411059517
Confusion Matrix: [[5383   52]
 [ 561  112]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.68      0.17      0.27       673

    accuracy                           0.90      6108
   macro avg       0.79      0.58      0.61      6108
weighted avg       0.88      0.90      0.87      6108
```

In [151]:
```
d = {'C':[0.1, 1, 10],'gamma':[0.1, 1, 10], 'kernel':['linear']}
lsvc = SVC()
clss = GridSearchCV(lsvc, d,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.7492, with best C: {'C': 0.1, 'gamma': 0.1, 'kernel': 'linea
r'}
```

In [ ]:
```
## Acuracy and confusion matrix below
```

In [152]:
```
predict_y = clss.predict(X_test)

print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.8996398166339227
[[5383   52]
 [ 561  112]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.68      0.17      0.27       673

    accuracy                           0.90      6108
   macro avg       0.79      0.58      0.61      6108
weighted avg       0.88      0.90      0.87      6108
```

RBF Kernel

In [154]:
```
r_svc = SVC(kernel='rbf', probability=True)
r_svc.fit(X_train, y_train)

predict_test = r_svc.predict(X_test)

print("Accuracy: "+ str(accuracy_score(y_test, predict_test)))
print("AUC: "+ str(roc_auc_score(y_test, r_svc.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_test)))
print(classification_report(y_test, predict_test))
```

```
Accuracy: 0.8993123772102161
AUC: 0.707121718103044
Confusion Matrix: [[5392   43]
 [ 572  101]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.95      5435
           1       0.70      0.15      0.25       673

    accuracy                           0.90      6108
   macro avg       0.80      0.57      0.60      6108
weighted avg       0.88      0.90      0.87      6108
```

In [155]:
```
j = {'C':[0.1, 1, 10],'gamma':[0.1, 1, 10], 'kernel':['rbf']}
r_svc = SVC()
clss = GridSearchCV(r_svc, j,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.7546, with best C: {'C': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}
```

In [156]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.8996398166339227
[[5383   52]
 [ 561  112]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      5435
           1       0.68      0.17      0.27       673

    accuracy                           0.90      6108
   macro avg       0.79      0.58      0.61      6108
weighted avg       0.88      0.90      0.87      6108
```

Poly Kernel

In [158]:
```
p_svc = SVC(kernel='poly', probability=True)
p_svc.fit(X_train, y_train)

predict_test = p_svc.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predict_test)))
print("AUC: "+ str(roc_auc_score(y_test, p_svc.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_test)))
print(classification_report(y_test, predict_test))
```

```
Accuracy: 0.8994760969220694
AUC: 0.6602914902720385
Confusion Matrix: [[5392   43]
 [ 571  102]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.95      5435
           1       0.70      0.15      0.25       673

    accuracy                           0.90      6108
   macro avg       0.80      0.57      0.60      6108
weighted avg       0.88      0.90      0.87      6108
```

In [159]:
```
l = {'C':[1],'gamma':[0.1, 1], 'kernel':['poly']}
p_svc = SVC()
clss = GridSearchCV(p_svc, l,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.7049, with best C: {'C': 1, 'gamma': 0.1, 'kernel': 'poly'}
```

In [160]:
```
predict_y = clss.predict(X_test)
print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.899803536345776
[[5393   42]
 [ 570  103]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.95      5435
           1       0.71      0.15      0.25       673

    accuracy                           0.90      6108
   macro avg       0.81      0.57      0.60      6108
weighted avg       0.88      0.90      0.87      6108
```

Decision Tree - PCA

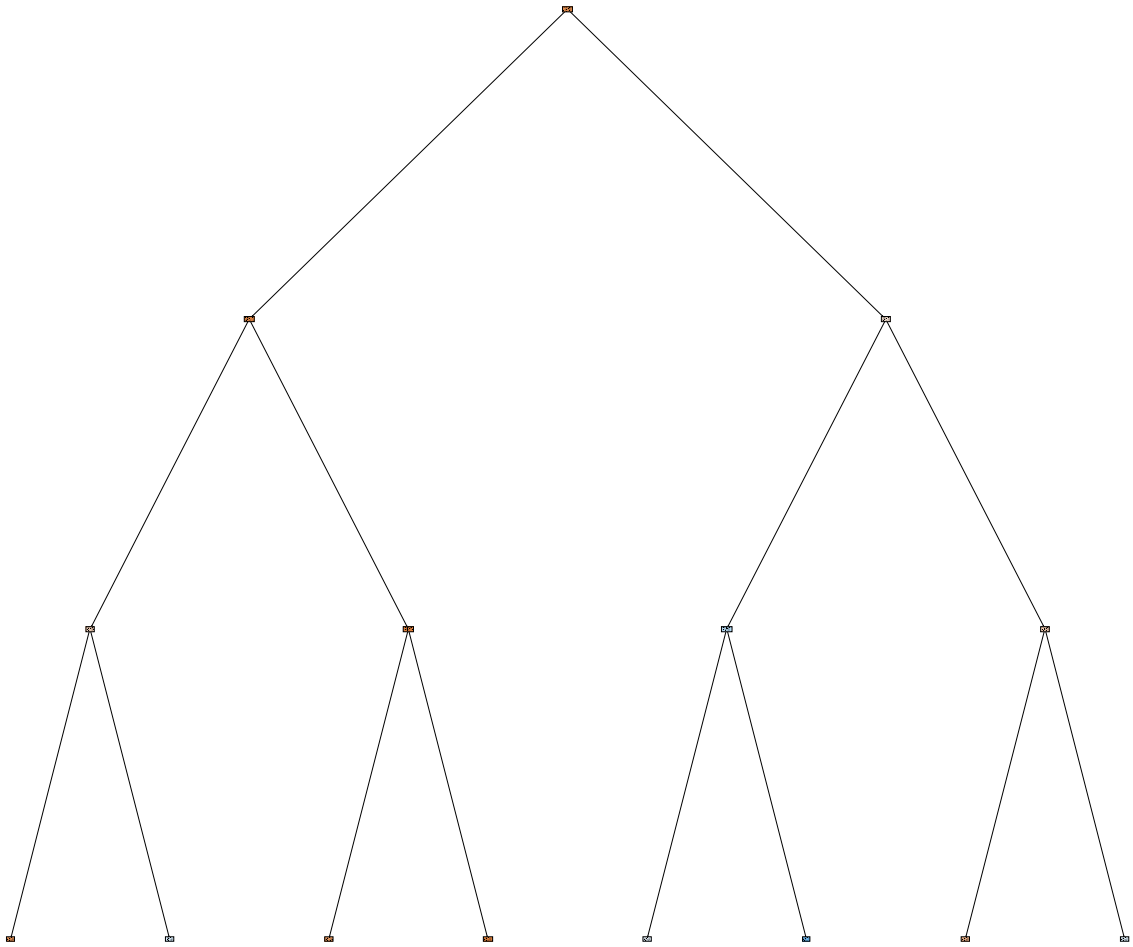
In [161]:
```
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)

predict_test = dtree.predict(X_test)


print("Accuracy: "+ str(accuracy_score(y_test, predict_test)))
print("AUC: "+ str(roc_auc_score(y_test, dtree.predict_proba(X_test)[:,1])))
print("Confusion Matrix: "+ str(confusion_matrix(y_test, predict_test)))
print(classification_report(y_test, predict_test))
```

```
Accuracy: 0.849705304518664
AUC: 0.6174142335941034
Confusion Matrix: [[4975  460]
 [ 458  215]]
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      5435
           1       0.32      0.32      0.32       673

    accuracy                           0.85      6108
   macro avg       0.62      0.62      0.62      6108
weighted avg       0.85      0.85      0.85      6108
```

In [162]:
```
m = {'max_depth': [1,3, 5, 7, 10]}
clss = GridSearchCV(DecisionTreeClassifier(), m,'roc_auc',refit=True,cv=3)
clss.fit(X_train,y_train)
print('Best roc_auc: {:.4}, with best C: {}'.format(clss.best_score_, clss.bes
t_params_))
```

```
Best roc_auc: 0.724, with best C: {'max_depth': 3}
```

In [302]:
```
import graphviz
from sklearn import tree
dtree_g=DecisionTreeClassifier(max_depth=3)
dtree_g.fit(X=X_train, y=y_train)

fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (10,10), dpi=1080)
tree.plot_tree(cart1,
               feature_names =X.columns,class_names=str(y.unique()),
               filled = True);
```



In [165]:
```
predict_y = clss.predict(X_test)

print(accuracy_score(y_test, predict_y))
print(confusion_matrix(y_test, predict_y))
print(classification_report(y_test, predict_y))
```

```
0.8984937786509496
[[5388   47]
 [ 573  100]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.95      5435
           1       0.68      0.15      0.24       673

    accuracy                           0.90      6108
   macro avg       0.79      0.57      0.59      6108
weighted avg       0.88      0.90      0.87      6108
```

Deep Learning

In [287]:
```
X_train = np.asarray(X_train_1)
X_test = np.asarray(X_test_1)
y_train = np.asarray(y_train)
y_test = np.asarray(y_test)
```

In [294]:
```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import roc_auc_score
import tensorflow as tf


deepLearning_model = Sequential()

deepLearning_model.add(Dense(75, activation= 'relu',kernel_initializer='normal', input_dim=43))

deepLearning_model.add(Dense(50,activation= 'softplus',kernel_initializer='normal'))

deepLearning_model.add(Dense(25, activation= 'tanh',kernel_initializer='normal'))

deepLearning_model.add(Dense(1, kernel_initializer= 'normal', activation= 'sigmoid'))

deepLearning_model.compile(optimizer= 'adam',loss='binary_crossentropy', metrics=['accuracy'])

deepLearning_model.fit(X_train, y_train, epochs= 25, batch_size=45)
```

```
Epoch 1/25
317/317 [==============================] - 0s 785us/step - loss: 0.3215 - acc
uracy: 0.8940
Epoch 2/25
317/317 [==============================] - 0s 770us/step - loss: 0.2431 - acc
uracy: 0.9053
Epoch 3/25
317/317 [==============================] - 0s 792us/step - loss: 0.2040 - acc
uracy: 0.9121
Epoch 4/25
317/317 [==============================] - 0s 791us/step - loss: 0.1905 - acc
uracy: 0.9135
Epoch 5/25
317/317 [==============================] - 0s 1ms/step - loss: 0.1850 - accur
acy: 0.9138
Epoch 6/25
317/317 [==============================] - 0s 997us/step - loss: 0.1843 - acc
uracy: 0.9140
Epoch 7/25
317/317 [==============================] - 0s 816us/step - loss: 0.1805 - acc
uracy: 0.9152
Epoch 8/25
317/317 [==============================] - 0s 773us/step - loss: 0.1789 - acc
uracy: 0.9162
Epoch 9/25
317/317 [==============================] - 0s 758us/step - loss: 0.1789 - acc
uracy: 0.9155
Epoch 10/25
317/317 [==============================] - 0s 800us/step - loss: 0.1774 - acc
uracy: 0.9168
Epoch 11/25
317/317 [==============================] - 0s 755us/step - loss: 0.1755 - acc
uracy: 0.9167
Epoch 12/25
317/317 [==============================] - 0s 810us/step - loss: 0.1733 - acc
uracy: 0.9184
Epoch 13/25
317/317 [==============================] - 0s 1ms/step - loss: 0.1733 - accur
acy: 0.9177
Epoch 14/25
317/317 [==============================] - 0s 783us/step - loss: 0.1738 - acc
uracy: 0.9181
Epoch 15/25
317/317 [==============================] - 0s 810us/step - loss: 0.1711 - acc
uracy: 0.9180
Epoch 16/25
317/317 [==============================] - 0s 833us/step - loss: 0.1717 - acc
uracy: 0.9189
Epoch 17/25
317/317 [==============================] - 0s 871us/step - loss: 0.1698 - acc
uracy: 0.9216
Epoch 18/25
317/317 [==============================] - 0s 950us/step - loss: 0.1675 - acc
uracy: 0.9219
Epoch 19/25
317/317 [==============================] - 0s 951us/step - loss: 0.1665 - acc
uracy: 0.9210
```

```
Epoch 20/25
317/317 [==============================] - 0s 843us/step - loss: 0.1670 - acc
uracy: 0.9215
Epoch 21/25
317/317 [==============================] - 0s 849us/step - loss: 0.1655 - acc
uracy: 0.9209
Epoch 22/25
317/317 [==============================] - 0s 873us/step - loss: 0.1660 - acc
uracy: 0.9230
Epoch 23/25
317/317 [==============================] - 0s 824us/step - loss: 0.1650 - acc
uracy: 0.9229
Epoch 24/25
317/317 [==============================] - 0s 1ms/step - loss: 0.1627 - accur
acy: 0.9246
Epoch 25/25
317/317 [==============================] - 0s 840us/step - loss: 0.1632 - acc
uracy: 0.9250
```

Out[294]: <tensorflow.python.keras.callbacks.History at 0x2506abaf880>

In [295]:
```
predict_y = deepLearning_model.predict(X_test)
print('Acc Score Count',accuracy_score(y_test, predict_y.round(), normalize =
False))
print('Confusion Matrix',confusion_matrix(y_test, predict_y.round()))
print('Classification Report',classification_report(y_test, predict_y.round
()))
```

```
Acc Score Count 5599
Confusion Matrix [[5244  191]
 [ 318  355]]
Classification Report               precision    recall  f1-score   support

           0       0.94      0.96      0.95      5435
           1       0.65      0.53      0.58       673

    accuracy                           0.92      6108
   macro avg       0.80      0.75      0.77      6108
weighted avg       0.91      0.92      0.91      6108
```

Conclusion

Ensemble learning

Model	Hard Voting	Soft Voting	Bagging Decision Tree	Bagging Logistic Regressor	Pasting Decision Tree	Pasting Logistics Regressor	Adaboosting Decision Tree	Adaboosting Logistic Regressor	Gradient Boosting	De Learni
AUC	0.61	0.606	0.909	0.903	0.904	0.903	0.912	0.91	0.918	
Accuracy	0.9	0.902	0.904	0.89	0.9	0.899	0.91	0.9	0.91	0

Below are the results from the project 1

Model	Logistic Regression	Logistic Regression	KNN Classification	KNN	Linear SVC	Linear SVC	SVM Linear	SVM Linear	SVM RBF	SVM RBI
Params	default	C = 1000	default	n=10	default	C = 1.0	default	C=10, gamma=0.1	default	C=1,gamma=0.
AUC	0.91	0.93	0.789	0.795	--	0.931	0.929	0.93	0.922	0.91

Below is the result after Project 2 after PCA

Model	Logistic Regression	KNN	Linear SVC	SVM Linear	SVM RBF	SVM Poly	Decision Tree
Params	C = 0.03	n=10	C = 0.01	C=0.1, gamma=0.1	C=0.1,gamma=0.1	C=0.1, gamma=0.1	max_depth = 3
AUC	0.778	0.73	0.776	0.7492	0.7546	0.7049	0.724

As discussed in project part 1 our dataset is imbalanced so we have used AUC instead of accuracy to evaluate our models. Hence, on the similar basis when we compare our current result with the previous models we can see that when we used PCA and performed the gridsearch for best parameter we were not able to match the preivous outcome of AUC.

Hence the best model will be Logistic regression with C=1000 and PCA does not help us in finding the better model.

In [383]:
```
lr = LogisticRegression(C=1000)
lr.fit(X_train, y_train)
predictions = lr.predict(X_test)
print("Accuracy: "+ str(accuracy_score(y_test, predictions)))
print("AUC: ", (roc_auc_score(y_test, lr.predict_proba(X_test)[:,1])))
print("Confusion Matrix: ", confusion_matrix(y_test, predictions), sep = '\n')
print(classification_report(y_test, predictions))
```

```
Accuracy: 0.9127373935821873
AUC:  0.9304319179387357
Confusion Matrix:
[[5308  127]
 [ 406  267]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      5435
           1       0.68      0.40      0.50       673

    accuracy                           0.91      6108
   macro avg       0.80      0.69      0.73      6108
weighted avg       0.90      0.91      0.90      6108
```