Implementation of SVC ¶

Accessing the Dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
db = pd.read_csv('diabetes.csv')
```

In [3]:
```
db.head()
```

Out[3]:

	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

In [4]:
```
y = db['Outcome']
y
```

Out[4]:
```
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
```

In [5]:
```
y.value_counts()
```

Out[5]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [6]:
```
X = db.drop(['Outcome'], axis = 1) # dropping target variable
```

In [7]:
```
X.head()
```

Out[7]:

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

In [8]:
```
X.shape
```

Out[8]:
```
(768, 8)
```

In [9]:
```
X.describe()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

no null values, only numerical values

Splitting the Data

In [10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 10)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[10]:
```
((614, 8), (154, 8), (614,), (154,))
```

Building the Model

In [11]:
```
from sklearn.svm import SVC

svc_lin = SVC(kernel = 'linear', probability = True) #we don't have a predicted class, we need prob val
svc_lin = svc_lin.fit(X_train, y_train)
y_pred = svc_lin.predict(X_test)

y_pred
```

Out[11]:
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0],
      dtype=int64)
```

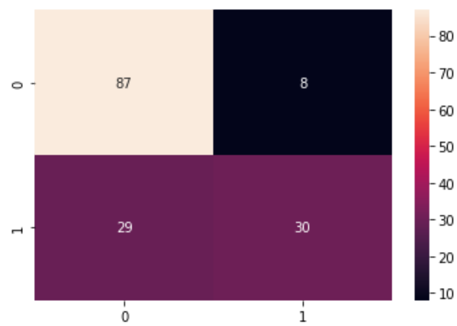
In [12]:
```
y_test
```

Out[12]:
```
568    0
620    0
456    0
197    1
714    0
      ..
264    1
706    1
194    0
179    1
514    0
Name: Outcome, Length: 154, dtype: int64
```

Calculating the Performance

In [13]:
```
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score

cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
score = roc_auc_score(y_test, y_pred)
fpr, tpr, _ = roc_curve(y_test, y_pred)#false positive rate & true positive rate

sns.heatmap(cm, annot = True)
print('The Report,:\n', report)
print('The ROC-AUC-Score', score)
```

```
The Report,:
               precision    recall  f1-score   support

           0       0.75      0.92      0.82        95
           1       0.79      0.51      0.62        59

    accuracy                           0.76       154
   macro avg       0.77      0.71      0.72       154
weighted avg       0.77      0.76      0.75       154

The ROC-AUC-Score 0.7121320249776985
```



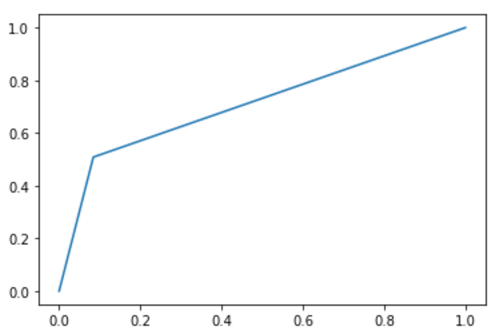
87 --> true positive

8 --> false positive

29 --> false positive

30 --> true negative

In [14]:
```
plt.plot(fpr, tpr);
```



In [15]:
```
y_pred_prob = svc_lin.predict_proba(X_test) #probability values are different in different machines due to 5-fold cross validatic
y_pred_prob
```

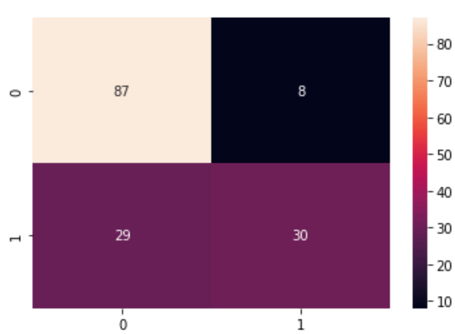
Out[15]:
```
array([[0.45441093, 0.54558907],
       [0.78710559, 0.21289441],
       [0.51941624, 0.48058376],
       [0.87749624, 0.12250376],
       [0.88402775, 0.11597225],
       [0.93732449, 0.06267551],
       [0.89674568, 0.10325432],
       [0.68440209, 0.31559791],
       [0.91873041, 0.08126959],
       [0.60758483, 0.39241517],
       [0.90784189, 0.09215811],
       [0.73960385, 0.26039615],
       [0.14436853, 0.85563147],
       [0.68257124, 0.31742876],
       [0.87392465, 0.12607535],
       [0.31487872, 0.68512128],
       [0.25121606, 0.74878394],
       [0.94036901, 0.05963099],
       [0.88671946, 0.11328054],
```

In [16]:
```
y_pred
```

Out[16]:
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0],
      dtype=int64)
```

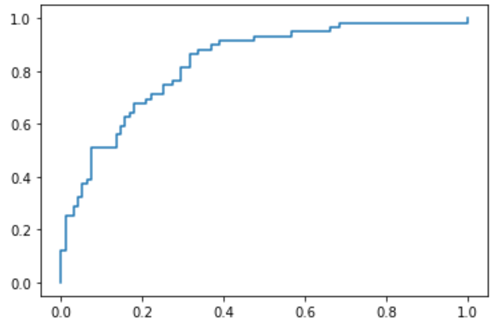
In [17]:
```
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score

cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
score = roc_auc_score(y_test, y_pred)
fpr, tpr, _ = roc_curve(y_test, y_pred_prob[:,1])#false positive rate & true positive rate

sns.heatmap(cm, annot = True)
print('The Report,:\n', report)
print('The ROC-AUC-Score', score)
```

```
The Report,:
               precision    recall  f1-score   support

           0       0.75      0.92      0.82        95
           1       0.79      0.51      0.62        59

    accuracy                           0.76       154
   macro avg       0.77      0.71      0.72       154
weighted avg       0.77      0.76      0.75       154

The ROC-AUC-Score 0.7121320249776985
```



In [18]:
```
plt.plot(fpr, tpr)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x1a9284bf520>]
```



doesn't have adjusted values now

Area under the curve:

random model-- 0.5

perfect model-- 1

our model-- 0.71

Hyper-paramter Tuning

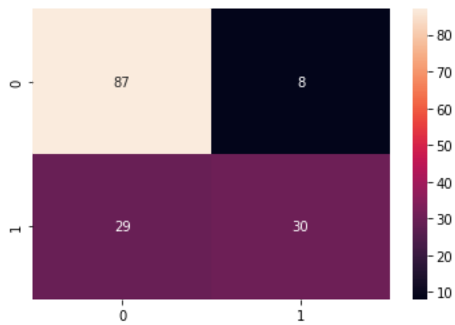
changing the value of a parameter for maximum accuracy

kernel

In [19]:
```
def SVC_tuning_kernel(kernel):
    model = SVC(kernel = kernel)
    model = model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    score = roc_auc_score(y_test, y_pred)
    print('The SVC with kernel:\n', kernel)
    print()
    print(' ********************* ')
    print('Confusion Matrix:\n', cm)
    print('The report:\n', report)
    print('The ROC-AUC-Score:', score)
    sns.heatmap(cm, annot = True);
```

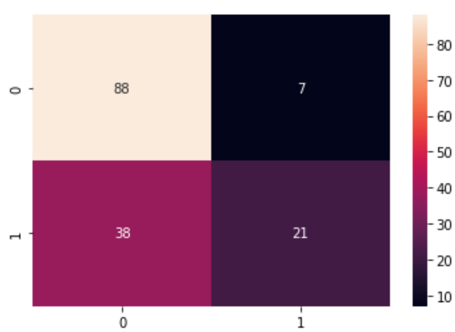
In [20]:
```
## Calling the function

SVC_tuning_kernel('linear')# argument to be passed has to be of type str
```

```
The SVC with kernel:
 linear

 ********************
Confusion Matrix:
 [[87  8]
 [29 30]]
The report:
               precision    recall  f1-score   support

           0       0.75      0.92      0.82        95
           1       0.79      0.51      0.62        59

    accuracy                           0.76       154
   macro avg       0.77      0.71      0.72       154
weighted avg       0.77      0.76      0.75       154

The ROC-AUC-Score: 0.7121320249776985
```



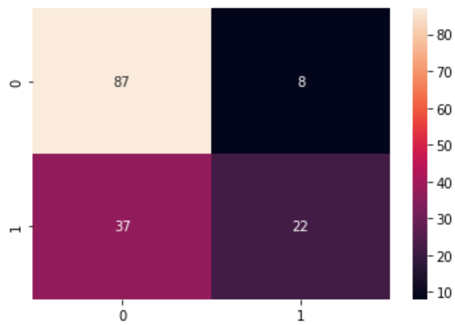
In [21]:
```
# Calling the function again with another kernel
SVC_tuning_kernel('poly')
```

```
The SVC with kernel:
 poly

 ********************
Confusion Matrix:
 [[88  7]
 [38 21]]
The report:
               precision    recall  f1-score   support

           0       0.70      0.93      0.80        95
           1       0.75      0.36      0.48        59

    accuracy                           0.71       154
   macro avg       0.72      0.64      0.64       154
weighted avg       0.72      0.71      0.68       154

The ROC-AUC-Score: 0.6411239964317573
```



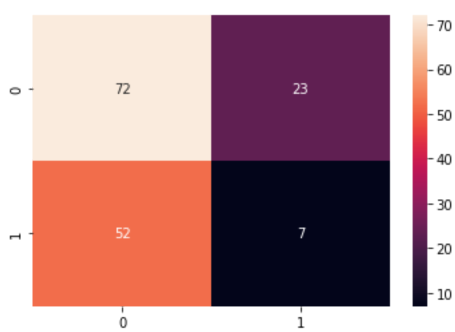
In [22]:
```
SVC_tuning_kernel('rbf')
```

```
The SVC with kernel:
 rbf

 ********************
Confusion Matrix:
 [[87  8]
 [37 22]]
The report:
              precision    recall  f1-score   support

           0       0.70      0.92      0.79        95
           1       0.73      0.37      0.49        59

    accuracy                           0.71       154
   macro avg       0.72      0.64      0.64       154
weighted avg       0.71      0.71      0.68       154

The ROC-AUC-Score: 0.6443354148082068
```



In [23]:
```
SVC_tuning_kernel('sigmoid')
```

```
The SVC with kernel:
 sigmoid

 ********************
Confusion Matrix:
 [[72 23]
 [52  7]]
The report:
              precision    recall  f1-score   support

           0       0.58      0.76      0.66        95
           1       0.23      0.12      0.16        59

    accuracy                           0.51       154
   macro avg       0.41      0.44      0.41       154
weighted avg       0.45      0.51      0.47       154

The ROC-AUC-Score: 0.4382694023193577
```



The best kernel after tuning is **linear**

In [27]:
```
# Tuning regularisation paramter
# Regularisation -

def SVC_tuning_C(C_list): # C_list --> a list of C values
    for c in C_list:
        model = SVC(kernel = 'linear', C = c)
        model = model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        score = roc_auc_score(y_test, y_pred)
        print('C:', c, '===>', 'Score:', score)
```

In [28]:
```
C_list = [0.1, 1, 1.1, 2, 3, 4, 5, 10, 15, 20, 25, 30]
```

In [29]:
```
SVC_tuning_C(C_list)
```

```
C: 0.1 ===> Score: 0.703657448706512
C: 1 ===> Score: 0.7121320249776985
C: 1.1 ===> Score: 0.7121320249776985
C: 2 ===> Score: 0.7068688670829617
C: 3 ===> Score: 0.7068688670829617
C: 4 ===> Score: 0.7068688670829617
C: 5 ===> Score: 0.7068688670829617
C: 10 ===> Score: 0.7016057091882248
C: 15 ===> Score: 0.7016057091882248
C: 20 ===> Score: 0.7290811775200713
C: 25 ===> Score: 0.7068688670829617
C: 30 ===> Score: 0.7068688670829617
```

In [30]:
```
C_list2 = [18, 19, 21, 22]

SVC_tuning_C(C_list2)
```

```
C: 18 ===> Score: 0.715343443354148
C: 19 ===> Score: 0.7068688670829617
C: 21 ===> Score: 0.7068688670829617
C: 22 ===> Score: 0.7068688670829617
```

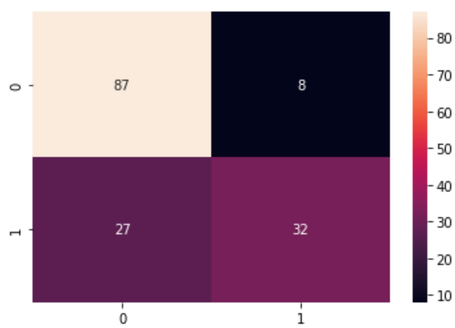
After tuning the best value of C is **20**

The best model is the one with **kernel = linear** and **C = 20**

The Final Model

In [31]:
```
svc = SVC(kernel = 'linear', C = 20, probability = True)
svc = svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
y_pred_prob = svc.predict_proba(X_test)

cm = confusion_matrix(y_test, y_pred)
score = roc_auc_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
dpr, tpr,_ = roc_curve(y_test, y_pred_prob[:,1])

print('The Confusion Matrix:')
sns.heatmap(cm, annot = True)
print('ROC-AUC-Score:', score)
print('The report:', report)
```

```
The Confusion Matrix:
ROC-AUC-Score: 0.7290811775200713
The report:               precision    recall  f1-score   support

           0       0.76      0.92      0.83        95
           1       0.80      0.54      0.65        59

    accuracy                           0.77       154
   macro avg       0.78      0.73      0.74       154
weighted avg       0.78      0.77      0.76       154
```



In [ ]: