KNN

In [1]:

```
#Loading basic dependencies

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
from sklearn.model_selection import train_test_split as tts
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [2]:

```
data = pd.read_csv('Iris.csv')
```

In [3]:

```
data.shape
```

Out[3]:

```
(150, 6)
```

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [5]:

```
data.describe()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [6]:

```
y = data['Species']
X = data.drop(['Species'],axis=1)
```

In [8]:

```
#Converting target categorical column to numerical column
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

In [9]:

```
xtrain,xtest,ytrain,ytest = tts(X,y,test_size=0.2,random_state=42)
```

In [11]:

```
from sklearn.neighbors import KNeighborsClassifier
```

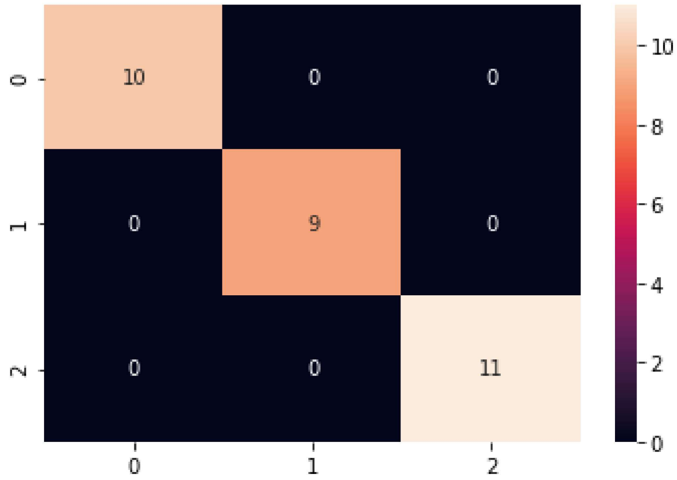
In [12]:

```
# KNN Model
clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(xtrain,ytrain)
ypred = clf.predict(xtest)
```

In [14]:

```
cmKNN = confusion_matrix(ypred,ytest)
sns.heatmap(cmKNN, annot=True)
```

Out[14]: <AxesSubplot:>



In [15]:
```
print(accuracy_score(ypred,ytest))
```

```
1.0
```

In [17]:
```
print(classification_report(ypred,ytest))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

Conclusion: KNN is a supervised ML Algorithm which has yielded 100% accuracy in our Iris dataset.

In [ ]:
```

```

SVM

In [1]:
```
#Loading basic dependencies

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
from sklearn.model_selection import train_test_split as tts
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [2]:
```
data = pd.read_csv('Iris.csv') #Reading data
```

In [3]:
```
data.info() #Getting basic info about data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [4]:
```
data.keys()
```

Out[4]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [5]:
```
data.sample() #Getting sample row of data
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
126	127	6.2	2.8	4.8	1.8	Iris-virginica

In [6]:

```
data.describe() #Getting descriptive statistics of data
```

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [7]:

```
data.shape #getting shape of data
```

Out[7]:

```
(150, 6)
```

In [8]:

```
# Getting dependent and independent variables
y = data['Species']
X = data.drop(['Species'], axis=1)
```

In [9]:

```
X.head()
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2

In [10]:
```
y[:5]
```

Out[10]:
```
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: Species, dtype: object
```

In [11]:
```
#Converting target categorical column to numerical column
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

In [12]:
```
y = le.fit_transform(y)
```

In [13]:
```
y
```

Out[13]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [14]:
```
#Train-Test split
xtrain,xtest,ytrain,ytest = tts(X,y,test_size=0.25, random_state=42)
```

In [15]:
```
# Getting shape of training and testing data
print(xtrain.shape)
```

```
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(112, 5)
(38, 5)
(112,)
(38,)
```

In [16]:

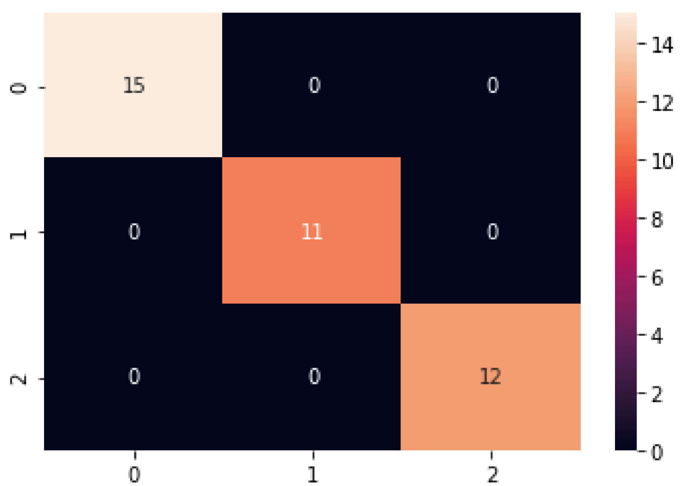
```
# Applying SVM linear Kernel
svc_clf = SVC(kernel='linear')
svc_clf.fit(xtrain,ytrain) #fitting SVM linear kernel
ypred = svc_clf.predict(xtest) #predicting based on SVM kernel
```

In [17]:

```
#Getting confusion matrix
cm = confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

Out[17]:

```
<AxesSubplot:>
```



In [18]:

```
print(accuracy_score(ypred,ytest))
```

```
1.0
```

In [19]:

```
print(classification_report(ypred,ytest))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

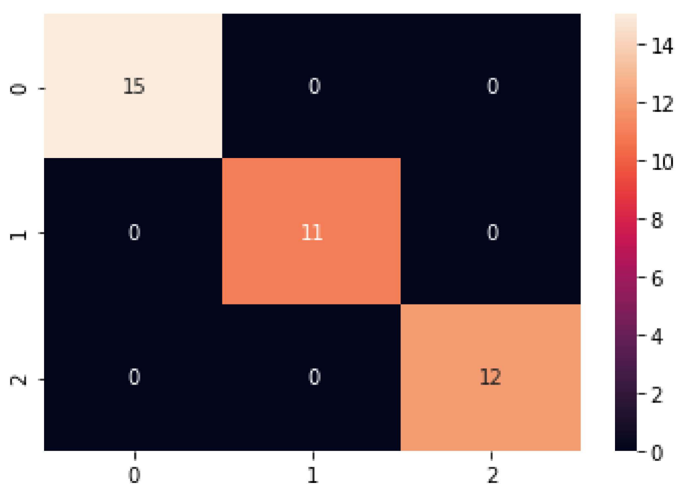
In [20]:

```
# Applying Radial Basis Function Kernel
svc_clf_rbf = SVC(kernel='rbf')
svc_clf_rbf.fit(xtrain,ytrain) # Fitting training data on rbf kernel
ypred = svc_clf_rbf.predict(xtest) # predicting observations based on rbf Kernel
```

In [21]:

```
# Getting confusionmatrix obtained by model fitted by RBF Kernel
cm2 = confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

Out[21]: <AxesSubplot:>



In [22]:

```
print(accuracy_score(ypred,ytest)) #getting accuracy score
```

```
1.0
```

In [23]:
```
print(classification_report(ypred,ytest)) #getting classification report
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

Conclusion: We conclude that in SVM both Linear and RBF Kernel Worked best and gave 100% accuracy in Iris dataset.

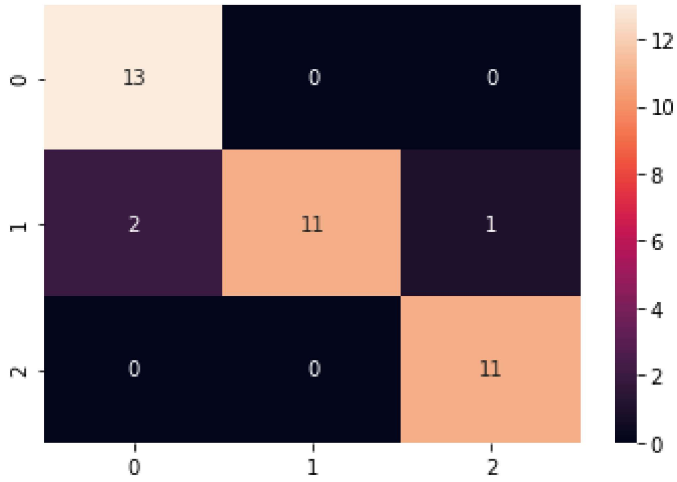
Naive Bayes

In [32]:
```
#Importing Naive Bayes
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
```

In [27]:
```
clf = MultinomialNB() #fitting Multinomial Naive Bayes
clf.fit(xtrain,ytrain)
ypred = clf.predict(xtest)
```

In [29]:
```
#getting confusion_matrix obtained by fitting Multinomial naive Bayes
cm = confusion_matrix(ypred,ytest)
sns.heatmap(cm,annot=True)
```

Out[29]: <AxesSubplot:>



In [30]:

```
print(accuracy_score(ypred,ytest)) #getting accuracy score
```

```
0.9210526315789473
```

In [31]:

```
print(classification_report(ypred,ytest)) #getting classification report
```

```
              precision    recall  f1-score   support

           0       0.87      1.00      0.93        13
           1       1.00      0.79      0.88        14
           2       0.92      1.00      0.96        11

    accuracy                           0.92        38
   macro avg       0.93      0.93      0.92        38
weighted avg       0.93      0.92      0.92        38
```

In [38]:

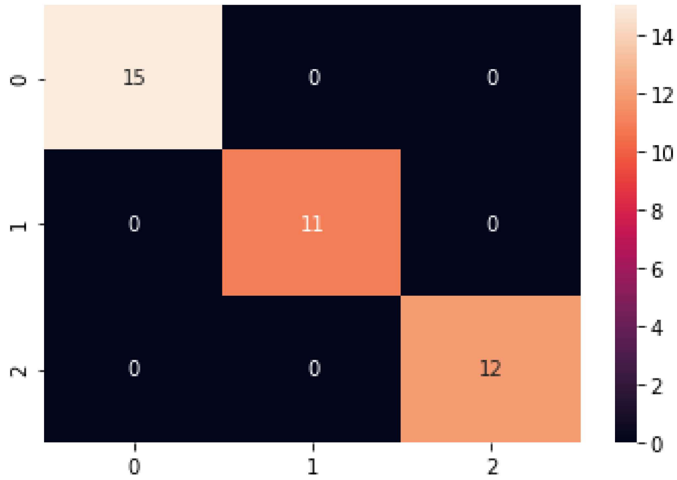
```
clf2 = GaussianNB() #Fitting Gaussian Naive Bayes
clf2.fit(xtrain,ytrain)
ypred2 = clf2.predict(xtest)
```

In [39]:

```
cm2 = confusion_matrix(ypred2,ytest) #Getting Confusion Matrix obtained by applying Gaussian Naive Bayes
sns.heatmap(cm2,annot=True)
```

Out[39]:

```
<AxesSubplot:>
```



In [35]:

```
print(accuracy_score(ypred2,ytest)) #getting accuracy score
```

```
1.0
```

In [37]:

```
print(classification_report(ypred2,ytest)) #getting classification report
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

We observe that Gaussian naive bayes performed better than Multinomial naive Bayes because Multinomial Naive Bayes perform better in case of discrete values and Gaussian Naive Bayes perform better in case of continuous values. Because our dataset has continuous values, Gaussian Naive Bayes performed better than Multinomial Naive Bayes

In [ ]: