POST ASSIGNMENT

MACHINE LEARNING

DUSANE ASHISH CHANDRAKANT
M. TECH (ACDS)

{ PRN : 170847980003 }

Date: 24-03-2018

```
'''
*********************************************************************************
1. Load bank.csv data into python script. Split data into validation and training
dataset. Design decision tree algorithm to predict if individual can get load or
not.
*********************************************************************************
'''

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics

#loading dataset
datapath="/home/student/bank.csv"
data=pd.read_csv(datapath,sep=";")
#print(data)

#preprocessing
a={"unknown":0,"other":1,"failure":2,"success":3}
pout=[]
for i in data["poutcome"]:
    pout.append(a[i])
data["poutcome_new"]=pd.Series(pout)
#print(data)

#preprocessing marital
b={ "married":0,"divorced":1,"single":2}
mar=[]
for i in data["marital"]:
    mar.append(b[i])
data["marital_new"]=pd.Series(mar)

#preprocessing education
e={"unknown":0,"secondary":1,"primary":2,"tertiary":3}
edu=[]
for i in data["education"]:
    edu.append(e[i])
data["education_new"]=pd.Series(edu)

#preprocessing job
j={"admin.":0,"unknown":1,"unemployed":2,"management":3,"housemaid":4,"entrepre
neur":5,"student":6,"blue-collar":7,"self-
employed":8,"retired":9,"technician":10,"services":11}
jo=[]
for i in data["job"]:
    jo.append(j[i])
data["job_new"]=pd.Series(jo)

#preprocessing default,loan and y
```

```
dic={"yes":1,"no":0}
d=[]
l=[]
y1=[]
ho=[]
for i,j,k,h in zip(data["default"],data["loan"],data["y"],data["housing"]):
    d.append(dic[i])
    l.append(dic[j])
    y1.append(dic[k])
    ho.append(dic[h])
data["default_new"]=pd.Series(d)
data["loan_new"]=pd.Series(l)
data["y_new"]=pd.Series(y1)
data["housing_new"]=pd.Series(ho)

#preprocessing month
month={"jan":1,"feb":2,"mar":3,"apr":4,"may":5,"jun":6,"jul":7,"aug":8,"sep":9,
"oct":10,"nov":11,"dec":12}
mo=[]
for m in data["month"]:
    mo.append(month[m])
data["month_new"]=pd.Series(mo)

#splitting dataset into training and validation
X=data.drop(["poutcome","marital","education","job","default","loan","y","housi
ng","month","y_new","contact","month_new","day","pdays"],axis=1)
y=data["y_new"]
#print(y)

#splitting data
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1
23,stratify=y)
print("x_train=",len(x_train))
print("x_test=",len(x_test))
print("y_train=",len(y_train))
print("y_test=",len(y_test))

#fitting to model
model = tree.DecisionTreeClassifier(criterion='gini')
model.fit(x_train,y_train)
pred=model.predict(x_test)

label=["no","yes"]
#printing output
for i,j in zip(pd.Series(pred),y_test):
    print("Predicted=",label[i],'\t',"Actual=",label[j])

print('Accuracy Score = ', metrics.accuracy_score(pred,y_test))
```

'''

```
**********************************************************************
OUTPUT :~
**********************************************************************

('x_train=', 3616)
('x_test=', 905)
('y_train=', 3616)
('y_test=', 905)

('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
```

```
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
```

```
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')

('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
```

```
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')

('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
```

```
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')

('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
```

```
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')

('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
```

```
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
```

```
('Predicted=', 'no', '\t', 'Actual=', 'yes')
('Predicted=', 'yes', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'no', '\t', 'Actual=', 'no')
('Predicted=', 'yes', '\t', 'Actual=', 'yes')

('Accuracy Score =' 0.8419698516574585)

'''


'''
```

****

2. Download titanic dataset from Kaggle. Design a model to find age of passenger for which it is missing. For this first separate rows with age columns present from entire data. Treat it as training data. Split training data into train and validation dataset. Use appropriate algorithm to create model. Now separate rows with missing age column and treat it as test dataset. Predict age values on test dataset

****

```
''''

import pandas as pd
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

data=pd.read_csv("/home/student/train.csv")

df= data[data['Age'].isnull()]
#print('df=',df)

#data Preprocessing
dic={"male":0,"female":1}
g=[]
for i in data["Sex"]:
    g.append(dic[i])
data["gender"]=pd.Series(g)

data.dropna(subset=['Age'], inplace = True)


#splitting data
x_train=data.drop(["Sex","Name","Cabin","Embarked","Ticket","gender","Age"],axi
s=1)
```

```
print(x_train.isnull().sum())
#print(X)

y_train=data["Age"]
#print("y_train=",y_train)

x_test=df.drop(["Sex","Name","Cabin","Embarked","Ticket","Age"],axis=1)
#print(x_test)

y_test=df["Age"]
print(y_test)

#Fitting and training data

linear=linear_model.LinearRegression()
linear.fit(x_train,y_train)

#predicting
predicted=linear.predict(x_test)

for i,j in zip(pd.Series(predicted),y_test):
    print('Predicted=',round(i,3),'\t Actual=',j)



'''
*****************************************************************
OUTPUT :~
*****************************************************************

PassengerId    0
Survived       0
Pclass         0
SibSp          0
Parch          0
Fare           0
dtype: int64
5      NaN
17     NaN
19     NaN
26     NaN
28     NaN
29     NaN
31     NaN
32     NaN
36     NaN
42     NaN
45     NaN
46     NaN
47     NaN
48     NaN
55     NaN
```

```
64     NaN
65     NaN
76     NaN
77     NaN
82     NaN
87     NaN
95     NaN
101    NaN
107    NaN
109    NaN
121    NaN
126    NaN
128    NaN
140    NaN
154    NaN
        ..
718    NaN
727    NaN
732    NaN
738    NaN
739    NaN
740    NaN
760    NaN
766    NaN
768    NaN
773    NaN
776    NaN
778    NaN
783    NaN
790    NaN
792    NaN
793    NaN
815    NaN
825    NaN
826    NaN
828    NaN
832    NaN
837    NaN
839    NaN
846    NaN
849    NaN
859    NaN
863    NaN
868    NaN
878    NaN
888    NaN

Name: Age, Length: 177, dtype: float64

Predicted= 28.951      Actual= nan
Predicted= 30.131      Actual= nan
Predicted= 22.189      Actual= nan
Predicted= 28.976      Actual= nan
```

```
Predicted= 22.183       Actual= nan
Predicted= 28.967       Actual= nan
Predicted= 32.188       Actual= nan
Predicted= 22.186       Actual= nan
Predicted= 22.195       Actual= nan
Predicted= 28.972       Actual= nan
Predicted= 28.971       Actual= nan
Predicted= 24.874       Actual= nan
Predicted= 22.191       Actual= nan
Predicted= 20.796       Actual= nan
Predicted= 37.838       Actual= nan
Predicted= 44.742       Actual= nan
Predicted= 17.305       Actual= nan
Predicted= 28.984       Actual= nan
Predicted= 28.982       Actual= nan
Predicted= 22.203       Actual= nan
Predicted= 28.985       Actual= nan
Predicted= 28.988       Actual= nan
Predicted= 28.992       Actual= nan
Predicted= 22.211       Actual= nan
Predicted= 17.982       Actual= nan
Predicted= 28.997       Actual= nan
Predicted= 29.003       Actual= nan
Predicted= 17.221       Actual= nan
Predicted= 27.308       Actual= nan
Predicted= 29.019       Actual= nan
Predicted= 29.0         Actual= nan
Predicted= -5.386       Actual= nan
Predicted= 36.791       Actual= nan
Predicted= 44.805       Actual= nan
Predicted= 16.003       Actual= nan
Predicted= -5.379       Actual= nan
Predicted= 36.942       Actual= nan
Predicted= 44.451       Actual= nan
Predicted= 18.137       Actual= nan
Predicted= 29.027       Actual= nan
Predicted= 22.243       Actual= nan
Predicted= -5.371       Actual= nan
Predicted= 25.047       Actual= nan
Predicted= 29.034       Actual= nan
Predicted= 16.021       Actual= nan
Predicted= 29.044       Actual= nan
Predicted= 24.956       Actual= nan
Predicted= 18.156       Actual= nan
Predicted= 29.053       Actual= nan
Predicted= 37.256       Actual= nan
Predicted= 29.049       Actual= nan
Predicted= 29.051       Actual= nan
Predicted= 44.764       Actual= nan
Predicted= 22.269       Actual= nan
Predicted= 37.2         Actual= nan
Predicted= 44.843       Actual= nan
Predicted= 44.822       Actual= nan
```

```
Predicted= 37.997       Actual= nan
Predicted= 22.278       Actual= nan
Predicted= 14.075       Actual= nan
Predicted= 30.24        Actual= nan
Predicted= 29.06        Actual= nan
Predicted= 36.8         Actual= nan
Predicted= -5.329       Actual= nan
Predicted= 14.085       Actual= nan
Predicted= 32.484       Actual= nan
Predicted= 29.073       Actual= nan
Predicted= 18.184       Actual= nan
Predicted= 44.732       Actual= nan
Predicted= 29.09        Actual= nan
Predicted= 22.296       Actual= nan
Predicted= 22.297       Actual= nan
Predicted= 24.983       Actual= nan
Predicted= 22.309       Actual= nan
Predicted= 22.302       Actual= nan
Predicted= 33.266       Actual= nan
Predicted= 29.09        Actual= nan
Predicted= 29.094       Actual= nan
Predicted= 16.083       Actual= nan
Predicted= 29.099       Actual= nan
Predicted= 29.115       Actual= nan
Predicted= 37.247       Actual= nan
Predicted= 29.098       Actual= nan
Predicted= 29.102       Actual= nan
Predicted= 29.114       Actual= nan
Predicted= 29.107       Actual= nan
Predicted= 18.213       Actual= nan
Predicted= 22.323       Actual= nan
Predicted= 24.947       Actual= nan
Predicted= 29.112       Actual= nan
Predicted= 33.747       Actual= nan
Predicted= 29.118       Actual= nan
Predicted= 29.115       Actual= nan
Predicted= 37.265       Actual= nan
Predicted= 29.121       Actual= nan
Predicted= 29.129       Actual= nan
Predicted= 44.522       Actual= nan
Predicted= 37.27        Actual= nan
Predicted= 16.11        Actual= nan
Predicted= 24.96        Actual= nan
Predicted= 29.03        Actual= nan
Predicted= 29.021       Actual= nan
Predicted= 29.135       Actual= nan
Predicted= 38.128       Actual= nan
Predicted= 29.131       Actual= nan
Predicted= 28.893       Actual= nan
Predicted= 29.148       Actual= nan
Predicted= 29.148       Actual= nan
Predicted= 42.008       Actual= nan
Predicted= 29.151       Actual= nan
```

```
Predicted= 20.552       Actual= nan
Predicted= 29.045       Actual= nan
Predicted= 30.302       Actual= nan
Predicted= 29.149       Actual= nan
Predicted= 41.932       Actual= nan
Predicted= 29.153       Actual= nan
Predicted= 29.149       Actual= nan
Predicted= 29.15        Actual= nan
Predicted= 29.163       Actual= nan
Predicted= 22.373       Actual= nan
Predicted= 25.073       Actual= nan
Predicted= 29.147       Actual= nan
Predicted= 29.158       Actual= nan
Predicted= 27.576       Actual= nan
Predicted= 30.033       Actual= nan
Predicted= 29.174       Actual= nan
Predicted= 29.165       Actual= nan
Predicted= 44.709       Actual= nan
Predicted= 29.181       Actual= nan
Predicted= 18.284       Actual= nan
Predicted= 29.171       Actual= nan
Predicted= 29.177       Actual= nan
Predicted= 45.352       Actual= nan
Predicted= 25.069       Actual= nan
Predicted= 21.67        Actual= nan
Predicted= 29.186       Actual= nan
Predicted= 29.182       Actual= nan
Predicted= 22.399       Actual= nan
Predicted= 29.184       Actual= nan
Predicted= 29.189       Actual= nan
Predicted= 33.818       Actual= nan
Predicted= 37.337       Actual= nan
Predicted= 29.189       Actual= nan
Predicted= 21.687       Actual= nan
Predicted= 22.416       Actual= nan
Predicted= 17.527       Actual= nan
Predicted= 44.983       Actual= nan
Predicted= 29.092       Actual= nan
Predicted= 22.426       Actual= nan
Predicted= 37.357       Actual= nan
Predicted= 29.212       Actual= nan
Predicted= 29.212       Actual= nan
Predicted= 38.157       Actual= nan
Predicted= 29.121       Actual= nan
Predicted= 44.807       Actual= nan
Predicted= 24.994       Actual= nan
Predicted= 29.234       Actual= nan
Predicted= 29.227       Actual= nan
Predicted= 29.228       Actual= nan
Predicted= 23.421       Actual= nan
Predicted= 29.232       Actual= nan
Predicted= -5.167       Actual= nan
Predicted= 44.949       Actual= nan
```

```
Predicted= 45.415      Actual= nan
Predicted= 29.256      Actual= nan
Predicted= 28.518      Actual= nan
Predicted= 22.461      Actual= nan
Predicted= 29.255      Actual= nan
Predicted= 29.244      Actual= nan
Predicted= 38.195      Actual= nan
Predicted= -5.149      Actual= nan
Predicted= 33.327      Actual= nan
Predicted= 29.264      Actual= nan
Predicted= -5.143      Actual= nan
Predicted= 29.233      Actual= nan
Predicted= 29.26       Actual= nan
Predicted= 23.457      Actual= nan

'''

'''
```

****
3. Load iris dataset. Use K Means to cluster all datapoint into cluster using python code. Use elbow method to find number of clusters.
****

```
'''

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics

#loading data
datapath="/home/student/Iris.csv"
data=pd.read_csv(datapath,sep=",",names=["sepal_length","sepal_width","Petal_le
ngth","Petal_width","Class"])
#print(data)

#preprocessing
dic={"Iris-setosa":0,"Iris-versicolor":1,"Iris-virginica":2}
cl=[]
for i in data["Class"]:
    cl.append(dic[i])
data["Class_new"]=pd.Series(cl)

X=data.drop("Class",axis=1)
print(X)
y=data["Class_new"]

#Splitting
x_train=X[:120]
x_test=X[120:]
y_train=y[:120]
```

```
y_test=y[120:]
"""
print("x_train=",len(x_train))
print("x_test=",len(x_test))
print("y_train=",len(y_train))
print("y_test=",len(y_test))
"""
distortions = []
K = range(1,10)#possible number of clusters
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X)
    kmeanModel.fit(X)
    distortions.append(sum(np.min(cdist(X,         kmeanModel.cluster_centers_,
'euclidean'), axis=1)) / X.shape[0])

print('distortions=',distortions)
# Plot the elbow
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()

k_means=KMeans(n_clusters=3,random_state=0)
k_means.fit(x_train)
predicted=k_means.predict(x_test)
centroids=k_means.cluster_centers_
labels=k_means.labels_

for i,j in zip(pd.Series(predicted),y_test):
    print("Predicted Cluster=",i,'\t',"Actual Cluster=",j)

print('Accuracy Score=',metrics.accuracy_score(predicted,y_test))


'''
```

****

OUTPUT :~

****

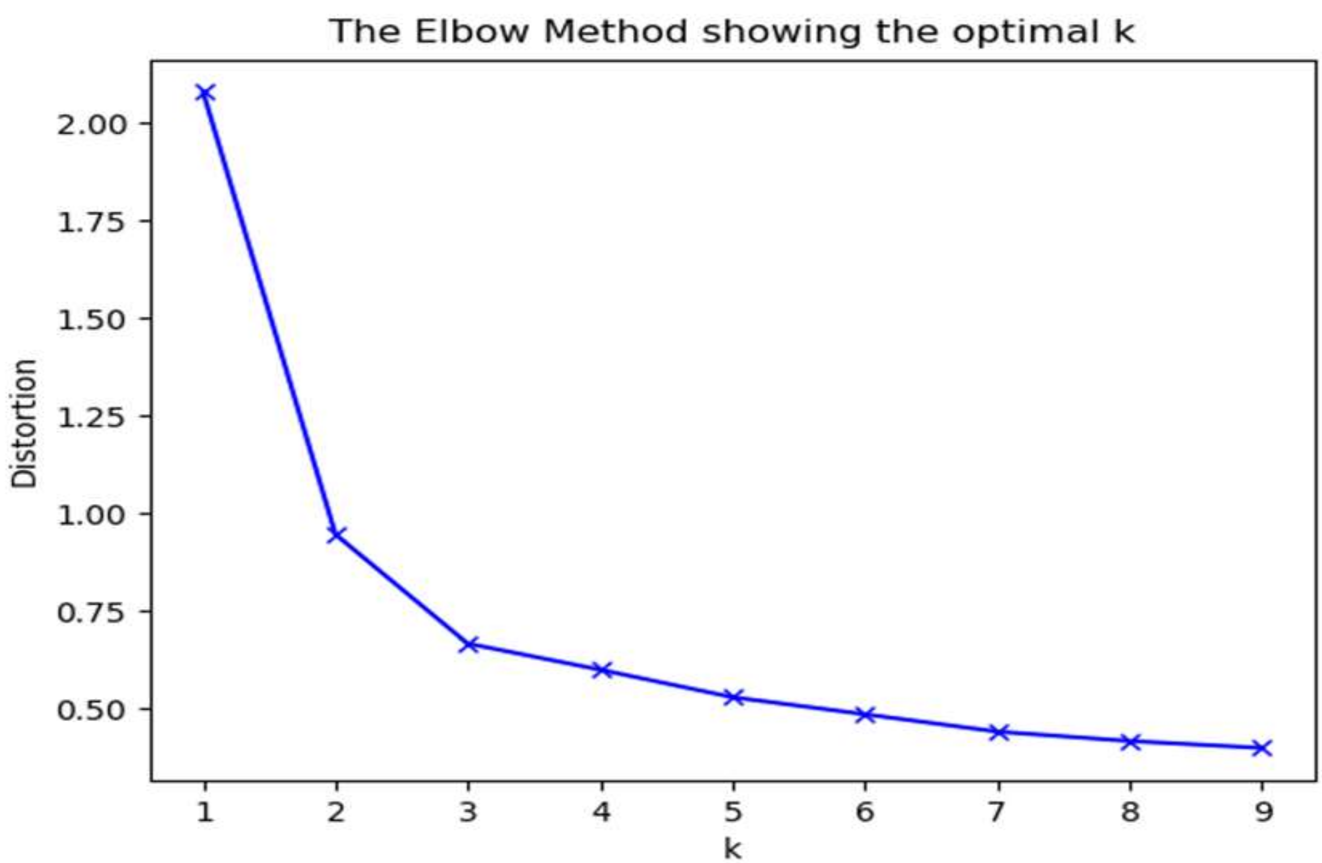
```
distortions=   [2.0802006916346016,   0.9467986410243767,   0.6671852795977241,
0.6002602672284263,          0.5284154323556666,          0.48413335634239646,
0.44301507282909064, 0.41788673917433583, 0.4007945028681197]

Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 0   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
```

```
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2
Predicted Cluster= 2   Actual Cluster= 2

Accuracy Score= 0.9569866666666667

'''
```



```
'''
*******************************************************************************
4. Collect tweets from twitter api.i.e. download twitter data. Perform
text/string analysis on data to find if its sentiment is positive or negative.
*******************************************************************************
'''

import nltk, os
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem.porter import PorterStemmer
from sklearn.naive_bayes import GaussianNB
import pickle

class EmotionClassifier():
    dataPath= "/home/student/Emotions/"

    def __init__(self):#__init__=default method
        self.labels=['Motivational','Emotinal','Happy','Anxiety','Anger']

self.vectorizer=TfidfVectorizer(max_features=None,strip_accents='unicode',analy
zer='words',ngram_range=(1,3),use_idf=1,smooth_idf = 1,stop_words = 'english')

    def getData(self):

        x_data=[]
        y_data=[]
        labels=['Motivational','Emotinal','Happy','Anxiety','Anger']
        for label in labels:
            with open(self.dataPath + label + ".txt") as file_name:
                data_file=file_name.readlines()
                for data in data_file:
                    tokens=nltk.word_tokenize(data)
                    stems=[]
                    for item in map(lambda x : x.decode("utf-8"), tokens):
                        stems.append(PorterStemmer().stem(item))
                    data = " ".join(stems)
                    x_data.append(data)
                    y_data.append(self.labels.index(label))
        return x_data,y_data

    def train(self):
        C=0.01
        x_train, y_train = self.getData()
        vectors_train = self.vectorizer.fit_transform(x_train)
        if os.path.exists("./emotion_classifier.pkl"):
            with open('./emotion_classifier.pkl', 'rb') as fid:
                self.classifier = pickle.load(fid)
                self.classifier.fit(vectors_train, y_train)
        else:
            self.classifier = GaussianNB()
            self.classifier.fit(vectors_train, y_train)
            with open('./emotion_classifier.pkl', 'wb+') as fid:
```

```
                pickle.dump(self.classifier, fid)

    def test(self, query):
        self.train()
        tokens = nltk.word_tokenize(query)
        stems = []
        for item in map(lambda x : x.decode("utf-8"), tokens):
            stems.append(PorterStemmer().stem(item))
        query = " ".join(stems)
        classifier_result = {"class":"", "confidence":""}
        vectors_test = self.vectorizer.transform([query])
        pred = self.classifier.predict(vectors_test)
        predp = self.classifier.predict_proba(vectors_test)
        predp = predp.tolist()[0]
        print("predp : : ", predp)
        max_ind =  predp.index(max(predp))
        classifier_result["class"] = self.labels[max_ind]
        classifier_result["confidence"] = str(max(predp)*100.0)

        return pred,self.labels[pred[0]]


if __name__ == '__main__':
    log = EmotionClassifier()
    while(True):
        query = input("*********** Enter Query ***********\n\n")
        result = log.test(query)
        print(result)
        print("\n")


'''
```

****
5. Load dow jones sensex data. Use linear regression to predict open, close and low prices.(3 different regression models to predict each open, close and low price one at a time.
****

```
'''


import pandas as pd
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn import linear_model

#Loading data
dataPath="/home/student/dow_jones_index_data.csv"
data=pd.read_csv(dataPath,sep=",")

#print data
#print len(data["stock"].unique())
```

```
#data Preprocessing
stock_dummy=[]
st={
    'AA':0,  'AXP':1,  'BA':2,  'BAC':3,  'CAT':4,  'CSCO':5,  'CVX':6,  'DD':7,
'DIS':8, 'GE':9, 'HD':10, 'HPQ':11, 'IBM':12,
    'INTC':13,  'JNJ':14,  'JPM':15,  'KRFT':16,  'KO':17,  'MCD':18,  'MMM':19,
'MRK':20, 'MSFT':21, 'PFE':22, 'PG':23, 'T':24,
    'TRV':25, 'UTX':26, 'VZ':27, 'WMT':28, 'XOM':29
   }

for s in data["stock"]:
    stock_dummy.append(st[s])

data["stock_dummy"]=pd.Series(stock_dummy)

next_open=[]
next_close=[]
o=[]
h=[]
lo=[]
c=[]
for                                i,j,k,l,m,n                                in
zip(data["next_weeks_open"],data["next_weeks_close"],data["open"],data["high"],
data["low"],data["close"]):
    i=float(i.replace("$","")) #replacing $ with empty space because with $ it
will consider the value as string
    j=float(j.replace("$",""))
    k=float(k.replace("$",""))
    l=float(l.replace("$",""))
    m=float(m.replace("$",""))
    n=float(n.replace("$",""))
    next_open.append(i)
    next_close.append(j)
    o.append(k)
    h.append(l)
    lo.append(m)
    c.append(n)
data["next_weeks_open_new"]=pd.Series(next_open)
data["next_weeks_close_new"]=pd.Series( next_close)
data["open_new"]=pd.Series(o)
data["high_new"]=pd.Series(h)
data["low_new"]=pd.Series(lo)
data["close_new"]=pd.Series(c)

data["date"]=pd.to_datetime(data["date"],format="%m/%d/%Y")

data.fillna(data.mean(),inplace=True)#This is done to replace "na" with mean
#X=data.drop(["stock","next_weeks_open","next_weeks_close","open","high","low",
"close","percent_return_next_dividend"],axis=1)


X=data.drop(["next_weeks_open","next_weeks_close","open","open_new","high","clo
se","low","date","stock"],axis=1)
```

```
y=data["open_new"]


#Spliting data in training and testing
x_train=X[:730]
x_test=X[730:]
y_train=y[:730]
y_test=y[730:]

#create linear regression object
linear=linear_model.LinearRegression()

#Train model
linear.fit(x_train,y_train)

#predict Output
predicted=linear.predict(x_test)

for i,j in zip(pd.Series(predicted),y_test):
    print("Predicted=",round(i,1),'\t',"Actual=",round(j,1))
print('mean_sq_err=',mean_squared_error(predicted,y_test))

X=data.drop(["next_weeks_open","next_weeks_close","open","close_new","high","cl
ose","low","date","stock"],axis=1)
y=data["close_new"]


#Spliting data in training and testing
x_train=X[:730]
x_test=X[730:]
y_train=y[:730]
y_test=y[730:]


#create linear regression object
linear=linear_model.LinearRegression()

#Train model
linear.fit(x_train,y_train)

#predict Output
predicted=linear.predict(x_test)


for i,j in zip(pd.Series(predicted),y_test):
    print("Predicted=",round(i,1),'\t',"Actual=",round(j,1))
print('mean_sq_err=',mean_squared_error(predicted,y_test))

X=data.drop(["next_weeks_open","next_weeks_close","open","open_new","high","clo
se","low","date","stock","low_new"],axis=1)
y=data["low_new"]
```

```
#Spliting data in training and testing
x_train=X[:730]
x_test=X[730:]
y_train=y[:730]
y_test=y[730:]

#create linear regression object
linear=linear_model.LinearRegression()

#Train model
linear.fit(x_train,y_train)

#predict Output
predicted=linear.predict(x_test)

for i,j in zip(pd.Series(predicted),y_test):
    print("Predicted=",round(i,1),'\t',"Actual=",round(j,1))

print('\n\nmean_sq_err=',mean_squared_error(predicted,y_test))



'''
*****************************************************************
OUTPUT :~
*****************************************************************

Predicted= 55.0   Actual= 55.0
Predicted= 55.7   Actual= 55.6
Predicted= 55.0   Actual= 54.9
Predicted= 54.8   Actual= 54.9
Predicted= 53.9   Actual= 53.9
Predicted= 52.6   Actual= 52.9
Predicted= 53.0   Actual= 52.7
Predicted= 83.5   Actual= 83.9
Predicted= 84.6   Actual= 84.3
Predicted= 84.8   Actual= 86.0
Predicted= 83.7   Actual= 83.1
Predicted= 86.5   Actual= 86.3
Predicted= 86.9   Actual= 88.1
Predicted= 82.4   Actual= 83.0
Predicted= 80.7   Actual= 80.2
Predicted= 80.8   Actual= 80.2
Predicted= 82.7   Actual= 83.3
Predicted= 81.0   Actual= 80.9
Predicted= 79.9   Actual= 80.0
Predicted= 79.3   Actual= 78.7
mean_sq_err= 0.27804629317898054

student@student-OptiPlex-3020:~$ python 5LinearRegressions.py

Predicted= 55.0   Actual= 55.0
```

```
Predicted= 55.7   Actual= 55.6
Predicted= 55.0   Actual= 54.9
Predicted= 54.8   Actual= 54.9
Predicted= 53.9   Actual= 53.9
Predicted= 52.6   Actual= 52.9
Predicted= 53.0   Actual= 52.7
Predicted= 83.5   Actual= 83.9
Predicted= 84.6   Actual= 84.3
Predicted= 84.8   Actual= 86.0
Predicted= 83.7   Actual= 83.1
Predicted= 86.5   Actual= 86.3
Predicted= 86.9   Actual= 88.1
Predicted= 82.4   Actual= 83.0
Predicted= 80.7   Actual= 80.2
Predicted= 80.8   Actual= 80.2
Predicted= 82.7   Actual= 83.3
Predicted= 81.0   Actual= 80.9
Predicted= 79.9   Actual= 80.0
Predicted= 79.3   Actual= 78.7

mean_sq_err= 0.28804629317898054

student@student-OptiPlex-3020:~$ python 5LinearRegressions.py

Predicted= 55.7   Actual= 55.7
Predicted= 55.2   Actual= 55.3
Predicted= 54.9   Actual= 54.7
Predicted= 53.8   Actual= 53.7
Predicted= 53.0   Actual= 52.7
Predicted= 52.7   Actual= 52.8
Predicted= 52.6   Actual= 52.4
Predicted= 84.3   Actual= 84.7
Predicted= 85.9   Actual= 86.0
Predicted= 83.6   Actual= 84.3
Predicted= 86.0   Actual= 86.4
Predicted= 87.8   Actual= 88.0
Predicted= 83.3   Actual= 82.7
Predicted= 80.7   Actual= 80.9
Predicted= 80.8   Actual= 81.6
Predicted= 82.9   Actual= 82.6
Predicted= 81.2   Actual= 81.2
Predicted= 80.2   Actual= 79.8
Predicted= 78.9   Actual= 79.0
Predicted= 77.5   Actual= 76.8

mean_sq_err= 0.16849483657963014

Predicted= 54.7   Actual= 54.7
Predicted= 54.5   Actual= 55.0
Predicted= 54.1   Actual= 54.1
Predicted= 53.2   Actual= 53.0
Predicted= 52.3   Actual= 52.7
Predicted= 52.0   Actual= 51.8
```

```
Predicted= 51.8  Actual= 52.4
Predicted= 83.1  Actual= 82.6
Predicted= 84.1  Actual= 84.1
Predicted= 83.4  Actual= 82.4
Predicted= 84.3  Actual= 82.4
Predicted= 85.8  Actual= 85.9
Predicted= 82.8  Actual= 81.6
Predicted= 80.4  Actual= 79.4
Predicted= 80.2  Actual= 79.6
Predicted= 80.7  Actual= 80.1
Predicted= 80.5  Actual= 80.2
Predicted= 78.8  Actual= 79.7
Predicted= 78.1  Actual= 78.3
Predicted= 76.2  Actual= 76.8

mean_sq_err= 0.4846843012806531
'''


'''
********************************************************************************
```

6. Download sonar dataset. Check description of the data. Understand dataset, check for missing values if any. Perform dummy variable conversion if required. Design appropriate model to predict if the data point belongs to Rock or Mine class.

```
********************************************************************************
'''

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
#data loading
datapath="/home/student/rock1.csv"
data=pd.read_csv(datapath,sep=",",names=["V1","V2","V3","V4","V5","V6","V7","V8
","V9","V10","V11","V12","V13","V14","V15","V16","V17","V18","V19","V20","V21",
"V22","V23","V24","V25",
"V26","V27","V28","V29","V30","V31","V32","V33","V34","V35","V36","V37","V38","
V39","V40","V41","V42","V43","V44","V45",    "V46",    "V47","V48","V49","V50",
"V51", "V52", "V53", "V54","V55", "V56", "V57", "V58", "V59", "V60", "Class"])


#Data Cleaning(Finding missing data)
print("Null present..?=",data.isnull().sum())

dic={"R":0,"M":1}
c=[]
for i in data["Class"]:
    c.append(dic[i])
data["Class_new"]=pd.Series(c)

#Splitting dataset into training and validation

X=data.drop(["Class","Class_new"],axis=1)
```

```
#print(X)
y=data["Class_new"]
#print(y)

x_train=X[:180]
x_test=X[180:]
y_train=y[:180]
y_test=y[180:]

model=RandomForestClassifier(n_estimators=700,max_features='log2')
model.fit(x_train, y_train)
pred = model.predict(x_test)

label=["Rock","Mine"]
for i,j in zip(pd.Series(pred), y_test):
    print("predicted=",label[i],"\tActual=",label[j])

print('Accuracy Score=',metrics.accuracy_score(pred,y_test))


'''
******************************************************************
OUTPUT :~
******************************************************************

Null present..?= V1       0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
```

```
V29      0
V30      0
         ..
V32      0
V33      0
V34      0
V35      0
V36      0
V37      0
V38      0
V39      0
V40      0
V41      0
V42      0
V43      0
V44      0
V58      0
V59      0
V60      0
Class    0

Length: 61, dtype: int64

predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Mine  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Rock  Actual= Mine
predicted= Rock  Actual= Mine

Accuracy Score= 0.6625769875714286
```

```
'''
*******************************************************************************
7. Load breast cancer CSV. Treat string column bare nuclei and convert it into
integers by iterating over pandas series and typecasting it in integer. Design
random forest classifier to predict type of the cancer.
*******************************************************************************
'''

import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

#loading data
datapath="/home/student/breast_cancer.csv"
data=pd.read_csv(datapath,sep=",",names=["Sample code number","Clump
thickness","Uniformity of cell size","Uniformity of cell shape","Marginal
adhesion","Single epithelial cell size","Bare Nuclei","Bland Chromatin","Normal
Nucleoli","Mitoses","class"])

#splitting dataset
X=data.drop(["Bare Nuclei","class"],axis=1)
y=data["class"]
X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size=0.2,random_state=123,stratify=y)

#Training data

clf=RandomForestClassifier(n_estimators=700,max_features='log2')
clf.fit(X_train, y_train)
importances = clf.feature_importances_
indices = np.argsort(importances)
pred = clf.predict(X_test)

#Printing output
for i,j in zip(pd.Series(pred),y_test):
    print("Predicted=",int(round(i)),"\tActual=",j)

print('Accuracy Score =', metrics.accuracy_score(pred,y_test))


'''
*****************************************************************
OUTPUT :~
*****************************************************************

Predicted= 2    Actual= 2
Predicted= 4    Actual= 4
Predicted= 2    Actual= 2
Predicted= 2    Actual= 2
Predicted= 4    Actual= 4
```

```
Predicted= 2     Actual= 2
Predicted= 4     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 4     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 4     Actual= 4
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 4     Actual= 4
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2
Predicted= 2     Actual= 2

Accuracy Score = 0.9376998571428572

'''


'''
```

****
8. Download auto mpg data from kaggle.https://www.kaggle.com/uciml/autompg-dataset . Design a model to predict mileage per gallon performance of a vehicle.
****

```
'''

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import mean_squared_error


datapath= "/home/student/auto-mpg.csv"
```

```
data=pd.read_csv(datapath,sep=",")
#print(data)

data1 = data[data.horsepower != '?']


X=data1.drop(["mpg","car name"],axis=1)
#print(X)
y=data1["mpg"]

#print('y=',y)

#print(data1.isnull().any())
x_train=X[:350]
x_test=X[350:]
y_train=y[:350]
y_test=y[350:]

linear=linear_model.LinearRegression()
linear.fit(x_train,y_train)
predicted=linear.predict(x_test)

for i,j in zip(pd.Series(predicted),y_test):
    print('Predicted=',i,'\t Actual=',j)

print( 'mean_sq_err=',mean_squared_error(predicted,y_test))

'''
*******************************************************************
OUTPUT :~
*******************************************************************

Predicted= 33.560632551226774     Actual= 33.7
Predicted= 32.77634861744495      Actual= 32.4
Predicted= 30.876814141711446     Actual= 32.9
Predicted= 31.26140112825738      Actual= 31.6
Predicted= 26.469214016323686     Actual= 28.1
Predicted= 26.18909514811042      Actual= 30.7
Predicted= 28.741931363625884     Actual= 25.4
Predicted= 28.199163640702185     Actual= 24.2
Predicted= 23.857247814822134     Actual= 22.4
Predicted= 23.06360798373601      Actual= 26.6
Predicted= 25.947968737719545     Actual= 20.2
Predicted= 23.874173571664574     Actual= 17.6
Predicted= 29.039001616589797     Actual= 28.0
Predicted= 28.800780278228277     Actual= 27.0
Predicted= 30.2856709415157       Actual= 34.0
Predicted= 29.1879608033728       Actual= 31.0
Predicted= 29.844853060306722     Actual= 29.0
Predicted= 28.75016273212748      Actual= 27.0
Predicted= 27.722145716798234     Actual= 24.0
Predicted= 34.29443499114235      Actual= 36.0
Predicted= 35.393709503738705     Actual= 37.0
```

```
Predicted= 35.71586911966588      Actual= 31.0
Predicted= 32.14755744030842      Actual= 38.0
Predicted= 31.996694115016826     Actual= 36.0
Predicted= 34.59345045971408      Actual= 36.0
Predicted= 34.32991898916876      Actual= 36.0
Predicted= 34.236075115520734     Actual= 34.0
Predicted= 35.699674070896776     Actual= 38.0
Predicted= 35.71649789901129      Actual= 32.0
Predicted= 35.54492580635198      Actual= 38.0
Predicted= 26.745004389465908     Actual= 25.0
Predicted= 27.92090054415669      Actual= 38.0
Predicted= 29.626516427948363     Actual= 26.0
Predicted= 28.09987775432403      Actual= 22.0
Predicted= 31.717856645695438     Actual= 32.0
Predicted= 30.72129468897828      Actual= 36.0
Predicted= 27.41761692936848      Actual= 27.0
Predicted= 28.25606288771628      Actual= 27.0
Predicted= 33.8270334444055       Actual= 44.0
Predicted= 31.146619814601987     Actual= 32.0
Predicted= 29.1521007673318       Actual= 28.0
Predicted= 28.528093102567116     Actual= 31.0

mean_sq_err= 13.926463678743081

'''

'''
```

****

09. Using rapid miner load wine quality dataset. Perform agglomerative clustering to group the data. Draw dendrites to confirm on number of clusters.

****

Above Shows dendogram plot.

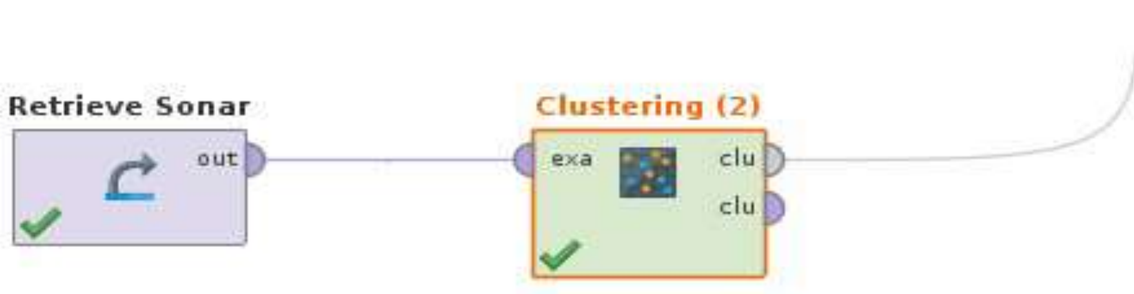
Hierarchical Cluster Model

```
Number of clusters :3197
Number of items :1599
```

****

10. Load sonar dataset from sample datasets in Rapid miner. Use appropriate algorithm to predict class of the specimen if rock or mine in rapid miner.

****



Cluster Model

```
Cluster 0: 100 items
Cluster 1: 108 items
Total number of items: 208
```

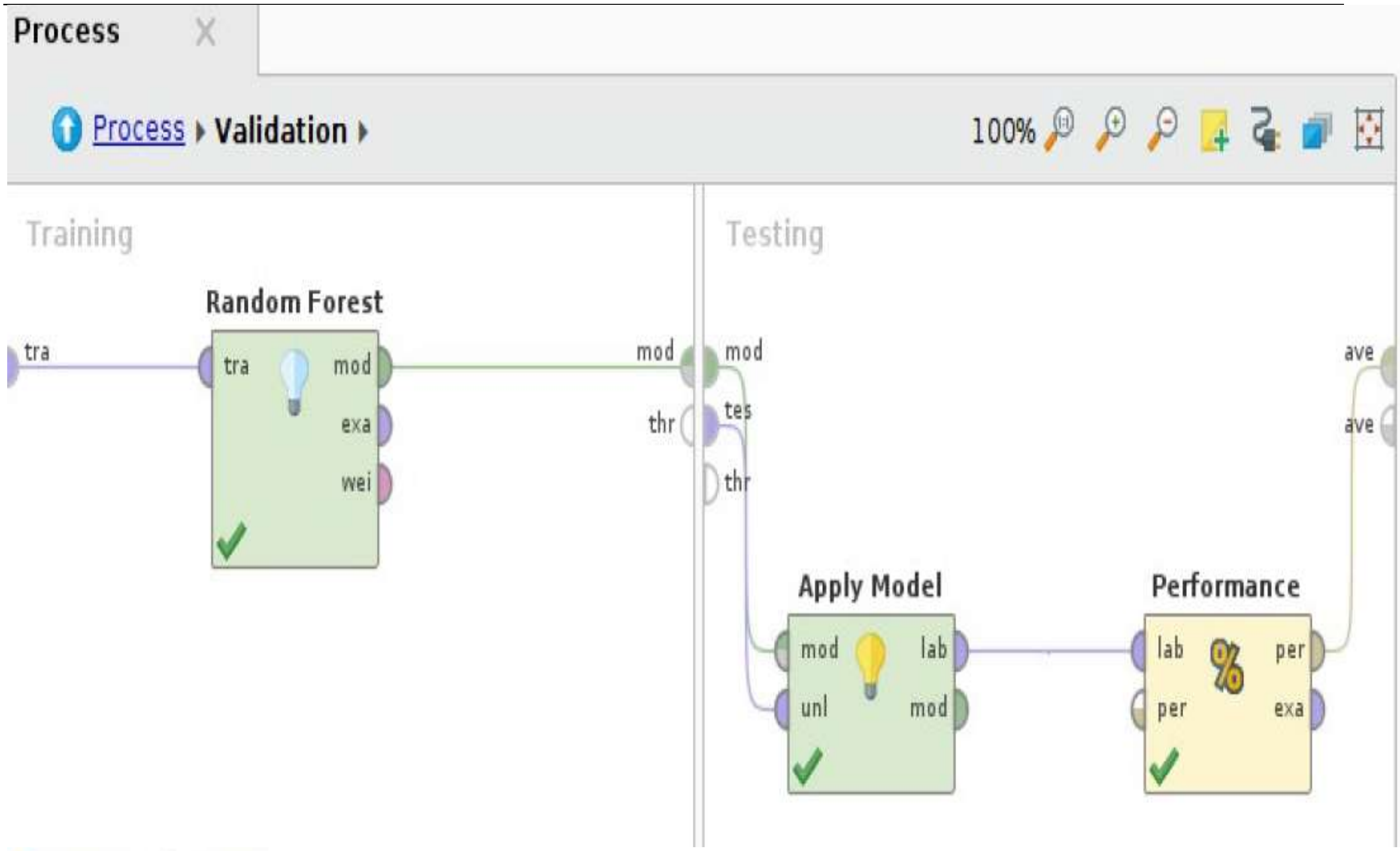


Table View   Plot View

accuracy: 70.97%

	true Rock	true Mine	class precision
pred. Rock	17	6	73.91%
pred. Mine	12	27	69.23%
class recall	58.62%	81.82%