------------------------KIDNEY STONE DATASET-------------------------------

IMPORTING THE LIBRARIES ¶

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
import warnings
import os
warnings.filterwarnings("ignore")
import datetime
import re
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,r2_scor
from sklearn.tree import DecisionTreeClassifier
```

Loading the dataset

In [2]:

```
data=pd.read_csv(r"C:\Users\Admin\Downloads\kidney-stone-dataset.csv")
data.head()
```

Out[2]:

	Unnamed: 0	gravity	ph	osmo	cond	urea	calc	target
0	0	1.021	4.91	725	14.0	443	2.45	0
1	1	1.017	5.74	577	20.0	296	4.49	0
2	2	1.008	7.20	321	14.9	101	2.36	0
3	3	1.011	5.51	408	12.6	224	2.15	0
4	4	1.005	6.52	187	7.5	91	1.16	0

In [3]:

```
data.dropna()
```

Out[3]:

	Unnamed: 0	gravity	ph	osmo	cond	urea	calc	target
0	0	1.021000	4.910000	725	14.000000	443	2.450000	0
1	1	1.017000	5.740000	577	20.000000	296	4.490000	0
2	2	1.008000	7.200000	321	14.900000	101	2.360000	0
3	3	1.011000	5.510000	408	12.600000	224	2.150000	0
4	4	1.005000	6.520000	187	7.500000	91	1.160000	0
...	...	...	...	...	...	...	...	...
85	85	1.021452	5.556081	756	24.241481	367	7.669120	1
86	86	1.016501	6.900257	549	20.549790	204	5.775256	1
87	87	1.032754	5.443491	1085	23.188653	576	8.664169	1
88	88	1.023870	5.106433	325	12.124689	50	0.781620	1
89	89	1.013723	6.308943	472	16.907792	174	2.556405	1

90 rows × 8 columns

In [4]:

```
data.dtypes
```

Out[4]:

```
Unnamed: 0      int64
gravity       float64
ph            float64
osmo            int64
cond          float64
urea            int64
calc          float64
target          int64
dtype: object
```

In [5]:

```
data.describe()
```

Out[5]:

	Unnamed: 0	gravity	ph	osmo	cond	urea	calc	t
count	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.00
mean	44.500000	1.017952	6.036651	602.333333	20.621687	258.200000	4.017788	0.50
std	26.124701	0.006780	0.711801	238.459805	7.654448	135.381127	3.016273	0.50
min	0.000000	1.005000	4.760000	187.000000	5.100000	10.000000	0.170000	0.00
25%	22.250000	1.012258	5.536520	411.500000	14.150000	148.250000	1.412500	0.00
50%	44.500000	1.018000	5.936247	572.000000	21.177172	231.500000	3.230000	0.50
75%	66.750000	1.023000	6.490000	778.000000	26.075000	366.250000	5.965127	1.00
max	89.000000	1.034000	7.940000	1236.000000	38.000000	620.000000	13.000000	1.00

In [6]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  90 non-null     int64  
 1   gravity     90 non-null     float64
 2   ph          90 non-null     float64
 3   osmo        90 non-null     int64  
 4   cond        90 non-null     float64
 5   urea        90 non-null     int64  
 6   calc        90 non-null     float64
 7   target      90 non-null     int64  
dtypes: float64(4), int64(4)
memory usage: 5.8 KB
```

In [7]:

```
data.shape
```

Out[7]:

```
(90, 8)
```

In [8]:

```
data.columns
```

Out[8]:

```
Index(['Unnamed: 0', 'gravity', 'ph', 'osmo', 'cond', 'urea', 'calc',
       'target'],
      dtype='object')
```

In [9]:

```
data.value_counts
```

Out[9]:

```
<bound method DataFrame.value_counts of     Unnamed: 0   gravity        ph  osmo       cond  urea      calc  target
0            0  1.021000  4.910000   725  14.000000   443  2.450000       0
1            1  1.017000  5.740000   577  20.000000   296  4.490000       0
2            2  1.008000  7.200000   321  14.900000   101  2.360000       0
3            3  1.011000  5.510000   408  12.600000   224  2.150000       0
4            4  1.005000  6.520000   187   7.500000    91  1.160000       0
..         ...       ...       ...   ...        ...   ...       ...     ...
85          85  1.021452  5.556081   756  24.241481   367  7.669120       1
86          86  1.016501  6.900257   549  20.549790   204  5.775256       1
87          87  1.032754  5.443491  1085  23.188653   576  8.664169       1
88          88  1.023870  5.106433   325  12.124689    50  0.781620       1
89          89  1.013723  6.308943   472  16.907792   174  2.556405       1

[90 rows x 8 columns]>
```

In [10]:

```
data.isnull().sum()
```

Out[10]:

```
Unnamed: 0    0
gravity       0
ph            0
osmo          0
cond          0
urea          0
calc          0
target        0
dtype: int64
```

In [ ]:

```

```

VISUALIZING THE DATA

In [11]:

```
data.corr()
```

Out[11]:

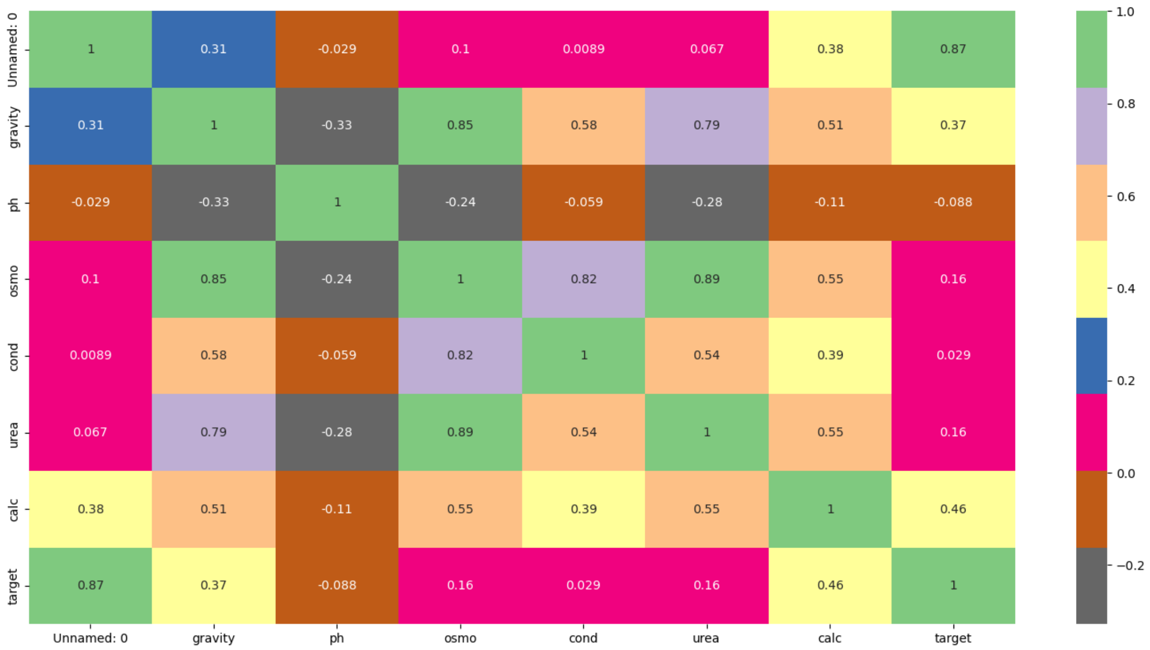
	Unnamed: 0	gravity	ph	osmo	cond	urea	calc	targ
Unnamed: 0	1.000000	0.307917	-0.028881	0.104555	0.008868	0.066740	0.383048	0.8660
gravity	0.307917	1.000000	-0.328780	0.846836	0.575920	0.790409	0.510105	0.3652
ph	-0.028881	-0.328780	1.000000	-0.235852	-0.058851	-0.284943	-0.113775	-0.0876
osmo	0.104555	0.846836	-0.235852	1.000000	0.820609	0.891854	0.551825	0.1562
cond	0.008868	0.575920	-0.058851	0.820609	1.000000	0.543052	0.385034	0.0285
urea	0.066740	0.790409	-0.284943	0.891854	0.543052	1.000000	0.551690	0.1566
calc	0.383048	0.510105	-0.113775	0.551825	0.385034	0.551690	1.000000	0.4643
target	0.866079	0.365280	-0.087613	0.156219	0.028540	0.156647	0.464382	1.0000

In [12]:

```
plt.figure(figsize=(18,9))
sns.heatmap(data.corr(),annot = True, cmap ="Accent_r")
```

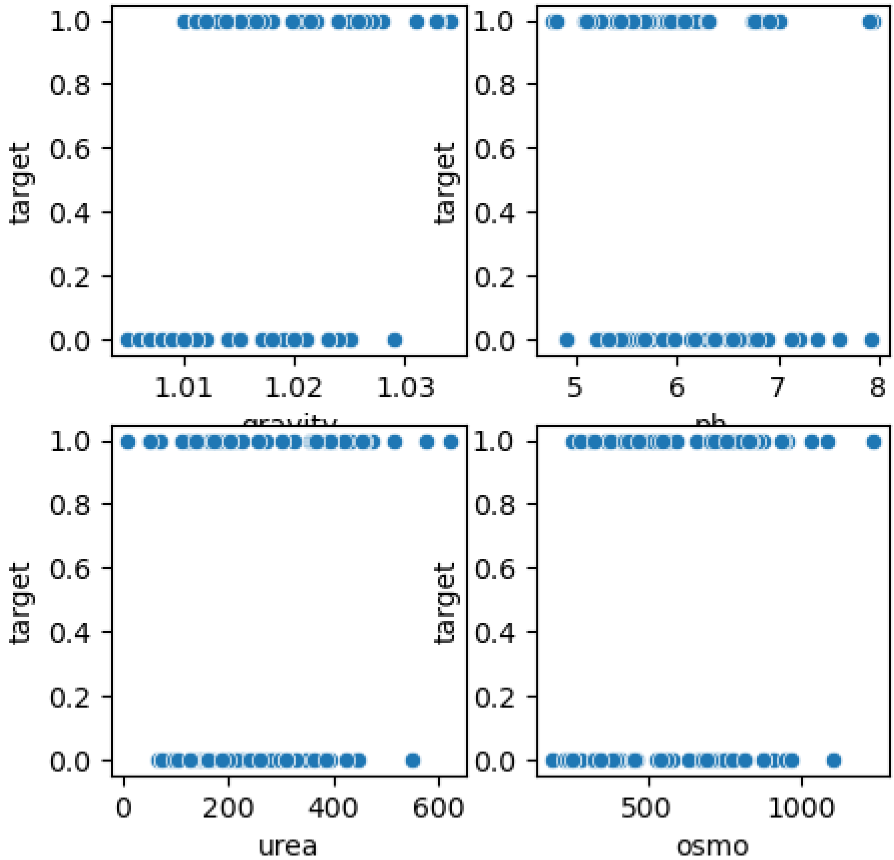
Out[12]:

<AxesSubplot:>



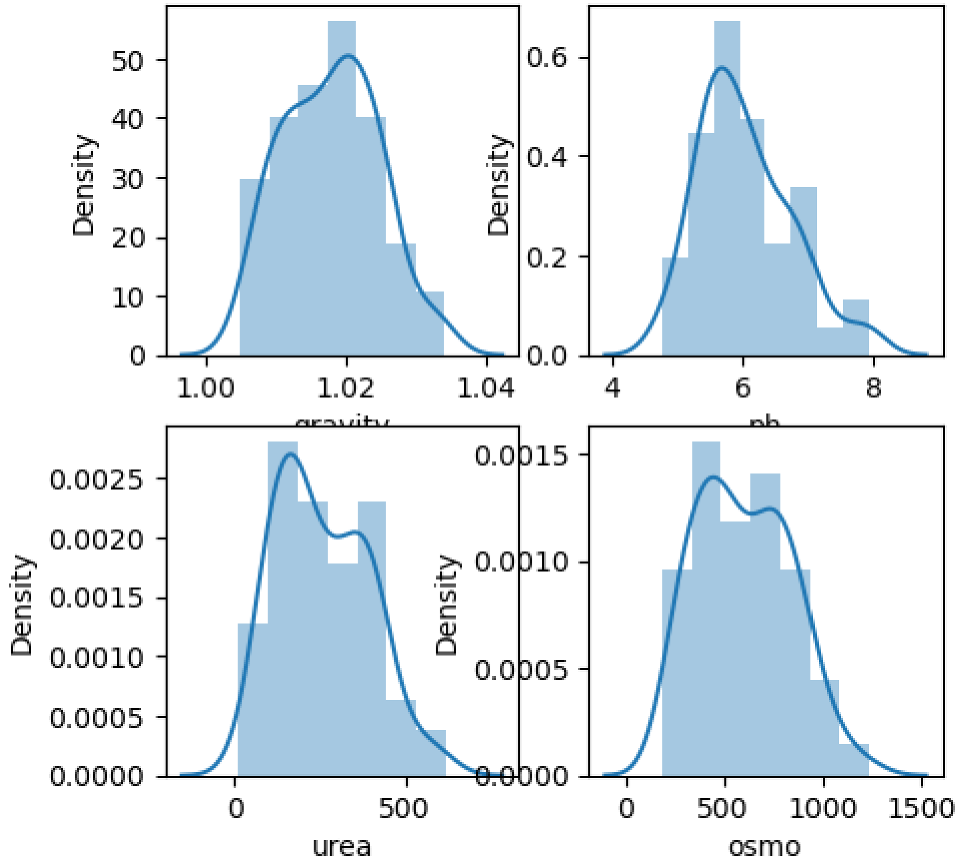
In [13]:

```
features = ['gravity','ph','urea','osmo']

plt.subplots(figsize=(5,5))

for i, col in enumerate(features):
  plt.subplot(2,2,i+1)
  sns.scatterplot(data=data,x=col,y='target')
plt.show()
```



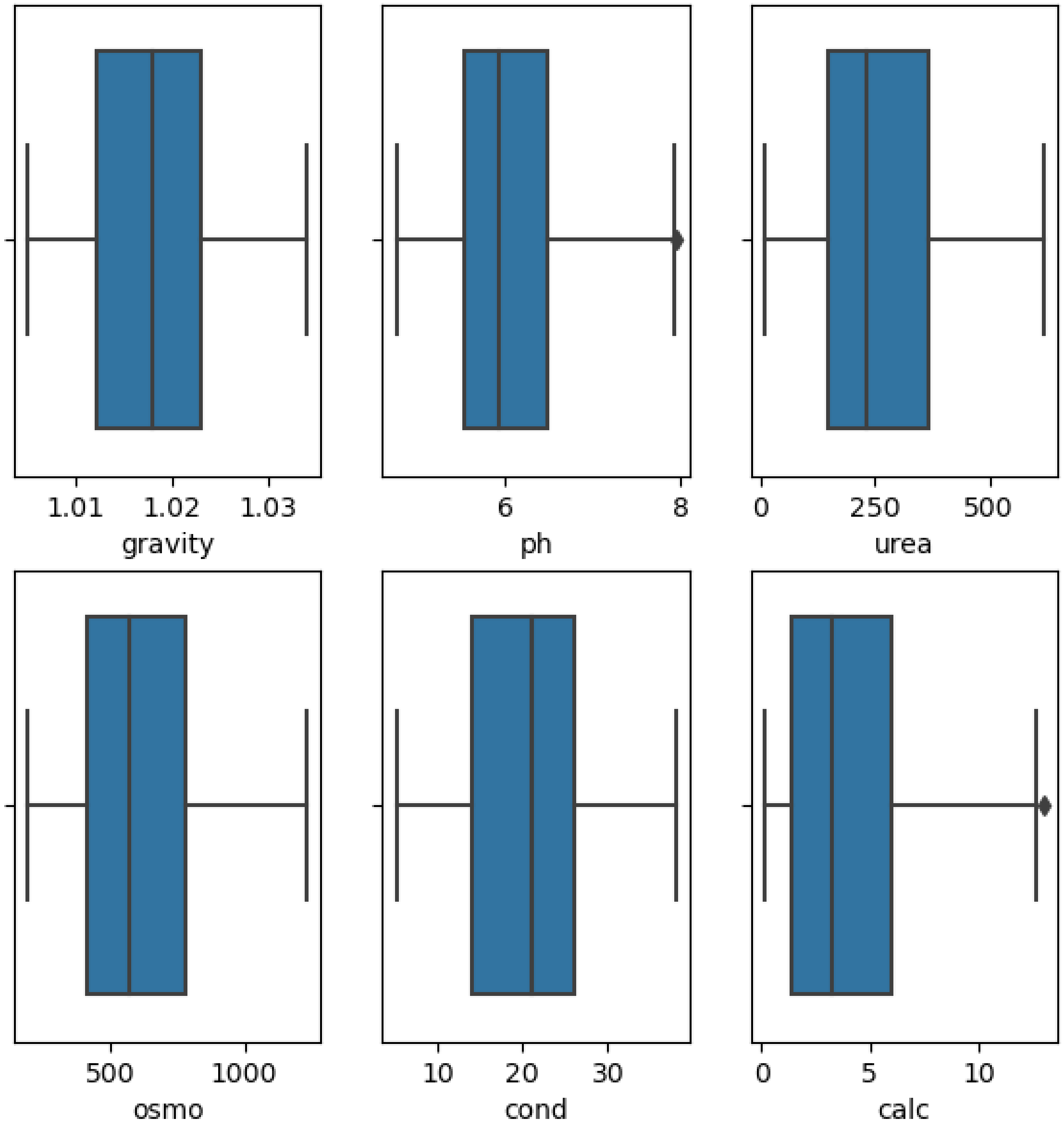
In [14]:

```
features = ['gravity','ph','urea','osmo']
plt.subplots(figsize=(5,5))

for i, col in enumerate(features):
  plt.subplot(2,2,i+1)
  sns.distplot(data[col])
plt.show()
```



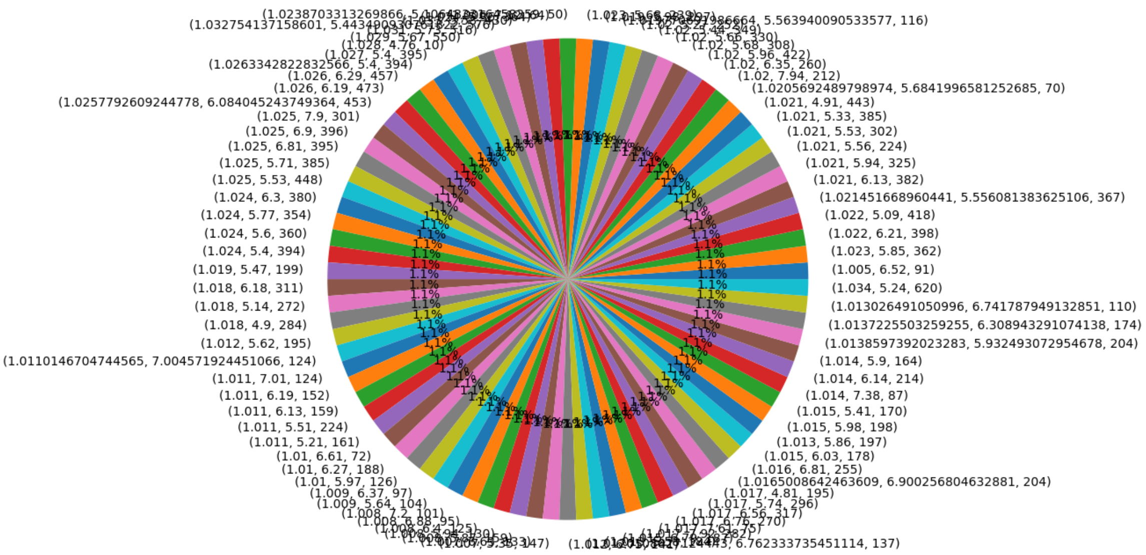
In [15]:

```
features = ['gravity','ph','urea','osmo','cond','calc']

plt.subplots(figsize=(7,7))

for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sns.boxplot(data[col])
plt.show()
```



In [16]:

```
features = [['gravity','ph','urea']]

plt.subplots(figsize=(30,30))

for i, col in enumerate(features):

    plt.subplot(1,3,i+1)
    x=data[col].value_counts()
    plt.pie(x.values,labels=x.index,autopct='%1.1f%%')
plt.show()
```

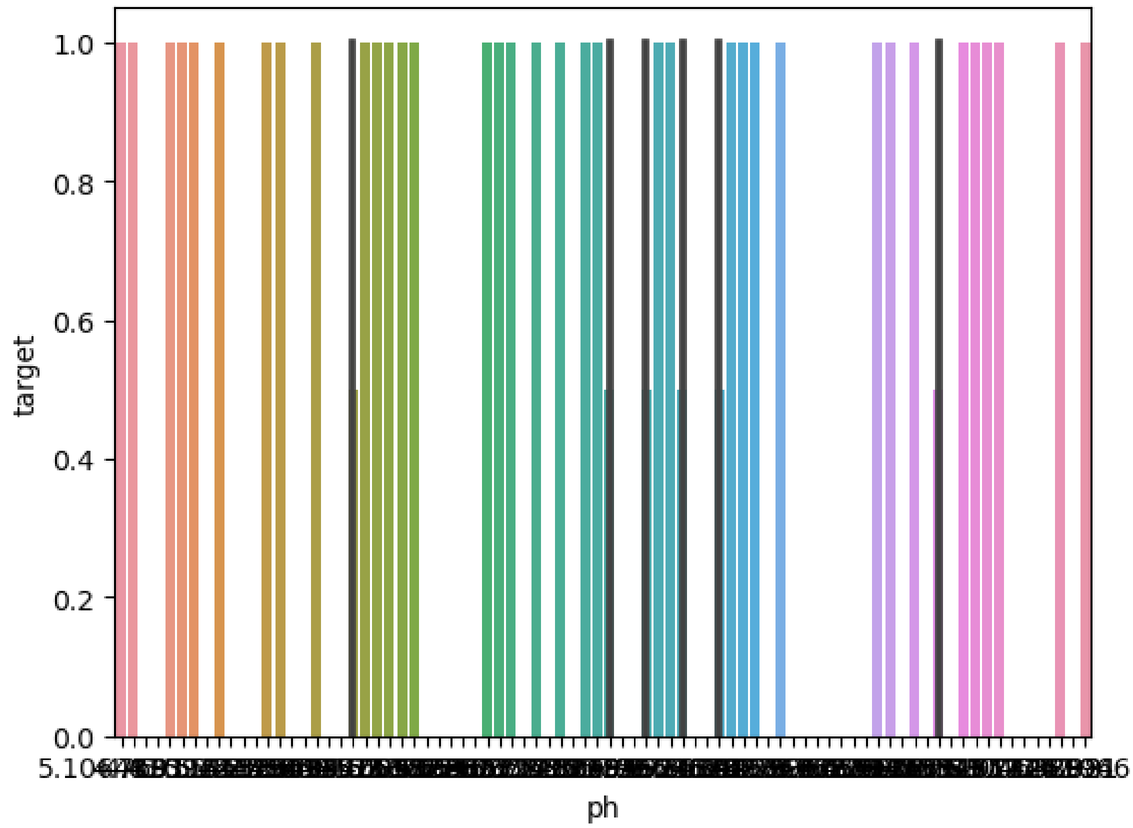


In [17]:

```
sns.barplot(x='ph',y='target',data=data)
```

Out[17]:

```
<AxesSubplot:xlabel='ph', ylabel='target'>
```



TRAINING AND TESTING DATA

In [18]:

```
x=data.iloc[:,0:7].values
x.shape
```

Out[18]:

```
(90, 7)
```

In [19]:

```
y=data.iloc[:,7].values
print(y.shape)
```

```
(90,)
```

In [20]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

LOGISTIC REGRESSION

In [21]:

```
reg = LogisticRegression()
reg.fit(x_train,y_train)
```

Out[21]:

```
LogisticRegression()
```

In [22]:

```
y_pred=reg.predict(x_test)
y_pred
```

Out[22]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0], dtype=int64)
```

In [23]:

```
y_test
```

Out[23]:

```
array([1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0], dtype=int64)
```

In [24]:

```
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print("Training Score: ",reg.score(x_train,y_train)*100)
```

```
              precision    recall  f1-score   support

           0       1.00      0.88      0.93         8
           1       0.91      1.00      0.95        10

    accuracy                           0.94        18
   macro avg       0.95      0.94      0.94        18
weighted avg       0.95      0.94      0.94        18

[[ 7  1]
 [ 0 10]]
Training Score:  100.0
```

In [25]:

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
data
```

Out[25]:

	Actual	Predicted
0	1	1
1	1	1
2	0	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	1	1
9	0	0
10	0	0
11	0	0
12	1	1
13	0	0
14	0	0
15	1	1
16	0	0
17	0	0

In [26]:

```
print(accuracy_score(y_test,y_pred)*100)
```

```
94.44444444444444
```

In [27]:

```
from sklearn.model_selection import GridSearchCV
param = {
         'penalty':['l1','l2'],
         'C':[0.001, 0.01, 0.1, 1, 10, 20,100, 1000]
}
lr= LogisticRegression(penalty='l1')
cv=GridSearchCV(reg,param,cv=5,n_jobs=-1)
cv.fit(x_train,y_train)
cv.predict(x_test)
```

Out[27]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0], dtype=int64)
```

In [28]:

```
print("Best CV score", cv.best_score_*100)
```

```
Best CV score 100.0
```

In [ ]:

```

```

RANDOM FOREST

In [29]:

```
ss=StandardScaler()
x_train=ss.fit_transform(x_train)
x_test=ss.transform(x_test)
```

In [30]:

```
clfr=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)

clfr.fit(x_train,y_train)
```

Out[30]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=
0)
```

In [31]:

```
clfr1=RandomForestClassifier(n_estimators=10,criterion='gini',random_state=0)

clfr1.fit(x_train,y_train)
```

Out[31]:

```
RandomForestClassifier(n_estimators=10, random_state=0)
```

In [32]:

```
ypre1=clfr.predict(x_test)# entropy ypre calculation

ypre2=clfr1.predict(x_test)# gini ypre calculation
```

In [33]:

```
print('entropy Accuracy Score:')
accuracy_score(y_test,ypre1)*100
```

```
entropy Accuracy Score:
```

Out[33]:

```
100.0
```

In [34]:

```
print('gini Accuracy Score:')
accuracy_score(y_test,ypre2)*100
```

```
gini Accuracy Score:
```

Out[34]:

```
94.44444444444444
```

In [35]:

```
print('entropy - confusion matrix\n--------------------\n')
print(confusion_matrix(y_test,ypre1))
print('gini - confusion matrix\n--------------------\n')
print(confusion_matrix(y_test,ypre2))
```

```
entropy - confusion matrix
--------------------

[[ 8  0]
 [ 0 10]]
gini - confusion matrix
--------------------

[[ 7  1]
 [ 0 10]]
```

In [36]:

```
print('entropy result\n--------------')
print(classification_report(y_test,ypre1))
print('gini index result\n----------------------')
print(classification_report(y_test,ypre2))
```

```
entropy result
--------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      1.00      1.00        10

    accuracy                           1.00        18
   macro avg       1.00      1.00      1.00        18
weighted avg       1.00      1.00      1.00        18

gini index result
----------------------
              precision    recall  f1-score   support

           0       1.00      0.88      0.93         8
           1       0.91      1.00      0.95        10

    accuracy                           0.94        18
   macro avg       0.95      0.94      0.94        18
weighted avg       0.95      0.94      0.94        18
```

DecisionTreeClassifier

In [37]:

```
dtree = DecisionTreeClassifier(max_depth=6, random_state=1)

dtree.fit(x_train,y_train)
```

Out[37]:

```
DecisionTreeClassifier(max_depth=6, random_state=1)
```

In [38]:

```
y_pred3=dtree.predict(x_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,mean_sq
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print("Training Score: ",dtree.score(x_train,y_train)*100)
```

```
              precision    recall  f1-score   support

           0       1.00      0.88      0.93         8
           1       0.91      1.00      0.95        10

    accuracy                           0.94        18
   macro avg       0.95      0.94      0.94        18
weighted avg       0.95      0.94      0.94        18

[[ 7  1]
 [ 0 10]]
Training Score:  100.0
```

In [39]:

```
print(accuracy_score(y_test,y_pred3)*100)
```

```
100.0
```

Comparing Actual and Predicted Output

In [40]:

```
data = pd.DataFrame({'Actual': y_test, 'Logreg': y_pred,'Rf(Entropy)': ypre1, 'Rf(Gini)':
data
```

Out[40]:

	Actual	Logreg	Rf(Entropy)	Rf(Gini)	DT
0	1	1	1	1	1
1	1	1	1	1	1
2	0	1	0	0	0
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1
6	1	1	1	1	1
7	1	1	1	1	1
8	1	1	1	1	1
9	0	0	0	1	0
10	0	0	0	0	0
11	0	0	0	0	0
12	1	1	1	1	1
13	0	0	0	0	0
14	0	0	0	0	0
15	1	1	1	1	1
16	0	0	0	0	0
17	0	0	0	0	0

In [41]:

```
data=pd.DataFrame({'Models':['Logreg','Rf(Entropy)','Rf(Gini)','DT'],
                'Accuracy':[accuracy_score(y_test,y_pred)*100,accuracy_score(y_test,ypre1
                            accuracy_score(y_test,ypre2)*100,accuracy_score(y_test,y_pred
data
```

Out[41]:

	Models	Accuracy
0	Logreg	94.444444
1	Rf(Entropy)	100.000000
2	Rf(Gini)	94.444444
3	DT	100.000000

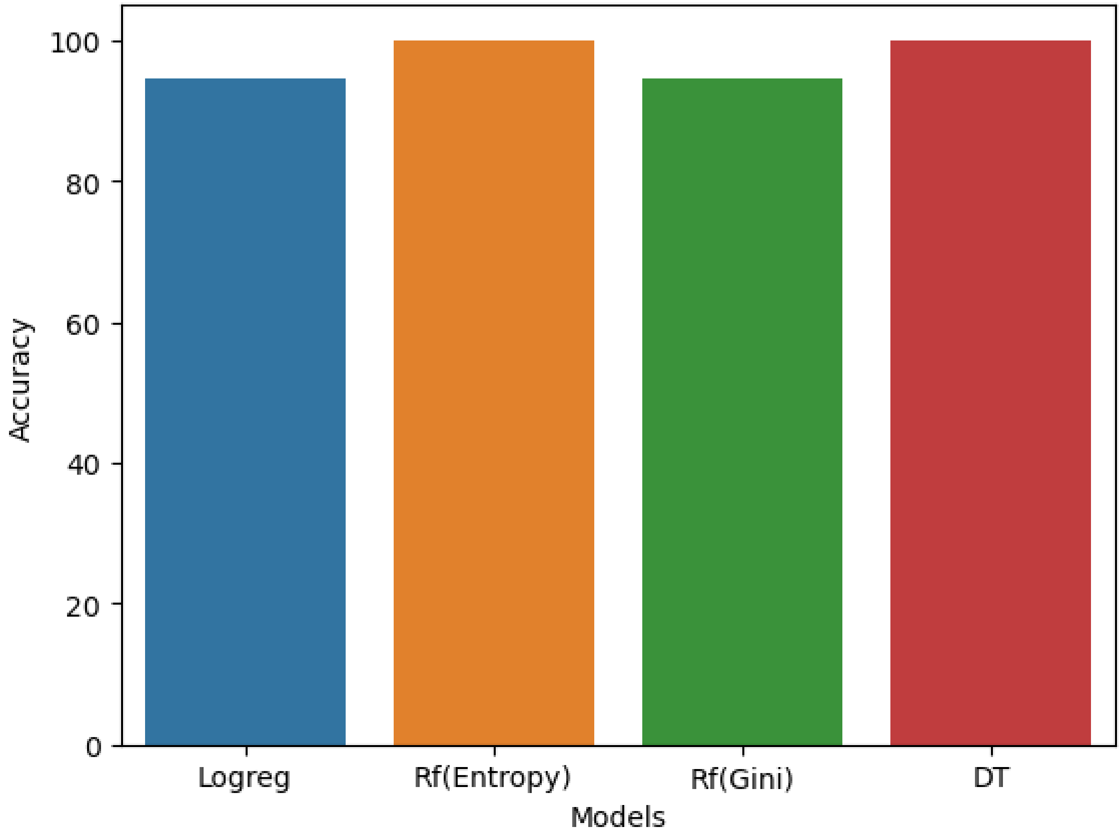
Visualise the Accuracy score

In [42]:

```
sns.barplot(data['Models'],data['Accuracy'])
```

Out[42]:

```
<AxesSubplot:xlabel='Models', ylabel='Accuracy'>
```



Type *Markdown* and LaTeX: α^2

In [ ]:

In [ ]:

Predict Kidney stones for New Patient

In [43]:

```
a={'index':0,'gravity':1.432,'ph':3,'osmo':187,'cond':7.5,'urea':224,'calc':1.39}
```

In [44]:

```
df=pd.DataFrame(a,index=[0])
df
```

Out[44]:

	index	gravity	ph	osmo	cond	urea	calc
0	0	1.432	3	187	7.5	224	1.39

In [45]:

```
#predict using logreg model

new=reg.predict(df)
new
```

Out[45]:

```
array([0], dtype=int64)
```

save model using joblib

In [46]:

```
reg = LogisticRegression()
reg.fit(x,y)
```

Out[46]:

```
LogisticRegression()
```

In [47]:

```
import joblib
```

In [48]:

```
joblib.dump(reg,'model_joblib_reg')
```

Out[48]:

```
['model_joblib_reg']
```

In [49]:

```
c=joblib.load('model_joblib_reg')
```

In [50]:

```
c.predict(df)
```

Out[50]:

```
array([0], dtype=int64)
```

GUI

In [51]:

```
from tkinter import *
```

In [52]:

```
import joblib
```

In [53]:

```
def show_entry():
    p1=float(e1.get())
    p2=float(e2.get())
    p3=float(e3.get())
    p4=float(e4.get())
    p5=float(e5.get())
    p6=float(e6.get())
    p7=float(e7.get())

    c=joblib.load('model_joblib_reg')
    result=c.predict([[p1,p2,p3,p4,p5,p6,p7]])

    Label(master,text='Kidney Stone Prediction').place(x=20,y=365)
    Label(master,text=result).place(x=20,y=405)


master=Tk()
master.geometry('1600x1000')
master.title('Kidney Stone Prediction')

l= Label(master,text='Kidney Stone Prediction', bg = 'lavender', fg = 'red', font =('cali
l.place(x = 20, y = 45)

l2=Label(master,text='enter index value', bg = 'lavender', fg = 'purple', font =('calibri
l2.place(x = 20, y = 85)

l3=Label(master,text='enter gravity', bg = 'lavender', fg = 'purple', font =('calibri', 1
l3.place(x = 20, y = 125)

l4=Label(master,text='enter ph', bg = 'lavender', fg = 'purple', font =('calibri', 15, 'b
l4.place(x = 20, y = 165)

l5=Label(master,text='enter osmo', bg = 'lavender', fg = 'purple', font =('calibri', 15,
l5.place(x = 20, y = 205)

l6=Label(master,text='enter cond', bg = 'lavender', fg = 'purple', font =('calibri', 15,
l6.place(x = 20, y = 245)

l7=Label(master,text='enter urea', bg = 'lavender', fg = 'purple', font =('calibri', 15,
l7.place(x = 20, y = 285)

l8=Label(master,text='enter calc', bg = 'lavender', fg = 'purple', font =('calibri', 15,
l8.place(x = 20, y = 325)

e1 = Entry(master, width = 15, fg = 'black', font = ('calibri', 15, 'bold'))
e1.place(x = 220, y = 85)

e2 = Entry(master, width = 15, fg = 'black', font = ('calibri', 15, 'bold'))
e2.place(x = 220, y = 125)

e3 = Entry(master, width = 15, fg = 'black', font = ('calibri', 15, 'bold'))
e3.place(x = 220, y = 165)

e4 = Entry(master, width = 15, fg = 'black', font = ('calibri', 15, 'bold'))
e4.place(x = 220, y = 205)

e5 = Entry(master, width = 15, fg = 'black', font = ('calibri', 15, 'bold'))
e5.place(x = 220, y = 245)
```

```
e6 = Entry(master, width = 15, fg = 'black', font = ('calibri', 15, 'bold'))
e6.place(x = 220, y = 285)

e7 = Entry(master, width = 15, fg = 'black', font = ('calibri', 15, 'bold'))
e7.place(x = 220, y = 325)

b = Button(master, text = 'Predict', bg = 'orange', fg = 'black', width = 5, padx = 10, c
b.pack()



master.mainloop()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: