diabetes-prediction-model-training

November 16, 2023

[1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
df = pd.read_csv("diabetes.csv")
df.head()
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
```

```
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[4]:
```
df.isnull().sum()
```

[4]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[5]:
```
df.duplicated().sum()
```

[5]:
```
0
```

[6]:
```
df.nunique()
```

[6]:
```
Pregnancies                 17
Glucose                    136
BloodPressure               47
SkinThickness               51
Insulin                    186
BMI                        248
DiabetesPedigreeFunction   517
Age                         52
Outcome                      2
dtype: int64
```

[7]:
```
df.shape
```

[7]:
```
(768, 9)
```

[8]:
```
df.describe()
```

[8]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
```

```
25%       1.000000   99.000000       62.000000        0.000000    0.000000
50%       3.000000  117.000000       72.000000       23.000000   30.500000
75%       6.000000  140.250000       80.000000       32.000000  127.250000
max      17.000000  199.000000      122.000000       99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[9]:
```
#If we observe above in the min the minumum values of some parameters are 0.
 ↪0000 like for gulcose,bloodpressure etc...
#so, we replace those values with mean of its column..
df['Glucose'] = df['Glucose'].replace(0,df['Glucose'].mean())
df['BloodPressure'] = df['BloodPressure'].replace(0,df['BloodPressure'].mean())
df['SkinThickness'] = df['SkinThickness'].replace(0,df['SkinThickness'].mean())
df['Insulin'] = df['Insulin'].replace(0,df['Insulin'].mean())
df['BMI'] = df['BMI'].replace(0,df['BMI'].mean())
```

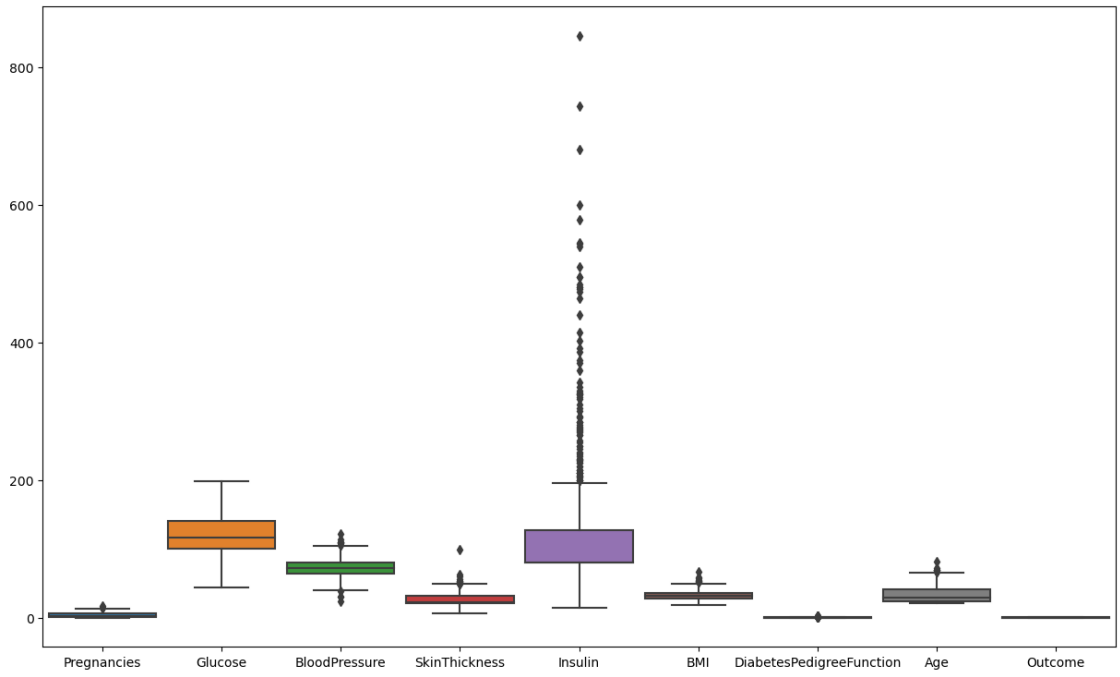
[10]:
```
df.describe()
```

[10]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  121.681605      72.254807      26.606479  118.660163
std       3.369578   30.436016      12.115932       9.631241   93.080358
min       0.000000   44.000000      24.000000       7.000000   14.000000
25%       1.000000   99.750000      64.000000      20.536458   79.799479
50%       3.000000  117.000000      72.000000      23.000000   79.799479
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    32.450805                  0.471876   33.240885    0.348958
std      6.875374                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.500000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[11]:
```
plt.figure(figsize=(15,9))
sns.boxplot(data = df,width = 0.90)
```

[11]: <AxesSubplot: >



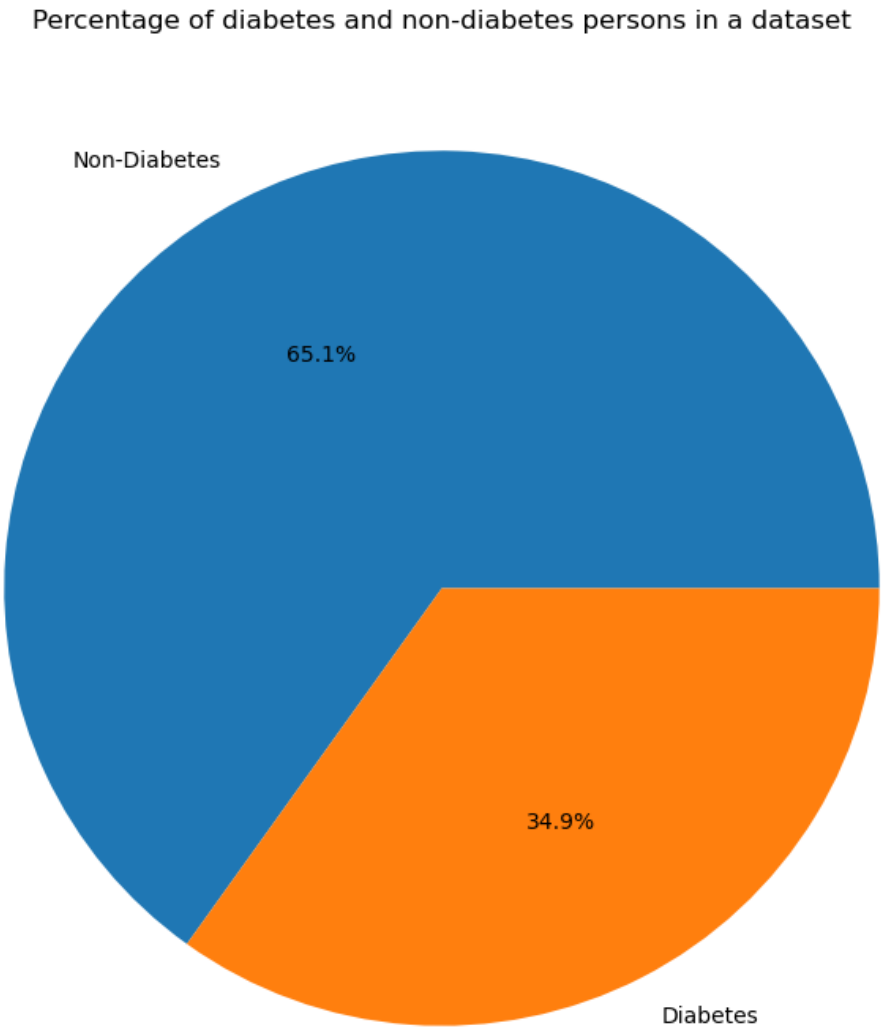
[12]:
```
sns.heatmap(df.corr(),annot = True)
```

[12]: <AxesSubplot: >

[13]:
```
df["Outcome"].value_counts()
```

[13]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

[14]:
```
plt.figure(figsize=(15,9))
plt.title("Percentage of diabetes and non-diabetes persons in a dataset")
labels = ["Non-Diabetes","Diabetes"]
x = [((500/768)*100),((268/768)*100)]
plt.pie(x,labels =labels, autopct='%1.1f%%')
plt.show()
```



[15]:
```
#Independent and dependent features
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[16]:
```
x
```

[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	85.0	66.0	29.000000	79.799479	26.6	
2	8	183.0	64.0	20.536458	79.799479	23.3	
3	1	89.0	66.0	23.000000	94.000000	28.1	
4	0	137.0	40.0	35.000000	168.000000	43.1	

```
..            …      …              …              …          …   …
763           10    101.0           76.0       48.000000  180.000000  32.9
764            2    122.0           70.0       27.000000   79.799479  36.8
765            5    121.0           72.0       23.000000  112.000000  26.2
766            1    126.0           60.0       20.536458   79.799479  30.1
767            1     93.0           70.0       31.000000   79.799479  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        …   …
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

[17]:
```
y
```

[17]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

1 Train_test_split

[18]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
 ↪33,random_state=80)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(514, 8)
(254, 8)
(514,)
(254,)
```

2 Feature Scaling

[19]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

3 Logistic Regression

[20]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import␣
 ↪accuracy_score,classification_report,confusion_matrix
regressor = LogisticRegression()
regressor.fit(x_train_scaled,y_train)
```

[20]:
```
LogisticRegression()
```

[21]:
```
y_pred = regressor.predict(x_test_scaled)
print(accuracy_score(y_pred,y_test))
print(confusion_matrix(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
0.7677165354330708
[[146  38]
 [ 21  49]]
              precision    recall  f1-score   support

           0       0.87      0.79      0.83       184
           1       0.56      0.70      0.62        70

    accuracy                           0.77       254
   macro avg       0.72      0.75      0.73       254
weighted avg       0.79      0.77      0.77       254
```

[22]:
```
# Hyperparameter tuning
parameters= {
   'penalty' : ['l1', 'l2', 'elasticnet'],
    'C' : [1,10,20],
    'solver':['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag',␣
 ↪'saga']
```

```
    }
```

[23]:
```
from sklearn.model_selection import GridSearchCV
regressor = LogisticRegression()
regressor_cv =␣
 ↪GridSearchCV(regressor,param_grid=parameters,cv=5,scoring='accuracy')
regressor_cv.fit(x_train_scaled,y_train)
```

[23]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': [1, 10, 20],
                         'penalty': ['l1', 'l2', 'elasticnet'],
                         'solver': ['lbfgs', 'liblinear', 'newton-cg',
                                    'newton-cholesky', 'sag', 'saga']},
             scoring='accuracy')
```

[24]:
```
regressor_cv.best_params_
```

[24]:
```
{'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

[25]:
```
regressor1 = LogisticRegression(C = 1,penalty='l1',solver='liblinear')
regressor1.fit(x_train_scaled,y_train)
y_pred1 = regressor1.predict(x_test_scaled)
print(accuracy_score(y_pred1,y_test))
print(confusion_matrix(y_pred1,y_test))
print(classification_report(y_pred1,y_test))
```

```
0.7637795275590551
[[145  38]
 [ 22  49]]
              precision    recall  f1-score   support

           0       0.87      0.79      0.83       183
           1       0.56      0.69      0.62        71

    accuracy                           0.76       254
   macro avg       0.72      0.74      0.72       254
weighted avg       0.78      0.76      0.77       254
```

[26]:
```
from sklearn.model_selection import RandomizedSearchCV
regressor_cv1 =␣
 ↪RandomizedSearchCV(regressor,param_distributions=parameters,cv=10,scoring='accuracy',verbos
 ↪= 3)
regressor_cv1.fit(x_train_scaled,y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[CV 1/10] END …C=20, penalty=l2, solver=sag;, score=0.750 total time=   0.0s
```

```
[CV 2/10] END …C=20, penalty=l2, solver=sag;, score=0.750 total time=   0.0s
[CV 3/10] END …C=20, penalty=l2, solver=sag;, score=0.769 total time=   0.0s
[CV 4/10] END …C=20, penalty=l2, solver=sag;, score=0.808 total time=   0.0s
[CV 5/10] END …C=20, penalty=l2, solver=sag;, score=0.667 total time=   0.0s
[CV 6/10] END …C=20, penalty=l2, solver=sag;, score=0.745 total time=   0.0s
[CV 7/10] END …C=20, penalty=l2, solver=sag;, score=0.627 total time=   0.0s
[CV 8/10] END …C=20, penalty=l2, solver=sag;, score=0.882 total time=   0.0s
[CV 9/10] END …C=20, penalty=l2, solver=sag;, score=0.824 total time=   0.0s
[CV 10/10] END …C=20, penalty=l2, solver=sag;, score=0.902 total time=   0.0s
[CV 1/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 2/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 3/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 4/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 5/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 6/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 7/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 8/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 9/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 10/10] END C=20, penalty=elasticnet, solver=lbfgs;, score=nan total time=
0.0s
[CV 1/10] END …C=20, penalty=l1, solver=saga;, score=0.750 total time=   0.0s
[CV 2/10] END …C=20, penalty=l1, solver=saga;, score=0.750 total time=   0.0s
[CV 3/10] END …C=20, penalty=l1, solver=saga;, score=0.769 total time=   0.0s
[CV 4/10] END …C=20, penalty=l1, solver=saga;, score=0.788 total time=   0.0s
[CV 5/10] END …C=20, penalty=l1, solver=saga;, score=0.667 total time=   0.0s
[CV 6/10] END …C=20, penalty=l1, solver=saga;, score=0.745 total time=   0.0s
[CV 7/10] END …C=20, penalty=l1, solver=saga;, score=0.627 total time=   0.0s
[CV 8/10] END …C=20, penalty=l1, solver=saga;, score=0.882 total time=   0.0s
[CV 9/10] END …C=20, penalty=l1, solver=saga;, score=0.824 total time=   0.0s
[CV 10/10] END …C=20, penalty=l1, solver=saga;, score=0.902 total time=   0.0s
[CV 1/10] END …C=10, penalty=l2, solver=sag;, score=0.750 total time=   0.0s
[CV 2/10] END …C=10, penalty=l2, solver=sag;, score=0.750 total time=   0.0s
[CV 3/10] END …C=10, penalty=l2, solver=sag;, score=0.769 total time=   0.0s
[CV 4/10] END …C=10, penalty=l2, solver=sag;, score=0.808 total time=   0.0s
[CV 5/10] END …C=10, penalty=l2, solver=sag;, score=0.667 total time=   0.0s
[CV 6/10] END …C=10, penalty=l2, solver=sag;, score=0.745 total time=   0.0s
[CV 7/10] END …C=10, penalty=l2, solver=sag;, score=0.627 total time=   0.0s
[CV 8/10] END …C=10, penalty=l2, solver=sag;, score=0.882 total time=   0.0s
[CV 9/10] END …C=10, penalty=l2, solver=sag;, score=0.824 total time=   0.0s
```

```
[CV 10/10] END …C=10, penalty=l2, solver=sag;, score=0.902 total time=   0.0s
[CV 1/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.750 total time=
0.0s
[CV 2/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.750 total time=
0.0s
[CV 3/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.769 total time=
0.0s
[CV 4/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.788 total time=
0.0s
[CV 5/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.667 total time=
0.0s
[CV 6/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.745 total time=
0.0s
[CV 7/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.627 total time=
0.0s
[CV 8/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.882 total time=
0.0s
[CV 9/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.824 total time=
0.0s
[CV 10/10] END C=1, penalty=l2, solver=newton-cholesky;, score=0.902 total time=
0.0s
[CV 1/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 2/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 3/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 4/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 5/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 6/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 7/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 8/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 9/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 10/10] END C=10, penalty=elasticnet, solver=sag;, score=nan total time=
0.0s
[CV 1/10] END …C=1, penalty=l2, solver=saga;, score=0.750 total time=   0.0s
[CV 2/10] END …C=1, penalty=l2, solver=saga;, score=0.750 total time=   0.0s
[CV 3/10] END …C=1, penalty=l2, solver=saga;, score=0.769 total time=   0.0s
[CV 4/10] END …C=1, penalty=l2, solver=saga;, score=0.788 total time=   0.0s
[CV 5/10] END …C=1, penalty=l2, solver=saga;, score=0.667 total time=   0.0s
[CV 6/10] END …C=1, penalty=l2, solver=saga;, score=0.745 total time=   0.0s
[CV 7/10] END …C=1, penalty=l2, solver=saga;, score=0.627 total time=   0.0s
```

```
[CV 8/10] END …C=1, penalty=l2, solver=saga;, score=0.882 total time=   0.0s
[CV 9/10] END …C=1, penalty=l2, solver=saga;, score=0.824 total time=   0.0s
[CV 10/10] END …C=1, penalty=l2, solver=saga;, score=0.902 total time=   0.0s
[CV 1/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 2/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 3/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 4/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 5/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 6/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 7/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 8/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 9/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan total
time=   0.0s
[CV 10/10] END C=20, penalty=elasticnet, solver=newton-cholesky;, score=nan
total time=   0.0s
[CV 1/10] END …C=10, penalty=l2, solver=saga;, score=0.750 total time=   0.0s
[CV 2/10] END …C=10, penalty=l2, solver=saga;, score=0.750 total time=   0.0s
[CV 3/10] END …C=10, penalty=l2, solver=saga;, score=0.769 total time=   0.0s
[CV 4/10] END …C=10, penalty=l2, solver=saga;, score=0.808 total time=   0.0s
[CV 5/10] END …C=10, penalty=l2, solver=saga;, score=0.667 total time=   0.0s
[CV 6/10] END …C=10, penalty=l2, solver=saga;, score=0.745 total time=   0.0s
[CV 7/10] END …C=10, penalty=l2, solver=saga;, score=0.627 total time=   0.0s
[CV 8/10] END …C=10, penalty=l2, solver=saga;, score=0.882 total time=   0.0s
[CV 9/10] END …C=10, penalty=l2, solver=saga;, score=0.824 total time=   0.0s
[CV 10/10] END …C=10, penalty=l2, solver=saga;, score=0.902 total time=   0.0s
[CV 1/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 2/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 3/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 4/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 5/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 6/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 7/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 8/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 9/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
[CV 10/10] END …C=20, penalty=l1, solver=sag;, score=nan total time=   0.0s
```

[26]:
```
RandomizedSearchCV(cv=10, estimator=LogisticRegression(),
                   param_distributions={'C': [1, 10, 20],
                                        'penalty': ['l1', 'l2', 'elasticnet'],
```

```
                                        'solver': ['lbfgs', 'liblinear',
                                                   'newton-cg',
                                                   'newton-cholesky', 'sag',
                                                   'saga']},
                   scoring='accuracy', verbose=3)
```

[27]:
```
regressor_cv1.best_params_
```

[27]:
```
{'solver': 'sag', 'penalty': 'l2', 'C': 20}
```

[40]:
```
regressor2 = LogisticRegression(C = 20,penalty='l2',solver='sag')
regressor2.fit(x_train_scaled,y_train)
y_pred2 = regressor2.predict(x_test_scaled)
print(accuracy_score(y_pred2,y_test))
print(confusion_matrix(y_pred2,y_test))
print(classification_report(y_pred2,y_test))
```

```
0.7677165354330708
[[146  38]
 [ 21  49]]
              precision    recall  f1-score   support

           0       0.87      0.79      0.83       184
           1       0.56      0.70      0.62        70

    accuracy                           0.77       254
   macro avg       0.72      0.75      0.73       254
weighted avg       0.79      0.77      0.77       254
```

4 Logistic Regression Accuracy is : 76.77%

[29]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(x_train_scaled,y_train)
```

[29]:
```
DecisionTreeClassifier()
```

[30]:
```
y_pred = classifier.predict(x_test_scaled)
print(accuracy_score(y_pred,y_test))
print(confusion_matrix(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
0.7165354330708661
[[129  34]
 [ 38  53]]
              precision    recall  f1-score   support
```

```
              0       0.77      0.79      0.78       163
              1       0.61      0.58      0.60        91

       accuracy                           0.72       254
      macro avg       0.69      0.69      0.69       254
   weighted avg       0.71      0.72      0.72       254
```

[31]:
```
parameters1 = {
    'criterion' : ['gini','entropy','log_loss'],
    'splitter' : ['best','random'],
    'max_depth' : [1,2,3,4,5],
    'max_features' : ['auto','sqrt','log2']
    }
```

[32]:
```
classifier1 =␣
 ↪GridSearchCV(classifier,param_grid=parameters1,cv=10,scoring='accuracy')
classifier1.fit(x_train_scaled,y_train)
```

[32]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [1, 2, 3, 4, 5],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'splitter': ['best', 'random']},
             scoring='accuracy')
```

[41]:
```
classifier1.best_params_
```

[41]:
```
{'criterion': 'gini',
 'max_depth': 4,
 'max_features': 'log2',
 'splitter': 'best'}
```

[42]:
```
classifier = DecisionTreeClassifier(criterion = 'gini',max_depth =␣
 ↪4,max_features = 'log2',splitter = 'best')
classifier.fit(x_train_scaled,y_train)
y_pred1 = classifier.predict(x_test_scaled)
print(accuracy_score(y_pred1,y_test))
print(confusion_matrix(y_pred1,y_test))
print(classification_report(y_pred1,y_test))
```

```
0.7204724409448819
[[137  41]
 [ 30  46]]
              precision    recall  f1-score   support

           0       0.82      0.77      0.79       178
```

```
           1       0.53      0.61      0.56        76

    accuracy                           0.72       254
   macro avg       0.67      0.69      0.68       254
weighted avg       0.73      0.72      0.73       254
```

[35]:
```
classifier2 =␣
 ↪RandomizedSearchCV(classifier,param_distributions=parameters1,cv=10,scoring='accuracy',verb
classifier2.fit(x_train_scaled,y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[CV 1/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.769 total time=   0.0s
[CV 2/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.769 total time=   0.0s
[CV 3/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.750 total time=   0.0s
[CV 4/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.712 total time=   0.0s
[CV 5/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.647 total time=   0.0s
[CV 6/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.784 total time=   0.0s
[CV 7/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.608 total time=   0.0s
[CV 8/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.686 total time=   0.0s
[CV 9/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.765 total time=   0.0s
[CV 10/10] END criterion=gini, max_depth=4, max_features=sqrt, splitter=best;,
score=0.804 total time=   0.0s
[CV 1/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.654 total time=   0.0s
[CV 2/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.692 total time=   0.0s
[CV 3/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.692 total time=   0.0s
[CV 4/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.712 total time=   0.0s
[CV 5/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.627 total time=   0.0s
[CV 6/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.706 total time=   0.0s
[CV 7/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.745 total time=   0.0s
[CV 8/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.765 total time=   0.0s
```

```
[CV 9/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.667 total time=   0.0s
[CV 10/10] END criterion=entropy, max_depth=5, max_features=sqrt,
splitter=random;, score=0.745 total time=   0.0s
[CV 1/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.692 total time=   0.0s
[CV 2/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.731 total time=   0.0s
[CV 3/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.712 total time=   0.0s
[CV 4/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.827 total time=   0.0s
[CV 5/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.647 total time=   0.0s
[CV 6/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.667 total time=   0.0s
[CV 7/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.706 total time=   0.0s
[CV 8/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.706 total time=   0.0s
[CV 9/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.627 total time=   0.0s
[CV 10/10] END criterion=gini, max_depth=5, max_features=log2, splitter=random;,
score=0.843 total time=   0.0s
[CV 1/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.635 total time=   0.0s
[CV 2/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.654 total time=   0.0s
[CV 3/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.712 total time=   0.0s
[CV 4/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.654 total time=   0.0s
[CV 5/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.647 total time=   0.0s
[CV 6/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.647 total time=   0.0s
[CV 7/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.647 total time=   0.0s
[CV 8/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.647 total time=   0.0s
[CV 9/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.647 total time=   0.0s
[CV 10/10] END criterion=log_loss, max_depth=1, max_features=log2,
splitter=random;, score=0.647 total time=   0.0s
[CV 1/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.827 total time=   0.0s
[CV 2/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.712 total time=   0.0s
```

```
[CV 3/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.673 total time=   0.0s
[CV 4/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.769 total time=   0.0s
[CV 5/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.588 total time=   0.0s
[CV 6/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.667 total time=   0.0s
[CV 7/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.667 total time=   0.0s
[CV 8/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.824 total time=   0.0s
[CV 9/10] END criterion=entropy, max_depth=4, max_features=auto, splitter=best;,
score=0.667 total time=   0.0s
[CV 10/10] END criterion=entropy, max_depth=4, max_features=auto,
splitter=best;, score=0.745 total time=   0.0s
[CV 1/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.750 total time=   0.0s
[CV 2/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.692 total time=   0.0s
[CV 3/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.731 total time=   0.0s
[CV 4/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.635 total time=   0.0s
[CV 5/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.686 total time=   0.0s
[CV 6/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.627 total time=   0.0s
[CV 7/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.608 total time=   0.0s
[CV 8/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.706 total time=   0.0s
[CV 9/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.627 total time=   0.0s
[CV 10/10] END criterion=entropy, max_depth=5, max_features=auto,
splitter=random;, score=0.627 total time=   0.0s
[CV 1/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.865 total time=   0.0s
[CV 2/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.712 total time=   0.0s
[CV 3/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.692 total time=   0.0s
[CV 4/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.731 total time=   0.0s
[CV 5/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.745 total time=   0.0s
[CV 6/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.784 total time=   0.0s
```

```
[CV 7/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.804 total time=   0.0s
[CV 8/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.647 total time=   0.0s
[CV 9/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.824 total time=   0.0s
[CV 10/10] END criterion=log_loss, max_depth=5, max_features=auto,
splitter=best;, score=0.824 total time=   0.0s
[CV 1/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.769 total time=   0.0s
[CV 2/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.654 total time=   0.0s
[CV 3/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.654 total time=   0.0s
[CV 4/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.731 total time=   0.0s
[CV 5/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.569 total time=   0.0s
[CV 6/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.647 total time=   0.0s
[CV 7/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.627 total time=   0.0s
[CV 8/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.647 total time=   0.0s
[CV 9/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.686 total time=   0.0s
[CV 10/10] END criterion=entropy, max_depth=2, max_features=auto,
splitter=random;, score=0.667 total time=   0.0s
[CV 1/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.769 total time=   0.0s
[CV 2/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.673 total time=   0.0s
[CV 3/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.712 total time=   0.0s
[CV 4/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.827 total time=   0.0s
[CV 5/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.529 total time=   0.0s
[CV 6/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.686 total time=   0.0s
[CV 7/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.627 total time=   0.0s
[CV 8/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.647 total time=   0.0s
[CV 9/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.529 total time=   0.0s
[CV 10/10] END criterion=gini, max_depth=5, max_features=auto, splitter=random;,
score=0.784 total time=   0.0s
```

```
[CV 1/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.712 total time=   0.0s
[CV 2/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.654 total time=   0.0s
[CV 3/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.654 total time=   0.0s
[CV 4/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.673 total time=   0.0s
[CV 5/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.647 total time=   0.0s
[CV 6/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.608 total time=   0.0s
[CV 7/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.627 total time=   0.0s
[CV 8/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.647 total time=   0.0s
[CV 9/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.667 total time=   0.0s
[CV 10/10] END criterion=log_loss, max_depth=1, max_features=sqrt,
splitter=random;, score=0.647 total time=   0.0s
```

[35]:
```
RandomizedSearchCV(cv=10,
                   estimator=DecisionTreeClassifier(criterion='entropy',
                                                    max_depth=3,
                                                    max_features='log2'),
                   param_distributions={'criterion': ['gini', 'entropy',
                                                      'log_loss'],
                                        'max_depth': [1, 2, 3, 4, 5],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'splitter': ['best', 'random']},
                   scoring='accuracy', verbose=3)
```

[36]:
```
classifier2.best_params_
```

[36]:
```
{'splitter': 'best',
 'max_features': 'auto',
 'max_depth': 5,
 'criterion': 'log_loss'}
```

[43]:
```
classifier = DecisionTreeClassifier(criterion = 'log_loss',max_depth =␣
↪5,max_features = 'auto',splitter = 'best')
classifier.fit(x_train_scaled,y_train)
y_pred2 = classifier.predict(x_test_scaled)
print(accuracy_score(y_pred2,y_test))
print(confusion_matrix(y_pred2,y_test))
print(classification_report(y_pred2,y_test))
```

```
0.7480314960629921
[[144  41]
 [ 23  46]]
              precision    recall  f1-score   support

           0       0.86      0.78      0.82       185
           1       0.53      0.67      0.59        69

    accuracy                           0.75       254
   macro avg       0.70      0.72      0.70       254
weighted avg       0.77      0.75      0.76       254
```

5 Decision Tree Classifier Accuracy is : 74.80%

[ ]:

diabetes-model-training1

November 16, 2023

[1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
df = pd.read_csv("diabetes.csv")
df.head()
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[3]:
```
#If we observe above in the min the minumum values of some parameters are 0.
 ↪0000 like for gulcose,bloodpressure etc...
#so, we replace those values with mean of its column..
df['Glucose'] = df['Glucose'].replace(0,df['Glucose'].mean())
df['BloodPressure'] = df['BloodPressure'].replace(0,df['BloodPressure'].mean())
df['SkinThickness'] = df['SkinThickness'].replace(0,df['SkinThickness'].mean())
df['Insulin'] = df['Insulin'].replace(0,df['Insulin'].mean())
df['BMI'] = df['BMI'].replace(0,df['BMI'].mean())
```

[4]:
```
df.describe()
```

[4]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  121.681605      72.254807      26.606479  118.660163
std       3.369578   30.436016      12.115932       9.631241   93.080358
min       0.000000   44.000000      24.000000       7.000000   14.000000
25%       1.000000   99.750000      64.000000      20.536458   79.799479
50%       3.000000  117.000000      72.000000      23.000000   79.799479
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    32.450805                  0.471876   33.240885    0.348958
std      6.875374                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.500000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[5]:
```
#Independent and dependent features
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[6]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
 ↪33,random_state=80)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(514, 8)
(254, 8)
(514,)
(254,)
```

[7]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

[8]:
```
parameters = {
    'C' : [0.1,1,10,100,200],
    'gamma' : [1,0.1,0.01,0.001,0.0001],
    'kernel' : ['linear','rbf','poly','sigmoid']
    }
```

[9]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
svc = SVC()
svc.fit(x_train_scaled,y_train)
```

[9]:
```
SVC()
```

[10]:
```
from sklearn.metrics import␣
↪accuracy_score,classification_report,confusion_matrix
ypred = svc.predict(x_test_scaled)
print(accuracy_score(ypred,y_test))
```

```
0.7677165354330708
```

[11]:
```
model_svc =␣
↪RandomizedSearchCV(svc,param_distributions=parameters,cv=10,scoring='accuracy')
model_svc.fit(x_train_scaled,y_train)
```

[11]:
```
RandomizedSearchCV(cv=10, estimator=SVC(),
                   param_distributions={'C': [0.1, 1, 10, 100, 200],
                                        'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                                        'kernel': ['linear', 'rbf', 'poly',
                                                   'sigmoid']},
                   scoring='accuracy')
```

[12]:
```
model_svc.best_params_
```

[12]:
```
{'kernel': 'sigmoid', 'gamma': 0.01, 'C': 1}
```

[15]:
```
svc = SVC(kernel = 'sigmoid',gamma = 0.01,C = 1)
svc.fit(x_train_scaled,y_train)
y_pred = svc.predict(x_test_scaled)
print(accuracy_score(y_pred,y_test))
print(confusion_matrix(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
0.7519685039370079
[[146  42]
 [ 21  45]]
              precision    recall  f1-score   support

           0       0.87      0.78      0.82       188
           1       0.52      0.68      0.59        66

    accuracy                           0.75       254
   macro avg       0.70      0.73      0.71       254
weighted avg       0.78      0.75      0.76       254
```

1 The Accuracy of Support Vector Classifier is : 76.77%

2 Naive Bayers Algorithm

[16]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train_scaled,y_train)
```

[16]:
```
GaussianNB()
```

[17]:
```
y_pred1 = gnb.predict(x_test_scaled)
print(accuracy_score(y_pred1,y_test))
print(confusion_matrix(y_pred1,y_test))
print(classification_report(y_pred1,y_test))
```

```
0.7204724409448819
[[136  40]
 [ 31  47]]
              precision    recall  f1-score   support

           0       0.81      0.77      0.79       176
           1       0.54      0.60      0.57        78

    accuracy                           0.72       254
   macro avg       0.68      0.69      0.68       254
weighted avg       0.73      0.72      0.72       254
```

[18]:
```
parameters1 = {
    'priors': [None],
    'var_smoothing': [0.00000001, 0.000000001, 0.00000001]
}
```

[20]:
```
random_gnb =␣
 ↪RandomizedSearchCV(gnb,param_distributions=parameters1,cv=9,scoring =␣
 ↪'accuracy',verbose = 3)
random_gnb.fit(x_train_scaled,y_train)
```

```
Fitting 9 folds for each of 3 candidates, totalling 27 fits
[CV 1/9] END ..priors=None, var_smoothing=1e-08;, score=0.810 total time=   0.0s
[CV 2/9] END ..priors=None, var_smoothing=1e-08;, score=0.719 total time=   0.0s
[CV 3/9] END ..priors=None, var_smoothing=1e-08;, score=0.737 total time=   0.0s
[CV 4/9] END ..priors=None, var_smoothing=1e-08;, score=0.702 total time=   0.0s
[CV 5/9] END ..priors=None, var_smoothing=1e-08;, score=0.649 total time=   0.0s
[CV 6/9] END ..priors=None, var_smoothing=1e-08;, score=0.667 total time=   0.0s
[CV 7/9] END ..priors=None, var_smoothing=1e-08;, score=0.825 total time=   0.0s
[CV 8/9] END ..priors=None, var_smoothing=1e-08;, score=0.789 total time=   0.0s
```

```
[CV 9/9] END ..priors=None, var_smoothing=1e-08;, score=0.860 total time=   0.0s
[CV 1/9] END ..priors=None, var_smoothing=1e-09;, score=0.810 total time=   0.0s
[CV 2/9] END ..priors=None, var_smoothing=1e-09;, score=0.719 total time=   0.0s
[CV 3/9] END ..priors=None, var_smoothing=1e-09;, score=0.737 total time=   0.0s
[CV 4/9] END ..priors=None, var_smoothing=1e-09;, score=0.702 total time=   0.0s
[CV 5/9] END ..priors=None, var_smoothing=1e-09;, score=0.649 total time=   0.0s
[CV 6/9] END ..priors=None, var_smoothing=1e-09;, score=0.667 total time=   0.0s
[CV 7/9] END ..priors=None, var_smoothing=1e-09;, score=0.825 total time=   0.0s
[CV 8/9] END ..priors=None, var_smoothing=1e-09;, score=0.789 total time=   0.0s
[CV 9/9] END ..priors=None, var_smoothing=1e-09;, score=0.860 total time=   0.0s
[CV 1/9] END ..priors=None, var_smoothing=1e-08;, score=0.810 total time=   0.0s
[CV 2/9] END ..priors=None, var_smoothing=1e-08;, score=0.719 total time=   0.0s
[CV 3/9] END ..priors=None, var_smoothing=1e-08;, score=0.737 total time=   0.0s
[CV 4/9] END ..priors=None, var_smoothing=1e-08;, score=0.702 total time=   0.0s
[CV 5/9] END ..priors=None, var_smoothing=1e-08;, score=0.649 total time=   0.0s
[CV 6/9] END ..priors=None, var_smoothing=1e-08;, score=0.667 total time=   0.0s
[CV 7/9] END ..priors=None, var_smoothing=1e-08;, score=0.825 total time=   0.0s
[CV 8/9] END ..priors=None, var_smoothing=1e-08;, score=0.789 total time=   0.0s
[CV 9/9] END ..priors=None, var_smoothing=1e-08;, score=0.860 total time=   0.0s
```

[20]:
```
RandomizedSearchCV(cv=9, estimator=GaussianNB(),
                   param_distributions={'priors': [None],
                                        'var_smoothing': [1e-08, 1e-09, 1e-08]},
                   scoring='accuracy', verbose=3)
```

[31]:
```
y_pred2 = random_gnb.predict(x_test_scaled)
print(accuracy_score(y_pred2,y_test))
```

```
0.7204724409448819
```

[28]:
```
from sklearn.model_selection import GridSearchCV
grid_gnb = GridSearchCV(gnb,param_grid=parameters1,cv=9,scoring =␣
 ↪'accuracy',verbose = 3)
grid_gnb.fit(x_train_scaled,y_train)
```

```
Fitting 9 folds for each of 3 candidates, totalling 27 fits
[CV 1/9] END ..priors=None, var_smoothing=1e-08;, score=0.810 total time=   0.0s
[CV 2/9] END ..priors=None, var_smoothing=1e-08;, score=0.719 total time=   0.0s
[CV 3/9] END ..priors=None, var_smoothing=1e-08;, score=0.737 total time=   0.0s
[CV 4/9] END ..priors=None, var_smoothing=1e-08;, score=0.702 total time=   0.0s
[CV 5/9] END ..priors=None, var_smoothing=1e-08;, score=0.649 total time=   0.0s
[CV 6/9] END ..priors=None, var_smoothing=1e-08;, score=0.667 total time=   0.0s
[CV 7/9] END ..priors=None, var_smoothing=1e-08;, score=0.825 total time=   0.0s
[CV 8/9] END ..priors=None, var_smoothing=1e-08;, score=0.789 total time=   0.0s
[CV 9/9] END ..priors=None, var_smoothing=1e-08;, score=0.860 total time=   0.0s
[CV 1/9] END ..priors=None, var_smoothing=1e-09;, score=0.810 total time=   0.0s
[CV 2/9] END ..priors=None, var_smoothing=1e-09;, score=0.719 total time=   0.0s
[CV 3/9] END ..priors=None, var_smoothing=1e-09;, score=0.737 total time=   0.0s
```

```
[CV 4/9] END ..priors=None, var_smoothing=1e-09;, score=0.702 total time=   0.0s
[CV 5/9] END ..priors=None, var_smoothing=1e-09;, score=0.649 total time=   0.0s
[CV 6/9] END ..priors=None, var_smoothing=1e-09;, score=0.667 total time=   0.0s
[CV 7/9] END ..priors=None, var_smoothing=1e-09;, score=0.825 total time=   0.0s
[CV 8/9] END ..priors=None, var_smoothing=1e-09;, score=0.789 total time=   0.0s
[CV 9/9] END ..priors=None, var_smoothing=1e-09;, score=0.860 total time=   0.0s
[CV 1/9] END ..priors=None, var_smoothing=1e-08;, score=0.810 total time=   0.0s
[CV 2/9] END ..priors=None, var_smoothing=1e-08;, score=0.719 total time=   0.0s
[CV 3/9] END ..priors=None, var_smoothing=1e-08;, score=0.737 total time=   0.0s
[CV 4/9] END ..priors=None, var_smoothing=1e-08;, score=0.702 total time=   0.0s
[CV 5/9] END ..priors=None, var_smoothing=1e-08;, score=0.649 total time=   0.0s
[CV 6/9] END ..priors=None, var_smoothing=1e-08;, score=0.667 total time=   0.0s
[CV 7/9] END ..priors=None, var_smoothing=1e-08;, score=0.825 total time=   0.0s
[CV 8/9] END ..priors=None, var_smoothing=1e-08;, score=0.789 total time=   0.0s
[CV 9/9] END ..priors=None, var_smoothing=1e-08;, score=0.860 total time=   0.0s
```

[28]:
```
GridSearchCV(cv=9, estimator=GaussianNB(var_smoothing=1e-08),
             param_grid={'priors': [None],
                         'var_smoothing': [1e-08, 1e-09, 1e-08]},
             scoring='accuracy', verbose=3)
```

[32]:
```
y_pred3 = grid_gnb.predict(x_test_scaled)
print(accuracy_score(y_pred3,y_test))
```

```
0.7204724409448819
```

3 Accuracy of Naive Bayes : 72.04%

4 Accuracy of All Models are :

[ ]:
```

```

5 LogisticRegression - 76.77%

6 Decision Tree Classifier - 74.80%

7 Naive Bayes : 72.04%

8 SVC - 76.77%

[ ]:
```

```