SKILL ACTIVITY NO: 3

Name:Shubham Ravasaheb Patil Date:19/8/2021
PRN:
School: School of Data Science
Program: Machine Learning
Batch: ML12
Module Name: Python Programming
Module Code: ML101

Title: Perform Classification on the Glass Dataset

Skills/Competencies to be acquired:

1. To gain an understanding of data and find clues from the data.
2. Assess assumptions on which statistical inference will be based.
3. To check the quality of data for further processing and cleaning if necessary.
4. To check for anomalies or outliers that may impact model.
5. Data Visualization.

Duration of activity: 1 Hour

1.What is the purpose of this activity?

Preview data.
Check total number of entries and column types.
Check any null values.
Check duplicate entries.
Plot distribution of numeric data (univariate and pairwise joint distribution).
Plot count distribution of categorical data.

2.Steps performed in this activity.

1)Exploratory Data Analysis
2)Balancing data
3)Applying classification models

3.What resources / materials / equipment / tools did you use for this activity?

1)Google colab
2)jupyter notebook
3)ml libraries

4.What skills did you acquire?

1)Oversampling imbalanced data
2)Hyperparameter tunning of models
3)Evaluation of model

5.Time taken to complete the activity? 2hr

Importing libraries

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

In [2]:
```
df = pd.read_csv('/content/glass.csv')
```

In [ ]:
```
df.head()
```

Out[ ]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

In [ ]:
```
df.shape[0]
```

Out[ ]:
```
214
```

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
RI      0
Na      0
Mg      0
Al      0
Si      0
K       0
Ca      0
Ba      0
Fe      0
Type    0
dtype: int64
```

There are no null values present in our data

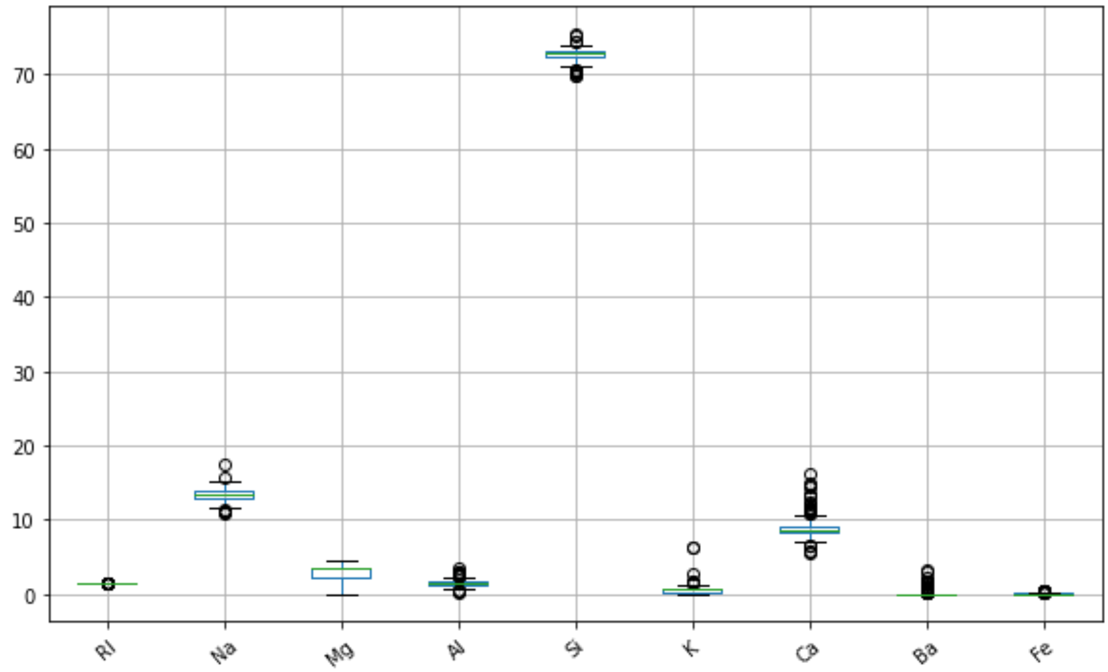
In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   RI      214 non-null    float64
 1   Na      214 non-null    float64
 2   Mg      214 non-null    float64
 3   Al      214 non-null    float64
 4   Si      214 non-null    float64
 5   K       214 non-null    float64
 6   Ca      214 non-null    float64
 7   Ba      214 non-null    float64
 8   Fe      214 non-null    float64
 9   Type    214 non-null    int64  
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
```

as you can see that the variable sex is or object type so we need to convert it into numerical type

Now we are going to do boxplot to see if the spread of the data

In [ ]:
```
plt.figure(figsize=[10,6])
d = df.drop(columns=['Type'])
d.boxplot()
plt.xticks(rotation = '40')
plt.show()
```



The values of variables vary significantly we scale the data

In [91]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
df['Al']=sc.fit_transform(df[['Al']])
df['Na']=sc.fit_transform(df[['Na']])
df['Mg']=sc.fit_transform(df[['Mg']])
df['RI']=sc.fit_transform(df[['RI']])
df['Si']=sc.fit_transform(df[['Si']])
df['K']=sc.fit_transform(df[['K']])
df['Ca']=sc.fit_transform(df[['Ca']])
df['Ba']=sc.fit_transform(df[['Ba']])
df['Fe']=sc.fit_transform(df[['Fe']])
```

In [ ]:
```python
df['Type'].unique()
```

Out[ ]:
```
array([1, 2, 3, 5, 6, 7])
```

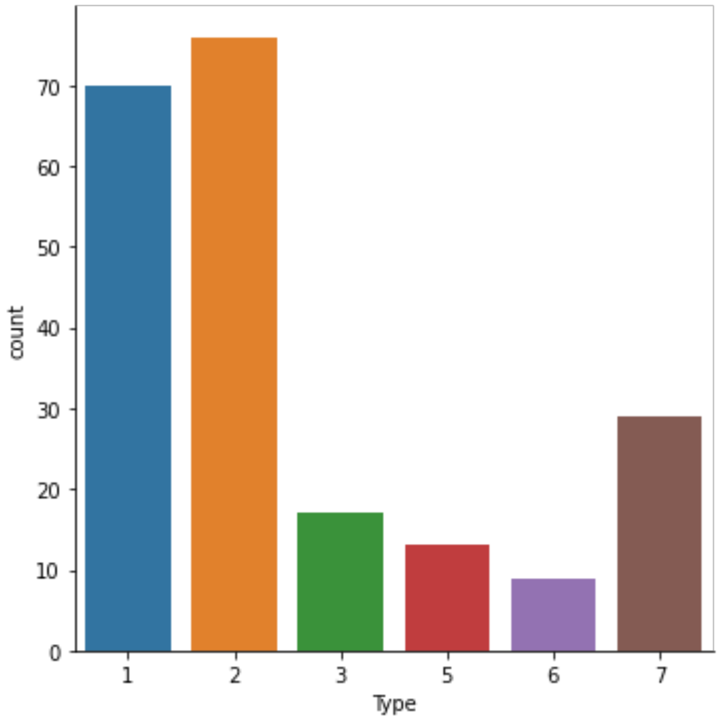
there are 7 types of glass

In [3]:
```python
df['Type'].value_counts()
```

Out[3]:
```
2    76
1    70
7    29
3    17
5    13
6     9
Name: Type, dtype: int64
```

In [4]:
```python
sns.catplot(x = 'Type',data = df,kind = 'count')
```

Out[4]:
```
<seaborn.axisgrid.FacetGrid at 0x7fb26b1f1310>
```



Here we can see that class 1 and 2 have more values than that of the other it classes it may cause low accuracy to other classes. To avoid that we should use oversampling on our data

Oversampling

Splitting the dataset into test and train part

In [170]:
```
x = df.drop(columns=['Type'])
y = df['Type']
```

In [171]:
```
# for oversampling we use smote
from imblearn.over_sampling import SMOTE
smote = SMOTE()
xover, yover = smote.fit_resample(x, y)
```

In [173]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(xover,yover,test_size =
0.2 , random_state = 1)
```

In [175]:
```
accuracy = []
```

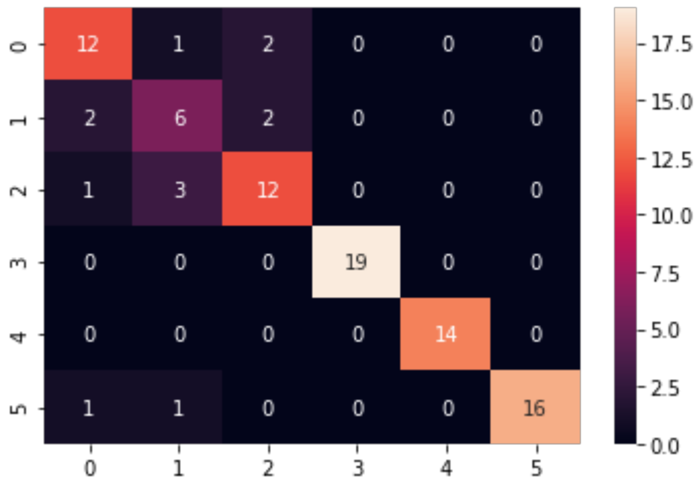
Implementing classification models on the splitted data

1) LogesticRegression

In [178]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [179]:
```
from sklearn.metrics import confusion_matrix,classification_report,a
ccuracy_score
from sklearn.metrics import precision_recall_fscore_support as score
acc=accuracy_score(ytest,ypred)
print("Accuracy is :",acc)
cm=confusion_matrix(ytest,ypred)

print(classification_report(ytest,ypred))
sns.heatmap(cm,annot=True)

plt.show()
```

```
Accuracy is :  0.8586956521739131
              precision    recall  f1-score   support

           1       0.75      0.80      0.77        15
           2       0.55      0.60      0.57        10
           3       0.75      0.75      0.75        16
           5       1.00      1.00      1.00        19
           6       1.00      1.00      1.00        14
           7       1.00      0.89      0.94        18

    accuracy                           0.86        92
   macro avg       0.84      0.84      0.84        92
weighted avg       0.87      0.86      0.86        92
```



hyper parameter tunning of logesticRegression

In [183]:

```
#model
import warnings
warnings.filterwarnings("ignore")
model=LogisticRegression()
#Parameters
penalty =['l1', 'l2', 'elasticnet']
C=[10,1,0.1,0.001,0.0001]
solver=['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
#grid
grid=dict(solver=solver,C=C,penalty=penalty)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=10,n_repeats=3,random_state=1)
#Grid Search cv
from sklearn.model_selection import GridSearchCV
gridcv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring="a
ccuracy",error_score=0)
result=gridcv.fit(x,y)
print(result.best_score_)
print(result.best_params_)
```

```
0.6523088023088025
{'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
```

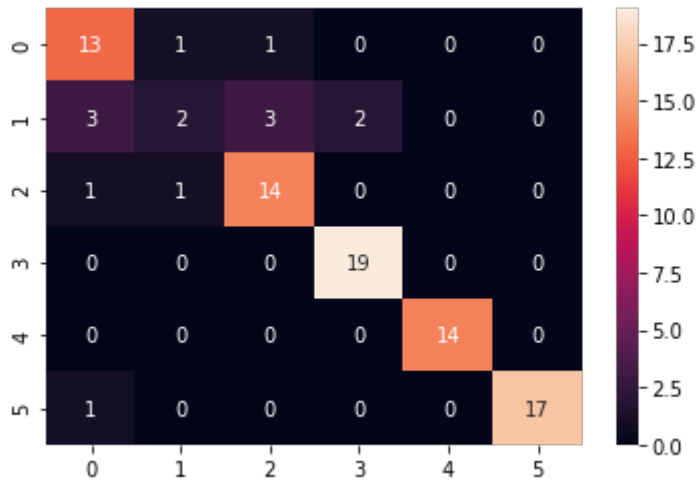
Retraining the logistic regression model on best parameters

In [181]:

```
model=LogisticRegression(C= 10, penalty= 'l1', solver='liblinear')
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

```
In [182]: #Model Evaluation
lr_pre,lr_recall,lr_fsc,support=score(ytest,ypred,average='macro')
lr_acc=accuracy_score(ytest,ypred)
print("Accuracy is :",acc)
cm=confusion_matrix(ytest,ypred)

sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
Accuracy is : 0.8586956521739131
              precision    recall  f1-score   support

           1       0.72      0.87      0.79        15
           2       0.50      0.20      0.29        10
           3       0.78      0.88      0.82        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.94      0.97        18

    accuracy                           0.86        92
   macro avg       0.82      0.81      0.80        92
weighted avg       0.84      0.86      0.84        92
```



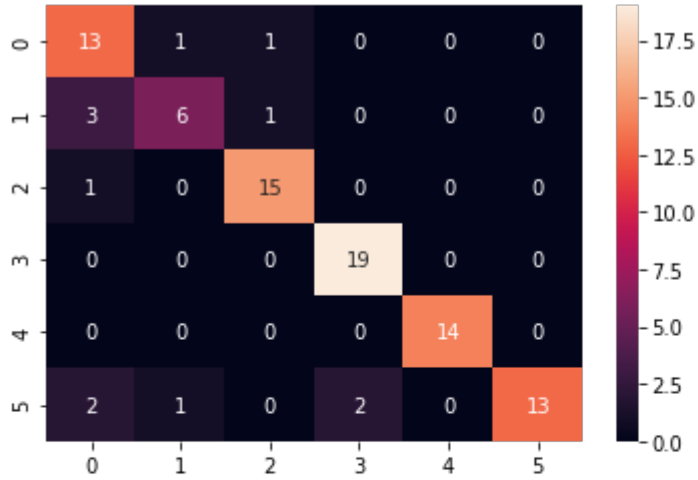
3) Support Vector Machines

```
In [184]: from sklearn.svm import SVC
model=SVC()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [185]:
```
#evalution
acc=accuracy_score(ytest,ypred)
print("Accuracy is: ",acc)
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is:  0.8695652173913043
              precision    recall  f1-score   support

           1       0.68      0.87      0.76        15
           2       0.75      0.60      0.67        10
           3       0.88      0.94      0.91        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.87        92
   macro avg       0.87      0.85      0.85        92
weighted avg       0.88      0.87      0.87        92
```

Out[185]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24b8aee90>



hyper parameter tunning of SVM

In [186]:

```
#model
model=SVC()
#parameters
kernel=['linear','poly','rbf','sigmoid']
C=[1,0.1,0.01,0.001]
gamma=['scale', 'auto']
#grid
grid=dict(kernel=kernel,C=C,gamma=gamma)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3,random_state=1)
from sklearn.model_selection import GridSearchCV
grid_cv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring="
accuracy")
#result
res=grid_cv.fit(xtrain,ytrain)
print(res.best_params_)
print(res.best_score_)
```

```
{'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
0.8343353627600203
```

Retraining the SVM model on best parameters

In [187]:

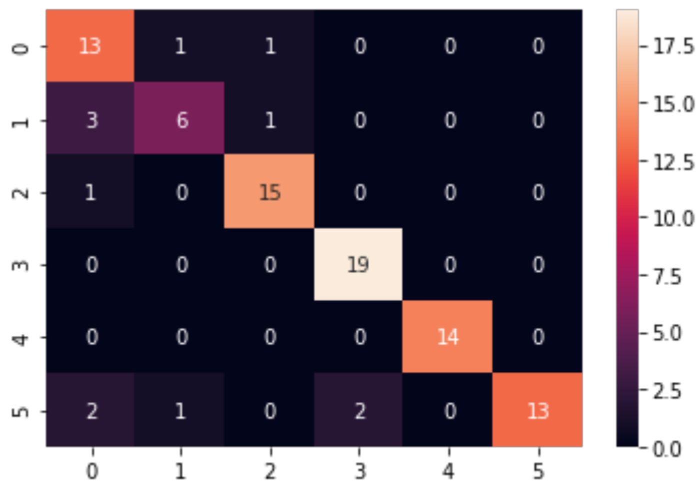
```
# For best parameter
from sklearn.svm import SVC
model=SVC(C= 1, gamma='auto', kernel='rbf')
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [188]:

```
#Model Evaluation
SVM_pre,SVM_recall,SVM_fsc,support=score(ytest,ypred,average='macro
')
SVM_acc = accuracy_score(ytest,ypred)
acc=accuracy_score(ytest,ypred)
print("Accuracy is: ",acc)
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is:  0.8695652173913043
              precision    recall  f1-score   support

           1       0.68      0.87      0.76        15
           2       0.75      0.60      0.67        10
           3       0.88      0.94      0.91        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.87        92
   macro avg       0.87      0.85      0.85        92
weighted avg       0.88      0.87      0.87        92
```

Out[188]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24b8d7f90>



4) KNN

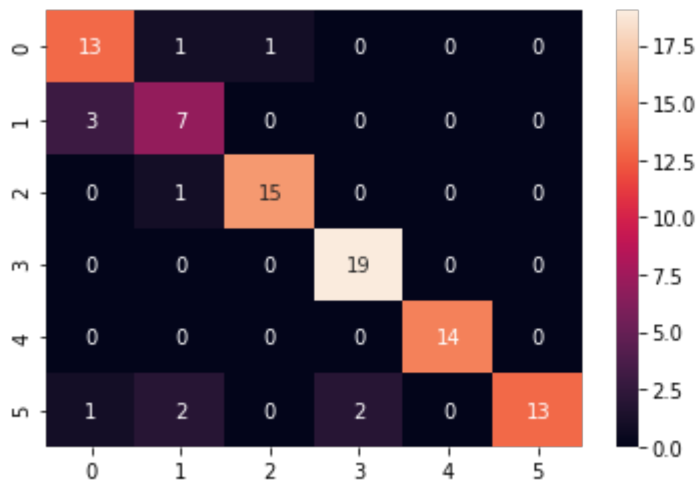
In [189]:

```
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier(n_neighbors=5)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [190]:
```
#Model Evaluation
acc=accuracy_score(ytest,ypred)
print("Accuracy is: ",acc)
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is:  0.8804347826086957
              precision    recall  f1-score   support

           1       0.76      0.87      0.81        15
           2       0.64      0.70      0.67        10
           3       0.94      0.94      0.94        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.88        92
   macro avg       0.87      0.87      0.87        92
weighted avg       0.89      0.88      0.88        92
```

Out[190]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24b867450>



hyper parameter tunning of KNN

In [191]:

```
#model
model=KNeighborsClassifier()
#parameter grid
#1. n_neighbors
#2.weights
#3.Metric
n_neighbors=range(1,31)
weights =['uniform', 'distance']
metric=["minkowski","euclidean","manhattan"]
grid=dict(n_neighbors=n_neighbors,weights=weights,metric=metric)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3,random_state=1)
#GridSearchCV
from sklearn.model_selection import GridSearchCV
grid_cv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring="
accuracy")
res=grid_cv.fit(xtrain,ytrain)
print(res.best_params_)
print(res.best_score_)
```

```
{'metric': 'manhattan', 'n_neighbors': 1, 'weights': 'uniform'}
0.8709538305428717
```

Retraining the KNN model on best parameters

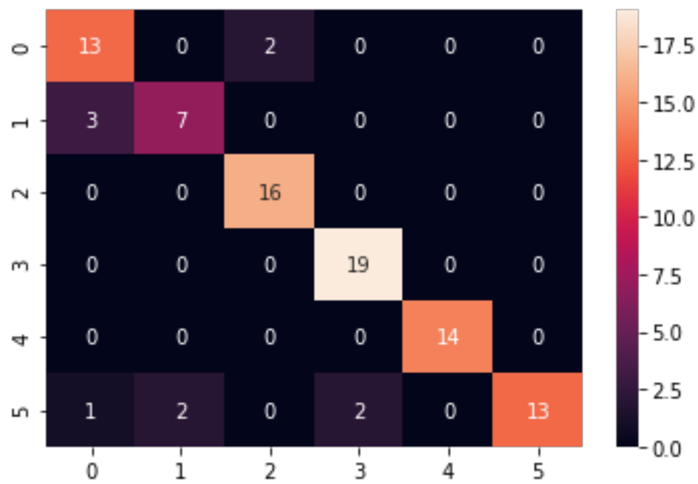
In [194]:

```
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier(n_neighbors=1,metric='manhattan',weights=
'uniform')
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [195]:
```
#Model Evaluation
KNN_pre,KNN_recall,KNN_fsc,support=score(ytest,ypred,average='macro
')
KNN_acc = accuracy_score(ytest,ypred)
acc=accuracy_score(ytest,ypred)
print("Accuracy is: ",acc)
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is:  0.8913043478260869
              precision    recall  f1-score   support

           1       0.76      0.87      0.81        15
           2       0.78      0.70      0.74        10
           3       0.89      1.00      0.94        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.89        92
   macro avg       0.89      0.88      0.88        92
weighted avg       0.90      0.89      0.89        92
```

Out[195]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24db3c990>



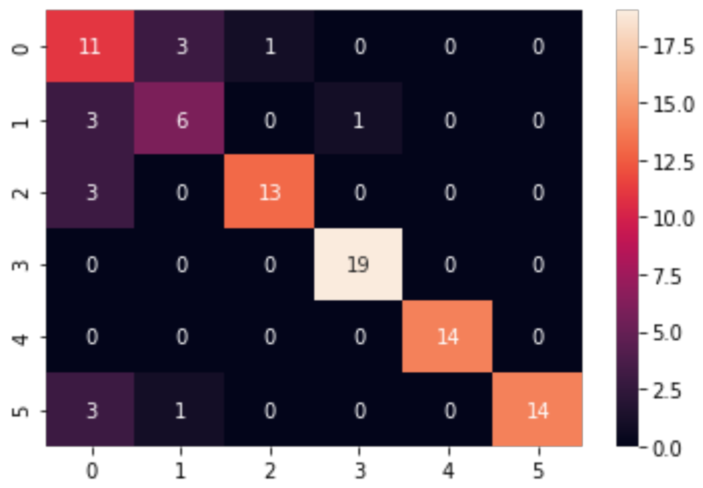
5)Decision Tree

In [196]:
```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [197]:
```
# evaluation
acc=accuracy_score(ytest,ypred)
print("Accuracy is: ",acc)
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is:  0.8369565217391305
              precision    recall  f1-score   support

           1       0.55      0.73      0.63        15
           2       0.60      0.60      0.60        10
           3       0.93      0.81      0.87        16
           5       0.95      1.00      0.97        19
           6       1.00      1.00      1.00        14
           7       1.00      0.78      0.88        18

    accuracy                           0.84        92
   macro avg       0.84      0.82      0.82        92
weighted avg       0.86      0.84      0.84        92
```

Out[197]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb24cdc5210>
```



hyper parameter tunning of DecisionTree

In [198]:
```
#model
model=DecisionTreeClassifier()
criterion =["gini", "entropy"]
splitter =["best", "random"]
max_features = ["auto", "sqrt", "log2"]
max_depth=range(1,11)
#parameters
grid=dict(criterion=criterion,splitter=splitter,max_depth=max_depth,
max_features=max_features)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=10,n_repeats=3,random_state=1)
#Grid Search CV
from sklearn.model_selection import GridSearchCV
grid_cv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring="
accuracy")
res=grid_cv.fit(xtrain,ytrain)
print(res.best_params_)
print(res.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'spl
itter': 'best'}
0.8433183183183185
```

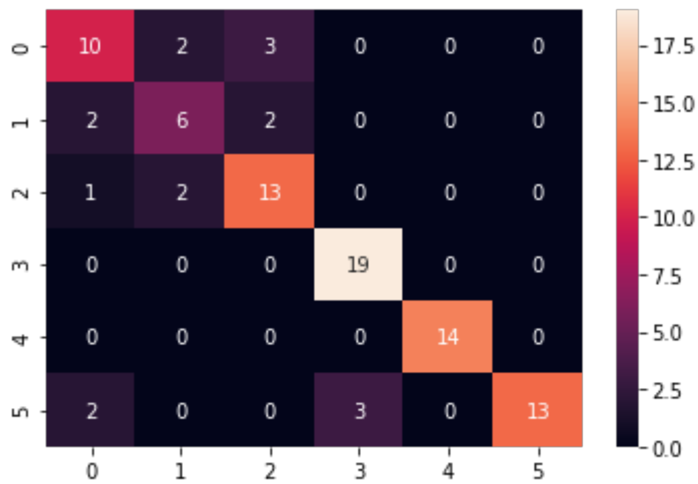
Retraining the model on best parameter

In [204]:
```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier(criterion='gini', max_depth=8, max_feat
ures='log2', splitter='best')
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [205]:
```
#Model Evaluation
DT_pre,DT_recall,DT_fsc,support=score(ytest,ypred,average='macro')
DT_acc = accuracy_score(ytest,ypred)
acc=accuracy_score(ytest,ypred)
print("Accuracy is: ",acc)
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is:  0.8152173913043478
              precision    recall  f1-score   support

           1       0.67      0.67      0.67        15
           2       0.60      0.60      0.60        10
           3       0.72      0.81      0.76        16
           5       0.86      1.00      0.93        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.82        92
   macro avg       0.81      0.80      0.80        92
weighted avg       0.83      0.82      0.81        92
```

Out[205]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fb24b73fd50>`



6) Ensemble Learning

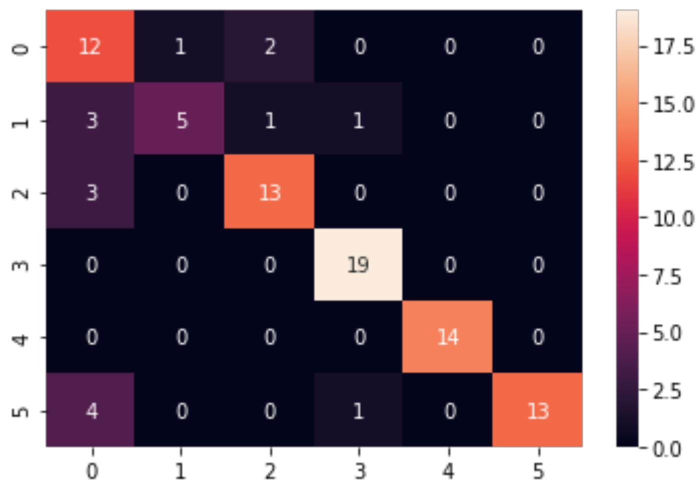
1) Bagging Metaestimator

In [206]:
```
from sklearn.ensemble import BaggingClassifier
model=BaggingClassifier()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [207]:
```python
#Model Evaluation
print("accuracy is :",accuracy_score(ytest,ypred))
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
accuracy is : 0.8260869565217391
              precision    recall  f1-score   support

           1       0.55      0.80      0.65        15
           2       0.83      0.50      0.62        10
           3       0.81      0.81      0.81        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.83        92
   macro avg       0.85      0.81      0.81        92
weighted avg       0.86      0.83      0.83        92
```

Out[207]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24c81c190>



Hyper parameter tunning of metaestimator

In [208]:
```
#model
model=BaggingClassifier()
n_estimators =[10,50,100,1000]
#grid
grid=dict(n_estimators=n_estimators)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3,random_state=1)
#GridSearchCV
from sklearn.model_selection import GridSearchCV
grid_cv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring='
accuracy')
#results
res=grid_cv.fit(xtrain,ytrain)
print("best parameters are :",res.best_params_)
print("best accuracy is :",res.best_score_)
```

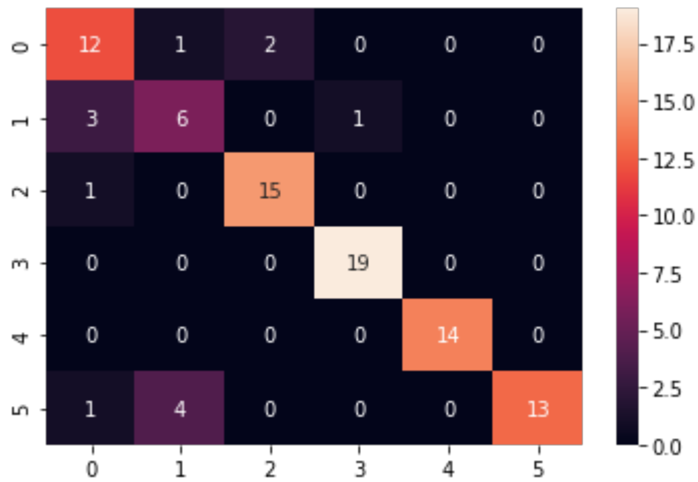
```
best parameters are : {'n_estimators': 100}
best accuracy is : 0.8737316083206494
```

Retraining the model on best parameters

In [211]:
```
from sklearn.ensemble import BaggingClassifier
model=BaggingClassifier( n_estimators= 100)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [212]:
```
#Model Evaluation
BM_pre,BM_recall,BM_fsc,support=score(ytest,ypred,average='macro')
BM_acc = accuracy_score(ytest,ypred)
from sklearn.metrics import accuracy_score,confusion_matrix,classifi
cation_report
print("accuracy is :",accuracy_score(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
accuracy is : 0.8586956521739131
              precision    recall  f1-score   support

           1       0.71      0.80      0.75        15
           2       0.55      0.60      0.57        10
           3       0.88      0.94      0.91        16
           5       0.95      1.00      0.97        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.86        92
   macro avg       0.85      0.84      0.84        92
weighted avg       0.87      0.86      0.86        92
```



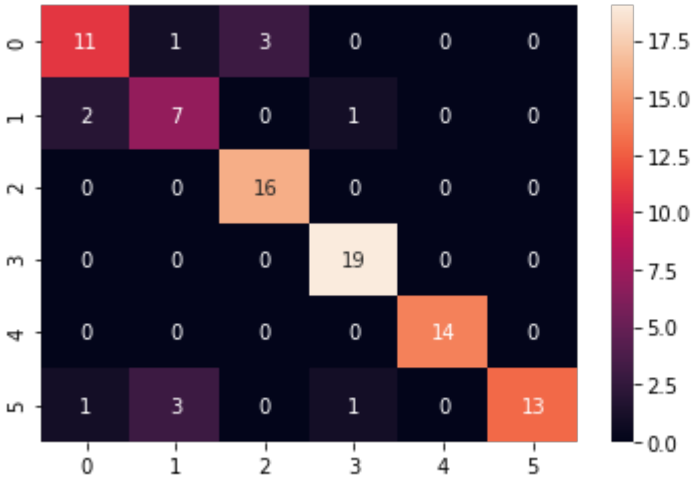
ii) RandomForest

In [213]:
```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [214]:
```python
#Model Evaluation
print("accuracy is :",accuracy_score(ytest,ypred))
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
accuracy is : 0.8695652173913043
              precision    recall  f1-score   support

           1       0.79      0.73      0.76        15
           2       0.64      0.70      0.67        10
           3       0.84      1.00      0.91        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.87        92
   macro avg       0.86      0.86      0.85        92
weighted avg       0.88      0.87      0.87        92
```

Out[214]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24b15e710>



hyper parameter tuning of the random forest

In [215]:
```
#model
model=RandomForestClassifier()
n_estimators =[10,50,100,1000]
criterion =["gini", "entropy"]
max_features =["auto", "sqrt", "log2"]
#grid
grid=dict(n_estimators=n_estimators,criterion=criterion,max_features
=max_features)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3,random_state=1)
#GridSearchCV
from sklearn.model_selection import GridSearchCV
grid_cv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring='
accuracy')
#results
res=grid_cv.fit(xtrain,ytrain)
print("best parameters are :",res.best_params_)
print("best accuracy is :",res.best_score_)
```

```
best parameters are : {'criterion': 'entropy', 'max_features': 'sq
rt', 'n_estimators': 50}
best accuracy is : 0.8993658041603247
```

Retraining the data on best parameters:

In [218]:
```
model=RandomForestClassifier(criterion='entropy',max_features='sqrt
',n_estimators=50)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

```
In [219]: #evaluation
RF_pre,RF_recall,RF_fsc,support=score(ytest,ypred,average='macro')
RF_acc = accuracy_score(ytest,ypred)
acc=accuracy_score(ytest,ypred)
print("Accuracy is: ",acc )
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is:  0.8695652173913043
              precision    recall  f1-score   support

           1       0.75      0.80      0.77        15
           2       0.67      0.80      0.73        10
           3       0.88      0.88      0.88        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.87        92
   macro avg       0.87      0.87      0.86        92
weighted avg       0.88      0.87      0.87        92
```

```
Out[219]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24b283110>
```

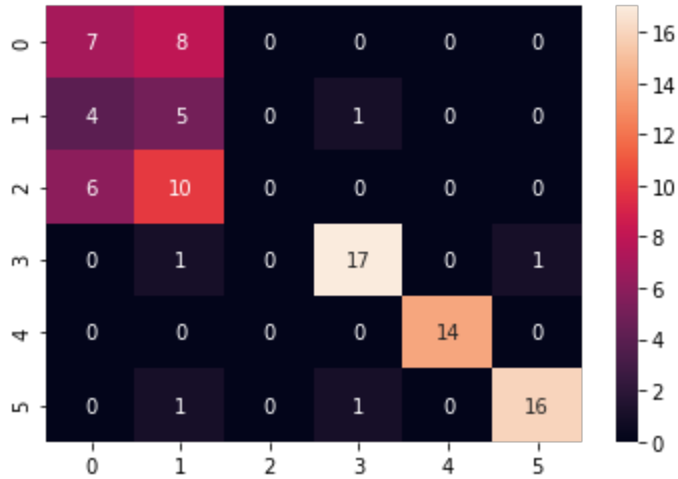
7) Boosting

i) Adaboost

```
In [220]: from sklearn.ensemble import AdaBoostClassifier
model=AdaBoostClassifier()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [221]:
```python
from sklearn.metrics import classification_report, accuracy_score,co
nfusion_matrix
print("Accuracy is :",accuracy_score(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
Accuracy is : 0.6413043478260869
              precision    recall  f1-score   support

           1       0.41      0.47      0.44        15
           2       0.20      0.50      0.29        10
           3       0.00      0.00      0.00        16
           5       0.89      0.89      0.89        19
           6       1.00      1.00      1.00        14
           7       0.94      0.89      0.91        18

    accuracy                           0.64        92
   macro avg       0.57      0.63      0.59        92
weighted avg       0.61      0.64      0.62        92
```



In [222]:
```
#model
model=AdaBoostClassifier()
n_estimators =[10,50,100,1000]
learning_rate =[0.1,1]
algorithm =["SAMME", "SAMME.R"]
#grid
grid=dict(n_estimators=n_estimators,learning_rate=learning_rate,algo
rithm=algorithm)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3,random_state=1)
#GridSearchCV
from sklearn.model_selection import GridSearchCV
grid_cv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring='
accuracy')
#results
res=grid_cv.fit(xtrain,ytrain)
print("best parameters are :",res.best_params_)
print("best accuracy is :",res.best_score_)
```

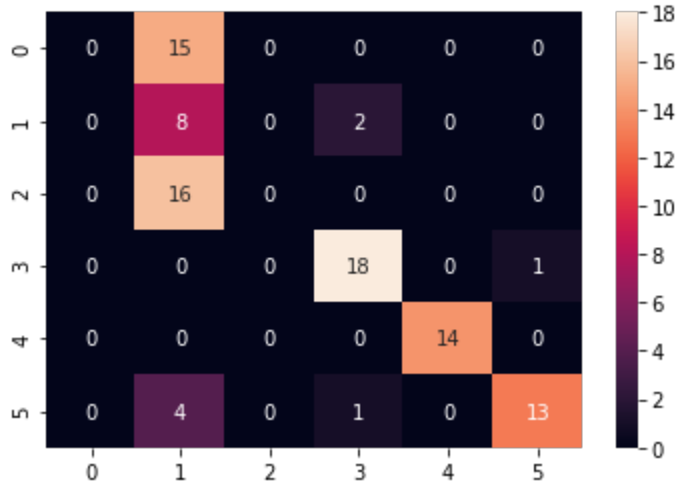
```
best parameters are : {'algorithm': 'SAMME.R', 'learning_rate': 0.
1, 'n_estimators': 50}
best accuracy is : 0.6291476407914763
```

Retraining the Adaboost model on best parameters

In [225]:
```
from sklearn.ensemble import AdaBoostClassifier
model=AdaBoostClassifier(algorithm= 'SAMME.R',learning_rate=0.1, n_e
stimators= 50)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [226]:
```
Ada_pre,Ada_recall,Ada_fsc,support=score(ytest,ypred,average='macro
')
Ada_acc = accuracy_score(ytest,ypred)
print("Accuracy is :",accuracy_score(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
Accuracy is : 0.5760869565217391
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        15
           2       0.19      0.80      0.30        10
           3       0.00      0.00      0.00        16
           5       0.86      0.95      0.90        19
           6       1.00      1.00      1.00        14
           7       0.93      0.72      0.81        18

    accuracy                           0.58        92
   macro avg       0.50      0.58      0.50        92
weighted avg       0.53      0.58      0.53        92
```

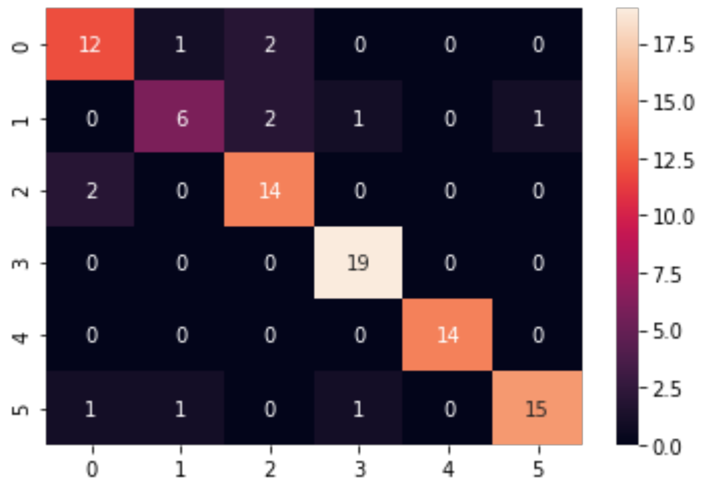


ii) GradientBoost

In [227]:
```
from sklearn.ensemble import GradientBoostingClassifier
model=GradientBoostingClassifier(n_estimators=100)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [228]:
```
#Evaluation
GradBoost_pre,GradBoost_recall,GradBoost_fsc,support=score(ytest,ypr
ed,average='macro')
GradBoost_acc = accuracy_score(ytest,ypred)
print("Accuracy is :",accuracy_score(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
Accuracy is : 0.8695652173913043
              precision    recall  f1-score   support

           1       0.80      0.80      0.80        15
           2       0.75      0.60      0.67        10
           3       0.78      0.88      0.82        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       0.94      0.83      0.88        18

    accuracy                           0.87        92
   macro avg       0.86      0.85      0.85        92
weighted avg       0.87      0.87      0.87        92
```

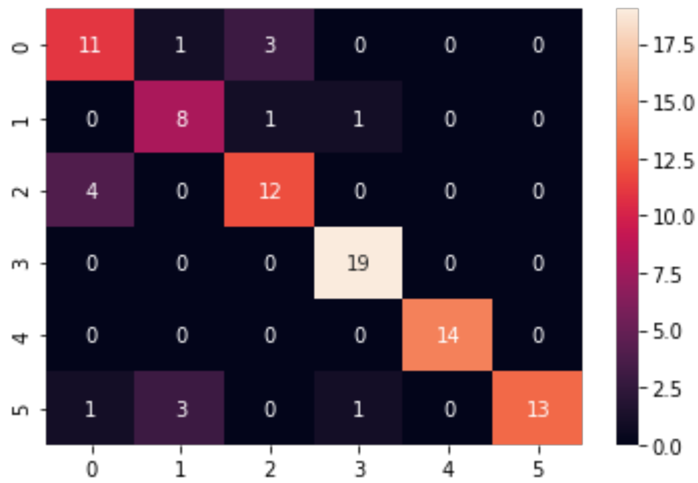


iii) XGBoost

In [229]:
```
from xgboost import XGBClassifier
model=XGBClassifier()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [230]:
```
print("Accuracy is :",accuracy_score(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
Accuracy is : 0.8369565217391305
              precision    recall  f1-score   support

           1       0.69      0.73      0.71        15
           2       0.67      0.80      0.73        10
           3       0.75      0.75      0.75        16
           5       0.90      1.00      0.95        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.84        92
   macro avg       0.83      0.83      0.83        92
weighted avg       0.85      0.84      0.84        92
```



In [231]:
```
#model
model=XGBClassifier()
n_estimators =[10,50,100]
learning_rate =[0.1,1]
#grid
grid=dict(n_estimators=n_estimators,learning_rate=learning_rate)
#cv
from sklearn.model_selection import RepeatedStratifiedKFold
cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3,random_state=1)
#GridSearchCV
from sklearn.model_selection import GridSearchCV
grid_cv=GridSearchCV(estimator=model,param_grid=grid,cv=cv,scoring='
accuracy')
#results
res=grid_cv.fit(xtrain,ytrain)
print("best parameters are :",res.best_params_)
print("best accuracy is :",res.best_score_)
```

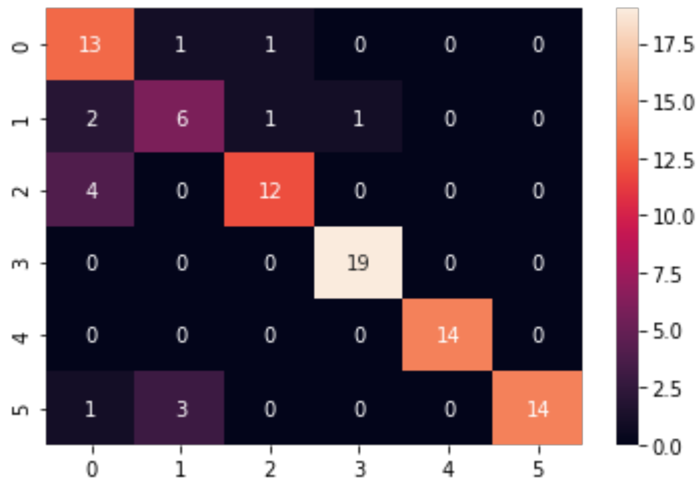
```
best parameters are : {'learning_rate': 1, 'n_estimators': 100}
best accuracy is : 0.8910451547437849
```

Retraining the XGBoost regression model on best parameters

In [234]:
```
from xgboost import XGBClassifier
model=XGBClassifier(learning_rate= 1,n_estimators= 100)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [235]:
```
#Evaluation
XGB_pre,XGB_recall,XGB_fsc,support=score(ytest,ypred,average='macro
')
XGB_acc = accuracy_score(ytest,ypred)
print("Accuracy is :",accuracy_score(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
Accuracy is : 0.8478260869565217
              precision    recall  f1-score   support

           1       0.65      0.87      0.74        15
           2       0.60      0.60      0.60        10
           3       0.86      0.75      0.80        16
           5       0.95      1.00      0.97        19
           6       1.00      1.00      1.00        14
           7       1.00      0.78      0.88        18

    accuracy                           0.85        92
   macro avg       0.84      0.83      0.83        92
weighted avg       0.86      0.85      0.85        92
```



7)Voting

In [236]:

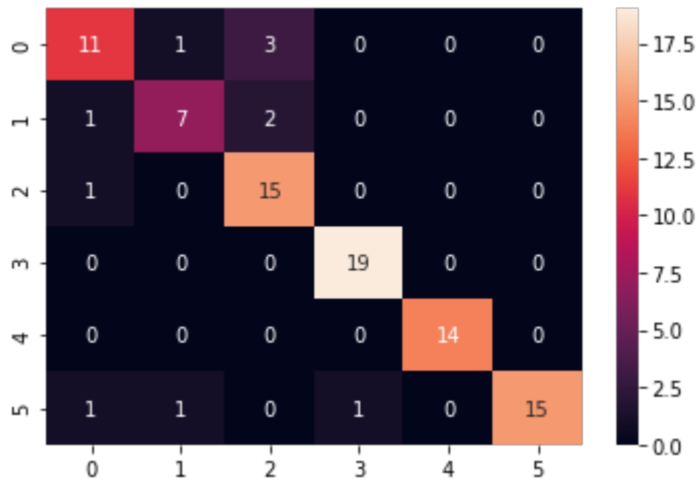
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifi
er,GradientBoostingClassifier
models=[
    ("lr",LogisticRegression()),
    ("knn",KNeighborsClassifier(n_neighbors=5)),
    ("GNB",GaussianNB()),
    ("RF",RandomForestClassifier(n_estimators=50)),
    ('ABC',AdaBoostClassifier(n_estimators=50)),
    ('GBC',GradientBoostingClassifier(n_estimators=50)),
    ('SVM',SVC(C=0.1,probability=True))
]
```

In [238]:

```
from sklearn.ensemble import VotingClassifier
model=VotingClassifier(estimators=models,voting="soft")
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [239]:
```python
#Evaluation
voting_pre,voting_recall,voting_fsc,support=score(ytest,ypred,averag
e='macro')
voting_acc = accuracy_score(ytest,ypred)
print("Accuracy is :",accuracy_score(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
print(classification_report(ytest,ypred))
```

```
Accuracy is : 0.8804347826086957
              precision    recall  f1-score   support

           1       0.79      0.73      0.76        15
           2       0.78      0.70      0.74        10
           3       0.75      0.94      0.83        16
           5       0.95      1.00      0.97        19
           6       1.00      1.00      1.00        14
           7       1.00      0.83      0.91        18

    accuracy                           0.88        92
   macro avg       0.88      0.87      0.87        92
weighted avg       0.89      0.88      0.88        92
```



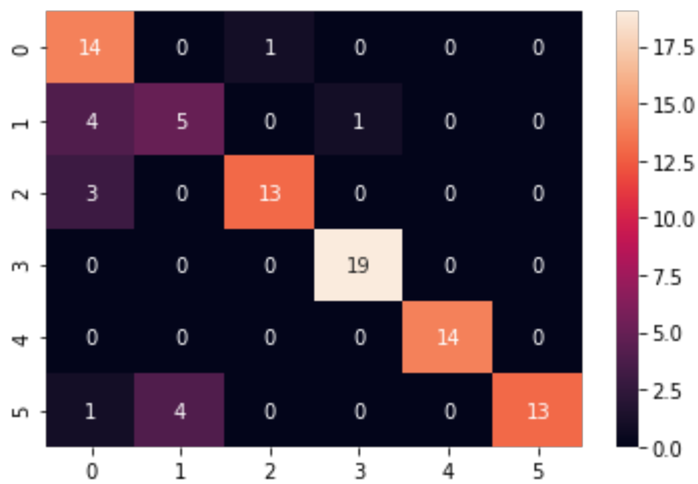
8) Stacking

In [241]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
#Base models
base_models=[ ('knn',KNeighborsClassifier(n_neighbors=5)),
              ('svm',SVC(C=0.1,kernel='linear')),
              ('DT',DecisionTreeClassifier())
]
#Final Model
from sklearn.linear_model import LogisticRegression
final_model=LogisticRegression()
#Stacking Classifier
from sklearn.ensemble import StackingClassifier
model=StackingClassifier(estimators=base_models,final_estimator=fina
l_model)
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)
```

In [242]:
```
#Evaluation
stacking_pre,stacking_recall,stacking_fsc,support=score(ytest,ypred,
average='macro')
stacking_acc = accuracy_score(ytest,ypred)
print("Accuracy is ",accuracy_score(ytest,ypred))
print(classification_report(ytest,ypred))
cm=confusion_matrix(ytest,ypred)
sns.heatmap(cm,annot=True)
```

```
Accuracy is  0.8478260869565217
              precision    recall  f1-score   support

           1       0.64      0.93      0.76        15
           2       0.56      0.50      0.53        10
           3       0.93      0.81      0.87        16
           5       0.95      1.00      0.97        19
           6       1.00      1.00      1.00        14
           7       1.00      0.72      0.84        18

    accuracy                           0.85        92
   macro avg       0.85      0.83      0.83        92
weighted avg       0.87      0.85      0.85        92
```

Out[242]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb24a9b2c90>



Evaluation

In [246]:
```
models = ['Logistic Regression','SVM','KNN','Decision Tree','Bagging
metaestimator','Random forest','AdaBoost','Gradient Boost','XGBoost
','Voting classifier','stacking Classifier']
accuracy = [lr_acc,SVM_acc,KNN_acc,DT_acc,BM_acc,RF_acc,Ada_acc,Grad
Boost_acc,XGB_acc,voting_acc,stacking_acc]
precision=[lr_pre,SVM_pre,KNN_pre,DT_pre,BM_pre,RF_pre,Ada_pre,GradB
oost_pre,XGB_pre,voting_pre,stacking_pre]
recall = [lr_recall,SVM_recall,KNN_recall,DT_recall,BM_recall,RF_rec
all,Ada_recall,GradBoost_recall,XGB_recall,voting_recall,stacking_re
call]
fscore = [lr_fsc,SVM_fsc,KNN_fsc,DT_fsc,BM_fsc,RF_fsc,Ada_fsc,GradBo
ost_fsc,XGB_fsc,voting_fsc,stacking_fsc]
Evaluation = pd.DataFrame({'No.':[x+1 for x in range(len(models))],'
Model':models,'Accuracy':accuracy,'Precision':precision,'Recall':rec
all,'fscore':fscore})
```

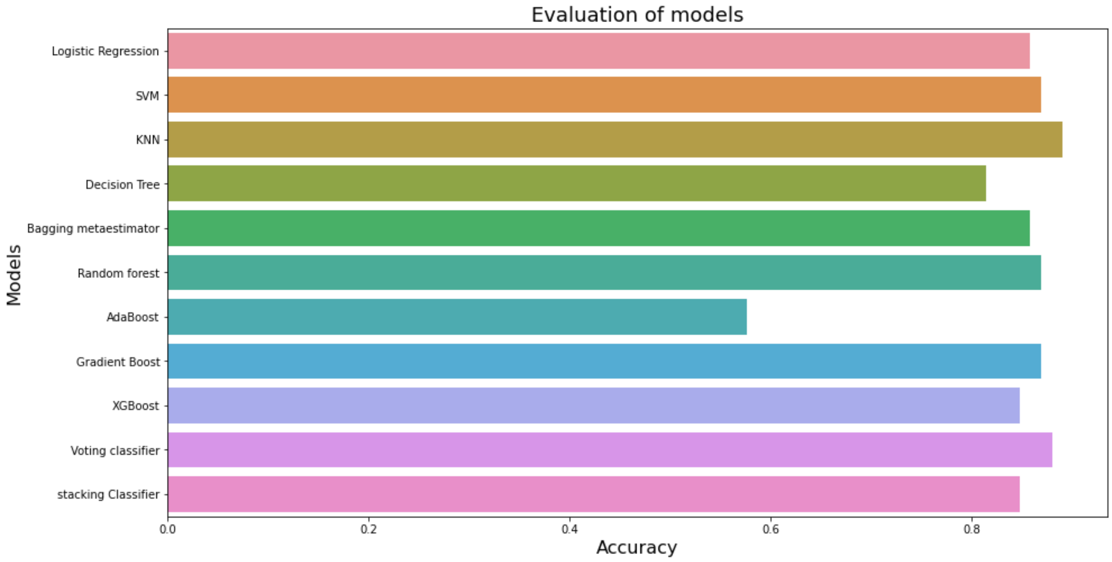
In [247]:
```
Evaluation.style.highlight_max(subset = ['Accuracy'],color = 'lightg
reen')
```

Out[247]:

	No.	Model	Accuracy	Precision	Recall	fscore
0	1	Logistic Regression	0.858696	0.817460	0.814352	0.803092
1	2	SVM	0.869565	0.870221	0.854398	0.854862
2	3	KNN	0.891304	0.889356	0.881481	0.879871
3	4	Decision Tree	0.815217	0.808754	0.800231	0.799485
4	5	Bagging metaestimator	0.858696	0.847282	0.843287	0.840598
5	6	Random forest	0.869565	0.866071	0.866204	0.860863
6	7	AdaBoost	0.576087	0.495293	0.578265	0.502398
7	8	Gradient Boost	0.869565	0.861673	0.851389	0.853758
8	9	XGBoost	0.847826	0.842857	0.832407	0.832036
9	10	Voting classifier	0.880435	0.877249	0.867361	0.868708
10	11	stacking Classifier	0.847826	0.845082	0.828009	0.827135

In [248]:
```
plt.figure(figsize = (15,8))
sns.barplot(x=accuracy, y=models)
plt.xlabel('Accuracy',fontdict={'fontsize':16})
plt.ylabel('Models',fontdict={'fontsize':16})
plt.title('Evaluation of models',fontdict={'fontsize':18})
```

Out[248]:
```
Text(0.5, 1.0, 'Evaluation of models')
```



We can conclude that KNN after oversampling showing high accuracy which is around 0.89 where and multinomial Adaboost is performing poor on the given data.