In [19]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sl
import seaborn as sns
```

In [20]:

```
data=pd.read_csv("C:/Users/Manju Sah/Downloads/AQI dataset 2020-22.csv")
```

In [21]:

```
data
```

Out[21]:

	City	Date	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI	AQI_Bucket
0	Indore	01-01-2020	231.0	100.0	68.0	6.0	6.0	42.0	33.0	231.0	Poor
1	Indore	02-01-2020	191.0	121.0	59.0	6.0	6.0	52.0	30.0	191.0	Moderate
2	Indore	03-01-2020	120.0	91.0	48.0	5.0	9.0	36.0	32.0	120.0	Moderate
3	Indore	04-01-2020	234.0	181.0	80.0	6.0	13.0	23.0	38.0	234.0	Poor
4	Indore	05-01-2020	118.0	93.0	67.0	5.0	8.0	22.0	38.0	118.0	Moderate
...	...	...	...	...	...	...	...	...	...	...	...
2187	Delhi	27-12-2022	376.0	308.0	93.0	11.0	14.0	103.0	12.0	376.0	Very Poor
2188	Delhi	28-12-2022	331.0	265.0	120.0	11.0	9.0	89.0	15.0	331.0	Very Poor
2189	Delhi	29-12-2022	338.0	317.0	169.0	12.0	17.0	97.0	17.0	338.0	Very Poor
2190	Delhi	30-12-2022	385.0	298.0	124.0	14.0	17.0	92.0	25.0	385.0	Very Poor

In [22]:

```
data.head()
```

Out[22]:

	City	Date	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI	AQI_Bucket
0	Indore	01-01-2020	231.0	100.0	68.0	6.0	6.0	42.0	33.0	231.0	Poor
1	Indore	02-01-2020	191.0	121.0	59.0	6.0	6.0	52.0	30.0	191.0	Moderate
2	Indore	03-01-2020	120.0	91.0	48.0	5.0	9.0	36.0	32.0	120.0	Moderate
3	Indore	04-01-2020	234.0	181.0	80.0	6.0	13.0	23.0	38.0	234.0	Poor
4	Indore	05-01-2020	118.0	93.0	67.0	5.0	8.0	22.0	38.0	118.0	Moderate

In [23]:

```
data.tail()
```

Out[23]:

	City	Date	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI	AQI_Bucket
2187	Delhi	27-12-2022	376.0	308.0	93.0	11.0	14.0	103.0	12.0	376.0	Very Poor
2188	Delhi	28-12-2022	331.0	265.0	120.0	11.0	9.0	89.0	15.0	331.0	Very Poor
2189	Delhi	29-12-2022	338.0	317.0	169.0	12.0	17.0	97.0	17.0	338.0	Very Poor
2190	Delhi	30-12-2022	385.0	298.0	124.0	14.0	17.0	92.0	25.0	385.0	Very Poor
2191	Delhi	31-12-2022	366.0	332.0	NaN	NaN	22.0	90.0	11.0	366.0	Very Poor

In [24]:

```
data.shape
```

Out[24]:

```
(2192, 11)
```

In [25]:

```
data.size
```

Out[25]:

```
24112
```

In [26]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2192 entries, 0 to 2191
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   City        2192 non-null   object 
 1   Date        2192 non-null   object 
 2   PM2.5       1984 non-null   float64
 3   PM10        2000 non-null   float64
 4   NO2         1996 non-null   float64
 5   NH3         1957 non-null   float64
 6   SO2         2008 non-null   float64
 7   CO          2031 non-null   float64
 8   O3          2122 non-null   float64
 9   AQI         2039 non-null   float64
 10  AQI_Bucket  2041 non-null   object 
dtypes: float64(8), object(3)
memory usage: 188.5+ KB
```

In [27]:

```
data.describe()
```

Out[27]:

	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI
count	1984.000000	2000.000000	1996.000000	1957.000000	2008.000000	2031.000000	2122.000000	2039.000000
mean	144.831653	159.569000	73.923347	6.990802	19.270916	62.087149	49.987747	175.290338
std	115.918024	105.773339	38.680785	3.862133	10.268169	31.849442	34.312646	111.701973
min	13.000000	9.000000	4.000000	1.000000	2.000000	6.000000	3.000000	32.000000
25%	52.000000	83.000000	44.000000	4.000000	13.000000	36.000000	29.000000	89.000000
50%	104.000000	127.000000	66.000000	6.000000	17.000000	55.000000	43.000000	138.000000
75%	211.000000	215.000000	96.000000	9.000000	24.000000	87.000000	63.000000	244.500000
max	485.000000	490.000000	332.000000	39.000000	140.000000	418.000000	409.000000	490.000000

In [28]:

```
data.columns
```

Out[28]:

```
Index(['City', 'Date', 'PM2.5', 'PM10', 'NO2', 'NH3', 'SO2', 'CO', 'O3', 'AQI',
       'AQI_Bucket'],
      dtype='object')
```

In [29]:

```
data.isnull().sum()
```

Out[29]:

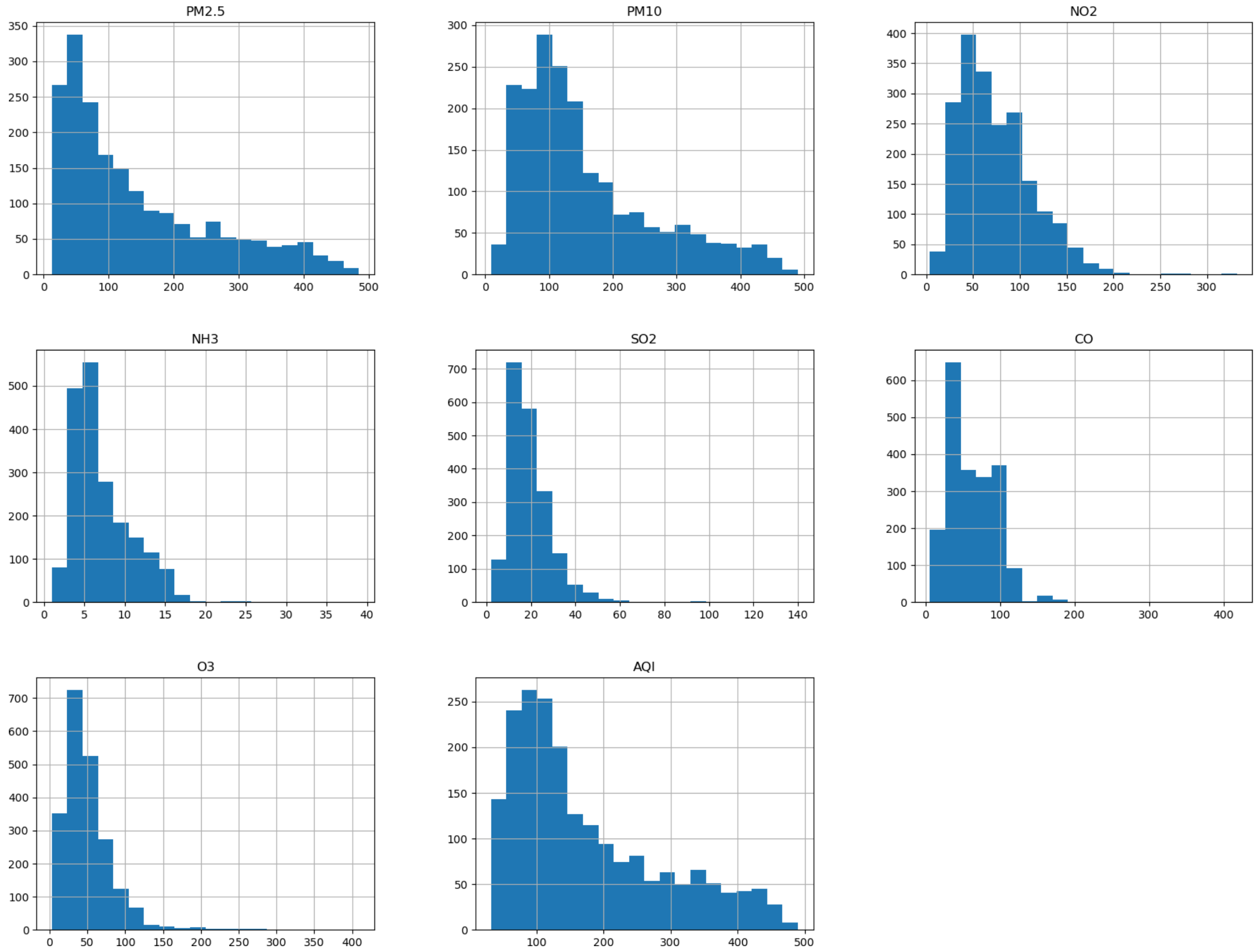
```
City            0
Date            0
PM2.5         208
PM10          192
NO2           196
NH3           235
SO2           184
CO            161
O3             70
AQI           153
AQI_Bucket    151
dtype: int64
```

In [30]:

```
data.hist(bins=20,figsize=(20,15))
```

Out[30]:

```
array([[<AxesSubplot:title={'center':'PM2.5'}>,
        <AxesSubplot:title={'center':'PM10'}>,
        <AxesSubplot:title={'center':'NO2'}>],
       [<AxesSubplot:title={'center':'NH3'}>,
        <AxesSubplot:title={'center':'SO2'}>,
        <AxesSubplot:title={'center':'CO'}>],
       [<AxesSubplot:title={'center':'O3'}>,
        <AxesSubplot:title={'center':'AQI'}>, <AxesSubplot:>]],
      dtype=object)
```



Scatter Plot

In [31]:

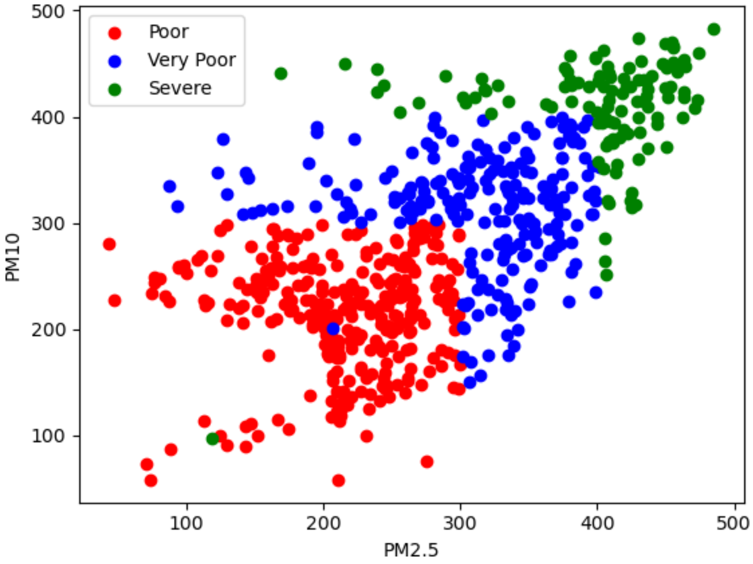
```
colors=['red','blue','green']
AQI_Bucket=['Poor','Very Poor','Severe']
```

In [32]:

```
for i in range(3):
    x=data[data['AQI_Bucket'] == AQI_Bucket[i]]
    plt.scatter(x["PM2.5"],x["PM10"], c=colors[i], label=AQI_Bucket[i])
plt.xlabel("PM2.5")
plt.ylabel("PM10")
plt.legend()
```

Out[32]:

```
<matplotlib.legend.Legend at 0x2648aaa43d0>
```



In [33]:

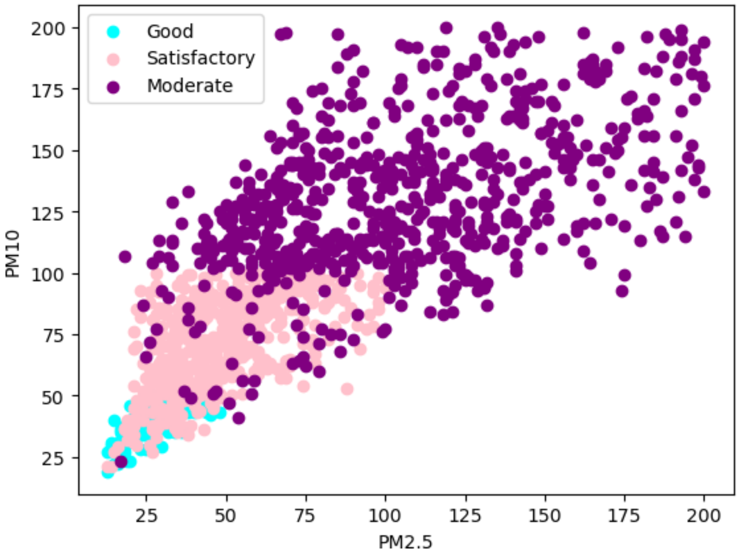
```
colors=['aqua','pink','purple']
AQI_Bucket=['Good','Satisfactory','Moderate']
```

In [34]:

```
for i in range(3):
    x=data[data['AQI_Bucket'] == AQI_Bucket[i]]
    plt.scatter(x["PM2.5"],x["PM10"], c=colors[i], label=AQI_Bucket[i])
plt.xlabel("PM2.5")
plt.ylabel("PM10")
plt.legend()
```

Out[34]:

```
<matplotlib.legend.Legend at 0x2648af72f10>
```

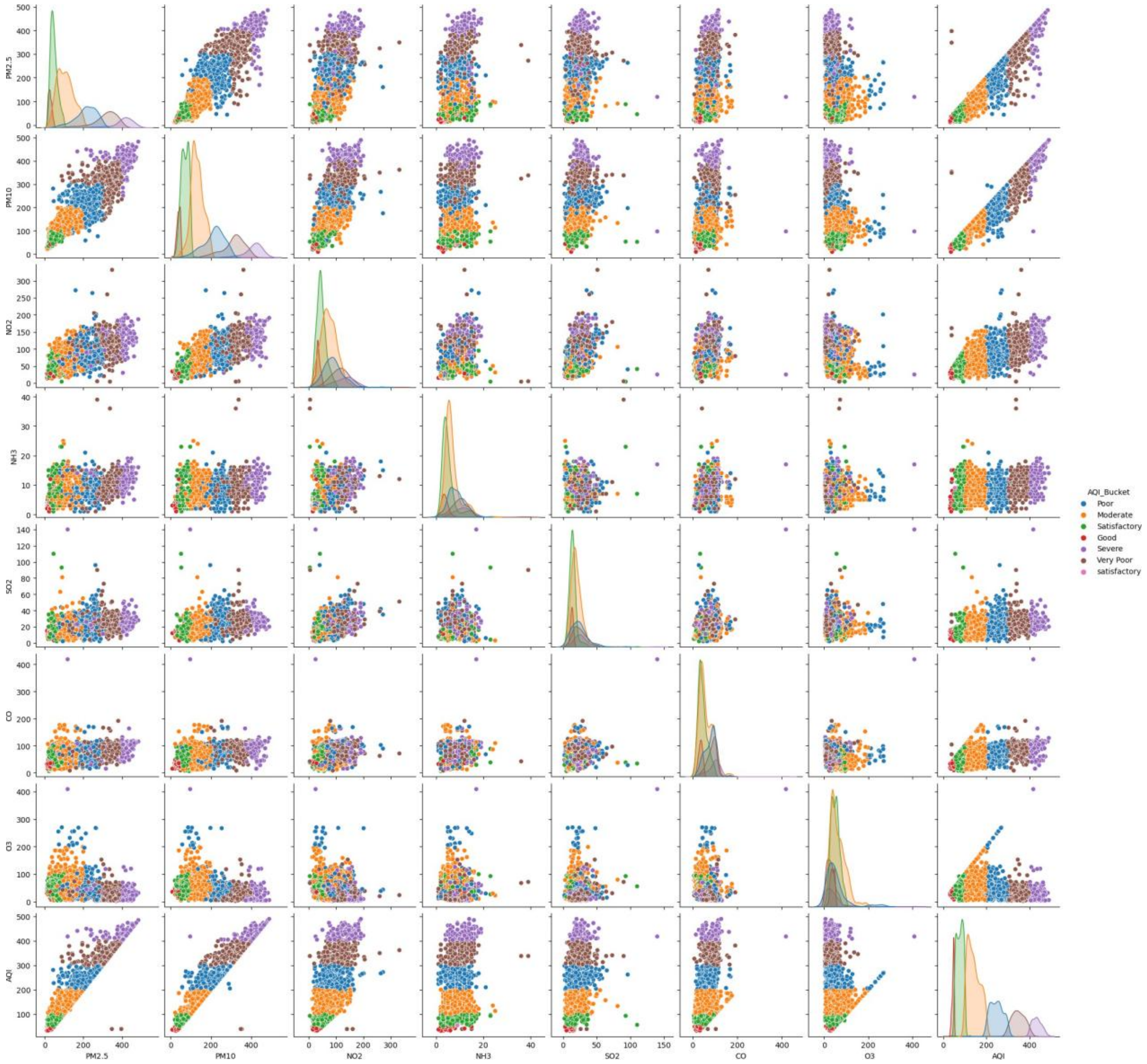


In [35]:

```
import seaborn as sns
sns.pairplot(data, hue='AQI_Bucket')
```

Out[35]:

```
<seaborn.axisgrid.PairGrid at 0x2648af7e340>
```



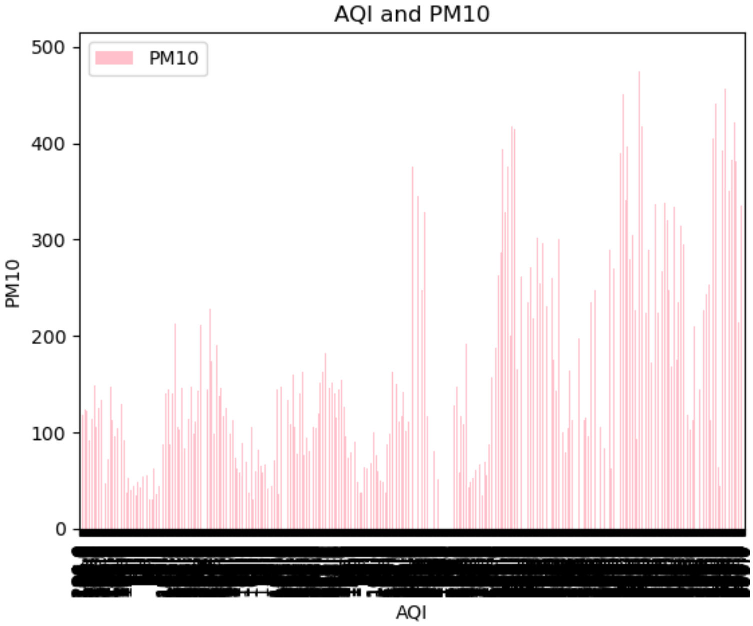
Bar Plot

In [36]:

```
data.plot(kind='bar',x='AQI',y='PM10',color='pink');
plt.title('AQI and PM10')
plt.xlabel("AQI")
plt.ylabel("PM10")
```

Out[36]:

```
Text(0, 0.5, 'PM10')
```

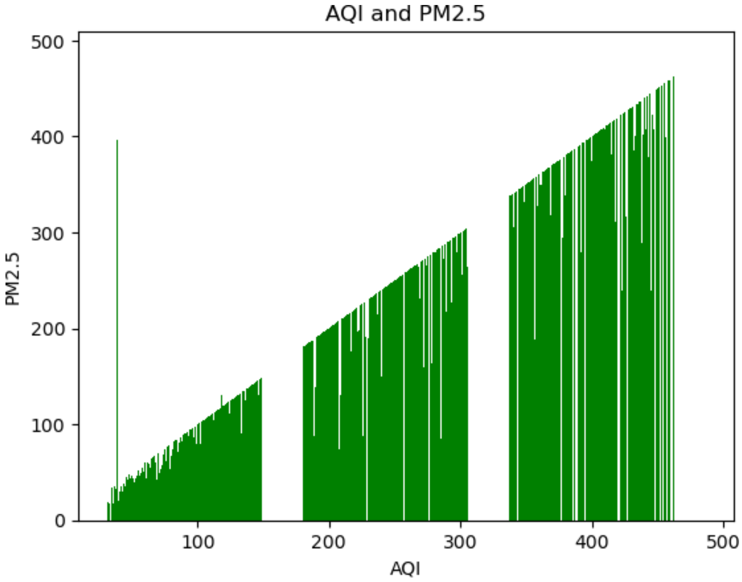


In [37]:

```
x=data['AQI']
y=data['PM2.5']
plt.bar(x,y,color='green')
plt.title('AQI and PM2.5')
plt.xlabel("AQI")
plt.ylabel("PM2.5")
```

Out[37]:

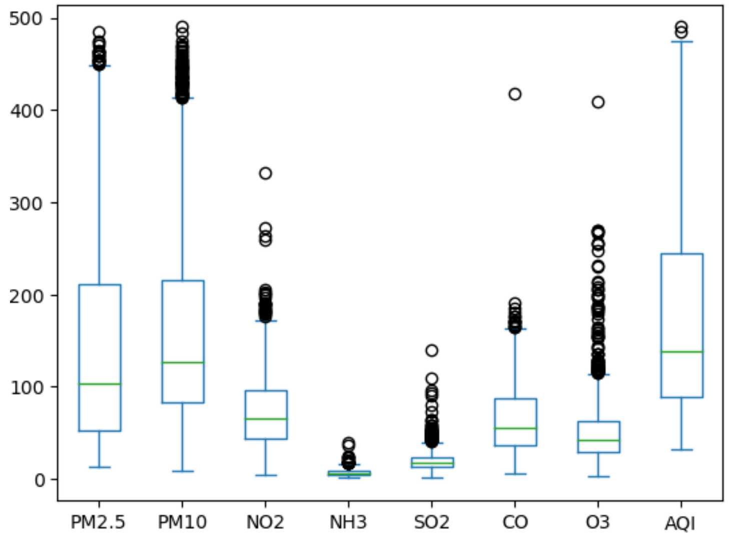
```
Text(0, 0.5, 'PM2.5')
```



Box Plot

In [38]:

```
data.plot(kind='box',y=['PM2.5','PM10','NO2','NH3','SO2','CO','O3','AQI']);
```



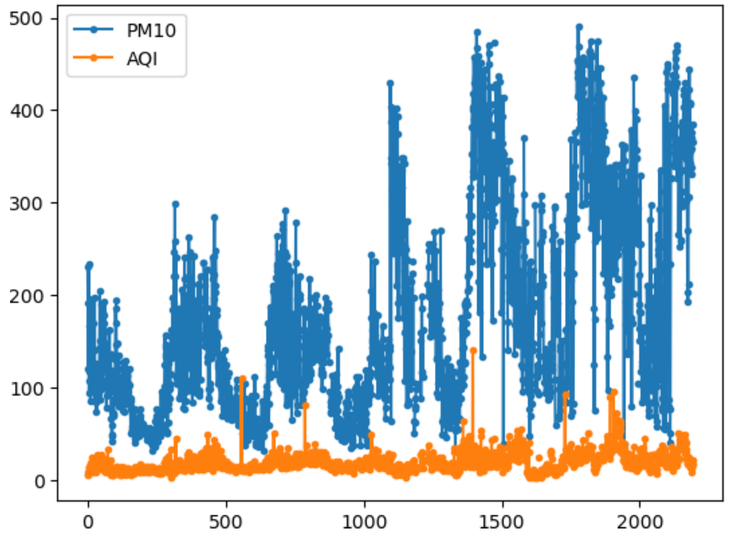
Line graph

In [39]:

```
#plt.plot(data.,label='PM2.5')
plt.plot(data.AQI,label='PM10',marker='.')
plt.plot(data.SO2,label='AQI',marker='.')
#plt.xlabel('')
#plt.ylabel('count')
plt.legend()
```

Out[39]:

```
<matplotlib.legend.Legend at 0x2649c4f1190>
```



In [40]:

```
change_satis = {"AQI_Bucket": { "satisfactory": "Satisfactory"}}
data=data.replace(change_satis)
```

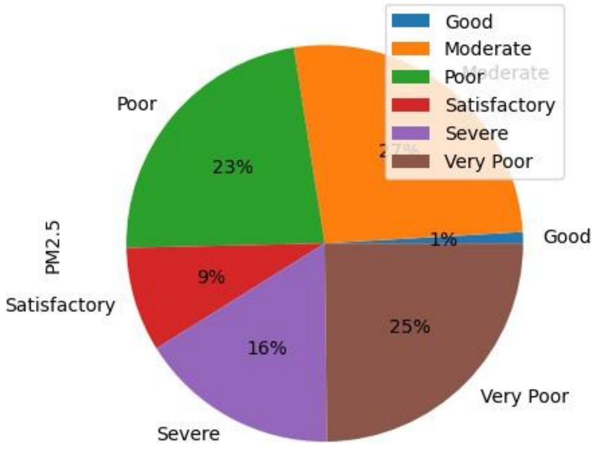
Pie Plot

In [41]:

```
data.groupby(['AQI_Bucket']).sum().plot(kind='pie', y='PM2.5',autopct='%1.0f%%')
```

Out[41]:

```
<AxesSubplot:ylabel='PM2.5'>
```



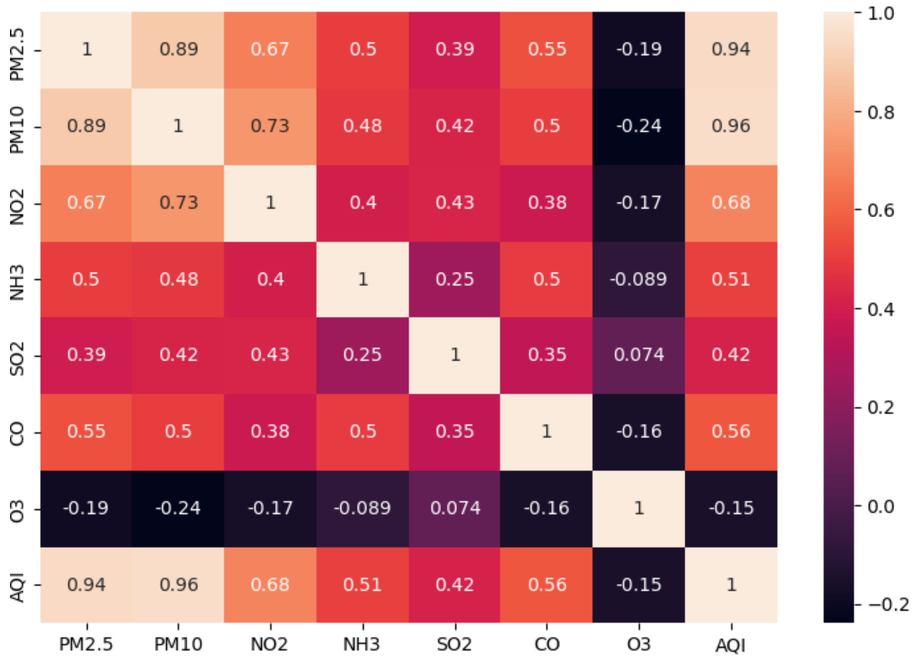
Correlation

In [42]:

```
corr=data.corr()
plt.subplots(figsize=(9,6))
sns.heatmap(corr,annot=True)
```

Out[42]:

```
<AxesSubplot:>
```



Finding categorical and continuous attribute

In [43]:

```
cate_val = []
cont_val = []
for column in data.columns:
 if data[column].nunique() <=10:
   cate_val.append(column)
 else:
   cont_val.append(column)
cate_val
```

Out[43]:

```
['City', 'AQI_Bucket']
```

In [44]:

```
cont_val
```

Out[44]:

```
['Date', 'PM2.5', 'PM10', 'NO2', 'NH3', 'SO2', 'CO', 'O3', 'AQI']
```

Data Preprocessing

In [45]:

```
data2 = data.copy()
```

In [46]:

```
#### replace null values with mean
```

In [47]:

```
data2 = data2.fillna(data2.mean())
```

```
C:\Users\Manju Sah\AppData\Local\Temp\ipykernel_19888\997279670.py:1: FutureWarning: Dropping of nuisance columns in Data
Frame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only vali
d columns before calling the reduction.
  data2 = data2.fillna(data2.mean())
```

In [48]:

```
data2
```

Out[48]:

	City	Date	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI	AQI_Bucket
0	Indore	01-01-2020	231.0	100.0	68.000000	6.000000	6.0	42.0	33.0	231.0	Poor
1	Indore	02-01-2020	191.0	121.0	59.000000	6.000000	6.0	52.0	30.0	191.0	Moderate
2	Indore	03-01-2020	120.0	91.0	48.000000	5.000000	9.0	36.0	32.0	120.0	Moderate
3	Indore	04-01-2020	234.0	181.0	80.000000	6.000000	13.0	23.0	38.0	234.0	Poor
4	Indore	05-01-2020	118.0	93.0	67.000000	5.000000	8.0	22.0	38.0	118.0	Moderate
...	...	...	...	...	...	...	...	...	...	...	...
2187	Delhi	27-12-2022	376.0	308.0	93.000000	11.000000	14.0	103.0	12.0	376.0	Very Poor
2188	Delhi	28-12-2022	331.0	265.0	120.000000	11.000000	9.0	89.0	15.0	331.0	Very Poor
2189	Delhi	29-12-2022	338.0	317.0	169.000000	12.000000	17.0	97.0	17.0	338.0	Very Poor
2190	Delhi	30-12-2022	385.0	298.0	124.000000	14.000000	17.0	92.0	25.0	385.0	Very Poor
2191	Delhi	31-12-2022	366.0	332.0	73.923347	6.990802	22.0	90.0	11.0	366.0	Very Poor

2192 rows × 11 columns

In [49]:

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2192 entries, 0 to 2191
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   City        2192 non-null   object 
 1   Date        2192 non-null   object 
 2   PM2.5       2192 non-null   float64
 3   PM10        2192 non-null   float64
 4   NO2         2192 non-null   float64
 5   NH3         2192 non-null   float64
 6   SO2         2192 non-null   float64
 7   CO          2192 non-null   float64
 8   O3          2192 non-null   float64
 9   AQI         2192 non-null   float64
 10  AQI_Bucket  2041 non-null   object 
dtypes: float64(8), object(3)
memory usage: 188.5+ KB
```

In [50]:

```
data2.columns
```

Out[50]:

```
Index(['City', 'Date', 'PM2.5', 'PM10', 'NO2', 'NH3', 'SO2', 'CO', 'O3', 'AQI',
       'AQI_Bucket'],
      dtype='object')
```

Encoding categorical values

In [51]:

```
dist=(data2['City'])
distset=set(dist)
dd=list(distset)
dictofwords={dd[i] : i for i in range(0, len(dd))}
data2['City']=data2['City'].map(dictofwords)
```

In [52]:

```
dist=(data2['AQI_Bucket'])
distset=set(dist)
dd=list(distset)
dictofwords={dd[i] : i for i in range(0, len(dd))}
data2['AQI_Bucket']=data2['AQI_Bucket'].map(dictofwords)
```

In [53]:

```
data2
```

Out[53]:

	City	Date	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI	AQI_Bucket
0	1	01-01-2020	231.0	100.0	68.000000	6.000000	6.0	42.0	33.0	231.0	6
1	1	02-01-2020	191.0	121.0	59.000000	6.000000	6.0	52.0	30.0	191.0	3
2	1	03-01-2020	120.0	91.0	48.000000	5.000000	9.0	36.0	32.0	120.0	3
3	1	04-01-2020	234.0	181.0	80.000000	6.000000	13.0	23.0	38.0	234.0	6
4	1	05-01-2020	118.0	93.0	67.000000	5.000000	8.0	22.0	38.0	118.0	3
...	...	...	...	...	...	...	...	...	...	...	...
2187	2	27-12-2022	376.0	308.0	93.000000	11.000000	14.0	103.0	12.0	376.0	4
2188	2	28-12-2022	331.0	265.0	120.000000	11.000000	9.0	89.0	15.0	331.0	4
2189	2	29-12-2022	338.0	317.0	169.000000	12.000000	17.0	97.0	17.0	338.0	4
2190	2	30-12-2022	385.0	298.0	124.000000	14.000000	17.0	92.0	25.0	385.0	4
2191	2	31-12-2022	366.0	332.0	73.923347	6.990802	22.0	90.0	11.0	366.0	4

2192 rows × 11 columns

In [54]:

```
data2.tail()
```

Out[54]:

	City	Date	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI	AQI_Bucket
2187	2	27-12-2022	376.0	308.0	93.000000	11.000000	14.0	103.0	12.0	376.0	4
2188	2	28-12-2022	331.0	265.0	120.000000	11.000000	9.0	89.0	15.0	331.0	4
2189	2	29-12-2022	338.0	317.0	169.000000	12.000000	17.0	97.0	17.0	338.0	4
2190	2	30-12-2022	385.0	298.0	124.000000	14.000000	17.0	92.0	25.0	385.0	4
2191	2	31-12-2022	366.0	332.0	73.923347	6.990802	22.0	90.0	11.0	366.0	4

In [55]:

```
data2.isnull().sum()
```

Out[55]:

```
City          0
Date          0
PM2.5         0
PM10          0
NO2           0
NH3           0
SO2           0
CO            0
O3            0
AQI           0
AQI_Bucket    0
dtype: int64
```

In [56]:

```
del(data2['AQI_Bucket'])
```

In [57]:

```
aqi_bins = [0,50,100,200,300,400,500]
aqi_labels = ["Good", "Satisfactory", "Moderate", "Poor","Very Poor","Severe"]
data2['AQIc'] = pd.cut(data2['AQI'], aqi_bins, labels=aqi_labels,
    right=True, include_lowest=True)
data2
```

Out[57]:

	City	Date	PM2.5	PM10	NO2	NH3	SO2	CO	O3	AQI	AQIc
0	1	01-01-2020	231.0	100.0	68.000000	6.000000	6.0	42.0	33.0	231.0	Poor
1	1	02-01-2020	191.0	121.0	59.000000	6.000000	6.0	52.0	30.0	191.0	Moderate
2	1	03-01-2020	120.0	91.0	48.000000	5.000000	9.0	36.0	32.0	120.0	Moderate
3	1	04-01-2020	234.0	181.0	80.000000	6.000000	13.0	23.0	38.0	234.0	Poor
4	1	05-01-2020	118.0	93.0	67.000000	5.000000	8.0	22.0	38.0	118.0	Moderate
...	...	...	...	...	...	...	...	...	...	...	...
2187	2	27-12-2022	376.0	308.0	93.000000	11.000000	14.0	103.0	12.0	376.0	Very Poor
2188	2	28-12-2022	331.0	265.0	120.000000	11.000000	9.0	89.0	15.0	331.0	Very Poor
2189	2	29-12-2022	338.0	317.0	169.000000	12.000000	17.0	97.0	17.0	338.0	Very Poor
2190	2	30-12-2022	385.0	298.0	124.000000	14.000000	17.0	92.0	25.0	385.0	Very Poor
2191	2	31-12-2022	366.0	332.0	73.923347	6.990802	22.0	90.0	11.0	366.0	Very Poor

2192 rows × 11 columns

features selection (remove unwanted columns)

In [58]:

```
data2=data2.drop('Date',1)
data2=data2.drop('AQI',1)
data2=data2.drop('City',1)
```

```
C:\Users\Manju Sah\AppData\Local\Temp\ipykernel_19888\1112469306.py:1: FutureWarning: In a future version of pandas all a
rguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  data2=data2.drop('Date',1)
C:\Users\Manju Sah\AppData\Local\Temp\ipykernel_19888\1112469306.py:2: FutureWarning: In a future version of pandas all a
rguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  data2=data2.drop('AQI',1)
C:\Users\Manju Sah\AppData\Local\Temp\ipykernel_19888\1112469306.py:3: FutureWarning: In a future version of pandas all a
rguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  data2=data2.drop('City',1)
```

In [59]:

```
data2.columns
```

Out[59]:

```
Index(['PM2.5', 'PM10', 'NO2', 'NH3', 'SO2', 'CO', 'O3', 'AQIc'], dtype='object')
```

Splitting the data into train and test data

In [60]:

```
features=data2.drop(['AQIc'], axis=1)
labels=data2['AQIc']
```

In [61]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,labels,test_size=0.2,random_state=2)
```

In [62]:

```
X_train
```

Out[62]:

	PM2.5	PM10	NO2	NH3	SO2	CO	O3
117	44.000000	96.000	56.000000	4.000000	5.000000	21.000000	35.000000
923	35.000000	49.000	49.000000	5.000000	13.000000	44.000000	24.000000
1289	144.831653	159.569	73.923347	6.990802	19.270916	62.087149	49.987747
1355	142.000000	165.000	60.000000	6.000000	28.000000	73.000000	44.000000
365	210.000000	142.000	52.000000	6.000000	23.000000	39.000000	49.000000
...	...	...	...	...	...	...	...
1071	44.000000	120.000	81.000000	6.000000	22.000000	35.000000	49.987747
433	70.000000	134.000	63.000000	4.000000	17.000000	58.000000	36.000000
674	182.000000	159.569	127.000000	8.000000	50.000000	59.000000	107.000000
1099	144.831653	159.569	73.923347	6.990802	19.270916	62.087149	35.000000
1608	144.831653	181.000	59.000000	10.000000	4.000000	100.000000	11.000000

1753 rows × 7 columns

In [63]:

```
X_test
```

Out[63]:

	PM2.5	PM10	NO2	NH3	SO2	CO	O3
278	79.000000	102.000	46.000000	3.000000	18.000000	69.000000	40.000000
2136	327.000000	233.000	110.000000	10.000000	27.000000	76.000000	41.000000
1657	131.000000	128.000	50.000000	9.000000	13.000000	91.000000	50.000000
1459	296.000000	200.000	97.000000	11.000000	22.000000	97.000000	46.000000
1779	335.000000	414.000	152.000000	12.000000	26.000000	112.000000	4.000000
...	...	...	...	...	...	...	...
1695	144.831653	159.569	73.923347	6.990802	19.270916	62.087149	49.987747
1372	144.831653	188.000	100.000000	5.000000	34.000000	80.000000	42.000000
391	156.000000	143.000	56.000000	4.000000	14.000000	39.000000	51.000000
459	85.000000	231.000	82.000000	6.000000	44.000000	36.000000	61.000000
176	33.000000	55.000	36.000000	4.000000	11.000000	29.000000	48.000000

439 rows × 7 columns

In [64]:

```
y_train
```

Out[64]:

```
117     Satisfactory
923             Good
1289        Moderate
1355        Moderate
365             Poor
            ...     
1071        Moderate
433         Moderate
674         Moderate
1099    Satisfactory
1608        Moderate
Name: AQIc, Length: 1753, dtype: category
Categories (6, object): ['Good' < 'Satisfactory' < 'Moderate' < 'Poor' < 'Very Poor' < 'Severe']
```

In [65]:

```
y_test
```

Out[65]:

```
278         Moderate
2136       Very Poor
1657        Moderate
1459            Poor
1779          Severe
            ...     
1695        Moderate
1372        Moderate
391         Moderate
459             Poor
176     Satisfactory
Name: AQIc, Length: 439, dtype: category
Categories (6, object): ['Good' < 'Satisfactory' < 'Moderate' < 'Poor' < 'Very Poor' < 'Severe']
```

Support Vector Machine

In [66]:

```
from sklearn import svm
from sklearn.metrics import accuracy_score
```

In [67]:

```
svm = svm.SVC()
```

In [68]:

```
svm.fit(X_train,y_train)
```

Out[68]:

```
SVC()
```

In [69]:

```
y_pred1 = svm.predict(X_test)
```

In [70]:

```
acc=accuracy_score(y_test,y_pred1)
per1=acc*100
print(per1)
```

```
88.8382687927107
```

In [71]:

```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

        Good       1.00      0.30      0.46        20
    Moderate       0.92      0.93      0.92       171
        Poor       0.93      0.84      0.88        67
Satisfactory       0.81      0.93      0.87       104
      Severe       0.93      0.96      0.94        26
   Very Poor       0.89      0.92      0.90        51

    accuracy                           0.89       439
   macro avg       0.91      0.81      0.83       439
weighted avg       0.90      0.89      0.88       439
```

AUC Plot

In [72]:

```
data3=data2.copy()
```

In [73]:

```
subset=data3.iloc[:,]
```

In [74]:

```
subset.shape
```

Out[74]:

```
(2192, 8)
```

In [75]:

```
#separate the labels/classes from the features/measurement
X=subset.iloc[:,1:]
y=subset.iloc[:,0]
print(y.shape,X.shape)
```

```
(2192,) (2192, 7)
```

In [76]:

```
#Lets encode target labels y with values between 0 and n_classes-1
#we will use the labelencoder to do this
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
label_encoder.fit(y)
y=label_encoder.transform(y)
#classes=label_encoder.classes_
classes=['Good','Satisfacory','Moderate','Poor','Very Poor','Severe']
```

In [77]:

```
# Normalize the data i.e scale the data between 0 and 1
from sklearn.preprocessing import MinMaxScaler
min_max_scaler=MinMaxScaler()
X_train_norm=min_max_scaler.fit_transform(X_train)
X_test_norm=min_max_scaler.fit_transform(X_test)
X_train_norm[0,0]
```

Out[77]:

```
0.06724511930585683
```

In [78]:

```
from sklearn.multiclass import OneVsRestClassifier
#from sklearn.svm import SVC
from sklearn import svm
from sklearn.metrics import roc_curve,auc
```

In [79]:

```
#y_predroc=clf.predict(X_test)
y_predroc = svm.predict(X_test)
pred_prob = svm.predict_proba(X_test)
#pred_prob.shape
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19888\2616250888.py in <module>
      1 #y_predroc=clf.predict(X_test)
----> 2 y_predroc = svm.predict(X_test)
      3 pred_prob = svm.predict_proba(X_test)
      4 #pred_prob.shape

AttributeError: module 'sklearn.svm' has no attribute 'predict'
```

In [80]:

```
from sklearn.preprocessing import label_binarize
#binarize the y_values
y_test_binarized=label_binarize(y_test,classes=np.unique(y_test))
#roc curve for classes
fpr={}
tpr={}
thresh={}
roc_auc=dict()
#n_class=classes.shape[0]
for i in range(6):
    fpr[i],tpr[i],thresh[i]=roc_curve(y_test_binarized[:,i],pred_prob[:,i])
    roc_auc[i]=auc(fpr[i],tpr[i])

    #plotting
    plt.plot(fpr[i],tpr[i],linestyle='--',
             label='%s vs Rest (AUC=%0.2f)'%(classes[i],roc_auc[i]))
plt.plot([0,1],[0,1],'b--')
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.title('ROC curve of SVM')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19888\3602692938.py in <module>
      9 #n_class=classes.shape[0]
     10 for i in range(6):
---> 11     fpr[i],tpr[i],thresh[i]=roc_curve(y_test_binarized[:,i],pred_prob[:,i])
     12     roc_auc[i]=auc(fpr[i],tpr[i])
     13

NameError: name 'pred_prob' is not defined
```

KNN

In [81]:

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

In [82]:

```
knn = KNeighborsClassifier()
```

In [83]:

```
knn.fit(X_train, y_train)
```

Out[83]:

```
KNeighborsClassifier()
```

In [84]:

```
y_pred2 = knn.predict(X_test)
```

```
C:\Users\Manju  Sah\Desktop\code\annaconda\lib\site-packages\sklearn\neighbors\_classification.py:228:  FutureWarning:  Unli
ke other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts
along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False t
o avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [85]:

```
acc=accuracy_score(y_test,y_pred2)
per2=acc*100
print(per2)
```

```
91.34396355353076
```

In [86]:

```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

        Good       0.89      0.85      0.87        20
    Moderate       0.93      0.95      0.94       171
        Poor       0.90      0.85      0.88        67
Satisfactory       0.92      0.93      0.92       104
      Severe       1.00      0.81      0.89        26
   Very Poor       0.85      0.90      0.88        51

    accuracy                           0.91       439
   macro avg       0.92      0.88      0.90       439
weighted avg       0.91      0.91      0.91       439
```

In [87]:

```
from sklearn.multiclass import OneVsRestClassifier
#from sklearn.svm import SVC
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve,auc
```

In [88]:

```
y_predroc = knn.predict(X_test)
pred_prob = knn.predict_proba(X_test)
pred_prob.shape
```

```
C:\Users\Manju  Sah\Desktop\code\annaconda\lib\site-packages\sklearn\neighbors\_classification.py:228:  FutureWarning:  Unli
ke other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts
along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False t
o avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

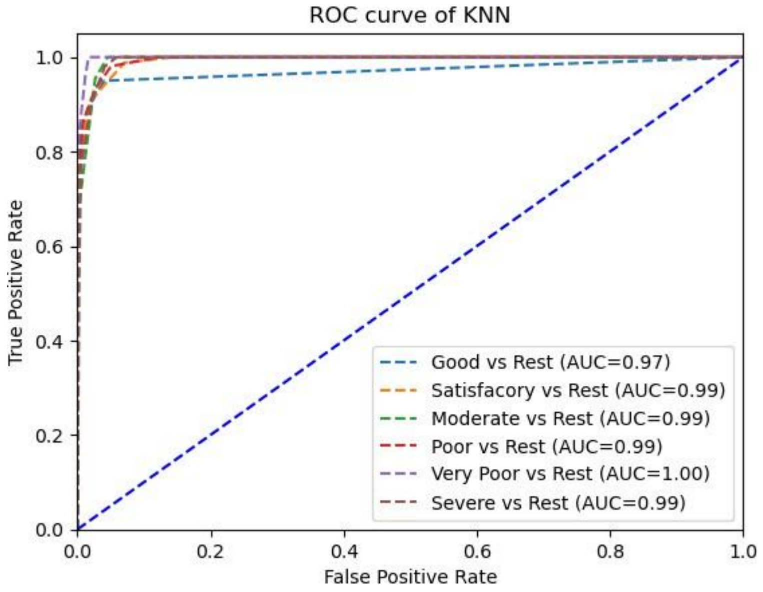
Out[88]:

```
(439, 6)
```

In [89]:

```
from sklearn.preprocessing import label_binarize
#binarize the y_values
y_test_binarized=label_binarize(y_test,classes=np.unique(y_test))
#roc curve for classes
fpr={}
tpr={}
thresh={}
roc_auc=dict()
#n_class=classes.shape[0]
for i in range(6):
    fpr[i],tpr[i],thresh[i]=roc_curve(y_test_binarized[:,i],pred_prob[:,i])
    roc_auc[i]=auc(fpr[i],tpr[i])

    #plotting
    plt.plot(fpr[i],tpr[i],linestyle='--',
             label='%s vs Rest (AUC=%0.2f)'%(classes[i],roc_auc[i]))
plt.plot([0,1],[0,1],'b--')
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.title('ROC curve of KNN')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



Logistic Regression

In [90]:

```
from sklearn.linear_model import LogisticRegression
```

In [91]:

```
log = LogisticRegression()
log.fit(X_train,y_train)
```

```
C:\Users\Manju Sah\Desktop\code\annaconda\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lb
fgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/
linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[91]:

```
LogisticRegression()
```

In [92]:

```
y_pred3 = log.predict(X_test)
```

In [93]:

```
acc=accuracy_score(y_test,y_pred3)
per3=acc*100
print(per3)
```

```
57.85876993166287
```

In [94]:

```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

        Good       0.00      0.00      0.00        20
    Moderate       0.63      0.85      0.72       171
        Poor       0.44      0.28      0.35        67
Satisfactory       0.60      0.49      0.54       104
      Severe       0.43      0.35      0.38        26
   Very Poor       0.51      0.57      0.54        51

    accuracy                           0.58       439
   macro avg       0.43      0.42      0.42       439
weighted avg       0.54      0.58      0.55       439
```

In [95]:

```
from sklearn.multiclass import OneVsRestClassifier
#from sklearn.svm import SVC
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve,auc
```

In [96]:

```
y_predroc = log.predict(X_test)
pred_prob = log.predict_proba(X_test)
pred_prob.shape
```

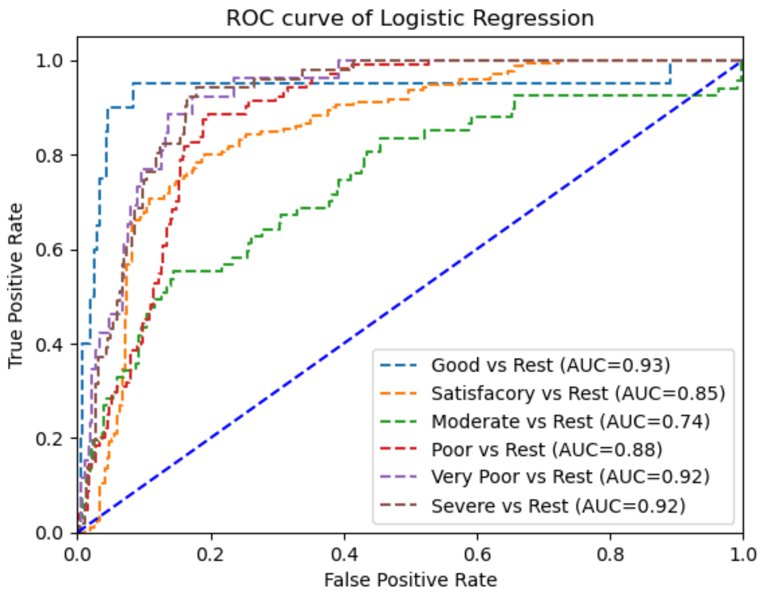
Out[96]:

```
(439, 6)
```

In [97]:

```
from sklearn.preprocessing import label_binarize
#binarize the y_values
y_test_binarized=label_binarize(y_test,classes=np.unique(y_test))
#roc curve for classes
fpr={}
tpr={}
thresh={}
roc_auc=dict()
#n_class=classes.shape[0]
for i in range(6):
    fpr[i],tpr[i],thresh[i]=roc_curve(y_test_binarized[:,i],pred_prob[:,i])
    roc_auc[i]=auc(fpr[i],tpr[i])

    #plotting
    plt.plot(fpr[i],tpr[i],linestyle='--',
             label='%s vs Rest (AUC=%0.2f)'%(classes[i],roc_auc[i]))
plt.plot([0,1],[0,1],'b--')
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.title('ROC curve of Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



Decision Tree Classifier

In [98]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [99]:

```
dt=DecisionTreeClassifier()
```

In [100]:

```
dt.fit(X_train, y_train)
```

Out[100]:

```
DecisionTreeClassifier()
```

In [101]:

```
y_pred4 = dt.predict(X_test)
```

In [102]:

```
acc=accuracy_score(y_test, y_pred4)
per4=acc*100
print(per4)
```

```
96.58314350797266
```

In [103]:

```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test,y_pred4))
```

```
              precision    recall  f1-score   support

        Good       0.86      0.95      0.90        20
    Moderate       0.98      0.95      0.96       171
        Poor       0.98      0.97      0.98        67
Satisfactory       0.94      0.97      0.96       104
      Severe       1.00      1.00      1.00        26
   Very Poor       0.98      0.98      0.98        51

    accuracy                           0.97       439
   macro avg       0.96      0.97      0.96       439
weighted avg       0.97      0.97      0.97       439
```

In [104]:

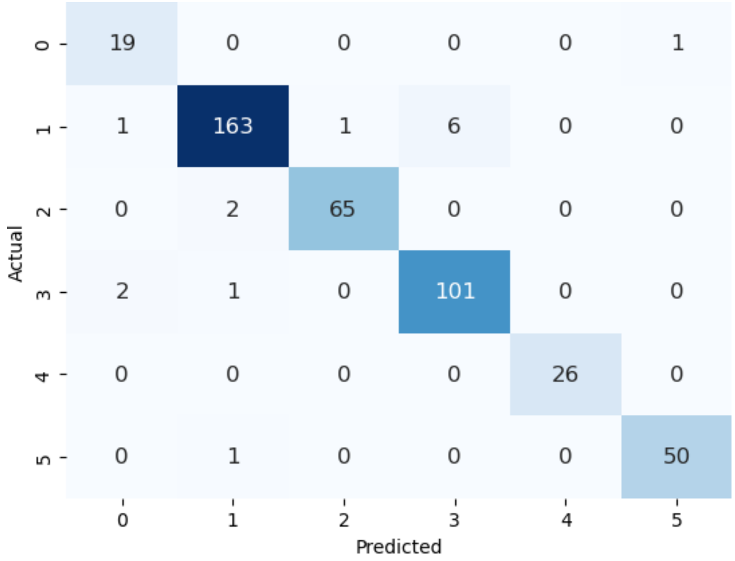
```
cm=confusion_matrix(y_test, y_pred4)
cm
```

Out[104]:

```
array([[ 19,   0,   0,   0,   0,   1],
       [  1, 163,   1,   6,   0,   0],
       [  0,   2,  65,   0,   0,   0],
       [  2,   1,   0, 101,   0,   0],
       [  0,   0,   0,   0,  26,   0],
       [  0,   1,   0,   0,   0,  50]], dtype=int64)
```

In [105]:

```
import seaborn as sns
import matplotlib.pyplot as plt
#sns.heatmap(cm,cmap='Greens',annot=True,
            #cbar_kws={'orientation':'vertical','label':'color bar'},
            #xticklabels=[0,1],yticklabels=[0,1] )
sns.heatmap(cm,annot=True, cmap='Blues', fmt='d',cbar=False, annot_kws={'size':12})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



AUC plot

In [106]:

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve,auc
```

In [107]:

```
#DT=OneVsRestClassifier(DecisionTreeClassifier)
#DT.fit(X_train,y_train)
y_predroc=dt.predict(X_test)
pred_prob=dt.predict_proba(X_test)
```

In [108]:

```
pred_prob.shape
```

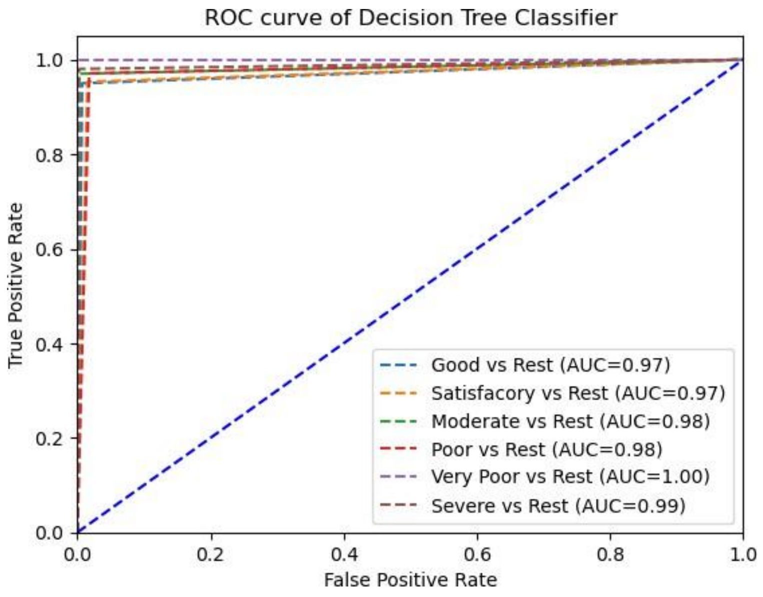
Out[108]:

```
(439, 6)
```

In [109]:

```
from sklearn.preprocessing import label_binarize
#binarize the y_values
y_test_binarized=label_binarize(y_test,classes=np.unique(y_test))
#roc curve for classes
fpr={}
tpr={}
thresh={}
roc_auc=dict()
#n_class=classes.shape[0]
for i in range(6):
    fpr[i],tpr[i],thresh[i]=roc_curve(y_test_binarized[:,i],pred_prob[:,i])
    roc_auc[i]=auc(fpr[i],tpr[i])

    #plotting
    plt.plot(fpr[i],tpr[i],linestyle='--',
             label='%s vs Rest (AUC=%0.2f)'%(classes[i],roc_auc[i]))
plt.plot([0,1],[0,1],'b--')
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.title('ROC curve of Decision Tree Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



Comparison

In [110]:

```
final_data = pd.DataFrame({'MODELS':['SVM','KNN','LR','DT'],
                           'ACCURACY':[per1, per2, per3, per4]})
```

In [111]:

```
final_data
```

Out[111]:

	MODELS	ACCURACY
0	SVM	88.838269
1	KNN	91.343964
2	LR	57.858770
3	DT	96.583144

In [112]:

```
import seaborn as sns
```

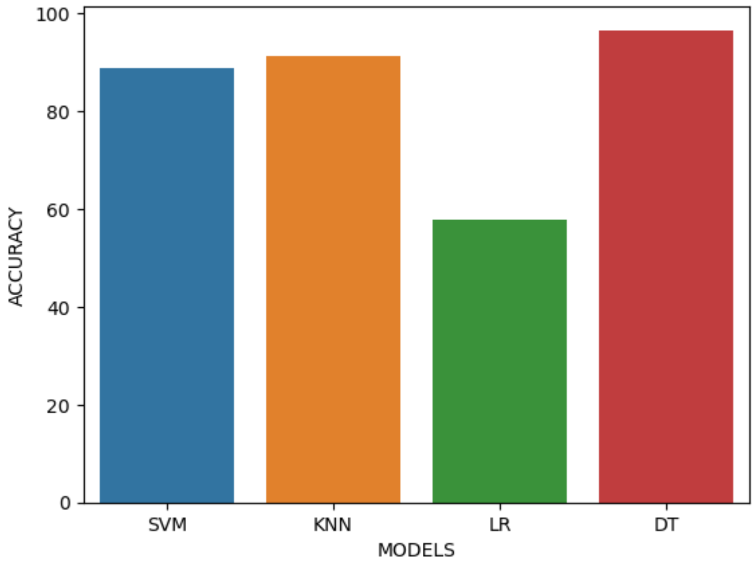
In [113]:

```
sns.barplot(final_data['MODELS'], final_data['ACCURACY'])
```

```
C:\Users\Manju Sah\Desktop\code\annaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following
variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other ar
guments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[113]:

```
<AxesSubplot:xlabel='MODELS', ylabel='ACCURACY'>
```



Prediction on New Data

Enter PM2.5, PM10, NO2, NH3, SO2, CO, O3 as input

In [114]:

```
dt.predict([[463, 454, 78, 5, 29, 102, 18]])
```

```
C:\Users\Manju Sah\Desktop\code\annaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid featu
re names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

Out[114]:

```
array(['Severe'], dtype=object)
```

THANKS