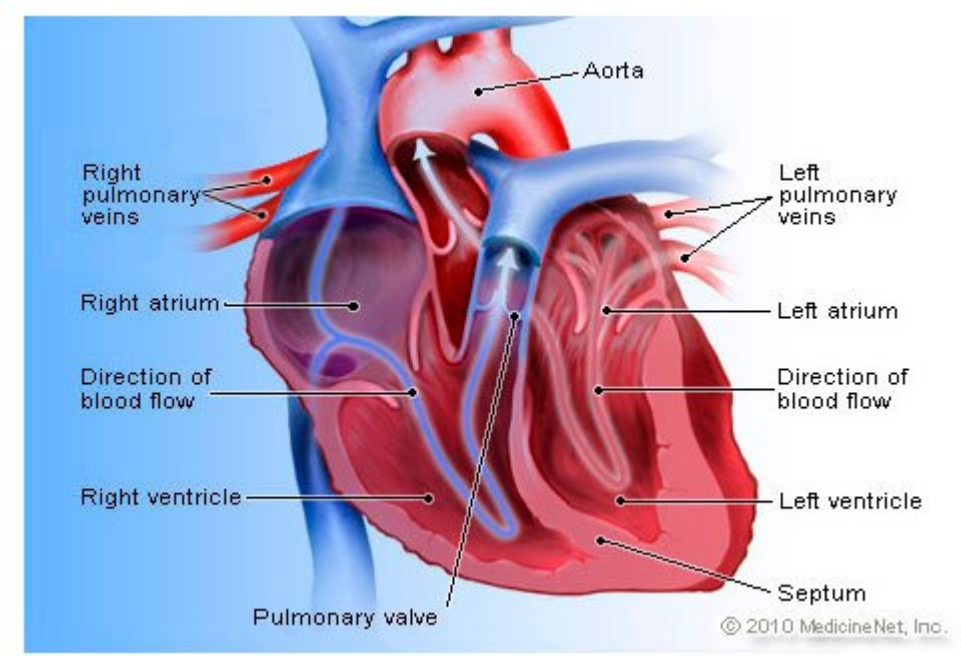
Project Report by : Ashwin Gorakhnath Dubey

Topic: Heart Failure prediction



Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide.

Heart failure is a common event caused by CVDs and this dataset contains 12 features that can be used to predict mortality by heart failure.Most cardiovascular diseases can be prevented by addressing behavioural risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol using population-wide strategies. People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

Objective: To create a classification filter (Using Logistics Regression & KNN Classification Algorithm) to predict Heart Failure. Also Comparing the performance of the filters

The overview of Dataset

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	age	anaemia	creatinine	diabetes	ejection_f	high_bloo	platelets	serum_cre	serum_so	sex	smoking	time	DEATH_EVENT		
2	75	0	582	0	20	1	265000	1.9	130	1	0	4	1		
3	55	0	7861	0	38	0	263358	1.1	136	1	0	6	1		
4	65	0	146	0	20	0	162000	1.3	129	1	1	7	1		
5	50	1	111	0	20	0	210000	1.9	137	1	0	7	1		
6	65	1	160	1	20	0	327000	2.7	116	0	0	8	1		
7	90	1	47	0	40	1	204000	2.1	132	1	1	8	1		
8	75	1	246	0	15	0	127000	1.2	137	1	0	10	1		
9	60	1	315	1	60	0	454000	1.1	131	1	1	10	1		
10	65	0	157	0	65	0	263358	1.5	138	0	0	10	1		
11	80	1	123	0	35	1	388000	9.4	133	1	1	10	1		
12	75	1	81	0	38	1	368000	4	131	1	1	10	1		
13	62	0	231	0	25	1	253000	0.9	140	1	1	10	1		
14	45	1	981	0	30	0	136000	1.1	137	1	0	11	1		
15	50	1	168	0	38	1	276000	1.1	137	1	0	11	1		
16	49	1	80	0	30	1	427000	1	138	0	0	12	0		
17	82	1	379	0	50	0	47000	1.3	136	1	0	13	1		
18	87	1	149	0	38	0	262000	0.9	140	1	0	14	1		
19	45	0	582	0	14	0	166000	0.8	127	1	0	14	1		
20	70	1	125	0	25	1	237000	1	140	0	0	15	1		
21	48	1	582	1	55	0	87000	1.9	121	0	0	15	1		
22	65	1	52	0	25	1	276000	1.3	137	0	0	16	0		
23	65	1	128	1	30	1	297000	1.6	136	0	0	20	1		
24	68	1	220	0	35	1	289000	0.9	140	1	1	20	1		
25	53	0	63	1	60	0	368000	0.8	135	1	0	22	0		

Loading the data set

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

In [3]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [4]:
```
df=pd.read_csv('heart_failure.csv')
df.head()
```

Out[4]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium
0	75.0	0	582	0	20	1	265000.00	1.9	130
1	55.0	0	7861	0	38	0	263358.03	1.1	136
2	65.0	0	146	0	20	0	162000.00	1.3	129
3	50.0	1	111	0	20	0	210000.00	1.9	137
4	65.0	1	160	1	20	0	327000.00	2.7	116

Geting data ready

In [5]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    int64  
 3   diabetes                  299 non-null    int64  
 4   ejection_fraction         299 non-null    int64  
 5   high_blood_pressure       299 non-null    int64  
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64  
 9   sex                       299 non-null    int64  
 10  smoking                   299 non-null    int64  
 11  time                      299 non-null    int64  
 12  DEATH_EVENT               299 non-null    int64  
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

In [6]:
```
1 df.isnull().sum()
```

Out[6]:
```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

In [7]:
```
1 df.describe()
```

Out[7]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creati
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.0
mean	60.833893	0.431438	581.839465	0.418060	38.083612	0.351171	263358.029264	1.3
std	11.894809	0.496107	970.287881	0.494067	11.834841	0.478136	97804.236869	1.0
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.000000	0.5
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500.000000	0.9
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000.000000	1.1
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500.000000	1.4
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.000000	9.4

In [8]:
```
1 df['DEATH_EVENT'].value_counts()
```

Out[8]:
```
0    203
1     96
Name: DEATH_EVENT, dtype: int64
```

In [9]:
```
1 heart=df.rename(columns = {'DEATH_EVENT':'Target'})
```

In [10]:
```
1 heart.head()
```

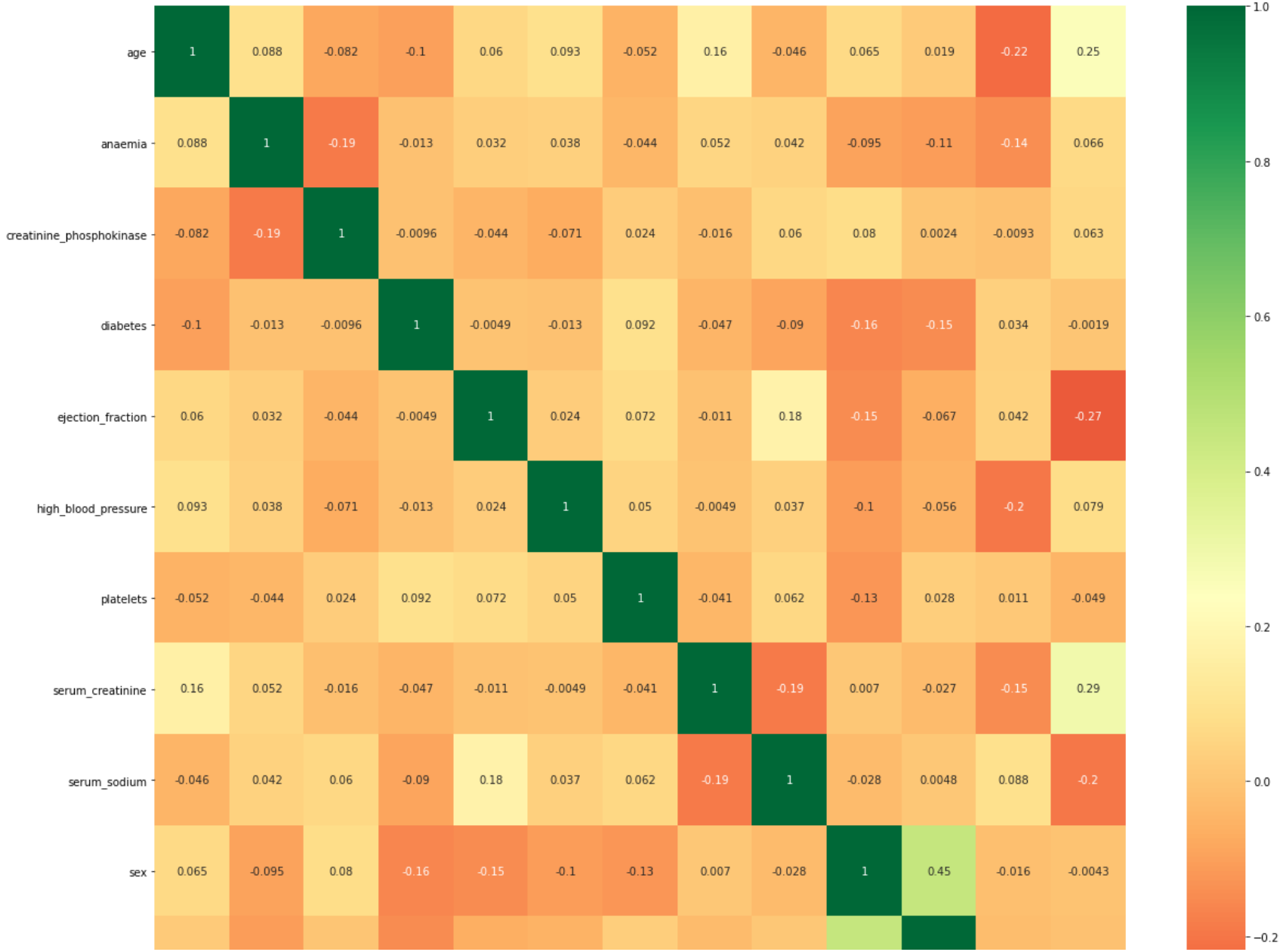
Out[10]:

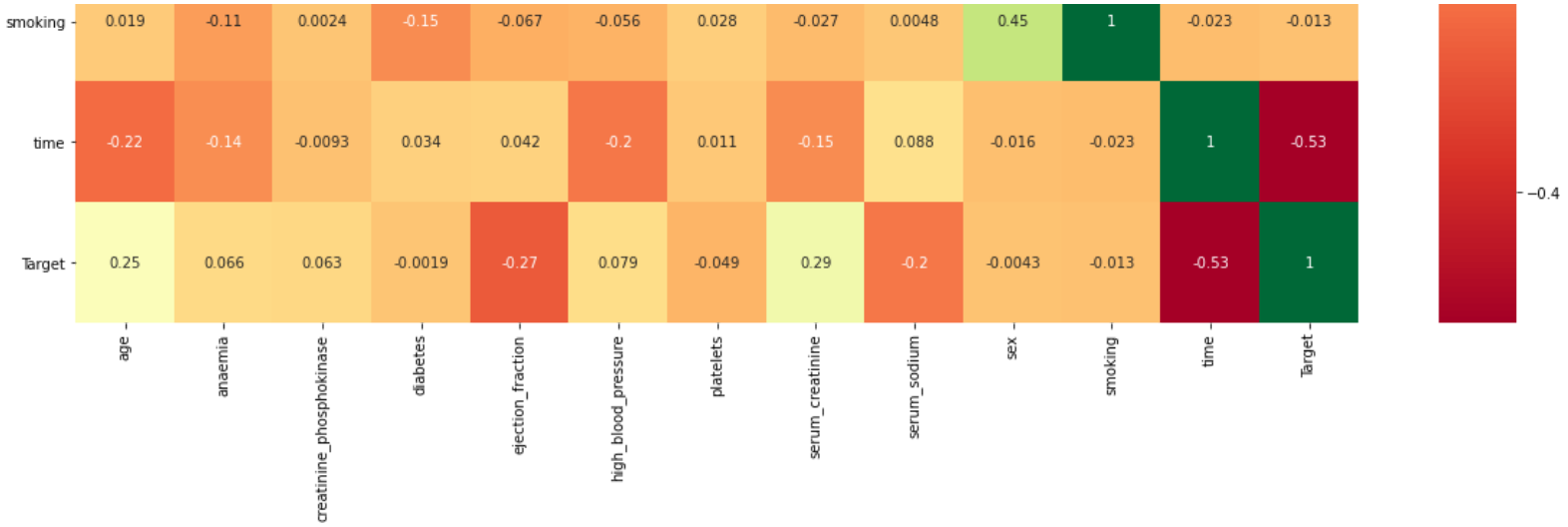
	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium
0	75.0	0	582	0	20	1	265000.00	1.9	130
1	55.0	0	7861	0	38	0	263358.03	1.1	136
2	65.0	0	146	0	20	0	162000.00	1.3	129
3	50.0	1	111	0	20	0	210000.00	1.9	137
4	65.0	1	160	1	20	0	327000.00	2.7	116

In [11]:
```
1 heart.shape
```

Out[11]:
```
(299, 13)
```

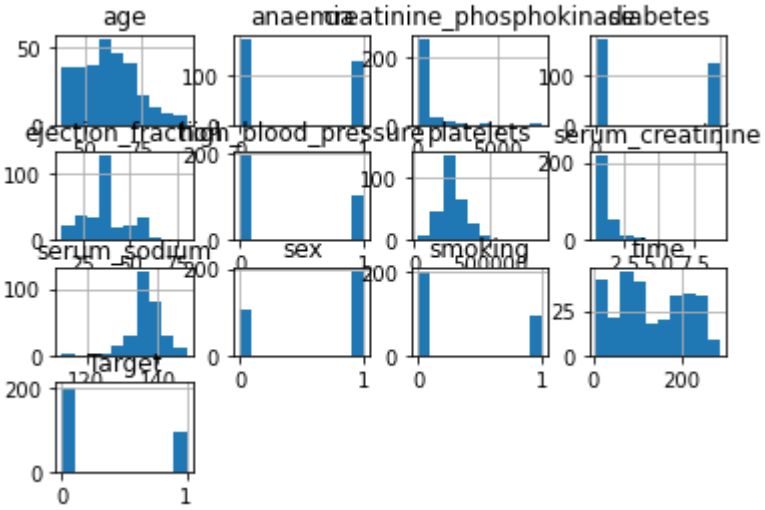
In [12]:
```
import seaborn as sns
corrmat=heart.corr()
top_corr_features=corrmat.index
plt.figure(figsize=(20,20))
g=sns.heatmap(heart[top_corr_features].corr(),annot=True,cmap='RdYlGn')
```




smoking 0.019 -0.11 0.0024 -0.15 -0.067 -0.056 0.028 -0.027 0.0048 0.45 1 -0.023 -0.013
time -0.22 -0.14 -0.0093 0.034 0.042 -0.2 0.011 -0.15 0.088 -0.016 -0.023 1 -0.53
Target 0.25 0.066 0.063 -0.0019 -0.27 0.079 -0.049 0.29 -0.2 -0.0043 -0.013 -0.53 1
age
anaemia
creatinine_phosphokinase
diabetes
ejection_fraction
high_blood_pressure
platelets
serum_creatinine
serum_sodium
sex
smoking
time
Target
−0.4

In [13]:
```
1 heart.hist()
```

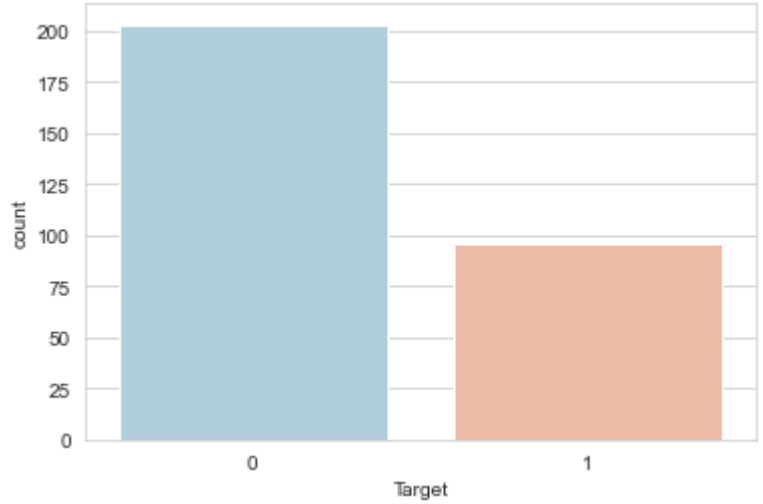
Out[13]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'anaemia'}>,
        <AxesSubplot:title={'center':'creatinine_phosphokinase'}>,
        <AxesSubplot:title={'center':'diabetes'}>],
       [<AxesSubplot:title={'center':'ejection_fraction'}>,
        <AxesSubplot:title={'center':'high_blood_pressure'}>,
        <AxesSubplot:title={'center':'platelets'}>,
        <AxesSubplot:title={'center':'serum_creatinine'}>],
       [<AxesSubplot:title={'center':'serum_sodium'}>,
        <AxesSubplot:title={'center':'sex'}>,
        <AxesSubplot:title={'center':'smoking'}>,
        <AxesSubplot:title={'center':'time'}>],
       [<AxesSubplot:title={'center':'Target'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



Checking out the data is balanced or not

In [14]:
```
sns.set_style('whitegrid')
sns.countplot(x='Target',data=heart,palette='RdBu_r')
```

Out[14]: <AxesSubplot:xlabel='Target', ylabel='count'>



Data Preprocessing

In [15]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
standardscaler = StandardScaler()

```

In [16]:
```
x=heart[['age','ejection_fraction','serum_creatinine','serum_sodium','time']]
y=heart['Target']
```

(1) KNN Alogorithm

In [17]:
```
x1=x
y1=y
```

In [18]:
```
x1_train, x1_test, y1_train, y1_test = train_test_split(x1,y1,test_size=0.3,random_state=16)
x1_train.shape, x1_test.shape, y1_train.shape, y1_test.shape

```

Out[18]:
```
((209, 5), (90, 5), (209,), (90,))
```

In [19]:
```
from sklearn.model_selection import cross_val_score
knn_scores=[]
for k in range(1,35):
    knn_classifier=KNeighborsClassifier(n_neighbors = k)
    score=cross_val_score(knn_classifier,x1_train,y1_train,cv=10)
    knn_scores.append(score.mean())
```

In [20]:
```
plt.plot([k for k in range(1,35)],knn_scores,color='blue')
for i in range (1,35):
    plt.text(i,knn_scores[i-1],(i,knn_scores[i-1]))
plt.xticks([i for i in range(1,35)])
plt.xlabel('Number of Neighbors(k)')
plt.ylabel('Score')
plt.title('K Neighbors Classifier score for different k values')
```

Out[20]:
```
Text(0.5, 1.0, 'K Neighbors Classifier score for different k values')
```

In [21]:
```
knn_classifier=KNeighborsClassifier()
cvs=cross_val_score(knn_classifier,x1_train,y1_train,cv=10)
```

In [22]:
```
knn_classifier.fit(x,y)
```

Out[22]:
```
KNeighborsClassifier()
```

In [23]:
```
knn_classifier.score(x1_train, y1_train)*100
```

Out[23]:
```
88.51674641148325
```

In [24]:
```
KNN_accuracy=knn_classifier.score(x1_test,y1_test)*100
```

In [25]:
```
KNN_accuracy
```

Out[25]:
```
84.44444444444444
```

(2) Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
```

In [27]:
```
from sklearn import metrics as sm
```

In [28]:
```
x2=x
y2=y
```

In [71]:
```
x2_train, x2_test, y2_train, y2_test = train_test_split(x2,y2,test_size=0.3,random_state=43)
x2_train.shape, x2_test.shape, y2_train.shape, y2_test.shape
```

Out[71]:
```
((209, 5), (90, 5), (209,), (90,))
```

In [72]:
```
logR=LogisticRegression(max_iter = 500).fit(x2_train,y2_train)
logR
```

Out[72]:
```
LogisticRegression(max_iter=500)
```

In [73]:
```
1 logR.predict_proba(x2_test)
```

Out[73]:
```
array([[0.96800692, 0.03199308],
       [0.30620862, 0.69379138],
       [0.75685739, 0.24314261],
       [0.97980448, 0.02019552],
       [0.19796348, 0.80203652],
       [0.83032735, 0.16967265],
       [0.98793324, 0.01206676],
       [0.71152816, 0.28847184],
       [0.90970966, 0.09029034],
       [0.29468815, 0.70531185],
       [0.75600512, 0.24399488],
       [0.0748705 , 0.9251295 ],
       [0.69015161, 0.30984839],
       [0.28339798, 0.71660202],
       [0.65523335, 0.34476665],
       [0.97493899, 0.02506101],
       [0.96913566, 0.03086434],
       [0.38167877, 0.61832123],
       [0.29779104, 0.70220896],
       [0.35333808, 0.64666192],
       [0.15433231, 0.84566769],
       [0.07783694, 0.92216306],
       [0.8810473 , 0.1189527 ],
       [0.85631754, 0.14368246],
       [0.09678959, 0.90321041],
       [0.46284501, 0.53715499],
       [0.09732507, 0.90267493],
       [0.20040619, 0.79959381],
       [0.97926343, 0.02073657],
       [0.21558425, 0.78441575],
       [0.94002665, 0.05997335],
       [0.93331776, 0.06668224],
       [0.98638097, 0.01361903],
       [0.28673486, 0.71326514],
       [0.76491102, 0.23508898],
       [0.99372534, 0.00627466],
       [0.64460685, 0.35539315],
       [0.93963088, 0.06036912],
       [0.749069  , 0.250931  ],
       [0.98183939, 0.01816061],
       [0.85717181, 0.14282819],
```

```
       [0.9671865 , 0.0328135 ],
       [0.9768663 , 0.0231337 ],
       [0.08289486, 0.91710514],
       [0.08615071, 0.91384929],
       [0.38055931, 0.61944069],
       [0.6606843 , 0.3393157 ],
       [0.36942019, 0.63057981],
       [0.77142021, 0.22857979],
       [0.87587902, 0.12412098],
       [0.77882314, 0.22117686],
       [0.14829819, 0.85170181],
       [0.14940221, 0.85059779],
       [0.974051  , 0.025949  ],
       [0.9922662 , 0.0077338 ],
       [0.94875029, 0.05124971],
       [0.68192189, 0.31807811],
       [0.99249545, 0.00750455],
       [0.93401897, 0.06598103],
       [0.32313484, 0.67686516],
       [0.0756424 , 0.9243576 ],
       [0.75285373, 0.24714627],
       [0.68590482, 0.31409518],
       [0.98352103, 0.01647897],
       [0.93315831, 0.06684169],
       [0.99585845, 0.00414155],
       [0.48840864, 0.51159136],
       [0.04030061, 0.95969939],
       [0.84013065, 0.15986935],
       [0.88781412, 0.11218588],
       [0.66054638, 0.33945362],
       [0.91859876, 0.08140124],
       [0.97895015, 0.02104985],
       [0.98017118, 0.01982882],
       [0.51150969, 0.48849031],
       [0.90685868, 0.09314132],
       [0.01738039, 0.98261961],
       [0.04684204, 0.95315796],
       [0.98611458, 0.01388542],
       [0.95895358, 0.04104642],
       [0.81163589, 0.18836411],
       [0.53745886, 0.46254114],
       [0.96371278, 0.03628722],
       [0.92669061, 0.07330939],
```

```
       [0.67798753, 0.32201247],
       [0.5450321 , 0.4549679 ],
       [0.97377237, 0.02622763],
       [0.85772317, 0.14227683],
       [0.8745253 , 0.1254747 ],
       [0.89792208, 0.10207792]])
```

In [74]:
```
1 logR.predict_proba(x2_test).shape
```

Out[74]: (90, 2)

In [75]:
```
1 print(logR.intercept_)
```

```
[8.44713022]
```

In [76]:
```
1 print(logR.coef_)
```

```
[[ 0.03251449 -0.07166284  0.47762846 -0.05102628 -0.02183472]]
```

In [77]:
```
1 pred=logR.predict(x2_test)
2 pred
```

Out[77]:
```
array([0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

In [78]:
```
df1 = pd.DataFrame({'actual': y2_test, 'predictions': pred})
df1.head(10)
```

Out[78]:

	actual	predictions
258	0	0
68	1	1
115	0	0
251	0	0
74	1	1
184	1	0
285	0	0
103	0	0
197	0	0
72	1	1

In [79]:
```
ct = pd.crosstab(df1['actual'], df1['predictions'])
ct
```

Out[79]:

predictions	0	1
actual		
0	51	3
1	11	25

In [80]:
```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,roc_auc_score
```

In [81]:
```
from sklearn.metrics import confusion_matrix
```

In [82]:
```
1 confusion_matrix(y2_test,pred)
```

Out[82]:
```
array([[51,  3],
       [11, 25]], dtype=int64)
```

In [83]:
```
1 LogR_accuracy=(sm.accuracy_score(y2_test,pred))*100
2 LogR_accuracy
```

Out[83]:
```
84.44444444444444
```

In [84]:
```
1 table=pd.DataFrame({'KNN accuracy':[KNN_accuracy], 'Logistic Regression':[LogR_accuracy]})
```

In [85]:
```
1 table
```

Out[85]:

	KNN accuracy	Logistic Regression
0	84.444444	84.444444

Submitted by : Ashwin Gorakhnath Dubey

Submitted to: EICT-IIT (Roorkee)