This project is aimed at building a predictive model to help a bank with its direct marketing. The bank has historical data for over 45,000 customers to build a predictive model to predict whether a customer will go for term deposit or not by using the attributes bank has in its database. Your goal is to study the data and build such a predictive model. You will evaluate your model by using N-fold cross-validation. Link to data for the project: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing (https://archive.ics.uci.edu/ml/datasets/Bank+Marketing) You will work with the full dataset with 20 attributes. While evaluating your model, you may want to take into account cost associated with different errors.

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [91]:

```
bdata =pd.read_csv(r"G:/Yamuna docs/College docs/Info Ret/Project 1/bank-addit
ional-full.csv",  delimiter=";",header='infer') #header='infer'
bdata.head()
```

Out[91]:

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

5 rows × 21 columns

In [92]:
```
#find the missing values in the dataset
print(bdata.isnull().sum())
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

In [93]:
```
#Dropping the duplicates
bdata = bdata.drop_duplicates()

bdata.shape
```

Out[93]:
```
(41176, 21)
```

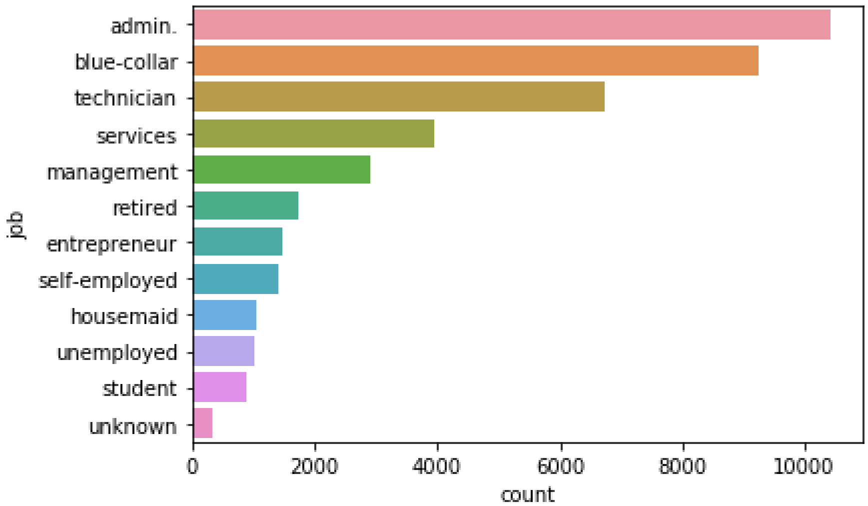
In [6]:
```
#summary statistics for categorical variables
bdata.describe(include=['object'])
cate = bdata.describe(include=['object']).columns
print(cate)
bdata.describe(include=['object'])
```

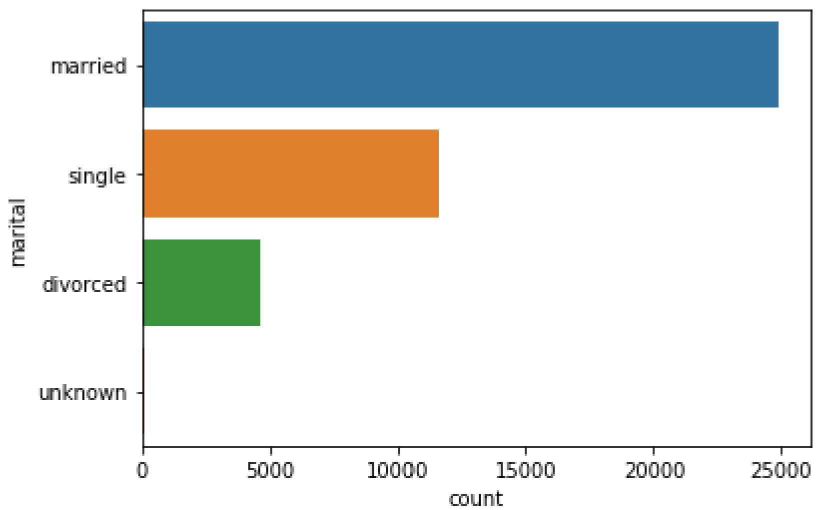
```
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contac
t',
       'month', 'day_of_week', 'poutcome', 'y'],
      dtype='object')
```

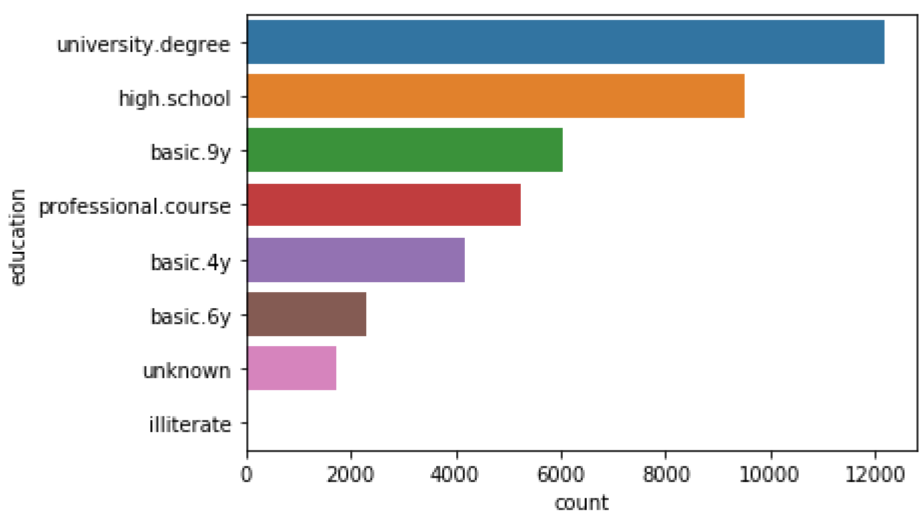
Out[6]:

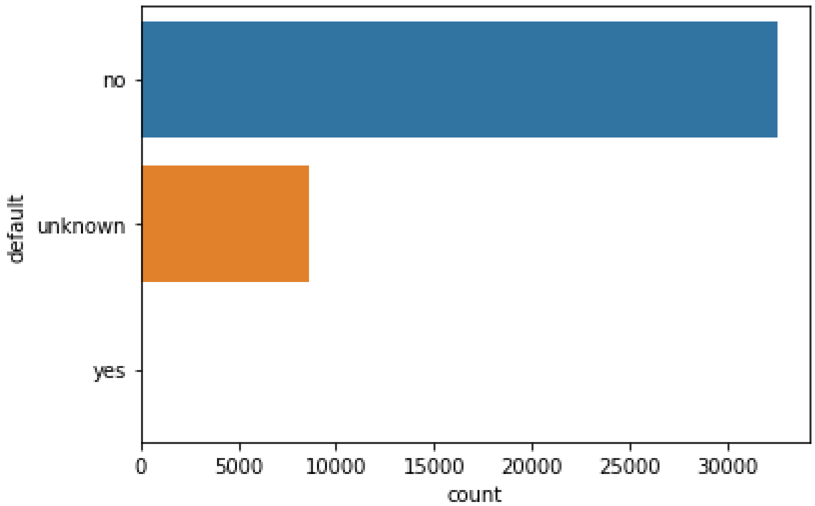
	job	marital	education	default	housing	loan	contact	month	day_of_week
count	41176	41176	41176	41176	41176	41176	41176	41176	41176
unique	12	4	8	3	3	3	2	10	5
top	admin.	married	university.degree	no	yes	no	cellular	may	thu
freq	10419	24921	12164	32577	21571	33938	26135	13767	8618

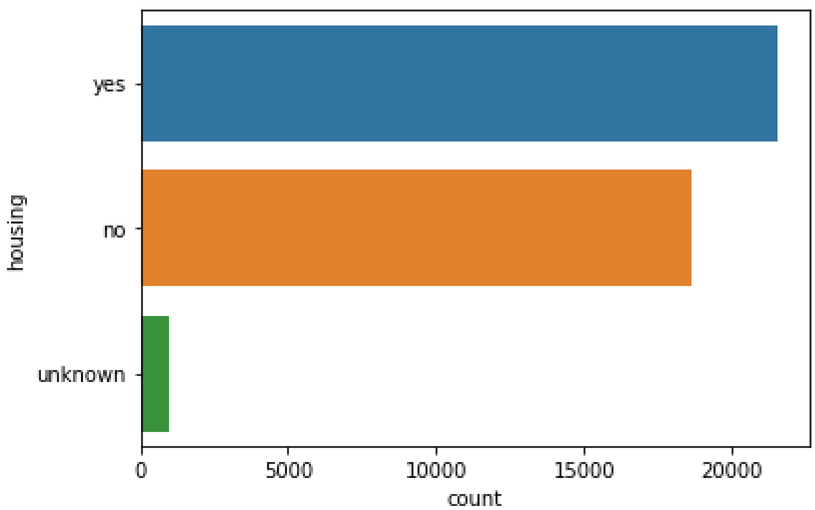
In [6]:
```
import seaborn as sns

#Bar plots of categorical features
for feature in bdata.dtypes[bdata.describe(include=['object']).columns].index:
    sns.countplot(y=feature, data=bdata, order = bdata[feature].value_counts()
.index)
    plt.show()
```

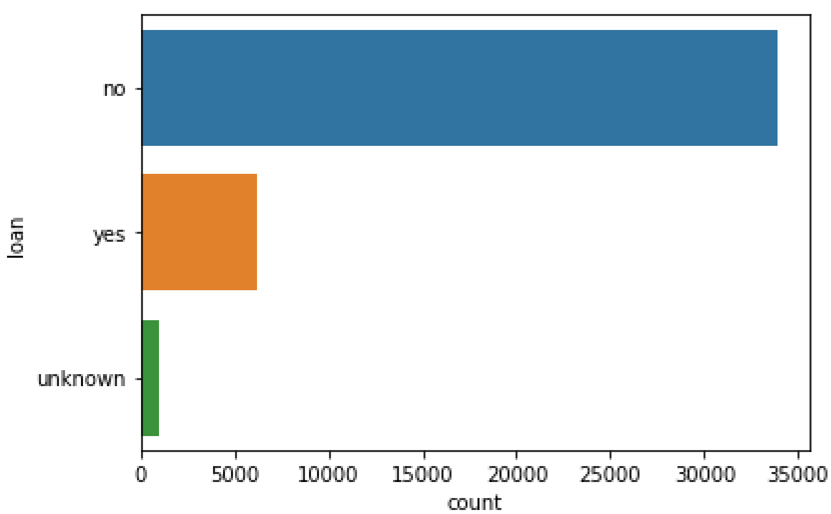

admin.
blue-collar
technician
services
management
retired
entrepreneur
self-employed
housemaid
unemployed
student
unknown
job
0
2000
4000
6000
8000
10000
count

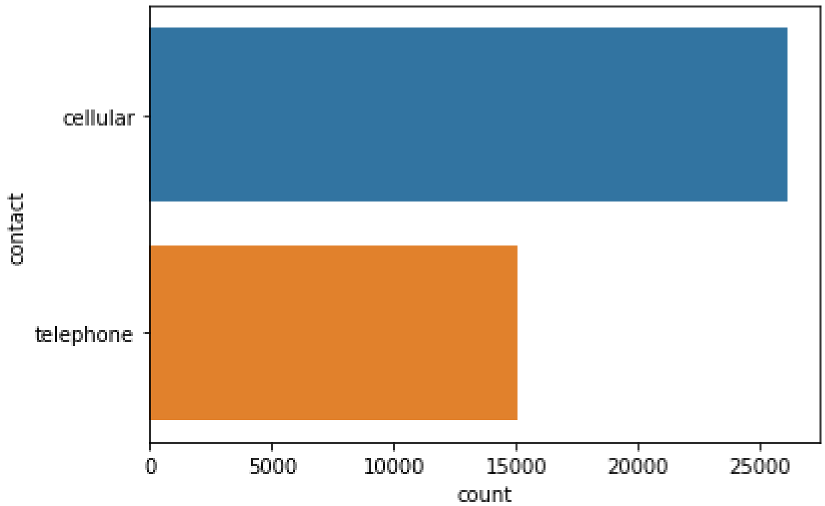

married
single
divorced
unknown
marital
0
5000
10000
15000
20000
25000
count

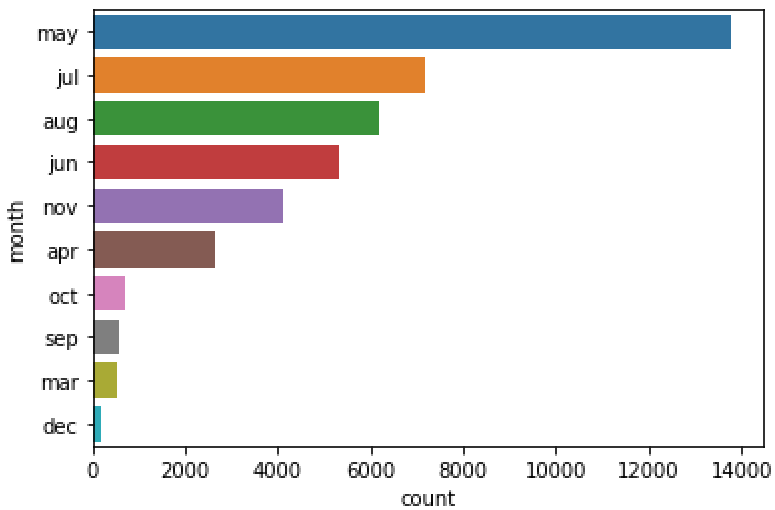

university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
unknown
illiterate
education
0
2000
4000
6000
8000
10000
12000
count

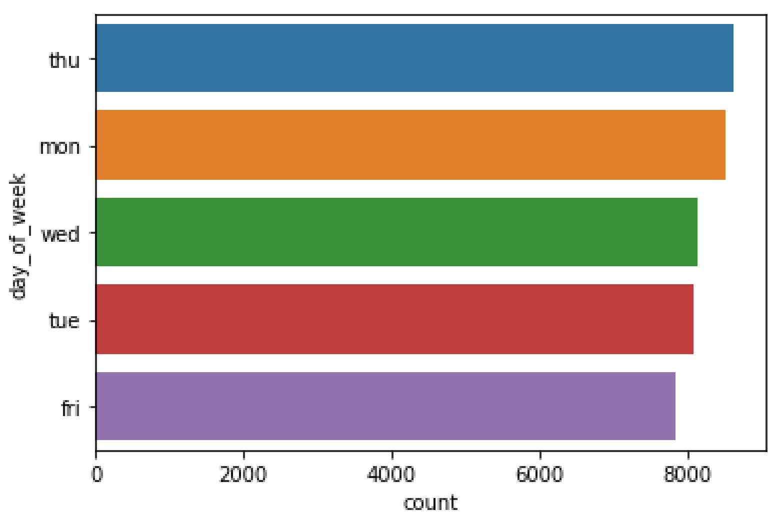

default
no
unknown
yes
0
5000
10000
15000
20000
25000
30000
count

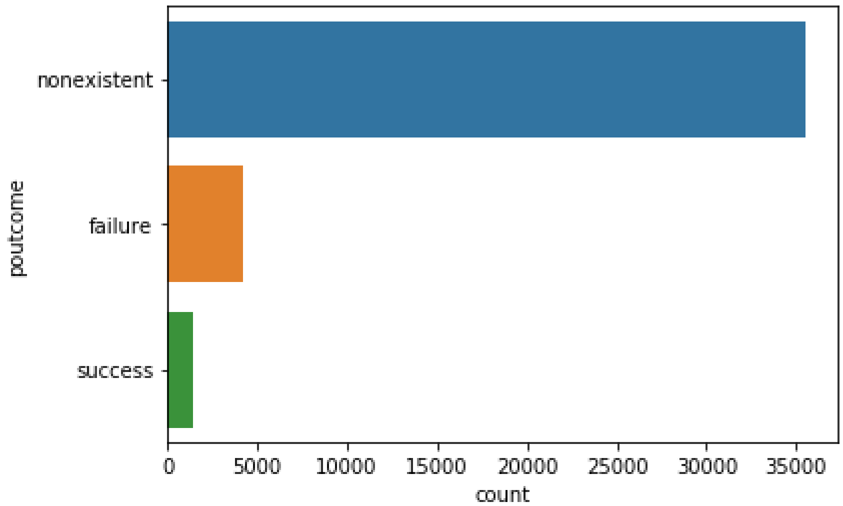

housing
yes
no
unknown
0
5000
10000
15000
20000
count

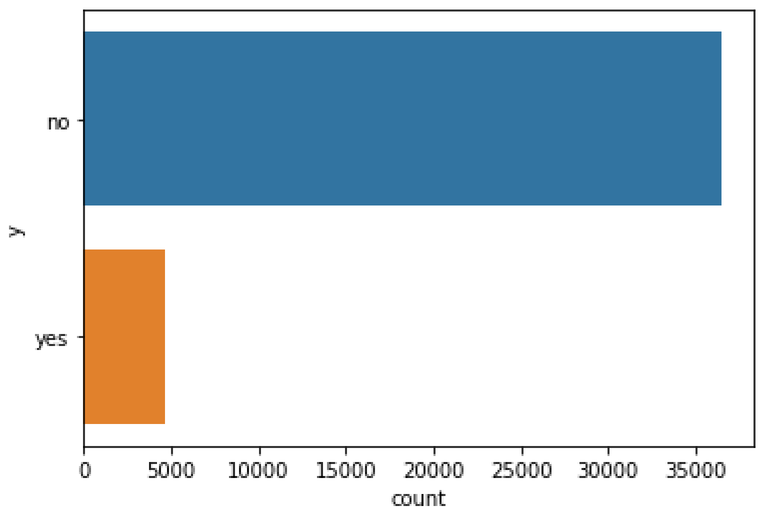

loan
no
yes
unknown
0
5000
10000
15000
20000
25000
30000
35000
count


contact
cellular
telephone
0
5000
10000
15000
20000
25000
count


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


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



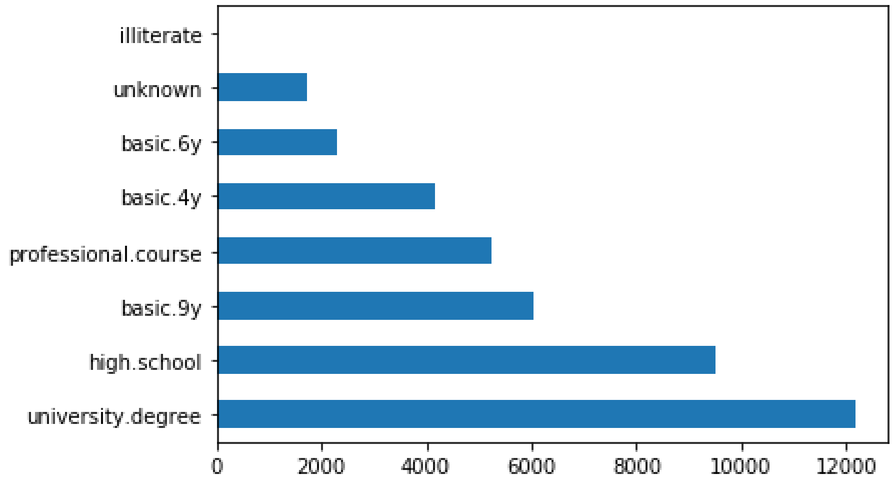


In [7]:
```
#pd.crosstab(index=bdata["education"], columns=bdata["y"])
bdata.education.value_counts()/bdata.education.count()
```

Out[7]:
```
university.degree      0.295415
high.school            0.231008
basic.9y               0.146809
professional.course    0.127259
basic.4y               0.101418
basic.6y               0.055639
unknown                0.042015
illiterate             0.000437
Name: education, dtype: float64
```

In [8]:
```
bdata.education.value_counts().plot(kind="barh")
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x2990acb5d30>

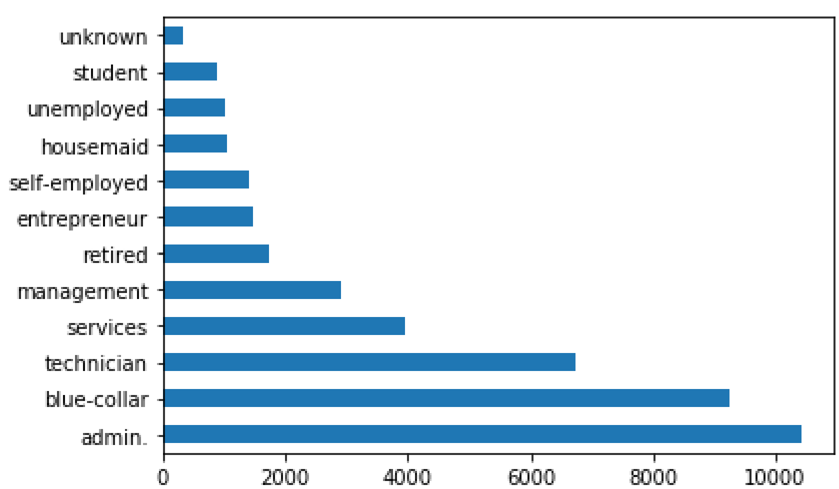


In [9]:
```
#pd.crosstab(index=bdata["job"], columns=bdata["y"])
bdata.job.value_counts()/bdata.job.count()
```

Out[9]:
```
admin.           0.253036
blue-collar      0.224718
technician       0.163663
services         0.096343
management       0.071012
retired          0.041723
entrepreneur     0.035360
self-employed    0.034510
housemaid        0.025743
unemployed       0.024626
student          0.021250
unknown          0.008014
Name: job, dtype: float64
```

In [10]:
```
bdata.job.value_counts().plot(kind="barh")
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x2990b06ccc0>

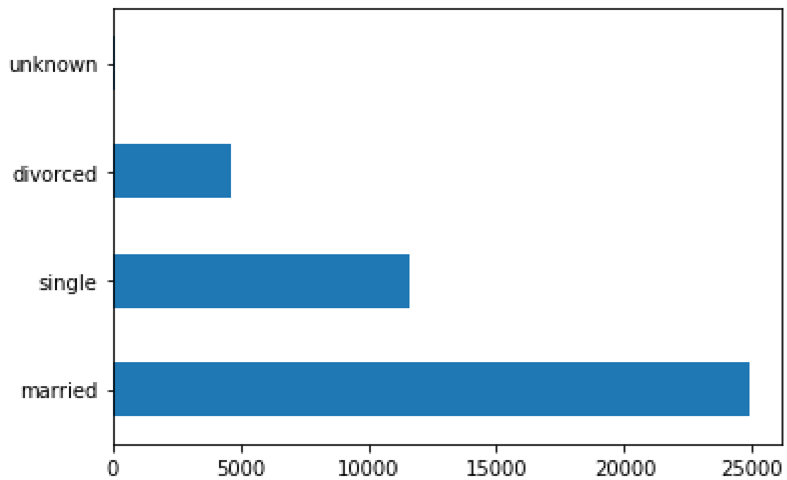


In [11]:
```
bdata.marital.value_counts()/bdata.marital.count()
```

Out[11]:
```
married     0.605231
single      0.280843
divorced    0.111983
unknown     0.001943
Name: marital, dtype: float64
```

In [12]:
```
bdata.marital.value_counts().plot(kind="barh")
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2990acb0470>

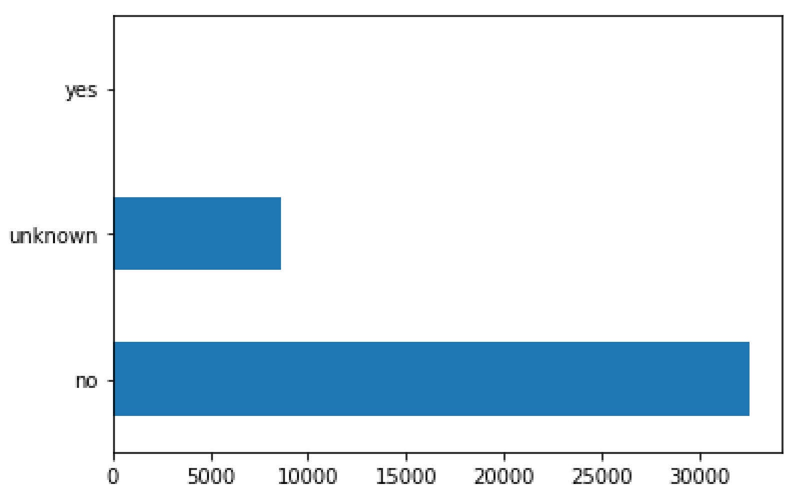


In [13]:
```
bdata.default.value_counts()/bdata.default.count()
```

Out[13]:
```
no         0.791165
unknown    0.208762
yes        0.000073
Name: default, dtype: float64
```

In [14]:
```
bdata.default.value_counts().plot(kind="barh")
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2990aeb5780>

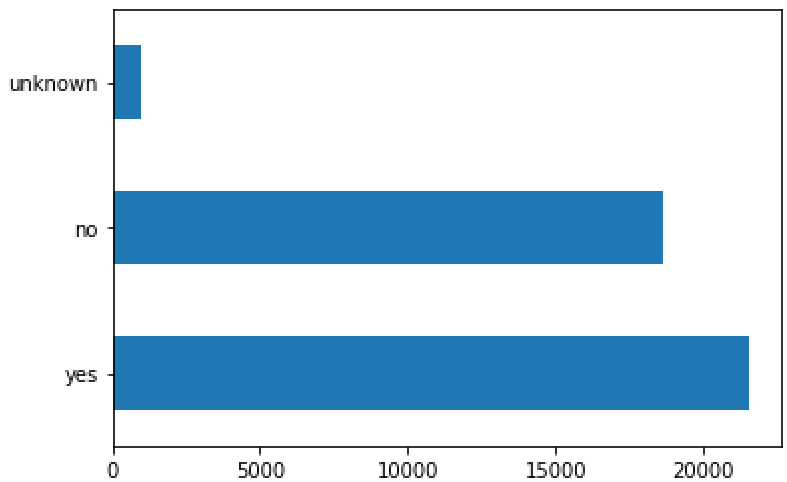


In [15]:
```
bdata.housing.value_counts()/bdata.housing.count()
```

Out[15]:
```
yes        0.523873
no         0.452084
unknown    0.024043
Name: housing, dtype: float64
```

In [16]:
```
bdata.housing.value_counts().plot(kind="barh")
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x2990aeb5cf8>

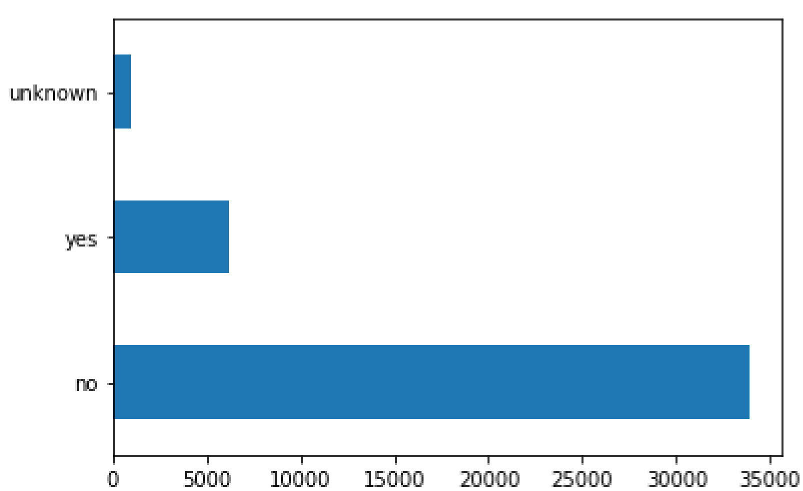


In [17]:
```
bdata.loan.value_counts()/bdata.loan.count()
```

Out[17]:
```
no         0.824218
yes        0.151739
unknown    0.024043
Name: loan, dtype: float64
```

In [18]:
```
bdata.loan.value_counts().plot(kind="barh")
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x2990ae4be48>

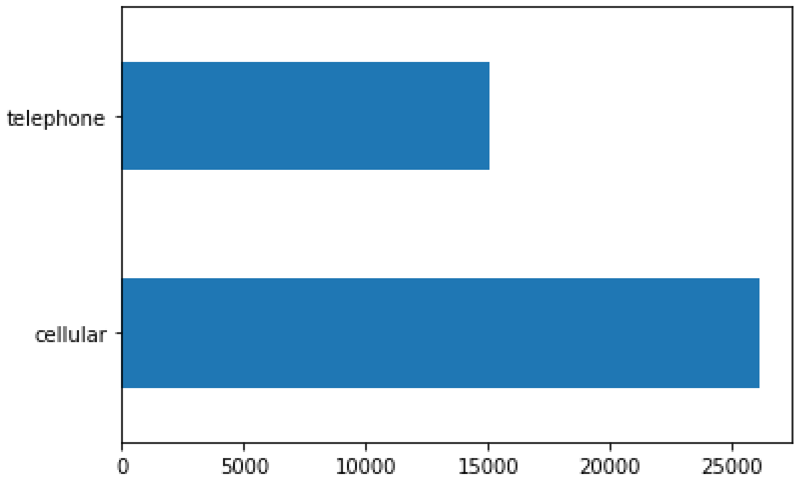


In [19]:
```
bdata.contact.value_counts()/bdata.contact.count()
```

Out[19]:
```
cellular     0.634714
telephone    0.365286
Name: contact, dtype: float64
```

In [20]:
```
bdata.contact.value_counts().plot(kind="barh")
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2990ae0db38>
```

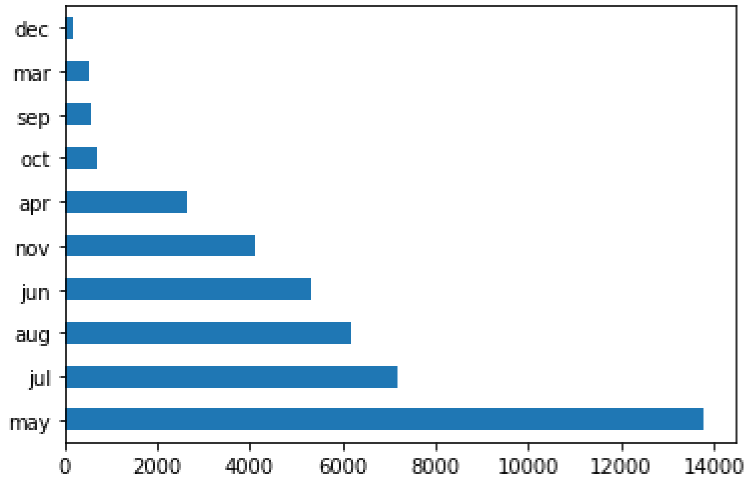


In [21]:
```
bdata.month.value_counts()/bdata.contact.count()
```

Out[21]:
```
may    0.334345
jul    0.174106
aug    0.149990
jun    0.129153
nov    0.099573
apr    0.063896
oct    0.017413
sep    0.013843
mar    0.013260
dec    0.004420
Name: month, dtype: float64
```

In [22]:
```
bdata.month.value_counts().plot(kind="barh")
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x2990b0c9cf8>

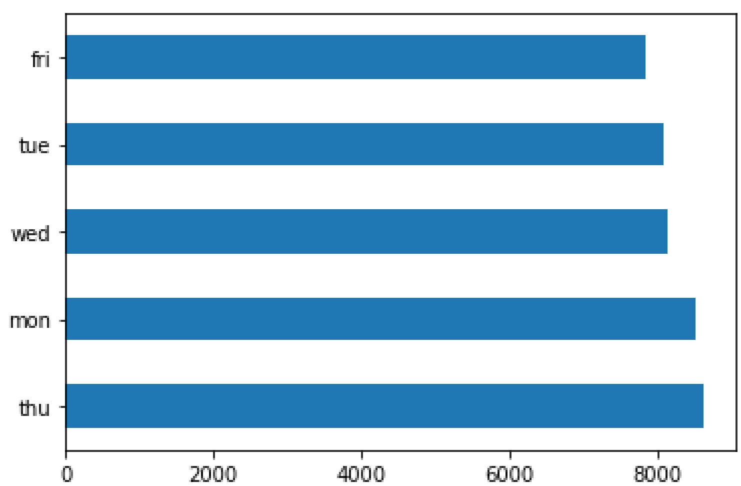


In [23]:
```
bdata.day_of_week.value_counts()/bdata.day_of_week.count()
```

Out[23]:
```
thu    0.209297
mon    0.206722
wed    0.197542
tue    0.196377
fri    0.190062
Name: day_of_week, dtype: float64
```

In [24]:
```
bdata.day_of_week.value_counts().plot(kind="barh")
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x2990b14ec88>

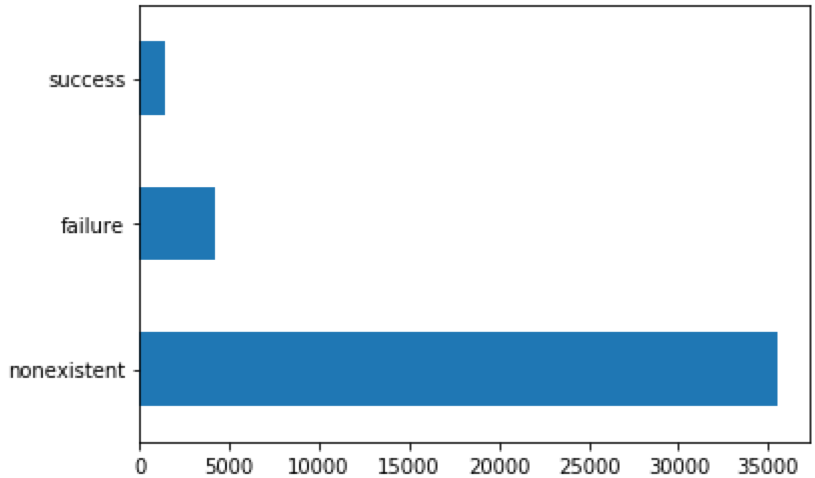


In [25]:
```
bdata.poutcome.value_counts()/bdata.poutcome.count()
```

Out[25]:
```
nonexistent    0.863391
failure        0.103264
success        0.033345
Name: poutcome, dtype: float64
```

In [26]:
```
bdata.poutcome.value_counts().plot(kind="barh")
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x2990b1b4dd8>

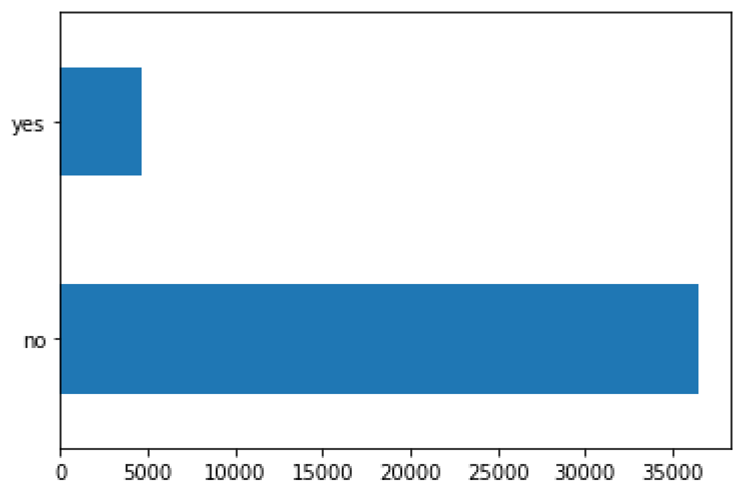


In [27]:
```
bdata.y.value_counts()/bdata.y.count()
```

Out[27]:
```
no     0.887337
yes    0.112663
Name: y, dtype: float64
```

In [28]:
```
bdata.y.value_counts().plot(kind="barh")
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x2990b062ac8>



In [29]:
```
#bdata.duration.value_counts()/bdata.duration.count()
```

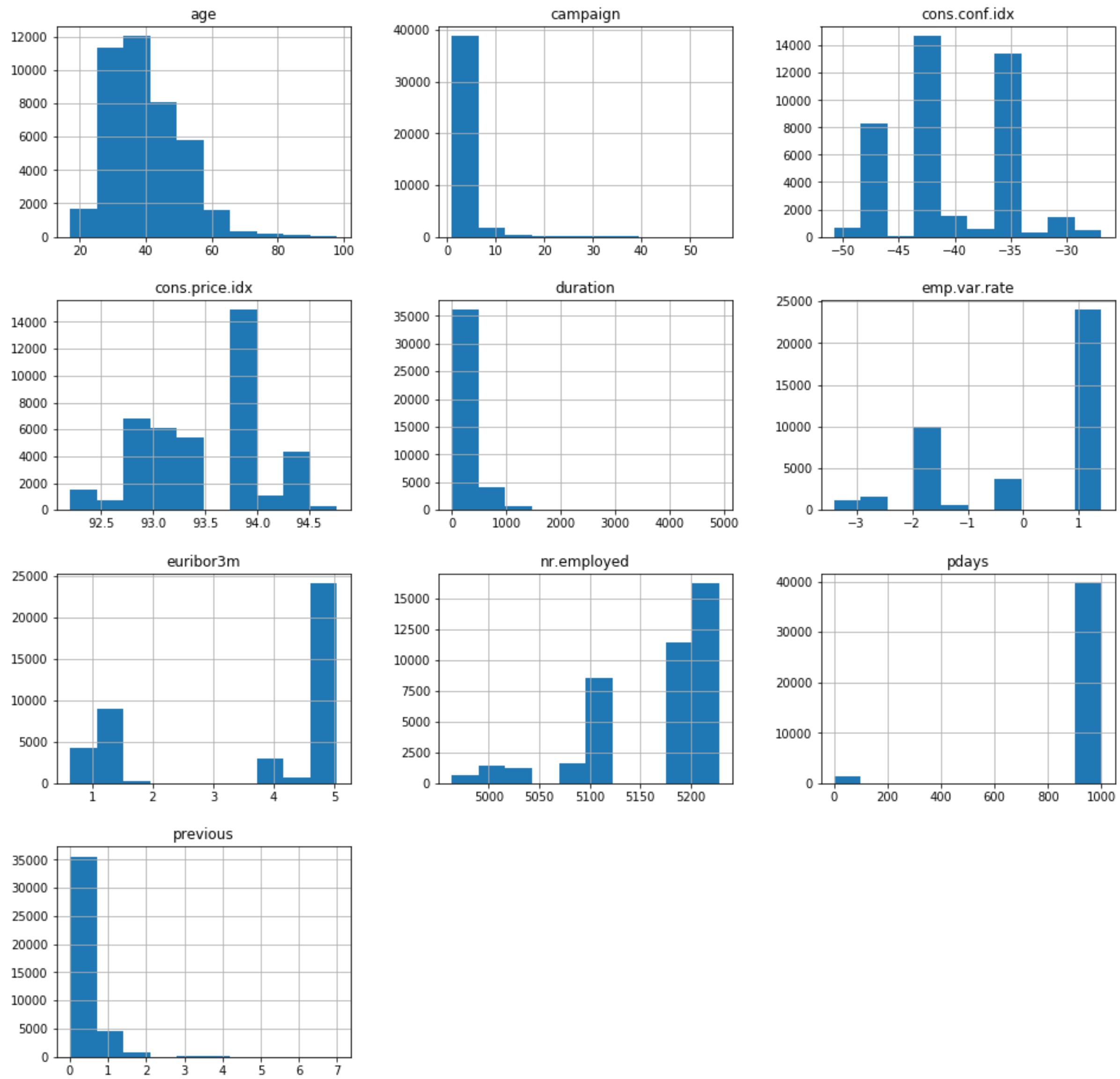
In [30]:
```
#summary statistics for quantitative variables
bdata.describe()
cont = bdata.describe().columns
print(cont)
bdata.describe()
```

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

Out[30]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.
count	41176.00000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	4117
mean	40.02380	258.315815	2.567879	962.464810	0.173013	0.081922	9
std	10.42068	259.305321	2.770318	186.937102	0.494964	1.570883	
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	9.
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	9
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	9
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	9
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	9

In [195]:
```
#Histogram Visualizing Distribution of quantitative variables

cont_histogram = bdata.hist(column=cont, figsize = (16,16))
```



In [7]:
```
#Statistical Summary
bdata.describe()
```

Out[7]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.
count	41176.00000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	4117
mean	40.02380	258.315815	2.567879	962.464810	0.173013	0.081922	9
std	10.42068	259.305321	2.770318	186.937102	0.494964	1.570883	
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	9.
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	9
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	9
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	9
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	9

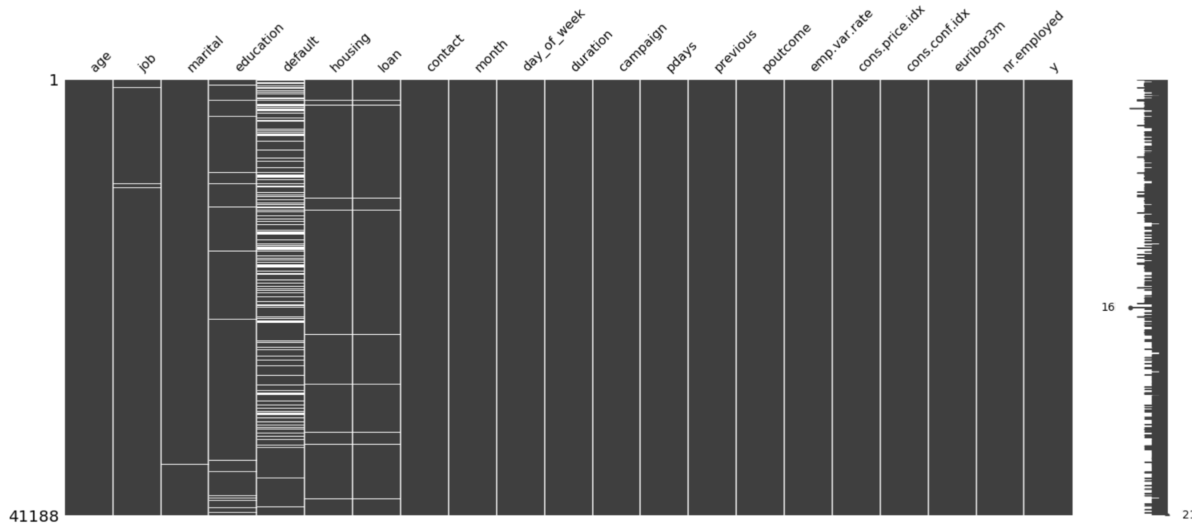
In [32]:
```
#find the unknown values in the dataset
unknown_values = ["unknown", "na", "--"]
bdata = pd.read_csv(r"G:/Yamuna docs/College docs/Info Ret/Project 1/bank-additional-full.csv",  delimiter=";",header='infer', na_values = unknown_values)
print(bdata.isnull().sum())
bdata.isnull().sum().sum()
```

```
age                  0
job                330
marital             80
education         1731
default           8597
housing            990
loan               990
contact              0
month                0
day_of_week          0
duration             0
campaign             0
pdays                0
previous             0
poutcome             0
emp.var.rate         0
cons.price.idx       0
cons.conf.idx        0
euribor3m            0
nr.employed          0
y                    0
dtype: int64
```

Out[32]: 12718

In [292]:
```
import missingno as msno
msno.matrix(bdata)
```

Out[292]: <matplotlib.axes._subplots.AxesSubplot at 0x133d1220828>



In [215]:
```
bdata = bdata[bdata.job != 'unknown']
bdata = bdata[bdata.marital != 'unknown']

bdata = bdata[bdata.education != 'unknown']
bdata = bdata[bdata.education != 'illiterate']

#Dropping the unknown housing loan status
bdata = bdata[bdata.housing != 'unknown']

#Dropping the unknown personal loan status
bdata = bdata[bdata.loan != 'unknown']

#Deleting the 'default' column
del bdata['default']

#Deleting the 'duration' column
#del bdata['duration']
bdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41170 entries, 0 to 41187
Data columns (total 20 columns):
age                41170 non-null int64
job                40840 non-null object
marital            41090 non-null object
education          39439 non-null object
housing            40180 non-null object
loan               40180 non-null object
contact            41170 non-null object
month              41170 non-null object
day_of_week        41170 non-null object
duration           41170 non-null int64
campaign           41170 non-null int64
pdays              41170 non-null int64
previous           41170 non-null int64
poutcome           41170 non-null object
emp.var.rate       41170 non-null float64
cons.price.idx     41170 non-null float64
cons.conf.idx      41170 non-null float64
euribor3m          41170 non-null float64
nr.employed        41170 non-null float64
y                  41170 non-null object
dtypes: float64(5), int64(5), object(10)
memory usage: 6.6+ MB
```

In [94]:
```
#combining spares categories
bdata.job.replace(['entrepreneur', 'self-employed'], 'self-employed', inplace=
True)

bdata.job.replace(['admin.', 'management'], 'administration_management', inpla
ce=True)

bdata.job.replace(['blue-collar', 'technician'], 'blue-collar', inplace=True)

bdata.job.replace(['retired', 'unemployed'], 'no_active_income', inplace=True)

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

bdata.marital.replace(['single', 'divorced'], 'single', inplace=True)

bdata.education.replace(['basic.9y', 'basic.6y', 'basic.4y'], 'basic_school',
inplace=True)
```

In [58]:
```
catanz = bdata.loc[( (bdata['pdays'] == 999) & (bdata['poutcome'] != 'nonexist
ent') )]

#Counting the values
catanz.poutcome.value_counts()
```

Out[58]:
```
failure    4110
Name: poutcome, dtype: int64
```

In [59]:
```
ind_999 = bdata.loc[(bdata['pdays'] == 999) & (bdata['poutcome'] != 'nonexiste
nt')]['pdays'].index.values

#Assigning NaNs instead of '999'
bdata.loc[ind_999, 'pdays'] = np.nan

#Checking if the NaNs were assigned correctly
bdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41176 entries, 0 to 41187
Data columns (total 21 columns):
age               41176 non-null int64
job               41176 non-null object
marital           41176 non-null object
education         41176 non-null object
default           41176 non-null object
housing           41176 non-null object
loan              41176 non-null object
contact           41176 non-null object
month             41176 non-null object
day_of_week       41176 non-null object
duration          41176 non-null int64
campaign          41176 non-null int64
pdays             37066 non-null float64
previous          41176 non-null int64
poutcome          41176 non-null object
emp.var.rate      41176 non-null float64
cons.price.idx    41176 non-null float64
cons.conf.idx     41176 non-null float64
euribor3m         41176 non-null float64
nr.employed       41176 non-null float64
y                 41176 non-null object
dtypes: float64(6), int64(4), object(11)
memory usage: 8.2+ MB
```

In [187]:
```
#Dropping NAs from the dataset
bdata = bdata.dropna()
bdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 0 entries
Data columns (total 20 columns):
age               0 non-null float64
job               0 non-null float64
marital           0 non-null float64
education         0 non-null float64
housing           0 non-null float64
loan              0 non-null float64
contact           0 non-null float64
month             0 non-null float64
day_of_week       0 non-null float64
duration          0 non-null int64
campaign          0 non-null float64
pdays             0 non-null float64
previous          0 non-null float64
poutcome          0 non-null float64
emp.var.rate      0 non-null float64
cons.price.idx    0 non-null float64
cons.conf.idx     0 non-null float64
euribor3m         0 non-null float64
nr.employed       0 non-null float64
y                 0 non-null int32
dtypes: float64(18), int32(1), int64(1)
memory usage: 0.0 bytes
```

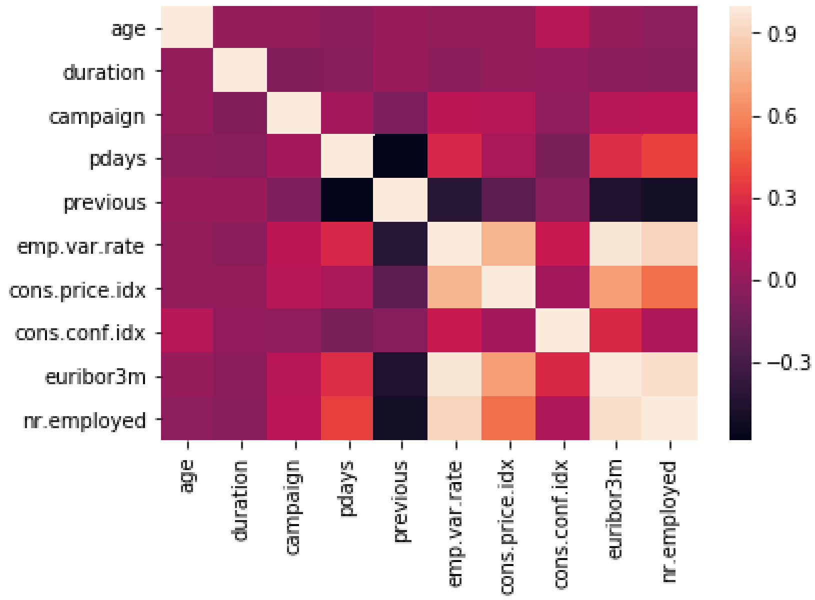
In [34]:
```
#correleation matrix
bdata.corr()
```

Out[34]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx
age	1.000000	-0.000866	0.004594	-0.034369	0.024365	-0.000371	0.000857
duration	-0.000866	1.000000	-0.071699	-0.047577	0.020640	-0.027968	0.005312
campaign	0.004594	-0.071699	1.000000	0.052584	-0.079141	0.150754	0.127836
pdays	-0.034369	-0.047577	0.052584	1.000000	-0.587514	0.271004	0.078889
previous	0.024365	0.020640	-0.079141	-0.587514	1.000000	-0.420489	-0.203130
emp.var.rate	-0.000371	-0.027968	0.150754	0.271004	-0.420489	1.000000	0.775334
cons.price.idx	0.000857	0.005312	0.127836	0.078889	-0.203130	0.775334	1.000000
cons.conf.idx	0.129372	-0.008173	-0.013733	-0.091342	-0.050936	0.196041	0.058986
euribor3m	0.010767	-0.032897	0.135133	0.296899	-0.454494	0.972245	0.688230
nr.employed	-0.017725	-0.044703	0.144095	0.372605	-0.501333	0.906970	0.522034

In [35]:
```
import seaborn as sns
#visualise correlation
sns.heatmap(bdata.corr(), )
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x2990b063080>



In [ ]:
```
From the above analysis, data is non-linear and its an asymmetric type.
The feature selection will not depend on the correlation matrix.
```

In [36]:
```
import pandas as pd
#Class Distribution
bdata['y'].value_counts()
#bdata.groupby('y').count()#.toPandas()
```

Out[36]:
```
no     36548
yes     4640
Name: y, dtype: int64
```

In [767]:
```
#features = bdata[bdata.columns.difference(['y'])]
#labels = bdata['y']
#labels
#bdata.fillna(0, inplace=True)
#bad_chars = [';', ':', '!', "*", '.']
#bdata = ''.join(i for i in bdata if not i in bad_chars)
#bdata=bdata.replace('\.','',regex=True).astype(object)
```

In [95]:
```
#bdata.drop(['duration','contact','month','default','day_of_week','pdays',],ax
is=1,inplace=True)
bdata.drop(['default'],axis=1, inplace=True)
```

In [96]:
```
#converting categorical features into numeric values
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
le = preprocessing.LabelEncoder()
```

In [77]:
```
bdata.job = bdata.job.replace(['<'], '0',inplace=True)
bdata.marital = bdata.marital.replace(['<'], '0',inplace=True)
bdata.education = bdata.education.replace(['<'], '0',inplace=True)
bdata.housing = bdata.housing.replace(['<'], '0',inplace=True)
bdata.loan = bdata.loan.replace(['<'], '0',inplace=True)
bdata.poutcome = bdata.poutcome.replace(['<'], '0',inplace=True)
#bdata.default = bdata.default.replace(['<'], '0',inplace=True)
```

In [97]:
```
bdata.job = le.fit_transform(bdata.job)
bdata.marital = le.fit_transform(bdata.marital)
bdata.education = le.fit_transform(bdata.education)
bdata.housing = le.fit_transform(bdata.housing)
bdata.loan = le.fit_transform(bdata.loan)
bdata.poutcome = le.fit_transform(bdata.poutcome)
bdata.y = le.fit_transform(bdata.y)
bdata.day_of_week =  le.fit_transform(bdata.day_of_week)
#bdata.default =  le.fit_transform(bdata.default)
bdata.contact =  le.fit_transform(bdata.contact)
bdata.month =  le.fit_transform(bdata.month)

bdata.head()
```

Out[97]:

	age	job	marital	education	housing	loan	contact	month	day_of_week	duration	campaig
0	56	4	0	0	0	0	1	6	1	261	
1	57	4	0	1	0	0	1	6	1	149	
2	37	4	0	1	2	0	1	6	1	226	
3	40	0	0	0	0	0	1	6	1	151	
4	56	4	0	1	0	2	1	6	1	307	

In [98]:
```
bdata.shape
```

Out[98]:
```
(41176, 20)
```

In [99]:
```
num_features = ['age', 'job', 'marital', 'education',  'housing', 'loan',
       'contact', 'month', 'day_of_week',  'campaign',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed']

scaled_features = {}
for each in num_features:
    mean, std = bdata[each].mean(), bdata[each].std()
    scaled_features[each] = [mean, std]
    bdata.loc[:, each] = (bdata[each] - mean)/std
print(pd.DataFrame(scaled_features))
```

```
        age       job   marital  education   housing      loan   contact  \
0  40.02380  1.373373  0.396712   2.005391  1.071789  0.327521  0.365286
1  10.42068  1.472977  0.493177   1.770190  0.985305  0.723700  0.481516

      month  day_of_week  campaign  previous  poutcome  emp.var.rate  \
0  4.231033     2.004614  2.567879  0.173013  0.930081      0.081922
1  2.319973     1.397692  2.770318  0.494964  0.362937      1.570883

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed
0       93.575720     -40.502863   3.621293  5167.034870
1        0.578839       4.627860   1.734437    72.251364
```

In [100]:
```
X = bdata.iloc[:,0:19]
X.columns
```

Out[100]:
```
Index(['age', 'job', 'marital', 'education', 'housing', 'loan', 'contact',
       'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'previous',
       'poutcome', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
       'euribor3m', 'nr.employed'],
      dtype='object')
```

In [101]:
```
y = bdata.iloc[:,19]
```

In [102]:
```
y.value_counts()
```

Out[102]:
```
0    36537
1     4639
Name: y, dtype: int64
```

In [103]:
```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
x_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test
_size=0.2, random_state=0)
```

In [104]:
```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [105]:
```
x_train.shape, y_train.shape
```

Out[105]:
```
((32940, 19), (32940,))
```

In [106]:
```
x_test.shape, y_test.shape
```

Out[106]:
```
((8236, 19), (8236,))
```

In [107]:
```
model=LogisticRegression(penalty='l2', max_iter=10)
modellr=model.fit(x_train, y_train)
modellr
```

```
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
```

Out[107]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [108]:
```
prediction=model.predict(x_test) #predict
```

In [109]:
```
from sklearn.metrics import accuracy_score
print('accuracy_score',accuracy_score(y_test, prediction))
```

```
accuracy_score 0.9091792132102963
```

In [110]:
```
accuracy = cross_val_score(modellr, x_train, y_train, scoring='accuracy', cv =
k_fold, n_jobs=1)
print('Accuracy of each fold:',accuracy)
#get the mean of each fold
Logic = accuracy.mean() * 100
print("Accuracy of Model with Cross Validation:",Logic)
```

```
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\sklearn\linear_model\log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)

Accuracy of each fold: [0.91044323 0.9058895  0.91044323 0.90892532 0.9119611
4 0.91469338
 0.9058895  0.91165756 0.91621129 0.91105039]
Accuracy of Model with Cross Validation: 91.07164541590771
```

In [111]:
```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix = confusion_matrix(y_test, prediction)
print(confusion_matrix)
```

```
[[7102  167]
 [ 581  386]]
```

In [112]:
```
print(classification_report(y_test,prediction))
```

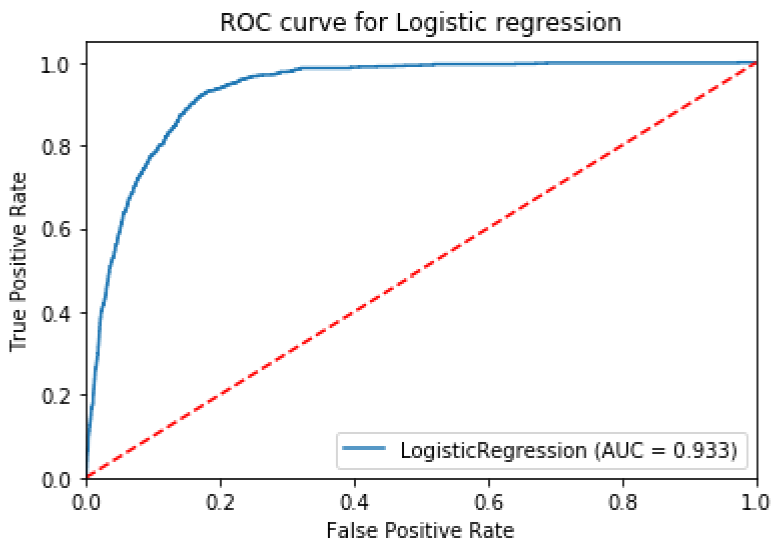
```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      7269
           1       0.70      0.40      0.51       967

    accuracy                           0.91      8236
   macro avg       0.81      0.69      0.73      8236
weighted avg       0.90      0.91      0.90      8236
```

In [113]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

from sklearn import metrics
y_pred_probal = model.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_probal)
auc = metrics.roc_auc_score(y_test, y_pred_probal)
resultl = roc_auc_score(y_test, prediction)
print('AUC',auc)
```

```
AUC 0.9334926988758058
```

In [114]:
```
#Obtaining the ROC score
roc_auc = roc_auc_score(y_test, y_pred_probal)
print(roc_auc)
#Obtaining false and true positives & thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probal)
plt.plot(fpr, tpr, label='LogisticRegression (AUC = %0.03f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for Logistic regression')
plt.legend(loc="lower right")
plt.show()
```

```
0.9334926988758058
```



In [115]:
```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion="gini", max_depth=7)
model = tree.fit(x_train,y_train)
model
```

Out[115]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
                       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=False,
                       random_state=None, splitter='best')
```

In [116]:
```
dec_pred=tree.predict(x_test)
print('Accuracy_score', accuracy_score(y_test, dec_pred))
```

```
Accuracy_score 0.9082078678970374
```

In [117]:
```
#mat = confusion_matrix([y_test],[dec_pred],labels=['no','yes'])
#print(mat)
#y_test = label_binarize(y_test,classes=['no','yes'])
#y_pred = label_binarize(y_pred,classes=['no','yes'])
#print("Precision: ",round(precision_score(y_test,dec_pred),2),"Recall: ",roun
d(recall_score(y_test,dec_pred),2))
#confusion_matrix = pd.crosstab(['y_test'], bdata['dec_pred'], rownames=['Actu
al'], colnames=['Predicted'])
#print (confusion_matrix)
```

In [118]:
```
accuracy = cross_val_score(model, x_train, y_train, scoring='accuracy', cv = k
_fold, n_jobs=1)
print('Accuracy of each fold:',accuracy)
#get the mean of each fold
Decision = accuracy.mean()*100
print("Accuracy of Model with Cross Validation:",accuracy.mean() * 100)
```

```
Accuracy of each fold: [0.91256831 0.91256831 0.90801457 0.91256831 0.9101396
5 0.91378264
 0.91621129 0.91256831 0.91833637 0.91469338]
Accuracy of Model with Cross Validation: 91.31451123254402
```

In [119]:
```
print(classification_report(y_test,dec_pred))
```

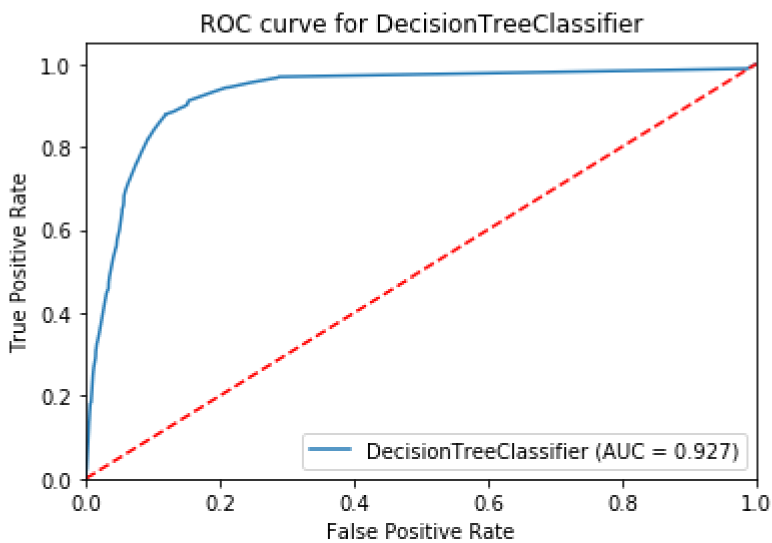
```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      7269
           1       0.65      0.48      0.55       967

    accuracy                           0.91      8236
   macro avg       0.79      0.72      0.75      8236
weighted avg       0.90      0.91      0.90      8236
```

In [120]:
```
from sklearn import metrics
dec_pred_proba = model.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  dec_pred_proba)
auc = metrics.roc_auc_score(y_test, dec_pred_proba)
resultdec = roc_auc_score(y_test, dec_pred_proba)
print(auc)
```

```
0.9270856406979932
```

In [121]:
```
#Obtaining the ROC score
roc_auc = roc_auc_score(y_test, dec_pred_proba)
print(roc_auc)
#Obtaining false and true positives & thresholds
fpr, tpr, thresholds = roc_curve(y_test, dec_pred_proba)
plt.plot(fpr, tpr, label='DecisionTreeClassifier (AUC = %0.03f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for DecisionTreeClassifier')
plt.legend(loc='lower right')
plt.show()
```

```
0.9270856406979932
```



In [122]:
```
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(x_train,y_train)

rd_pred=clf.predict(x_test)
```

In [123]:
```
print("Accuracy:",metrics.accuracy_score(y_test, rd_pred))
```

```
Accuracy: 0.9116075764934434
```

In [124]:
```
accuracy = cross_val_score(clf, x_train, y_train, scoring='accuracy', cv = k_f
old, n_jobs=1)
print('Accuracy of each fold:',accuracy)
#get the mean of each fold
Random = accuracy.mean()*100
print("Accuracy of Model with Cross Validation:",Random)
```

```
Accuracy of each fold: [0.91560413 0.91621129 0.91165756 0.91044323 0.9162112
9 0.91347905
 0.91499696 0.9177292  0.92380085 0.91530055]
Accuracy of Model with Cross Validation: 91.55434122647237
```

In [125]:
```
print(classification_report(y_test,rd_pred))
```

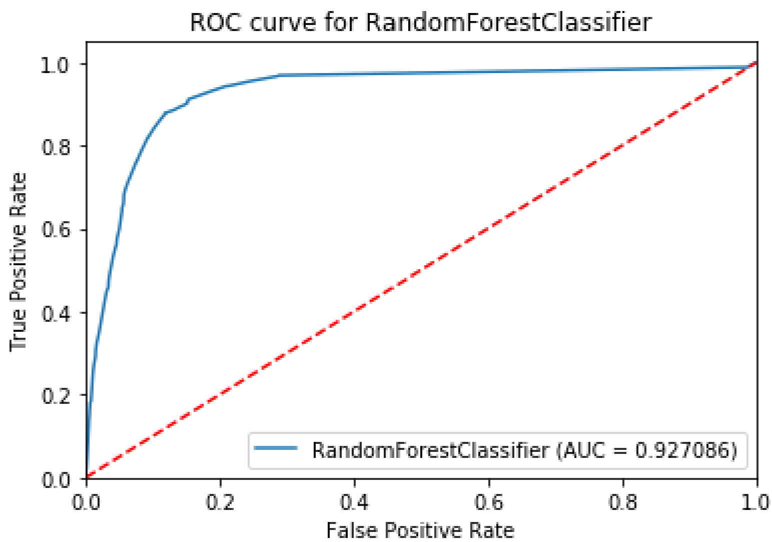
```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      7269
           1       0.67      0.49      0.57       967

    accuracy                           0.91      8236
   macro avg       0.80      0.73      0.76      8236
weighted avg       0.90      0.91      0.91      8236
```

In [126]:
```
from sklearn import metrics
rd_pred_proba = model.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  rd_pred_proba)
auc = metrics.roc_auc_score(y_test, rd_pred_proba)
resultrd = roc_auc_score(y_test, rd_pred)
print(auc)
```

```
0.9270856406979932
```

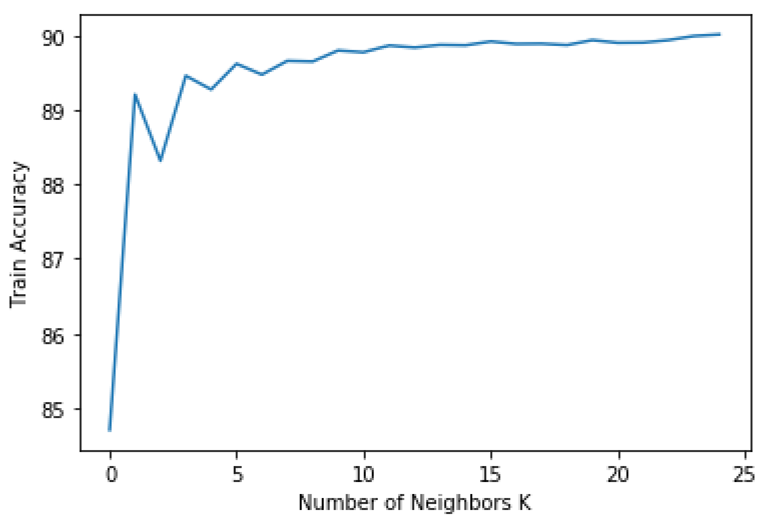
In [127]:
```
#Obtaining the ROC score
roc_auc = roc_auc_score(y_test, rd_pred_proba)
print(roc_auc)
#Obtaining false and true positives & thresholds
fpr, tpr, thresholds = roc_curve(y_test, rd_pred_proba)
plt.plot(fpr, tpr, label='RandomForestClassifier (AUC = %f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for RandomForestClassifier')
plt.legend(loc='lower right')
plt.show()
```

```
0.9270856406979932
```



In [49]:
```
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier

#Neighbors
neighbors = np.arange(0,25)

#Create empty list that will hold cv scores
cv_scores = []

#Perform 10-fold cross validation on training set for odd values of k:
for k in neighbors:
    k_value = k+1
    knn = KNeighborsClassifier(n_neighbors = k_value, weights='uniform', p=2,
metric='euclidean')
    kfold = model_selection.KFold(n_splits=10, random_state=123)
    scores = model_selection.cross_val_score(knn, x_train, y_train, cv=kfold,
scoring='accuracy')
    cv_scores.append(scores.mean()*100)
    print("k=%d %0.2f (+/- %0.2f)" % (k_value, scores.mean()*100, scores.std()
*100))

optimal_k = neighbors[cv_scores.index(max(cv_scores))]
print ("The optimal number of neighbors is %d with %0.1f%%" % (optimal_k, cv_s
cores[optimal_k]))

plt.plot(neighbors, cv_scores)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Train Accuracy')
plt.show()
```

```
k=1 84.70 (+/- 0.45)
k=2 89.21 (+/- 0.42)
k=3 88.32 (+/- 0.41)
k=4 89.46 (+/- 0.40)
k=5 89.28 (+/- 0.37)
k=6 89.62 (+/- 0.27)
k=7 89.48 (+/- 0.38)
k=8 89.66 (+/- 0.34)
k=9 89.65 (+/- 0.43)
k=10 89.80 (+/- 0.39)
k=11 89.78 (+/- 0.46)
k=12 89.87 (+/- 0.43)
k=13 89.84 (+/- 0.38)
k=14 89.88 (+/- 0.34)
k=15 89.87 (+/- 0.33)
k=16 89.92 (+/- 0.32)
k=17 89.89 (+/- 0.39)
k=18 89.89 (+/- 0.35)
k=19 89.88 (+/- 0.38)
k=20 89.94 (+/- 0.31)
k=21 89.91 (+/- 0.40)
k=22 89.91 (+/- 0.34)
k=23 89.94 (+/- 0.35)
k=24 90.00 (+/- 0.35)
k=25 90.02 (+/- 0.39)
The optimal number of neighbors is 24 with 90.0%
```



In [128]:

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=22)
knn.fit(x_train, y_train)
knnpred = knn.predict(x_test)

#print(confusion_matrix(y_test, knnpred))
print("Accuracy:",metrics.accuracy_score(y_test, knnpred))
```

```
Accuracy: 0.9008013598834386
```

In [129]:
```
accuracy = cross_val_score(knn, x_train, y_train, scoring='accuracy', cv = k_f
old, n_jobs=1)
print('Accuracy of each fold:',accuracy)
#get the mean of each fold
KNN = accuracy.mean()*100
print("Accuracy of Model with Cross Validation:",KNN)
```

```
Accuracy of each fold: [0.90315726 0.89678203 0.90680024 0.89860352 0.9013357
6 0.91105039
 0.90497875 0.90680024 0.9143898  0.9058895 ]
Accuracy of Model with Cross Validation: 90.49787492410442
```

In [130]:
```
#confusion_matrix(y_test, knnpred)
print(classification_report(y_test, knnpred))
```

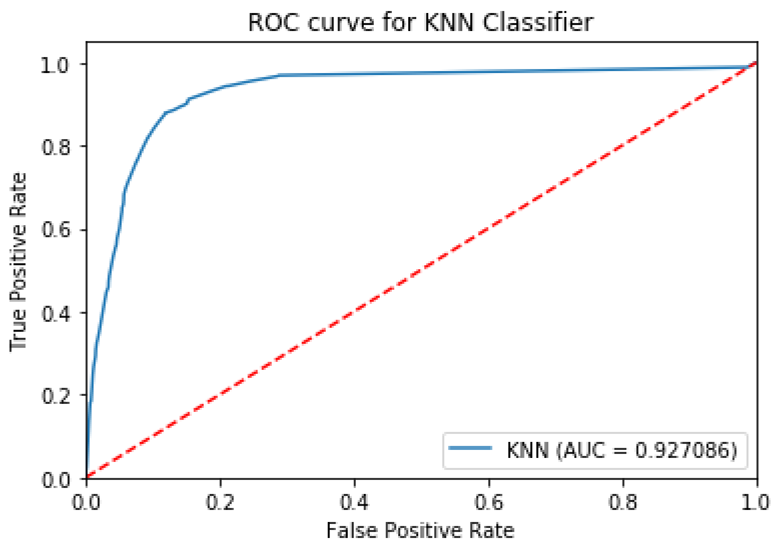
```
              precision    recall  f1-score   support

           0       0.91      0.98      0.95      7269
           1       0.69      0.29      0.40       967

    accuracy                           0.90      8236
   macro avg       0.80      0.63      0.67      8236
weighted avg       0.89      0.90      0.88      8236
```

In [131]:
```
knn_pred_proba = model.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  knn_pred_proba)
auc = metrics.roc_auc_score(y_test, knn_pred_proba)
resultknn = roc_auc_score(y_test, knnpred)
print(auc)
```

```
0.9270856406979932
```

In [132]:
```
#Obtaining the ROC score
roc_auc = roc_auc_score(y_test, knn_pred_proba)
print(roc_auc)
#Obtaining false and true positives & thresholds
fpr, tpr, thresholds = roc_curve(y_test, knn_pred_proba)
plt.plot(fpr, tpr, label='KNN (AUC = %f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for KNN Classifier')
plt.legend(loc='lower right')
plt.show()
```

```
0.9270856406979932
```



In [133]:
```
from sklearn.naive_bayes import GaussianNB
gaussiannb= GaussianNB()
gaussiannb.fit(x_train, y_train)
gaussiannbpred = gaussiannb.predict(x_test)
#gauss = gaussiannb.predict(x_test)

print(accuracy_score(y_test, gaussiannbpred ))
#print('accuracy_score',round(accuracy_score(y_test, gaussiannbpred),2)*100)
#GAUSIAN = (cross_val_score(gaussiannb, x_train, y_train, cv=k_fold, n_jobs=1,
scoring = 'accuracy').mean())
```

```
0.8453132588635259
```

In [134]:
```
accuracy = cross_val_score(gaussiannb, x_train, y_train, scoring='accuracy', c
v = k_fold, n_jobs=1)
print('Accuracy of each fold:',accuracy)
#get the mean of each fold
GAUSIAN = accuracy.mean()*100
print("Accuracy of Model with Cross Validation:",accuracy.mean() * 100)
```

```
Accuracy of each fold: [0.83333333 0.84031573 0.83849423 0.83697632 0.8503339
4 0.84669095
 0.83758349 0.84092289 0.83515483 0.83151184]
Accuracy of Model with Cross Validation: 83.91317547055252
```

In [135]:
```
print(classification_report(y_test, gaussiannbpred))
```

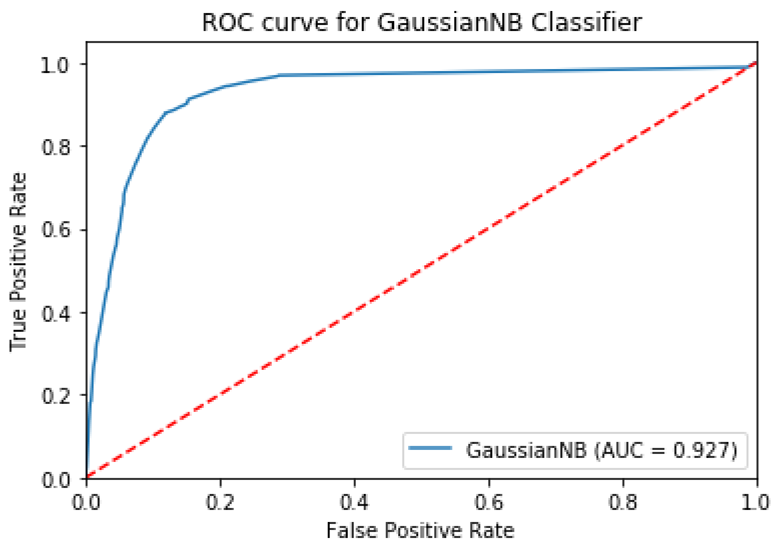
```
              precision    recall  f1-score   support

           0       0.94      0.88      0.91      7269
           1       0.39      0.59      0.47       967

    accuracy                           0.85      8236
   macro avg       0.67      0.74      0.69      8236
weighted avg       0.88      0.85      0.86      8236
```

In [136]:
```
gauss_pred_proba = model.predict_proba(x_test)[::,1]
fpr1, tpr1, _ = metrics.roc_curve(y_test,  gauss_pred_proba)
auc = metrics.roc_auc_score(y_test, gauss_pred_proba)
resultgauss = roc_auc_score(y_test, gaussiannbpred)
print(auc)
```

```
0.9270856406979932
```

In [137]:
```
#Obtaining the ROC score
roc_auc = roc_auc_score(y_test, gauss_pred_proba)
print('AUC',roc_auc)
#Obtaining false and true positives & thresholds
fpr1, tpr1, thresholds = roc_curve(y_test, gauss_pred_proba)
plt.plot(fpr1, tpr1, label='GaussianNB (AUC = %0.03f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for GaussianNB Classifier')
plt.legend(loc='lower right')
plt.show()
```

```
AUC 0.9270856406979932
```



In [138]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbk = GradientBoostingClassifier()
gbk.fit(x_train, y_train)
gbkpred = gbk.predict(x_test)
#print(confusion_matrix(y_test, gbkpred ))
print('accuracy_score',(accuracy_score(y_test, gbkpred)))
#GradientBoosting = (cross_val_score(gbk, x_train, y_train, cv=k_fold, n_jobs=
1, scoring = 'accuracy').mean()))
```

```
accuracy_score 0.91452161243322
```

In [139]:
```
accuracy = cross_val_score(gbk, x_train, y_train, scoring='accuracy', cv = k_f
old, n_jobs=1)
print('Accuracy of each fold:',accuracy)
GradientBoosting = accuracy.mean() * 100
#get the mean of each fold
print("Accuracy of Model with Cross Validation:",GradientBoosting)
```

```
Accuracy of each fold: [0.92076503 0.91590771 0.91317547 0.91742562 0.9198542
8 0.91863995
 0.91317547 0.91712204 0.92380085 0.91530055]
Accuracy of Model with Cross Validation: 91.75166970248938
```

In [140]:
```
print(classification_report(y_test, gbkpred))
```

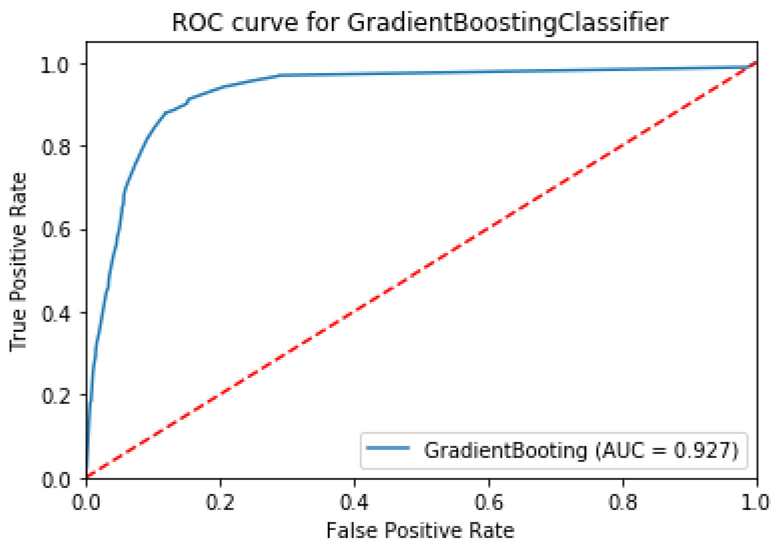
```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      7269
           1       0.68      0.52      0.59       967

    accuracy                           0.91      8236
   macro avg       0.81      0.75      0.77      8236
weighted avg       0.91      0.91      0.91      8236
```

In [141]:
```
gbk_pred_proba = model.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  gbk_pred_proba)
auc = metrics.roc_auc_score(y_test, gbk_pred_proba)
resultgbk = roc_auc_score(y_test, gbkpred)
print(auc)
```

```
0.9270856406979932
```

In [142]:
```
#Obtaining the ROC score
roc_auc = roc_auc_score(y_test, gbk_pred_proba)
print('AUC',roc_auc)
#Obtaining false and true positives & thresholds
fpr, tpr, thresholds = roc_curve(y_test, gbk_pred_proba)
plt.plot(fpr, tpr, label='GradientBooting (AUC = %0.03f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for GradientBoostingClassifier')
plt.legend(loc='lower right')
plt.show()
```

```
AUC 0.9270856406979932
```



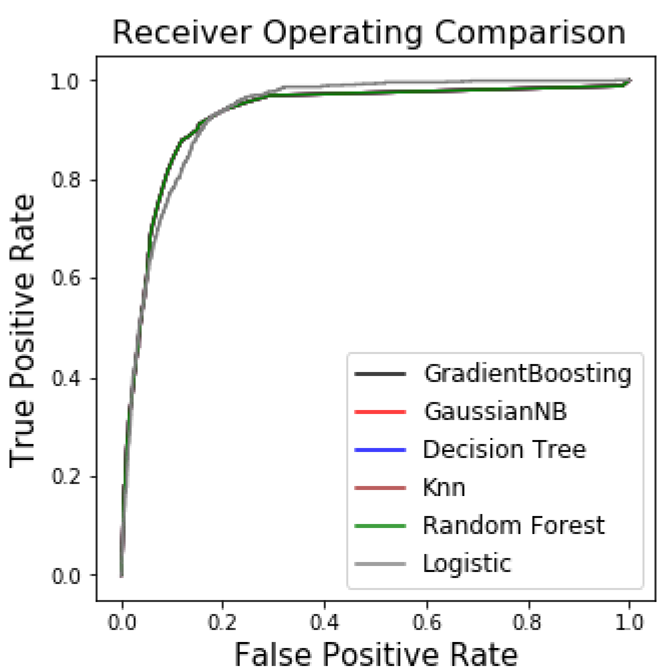
In [144]:
```
models = pd.DataFrame({
                'Models': ['Logistic Model','Decision Tree Classifier','Random
Forest Classifier','K-Near Neighbors',  'Gausian NB', 'Gradient Boosting'],
                'Accuracy':  [Logic, Decision, Random, KNN, GAUSIAN, GradientB
oosting],
                'AUC' : [resultl, resultdec, resultrd, resultknn, resultgauss,
resultgbk]    })

models.sort_values(by='Accuracy',ascending=False)
```

Out[144]:

	Models	Accuracy	AUC
5	Gradient Boosting	91.751670	0.745367
2	Random Forest Classifier	91.554341	0.729820
1	Decision Tree Classifier	91.314511	0.927086
0	Logistic Model	91.071645	0.688099
3	K-Near Neighbors	90.497875	0.634491
4	Gausian NB	83.913175	0.736643

In [148]:
```
#fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(nrows = 2, ncols = 3, figsize =
(15, 4))
fig, ax_arr = plt.subplots( nrows = 1, ncols = 1,figsize = (5,5))

fpr1, tpr1, thresholds = roc_curve(y_test, gbk_pred_proba)
ax_arr.plot(fpr1, tpr1, 'b', label = 'GradientBoosting', color='black')
fpr2, tpr2, thresholds = roc_curve(y_test, gauss_pred_proba)
#fpr1, tpr1, thresholds = roc_curve(y_test, gauss_pred_proba)
#plt.plot(fpr, tpr, label='GaussianNB (AUC = %0.03f)' % roc_auc)
ax_arr.plot(fpr2, tpr2,'b', label = 'GaussianNB', color='red')
fpr3, tpr3, thresholds = roc_curve(y_test, knn_pred_proba)
#plt.plot(fpr, tpr, label='KNN (AUC = %0.03f)' % roc_auc)
ax_arr.plot(fpr3, tpr3, 'b', label = 'Decision Tree', color='blue')
fpr4, tpr4, thresholds = roc_curve(y_test, rd_pred_proba)
#plt.plot(fpr, tpr, label='RandomForestClassifier (AUC = %0.03f)' % roc_auc)
ax_arr.plot(fpr4, tpr4, 'b', label = 'Knn', color='brown')
fpr5, tpr5, thresholds = roc_curve(y_test, dec_pred_proba)
#plt.plot(fpr, tpr, label='DecisionTreeClassifier (AUC = %0.03f)' % roc_auc)
ax_arr.plot(fpr5, tpr5, 'b', label = 'Random Forest', color='green')
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probal)
#plt.plot(fpr, tpr, label='LogisticRegression (AUC = %0.03f)' % roc_auc)
ax_arr.plot(fpr, tpr, 'b', label = 'Logistic', color='grey')
ax_arr.set_title('Receiver Operating Comparison ',fontsize=16)
ax_arr.set_ylabel('True Positive Rate',fontsize=15)
ax_arr.set_xlabel('False Positive Rate',fontsize=15)
ax_arr.legend(loc = 'lower right', prop={'size': 12})

#plt.subplots_adjust(wspace=0.2)
#plt.tight_layout()
```

Out[148]: <matplotlib.legend.Legend at 0x2afe4abb518>



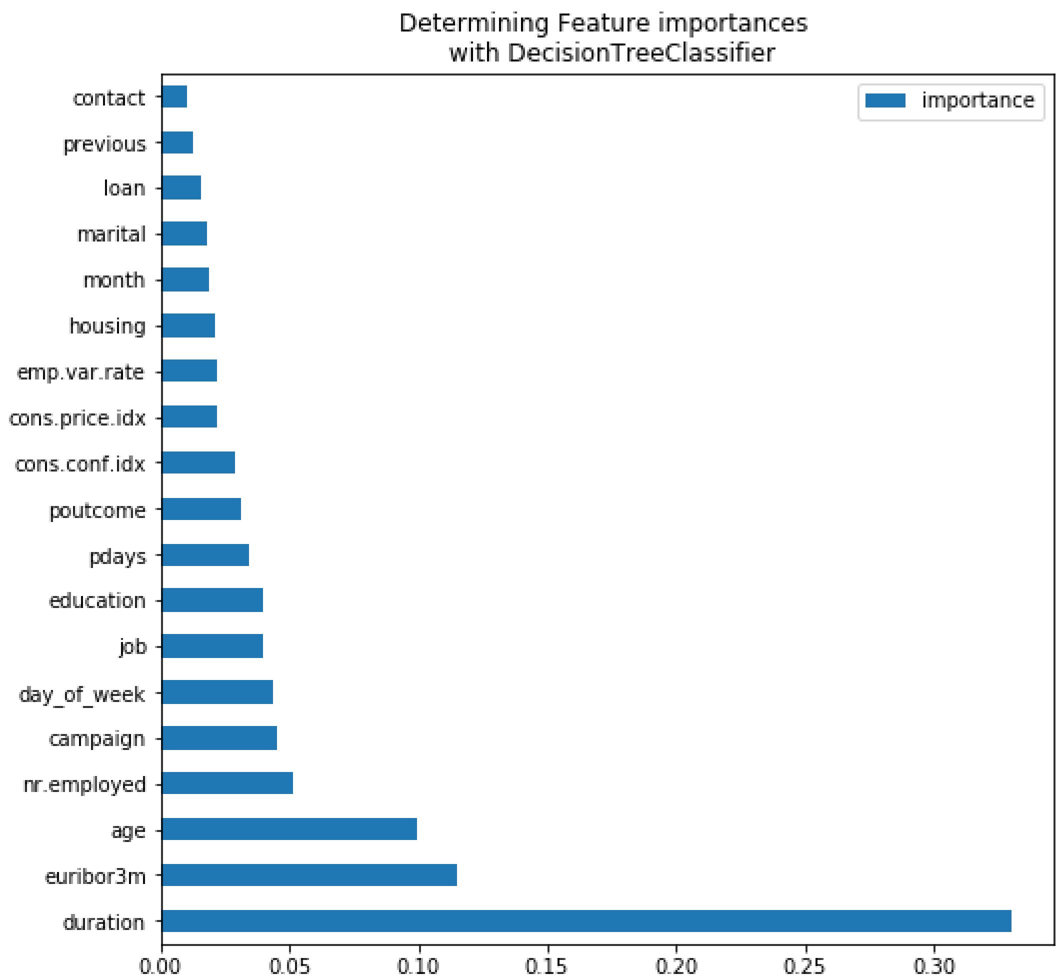
In [146]:
```
#df = pd.read_csv(r"C:\Users\bhara\Desktop\PR\FinalData\full_le.csv")
#X = df.drop('y', axis=1).values
#y = df['y'].values
#pp=df.drop('y', axis=1)
#x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, ra
ndom_state=42)
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(x_train, y_train)
feature_importances = pd.DataFrame(rfc.feature_importances_,index = X.columns,
columns=['importance']).sort_values('importance',ascending=False)
feature_importances
```

Out[146]:

	importance
duration	0.330269
euribor3m	0.114898
age	0.099301
nr.employed	0.051417
campaign	0.044899
day_of_week	0.043404
job	0.039969
education	0.039507
pdays	0.034222
poutcome	0.031343
cons.conf.idx	0.029107
cons.price.idx	0.022261
emp.var.rate	0.022013
housing	0.021167
month	0.019161
marital	0.018103
loan	0.015799
previous	0.012909
contact	0.010250

In [147]:
```
ax = feature_importances.plot.barh(rot=0, figsize = (8,8))
plt.title("Determining Feature importances \n with DecisionTreeClassifier", fo
ntsize=12)
```

Out[147]:
```
Text(0.5, 1.0, 'Determining Feature importances \n with DecisionTreeClassifie
r')
```



In [ ]:

In [ ]: