BANK MARKETING DATASET

Loading the data

->Now we are importing the data.

->By using pandas Library we can read a particular file that we want.

In [ ]:
```
# Run this cell to mount your Google Drive.
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
import pandas as pd
data = pd.read_csv('/content/drive/MyDrive/AAIC/bank-additional-full.csv',sep=
';')
```

->We can see shape of the data,it will be in rows and columns.

In [ ]:
```
data.shape
```

Out[ ]:
```
(41188, 21)
```

->By using head() method we can see first five rows in a table.

In [ ]:
```
data.head(5)
```

Out[ ]:

	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

->By using the columns method we can see how many columns present in table.

In [ ]:
```
data.columns
```

Out[ ]:
```
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')
```

->The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series.

In [ ]:
```
data.describe()
```

Out[ ]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	4118
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	9
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	
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

->The info method is used to show us information about a dataframe including with index dtype and column dtype,Null values,Memory usage.

In [ ]:
```
data.info()
```

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

-> We don't have Null values but we have 'duration', 'contact', 'month', 'day_of_week', 'default', 'pdays' these features are not so useful so its better to drop these features.

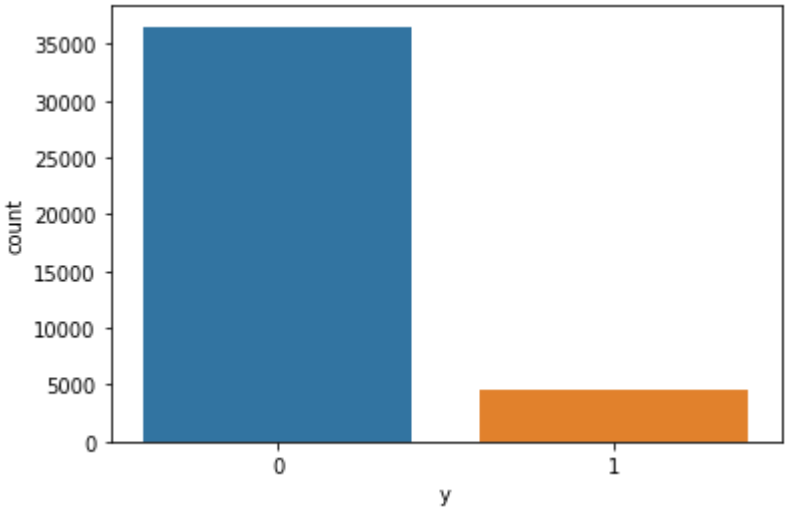
Data Visualization

It shows the class distribution to see whether data is balanced or not.

In [ ]:
```
import seaborn as sns
print(data.y.value_counts())
sns.countplot(x='y', data=data)
```

```
0    36548
1     4640
Name: y, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f492354d748>



-> we can see it is highly imbalanced dataset, so we should use AUC as metric it will perform good when data is imbalanced.

Univariate Analysis of Categorical Variables:-

->We have Job, Marital, Housing, Poutcome, Education, Loan, Contact, Duration, Month, Default has Categorical variables.

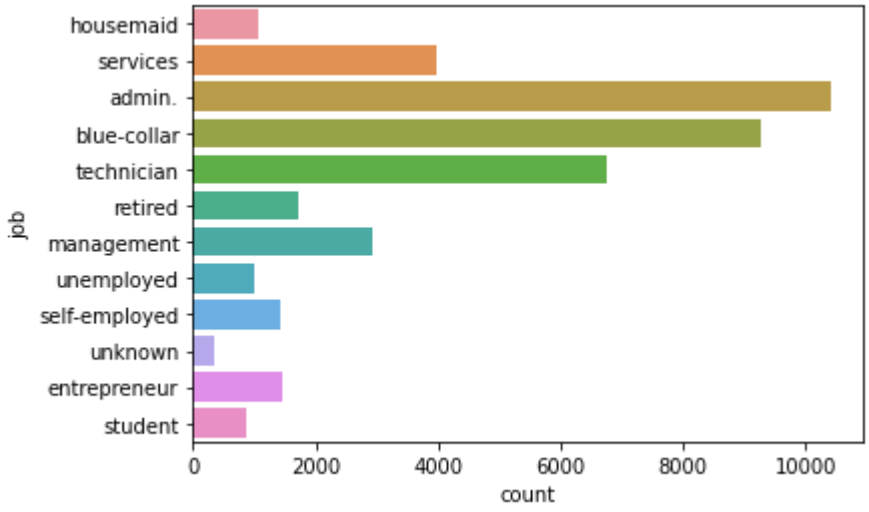
->Plots such as pdf,cdf, Box-plot, voilin plots comes use univariate.

Job:-

In [ ]:
```
print(data.job.value_counts())
sns.countplot(y='job', data=data)
```

```
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: job, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf559b7278>



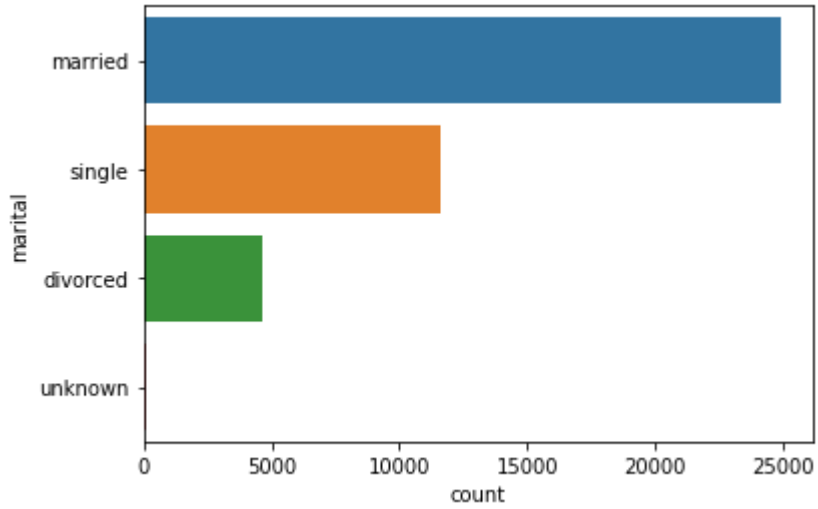
1.We can see services, admin, blue-collar, technician job holders subscribed more than others.

Marital:-

In [ ]:
```
print(data.marital.value_counts())
sns.countplot(y='marital', data=data)
```

```
married     24928
single      11568
divorced     4612
unknown        80
Name: marital, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf554fbef0>



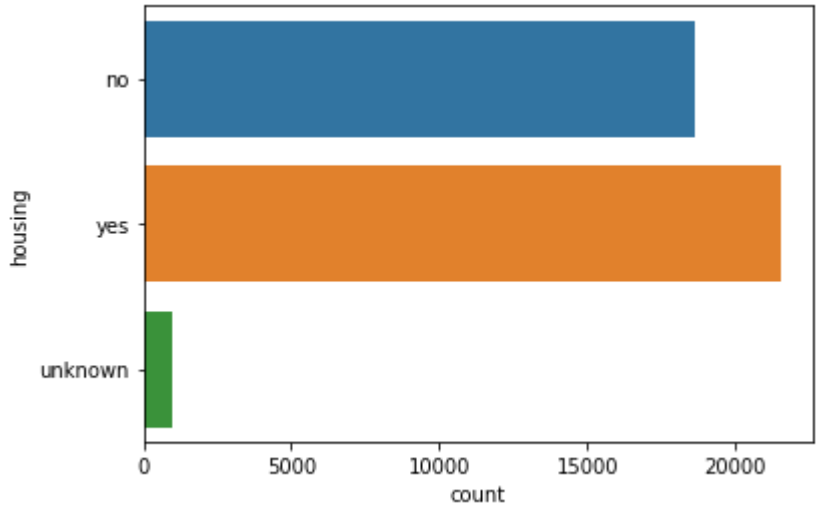
1.We can see married people suscribed more than others.

Housing:-

In [ ]:
```
print(data.housing.value_counts())
sns.countplot(y='housing', data=data)
```

```
yes        21576
no         18622
unknown      990
Name: housing, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf55462f60>



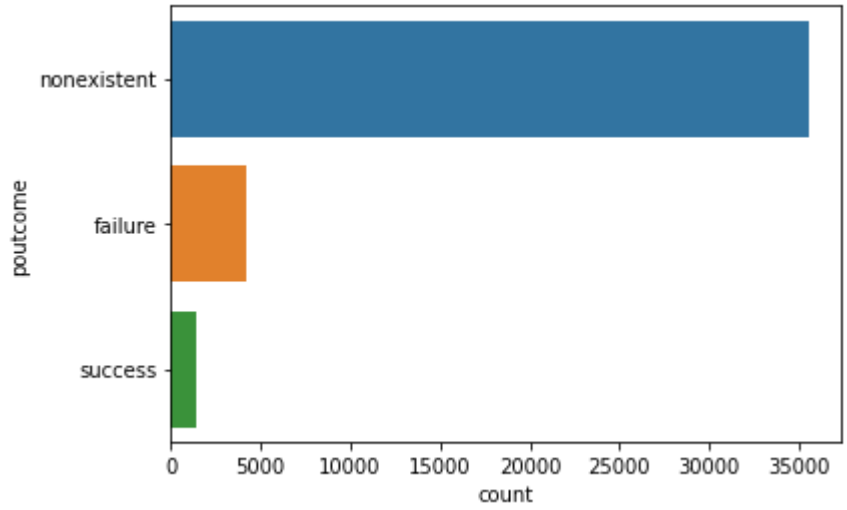
1.We can see Majority of people have a housing loan.

Poutcome:-

In [ ]:
```
print(data.poutcome.value_counts())
sns.countplot(y='poutcome', data=data)
```

```
nonexistent    35563
failure         4252
success         1373
Name: poutcome, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf55439320>



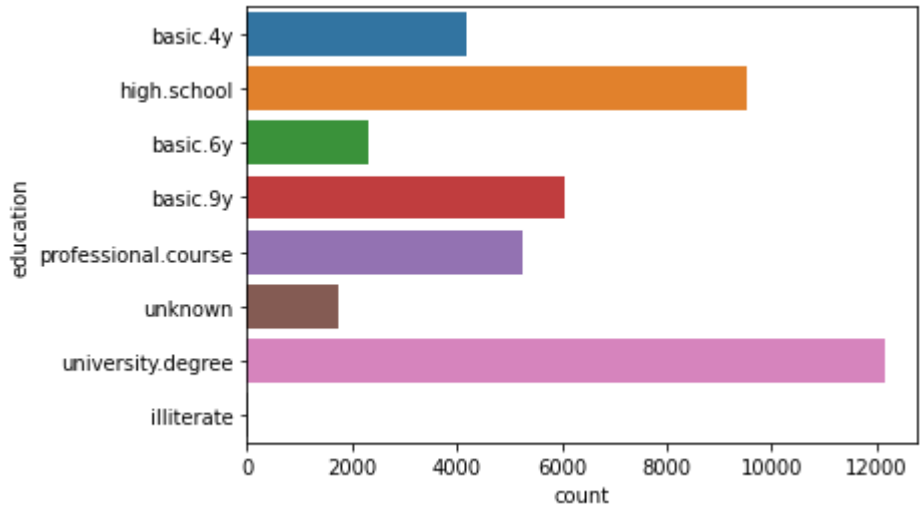
1.It means the previous marketing campaign does not exist.

Education:-

In [ ]:
```
print(data.education.value_counts())
sns.countplot(y='education', data=data)
```

```
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: education, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf55365b38>



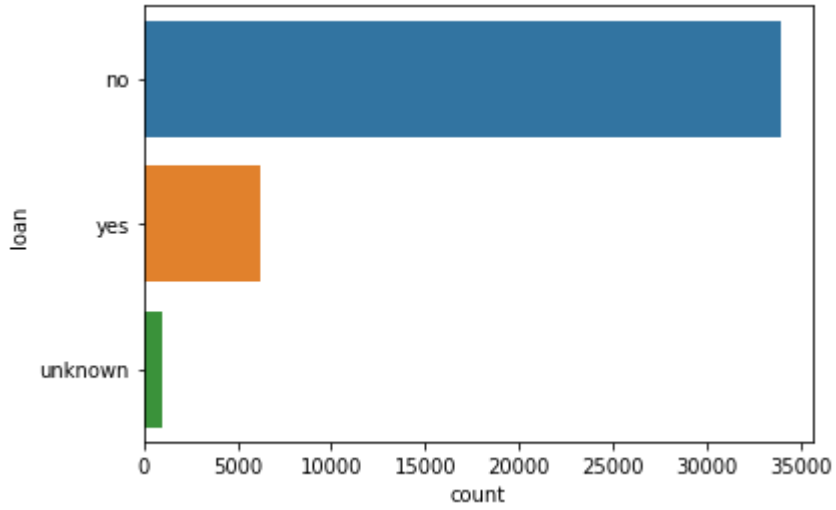
1.We can see people of high.school and university.degree subscribed more than others.

Loan:-

In [ ]:
```
print(data.loan.value_counts())
sns.countplot(y='loan', data=data)
```

```
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf5535b2e8>



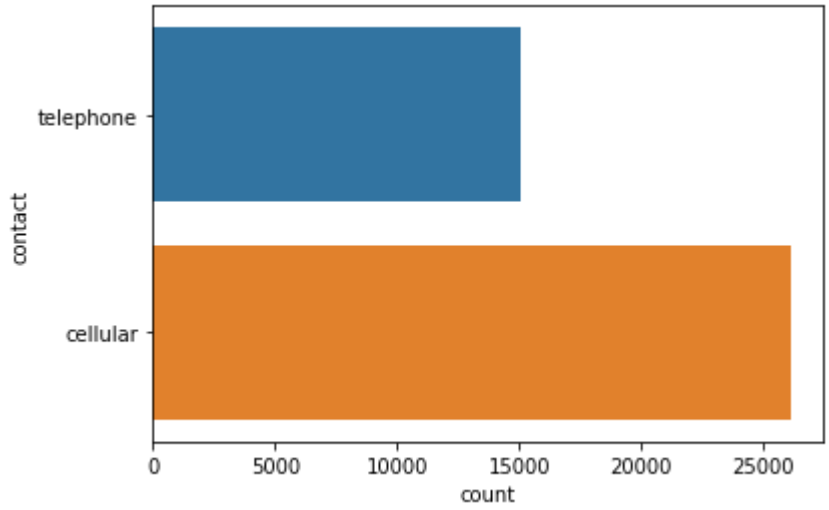
1.We can see that Majority of people didn't have personal loan.

Contact:-

In [ ]:
```
print(data.loan.value_counts())
sns.countplot(y='contact', data=data)
```

```
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf5543d1d0>

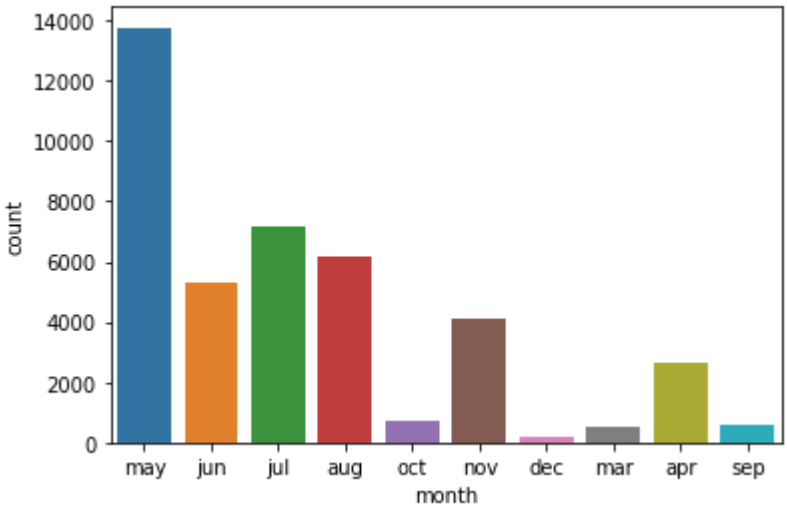


1.Here,if we can see cellular(mobile phone) type customers subscribed more but generally we can't predict on contact type base.

Month:-

In [ ]:
```
print(data.loan.value_counts())
ax = sns.countplot(x="month", data=data)
```

```
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
```



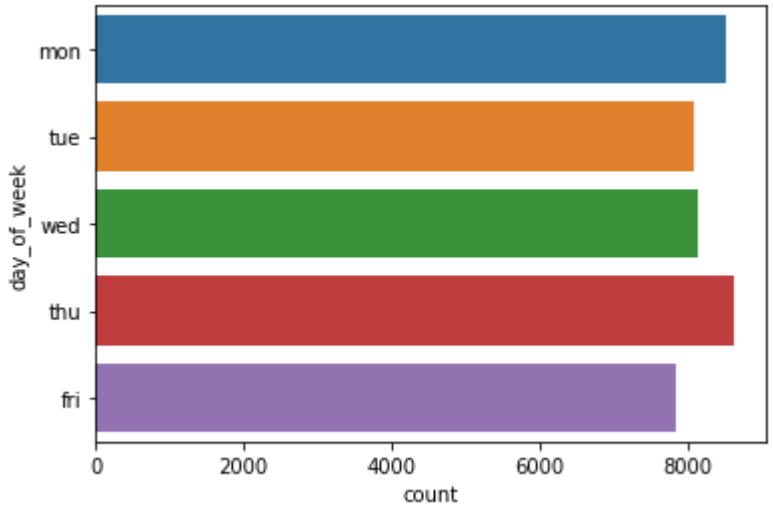
1.Here,if we can see in May month they subscribe more but we can't predict only on month base.

Day_of_week:-

In [ ]:
```
print(data.loan.value_counts())
sns.countplot(y='day_of_week', data=data)
```

```
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf551f2a90>



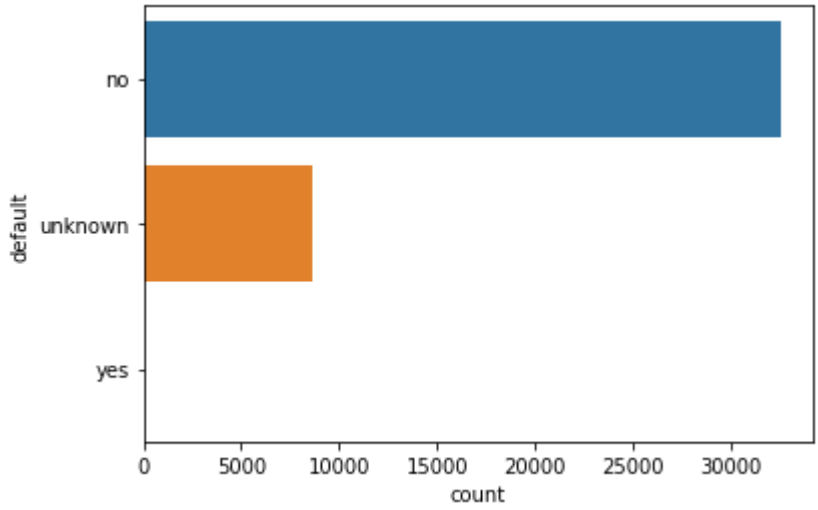
1.Here also we can see maximum in all days are equally subscribed by customers.So,we can't say a particular day.

Default:-

In [ ]:
```
print(data.loan.value_counts())
sns.countplot(y='default', data=data)
```

```
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf551d7f98>



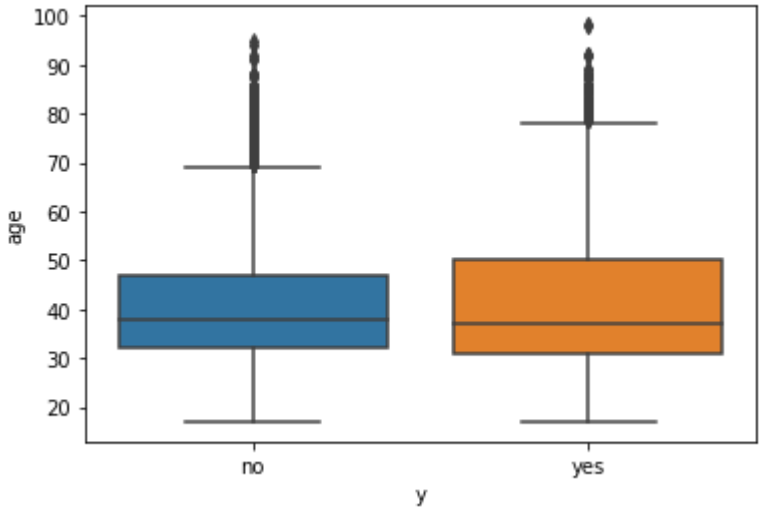
1.A default on credit is when you fail to pay an expected debt and here if we can see there are no subscribers and few were subscribed are unknown(no use).

Univariate Analysis of Numerical Values:-

->we have Age, Campaign, Previous, emp.var.rate, euribor3m, cons.price.idx, nr.employed, Pdays, Duration has numerical values.

Age:-

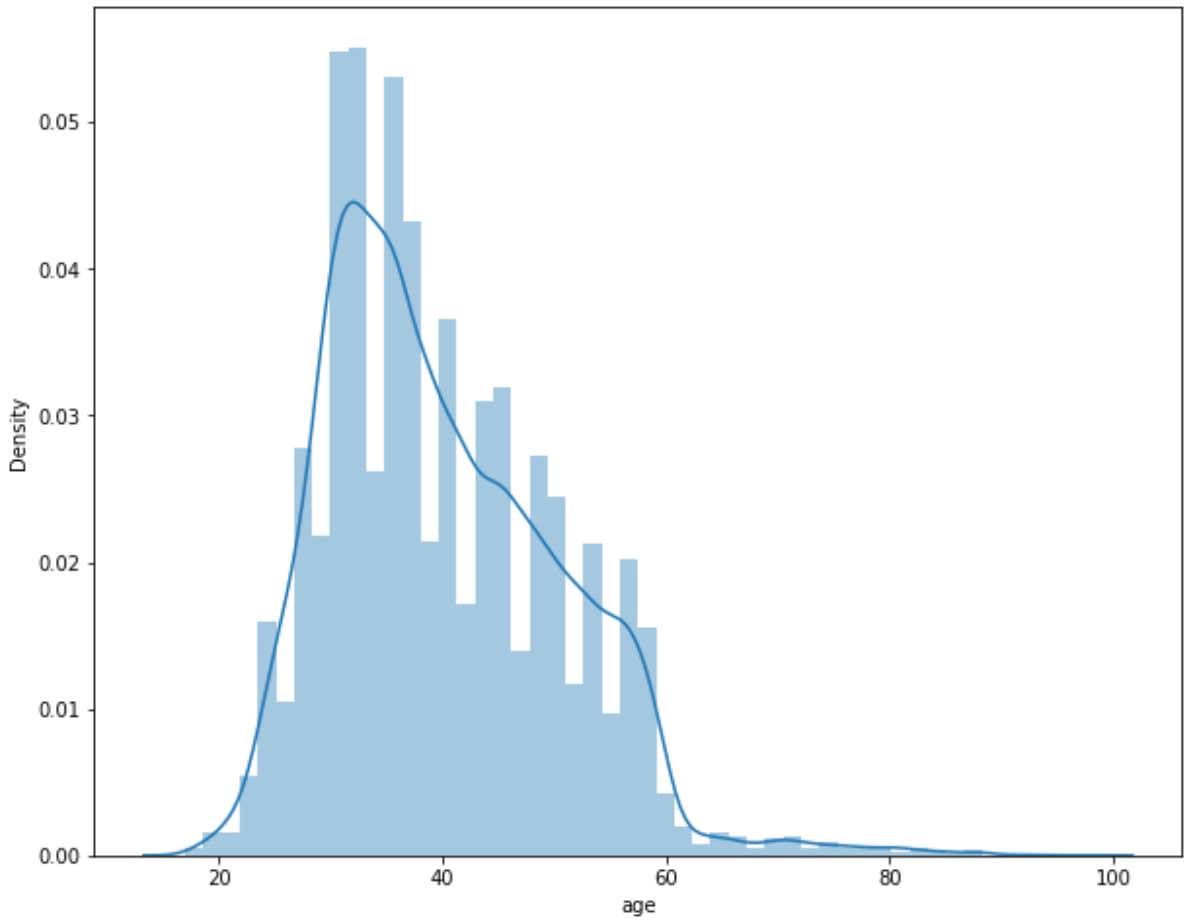
In [ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
sns.boxplot(data=data, x="y", y="age")
plt.show()
```



In [ ]:
```
plt.figure(figsize=(10,8))
sns.distplot(data["age"])
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf550bb358>

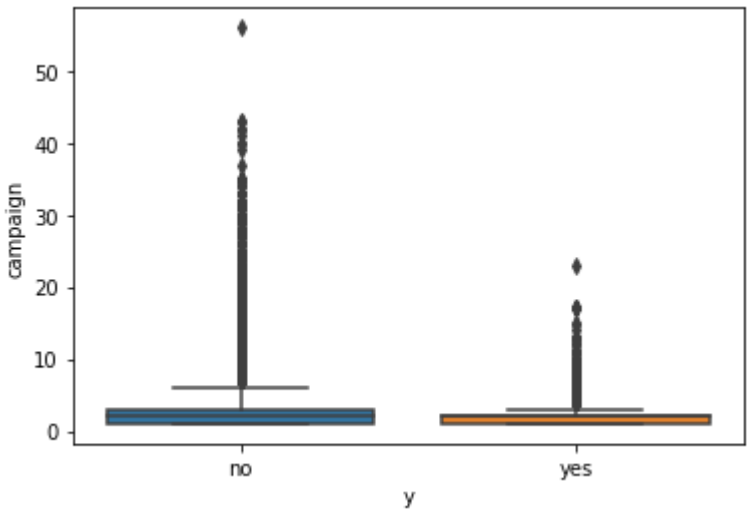


1.Here if you see both box-plots slightly overlapped and it indicates that people who subscribed the term deposit and people doesn't subscribed are same at 38-40.

2.And if see dist-plot then pdf increases at 30 and falling slightly at 65 that means people subscribed more at age between 30 and 65.

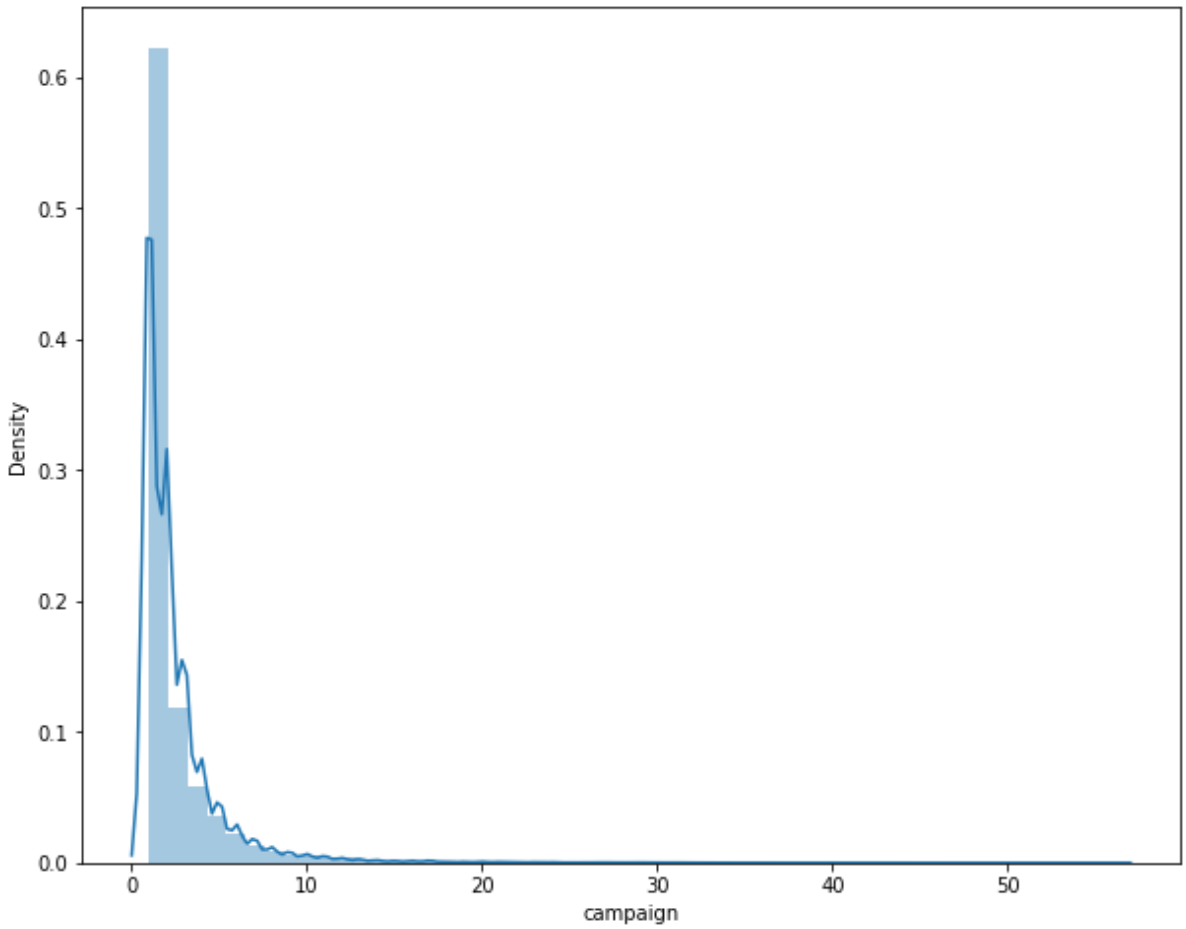
Campaign:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="campaign")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["campaign"])
plt.show()
```

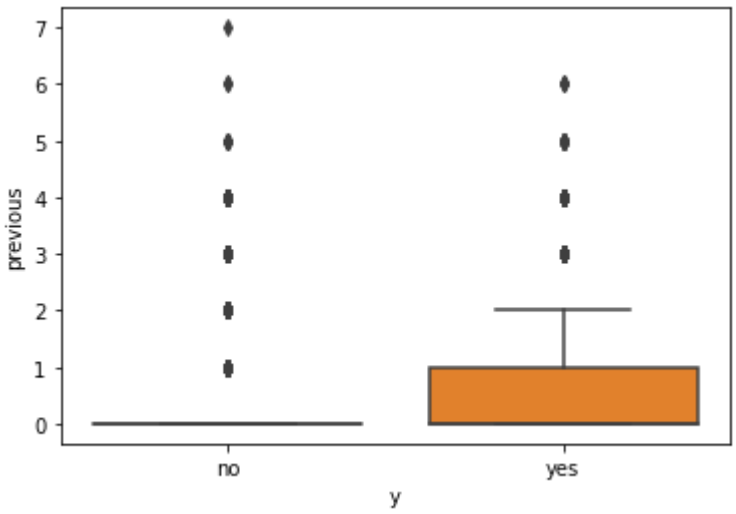
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



1.By seeing both box-plot and dist-plot we can say that majority of calls contacted with customers are 1 or 2.

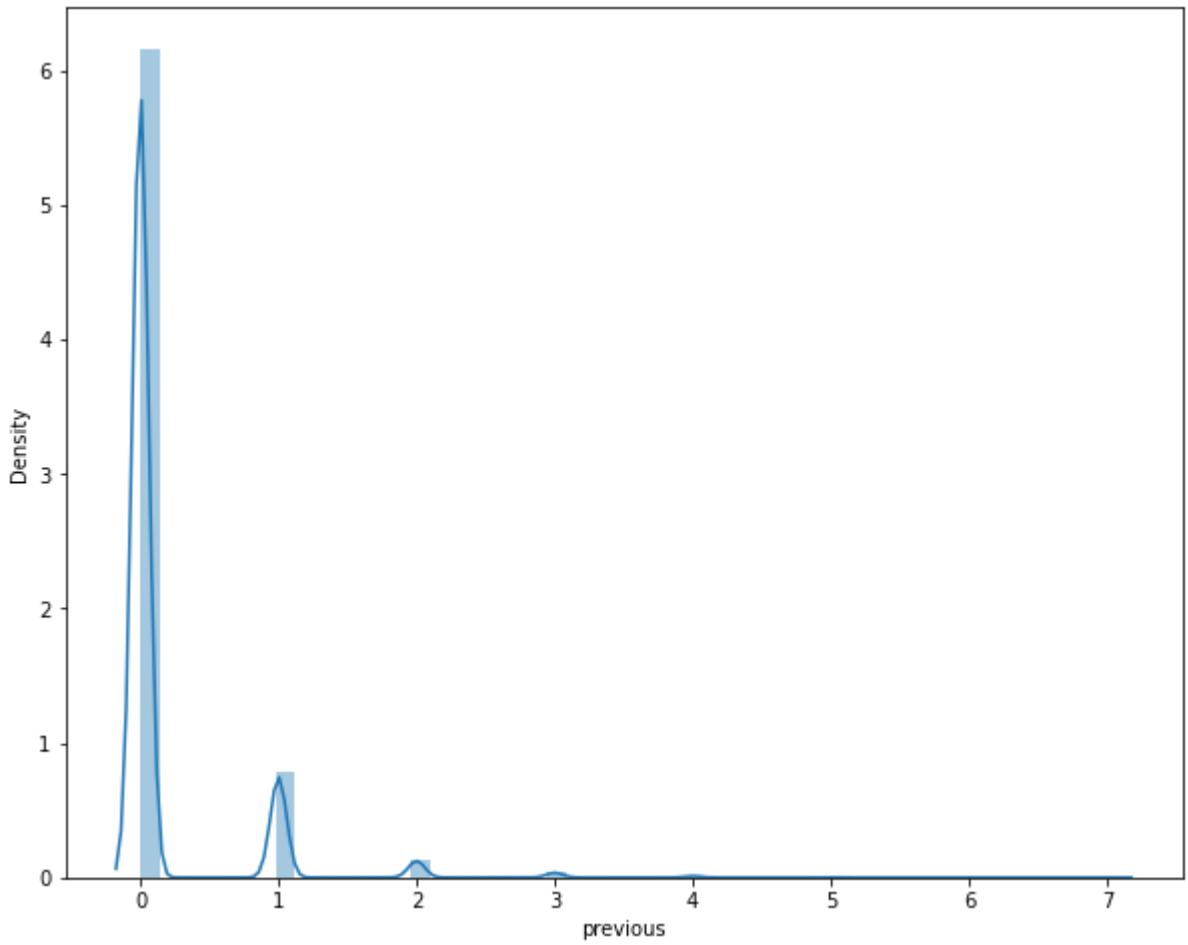
Previous:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="previous")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["previous"])
plt.show()
```

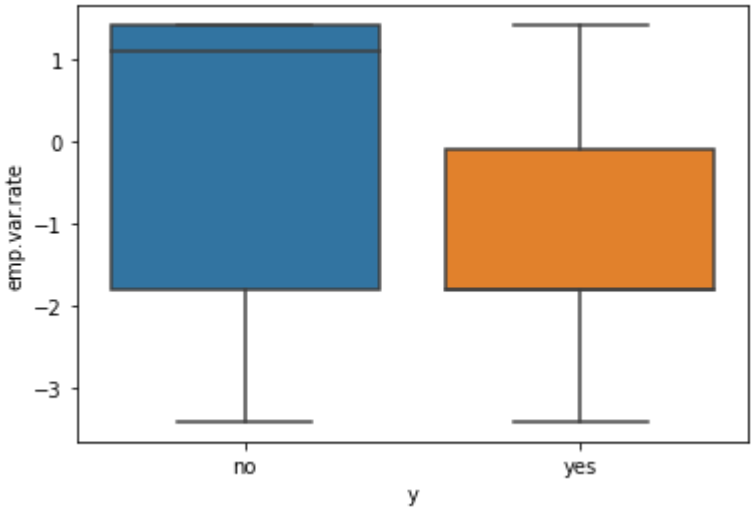
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



1.The previous feature is very similarly distributed for both the classes.

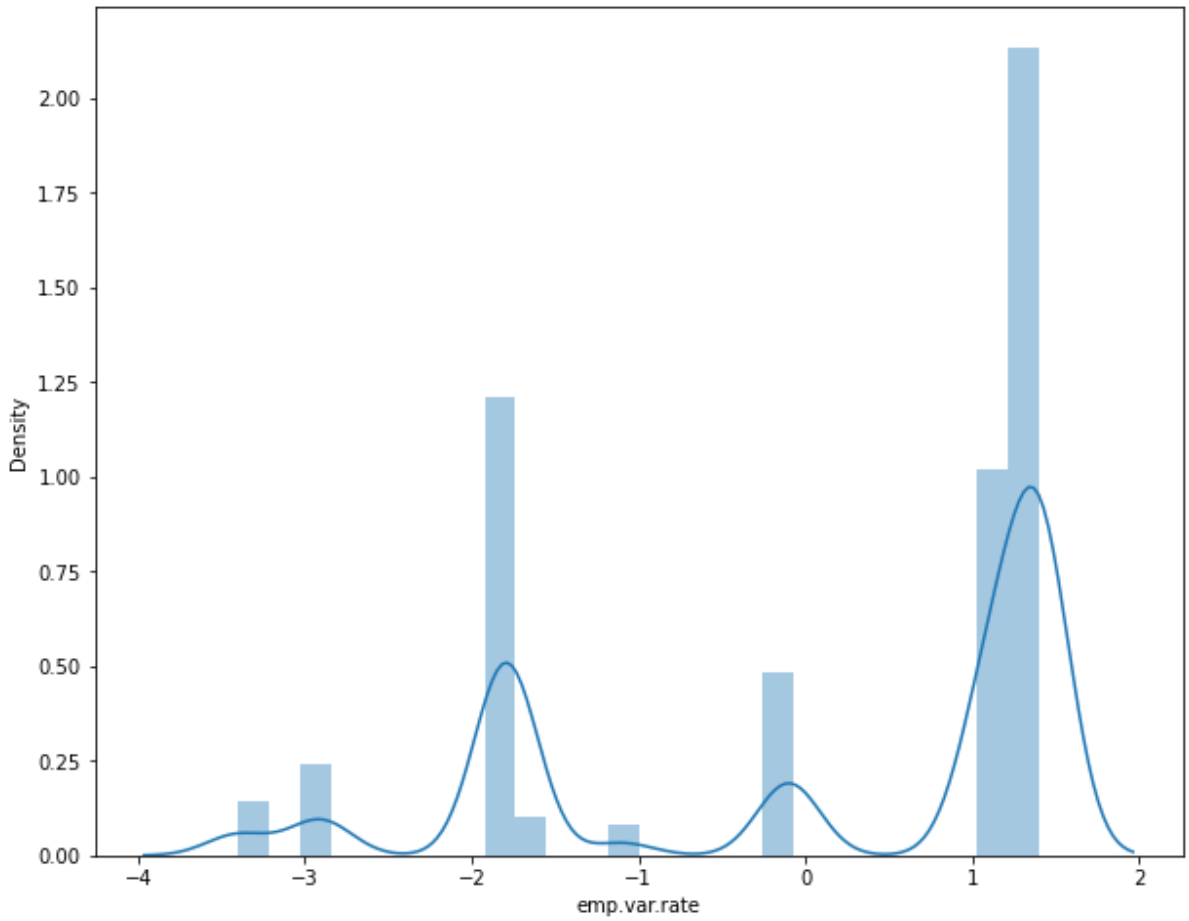
emp.var.rate:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="emp.var.rate")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["emp.var.rate"])
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```

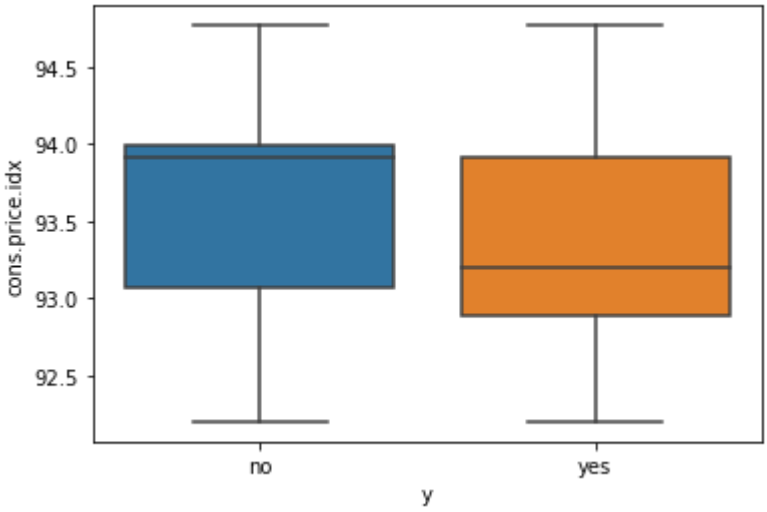


1.Employment variable rate its an interest rate on loan or security that fluctuates over time because it is based on an underlying benchmark interest rate or index that changes periodically(quarterly-indicator).

2.There is a future scope that people will subscribed term deposit.

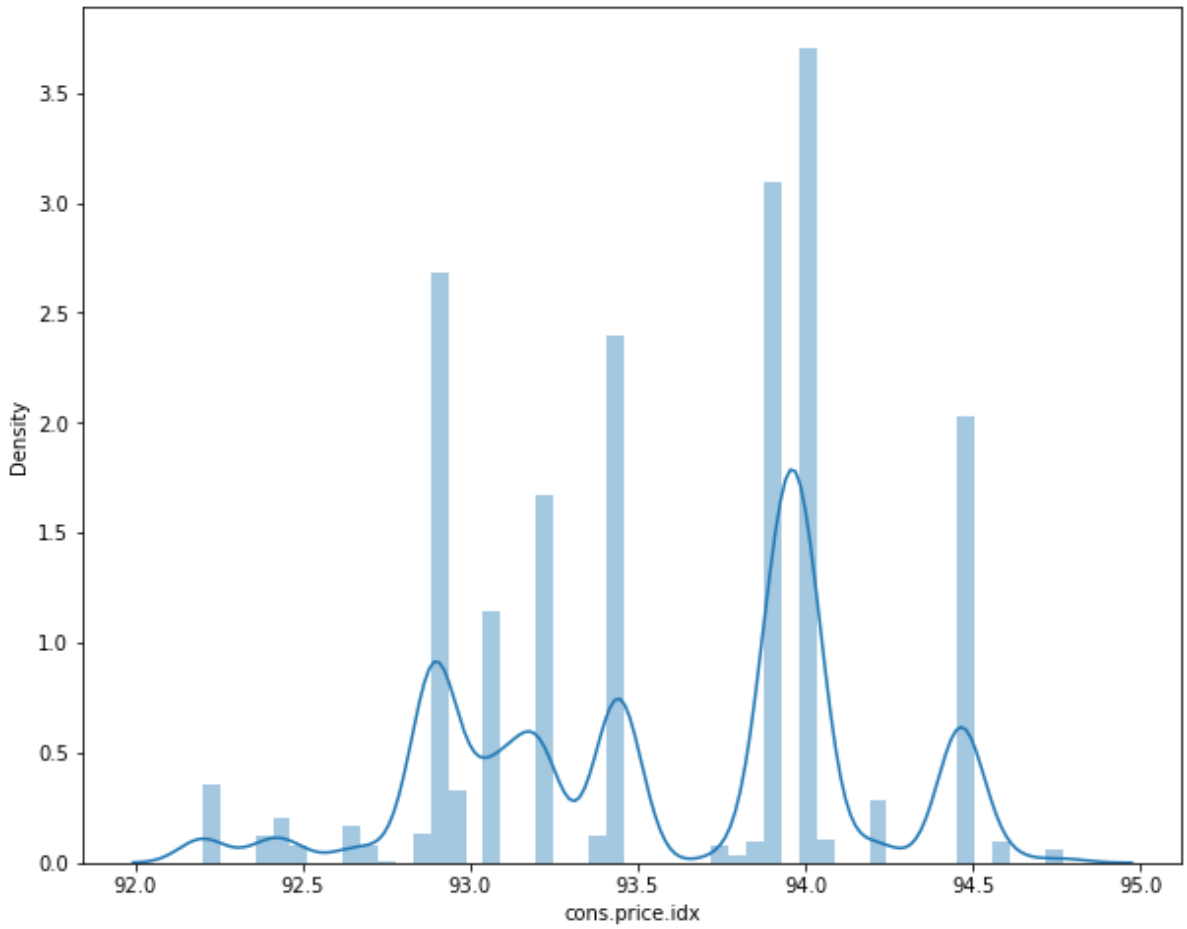
Cons.price.idx:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="cons.price.idx")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["cons.price.idx"])
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```

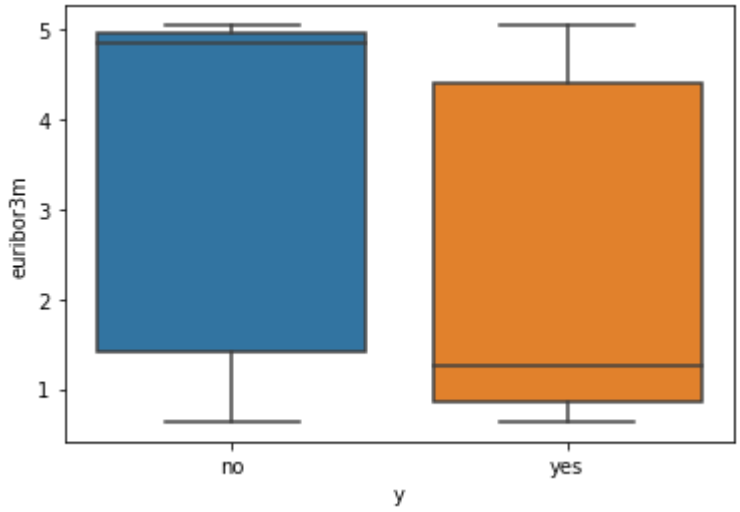


1.This is the important feature.

2.By seeing both box-plot and dist-plot we can say there is a future scope

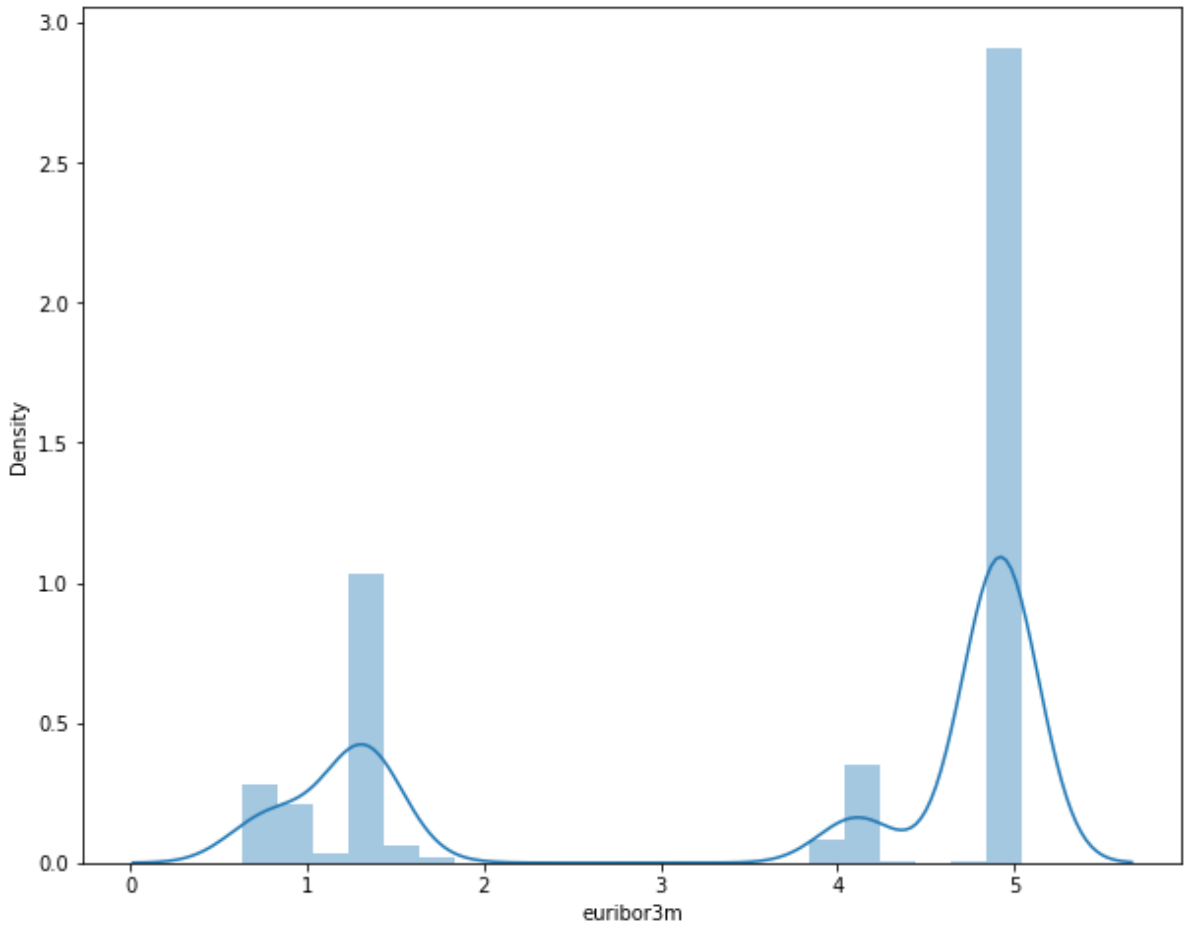
euribor3m:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="euribor3m")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["euribor3m"])
plt.show()
```

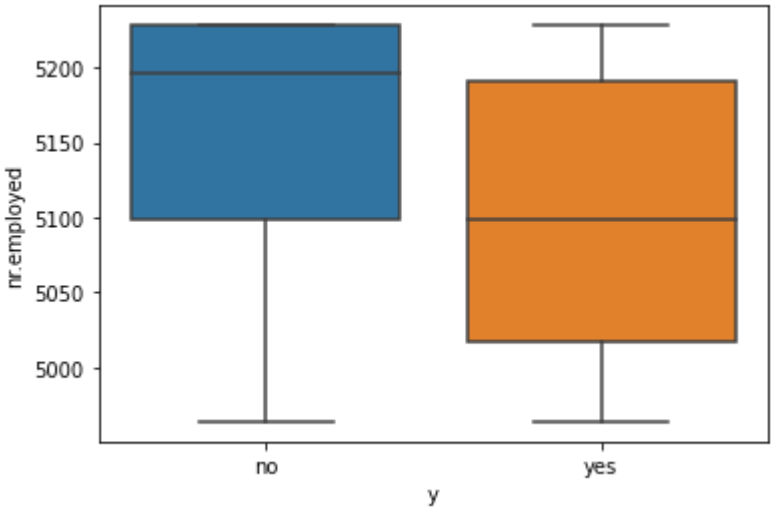
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



1.By seeing the box-plot and dist-plot we can say that in future people are more likely to subscribe.

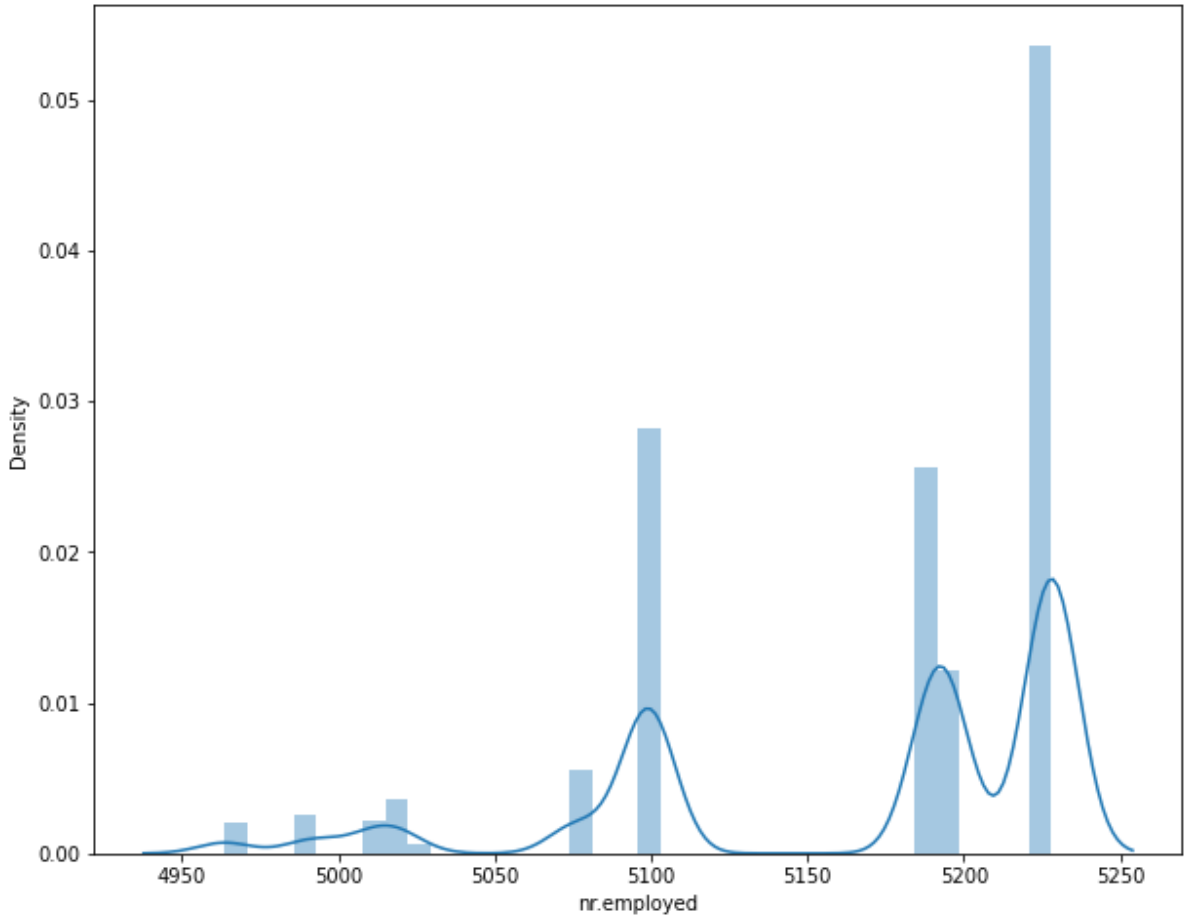
nr.employed:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="nr.employed")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["nr.employed"])
plt.show()
```

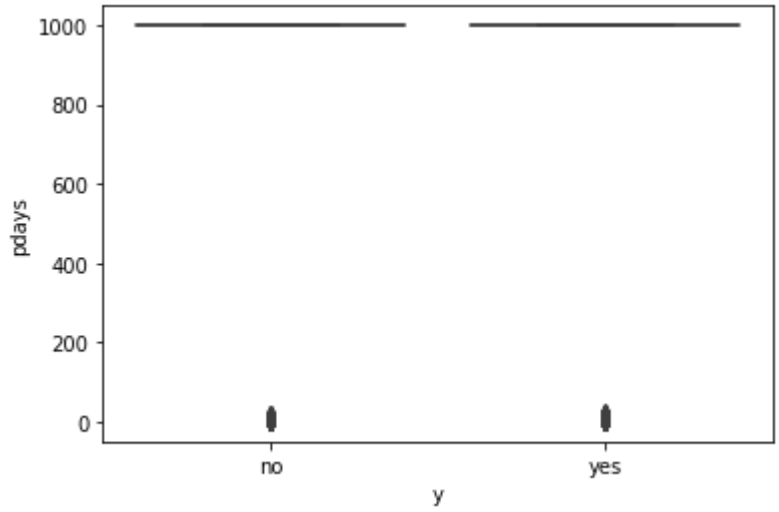
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



1.By seeing both box-plot and dist-plot we can say in future the no .of employees will subscribe more.

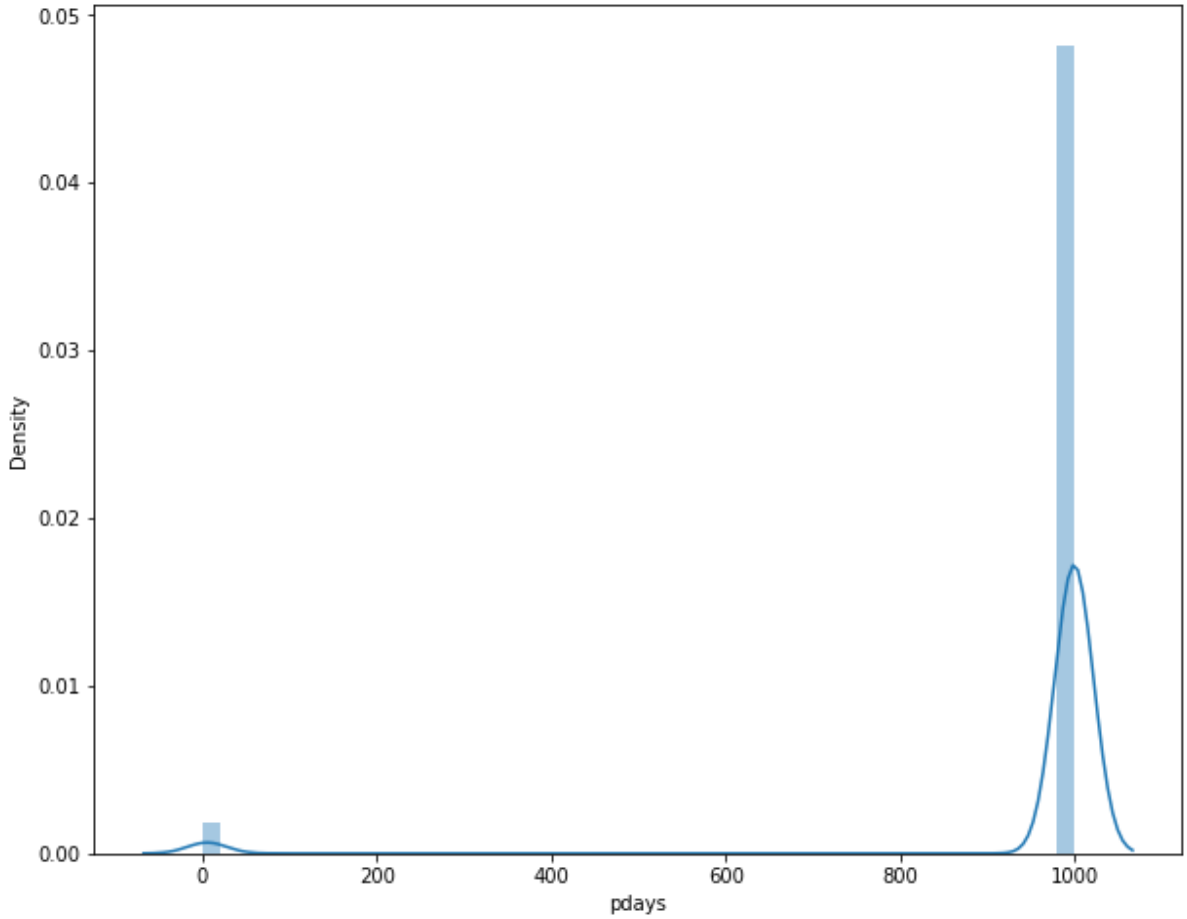
Pdays:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="pdays")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["pdays"])
plt.show()
```

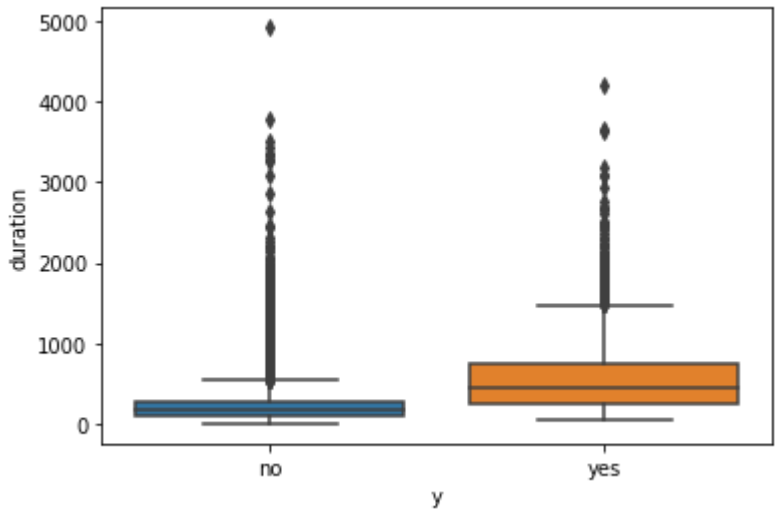
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



1.Here we can see number of days that passed by after the client was last contacted from a previous campaign is like 2.7 years back and also we can't predict by using this feature.

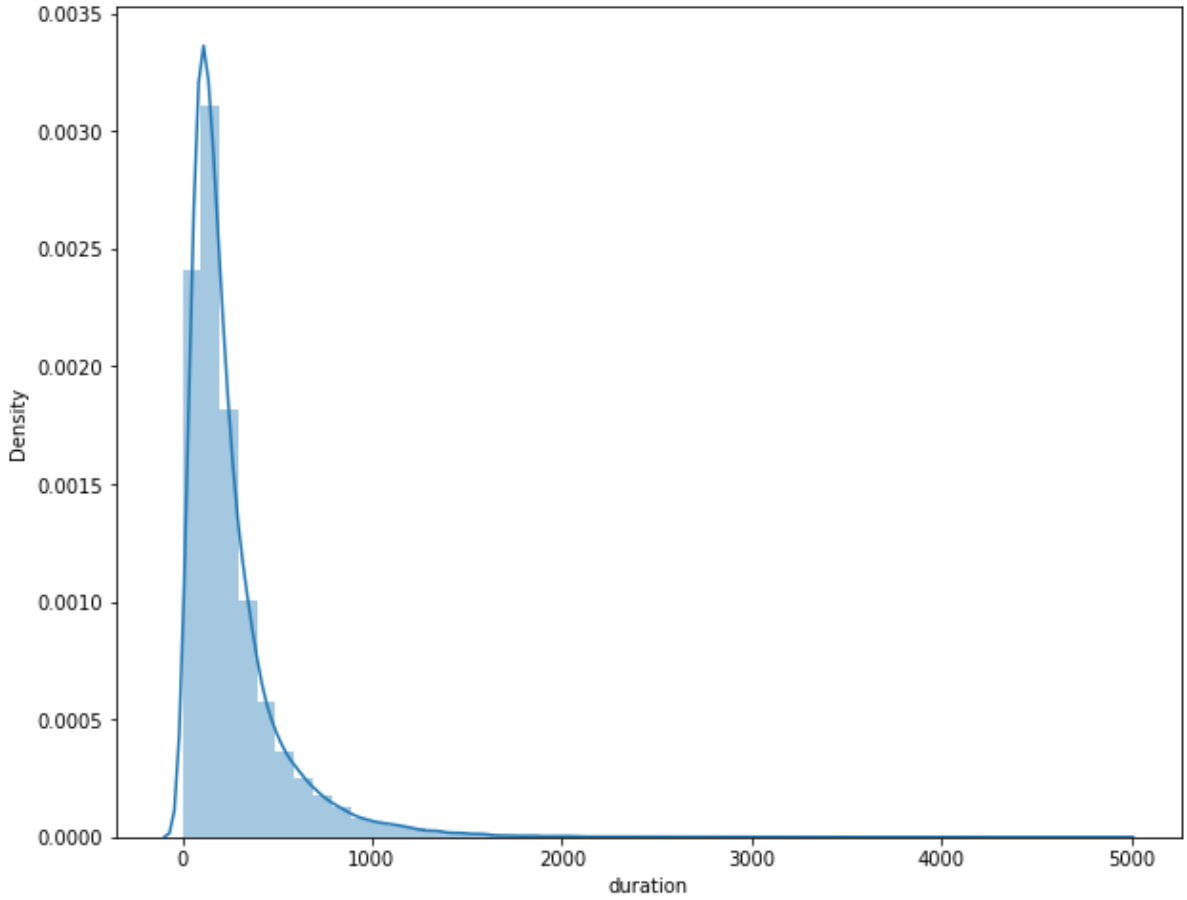
Duration:-

In [ ]:
```
%matplotlib inline
sns.boxplot(data=data, x="y", y="duration")
plt.show()
```



In [ ]:
```
%matplotlib inline
plt.figure(figsize=(10,8))
sns.distplot(data["duration"])
plt.show()
```

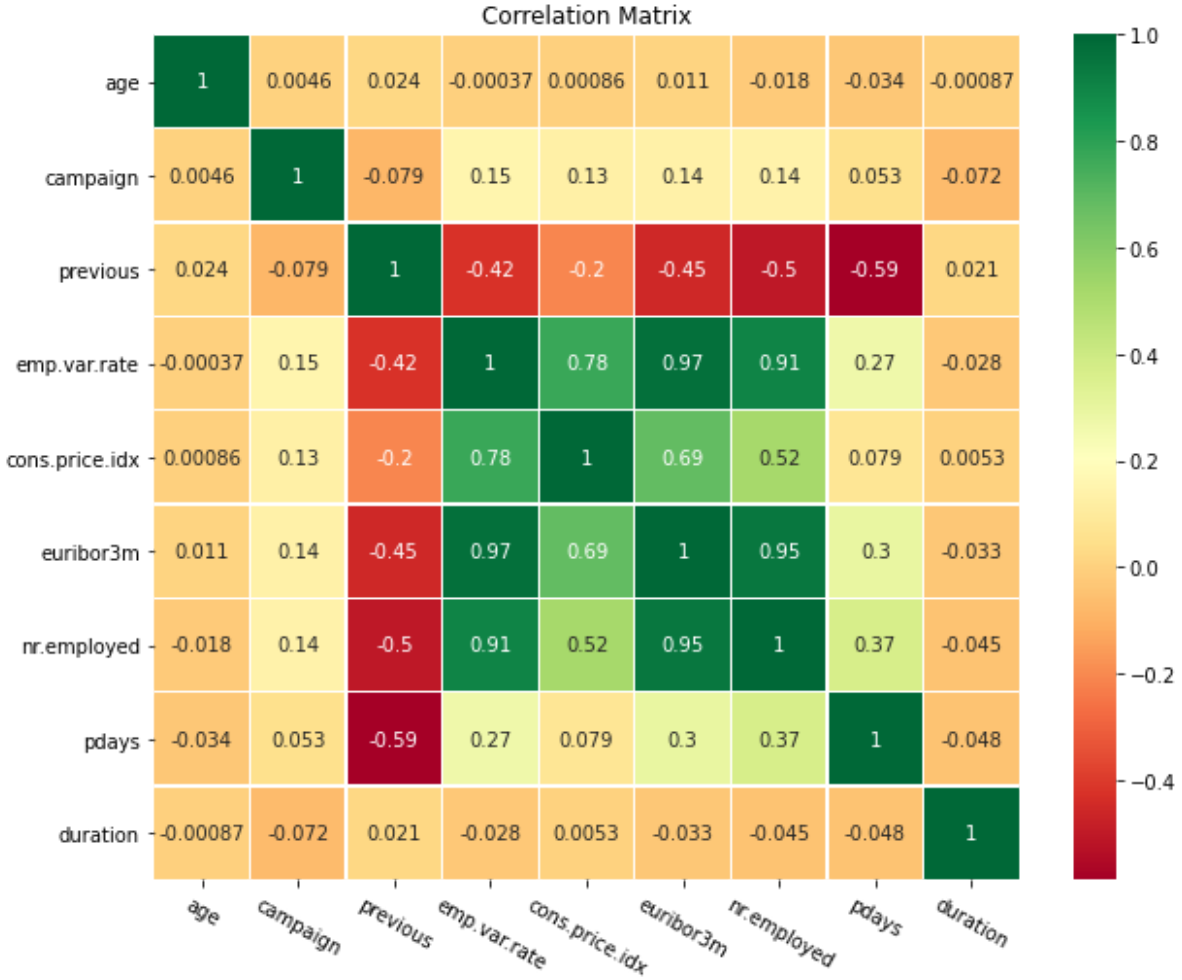
```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



1.Where the call duration is between 15-30 mins we can see more number of clients subscribing to Term Deposit plan and duration between 0-5 min and 5-10 min are less subscribed but we don't know before the call whwther he will subscribe are not.

Correlation Matrix:-

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
corr_data = data[['age', 'campaign', 'previous', 'emp.var.rate', 'cons.price.idx', 'euribor3m', 'nr.employed', 'pdays', 'duration']]
corr = corr_data.corr()

cor_plot = sns.heatmap(corr,annot=True,cmap='RdYlGn',linewidths=0.2,annot_kws={'size':10})
fig=plt.gcf()
fig.set_size_inches(10,8)
plt.xticks(fontsize=10,rotation=-30)
plt.yticks(fontsize=10)
plt.title('Correlation Matrix')
plt.show()
```

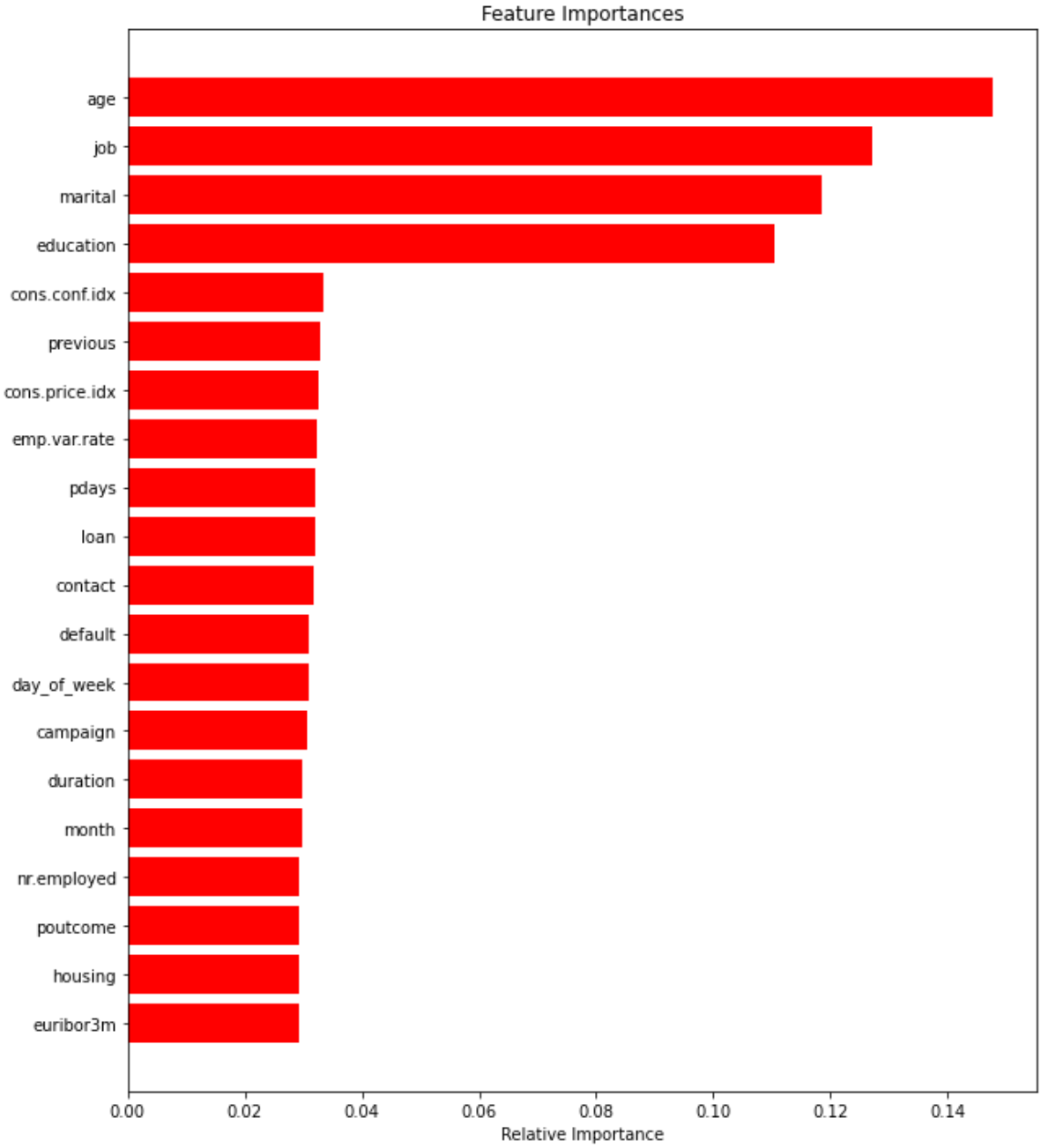


1.nr.employed is 0.95 correlated to euribor3m and euribor3m is 0.97 correlated to emp.var.rate.

2.emp.var.rate is 0.91 correlate with nr.employed.

Feature Importance:-

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

# Build a classification task using 4 informative features
X, y = make_classification(n_samples=1000,
                           n_features=20,
                           n_informative=4,
                           n_redundant=0,
                           n_repeated=0,
                           n_classes=2,
                           random_state=0,
                           shuffle=False)

# Build a forest and compute the feature importances
forest = RandomForestClassifier(n_estimators=250,
                                random_state=0)

forest.fit(X, y)
features = data.columns
importances = forest.feature_importances_
indices = (np.argsort(importances))
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



1.From the above plot we can see age,job,marital,education is four important features.

Data Preprocessing:-

One Hot Encoding:-

In [ ]:
```
y_valus ={'yes' : 1, 'no' : 0}
data['y'] = data['y'].map(lambda x: y_valus[x])
data['y'].value_counts()
```

Out[ ]:
```
0    36548
1     4640
Name: y, dtype: int64
```

In [ ]:
```
y = data['y']
```

In [ ]:
```
data.drop(['y'], axis = 1, inplace = True)
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(41188, 20)
```

In [ ]:
```
y.head(2)
```

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

In [ ]:
```
pip install --upgrade scikit-learn
```

```
Requirement already up-to-date: scikit-learn in /usr/local/lib/python3.6/dist
-packages (0.24.1)
Requirement already satisfied, skipping upgrade: scipy>=0.19.1 in /usr/local/
lib/python3.6/dist-packages (from scikit-learn) (1.4.1)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in /usr/local/l
ib/python3.6/dist-packages (from scikit-learn) (1.0.0)
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in /usr/local/
lib/python3.6/dist-packages (from scikit-learn) (1.19.5)
Requirement already satisfied, skipping upgrade: threadpoolctl>=2.0.0 in /us
r/local/lib/python3.6/dist-packages (from scikit-learn) (2.1.0)
```

In [ ]:
```
# splitting the data
from sklearn import model_selection
x_train, x_test, y_train, y_test = model_selection.train_test_split(data, y, t
est_size=0.3, random_state=42)
x_train, x_cv, y_train, y_cv = model_selection.train_test_split(x_train, y_tra
in, test_size=0.3)
```

In [ ]:
```
print("the shape of x_train data,{} and shape of x_test data {}:".format(x_tra
in.shape,x_test.shape))
print("the shape of y_train data,{} and shape of y_test data {}:".format(y_tra
in.shape,y_test.shape))
print("the shape of x_cv data,{} and shape of y_cv data {}:".format(x_cv.shape
,y_cv.shape))
```

```
the shape of x_train data,(20181, 20) and shape of x_test data (12357, 20):
the shape of y_train data,(20181,) and shape of y_test data (12357,):
the shape of x_cv data,(8650, 20) and shape of y_cv data (8650,):
```

In [ ]:
```
import sklearn
sklearn.preprocessing.OneHotEncoder
```

Out[ ]:
```
sklearn.preprocessing._encoders.OneHotEncoder
```

In [ ]:
```
#Numerical features
real_feature_x_train_age = ['age']
real_feature_x_train_campaign = ['campaign']
real_feature_x_train_previous = ['previous']
real_feature_x_train_emp_var_rate = ['emp.var.rate']
real_feature_x_train_cons_price_idx = ['cons.price.idx']
real_feature_x_train_cons_conf_idx = ['cons.conf.idx']
real_feature_x_train_euribor3m = ['euribor3m']
real_feature_x_train_nr_employed = ['nr.employed']

real_feature_x_test_age = ['age']
real_feature_x_test_campaign = ['campaign']
real_feature_x_test_previous = ['previous']
real_feature_x_test_emp_var_rate = ['emp.var.rate']
real_feature_x_test_cons_price_idx = ['cons.price.idx']
real_feature_x_test_cons_conf_idx = ['cons.conf.idx']
real_feature_x_test_euribor3m = ['euribor3m']
real_feature_x_test_nr_employed = ['nr.employed']

real_feature_x_cv_age = ['age']
real_feature_x_cv_campaign = ['campaign']
real_feature_x_cv_previous = ['previous']
real_feature_x_cv_emp_var_rate = ['emp.var.rate']
real_feature_x_cv_cons_price_idx = ['cons.price.idx']
real_feature_x_cv_cons_conf_idx = ['cons.conf.idx']
real_feature_x_cv_euribor3m = ['euribor3m']
real_feature_x_cv_nr_employed = ['nr.employed']
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
SS = StandardScaler()
x_train_real_feature_age_scale = SS.fit_transform(x_train[real_feature_x_train
_age])
x_test_real_feature_age_scale = SS.fit_transform(x_test[real_feature_x_test_ag
e])
x_cv_real_feature_age_scale = SS.fit_transform(x_cv[real_feature_x_cv_age])
```

In [ ]:
```
x_train_real_feature_campaign_scale = SS.fit_transform(x_train[real_feature_x_
train_campaign])
x_test_real_feature_campaign_scale = SS.fit_transform(x_test[real_feature_x_te
st_campaign])
x_cv_real_feature_campaign_scale = SS.fit_transform(x_cv[real_feature_x_cv_cam
paign])
```

In [ ]:
```
x_train_real_feature_previous_scale = SS.fit_transform(x_train[real_feature_x_
train_previous])
x_test_real_feature_previous_scale = SS.fit_transform(x_test[real_feature_x_te
st_previous])
x_cv_real_feature_previous_scale = SS.fit_transform(x_cv[real_feature_x_cv_pre
vious])
```

In [ ]:
```
x_train_real_feature_emp_var_rate_scale = SS.fit_transform(x_train[real_featur
e_x_train_emp_var_rate])
x_test_real_feature_emp_var_rate_scale = SS.fit_transform(x_test[real_feature_
x_test_emp_var_rate])
x_cv_real_feature_emp_var_rate_scale = SS.fit_transform(x_cv[real_feature_x_cv
_emp_var_rate])
```

In [ ]:
```
x_train_real_feature_cons_price_idx_scale = SS.fit_transform(x_train[real_feat
ure_x_train_cons_price_idx])
x_test_real_feature_cons_price_idx_scale = SS.fit_transform(x_test[real_featur
e_x_test_cons_price_idx])
x_cv_real_feature_cons_price_idx_scale = SS.fit_transform(x_cv[real_feature_x_
cv_cons_price_idx])
```

In [ ]:
```
x_train_real_feature_cons_conf_idx_scale = SS.fit_transform(x_train[real_featu
re_x_train_cons_conf_idx])
x_test_real_feature_cons_conf_idx_scale = SS.fit_transform(x_test[real_feature
_x_test_cons_conf_idx])
x_cv_real_feature_cons_conf_idx_scale = SS.fit_transform(x_cv[real_feature_x_c
v_cons_conf_idx])
```

In [ ]:
```
x_train_real_feature_euribor3m_scale = SS.fit_transform(x_train[real_feature_x
_train_euribor3m])
x_test_real_feature_euribor3m_scale = SS.fit_transform(x_test[real_feature_x_t
est_euribor3m])
x_cv_real_feature_euribor3m_scale = SS.fit_transform(x_cv[real_feature_x_cv_eu
ribor3m])
```

In [ ]:
```
x_train_real_feature_nr_employed_scale = SS.fit_transform(x_train[real_feature
_x_train_nr_employed])
x_test_real_feature_nr_employed_scale = SS.fit_transform(x_test[real_feature_x
_test_nr_employed])
x_cv_real_feature_nr_employed_scale = SS.fit_transform(x_cv[real_feature_x_cv_
nr_employed])
```

In [ ]:
```
import numpy as np
real_feature_x_train = np.concatenate((x_train_real_feature_age_scale,x_train_
real_feature_campaign_scale,x_train_real_feature_previous_scale,x_train_real_f
eature_emp_var_rate_scale,x_train_real_feature_cons_price_idx_scale,x_train_re
al_feature_cons_conf_idx_scale,x_train_real_feature_euribor3m_scale,x_train_re
al_feature_nr_employed_scale),axis = 1)
real_feature_x_test = np.concatenate((x_test_real_feature_age_scale,x_test_rea
l_feature_campaign_scale,x_test_real_feature_previous_scale,x_test_real_featur
e_emp_var_rate_scale,x_test_real_feature_cons_price_idx_scale,x_test_real_feat
ure_cons_conf_idx_scale,x_test_real_feature_euribor3m_scale,x_test_real_featur
e_nr_employed_scale),axis = 1)
real_feature_x_cv = np.concatenate((x_cv_real_feature_age_scale,x_cv_real_feat
ure_campaign_scale,x_cv_real_feature_previous_scale,x_cv_real_feature_emp_var_
rate_scale,x_cv_real_feature_cons_price_idx_scale,x_cv_real_feature_cons_conf_
idx_scale,x_cv_real_feature_euribor3m_scale,x_cv_real_feature_nr_employed_scal
e),axis = 1)
```

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
OHE = OneHotEncoder()
```

In [ ]:
```
# Categorical features
cat_feat_x_train_job = x_train['job'].values
cat_feat_x_train_marital = x_train['marital'].values
cat_feat_x_train_education = x_train['education'].values
cat_feat_x_train_housing = x_train['housing'].values
cat_feat_x_train_loan = x_train['loan'].values
cat_feat_x_train_poutcome = x_train['poutcome'].values

cat_feat_x_test_job = x_test['job'].values
cat_feat_x_test_marital = x_test['marital'].values
cat_feat_x_test_education = x_test['education'].values
cat_feat_x_test_housing = x_test['housing'].values
cat_feat_x_test_loan = x_test['loan'].values
cat_feat_x_test_poutcome = x_test['poutcome'].values

cat_feat_x_cv_job = x_cv['job'].values
cat_feat_x_cv_marital = x_cv['marital'].values
cat_feat_x_cv_education = x_cv['education'].values
cat_feat_x_cv_housing = x_cv['housing'].values
cat_feat_x_cv_loan = x_cv['loan'].values
cat_feat_x_cv_poutcome = x_cv['poutcome'].values
```

In [ ]:
```
x_train_job = cat_feat_x_train_job.reshape(-1,1)
x_test_job = cat_feat_x_test_job.reshape(-1,1)
x_cv_job = cat_feat_x_cv_job.reshape(-1,1)
ohe =  OneHotEncoder()
ohe.fit(x_train_job)
job_train = ohe.transform(x_train_job)
job_test = ohe.transform(x_test_job)
job_cv = ohe.transform(x_cv_job)
```

In [ ]:
```
x_train_marital= cat_feat_x_train_marital.reshape(-1,1)
x_test_marital = cat_feat_x_test_marital.reshape(-1,1)
x_cv_marital = cat_feat_x_cv_marital.reshape(-1,1)
ohe =  OneHotEncoder()
ohe.fit(x_train_marital)
marital_train = ohe.transform(x_train_marital)
marital_test = ohe.transform(x_test_marital)
marital_cv = ohe.transform(x_cv_marital)
```

In [ ]:
```
x_train_education= cat_feat_x_train_education.reshape(-1,1)
x_test_education = cat_feat_x_test_education.reshape(-1,1)
x_cv_education = cat_feat_x_cv_education.reshape(-1,1)
ohe =  OneHotEncoder()
ohe.fit(x_train_education)
education_train = ohe.transform(x_train_education)
education_test = ohe.transform(x_test_education)
education_cv = ohe.transform(x_cv_education)
```

In [ ]:
```
x_train_housing= cat_feat_x_train_housing.reshape(-1,1)
x_test_housing = cat_feat_x_test_housing.reshape(-1,1)
x_cv_housing= cat_feat_x_cv_housing.reshape(-1,1)
ohe =  OneHotEncoder()
ohe.fit(x_train_housing)
housing_train = ohe.transform(x_train_housing)
housing_test = ohe.transform(x_test_housing)
housing_cv = ohe.transform(x_cv_housing)
```

In [ ]:
```
x_train_loan = cat_feat_x_train_loan.reshape(-1,1)
x_test_loan = cat_feat_x_test_loan.reshape(-1,1)
x_cv_loan = cat_feat_x_cv_loan.reshape(-1,1)
ohe =  OneHotEncoder()
ohe.fit(x_train_loan)
loan_train = ohe.transform(x_train_loan)
loan_test = ohe.transform(x_test_loan)
loan_cv = ohe.transform(x_cv_loan)
```

In [ ]:
```
x_train_poutcome = cat_feat_x_train_poutcome.reshape(-1,1)
x_test_poutcome = cat_feat_x_test_poutcome.reshape(-1,1)
x_cv_poutcome = cat_feat_x_cv_poutcome.reshape(-1,1)
ohe =  OneHotEncoder()
ohe.fit(x_train_poutcome)
poutcome_train = ohe.transform(x_train_poutcome)
poutcome_test = ohe.transform(x_test_poutcome)
poutcome_cv = ohe.transform(x_cv_poutcome)
```

In [ ]:
```
from scipy.sparse import hstack
```

In [ ]:
```
cat_x_train = hstack([job_train,marital_train,education_train,housing_train,loan_train,poutcome_train]).tocsr()
cat_x_test = hstack([job_test,marital_test,education_test,housing_test,loan_test,poutcome_test]).tocsr()
cat_x_cv = hstack([job_cv,marital_cv,education_cv,housing_cv,loan_cv,poutcome_cv]).tocsr()
```

In [ ]:
```
cat_x_train.shape
```

Out[ ]:
```
(20181, 33)
```

In [ ]:
```
cat_x_test.shape
```

Out[ ]:
```
(12357, 33)
```

In [ ]:
```
cat_x_cv.shape
```

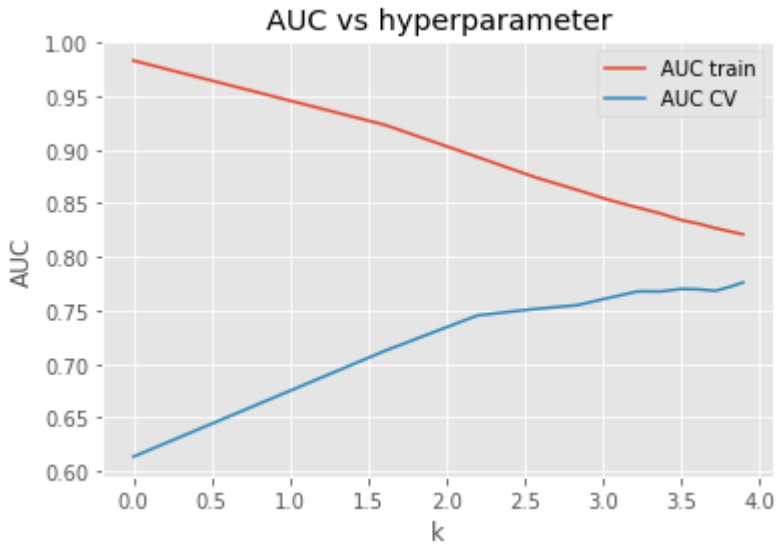
Out[ ]:
```
(8650, 33)
```

In [ ]:
```
ohe_x_train = hstack([cat_x_train,real_feature_x_train])
ohe_x_test = hstack([cat_x_test,real_feature_x_test])
ohe_x_cv = hstack([cat_x_cv,real_feature_x_cv])
```

Machine Learning Models

Applying KNN Brute Force Algorithm

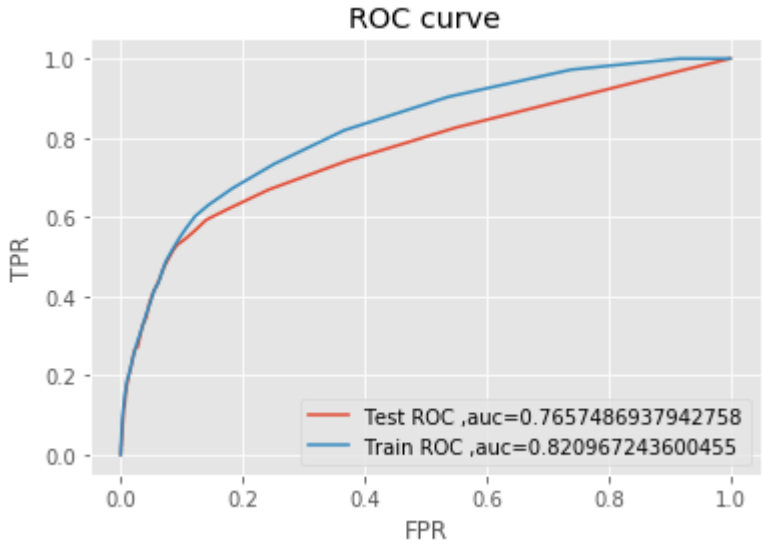
In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import math

k = list(range(1,50,4))

train_auc = []
cv_auc = []

for i in k:
    clf = KNeighborsClassifier(n_neighbors = i,algorithm='brute')
    clf.fit(ohe_x_train,y_train)
    prob_cv = clf.predict_proba(ohe_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(ohe_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_k = k[cv_auc.index(max(cv_auc))]
k = [math.log(x) for x in k]

#plot auc vs alpha
x = plt.subplot( )
x.plot(k, train_auc, label='AUC train')
x.plot(k, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('k')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal alpha for which auc is maximum : ',optimal_k)
```



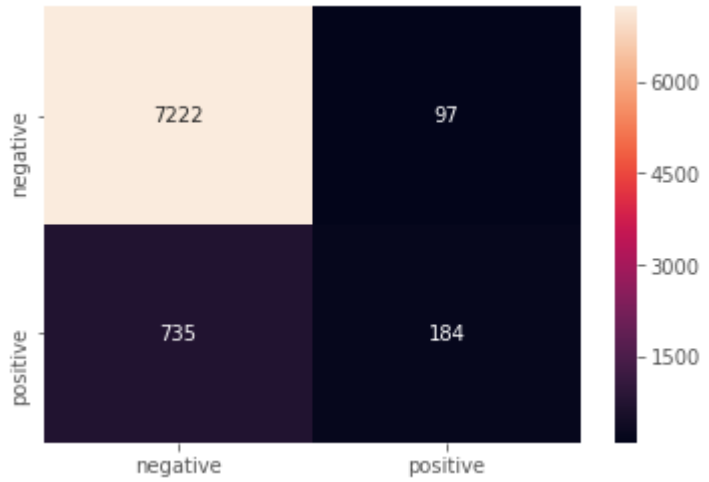

```
optimal alpha for which auc is maximum :  49
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
```

In [ ]:
```
#Testing AUC on Test data
clf = KNeighborsClassifier(n_neighbors = optimal_k,algorithm='brute')
clf.fit(ohe_x_train,y_train)
pred_test = clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



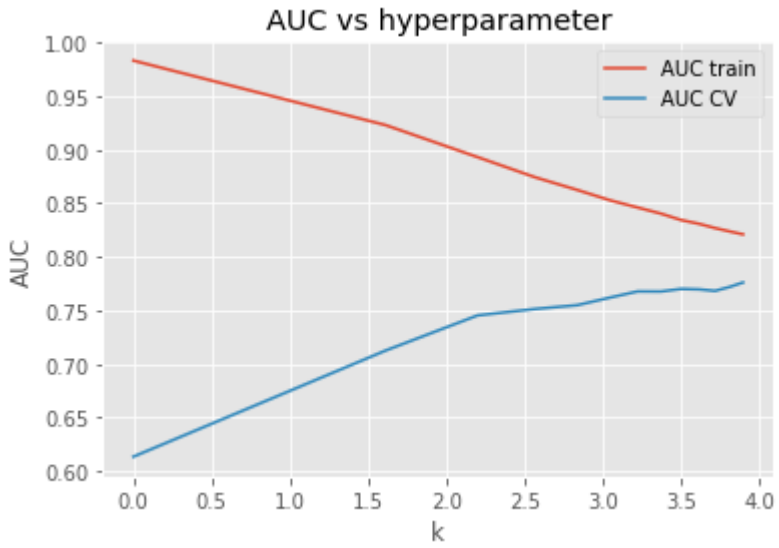

```
AUC on Test data is 0.7657486937942758
AUC on Train data is 0.820967243600455
---------------------------
```



In [ ]:
```
results1 = pd.DataFrame(columns=['model', 'Classifier' ,"hyper perameter", 'Train-AUC', 'Test-AUC' ])
new = ['KNN with Brute force','KNeighborsClassifier',"k = 49",0.8209,0.7657]
results1.loc[0] = new
```

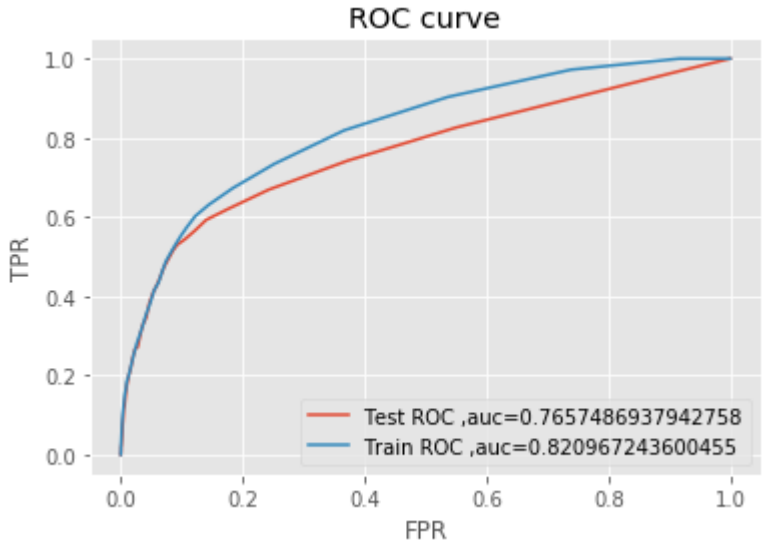
Applying KNN kd-Tree

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import math

k = list(range(1,50,4))

train_auc = []
cv_auc = []

for i in k:
    clf = KNeighborsClassifier(n_neighbors = i,algorithm='kd_tree')
    clf.fit(ohe_x_train,y_train)
    prob_cv = clf.predict_proba(ohe_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(ohe_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_k = k[cv_auc.index(max(cv_auc))]
k = [math.log(x) for x in k]

#plot auc vs alpha
x = plt.subplot( )
x.plot(k, train_auc, label='AUC train')
x.plot(k, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('k')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal alpha for which auc is maximum : ',optimal_k)
```

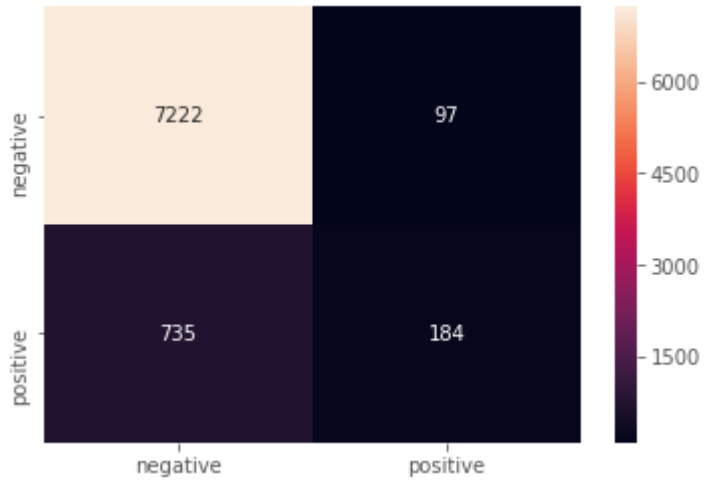



```
optimal alpha for which auc is maximum :  49
```

In [ ]:
```
#Testing AUC on Test data
clf = KNeighborsClassifier(n_neighbors = optimal_k,algorithm='kd_tree')
clf.fit(ohe_x_train,y_train)
pred_test = clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



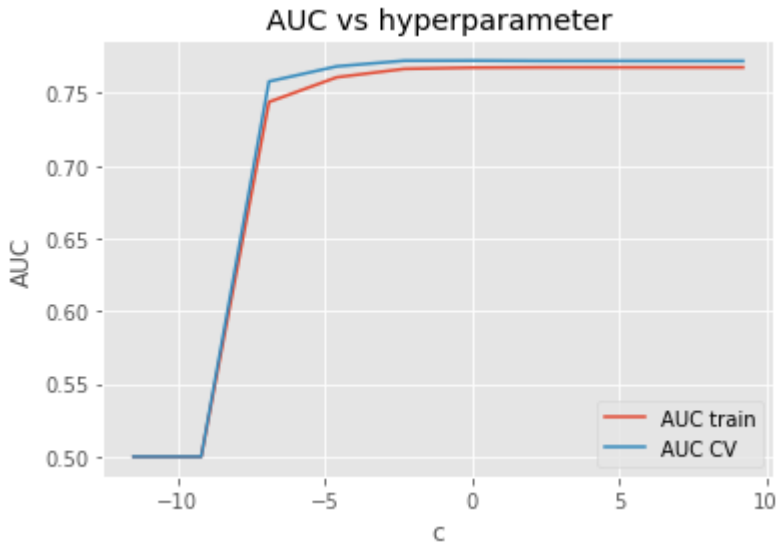

```
AUC on Test data is 0.7657486937942758
AUC on Train data is 0.820967243600455
---------------------------
```



In [ ]:
```
new = ['KNN with kd-tree','KNeighborsClassifier',"k = 49",0.8209,0.7657]
results1.loc[1] = new
```

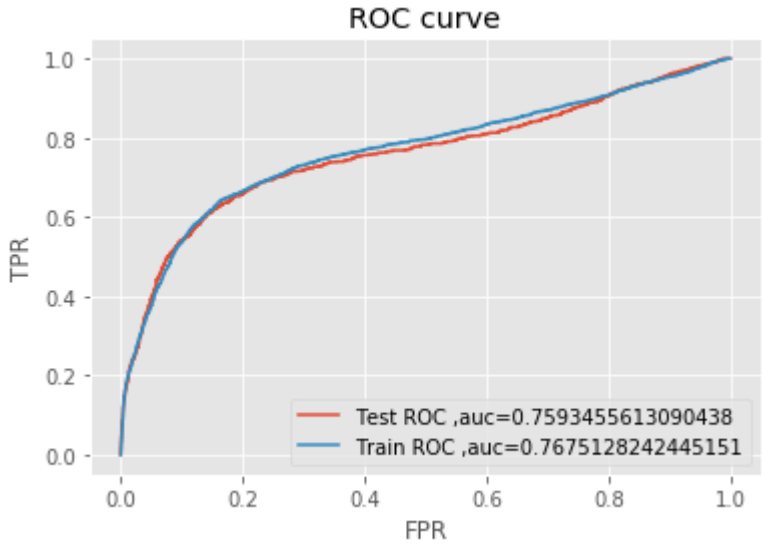
Applying Logistic Regression with L1 regularization

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import math

c = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001]

train_auc = []
cv_auc = []

for i in c:
    clf = LogisticRegression(penalty='l1',C=i)
    clf.fit(ohe_x_train,y_train)
    prob_cv = clf.predict_proba(ohe_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(ohe_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_c= c[cv_auc.index(max(cv_auc))]
c = [math.log(x) for x in c]

#plot auc vs alpha
x = plt.subplot( )
x.plot(c, train_auc, label='AUC train')
x.plot(c, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('c')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal c for which auc is maximum : ',optimal_c)
```

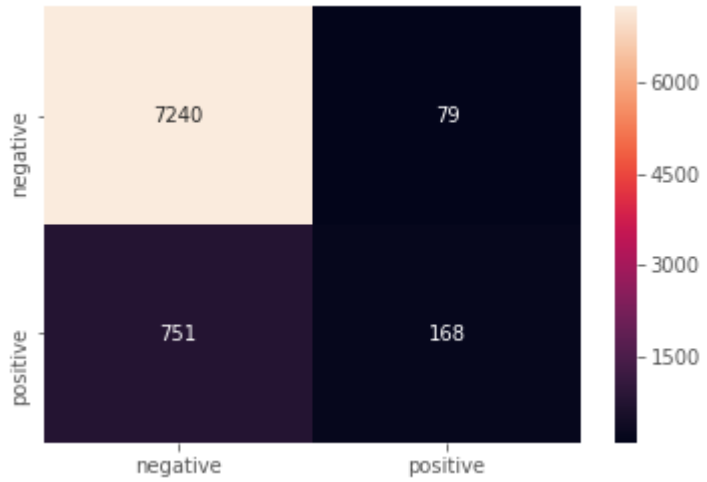



```
optimal c for which auc is maximum :  1
```

In [ ]:
```
#Testing AUC on Test data
clf = LogisticRegression(penalty='l1',C=optimal_c)
clf.fit(ohe_x_train,y_train)
pred_test = clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)


#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
from sklearn.metrics import confusion_matrix
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



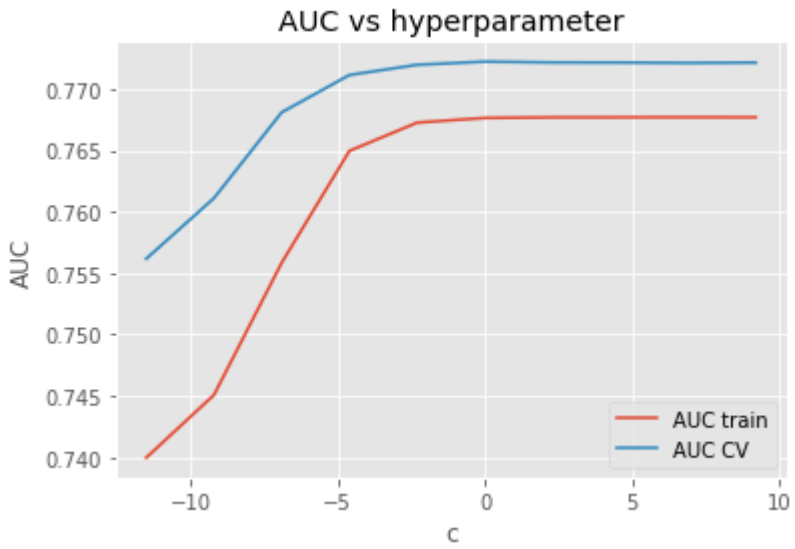

```
AUC on Test data is 0.7593455613090438
AUC on Train data is 0.7675128242445151
---------------------------
```



In [ ]:
```
new = ['Logistic Regression with L1','LogisticRegression',"c = 1",0.7675,0.759
3]
results1.loc[2] = new
```

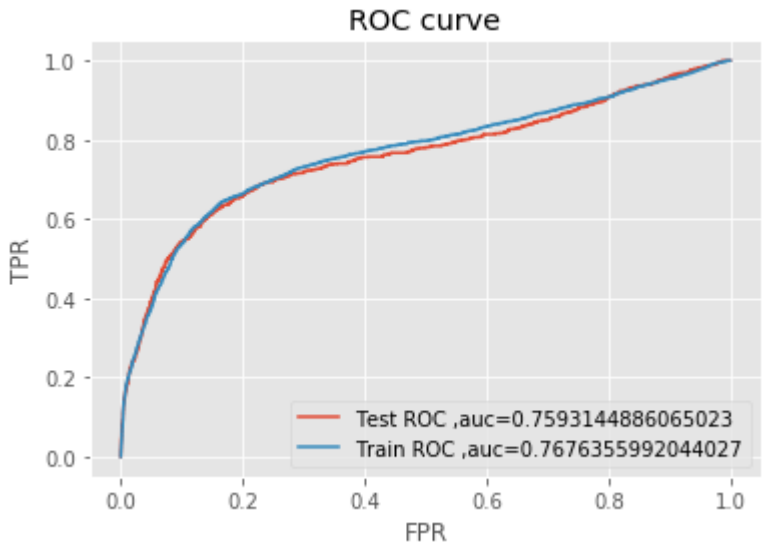
Applying Logistic Regression with L2 regularization

In [ ]:
```
c = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001]

train_auc = []
cv_auc = []

for i in c:
    clf = LogisticRegression(penalty='l2',C=i)
    clf.fit(ohe_x_train,y_train)
    prob_cv = clf.predict_proba(ohe_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(ohe_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_c= c[cv_auc.index(max(cv_auc))]
c = [math.log(x) for x in c]

#plot auc vs alpha
x = plt.subplot( )
x.plot(c, train_auc, label='AUC train')
x.plot(c, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('c')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal c for which auc is maximum : ',optimal_c)
```

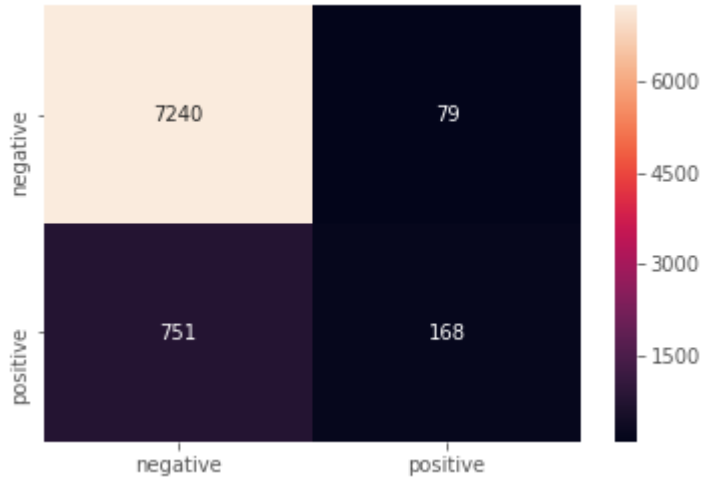



```
optimal c for which auc is maximum :  1
```

In [ ]:
```
#Testing AUC on Test data
clf = LogisticRegression(penalty='l2',C=optimal_c)
clf.fit(ohe_x_train,y_train)
pred_test = clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)


#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.7593144886065023
AUC on Train data is 0.7676355992044027
---------------------------
```

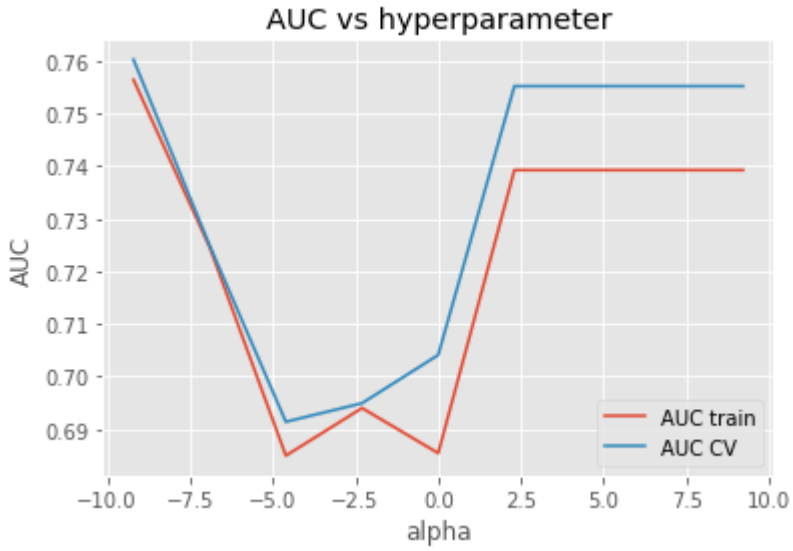


In [ ]:
```
new = ['Logistic Regression with L2','LogisticRegression',"c = 1",0.7676,0.7593]
results1.loc[3] = new
```

Applying Linear SVM

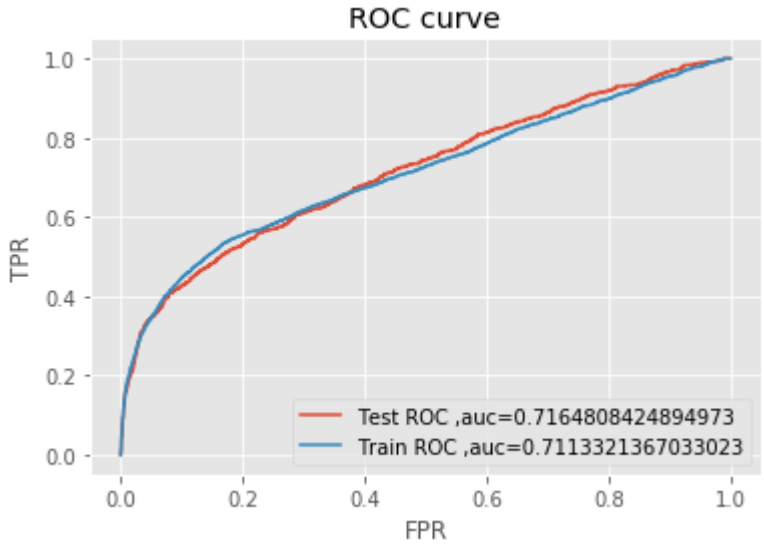
In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import math

alpha = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001]

train_auc = []
cv_auc = []

for i in alpha:
    model = SGDClassifier(alpha=i, loss = "hinge")
    clf = CalibratedClassifierCV(model, cv=3)
    clf.fit(ohe_x_train,y_train)
    prob_cv = clf.predict_proba(ohe_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(ohe_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_alpha= alpha[cv_auc.index(max(cv_auc))]
alpha=[math.log(x) for x in alpha]

#plot auc vs alpha
x = plt.subplot( )
x.plot(alpha, train_auc, label='AUC train')
x.plot(alpha, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('alpha')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

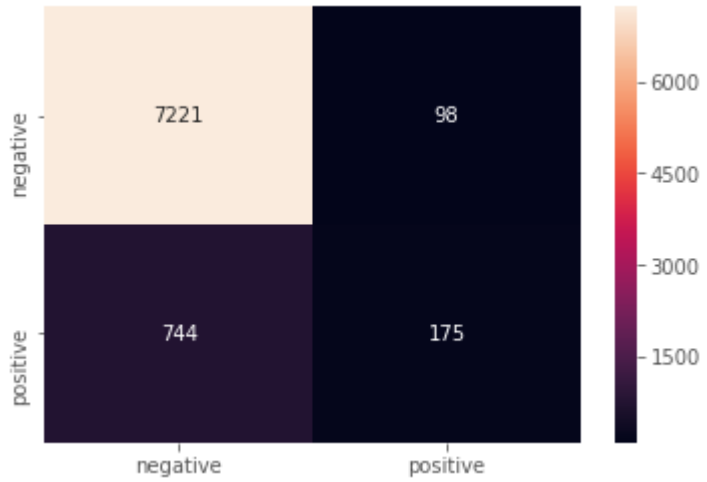



```
optimal alpha for which auc is maximum :  0.0001
```

In [ ]:
```
#Testing AUC on Test data
model = SGDClassifier(alpha = optimal_alpha)
clf = CalibratedClassifierCV(model, cv=3)
clf.fit(ohe_x_train,y_train)
pred_test = clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



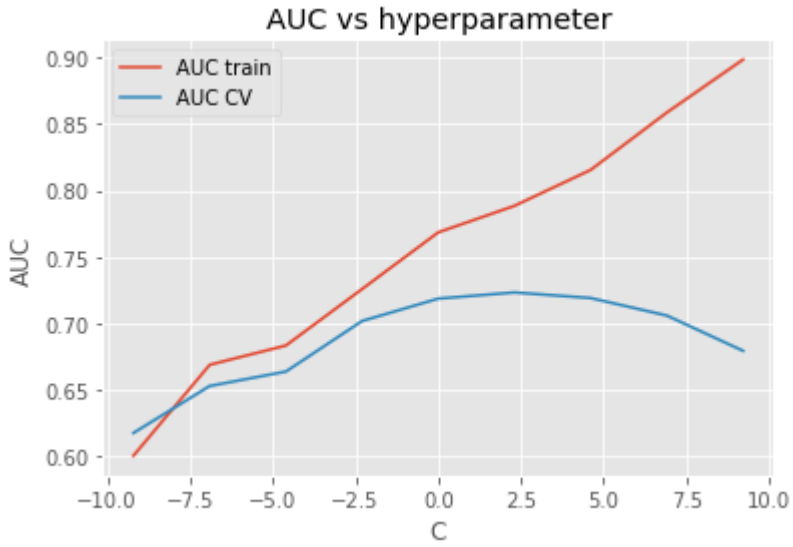

```
AUC on Test data is 0.7164808424894973
AUC on Train data is 0.7113321367033023
---------------------------
```



In [ ]:
```
new = ['Linear SVM','SGDClassifier',"alpha = 0.0001",0.7113,0.7164]
results1.loc[4] = new
```

Applying RBF SVM

In [ ]:
```
from sklearn.svm import SVC
C = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001]

train_auc = []
cv_auc = []

for i in C:
    model = SVC(C=i)
    clf = CalibratedClassifierCV(model, cv=3)
    clf.fit(ohe_x_train,y_train)
    prob_cv = clf.predict_proba(ohe_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(ohe_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_C= C[cv_auc.index(max(cv_auc))]
C=[math.log(x) for x in C]

#plot auc vs alpha
x = plt.subplot( )
x.plot(C, train_auc, label='AUC train')
x.plot(C, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('C')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal C for which auc is maximum : ',C)
```

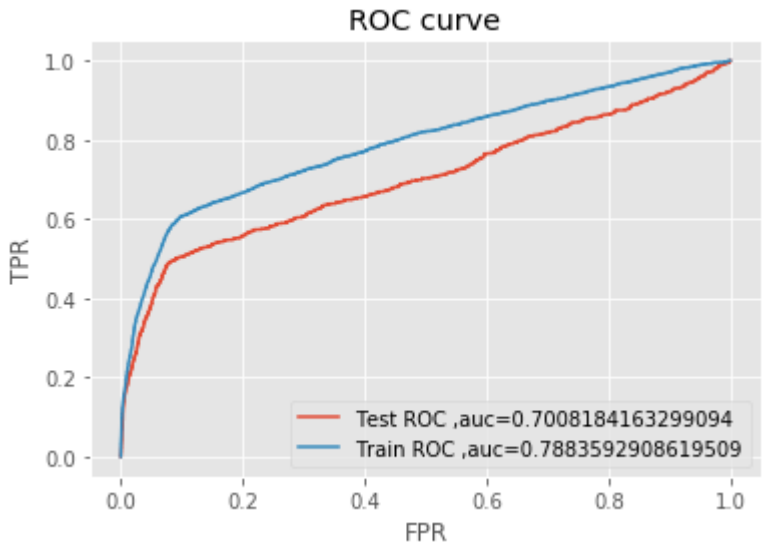



```
optimal C for which auc is maximum :  [9.210340371976184, 6.907755278982137,
4.605170185988092, 2.302585092994046, 0.0, -2.3025850929940455, -4.6051701859
88091, -6.907755278982137, -9.210340371976182]
```

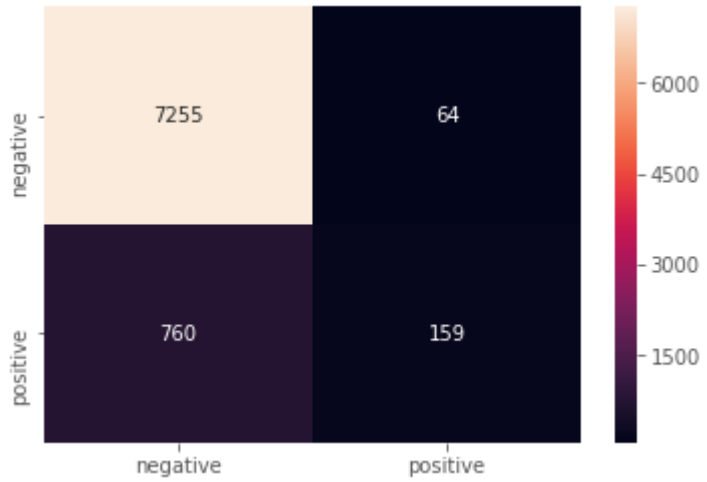
In [ ]:
```
print('optimal C for which auc is maximum : ',optimal_C)
```

```
optimal C for which auc is maximum :  10
```

In [ ]:
```
#Testing AUC on Test data
model =SVC(C = optimal_C)
clf = CalibratedClassifierCV(model, cv=3)
clf.fit(ohe_x_train,y_train)
pred_test = clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.7008184163299094
AUC on Train data is 0.7883592908619509
---------------------------
```



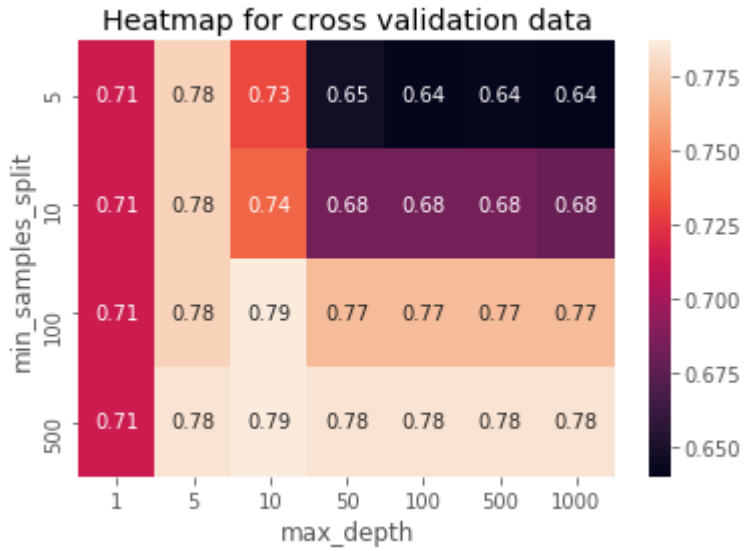
In [ ]:
```
new = ['RBF SVM','SVC',"alpha = 10 ",0.7883,0.7008]
results1.loc[5] = new
```

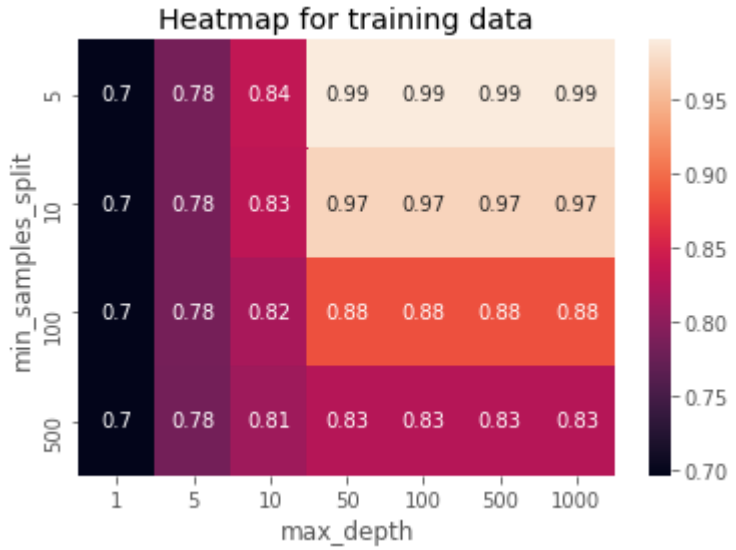
Applying Decision Tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
min_samples =  [5, 10, 100, 500]


param_grid={'min_samples_split':min_samples , 'max_depth':dept}
clf = DecisionTreeClassifier()
model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(ohe_x_train,y_train)
print("optimal min_samples_split",model.best_estimator_.min_samples_split)
print("optimal max_depth",model.best_estimator_.max_depth)
```

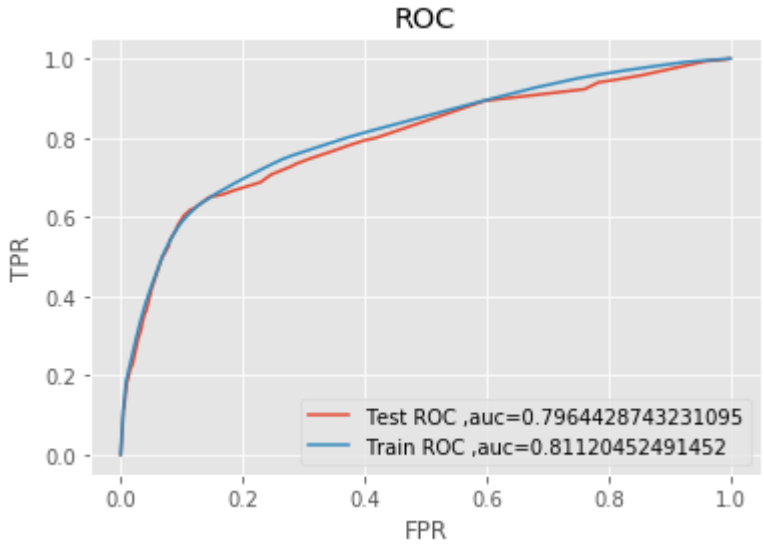
```
optimal min_samples_split 500
optimal max_depth 10
```

In [ ]:
```
import seaborn as sns
X = []
Y = []
cv_auc = []
train_auc = []
for n in min_samples:
    for d in dept:
        clf = DecisionTreeClassifier(max_depth = d,min_samples_split = n)
        clf.fit(ohe_x_train,y_train)
        pred_cv = clf.predict_proba(ohe_x_cv)[:,1]
        pred_train = clf.predict_proba(ohe_x_train)[:,1]
        X.append(n)
        Y.append(d)
        cv_auc.append(roc_auc_score(y_cv,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))

#Heatmap for cross validation data
data = pd.DataFrame({'min_samples_split': X, 'max_depth': Y, 'AUC': cv_auc})
data_pivoted = data.pivot("min_samples_split", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()

#Heatmap for training data
data = pd.DataFrame({'min_samples_split': X, 'max_depth': Y, 'AUC': train_auc
})
data_pivoted = data.pivot("min_samples_split", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for training data')
plt.show()
```

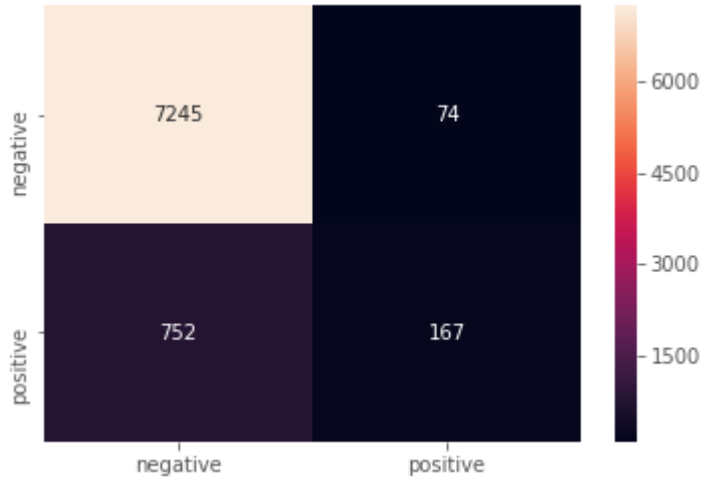

Heatmap for cross validation data
min_samples_split
5
10
100
500
0.71 0.78 0.73 0.65 0.64 0.64 0.64
0.71 0.78 0.74 0.68 0.68 0.68 0.68
0.71 0.78 0.79 0.77 0.77 0.77 0.77
0.71 0.78 0.79 0.78 0.78 0.78 0.78
1 5 10 50 100 500 1000
max_depth
0.775
0.750
0.725
0.700
0.675
0.650


Heatmap for training data
min_samples_split
5
10
100
500
0.7 0.78 0.84 0.99 0.99 0.99 0.99
0.7 0.78 0.83 0.97 0.97 0.97 0.97
0.7 0.78 0.82 0.88 0.88 0.88 0.88
0.7 0.78 0.81 0.83 0.83 0.83 0.83
1 5 10 50 100 500 1000
max_depth
0.95
0.90
0.85
0.80
0.75
0.70

In [ ]:
```
#training our model for max_depth=50,min_samples_split=500
clf = DecisionTreeClassifier(max_depth = 10,min_samples_split = 500)
clf.fit(ohe_x_train,y_train)
pred_test =clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2=metrics.roc_curve(y_train,pred_train)

#ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.7964428743231095
AUC on Train data is 0.81120452491452
---------------------------
```



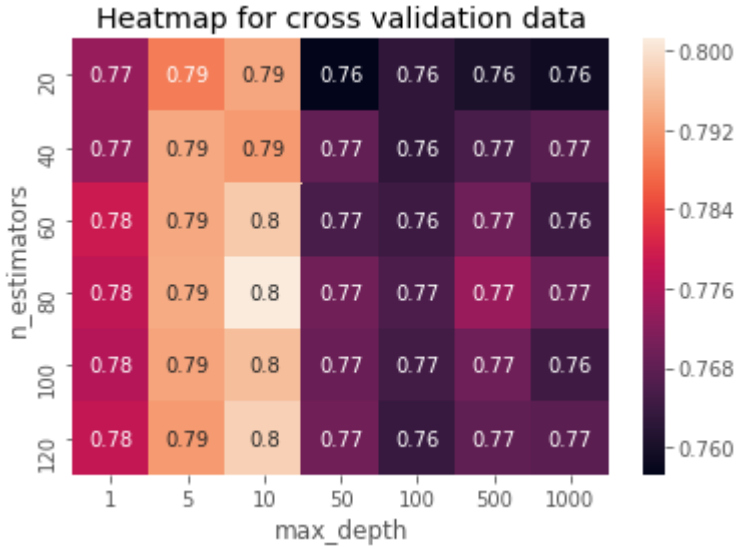
In [ ]:
```
new = ['Decision Tree','DecisionTreeClassifier',"max_depth = 10 & min_samples_
split = 500",0.8112,0.7964]
results1.loc[6] = new
```

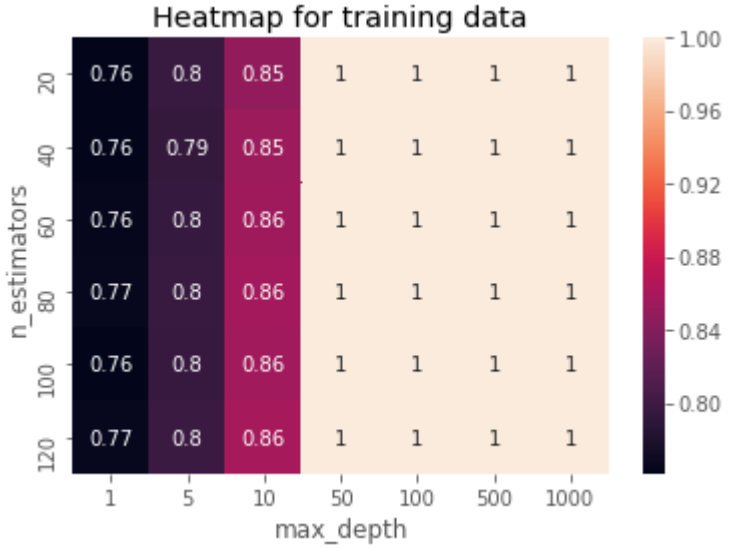
Applying Random Forest

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
n_estimators =  [20, 40, 60, 80, 100, 120]


param_grid={'n_estimators':n_estimators , 'max_depth':dept}
clf = RandomForestClassifier()
model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(ohe_x_train,y_train)
print("optimal n_estimators",model.best_estimator_.n_estimators)
print("optimal max_depth",model.best_estimator_.max_depth)
```

```
optimal n_estimators 60
optimal max_depth 10
```

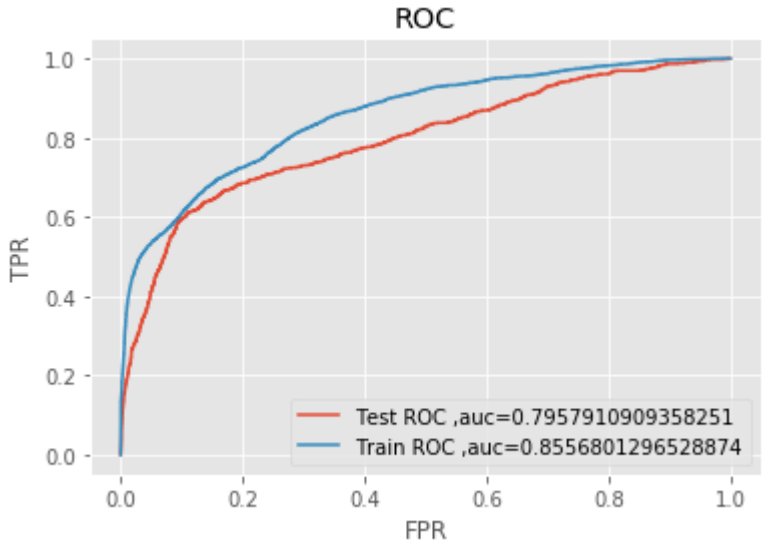
In [ ]:
```
import seaborn as sns
X = []
Y = []
cv_auc = []
train_auc = []
for n in n_estimators:
    for d in dept:
        clf = RandomForestClassifier(max_depth = d,n_estimators = n)
        clf.fit(ohe_x_train,y_train)
        pred_cv = clf.predict_proba(ohe_x_cv)[:,1]
        pred_train = clf.predict_proba(ohe_x_train)[:,1]
        X.append(n)
        Y.append(d)
        cv_auc.append(roc_auc_score(y_cv,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))

#Heatmap for cross validation data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': cv_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()

#Heatmap for training data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': train_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for training data')
plt.show()
```



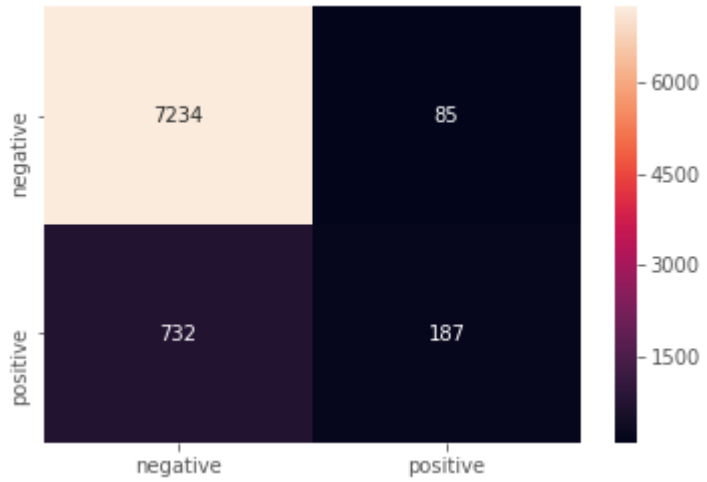


In [ ]:
```
optimal_n_estimators = model.best_estimator_.n_estimators
optimal_max_depth = model.best_estimator_.max_depth
```

In [ ]:
```
#training our model for max_depth=1000,n_estimators = 120
clf = RandomForestClassifier(max_depth = optimal_max_depth,n_estimators = opti
mal_n_estimators)
clf.fit(ohe_x_train,y_train)
pred_test =clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2=metrics.roc_curve(y_train,pred_train)

#ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.7957910909358251
AUC on Train data is 0.8556801296528874
---------------------------
```



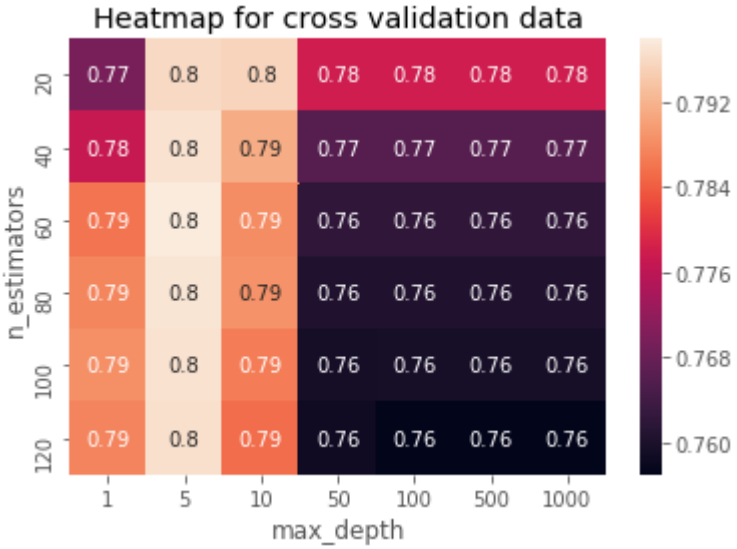
In [ ]:
```
new = ['Random Forest','RandomForestClassifier',"max_depth = 10 & min_samples_
split = 60",0.8556,0.7957]
results1.loc[7] = new
```

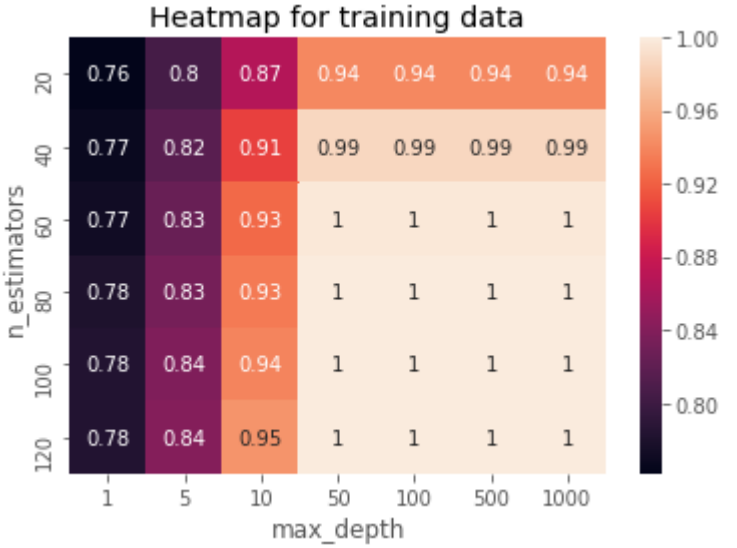
Applying XGBOOST

In [ ]:
```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
n_estimators =  [20, 40, 60, 80, 100, 120]


param_grid={'n_estimators':n_estimators , 'max_depth':dept}
clf = XGBClassifier()
model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(ohe_x_train,y_train)
print("optimal n_estimators",model.best_estimator_.n_estimators)
print("optimal max_depth",model.best_estimator_.max_depth)

optimal_n_estimators = model.best_estimator_.n_estimators
optimal_max_depth = model.best_estimator_.max_depth
```

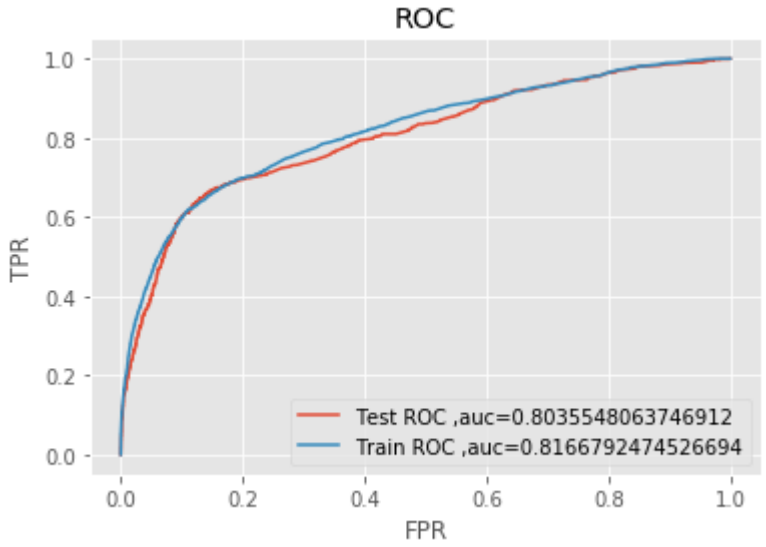
```
optimal n_estimators 40
optimal max_depth 5
```

In [ ]:
```
import seaborn as sns
X = []
Y = []
cv_auc = []
train_auc = []
for n in n_estimators:
    for d in dept:
        clf = XGBClassifier(max_depth = d,n_estimators = n)
        clf.fit(ohe_x_train,y_train)
        pred_cv = clf.predict_proba(ohe_x_cv)[:,1]
        pred_train = clf.predict_proba(ohe_x_train)[:,1]
        X.append(n)
        Y.append(d)
        cv_auc.append(roc_auc_score(y_cv,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))
optimal_depth=Y[cv_auc.index(max(cv_auc))]
optimal_n_estimator=X[cv_auc.index(max(cv_auc))]

#Heatmap for cross validation data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': cv_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()

#Heatmap for training data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': train_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for training data')
plt.show()
```

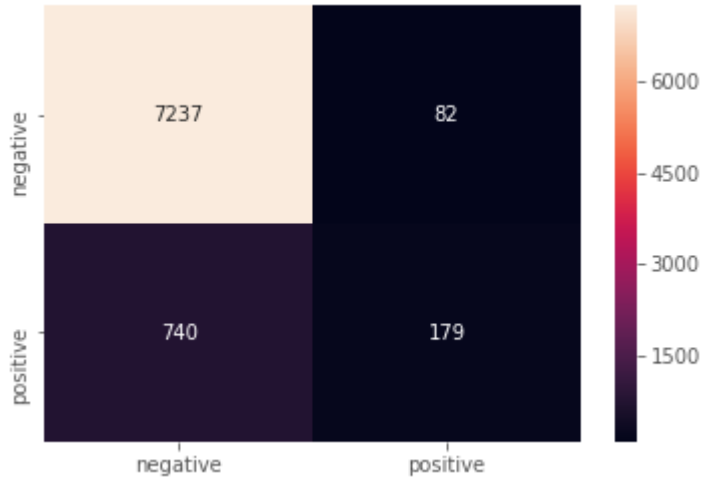

Heatmap for cross validation data
n_estimators
20
40
60
80
100
120
0.77 0.8 0.8 0.78 0.78 0.78 0.78
0.78 0.8 0.79 0.77 0.77 0.77 0.77
0.79 0.8 0.79 0.76 0.76 0.76 0.76
0.79 0.8 0.79 0.76 0.76 0.76 0.76
0.79 0.8 0.79 0.76 0.76 0.76 0.76
0.79 0.8 0.79 0.76 0.76 0.76 0.76
1 5 10 50 100 500 1000
max_depth
0.792
0.784
0.776
0.768
0.760


Heatmap for training data
n_estimators
20
40
60
80
100
120
0.76 0.8 0.87 0.94 0.94 0.94 0.94
0.77 0.82 0.91 0.99 0.99 0.99 0.99
0.77 0.83 0.93 1 1 1 1
0.78 0.83 0.93 1 1 1 1
0.78 0.84 0.94 1 1 1 1
0.78 0.84 0.95 1 1 1 1
1 5 10 50 100 500 1000
max_depth
1.00
0.96
0.92
0.88
0.84
0.80

In [ ]:
```
#training our model for max_depth=50,min_samples_split=500
clf = XGBClassifier(max_depth = 5,n_estimators = 40)
clf.fit(ohe_x_train,y_train)
pred_test =clf.predict_proba(ohe_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(ohe_x_train)[:,1]
fpr2,tpr2,thresholds2=metrics.roc_curve(y_train,pred_train)

#ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.8035548063746912
AUC on Train data is 0.8166792474526694
---------------------------
```



In [ ]:
```
new = ['XGBOOST','XGBClassifier',"max_depth = 5 & n_estimators = 40",0.8166,0.
8035]
results1.loc[8] = new
```

Response-Coding

In [ ]:
```
#splitting the data
x_train, x_test, y_train, y_test = model_selection.train_test_split(data, y, t
est_size=0.2, random_state=0)
x_train, x_cv, y_train, y_cv = model_selection.train_test_split(x_train, y_tra
in, test_size=0.2)
```

In [ ]:
```
print("the shape of x_train data,{} and shape of x_test data {}:".format(x_train.shape,x_test.shape))
print("the shape of y_train data,{} and shape of y_test data {}:".format(y_train.shape,y_test.shape))
print("the shape of x_cv data,{} and shape of y_cv data {}:".format(x_cv.shape,y_cv.shape))
```

```
the shape of x_train data,(26360, 14) and shape of x_test data (8238, 14):
the shape of y_train data,(26360,) and shape of y_test data (8238,):
the shape of x_cv data,(6590, 14) and shape of y_cv data (6590,):
```

In [ ]:
```
#Numerical features
real_feature_x_train_age = ['age']
real_feature_x_train_campaign = ['campaign']
real_feature_x_train_previous = ['previous']
real_feature_x_train_emp_var_rate = ['emp.var.rate']
real_feature_x_train_cons_price_idx = ['cons.price.idx']
real_feature_x_train_cons_conf_idx = ['cons.conf.idx']
real_feature_x_train_euribor3m = ['euribor3m']
real_feature_x_train_nr_employed = ['nr.employed']

real_feature_x_test_age = ['age']
real_feature_x_test_campaign = ['campaign']
real_feature_x_test_previous = ['previous']
real_feature_x_test_emp_var_rate = ['emp.var.rate']
real_feature_x_test_cons_price_idx = ['cons.price.idx']
real_feature_x_test_cons_conf_idx = ['cons.conf.idx']
real_feature_x_test_euribor3m = ['euribor3m']
real_feature_x_test_nr_employed = ['nr.employed']

real_feature_x_cv_age = ['age']
real_feature_x_cv_campaign = ['campaign']
real_feature_x_cv_previous = ['previous']
real_feature_x_cv_emp_var_rate = ['emp.var.rate']
real_feature_x_cv_cons_price_idx = ['cons.price.idx']
real_feature_x_cv_cons_conf_idx = ['cons.conf.idx']
real_feature_x_cv_euribor3m = ['euribor3m']
real_feature_x_cv_nr_employed = ['nr.employed']
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
SS = StandardScaler()
SS.fit(x_train[real_feature_x_train_age])
x_train_real_feature_age_scale = SS.transform(x_train[real_feature_x_train_age])
x_test_real_feature_age_scale = SS.transform(x_test[real_feature_x_test_age])
x_cv_real_feature_age_scale = SS.transform(x_cv[real_feature_x_cv_age])
```

In [ ]:
```
SS.fit(x_train[real_feature_x_train_campaign])
x_train_real_feature_campaign_scale = SS.transform(x_train[real_feature_x_train_campaign])
x_test_real_feature_campaign_scale = SS.transform(x_test[real_feature_x_test_campaign])
x_cv_real_feature_campaign_scale = SS.transform(x_cv[real_feature_x_cv_campaign])
```

In [ ]:
```
SS.fit(x_train[real_feature_x_train_previous])
x_train_real_feature_previous_scale = SS.transform(x_train[real_feature_x_train_previous])
x_test_real_feature_previous_scale = SS.transform(x_test[real_feature_x_test_previous])
x_cv_real_feature_previous_scale = SS.transform(x_cv[real_feature_x_cv_previous])
```

In [ ]:
```
SS.fit(x_train[real_feature_x_train_emp_var_rate])
x_train_real_feature_emp_var_rate_scale = SS.transform(x_train[real_feature_x_train_emp_var_rate])
x_test_real_feature_emp_var_rate_scale = SS.transform(x_test[real_feature_x_test_emp_var_rate])
x_cv_real_feature_emp_var_rate_scale = SS.transform(x_cv[real_feature_x_cv_emp_var_rate])
```

In [ ]:
```
SS.fit(x_train[real_feature_x_train_cons_price_idx])
x_train_real_feature_cons_price_idx_scale = SS.transform(x_train[real_feature_x_train_cons_price_idx])
x_test_real_feature_cons_price_idx_scale = SS.transform(x_test[real_feature_x_test_cons_price_idx])
x_cv_real_feature_cons_price_idx_scale = SS.transform(x_cv[real_feature_x_cv_cons_price_idx])
```

In [ ]:
```
SS.fit(x_train[real_feature_x_train_cons_conf_idx])
x_train_real_feature_cons_conf_idx_scale = SS.transform(x_train[real_feature_x_train_cons_conf_idx])
x_test_real_feature_cons_conf_idx_scale = SS.transform(x_test[real_feature_x_test_cons_conf_idx])
x_cv_real_feature_cons_conf_idx_scale = SS.transform(x_cv[real_feature_x_cv_cons_conf_idx])
```

In [ ]:
```
SS.fit(x_train[real_feature_x_train_euribor3m])
x_train_real_feature_euribor3m_scale = SS.transform(x_train[real_feature_x_train_euribor3m])
x_test_real_feature_euribor3m_scale = SS.transform(x_test[real_feature_x_test_euribor3m])
x_cv_real_feature_euribor3m_scale = SS.transform(x_cv[real_feature_x_cv_euribor3m])
```

In [ ]:
```
SS.fit(x_train[real_feature_x_train_nr_employed])
x_train_real_feature_nr_employed_scale = SS.transform(x_train[real_feature_x_t
rain_nr_employed])
x_test_real_feature_nr_employed_scale = SS.transform(x_test[real_feature_x_tes
t_nr_employed])
x_cv_real_feature_nr_employed_scale = SS.transform(x_cv[real_feature_x_cv_nr_e
mployed])
```

In [ ]:
```
real_feature_x_train = np.concatenate((x_train_real_feature_age_scale,x_train_
real_feature_campaign_scale,x_train_real_feature_previous_scale,x_train_real_f
eature_emp_var_rate_scale,x_train_real_feature_cons_price_idx_scale,x_train_re
al_feature_cons_conf_idx_scale,x_train_real_feature_euribor3m_scale,x_train_re
al_feature_nr_employed_scale),axis = 1)
real_feature_x_test = np.concatenate((x_test_real_feature_age_scale,x_test_rea
l_feature_campaign_scale,x_test_real_feature_previous_scale,x_test_real_featur
e_emp_var_rate_scale,x_test_real_feature_cons_price_idx_scale,x_test_real_feat
ure_cons_conf_idx_scale,x_test_real_feature_euribor3m_scale,x_test_real_featur
e_nr_employed_scale),axis = 1)
real_feature_x_cv = np.concatenate((x_cv_real_feature_age_scale,x_cv_real_feat
ure_campaign_scale,x_cv_real_feature_previous_scale,x_cv_real_feature_emp_var_
rate_scale,x_cv_real_feature_cons_price_idx_scale,x_cv_real_feature_cons_conf_
idx_scale,x_cv_real_feature_euribor3m_scale,x_cv_real_feature_nr_employed_scal
e),axis = 1)
```

In [ ]:
```
def get_gv_fea_dict(alpha, feature, df):
    value_count = x_train[feature].value_counts()

    gv_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,3):
            cls_cnt = x_train.loc[(y_train==k) & (x_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 20*alpha
))
        gv_dict[i]=vec
    return gv_dict

def get_gv_feature(alpha, feature, df):

    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = x_train[feature].value_counts()


    gv_fea = []
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/2,1/2])
    return gv_fea
```

In [ ]:
```
#response-coding of the job feature
# alpha is used for laplace smoothing
alpha = 1
# train job feature
train_job_feature_responseCoding = np.array(get_gv_feature(alpha, 'job', x_tra
in))
# test job feature
test_job_feature_responseCoding = np.array(get_gv_feature(alpha, 'job', x_test
))
# cross validation job feature
cv_job_feature_responseCoding = np.array(get_gv_feature(alpha, 'job', x_cv))
```

In [ ]:
```
train_job_feature_responseCoding.shape
```

Out[ ]:
```
(26360, 2)
```

In [ ]:
```
#response-coding of the marital feature
# alpha is used for laplace smoothing
alpha = 1
# train marital feature
train_marital_feature_responseCoding = np.array(get_gv_feature(alpha, 'marita
l', x_train))
# test marital feature
test_marital_feature_responseCoding = np.array(get_gv_feature(alpha, 'marital'
, x_test))
# cross validation marital feature
cv_marital_feature_responseCoding = np.array(get_gv_feature(alpha, 'marital',
x_cv))
```

In [ ]:
```
train_marital_feature_responseCoding.shape
```

Out[ ]:
```
(26360, 2)
```

In [ ]:
```
#response-coding of the education feature
# alpha is used for laplace smoothing
alpha = 1
# train education feature
train_education_feature_responseCoding = np.array(get_gv_feature(alpha, 'educa
tion', x_train))
# test education feature
test_education_feature_responseCoding = np.array(get_gv_feature(alpha, 'educat
ion', x_test))
# cross validation education feature
cv_education_feature_responseCoding = np.array(get_gv_feature(alpha, 'educatio
n', x_cv))
```

In [ ]:
```
#response-coding of the housing feature
# alpha is used for laplace smoothing
alpha = 1
# train housing feature
train_housing_feature_responseCoding = np.array(get_gv_feature(alpha, 'housin
g', x_train))
# test housing feature
test_housing_feature_responseCoding = np.array(get_gv_feature(alpha, 'housing'
, x_test))
# cross validation housing feature
cv_housing_feature_responseCoding = np.array(get_gv_feature(alpha, 'housing',
x_cv))
```

In [ ]:
```
#response-coding of the loan feature
# alpha is used for laplace smoothing
alpha = 1
# train loan feature
train_loan_feature_responseCoding = np.array(get_gv_feature(alpha, 'loan', x_t
rain))
# test loan feature
test_loan_feature_responseCoding = np.array(get_gv_feature(alpha, 'loan', x_te
st))
# cross validation loan feature
cv_loan_feature_responseCoding = np.array(get_gv_feature(alpha, 'loan', x_cv))
```

In [ ]:
```
#response-coding of the poutcome feature
# alpha is used for laplace smoothing
alpha = 1
# train poutcome feature
train_poutcome_feature_responseCoding = np.array(get_gv_feature(alpha, 'poutco
me', x_train))
# test poutcome feature
test_poutcome_feature_responseCoding = np.array(get_gv_feature(alpha, 'poutcom
e', x_test))
# cross validation poutcome feature
cv_poutcome_feature_responseCoding = np.array(get_gv_feature(alpha, 'poutcome'
, x_cv))
```

In [ ]:
```
import numpy as np
cat_x_train = np.concatenate((train_job_feature_responseCoding,train_marital_f
eature_responseCoding,train_education_feature_responseCoding,train_housing_fea
ture_responseCoding,train_loan_feature_responseCoding,train_poutcome_feature_r
esponseCoding),axis = 1)
```

In [ ]:
```
cat_x_test = np.concatenate((test_job_feature_responseCoding,test_marital_feat
ure_responseCoding,test_education_feature_responseCoding,test_housing_feature_
responseCoding,test_loan_feature_responseCoding,test_poutcome_feature_response
Coding),axis = 1)
cat_x_cv = np.concatenate((cv_job_feature_responseCoding,cv_marital_feature_re
sponseCoding,cv_education_feature_responseCoding,cv_housing_feature_responseCo
ding,cv_loan_feature_responseCoding,cv_poutcome_feature_responseCoding),axis =
1)
```

In [ ]:
```
cat_x_train.shape
```

Out[ ]: (26360, 12)

In [ ]:
```
cat_x_test.shape
```

Out[ ]: (8238, 12)

In [ ]:
```
cat_x_cv.shape
```

Out[ ]: (6590, 12)

In [ ]:
```
real_feature_x_train.shape
```

Out[ ]: (26360, 8)

In [ ]:
```
type(real_feature_x_train)
```

Out[ ]: numpy.ndarray

In [ ]:
```
type(cat_x_cv)
```

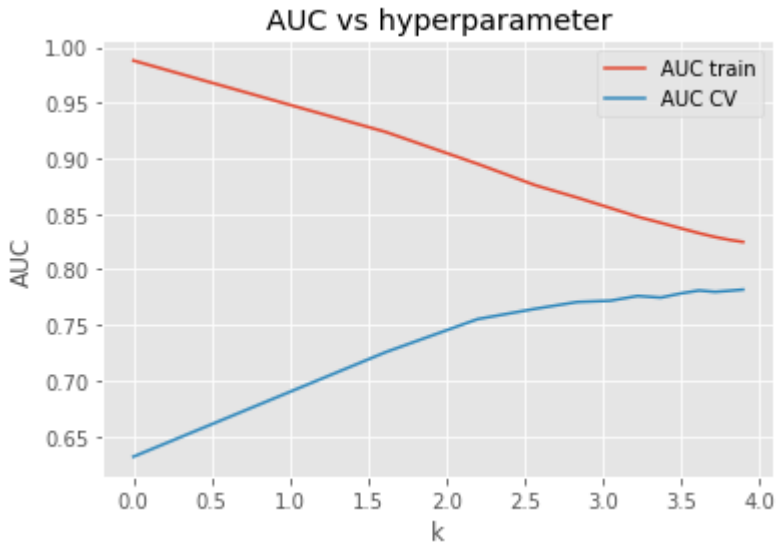
Out[ ]: numpy.ndarray

In [ ]:
```
rc_x_train = np.concatenate((cat_x_train,real_feature_x_train),axis = 1)
rc_x_test = np.concatenate((cat_x_test,real_feature_x_test),axis = 1)
rc_x_cv = np.concatenate((cat_x_cv,real_feature_x_cv),axis = 1)
```

Machine Learning Models

Applying KNN Brute Force Algorithm

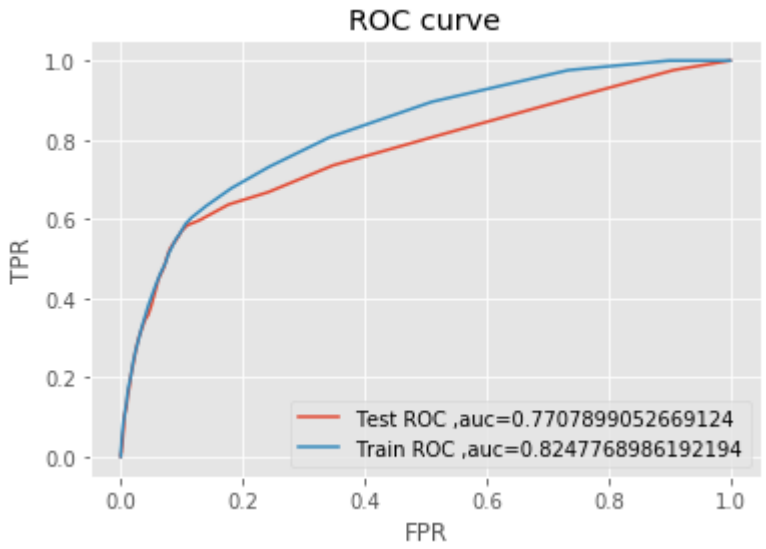
In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import math

k = list(range(1,50,4))

train_auc = []
cv_auc = []

for i in k:
    clf = KNeighborsClassifier(n_neighbors = i,algorithm='brute')
    clf.fit(rc_x_train,y_train)
    prob_cv = clf.predict_proba(rc_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(rc_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_k = k[cv_auc.index(max(cv_auc))]
k = [math.log(x) for x in k]

#plot auc vs alpha
x = plt.subplot( )
x.plot(k, train_auc, label='AUC train')
x.plot(k, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('k')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal alpha for which auc is maximum : ',optimal_k)
```



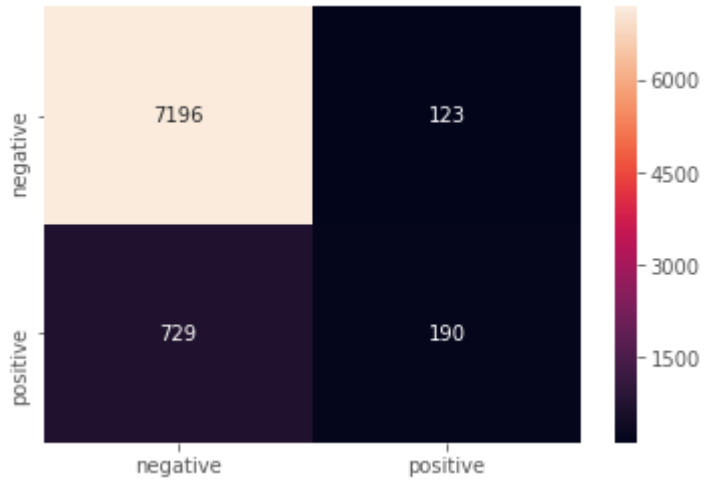

```
optimal alpha for which auc is maximum :  49
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
```

In [ ]:
```
#Testing AUC on Test data
clf = KNeighborsClassifier(n_neighbors = optimal_k,algorithm='brute')
clf.fit(rc_x_train,y_train)
pred_test = clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



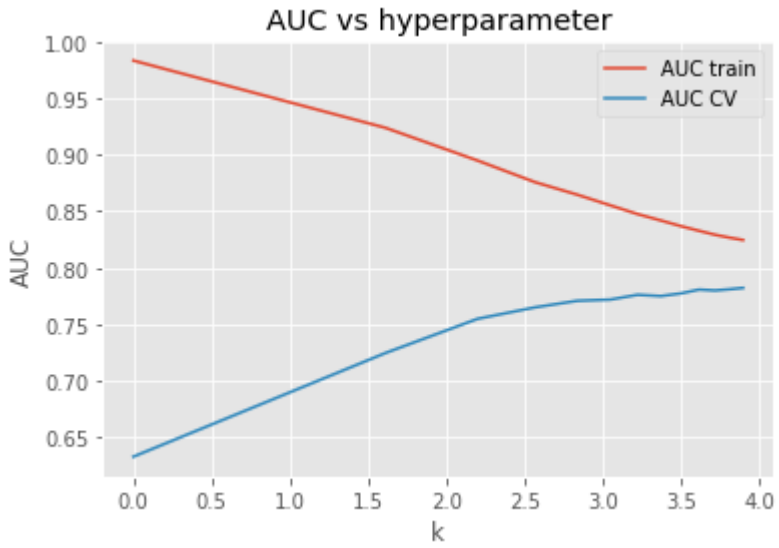

```
AUC on Test data is 0.7707899052669124
AUC on Train data is 0.8247768986192194
---------------------------
```



In [ ]:
```
results2 = pd.DataFrame(columns=['model', 'Classifier' ,"hyper perameter", 'Train-AUC', 'Test-AUC' ])
new = ['KNN with Brute force','KNeighborsClassifier',"k = 49",0.8247,0.7707]
results2.loc[0] = new
```

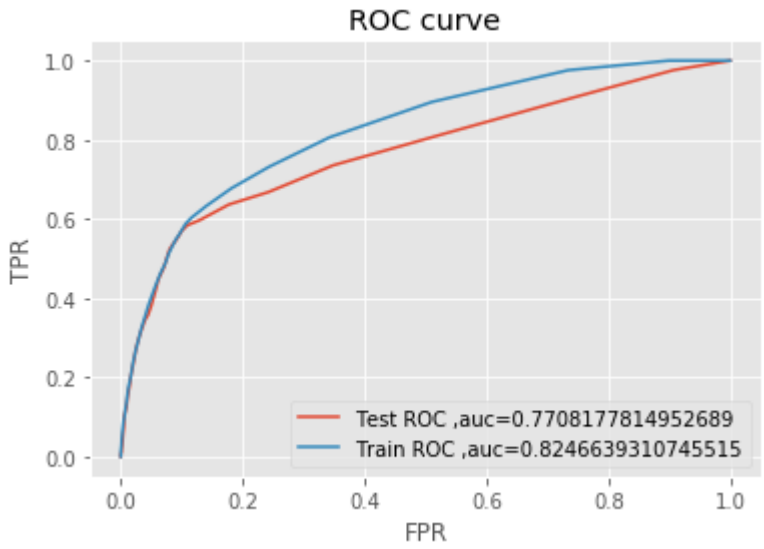
Applying KNN kd-Tree

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import math

k = list(range(1,50,4))

train_auc = []
cv_auc = []

for i in k:
    clf = KNeighborsClassifier(n_neighbors = i,algorithm='kd_tree')
    clf.fit(rc_x_train,y_train)
    prob_cv = clf.predict_proba(rc_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(rc_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_k = k[cv_auc.index(max(cv_auc))]
k = [math.log(x) for x in k]

#plot auc vs alpha
x = plt.subplot( )
x.plot(k, train_auc, label='AUC train')
x.plot(k, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('k')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal alpha for which auc is maximum : ',optimal_k)
```

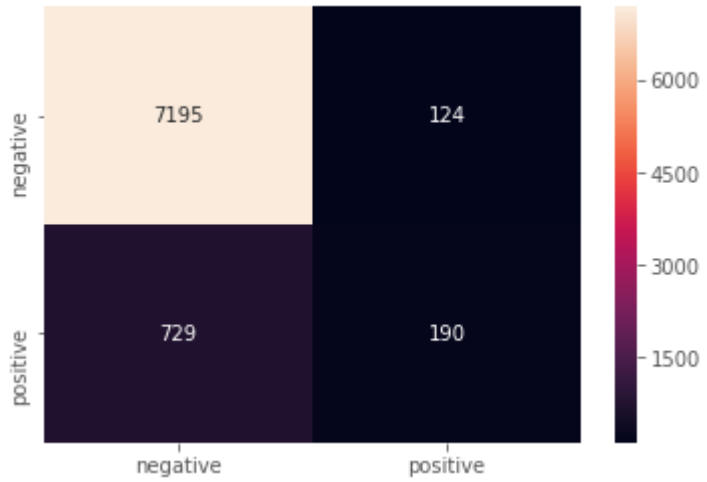



```
optimal alpha for which auc is maximum :  49
```

In [ ]:
```
#Testing AUC on Test data
clf = KNeighborsClassifier(n_neighbors = optimal_k,algorithm='kd_tree')
clf.fit(rc_x_train,y_train)
pred_test = clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



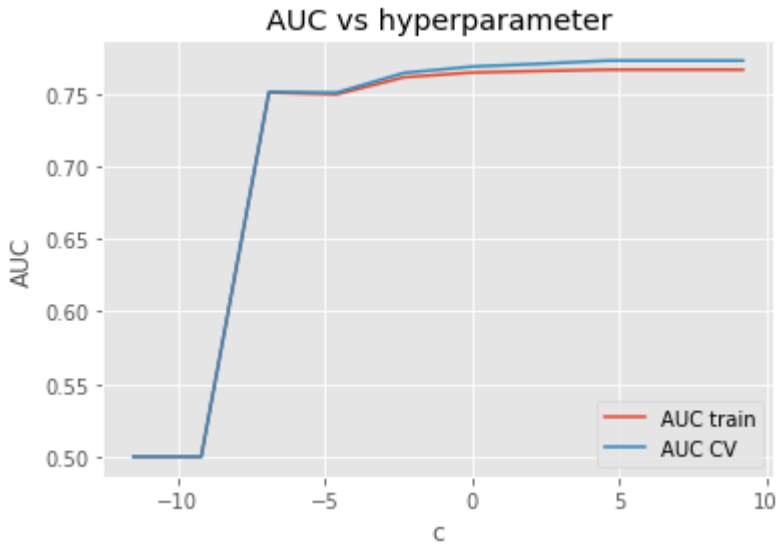

```
AUC on Test data is 0.7708177814952689
AUC on Train data is 0.8246639310745515
---------------------------
```



In [ ]:
```
new = ['KNN with kd-tree','KNeighborsClassifier',"k = 49",0.8246,0.7708]
results2.loc[1] = new
```

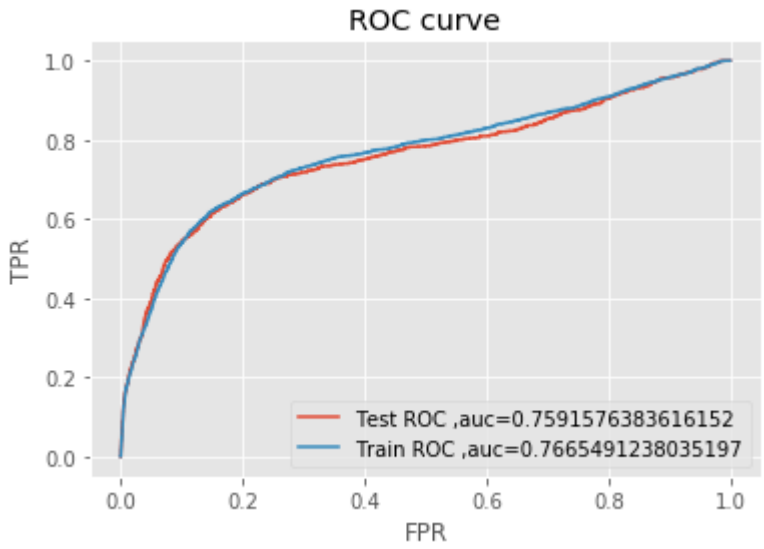
Applying Logistic Regression with L1 regularization

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import math

c = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001]

train_auc = []
cv_auc = []

for i in c:
    clf = LogisticRegression(penalty='l1',C=i)
    clf.fit(rc_x_train,y_train)
    prob_cv = clf.predict_proba(rc_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(rc_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_c= c[cv_auc.index(max(cv_auc))]
c = [math.log(x) for x in c]

#plot auc vs alpha
x = plt.subplot( )
x.plot(c, train_auc, label='AUC train')
x.plot(c, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('c')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal c for which auc is maximum : ',optimal_c)
```

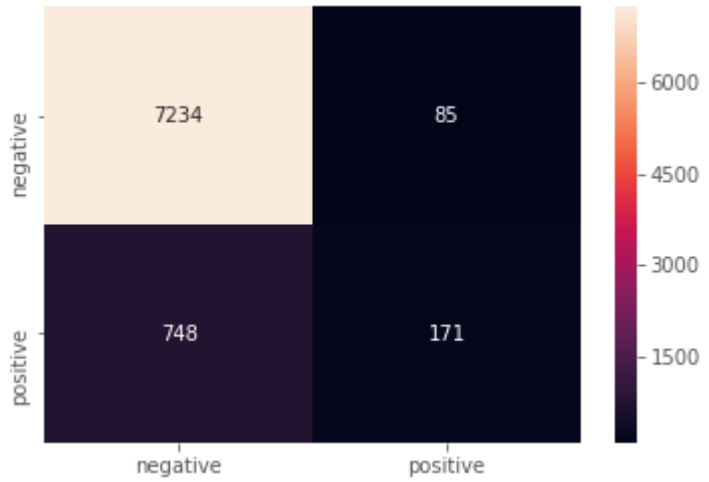



```
optimal c for which auc is maximum :  1000
```

In [ ]:
```
#Testing AUC on Test data
clf = LogisticRegression(penalty='l1',C=optimal_c)
clf.fit(rc_x_train,y_train)
pred_test = clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)


#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
from sklearn.metrics import confusion_matrix
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



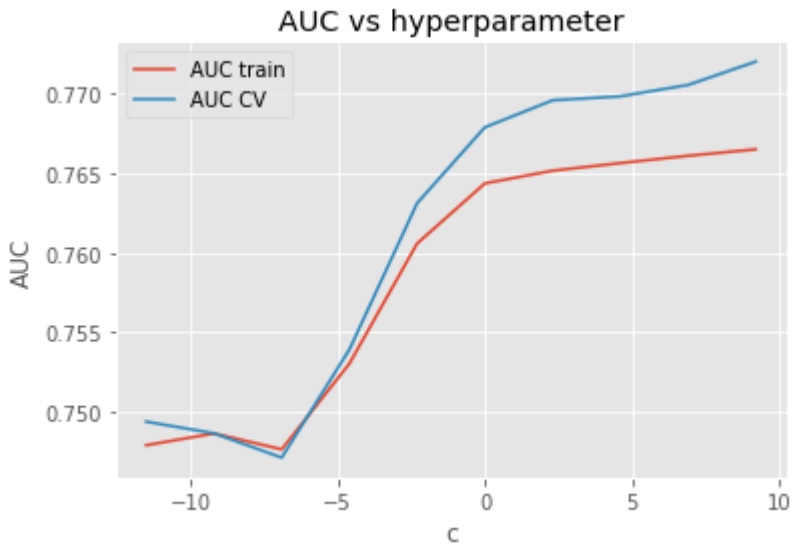

```
AUC on Test data is 0.7591576383616152
AUC on Train data is 0.7665491238035197
---------------------------
```



In [ ]:
```
new = ['Logistic Regression with L1','LogisticRegression',"c = 1000",0.7665,0.
7591]
results2.loc[2] = new
```

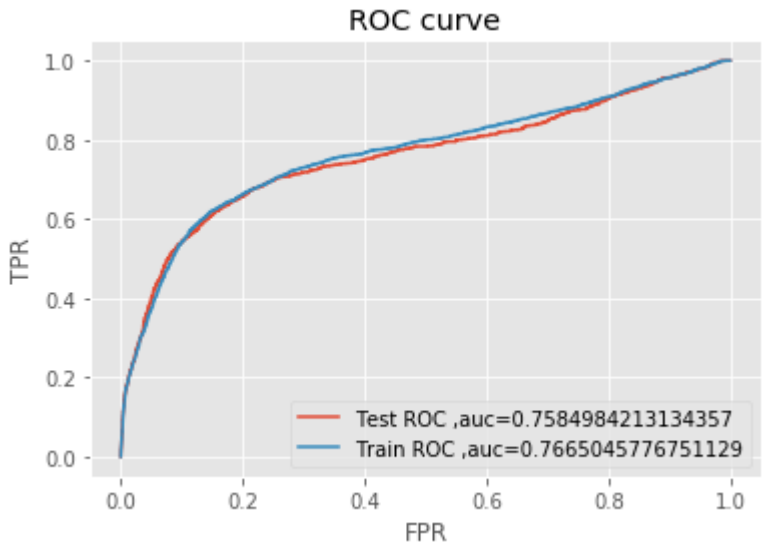
Applying Logistic Regression with L2 regularization

In [ ]:
```
c = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001]

train_auc = []
cv_auc = []

for i in c:
    clf = LogisticRegression(penalty='l2',C=i)
    clf.fit(rc_x_train,y_train)
    prob_cv = clf.predict_proba(rc_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(rc_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_c= c[cv_auc.index(max(cv_auc))]
c = [math.log(x) for x in c]

#plot auc vs alpha
x = plt.subplot( )
x.plot(c, train_auc, label='AUC train')
x.plot(c, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('c')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal c for which auc is maximum : ',optimal_c)
```

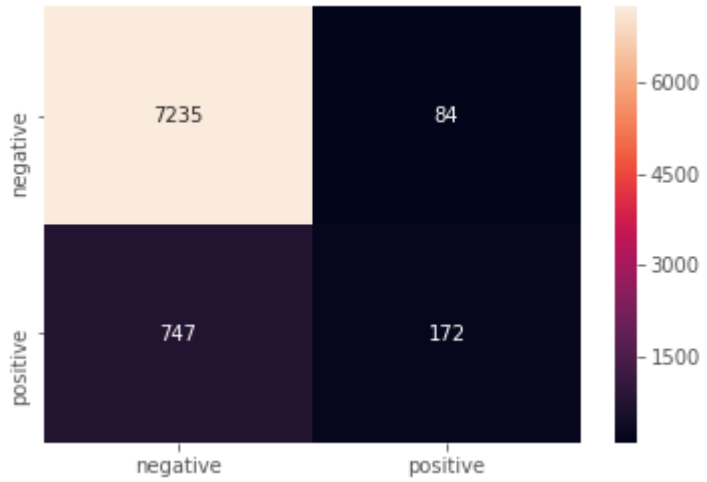



```
optimal c for which auc is maximum :  10000
```

In [ ]:
```
#Testing AUC on Test data
clf = LogisticRegression(penalty='l2',C=optimal_c)
clf.fit(rc_x_train,y_train)
pred_test = clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)


#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.7584984213134357
AUC on Train data is 0.7665045776751129
---------------------------
```

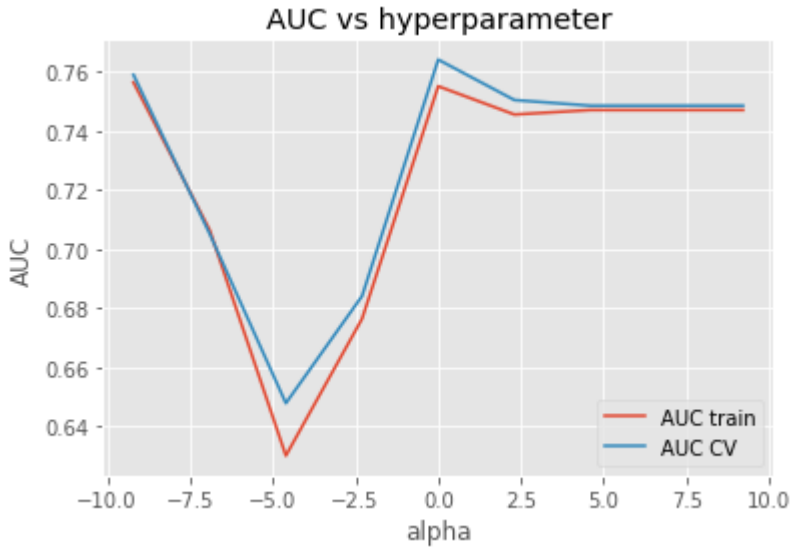


In [ ]:
```
new = ['Logistic Regression with L2','LogisticRegression',"c = 10000",0.7665,
0.7584]
results2.loc[3] = new
```

Applying Linear SVM

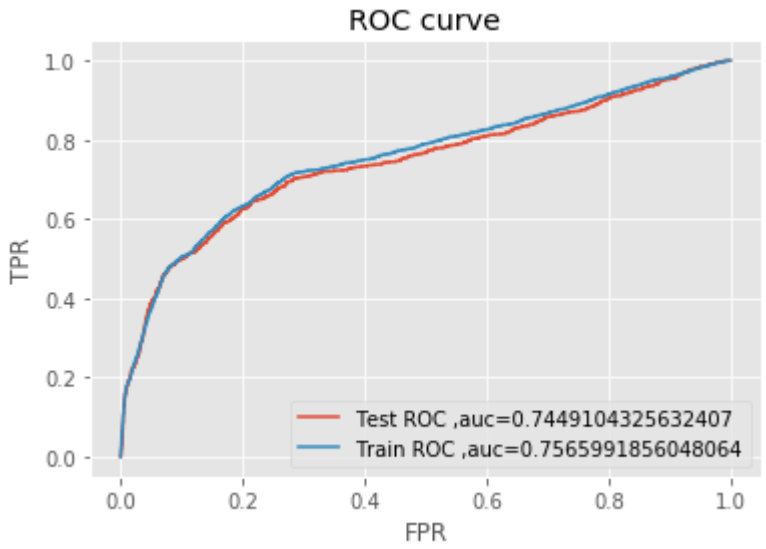
In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import math

alpha = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001]

train_auc = []
cv_auc = []

for i in alpha:
    model = SGDClassifier(alpha=i, loss = "hinge")
    clf = CalibratedClassifierCV(model, cv=3)
    clf.fit(rc_x_train,y_train)
    prob_cv = clf.predict_proba(rc_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(rc_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_alpha= alpha[cv_auc.index(max(cv_auc))]
alpha=[math.log(x) for x in alpha]

#plot auc vs alpha
x = plt.subplot( )
x.plot(alpha, train_auc, label='AUC train')
x.plot(alpha, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('alpha')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

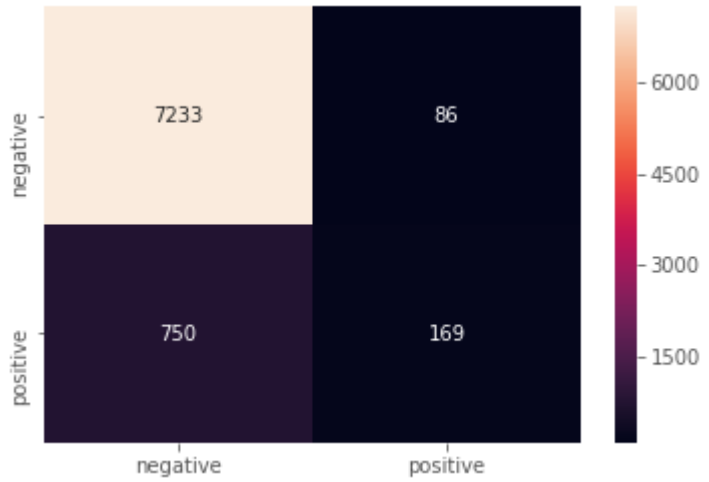



```
optimal alpha for which auc is maximum :  1
```

In [ ]:
```
#Testing AUC on Test data
model = SGDClassifier(alpha = optimal_alpha)
clf = CalibratedClassifierCV(model, cv=3)
clf.fit(rc_x_train,y_train)
pred_test = clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



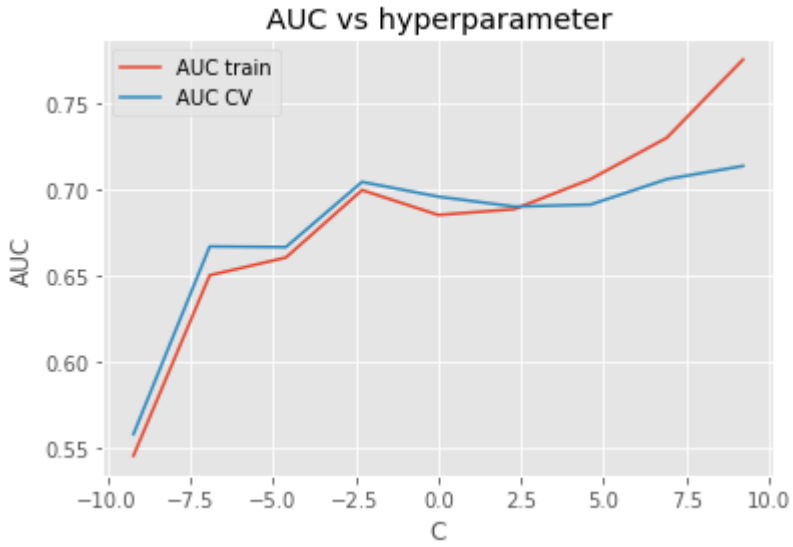

```
AUC on Test data is 0.7449104325632407
AUC on Train data is 0.7565991856048064
---------------------------
```



In [ ]:
```
new = ['Linear SVM','SGDClassifier',"alpha = 1",0.7565,0.7449]
results2.loc[4] = new
```

Applying RBF SVM

In [ ]:
```
from sklearn.svm import SVC
C = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001]

train_auc = []
cv_auc = []

for i in C:
    model = SVC(C=i)
    clf = CalibratedClassifierCV(model, cv=3)
    clf.fit(rc_x_train,y_train)
    prob_cv = clf.predict_proba(rc_x_cv)[:,1]
    cv_auc.append(roc_auc_score(y_cv,prob_cv))
    prob_train = clf.predict_proba(rc_x_train)[:,1]
    train_auc.append(roc_auc_score(y_train,prob_train))
optimal_C= C[cv_auc.index(max(cv_auc))]
C=[math.log(x) for x in C]

#plot auc vs alpha
x = plt.subplot( )
x.plot(C, train_auc, label='AUC train')
x.plot(C, cv_auc, label='AUC CV')
plt.title('AUC vs hyperparameter')
plt.xlabel('C')
plt.ylabel('AUC')
x.legend()
plt.show()

print('optimal C for which auc is maximum : ',C)
```

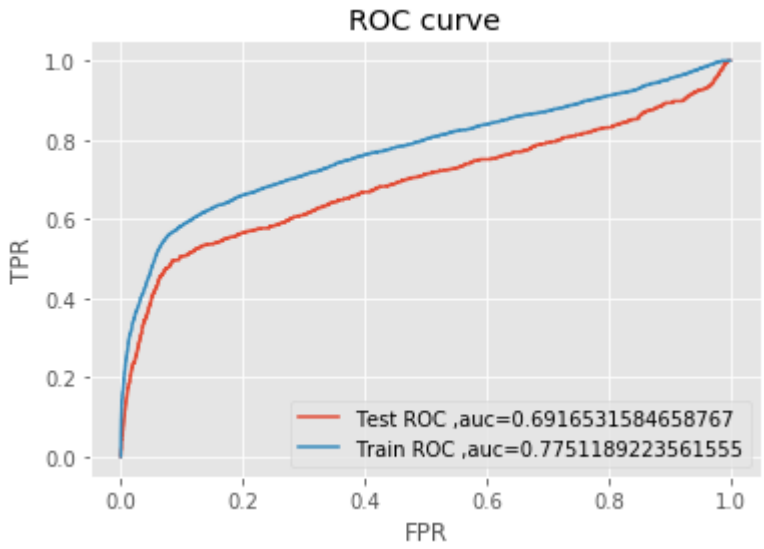



```
optimal C for which auc is maximum :  [9.210340371976184, 6.907755278982137,
4.605170185988092, 2.302585092994046, 0.0, -2.3025850929940455, -4.6051701859
88091, -6.907755278982137, -9.210340371976182]
```

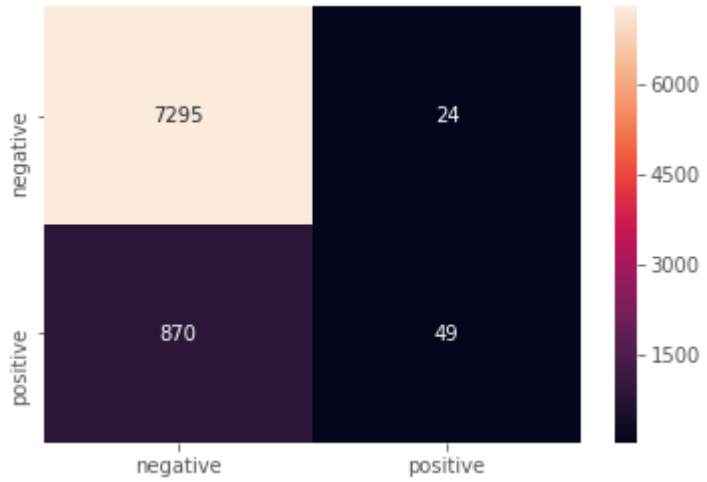
In [ ]:
```
print('optimal C for which auc is maximum : ',optimal_C)
```

```
optimal C for which auc is maximum :  10000
```

In [ ]:
```
#Testing AUC on Test data
model =SVC(C = optimal_C)
clf = CalibratedClassifierCV(model, cv=3)
clf.fit(rc_x_train,y_train)
pred_test = clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2 = metrics.roc_curve(y_train,pred_train)

#plot ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label ='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.6916531584658767
AUC on Train data is 0.7751189223561555
---------------------------
```



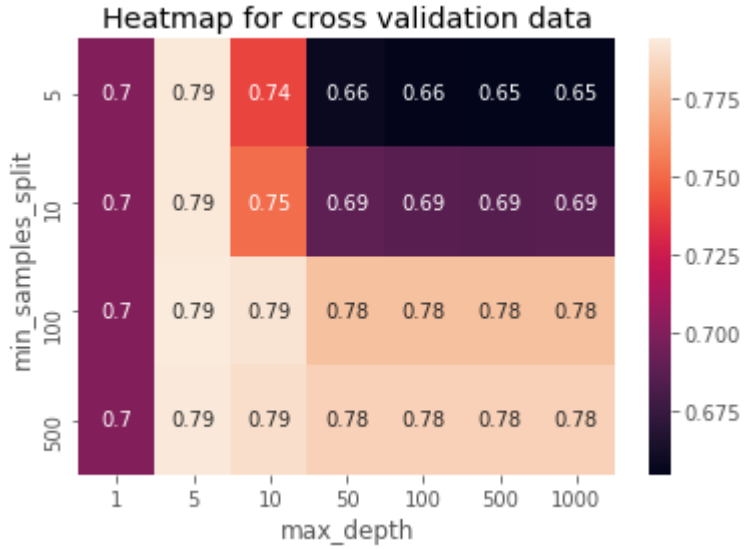
In [ ]:
```
new = ['RBF SVM','SVC',"alpha = 10000 ",0.7751,0.6916]
results2.loc[5] = new
```

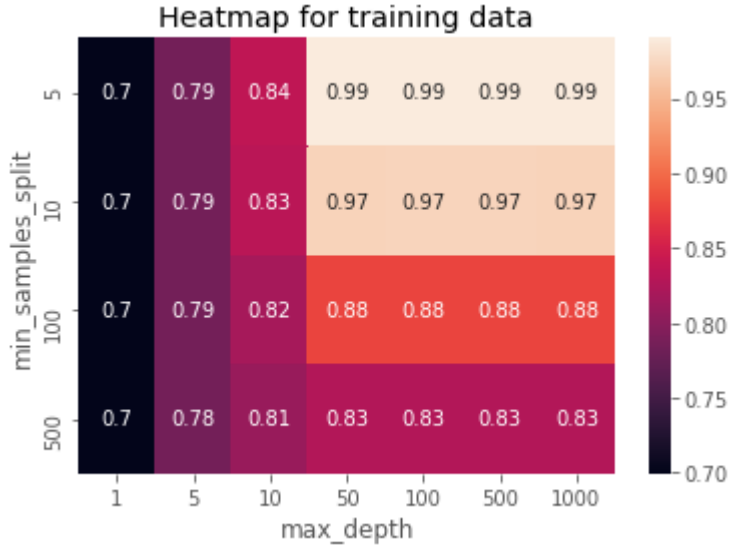
Applying Decision Tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
min_samples =  [5, 10, 100, 500]


param_grid={'min_samples_split':min_samples , 'max_depth':dept}
clf = DecisionTreeClassifier()
model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(rc_x_train,y_train)
print("optimal min_samples_split",model.best_estimator_.min_samples_split)
print("optimal max_depth",model.best_estimator_.max_depth)
```

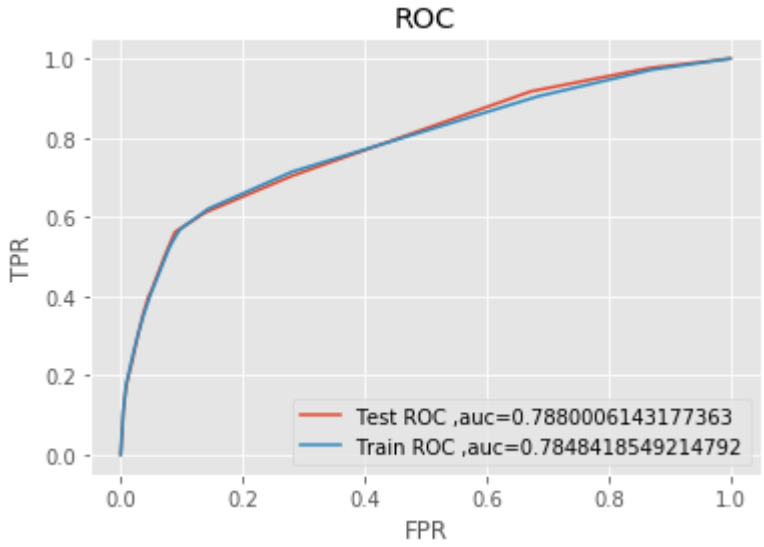
```
optimal min_samples_split 500
optimal max_depth 5
```

In [ ]:
```
import seaborn as sns
X = []
Y = []
cv_auc = []
train_auc = []
for n in min_samples:
    for d in dept:
        clf = DecisionTreeClassifier(max_depth = d,min_samples_split = n)
        clf.fit(rc_x_train,y_train)
        pred_cv = clf.predict_proba(rc_x_cv)[:,1]
        pred_train = clf.predict_proba(rc_x_train)[:,1]
        X.append(n)
        Y.append(d)
        cv_auc.append(roc_auc_score(y_cv,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))

#Heatmap for cross validation data
data = pd.DataFrame({'min_samples_split': X, 'max_depth': Y, 'AUC': cv_auc})
data_pivoted = data.pivot("min_samples_split", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()

#Heatmap for training data
data = pd.DataFrame({'min_samples_split': X, 'max_depth': Y, 'AUC': train_auc
})
data_pivoted = data.pivot("min_samples_split", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for training data')
plt.show()
```

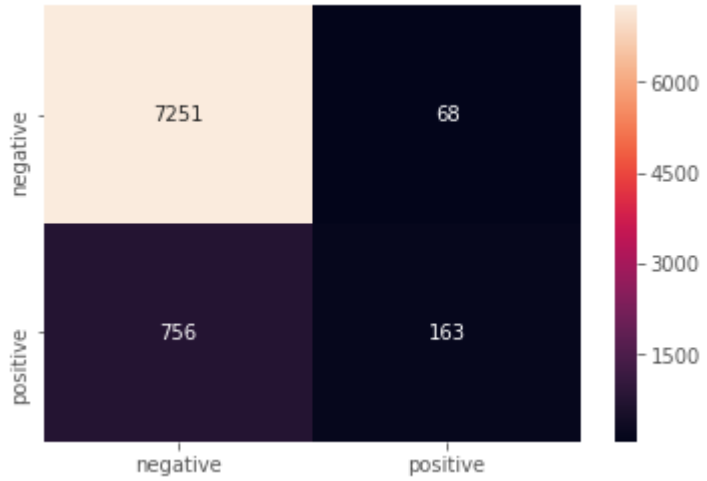

Heatmap for cross validation data
min_samples_split
5
10
100
500
0.7 0.79 0.74 0.66 0.66 0.65 0.65
0.7 0.79 0.75 0.69 0.69 0.69 0.69
0.7 0.79 0.79 0.78 0.78 0.78 0.78
0.7 0.79 0.79 0.78 0.78 0.78 0.78
1 5 10 50 100 500 1000
max_depth
0.775
0.750
0.725
0.700
0.675


Heatmap for training data
min_samples_split
5
10
100
500
0.7 0.79 0.84 0.99 0.99 0.99 0.99
0.7 0.79 0.83 0.97 0.97 0.97 0.97
0.7 0.79 0.82 0.88 0.88 0.88 0.88
0.7 0.78 0.81 0.83 0.83 0.83 0.83
1 5 10 50 100 500 1000
max_depth
0.95
0.90
0.85
0.80
0.75
0.70

In [ ]:
```
#training our model for max_depth=50,min_samples_split=500
clf = DecisionTreeClassifier(max_depth = 5,min_samples_split = 500)
clf.fit(rc_x_train,y_train)
pred_test =clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2=metrics.roc_curve(y_train,pred_train)

#ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.7880006143177363
AUC on Train data is 0.7848418549214792
---------------------------
```



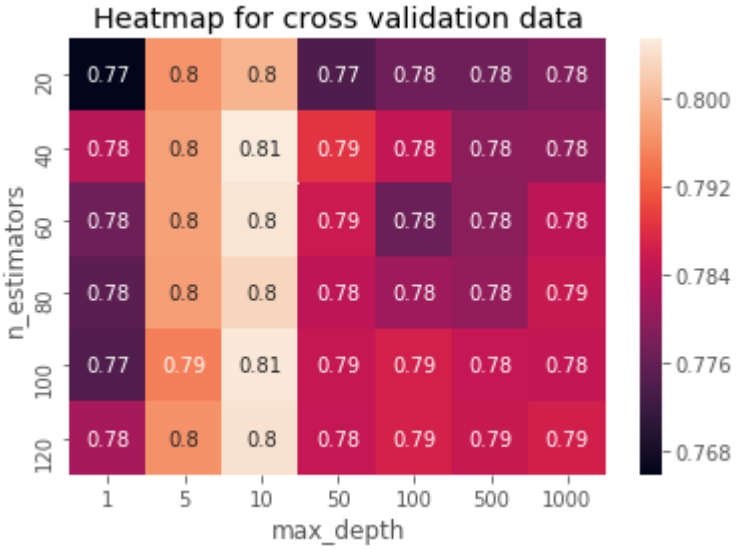
In [ ]:
```
new = ['Decision Tree','DecisionTreeClassifier',"max_depth = 5 & min_samples_s
plit = 500",0.7848,0.7880]
results2.loc[6] = new
```

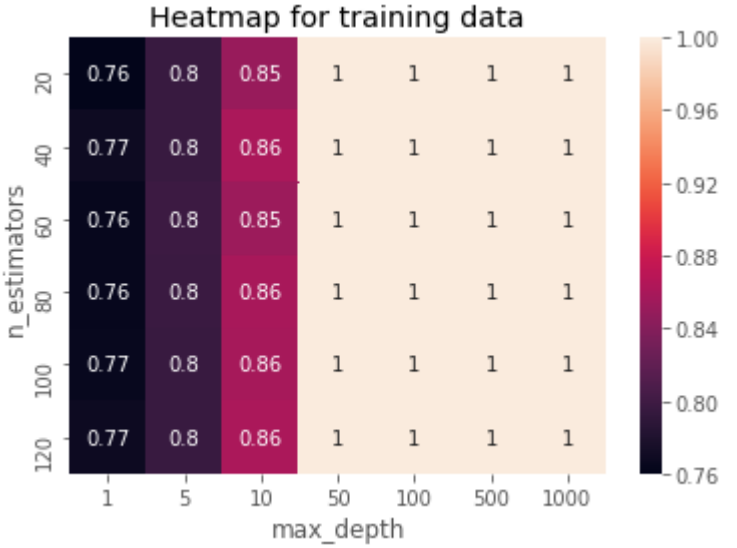
Applying Random Forest

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
n_estimators =  [20, 40, 60, 80, 100, 120]


param_grid={'n_estimators':n_estimators , 'max_depth':dept}
clf = RandomForestClassifier()
model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(rc_x_train,y_train)
print("optimal n_estimators",model.best_estimator_.n_estimators)
print("optimal max_depth",model.best_estimator_.max_depth)
```

```
optimal n_estimators 120
optimal max_depth 10
```

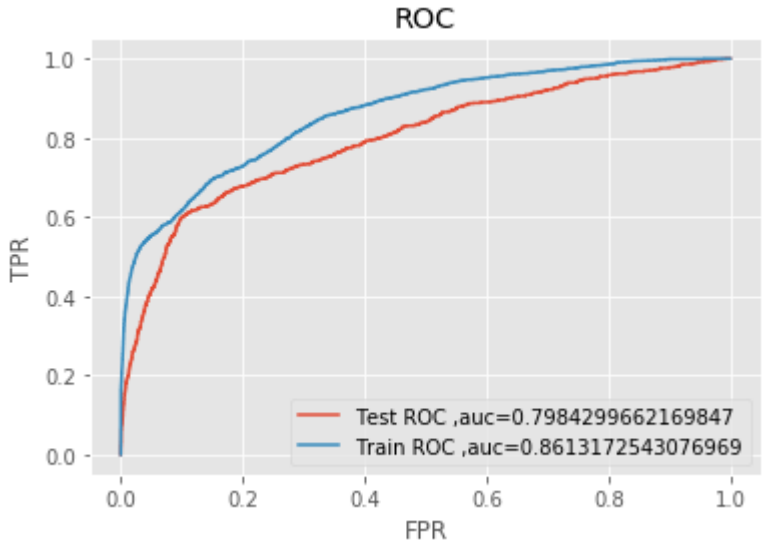
In [ ]:
```
import seaborn as sns
X = []
Y = []
cv_auc = []
train_auc = []
for n in n_estimators:
    for d in dept:
        clf = RandomForestClassifier(max_depth = d,n_estimators = n)
        clf.fit(rc_x_train,y_train)
        pred_cv = clf.predict_proba(rc_x_cv)[:,1]
        pred_train = clf.predict_proba(rc_x_train)[:,1]
        X.append(n)
        Y.append(d)
        cv_auc.append(roc_auc_score(y_cv,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))

#Heatmap for cross validation data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': cv_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()

#Heatmap for training data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': train_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for training data')
plt.show()
```



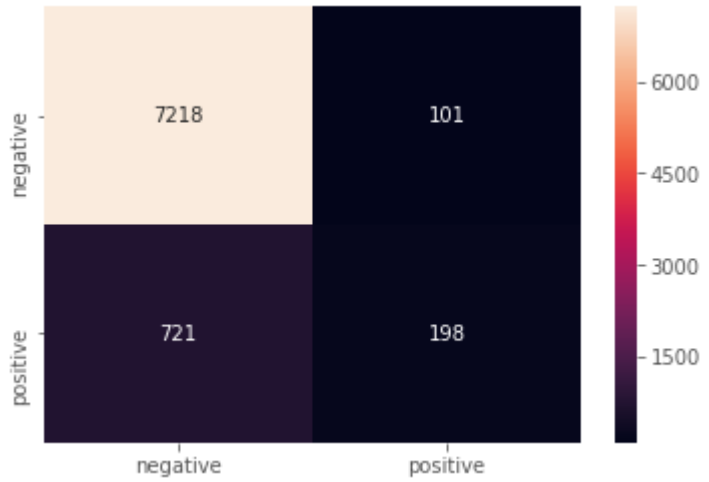


In [ ]:
```
optimal_n_estimators = model.best_estimator_.n_estimators
optimal_max_depth = model.best_estimator_.max_depth
```

In [ ]:
```
#training our model for max_depth=1000,n_estimators = 120
clf = RandomForestClassifier(max_depth = 10,n_estimators = 120)
clf.fit(rc_x_train,y_train)
pred_test =clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2=metrics.roc_curve(y_train,pred_train)

#ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.7984299662169847
AUC on Train data is 0.8613172543076969
---------------------------
```



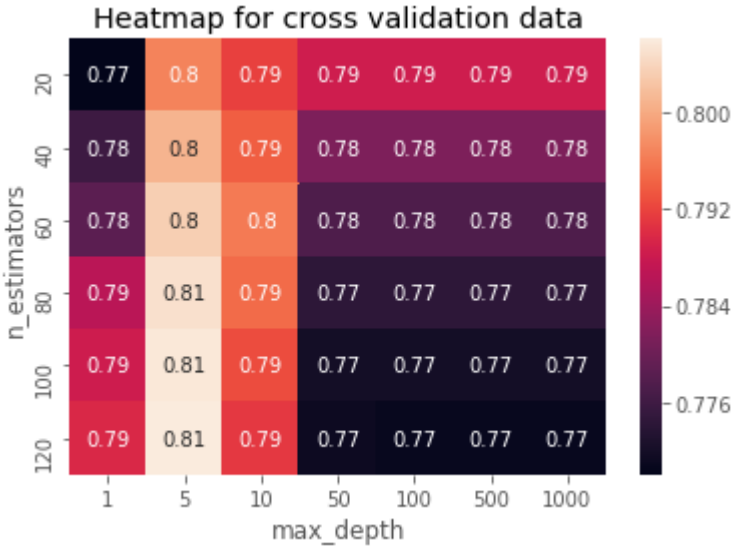
In [ ]:
```
new = ['Random Forest','RandomForestClassifier',"max_depth = 10 & min_samples_
split = 120",0.8613,0.7984]
results2.loc[7] = new
```

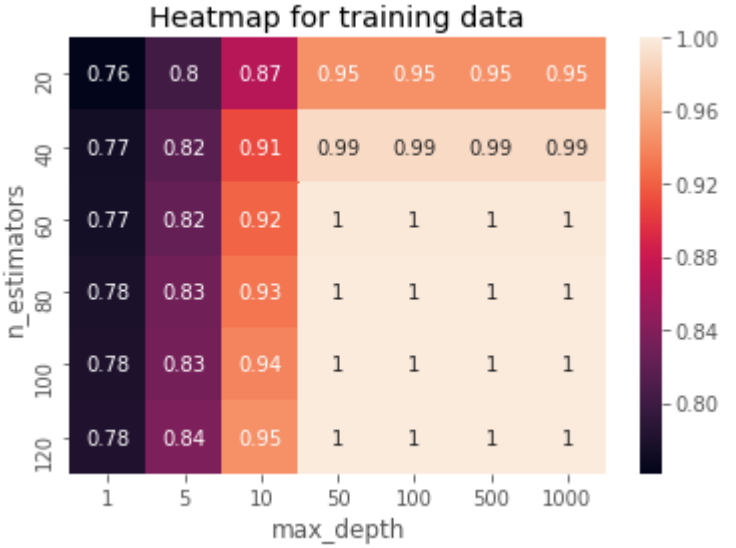
Applying XGBOOST

In [ ]:
```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

dept = [1, 5, 10, 50, 100, 500, 1000]
n_estimators =  [20, 40, 60, 80, 100, 120]


param_grid={'n_estimators':n_estimators , 'max_depth':dept}
clf = XGBClassifier()
model = GridSearchCV(clf,param_grid,scoring='roc_auc',n_jobs=-1,cv=3)
model.fit(rc_x_train,y_train)
print("optimal n_estimators",model.best_estimator_.n_estimators)
print("optimal max_depth",model.best_estimator_.max_depth)

optimal_n_estimators = model.best_estimator_.n_estimators
optimal_max_depth = model.best_estimator_.max_depth
```

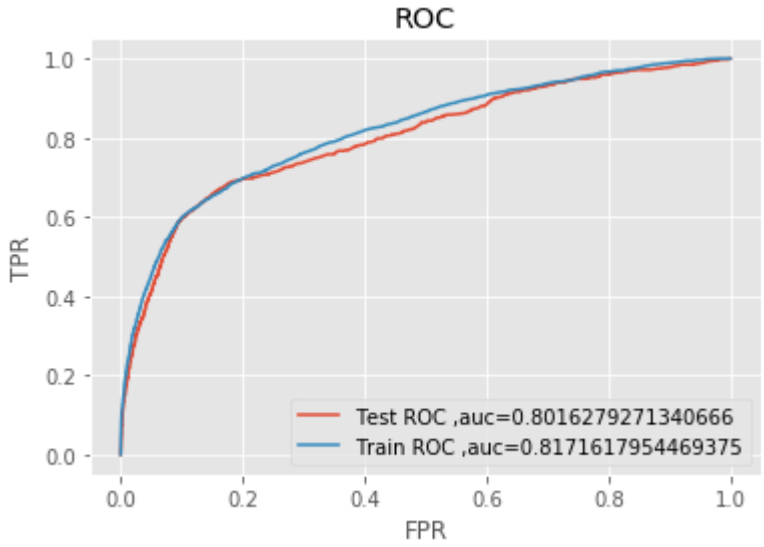
```
optimal n_estimators 40
optimal max_depth 5
```

In [ ]:
```
import seaborn as sns
X = []
Y = []
cv_auc = []
train_auc = []
for n in n_estimators:
    for d in dept:
        clf = XGBClassifier(max_depth = d,n_estimators = n)
        clf.fit(rc_x_train,y_train)
        pred_cv = clf.predict_proba(rc_x_cv)[:,1]
        pred_train = clf.predict_proba(rc_x_train)[:,1]
        X.append(n)
        Y.append(d)
        cv_auc.append(roc_auc_score(y_cv,pred_cv))
        train_auc.append(roc_auc_score(y_train,pred_train))
optimal_depth=Y[cv_auc.index(max(cv_auc))]
optimal_n_estimator=X[cv_auc.index(max(cv_auc))]

#Heatmap for cross validation data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': cv_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for cross validation data')
plt.show()

#Heatmap for training data
data = pd.DataFrame({'n_estimators': X, 'max_depth': Y, 'AUC': train_auc})
data_pivoted = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.title('Heatmap for training data')
plt.show()
```

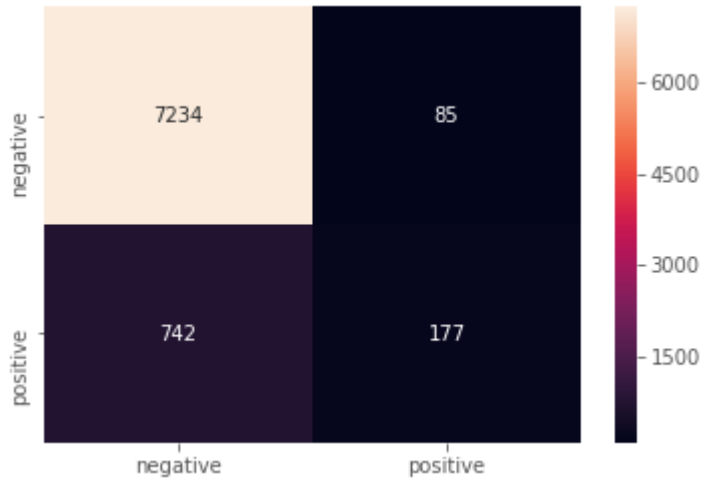

Heatmap for cross validation data
n_estimators
20
40
60
80
100
120
0.77 0.8 0.79 0.79 0.79 0.79 0.79
0.78 0.8 0.79 0.78 0.78 0.78 0.78
0.78 0.8 0.8 0.78 0.78 0.78 0.78
0.79 0.81 0.79 0.77 0.77 0.77 0.77
0.79 0.81 0.79 0.77 0.77 0.77 0.77
0.79 0.81 0.79 0.77 0.77 0.77 0.77
1 5 10 50 100 500 1000
max_depth
0.800
0.792
0.784
0.776


Heatmap for training data
n_estimators
20
40
60
80
100
120
0.76 0.8 0.87 0.95 0.95 0.95 0.95
0.77 0.82 0.91 0.99 0.99 0.99 0.99
0.77 0.82 0.92 1 1 1 1
0.78 0.83 0.93 1 1 1 1
0.78 0.83 0.94 1 1 1 1
0.78 0.84 0.95 1 1 1 1
1 5 10 50 100 500 1000
max_depth
1.00
0.96
0.92
0.88
0.84
0.80

In [ ]:
```
#training our model for max_depth=50,min_samples_split=500
clf = XGBClassifier(max_depth = 5,n_estimators = 40)
clf.fit(rc_x_train,y_train)
pred_test =clf.predict_proba(rc_x_test)[:,1]
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, pred_test)
pred_train = clf.predict_proba(rc_x_train)[:,1]
fpr2,tpr2,thresholds2=metrics.roc_curve(y_train,pred_train)

#ROC curve
x = plt.subplot( )
x.plot(fpr1, tpr1, label='Test ROC ,auc='+str(roc_auc_score(y_test,pred_test
)))
x.plot(fpr2, tpr2, label='Train ROC ,auc='+str(roc_auc_score(y_train,pred_trai
n)))
plt.title('ROC')
plt.xlabel('FPR')
plt.ylabel('TPR')
x.legend()
plt.show()

print("AUC on Test data is " +str(roc_auc_score(y_test,pred_test)))
print("AUC on Train data is " +str(roc_auc_score(y_train,pred_train)))

print("----------------------------")

# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_test.round()), index=c
lass_names, columns=class_names )
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```




```
AUC on Test data is 0.8016279271340666
AUC on Train data is 0.8171617954469375
---------------------------
```



In [ ]:
```
new = ['XGBOOST','XGBClassifier',"max_depth = 5 & n_estimators = 40",0.8171,0.
8016]
results2.loc[8] = new
```

LabelEncoder Performance Table

In [ ]:
```
results
```

Out[ ]:

	model	Classifier	hyper perameter	Train-AUC	Test-AUC
0	KNN with Brute force	KNeighborsClassifier	k = 37	0.8288	0.7587
1	KNN with kd-tree	KNeighborsClassifier	k = 33	0.8322	0.7573
2	Logistic Regression with L1	LogisticRegression	c = 10000	0.7642	0.7593
4	Linear SVM	SGDClassifier	alpha = 0.01	0.7537	0.7434
5	RBF SVM	SVC	c = 0.01	0.8771	0.7272
6	Decision Tree	DecisionTreeClassifier	max_depth = 10 & min_samples_split = 500	0.8131	0.7925
7	Random Forest	RandomForestClassifier	max_depth = 10 & min_samples_split = 100	0.8650	0.8035
8	XGBOOST	XGBClassifier	max_depth = 5 & n_estimators = 60	0.8267	0.8091

OneHotEncoder Performance Table

In [ ]:
```
results1
```

Out[ ]:

	model	Classifier	hyper perameter	Train-AUC	Test-AUC
0	KNN with Brute force	KNeighborsClassifier	k = 49	0.8209	0.7657
1	KNN with kd-tree	KNeighborsClassifier	k = 49	0.8209	0.7657
2	Logistic Regression with L1	LogisticRegression	c = 1	0.7675	0.7593
3	Logistic Regression with L2	LogisticRegression	c = 1	0.7676	0.7593
4	Linear SVM	SGDClassifier	alpha = 0.0001	0.7113	0.7164
5	RBF SVM	SVC	alpha = 10	0.7883	0.7008
6	Decision Tree	DecisionTreeClassifier	max_depth = 10 & min_samples_split = 500	0.8112	0.7964
7	Random Forest	RandomForestClassifier	max_depth = 10 & min_samples_split = 60	0.8556	0.7957
8	XGBOOST	XGBClassifier	max_depth = 5 & n_estimators = 40	0.8166	0.8035

Responce-Coding Performance Table

In [ ]:
```
results2
```

Out[ ]:

	model	Classifier	hyper perameter	Train-AUC	Test-AUC
0	KNN with Brute force	KNeighborsClassifier	k = 49	0.8247	0.7707
1	KNN with kd-tree	KNeighborsClassifier	k = 49	0.8246	0.7708
2	Logistic Regression with L1	LogisticRegression	c = 1000	0.7665	0.7591
3	Logistic Regression with L2	LogisticRegression	c = 10000	0.7665	0.7584
4	Linear SVM	SGDClassifier	alpha = 1	0.7565	0.7449
5	RBF SVM	SVC	alpha = 10000	0.7751	0.6916
6	Decision Tree	DecisionTreeClassifier	max_depth = 5 & min_samples_split = 500	0.7848	0.7880
7	Random Forest	RandomForestClassifier	max_depth = 10 & min_samples_split = 120	0.8613	0.7984
8	XGBOOST	XGBClassifier	max_depth = 5 & n_estimators = 40	0.8171	0.8016

Conclusion:

1. We load the data using pandas and do some Exploratory Data Analysis techniques
2. After applying the EDA we know some features are not importent then remove that features
3. We are using AUC as metric because this is higly imbalanced dataset
4. We split the whole data as train,test and cv then apply LabelEncoder to encode the all catagorical features
5. Apply all Machine Learning algorithms like KNN, Logistic Regression,Linear SVM, RBF SVM, Decision Tree, Random Forest and XGBoost with hyperparameter tuning
6. Decision Tree, Random Forest and XGBoost with some hyperparameter tuning gives best AUC values = 0.7925 , 0.8035 and 0.8091
7. Now we use OneHotEncoder to encode the all catagorical features
8. Apply all these Machine Learning algorithms with some hyperparameter tunings
9. When we use OneHotEncoder Decision Tree, Random Forest and XGBoost gives best Auc = 0.7964, 0.7957 & 0.8035
10. Fianlly we use respoce-coding to encode the categorical data and apply some machine learnig model with hyperparameter tuning
11. Random Forest and XGBoost gives best result AUC = 0.7984 & 0.8016

Ensembling:-

In [ ]:
```
!pip install scikit-learn==0.22.2.post1
```

```
Collecting scikit-learn==0.22.2.post1
  Downloading https://files.pythonhosted.org/packages/5e/d8/312e03adf4c78663e
17d802fe2440072376fee46cada1404f1727ed77a32/scikit_learn-0.22.2.post1-cp36-cp
36m-manylinux1_x86_64.whl (7.1MB)
     |████████████████████████████████| 7.1MB 9.7MB/s
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist
-packages (from scikit-learn==0.22.2.post1) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-
packages (from scikit-learn==0.22.2.post1) (1.0.0)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.6/dist
-packages (from scikit-learn==0.22.2.post1) (1.19.5)
Installing collected packages: scikit-learn
  Found existing installation: scikit-learn 0.24.1
    Uninstalling scikit-learn-0.24.1:
      Successfully uninstalled scikit-learn-0.24.1
Successfully installed scikit-learn-0.22.2.post1
```

In [ ]:
```
from sklearn.calibration import CalibratedClassifierCV
```

In [ ]:
```
import sklearn
```

In [ ]:
```
sklearn.__version__
```

Out[ ]:
```
'0.22.2.post1'
```

In [ ]:
```
cat_feat = ['job','marital','education','default','housing','loan','contact',
'month','day_of_week','poutcome']
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in cat_feat:
  data[i] = le.fit_transform(data[i])
```

In [ ]:
```
y = y #Splitting the Xi and Yi for ensembling
x = data
```

In [ ]:
```
x.shape
```

Out[ ]:
```
(41188, 21)
```

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

Out[ ]:
```
(41188,)
```

In [ ]:
```
from sklearn.model_selection import  train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split(x,y,test_size = 0.2, r
andom_state = 101)  #Doing the 80-20 split
```

In [ ]:
```
X1_train.shape
```

Out[ ]:
```
(32950, 21)
```

In [ ]:
```
X1_train.head()
```

Out[ ]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	durat
39577	79	5	1	2	0	0	0	0	6	3	
10104	54	9	1	6	0	2	0	1	4	2	
17235	36	7	1	3	0	0	0	0	3	0	
20926	32	0	2	6	0	2	0	0	1	2	
17626	52	2	1	2	0	2	0	0	3	1	

Balancing the dataset using SMOTE

In [ ]:
```
columns = X1_train.columns
```

In [ ]:
```
#Import the library for handling the imbalance dataset
from imblearn.over_sampling import SMOTE
Ov_sampling=SMOTE(random_state=100)
# now use SMOTE to oversample our train data which have features data_train_X
 and labels in data_train_y
ov_data_X,ov_data_y=Ov_sampling.fit_sample(X1_train,y1_train)
ov_data_X=pd.DataFrame(data=ov_data_X,columns=columns)
ov_data_y=pd.DataFrame(ov_data_y,columns=['Target'])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWar
ning: The module is deprecated in version 0.21 and will be removed in version
0.23 since we've dropped support for Python 2.7. Please rely on the official
version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: Futu
reWarning: The sklearn.neighbors.base module is  deprecated in version 0.22 a
nd will be removed in version 0.24. The corresponding classes / functions sho
uld instead be imported from sklearn.neighbors. Anything that cannot be impor
ted from sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: Futur
eWarning: Function safe_indexing is deprecated; safe_indexing is deprecated i
n version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

In [ ]:
```
print('length of oversampled data is   ',len(ov_data_X))
print('Number of no subscription in oversampled data ' ,len(ov_data_y[ov_data_
y['Target']==0]))
print('Number of subscription ' ,len(ov_data_y[ov_data_y['Target']==1]))
print('Proportion of no subscription data in oversampled data is ' ,len(ov_dat
a_y[ov_data_y['Target']==0])/len(ov_data_X))
print('Proportion of subscription data in oversampled data is ' ,len(ov_data_y
[ov_data_y['Target']==1])/len(ov_data_X))
```

```
length of oversampled data is    58538
Number of no subscription in oversampled data  29269
Number of subscription  29269
Proportion of no subscription data in oversampled data is  0.5
Proportion of subscription data in oversampled data is  0.5
```

In [ ]:
```
ov_data_y['Target'].value_counts()
```

Out[ ]:
```
1    29269
0    29269
Name: Target, dtype: int64
```

In [ ]:
```
ov_data_X.shape
```

Out[ ]:
```
(58538, 21)
```

In [ ]:
```
ov_data_y.shape
```

Out[ ]:
```
(58538, 1)
```

In [ ]:
```
from sklearn.model_selection import  train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split(ov_data_X,ov_data_y,st
ratify =ov_data_y ,test_size = 0.2, random_state = 101)  #Doing the 80-20 spli
t
```

In [ ]:
```
D1_X_train,D2_X_test,D1_Y_train,D2_Y_test=train_test_split(X1_train,y1_train,t
est_size = 0.5) #doing 50-50 split
D1_data=pd.concat([D1_X_train,D1_Y_train],axis=1)
D2_data=pd.concat([D2_X_test,D2_Y_test],axis=1)
```

In [ ]:
```
D1_data.shape
```

Out[ ]:
```
(23415, 22)
```

In [ ]:
```
D2_data.shape
```

Out[ ]:
```
(23415, 22)
```

In [ ]:
```
def sampling(n_estimators,D1_data):#sampling Randomly
    size=len(D1_data)
    samples=[]
    for i in range(n_estimators):
        S=D1_data.sample(n=size, replace=True, random_state=60000)
        samples.append(S)
    return samples
```

In [ ]:
```
n_estimators = 6
```

In [ ]:
```
D_samples=sampling(n_estimators,D1_data)#sampling from D1
```

In [ ]:
```
len(D_samples)
```

Out[ ]:
```
6
```

In [ ]:
```
def splitting(samples,n_estimators):#splitting Xi and Yi
    Xi=[]
    Yi=[]
    for i in range(n_estimators):
        a=samples[i]
        S_Yi=a['Target']
        S_Xi=a.drop(['Target'],axis=1)

        Xi.append(S_Xi)
        Yi.append(S_Yi)
    return Xi,Yi
```

In [ ]:
```
Xi,Yi=splitting(D_samples,n_estimators)#split xi and yi of the samples
```

In [ ]:
```
len(Xi)
```

Out[ ]:
```
6
```

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
```

In [ ]:
```
knn = KNeighborsClassifier()
log = LogisticRegression()
svm = SGDClassifier()
rf = RandomForestClassifier()
dt = DecisionTreeClassifier()
xgb = XGBClassifier()
```

In [ ]:
```
base_models = [knn, log,  svm, rf, dt, xgb]
```

In [ ]:
```
for i in range(n_estimators):
  print(base_models[i])
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
LogisticRegression(C=1.0, 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)
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=Tru
e,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       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')
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [ ]:
```
def modeling(n_estimators,Xi,Yi): #training on the base model
    #from sklearn.tree import DecisionTreeClassifier
    #Base_model=DecisionTreeClassifier()

    for i in range(n_estimators):
        BS=base_models[i].fit(Xi[i],Yi[i])

    return BS
```

In [ ]:
```
Base_models=modeling(n_estimators,Xi,Yi)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [ ]:
```
def base_pred(BS,D2_X_test,n_estimators):#pred DT with D2_X_test
    samples_prediction=[]

    for i in range(n_estimators):
        predictions=BS.predict(D2_X_test)
        samples_prediction.append(predictions)

    return samples_prediction
```

In [ ]:
```
samples_prediction=base_pred(Base_models,D2_X_test,n_estimators)
```

In [ ]:
```
def reshape(samples_prediction,n_estimators):#reshape
    SP_meta=[]
    for i in range(n_estimators):
        SP_meta=samples_prediction[i].reshape(-1,1)
    return SP_meta
```

In [ ]:
```
meta_samples=reshape(samples_prediction,n_estimators)
```

In [ ]:
```
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
def knn_hp(meta_samples,D2_Y_test):
  k = list(range(1,50,4))

  train_auc = []
  for i in k:
    clf = KNeighborsClassifier(n_neighbors = i,algorithm='kd_tree')
    clf.fit(meta_samples,D2_Y_test)
    prob_train = clf.predict(meta_samples)
    train_auc.append(accuracy_score(D2_Y_test,prob_train))
  optimal_k = k[train_auc.index(max(train_auc))]
  return optimal_k
```

In [ ]:
```
def log_hp(meta_samples,D2_Y_test):
  c = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001,0.00001]

  train_auc = []
  for i in c:
    clf = LogisticRegression(penalty='l2',C=i)
    clf.fit(meta_samples,D2_Y_test)
    prob_train = clf.predict(meta_samples)
    train_auc.append(accuracy_score(D2_Y_test,prob_train))
  optimal_c = c[train_auc.index(max(train_auc))]
  return optimal_c
```

In [ ]:
```
def svm_hp(meta_samples,D2_Y_test):
  alpha = [10000,1000,100,10,1,0.1,0.01,0.001,0.0001]

  train_auc = []
  for i in alpha:
    clf = SGDClassifier(alpha=i, loss = "hinge")
    clf.fit(meta_samples,D2_Y_test)
    prob_train = clf.predict(meta_samples)
    train_auc.append(accuracy_score(D2_Y_test,prob_train))
  optimal_alpha = alpha[train_auc.index(max(train_auc))]
  return optimal_alpha
```

In [ ]:
```
def dt_hp(meta_samples,D2_Y_test):
  min_samples =  [5, 10, 100, 500]

  train_auc = []
  for i in min_samples:
    clf = DecisionTreeClassifier(min_samples_split = i)
    clf.fit(meta_samples,D2_Y_test)
    prob_train = clf.predict(meta_samples)
    train_auc.append(accuracy_score(D2_Y_test,prob_train))
  optimal_min_samples = min_samples[train_auc.index(max(train_auc))]
  return optimal_min_samples
```

In [ ]:
```
def rf_hp(meta_samples,D2_Y_test):
  n_estimators =  [20, 40, 60, 80, 100, 120]

  train_auc = []
  for i in n_estimators:
    clf = RandomForestClassifier(n_estimators = i)
    clf.fit(meta_samples,D2_Y_test)
    prob_train = clf.predict(meta_samples)
    train_auc.append(accuracy_score(D2_Y_test,prob_train))
  optimal_n_estimators = n_estimators[train_auc.index(max(train_auc))]
  return optimal_n_estimators
```

In [ ]:
```
def xgb_hp(meta_samples,D2_Y_test):
  n_estimators =  [20, 40, 60, 80, 100, 120]

  train_auc = []
  for i in n_estimators:
    clf = XGBClassifier(n_estimators = i)
    clf.fit(meta_samples,D2_Y_test)
    prob_train = clf.predict(meta_samples)
    train_auc.append(accuracy_score(D2_Y_test,prob_train))
  optimal_n_estimators = n_estimators[train_auc.index(max(train_auc))]
  return optimal_n_estimators
```

In [ ]:
```
best_k = knn_hp(meta_samples,D2_Y_test)
best_c = log_hp(meta_samples,D2_Y_test)
best_alpha = svm_hp(meta_samples,D2_Y_test)
beat_min_samples = dt_hp(meta_samples,D2_Y_test)
best_rf = rf_hp(meta_samples,D2_Y_test)
best_xgb = xgb_hp(meta_samples,D2_Y_test)
```

In [ ]:
```
def xgb_model(meta_samples,D2_Y_test):
  from xgboost import XGBClassifier
  meta_xgb = XGBClassifier(n_estimators = best_xgb)
  meta_xgb.fit(meta_samples, D2_Y_test)
  return meta_xgb
```

In [ ]:
```
def rf_model(meta_samples,D2_Y_test):
  meta_rf = RandomForestClassifier(n_estimators = best_rf)
  meta_rf.fit(meta_samples, D2_Y_test)
  return meta_rf
```

In [ ]:
```
def dt_model(meta_samples,D2_Y_test):
  meta_dt = DecisionTreeClassifier(min_samples_split = beat_min_samples)
  meta_dt.fit(meta_samples, D2_Y_test)
  return meta_dt
```

In [ ]:
```
def svm_model(meta_samples,D2_Y_test):
  meta_svm = make_pipeline(StandardScaler(),SGDClassifier(max_iter=1000, tol=1
e-3, alpha=best_alpha))
  meta_svm.fit(meta_samples, D2_Y_test)
  return meta_svm
```

In [ ]:
```
def log_model(meta_samples,D2_Y_test):
  meta_log = LogisticRegression(penalty='l2',C = best_c)
  meta_log.fit(meta_samples, D2_Y_test)
  return meta_log
```

In [ ]:
```
def knn_model(meta_samples,D2_Y_test):
  meta_knn = KNeighborsClassifier(n_neighbors=best_k)
  meta_knn.fit(meta_samples, D2_Y_test)
  return meta_knn
```

In [ ]:
```
meta_model_knn = knn_model(meta_samples,D2_Y_test)#calling knn model
```

In [ ]:
```
meta_model_log = log_model(meta_samples,D2_Y_test)#calling log model
```

In [ ]:
```
meta_model_dt = dt_model(meta_samples,D2_Y_test)#calling dt model
```

In [ ]:
```
meta_model_rf = rf_model(meta_samples,D2_Y_test)#calling rf model
```

In [ ]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
meta_model_svm = svm_model(meta_samples,D2_Y_test)#calling svm model
```

In [ ]:
```
meta_model_xgb = xgb_model(meta_samples,D2_Y_test)#calling XGB model
```

In [ ]:
```
def test_pred(Base_models,twenty_test_pred,n_estimators):#test pred
    test_prediction=[]
    for i in range(n_estimators):
        predictions=Base_models.predict(X_test)
        test_prediction.append(predictions)
    return test_prediction
```

In [ ]:
```
twenty_test_pred=base_pred(Base_models,X1_test,n_estimators)
```

In [ ]:
```
tp_samples=reshape(twenty_test_pred,n_estimators)#reshape
```

In [ ]:
```
pred_final_knn = meta_model_knn.predict(tp_samples)
```

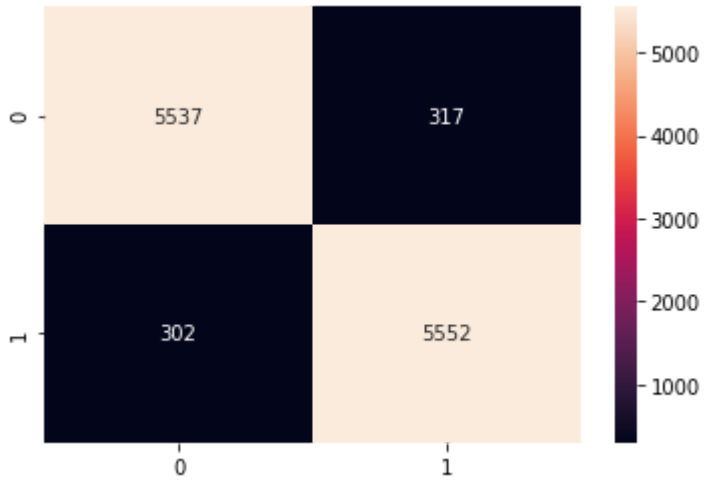
In [ ]:
```
pred_final_knn[:100]
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [ ]:
```
from sklearn.metrics import accuracy_score
acc_knn =accuracy_score(y1_test,pred_final_knn)
print('the accuracy for the custom ensemble implementation using knn ',acc_knn
)
```

```
the accuracy for the custom ensemble implementation using knn  0.947130167406
9013
```

In [ ]:
```
# Code for drawing seaborn heatmaps
#class_names = le.classes_
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
df_heatmap = pd.DataFrame(confusion_matrix(y1_test, pred_final_knn.round()))
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



In [ ]:
```
pred_final_log = meta_model_log.predict(tp_samples)
```

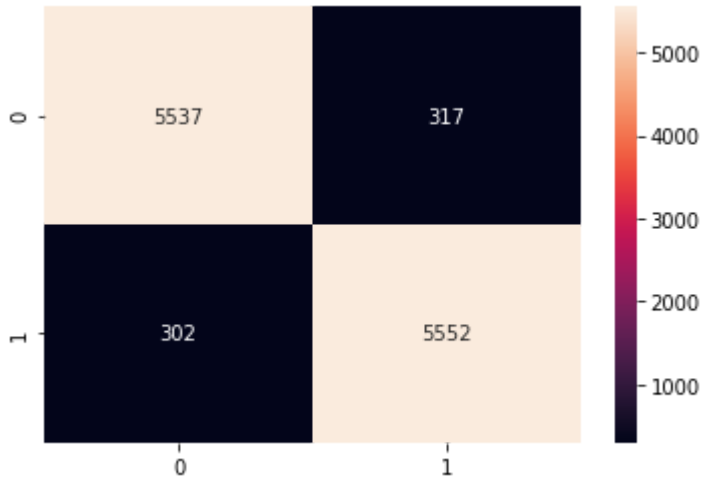
In [ ]:
```
pred_final_log[:100]
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [ ]:
```
acc_log = accuracy_score(y1_test,pred_final_log)
print('the accuracy for the custom ensemble implementation using logistic regr
ession',acc_log)
```

```
the accuracy for the custom ensemble implementation using logistic regression
0.9471301674069013
```

In [ ]:
```
# Code for drawing seaborn heatmaps
#class_names = le.classes_
from sklearn.metrics import confusion_matrix

df_heatmap = pd.DataFrame(confusion_matrix(y1_test, pred_final_log.round()))
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



In [ ]:
```
pred_final_svm = meta_model_svm.predict(tp_samples)
```

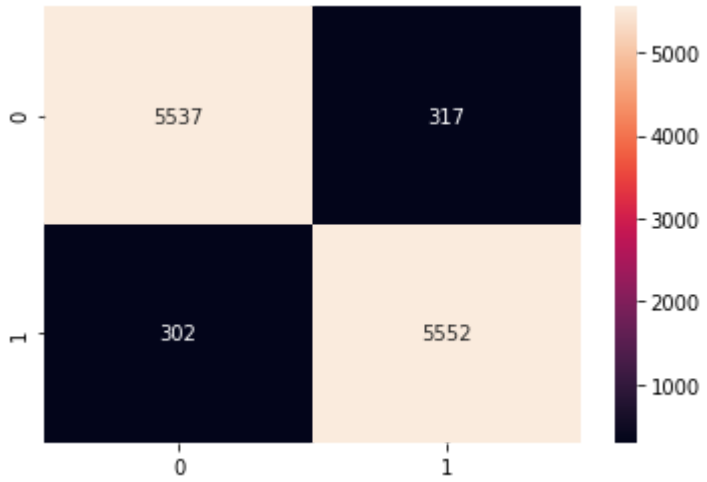
In [ ]:
```
pred_final_svm[:100]
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [ ]:
```
acc_svm = accuracy_score(y1_test,pred_final_svm)
print('the accuracy for the custom ensemble implementation using svm',acc_svm)
```

```
the accuracy for the custom ensemble implementation using svm 0.9471301674069
013
```

In [ ]:
```
# Code for drawing seaborn heatmaps
#class_names = le.classes_
from sklearn.metrics import confusion_matrix

df_heatmap = pd.DataFrame(confusion_matrix(y1_test, pred_final_svm.round()))
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



In [ ]:
```
pred_final_dt = meta_model_dt.predict(tp_samples)
```

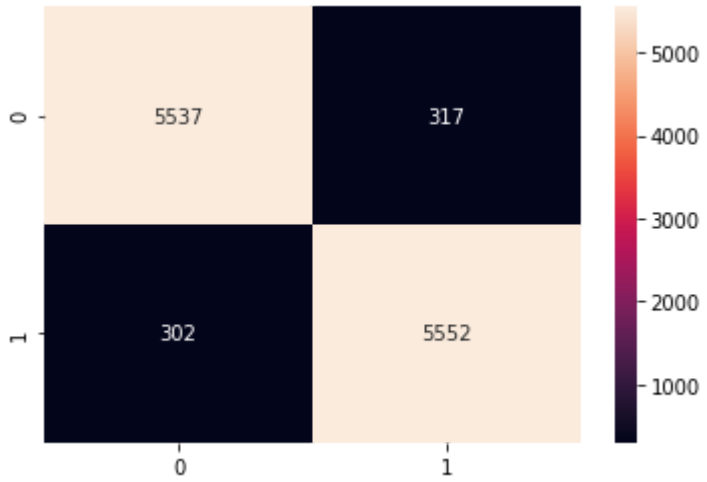
In [ ]:
```
pred_final_dt[:100]
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [ ]:
```
acc_dt = accuracy_score(y1_test,pred_final_dt)
print('the accuracy for the custom ensemble implementation using decision tre
e',acc_dt)
```

```
the accuracy for the custom ensemble implementation using decision tree 0.947
1301674069013
```

In [ ]:
```
# Code for drawing seaborn heatmaps
#class_names = le.classes_
from sklearn.metrics import confusion_matrix

df_heatmap = pd.DataFrame(confusion_matrix(y1_test, pred_final_dt.round()))
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



In [ ]:
```
pred_final_rf = meta_model_rf.predict(tp_samples)
```

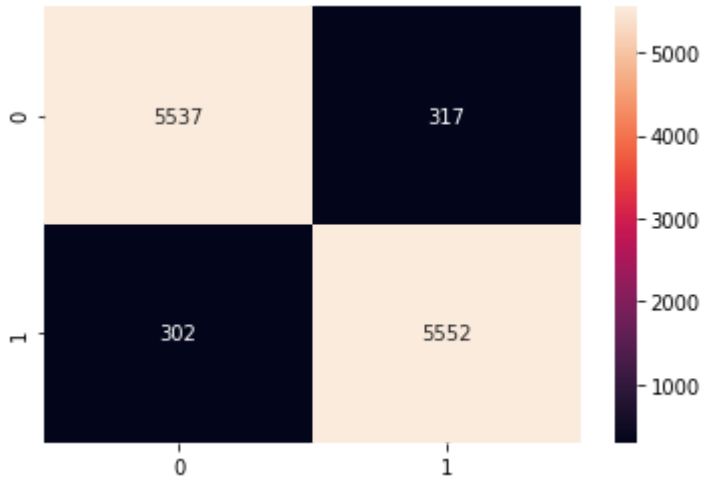
In [ ]:
```
pred_final_rf[:100]
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [ ]:
```
acc_rf = accuracy_score(y1_test,pred_final_rf)
print('the accuracy for the custom ensemble implementation using random fores
t',acc_rf)
```

```
the accuracy for the custom ensemble implementation using random forest 0.947
1301674069013
```

In [ ]:
```
# Code for drawing seaborn heatmaps
#class_names = le.classes_
from sklearn.metrics import confusion_matrix

df_heatmap = pd.DataFrame(confusion_matrix(y1_test, pred_final_rf.round()))
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



In [ ]:
```
pred_final_xgb = meta_model_xgb.predict(tp_samples)
```

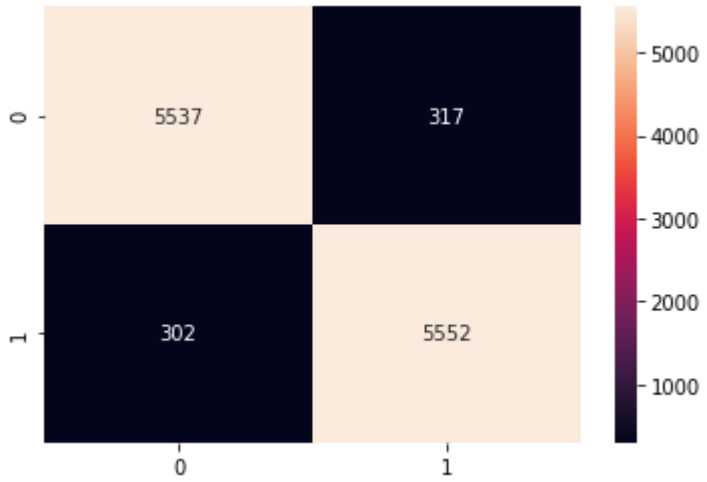
In [ ]:
```
pred_final_xgb[:100]
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [ ]:
```
acc_xgb = accuracy_score(y1_test,pred_final_xgb)
print('the accuracy for the custom ensemble implementation using XGBoost',acc_
xgb)
```

```
the accuracy for the custom ensemble implementation using XGBoost 0.947130167
4069013
```

In [ ]:
```
# Code for drawing seaborn heatmaps
#class_names = le.classes_
from sklearn.metrics import confusion_matrix

df_heatmap = pd.DataFrame(confusion_matrix(y1_test, pred_final_xgb.round()))
fig = plt.figure( )
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
```



Oberservations:-

1.The accuracy for the custom ensemble implementation using logistic regression is 0.9471.

2.The accuracy for the custom ensemble implementation using knn is 0.9471.

3.The accuracy for the custom ensemble implementation using svm is 0.9471.

4.The accuracy for the custom ensemble implementation using decision tree is 0.9471.

5.The accuracy for the custom ensemble implementation using random forest 0.9471.

6.The accuracy for the custom ensemble implementation using XGBoost 0.9471.

Final Pipeline:-

In [ ]:
```
data.info()
```

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

In [ ]:
```
data.columns
```

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

In [ ]:
```
cat_feat = ['job','marital','education','default', 'housing', 'loan','contact'
, 'month', 'day_of_week','poutcome','y']
```

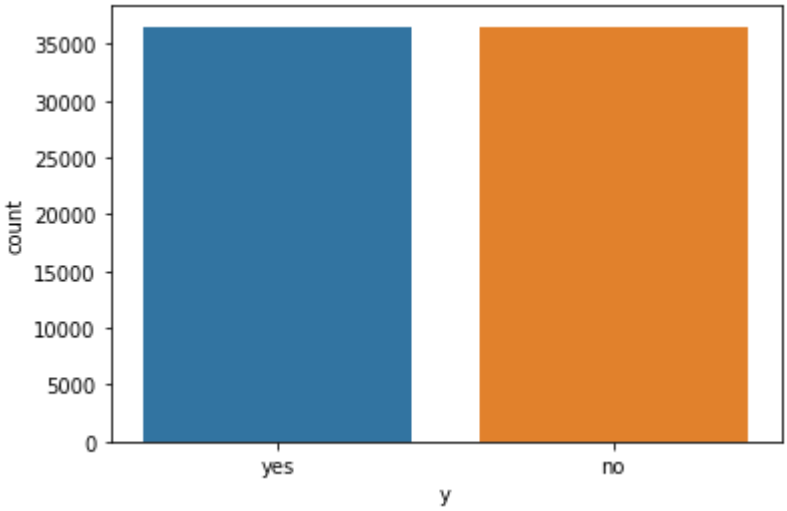
In [ ]:
```
"""
Processing the data
"""

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler


def process_data(data):
  data = data.fillna(0)
  print(data.shape)

  print("-----------------")

  #Check the data calss label
  import seaborn as sns
  print(data.y.value_counts())
  sns.countplot(x='y', data=data)

  print("-----------------")

  #Balance the dataset
  df_no = data[data['y']== 'no']
  df_yes = data[data['y'] == 'yes']
  if len(df_no) > len(df_yes):
    df_majority = df_no
    df_minority = df_yes
  else:
    df_minority = df_no
    df_majority = df_yes
  print('The len of minority class label:- ',len(df_minority))
  print('The len of majority class label:- ',len(df_majority))
  print("++++++++++++")


  from sklearn.utils import resample
  #df_1_smaple = resample(df_1, replace=True,  n_samples=10000,random_state=12
3)
  df_minority_smaple = resample(df_minority, replace=True,  n_samples=36548, r
andom_state=123)



  print("++++++++++++++++++")
  #df_up_sample = resample(df_minority, n_samples=len(df_minority), replace=Fa
lse)
  df_up_sample = pd.concat([df_minority_smaple,df_majority ])
  print('The shape of upsampled dataset:- ',df_up_sample.shape)

  print("-----------------")

  #Check the data calss label
  import seaborn as sns
  print(df_up_sample.y.value_counts())
  sns.countplot(x='y', data=df_up_sample)
```

```
  print("-----------------")


  from sklearn.utils import shuffle
  df = shuffle(df_up_sample)

  from sklearn.preprocessing import LabelEncoder
  le = LabelEncoder()
  for i in df[cat_feat]:
    df[i] = le.fit_transform(df[i])

  '''
  from sklearn.preprocessing import StandardScaler
  ss = StandardScaler()
  ss.fit(df)
  df = ss.transform(df)
  '''

  x =  df.loc[:, df.columns != 'y']
  y = df['y']

  from sklearn.model_selection import  train_test_split
  X1_train, X1_test, y1_train, y1_test = train_test_split(x,y,test_size = 0.2,
stratify = y ,random_state = 101)

  print("The shape of Train data:- ", X1_train.shape)
  print("The shape of test data:- ", X1_test.shape)

  print("-----------------")


  X1_train.info()
  print("-----------------")
  X1_test.info()
  print("-----------------")


  return X1_train, y1_train, X1_test, y1_test
```

In [ ]:
```
X_train, y_train, X_test, y_test = process_data(data)
```

```
(41188, 21)
-----------------
no     36548
yes     4640
Name: y, dtype: int64
-----------------
The len of minority class label:-  4640
The len of majority class label:-  36548
++++++++++++
++++++++++++++++++
The shape of upsampled dataset:-  (73096, 21)
-----------------
no     36548
yes    36548
Name: y, dtype: int64
-----------------
The shape of Train data:-  (58476, 20)
The shape of test data:-  (14620, 20)
-----------------
<class 'pandas.core.frame.DataFrame'>
Int64Index: 58476 entries, 14283 to 25154
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             58476 non-null  int64  
 1   job             58476 non-null  int64  
 2   marital         58476 non-null  int64  
 3   education       58476 non-null  int64  
 4   default         58476 non-null  int64  
 5   housing         58476 non-null  int64  
 6   loan            58476 non-null  int64  
 7   contact         58476 non-null  int64  
 8   month           58476 non-null  int64  
 9   day_of_week     58476 non-null  int64  
 10  duration        58476 non-null  int64  
 11  campaign        58476 non-null  int64  
 12  pdays           58476 non-null  int64  
 13  previous        58476 non-null  int64  
 14  poutcome        58476 non-null  int64  
 15  emp.var.rate    58476 non-null  float64
 16  cons.price.idx  58476 non-null  float64
 17  cons.conf.idx   58476 non-null  float64
 18  euribor3m       58476 non-null  float64
 19  nr.employed     58476 non-null  float64
dtypes: float64(5), int64(15)
memory usage: 9.4 MB
-----------------
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14620 entries, 37544 to 13188
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             14620 non-null  int64  
 1   job             14620 non-null  int64  
 2   marital         14620 non-null  int64  
 3   education       14620 non-null  int64  
 4   default         14620 non-null  int64  
```

```
 5   housing         14620 non-null  int64
 6   loan            14620 non-null  int64
 7   contact         14620 non-null  int64
 8   month           14620 non-null  int64
 9   day_of_week     14620 non-null  int64
 10  duration        14620 non-null  int64
 11  campaign        14620 non-null  int64
 12  pdays           14620 non-null  int64
 13  previous        14620 non-null  int64
 14  poutcome        14620 non-null  int64
 15  emp.var.rate    14620 non-null  float64
 16  cons.price.idx  14620 non-null  float64
 17  cons.conf.idx   14620 non-null  float64
 18  euribor3m       14620 non-null  float64
 19  nr.employed     14620 non-null  float64
dtypes: float64(5), int64(15)
memory usage: 2.3 MB
-----------------
```



In [ ]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(58476, 20)
(58476,)
```

In [ ]:
```
y_train = pd.DataFrame(list(y_train),columns=['Target'])
y_teat= pd.DataFrame(list(y_test),columns=['Target'])
```

In [ ]:
```
def ensembling_model(X_train, y_train, X_test, y_test):
  from sklearn.model_selection import  train_test_split
  D1_X_train,D2_X_test,D1_Y_train,D2_Y_test=train_test_split(X_train, y_train,
stratify = y_train ,test_size = 0.5) #doing 50-50 split
  D1_X_train.reset_index(drop=True, inplace=True)
  D1_Y_train.reset_index(drop=True, inplace=True)

  D2_X_test.reset_index(drop=True, inplace=True)
  D2_Y_test.reset_index(drop=True, inplace=True)

  D1_data=pd.concat([D1_X_train,D1_Y_train],axis=1)
  D2_data=pd.concat([D2_X_test,D2_Y_test],axis=1)

  def sampling(n_estimators,D1_data):#sampling Randomly
    size=len(D1_data)
    samples=[]
    for i in range(n_estimators):
      S=D1_data.sample(n=size, replace=True, random_state=60000)
      samples.append(S)
    return samples

  n_estimators = 6
  D_samples=sampling(n_estimators,D1_data)#sampling from D1


  #splitting Xi and Yi
  def splitting(samples,n_estimators):#splitting Xi and Yi
    Xi=[]
    Yi=[]
    for i in range(n_estimators):
      a=samples[i]
      S_Yi=a['Target']
      S_Xi=a.drop(['Target'],axis=1)

      Xi.append(S_Xi)
      Yi.append(S_Yi)
    return Xi,Yi

  Xi,Yi=splitting(D_samples,n_estimators)#split xi and yi of the samples

  from sklearn.pipeline import Pipeline
  from sklearn.neighbors import KNeighborsClassifier
  from sklearn.linear_model import LogisticRegression
  from sklearn.linear_model import SGDClassifier
  from sklearn.ensemble import RandomForestClassifier
  from sklearn.tree import DecisionTreeClassifier
  from xgboost import XGBClassifier

  knn = KNeighborsClassifier()
  log = LogisticRegression()
  svm = SGDClassifier()
  rf = RandomForestClassifier()
  dt = DecisionTreeClassifier()
  xgb = XGBClassifier()

  base_models = [knn, log,  svm, rf, dt, xgb]
```

```
#training on the base model
def modeling(n_estimators,Xi,Yi):
  #from sklearn.tree import DecisionTreeClassifier
  #Base_model=DecisionTreeClassifier()

  for i in range(n_estimators):
    BS=base_models[i].fit(Xi[i],Yi[i])

  return BS

Base_models=modeling(n_estimators,Xi,Yi)


  #pred DT with D2_X_test
def base_pred(BS,D2_X_test,n_estimators):
  samples_prediction=[]

  for i in range(n_estimators):
    predictions=BS.predict(D2_X_test)
    samples_prediction.append(predictions)

  return samples_prediction

samples_prediction=base_pred(Base_models,D2_X_test,n_estimators)

#reshape
def reshape(samples_prediction,n_estimators):
  SP_meta=[]
  for i in range(n_estimators):
    SP_meta=samples_prediction[i].reshape(-1,1)
  return SP_meta

meta_samples=reshape(samples_prediction,n_estimators)


#test pred
def test_pred(Base_models,twenty_test_pred,n_estimators):
  test_prediction=[]
  for i in range(n_estimators):
    predictions=Base_models.predict(X_test)
    test_prediction.append(predictions)
  return test_prediction

twenty_test_pred=base_pred(Base_models,X_test,n_estimators)
#reshape
tp_samples=reshape(twenty_test_pred,n_estimators)

from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")

# Create a pipeline
pipe = Pipeline([("classifier", RandomForestClassifier())])
# Create dictionary with candidate learning algorithms and their hyperparame
```

```
ters
  grid_param = [
                {"classifier": [LogisticRegression()],
                 "classifier__penalty": ['l2','l1'],
                 "classifier__C": np.logspace(0, 4, 10)
                 },

                {"classifier": [RandomForestClassifier()],
                 "classifier__n_estimators": [10, 100, 1000],
                 "classifier__max_depth":[5,8,15,25,30,None],
                 "classifier__min_samples_leaf":[1,2,5,10,15,100],
                 "classifier__max_leaf_nodes": [2, 5,10]},
               {"classifier": [KNeighborsClassifier()],
                  "classifier__n_neighbors": [1,5, 9, 13, 17, 21, 25, 29, 33, 3
7, 41, 45, 49]
                  },
               {"classifier": [DecisionTreeClassifier()],
                  "classifier__min_samples_split": [5, 10, 100, 500]
                  }
               ]
  # create a gridsearch of the pipeline, the fit the best model
  gridsearch = GridSearchCV(pipe, grid_param, cv=5, verbose=0,n_jobs=-1) # Fit
grid search
  best_model = gridsearch.fit(meta_samples,D2_Y_test)

  print("The train mean accuracy of the model is:",best_model.score(meta_sampl
es,D2_Y_test))
  print("The test mean accuracy of the model is:",best_model.score(tp_samples,
y_test))

  pred_final = best_model.predict(tp_samples)
  acc =accuracy_score(y_test,pred_final)
  print('the accuracy for the custom ensemble implementation  ',acc)


  # Code for drawing seaborn heatmaps
  #class_names = le.classes_
  from sklearn.metrics import confusion_matrix
  import matplotlib.pyplot as plt
  import seaborn as sns
  df_heatmap = pd.DataFrame(confusion_matrix(y_test, pred_final.round()))
  fig = plt.figure( )
  heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")
  return acc
```

In [ ]:
```
Final_output = ensembling_model(X_train, y_train, X_test, y_test)
```

```
The train mean accuracy of the model is: 0.8899377522402353
The test mean accuracy of the model is: 0.8902872777017784
the accuracy for the custom ensemble implementation   0.8902872777017784
```