In [1]:
```
import pandas as pd  #data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np #Linear algebra

# running this (by clicking run or pressing Shift+Enter) will list all files u
# You can write up to 20GB to the current directory (/kaggle/working/) that ge
# You can also write temporary files to /kaggle/temp/, but they won't be saved

import matplotlib.pyplot as plt #pyplot is a collection of command style funct
import seaborn as sns #It is advance data visualization library
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
sns.set_style('darkgrid') # It sets the style of the plots to have a dark back
```

Load Dataset

In [3]:
```
df=pd.read_csv('milkquality.csv') #df is name given to the dataset
```

Basic EDA

In [4]:
```
#Print first 5 rows of dataset
df.head()
```

Out[4]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	high
1	6.6	36	0	1	0	1	253	high
2	8.5	70	1	1	1	1	246	low
3	9.5	34	1	1	0	1	255	low
4	6.6	37	0	0	0	0	255	medium

In [5]:
```
#Print last 5 rows of dataset
df.tail()
```

Out[5]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
1054	6.7	45	1	1	0	0	247	medium
1055	6.7	38	1	0	1	0	255	high
1056	3.0	40	1	1	1	1	255	low
1057	6.8	43	1	0	1	0	250	high
1058	8.6	55	0	1	1	1	255	low

In [6]:
```
df.shape # which presents columns and rows of the dataset
```

Out[6]:
```
(1059, 8)
```

In [7]:
```
#the information contains the number of columns, column labels, column data typ
#memory usage, range index, and the number of cells in each column (non-null va

df.info() #actually prints the info.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1059 entries, 0 to 1058
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   pH          1059 non-null   float64
 1   Temprature  1059 non-null   int64  
 2   Taste       1059 non-null   int64  
 3   Odor        1059 non-null   int64  
 4   Fat         1059 non-null   int64  
 5   Turbidity   1059 non-null   int64  
 6   Colour      1059 non-null   int64  
 7   Grade       1059 non-null   object 
dtypes: float64(1), int64(6), object(1)
memory usage: 66.3+ KB
```

Basic Statistic details about the data

note only numerical columns would be displayed here unless parameter include="all"

- count tells us the number of NoN-empty rows in a feature.
- mean tells us the mean value of that feature.
- std tells us the Standard Deviation Value of that feature.
- min tells us the minimum value of that feature.
- 25%, 50%, and 75% are the percentile/quartile of each features. This quartile information helps us to detect Outliers.
- max tells us the maximum value of that feature.

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

Out[8]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colo
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.0000
mean	6.630123	44.226629	0.546742	0.432483	0.671388	0.491029	251.8404
std	1.399679	10.098364	0.498046	0.495655	0.469930	0.500156	4.3074
min	3.000000	34.000000	0.000000	0.000000	0.000000	0.000000	240.0000
25%	6.500000	38.000000	0.000000	0.000000	0.000000	0.000000	250.0000
50%	6.700000	41.000000	1.000000	0.000000	1.000000	0.000000	255.0000
75%	6.800000	45.000000	1.000000	1.000000	1.000000	1.000000	255.0000
max	9.500000	90.000000	1.000000	1.000000	1.000000	1.000000	255.0000

Data Cleaning

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
976
```

In [10]:
```
#We can remove the duplicates values using dropna..
df.drop_duplicates()
```

Out[10]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	high
1	6.6	36	0	1	0	1	253	high
2	8.5	70	1	1	1	1	246	low
3	9.5	34	1	1	0	1	255	low
4	6.6	37	0	0	0	0	255	medium
...	...	...	...	...	...	...	...	...
930	6.6	38	0	1	1	1	255	high
942	6.6	45	1	0	0	1	255	medium
957	6.8	41	1	1	1	0	255	high
985	6.5	45	1	0	0	0	246	medium
998	6.6	43	0	0	0	1	250	medium

83 rows × 8 columns

In [11]:
```
#Checking for null values which are present in dataset
df.isnull().sum()
```

Out[11]:
```
pH             0
Temprature     0
Taste          0
Odor           0
Fat            0
Turbidity      0
Colour         0
Grade          0
dtype: int64
```

In [12]:
```
df.nunique()
```

Out[12]:
```
pH             16
Temprature     17
Taste           2
Odor            2
Fat             2
Turbidity       2
Colour          9
Grade           3
dtype: int64
```

Value counts of Different Colums

In [13]:
```
df['Grade'].value_counts()
```

Out[13]:
```
low       429
medium    374
high      256
Name: Grade, dtype: int64
```

In [14]:
```
df['pH'].value_counts()
```

Out[14]:
```
6.8    249
6.5    189
6.6    159
6.7     82
3.0     70
9.0     61
8.6     40
7.4     39
4.5     37
9.5     24
8.1     24
5.5     23
8.5     22
4.7     20
5.6     19
6.4      1
Name: pH, dtype: int64
```

In [15]:
```
df['Taste'].value_counts()
```

Out[15]:
```
1    579
0    480
Name: Taste, dtype: int64
```

In [16]:
```
df['Grade'].value_counts()
```

Out[16]:
```
low       429
medium    374
high      256
Name: Grade, dtype: int64
```

In [17]:
```
df['Temprature'].value_counts()
```

Out[17]:
```
45    219
38    179
40    132
37     83
43     77
36     66
50     58
55     48
34     40
41     30
66     24
35     23
70     22
65     22
60     18
90     17
42      1
Name: Temprature, dtype: int64
```

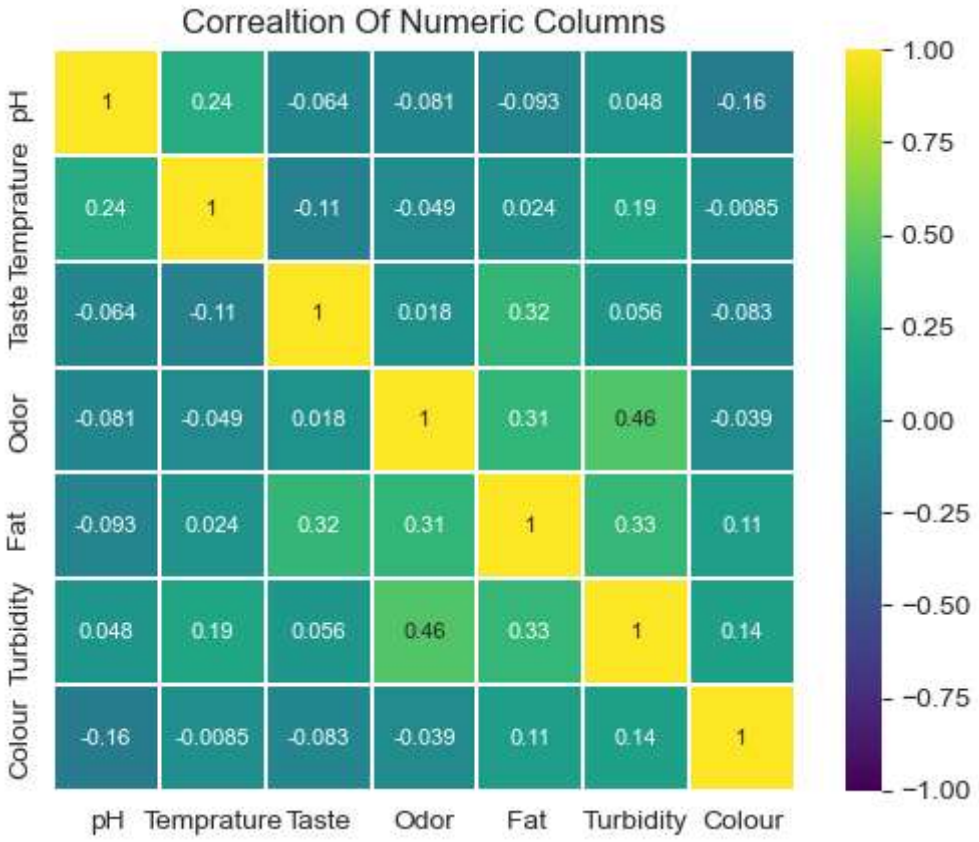
In [18]:
```
#Correlation of columns between each other
df.corr()
```

Out[18]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
pH	1.000000	0.244684	-0.064053	-0.081331	-0.093429	0.048384	-0.164565
Temprature	0.244684	1.000000	-0.109792	-0.048870	0.024073	0.185106	-0.008511
Taste	-0.064053	-0.109792	1.000000	0.017582	0.324149	0.055755	-0.082654
Odor	-0.081331	-0.048870	0.017582	1.000000	0.314505	0.457935	-0.039361
Fat	-0.093429	0.024073	0.324149	0.314505	1.000000	0.329264	0.114151
Turbidity	0.048384	0.185106	0.055755	0.457935	0.329264	1.000000	0.136436
Colour	-0.164565	-0.008511	-0.082654	-0.039361	0.114151	0.136436	1.000000

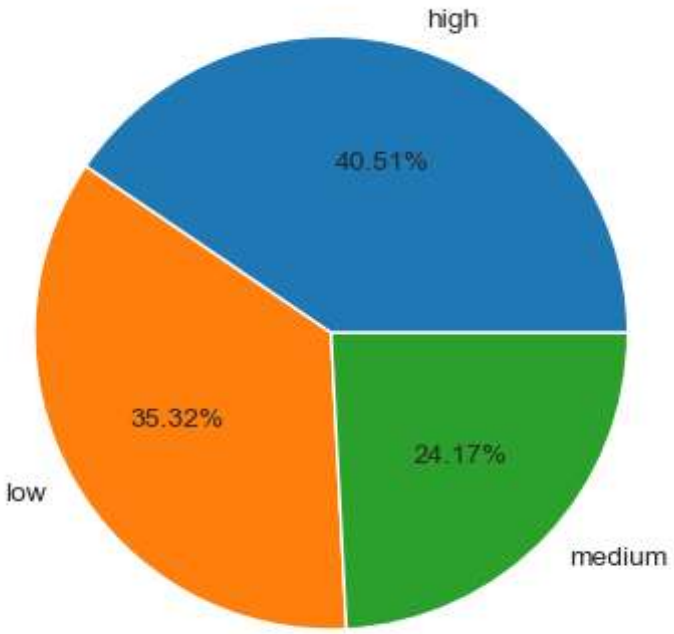
Data Visualization

In [19]:
```
heatmap=sns.heatmap(
    df.corr(),           # Compute the correlation matrix of the DataFrame
    annot=True,          # Display the correlation values on the heatmap
    cmap='viridis',      # Set the color map for the heatmap (using the 'virid
    vmax=1.0,            # Set the maximum value of the color range to 1.0
    vmin=-1.0,           # Set the minimum value of the color range to -1.0
    linewidths=0.1,      # Set the width of the lines between cells in the hea
    annot_kws={"size": 8},  # Set the font size for the annotations on the hea
    square=True          # Force the heatmap cells to be square-shaped
)
heatmap.set_title('Correaltion Of Numeric Columns') #Set the title for heatmap
plt.show()
```



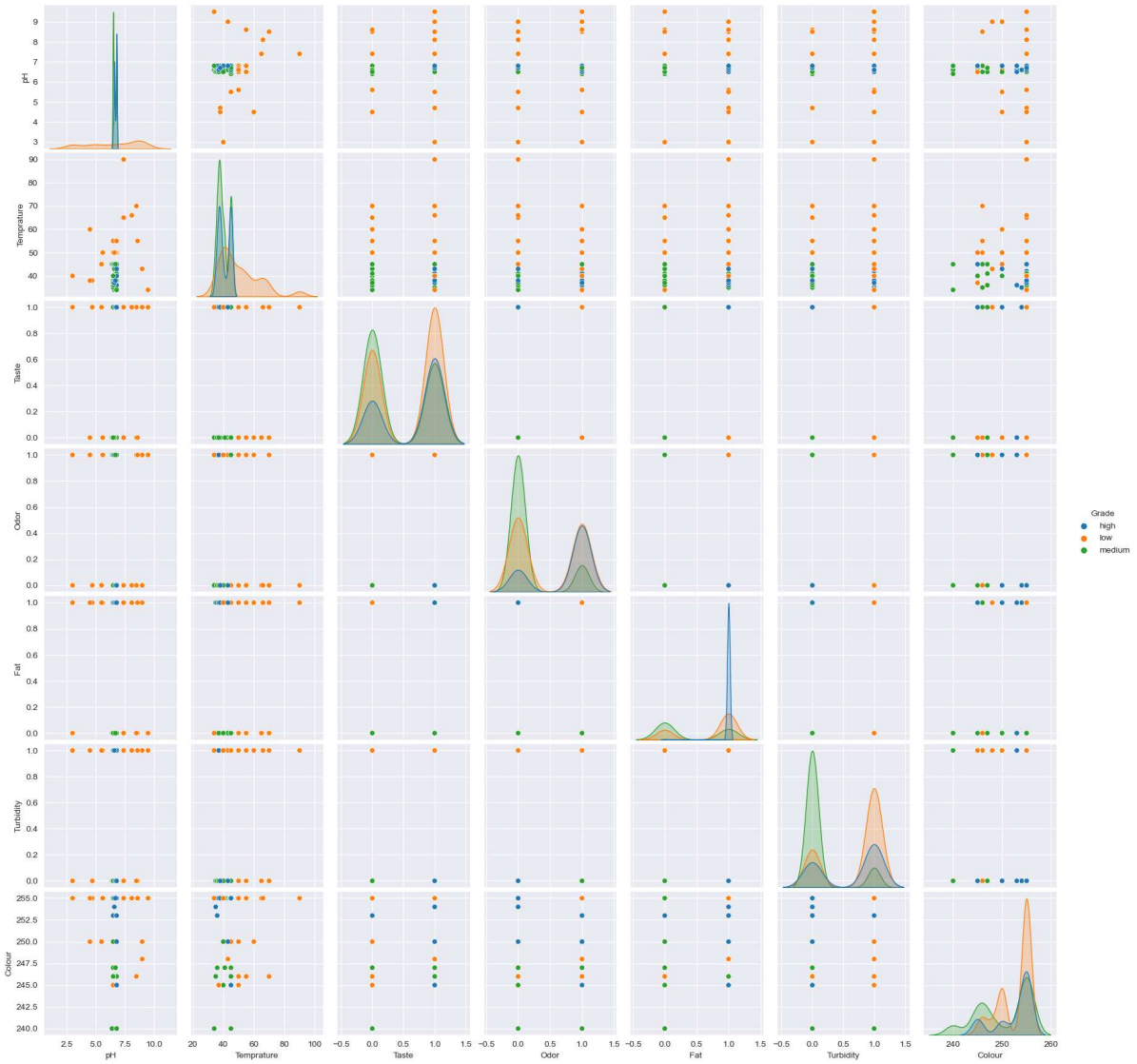
Grade Distribution

In [20]:
```
#Grade  distribution
plt.pie(df['Grade'].value_counts(),autopct='%1.2f%%',labels=np.unique(df['Grad
plt.show()
```



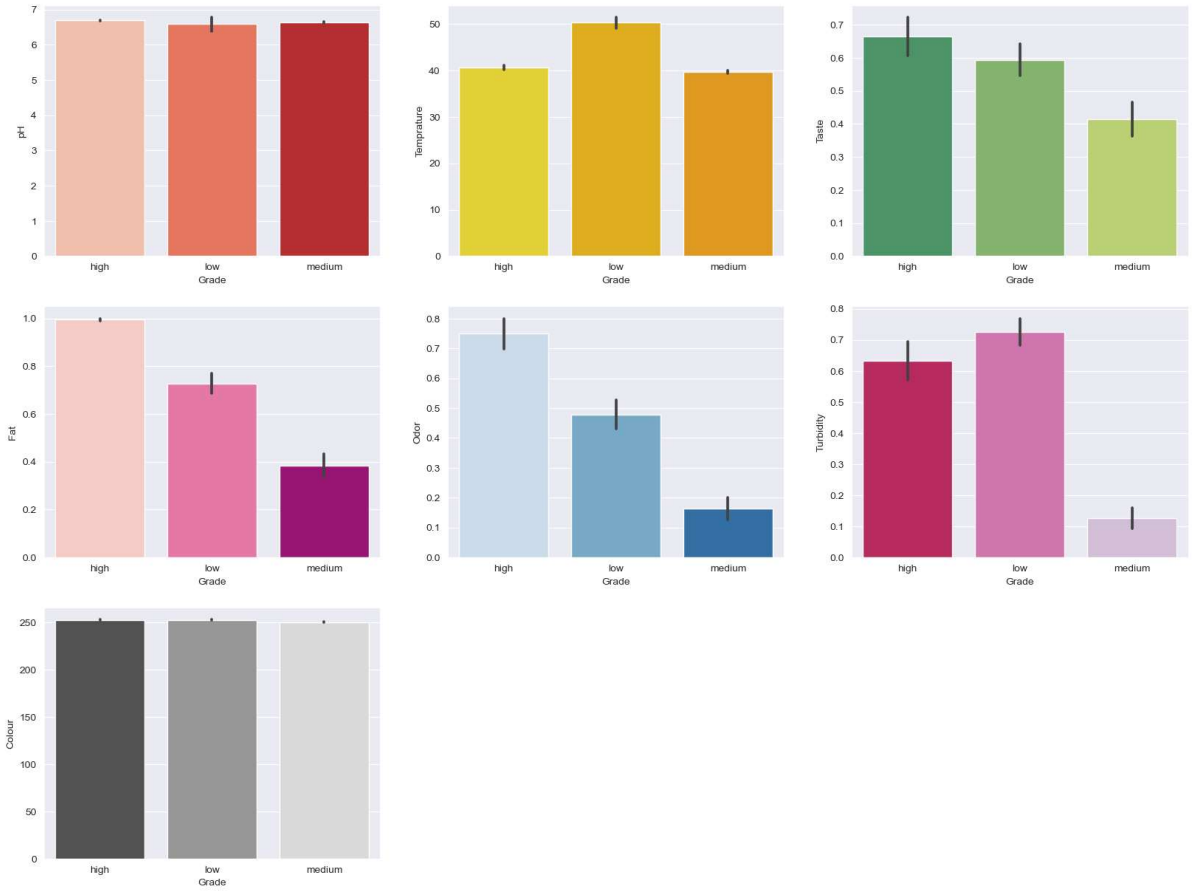
In [21]:
```
sns.pairplot(df,hue='Grade')
```

Out[21]:
```
<seaborn.axisgrid.PairGrid at 0x18ff2327760>
```



In [22]:
```
plt.figure(figsize=(20,15))

plt.subplot(3,3,1)
sns.barplot(x = 'Grade', y = 'pH', data = df, palette="Reds")

plt.subplot(3,3,2)
sns.barplot(x = 'Grade', y = 'Temprature', data = df, palette="Wistia")

plt.subplot(3,3,3)
sns.barplot(x = 'Grade', y = 'Taste', data = df, palette="summer")

plt.subplot(3,3,5)
sns.barplot(x = 'Grade', y = 'Odor', data = df, palette="Blues")

plt.subplot(3,3,4)
sns.barplot(x = 'Grade', y = 'Fat ', data = df, palette="RdPu")

plt.subplot(3,3,6)
sns.barplot(x = 'Grade', y = 'Turbidity', data = df, palette="PuRd_r")

plt.subplot(3,3,7)
sns.barplot(x = 'Grade', y = 'Colour', data = df, palette="Greys_r")
```

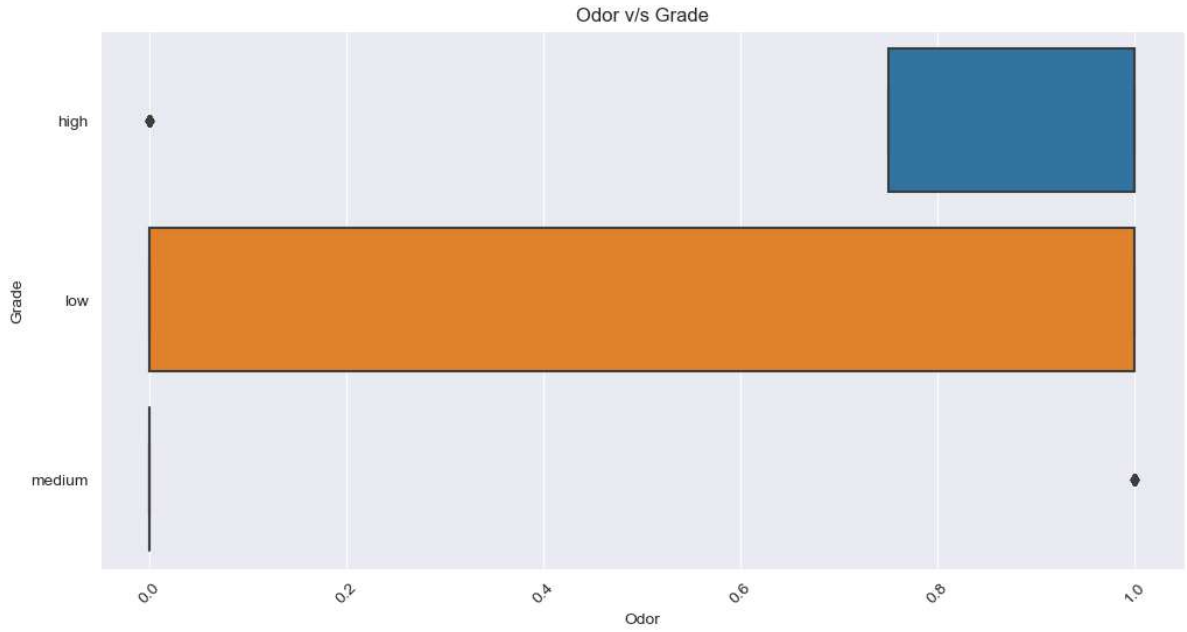
Out[22]: <AxesSubplot:xlabel='Grade', ylabel='Colour'>



Odor v/s Grade using Boxplot

In [23]:
```
plt.figure(figsize = (12, 6))
ax = sns.boxplot(y='Grade', x='Odor', data=df)
plt.setp(ax.artists, alpha=.5, linewidth=2, edgecolor="k")
plt.xticks(rotation=45)
plt.title('Odor v/s Grade')
```

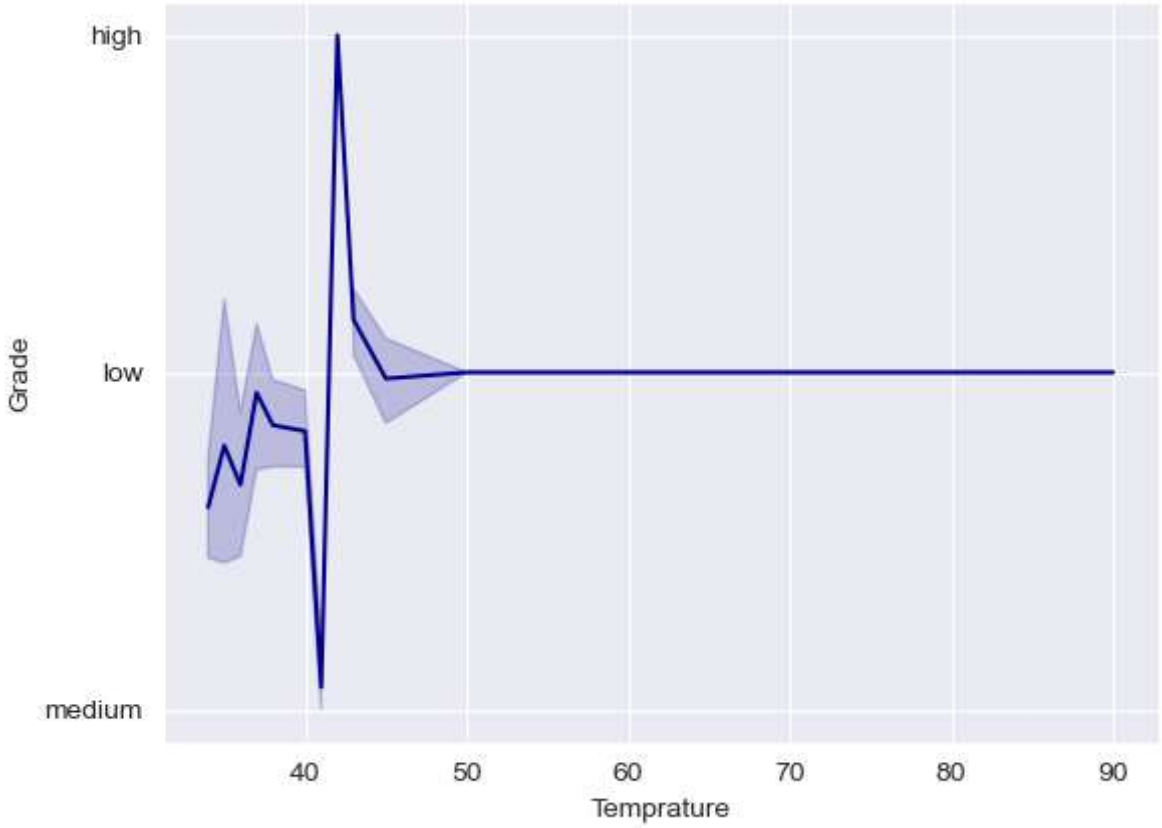
Out[23]:
```
Text(0.5, 1.0, 'Odor v/s Grade')
```



Temprature v/s Grade using Lineplot

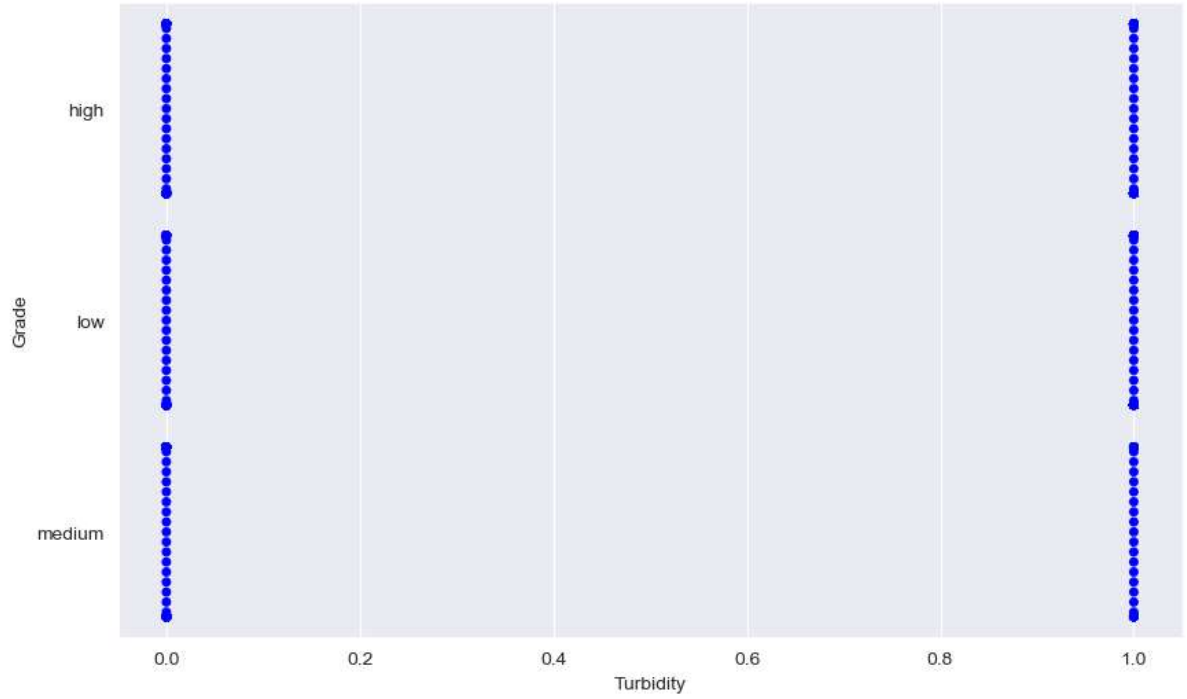
In [24]:
```
sns.lineplot(df['Temprature'],df['Grade'],color='darkblue')
```

Out[24]:
```
<AxesSubplot:xlabel='Temprature', ylabel='Grade'>
```



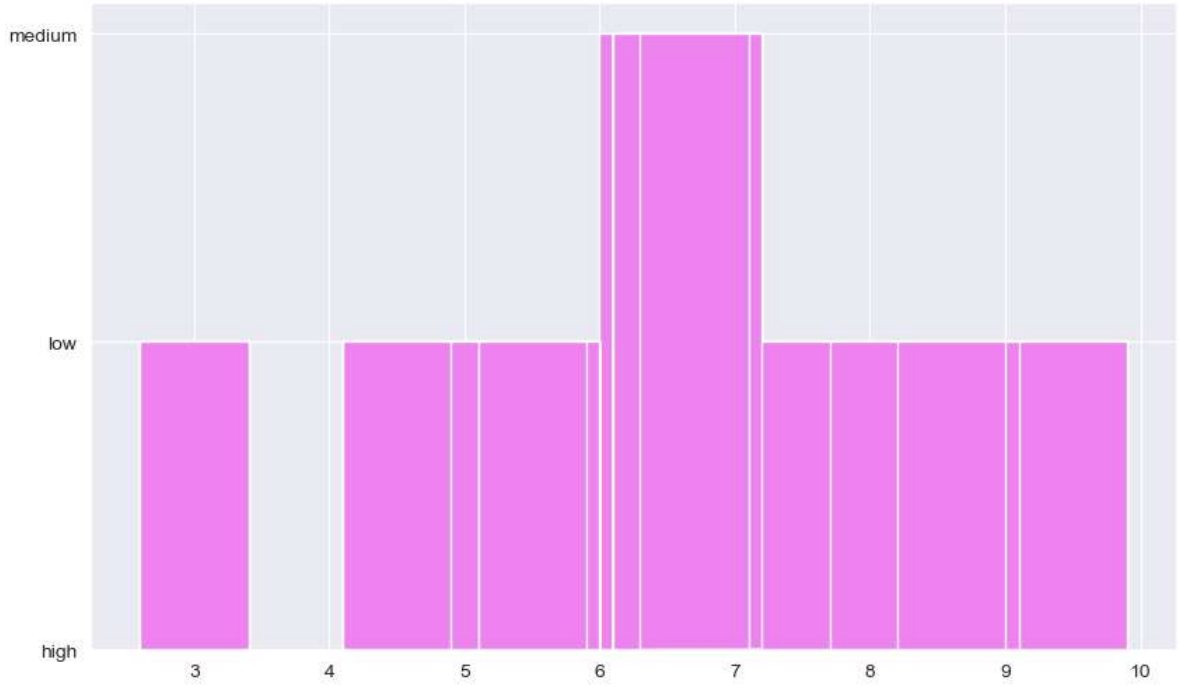
Turbidity v/s Grade using Swarmplot

In [25]:
```
plt.figure(figsize=(10,6))
sns.swarmplot(df['Turbidity'],df['Grade'],color='blue')
plt.show()
```



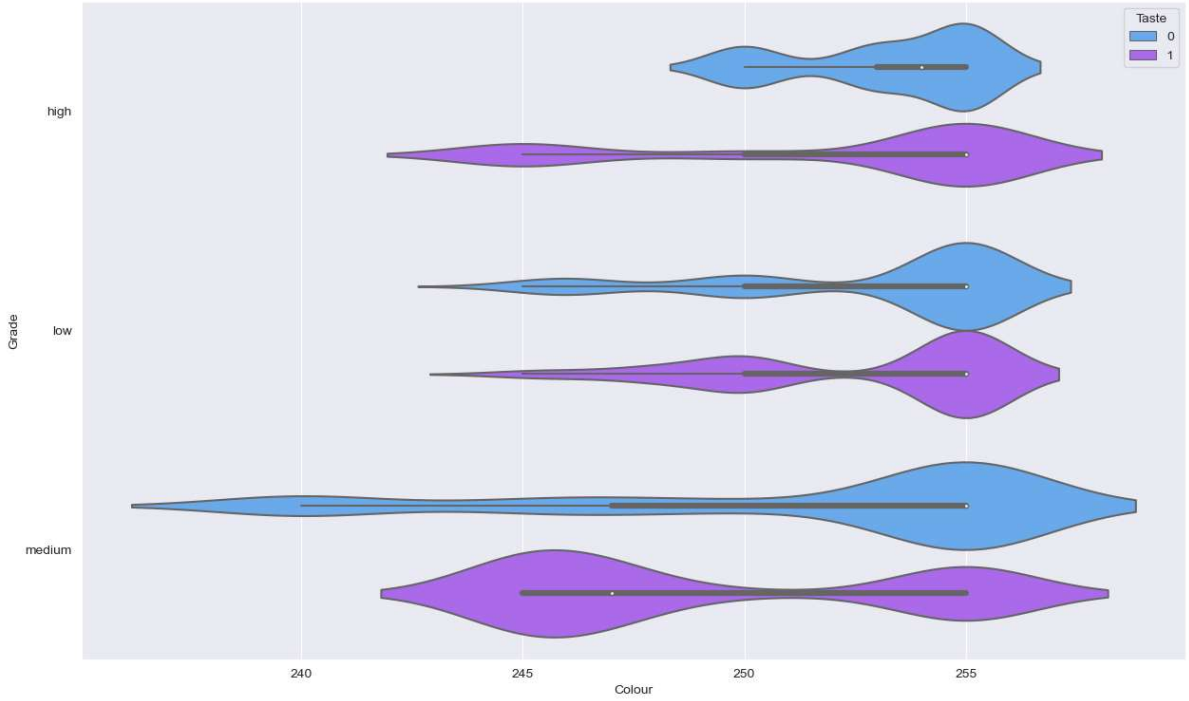
pH v/s Grade using Barplot

In [26]:
```
plt.figure(figsize=(10,6))
plt.bar(df['pH'],df['Grade'],color='violet')
plt.show()
```



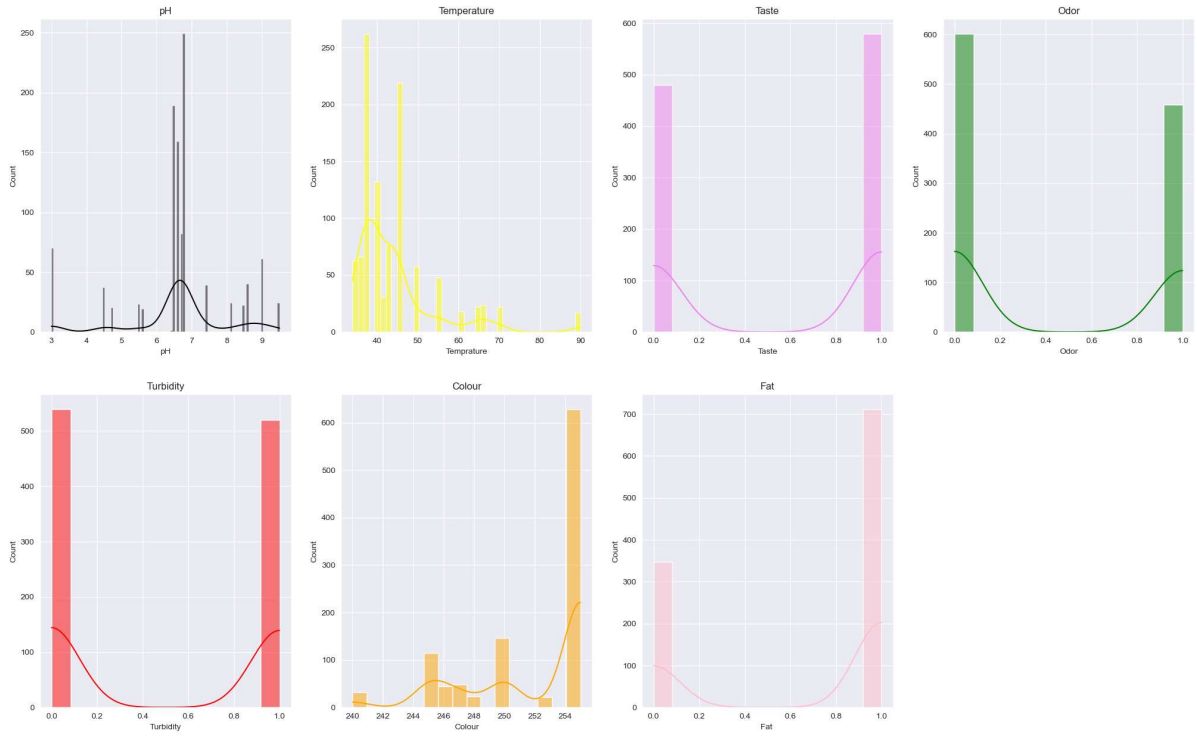
Taste v/s Grade using Scatterplot

In [27]:
```
plt.figure(figsize=(15,9))
sns.violinplot(df['Colour'],df['Grade'],hue=df['Taste'],palette='cool')
plt.show()
```

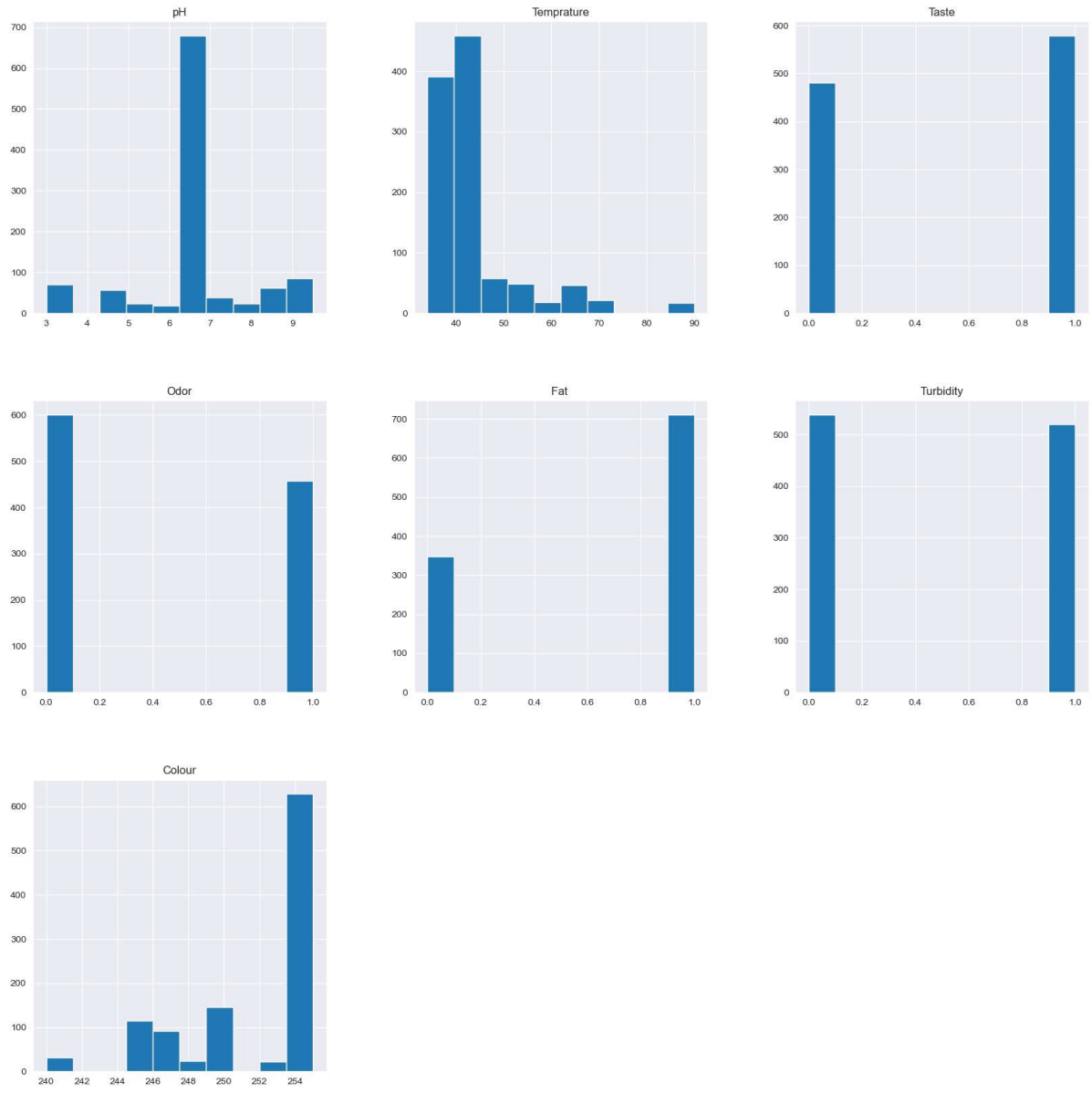


In [28]:
```
plt.figure(figsize=(25,15))

plt.subplot(2,4,1)
sns.histplot(df['pH'], color = 'black', kde = True).set_title('pH')

plt.subplot(2,4,2)
sns.histplot(df['Temprature'], kde = True, color = 'yellow').set_title('Temper

plt.subplot(2,4,3)
sns.histplot(df['Taste'], kde = True, color = 'violet').set_title('Taste')

plt.subplot(2,4,4)
sns.histplot(df['Odor'], kde = True, color = 'green').set_title('Odor')

plt.subplot(2,4,7)
sns.histplot(df['Fat '], kde = True, color = 'pink').set_title('Fat')

plt.subplot(2,4,5)
sns.histplot(df['Turbidity'], kde = True, color = 'red').set_title('Turbidity'

plt.subplot(2,4,6)
sns.histplot(df['Colour'], kde = True, color = 'orange').set_title('Colour')
```

Out[28]: Text(0.5, 1.0, 'Colour')

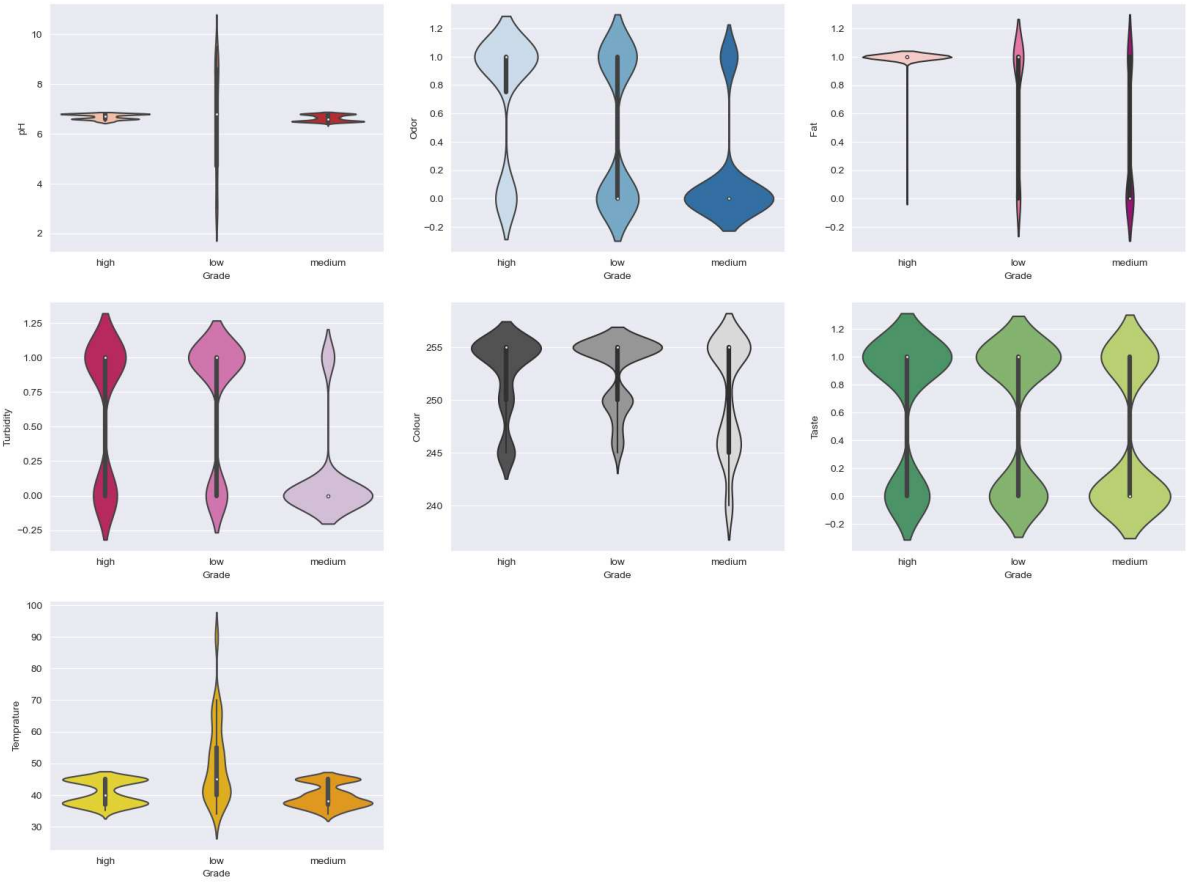


In [29]:
```
df.hist(figsize=(20,20))
plt.show()
```



In [30]:
```
plt.figure(figsize=(20,15))

plt.subplot(3,3,1)
sns.violinplot(x = 'Grade', y = 'pH', data = df, palette="Reds")

plt.subplot(3,3,7)
sns.violinplot(x = 'Grade', y = 'Temprature', data = df, palette="Wistia")

plt.subplot(3,3,6)
sns.violinplot(x = 'Grade', y = 'Taste', data = df, palette="summer")

plt.subplot(3,3,2)
sns.violinplot(x = 'Grade', y = 'Odor', data = df, palette="Blues")

plt.subplot(3,3,3)
sns.violinplot(x = 'Grade', y = 'Fat ', data = df, palette="RdPu")

plt.subplot(3,3,4)
sns.violinplot(x = 'Grade', y = 'Turbidity', data = df, palette="PuRd_r")

plt.subplot(3,3,5)
sns.violinplot(x = 'Grade', y = 'Colour', data = df, palette="Greys_r")
```

Out[30]: <AxesSubplot:xlabel='Grade', ylabel='Colour'>



We performing following operations

Logestic Regression

```
Decision Tree CLassifier

Random Forest Classifier

KNeighbors Classifier
```

Data Preparation

In [31]:
```
X=df.drop('Grade',axis=1) #independent vaiables, Predictors
```

In [32]:
```
y=df.Grade #dependent variable, prediction done on this column
```

In [33]:
```
X
```

Out[33]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
0	6.6	35	1	0	1	0	254
1	6.6	36	0	1	0	1	253
2	8.5	70	1	1	1	1	246
3	9.5	34	1	1	0	1	255
4	6.6	37	0	0	0	0	255
...	...	...	...	...	...	...	...
1054	6.7	45	1	1	0	0	247
1055	6.7	38	1	0	1	0	255
1056	3.0	40	1	1	1	1	255
1057	6.8	43	1	0	1	0	250
1058	8.6	55	0	1	1	1	255

1059 rows × 7 columns

In [34]:
```
y
```

Out[34]:
```
0         high
1         high
2          low
3          low
4       medium
         ...
1054    medium
1055      high
1056       low
1057      high
1058       low
Name: Grade, Length: 1059, dtype: object
```

Model Building

In [35]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state
```

In [36]:
```
X_train.head()
```

Out[36]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
5	6.6	37	1	1	1	1	255
310	6.6	37	1	1	1	1	255
557	4.7	38	1	0	1	0	255
104	5.6	50	0	1	1	1	255
97	6.5	37	0	0	0	0	255

In [37]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [38]:
```
X_train_std=scaler.fit_transform(X_train)
X_test_std=scaler.transform(X_test)
```

In [39]:
```
X_train_std
```

Out[39]:
```
array([[-0.01298657, -0.73413845,  0.93338744, ...,  0.66666667,
         0.98792712,  0.72665791],
       [-0.01298657, -0.73413845,  0.93338744, ...,  0.66666667,
         0.98792712,  0.72665791],
       [-1.35738312, -0.63281668,  0.93338744, ...,  0.66666667,
        -1.01222041,  0.72665791],
       ...,
       [-0.08374428, -0.83546022, -1.07136646, ...,  0.66666667,
        -1.01222041,  0.72665791],
       [-0.01298657,  0.58304458, -1.07136646, ..., -1.5       ,
         0.98792712, -0.47043749],
       [ 0.12852886,  0.07643572,  0.93338744, ...,  0.66666667,
        -1.01222041, -1.66753288]])
```

In [40]:
```
X_train
```

Out[40]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
5	6.6	37	1	1	1	1	255
310	6.6	37	1	1	1	1	255
557	4.7	38	1	0	1	0	255
104	5.6	50	0	1	1	1	255
97	6.5	37	0	0	0	0	255
...	...	...	...	...	...	...	...
330	9.0	43	1	0	1	1	250
466	6.7	45	1	1	1	0	245
121	6.5	36	0	0	1	0	255
1044	6.6	50	0	0	0	1	250
860	6.8	45	1	1	1	0	245

741 rows × 7 columns

In [41]:
```
y_train.head()
```

Out[41]:
```
5        high
310      high
557       low
104       low
97     medium
Name: Grade, dtype: object
```

In [42]:
```
df.columns
```

Out[42]:
```
Index(['pH', 'Temprature', 'Taste', 'Odor', 'Fat ', 'Turbidity', 'Colour',
       'Grade'],
      dtype='object')
```

Logistic Regression

- Logistic regression is a popular statistical model used for binary classification problems, where the dependent variable (also called the target or outcome variable) takes on two possible values, typically represented as 0 and 1. It is named after the logistic function used in the model.
- The logistic regression model estimates the probability of the dependent variable belonging to a certain class based on one or more independent variables (also known as predictors or features). The model uses a logistic or sigmoid function to transform the linear combination of the predictors into a probability value between 0 and 1.

In [43]:
```
#import Linear regression
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

In [44]:
```
lr.fit(X_train,y_train) #fitting the training and testing data
```

Out[44]:
```
LogisticRegression()
```

In [45]:
```
print('Training accuracy score',lr.score(X_train,y_train)) #accuracy of traini
print("Testing accuracy Score",lr.score(X_test,y_test)) #accuracy of test data
```

```
Training accuracy score 0.7206477732793523
Testing accuracy Score 0.6981132075471698
```

Confusion Matrix

- A `confusion matrix` is a table used to describe the performance of a classification model by displaying the counts of true positive, true negative, false positive, and false negative predictions. It is a useful tool for evaluating the accuracy of a classification algorithm and understanding the types of errors it makes.
- The confusion matrix allows you to calculate various performance metrics, including `accuracy` , `precision` , `recall (sensitivity)` , `specificity` , and `F1 score` .

In [46]:
```
from sklearn import metrics
```

In [47]:
```
confusion_matrix = metrics.confusion_matrix
```

In [48]:
```
y_pred=lr.predict(X_test)
```

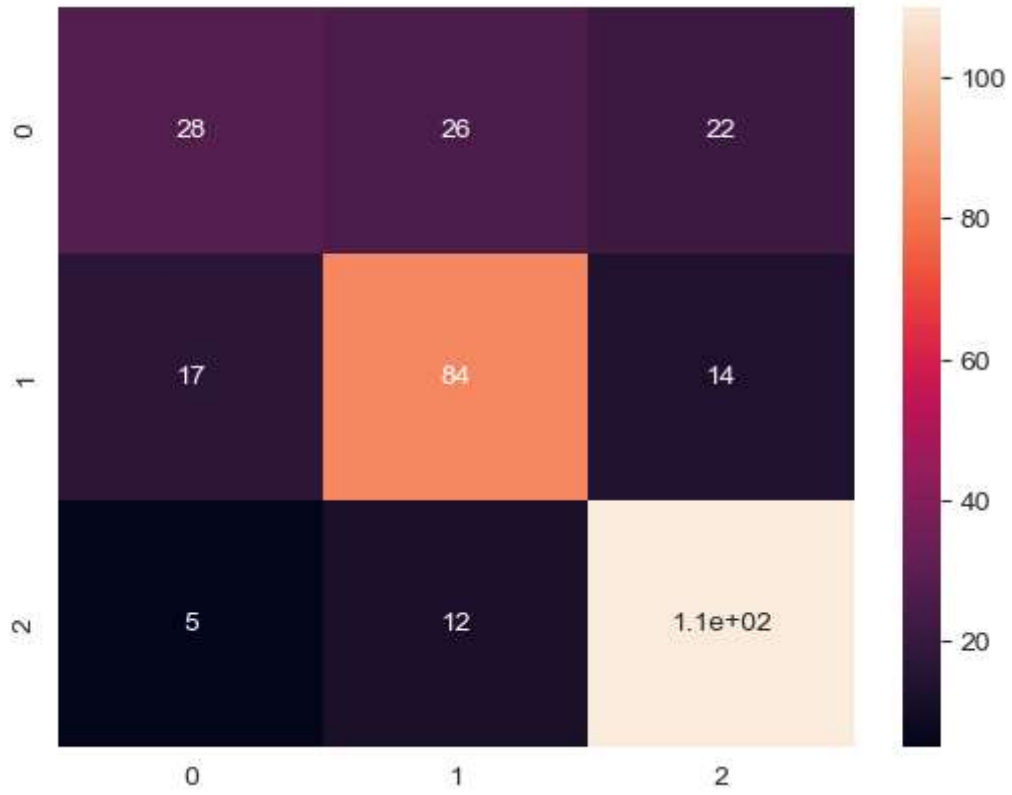
In [49]:
```
cf_matrix=confusion_matrix(y_test,y_pred)
```

In [50]:
```
cf_matrix
```

Out[50]:
```
array([[ 28,  26,  22],
       [ 17,  84,  14],
       [  5,  12, 110]], dtype=int64)
```

In [51]:
```
sns.heatmap(cf_matrix,annot=True)
```

Out[51]: <AxesSubplot:>



Classifiaction Report

In [52]:
```
from sklearn.metrics import classification_report
```

In [53]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

        high       0.56      0.37      0.44        76
         low       0.69      0.73      0.71       115
      medium       0.75      0.87      0.81       127

    accuracy                           0.70       318
   macro avg       0.67      0.65      0.65       318
weighted avg       0.68      0.70      0.68       318
```

Prediction

In [54]:
```
print(y_pred)
```

```
['low' 'low' 'low' 'low' 'medium' 'low' 'medium' 'low' 'high' 'medium'
 'low' 'medium' 'medium' 'medium' 'low' 'medium' 'low' 'medium' 'medium'
 'medium' 'high' 'medium' 'medium' 'medium' 'medium' 'medium' 'low' 'high'
 'medium' 'low' 'medium' 'medium' 'high' 'medium' 'low' 'medium' 'medium'
 'low' 'medium' 'medium' 'low' 'low' 'high' 'medium' 'high' 'high'
 'medium' 'medium' 'medium' 'medium' 'medium' 'low' 'high' 'medium'
 'medium' 'low' 'medium' 'low' 'medium' 'medium' 'medium' 'high' 'low'
 'high' 'low' 'high' 'medium' 'low' 'low' 'low' 'low' 'low' 'low' 'medium'
 'low' 'medium' 'medium' 'medium' 'low' 'low' 'medium' 'medium' 'low'
 'high' 'medium' 'medium' 'high' 'low' 'medium' 'low' 'low' 'low' 'low'
 'medium' 'high' 'low' 'low' 'medium' 'low' 'medium' 'low' 'low' 'low'
 'low' 'medium' 'medium' 'low' 'low' 'medium' 'low' 'low' 'medium'
 'medium' 'low' 'high' 'low' 'medium' 'low' 'high' 'high' 'medium' 'low'
 'low' 'low' 'medium' 'low' 'medium' 'medium' 'high' 'medium' 'medium'
 'medium' 'low' 'medium' 'low' 'high' 'medium' 'medium' 'medium' 'medium'
 'medium' 'low' 'low' 'medium' 'low' 'low' 'medium' 'low' 'low' 'medium'
 'medium' 'medium' 'medium' 'low' 'medium' 'high' 'low' 'high' 'low' 'low'
 'medium' 'medium' 'medium' 'low' 'medium' 'low' 'low' 'low' 'medium'
 'medium' 'high' 'medium' 'low' 'low' 'medium' 'low' 'medium' 'medium'
 'medium' 'low' 'low' 'medium' 'high' 'low' 'high' 'medium' 'medium'
 'high' 'medium' 'low' 'low' 'medium' 'high' 'low' 'low' 'low' 'low' 'low'
 'medium' 'medium' 'high' 'medium' 'high' 'high' 'medium' 'low' 'low'
 'high' 'low' 'medium' 'high' 'low' 'medium' 'low' 'high' 'low' 'low'
 'high' 'medium' 'low' 'medium' 'medium' 'medium' 'medium' 'low' 'medium'
 'medium' 'high' 'medium' 'high' 'medium' 'high' 'medium' 'high' 'medium'
 'low' 'medium' 'medium' 'medium' 'medium' 'low' 'low' 'low' 'low' 'low'
 'medium' 'low' 'medium' 'high' 'low' 'low' 'high' 'medium' 'high'
 'medium' 'medium' 'low' 'medium' 'low' 'high' 'medium' 'high' 'medium'
 'high' 'medium' 'low' 'high' 'medium' 'medium' 'medium' 'medium' 'medium'
 'low' 'medium' 'medium' 'low' 'low' 'low' 'medium' 'low' 'medium' 'high'
 'low' 'medium' 'low' 'low' 'low' 'medium' 'low' 'medium' 'low' 'low'
 'medium' 'high' 'medium' 'low' 'high' 'high' 'medium' 'medium' 'medium'
 'high' 'low' 'medium' 'low' 'low' 'low' 'high' 'medium' 'medium' 'medium'
 'medium' 'medium' 'medium' 'medium' 'low' 'low' 'medium']
```

Decision Tree Classifier

- Two criterion is there in this
- nut we using only one i.e.criterion entropy

In [55]:
```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(criterion='entropy',max_depth=3,random_state=42)
```

In [56]:
```
dtc.fit(X_train,y_train)
```

Out[56]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
```

In [57]:
```
print("Training accuracy sciore",dtc.score(X_train,y_train))
print("testing Accuracy score",dtc.score(X_test,y_test))
```

```
Training accuracy sciore 0.7557354925775979
testing Accuracy score 0.7547169811320755
```

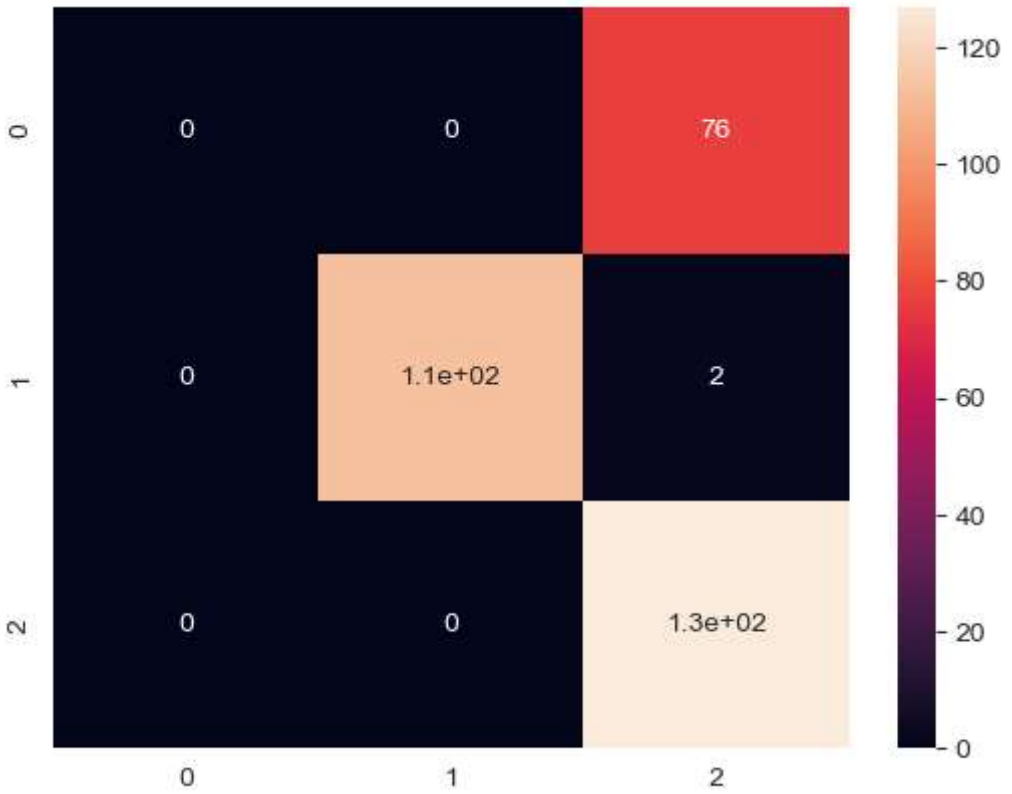
Confusion Matrix

In [58]:
```
y_pred=dtc.predict(X_test)
confusion_matrix(y_test,y_pred)
```

Out[58]:
```
array([[  0,   0,  76],
       [  0, 113,   2],
       [  0,   0, 127]], dtype=int64)
```

In [59]:
```
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
```

Out[59]: <AxesSubplot:>



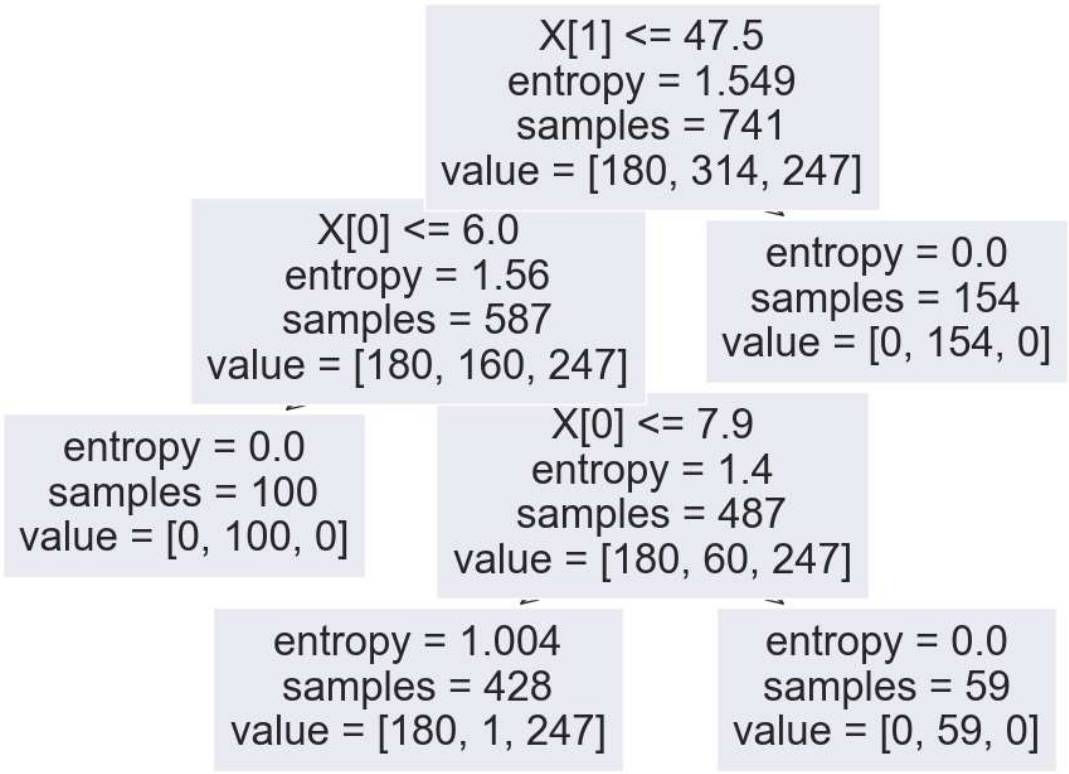
Classification Report

In [60]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

        high       0.00      0.00      0.00        76
         low       1.00      0.98      0.99       115
      medium       0.62      1.00      0.77       127

    accuracy                           0.75       318
   macro avg       0.54      0.66      0.59       318
weighted avg       0.61      0.75      0.66       318
```

Visualization Of Decision Tree

In [61]:
```
from sklearn import tree
plt.figure(figsize=(12,8))
tree.plot_tree(dtc.fit(X_train,y_train))
plt.show()
```



Random Forest

- `Random Forest` is a popular machine learning algorithm used for both regression and classification tasks. It is an ensemble learning method that combines multiple decision trees

to make predictions. Each decision tree is built on a randomly sampled subset of the data and features, hence the name `random forest`.

In [62]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
```

In [63]:
```
rfc.fit(X_train,y_train)
```

Out[63]:
```
RandomForestClassifier()
```

In [64]:
```
print("training accuracy score",rfc.score(X_train,y_train))
print("testing accuracy score",rfc.score(X_test,y_test))
```

```
training accuracy score 1.0
testing accuracy score 0.9968553459119497
```

In [65]:
```
y_pred=rfc.predict(X_test)
```

In [66]:
```
from sklearn.metrics import accuracy_score
print("model accuracy score with 100 decision trees:{0:0.4f}".format(accuracy_
```

```
model accuracy score with 100 decision trees:0.9969
```

Random Forest with n_estimatos =100

In [67]:
```
rfc_100=RandomForestClassifier(n_estimators=100)
```

In [68]:
```
rfc_100.fit(X_train,y_train)
```

Out[68]:
```
RandomForestClassifier()
```

In [69]:
```
rfc_100_pred=rfc.predict(X_test)
```

In [70]:
```
from sklearn.metrics import accuracy_score
print("model accuracy score with 100 decision trees:{0:0.4f}".format(accuracy_
```

```
model accuracy score with 100 decision trees:0.9969
```

In [71]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

        high       0.99      1.00      0.99        76
         low       1.00      0.99      1.00       115
      medium       1.00      1.00      1.00       127

    accuracy                           1.00       318
   macro avg       1.00      1.00      1.00       318
weighted avg       1.00      1.00      1.00       318
```

In [72]:
```
print(classification_report(y_test,rfc_100_pred))
```

```
              precision    recall  f1-score   support

        high       0.99      1.00      0.99        76
         low       1.00      0.99      1.00       115
      medium       1.00      1.00      1.00       127

    accuracy                           1.00       318
   macro avg       1.00      1.00      1.00       318
weighted avg       1.00      1.00      1.00       318
```

In [73]:
```
confusion_matrix(y_test,y_pred)
```

Out[73]:
```
array([[ 76,   0,   0],
       [  1, 114,   0],
       [  0,   0, 127]], dtype=int64)
```

KNeighborsClassifiers

- The k-Nearest Neighbors (k-NN) algorithm is a simple yet effective classification algorithm that predicts the class of a new sample based on its similarity to the k nearest neighbors in the training dataset. It is a non-parametric and lazy learning algorithm, meaning it doesn't make any assumptions about the underlying data distribution and doesn't explicitly build a model during the training phase.

In [74]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
```

In [75]:
```
knn.fit(X_train,y_train)
```

Out[75]:
```
KNeighborsClassifier()
```

In [76]:
```
print("training Accuracy score",knn.score(X_train,y_train))
print("testinhg accuracy score",knn.score(X_test,y_test))
```

```
training Accuracy score 0.9919028340080972
testinhg accuracy score 0.9874213836477987
```

In [77]:
```
knn_pred=knn.predict(X_test)
print("accuracy score",accuracy_score(y_test,knn_pred))
```

```
accuracy score 0.9874213836477987
```

In [78]:
```
confusion_matrix(y_test,knn_pred)
```

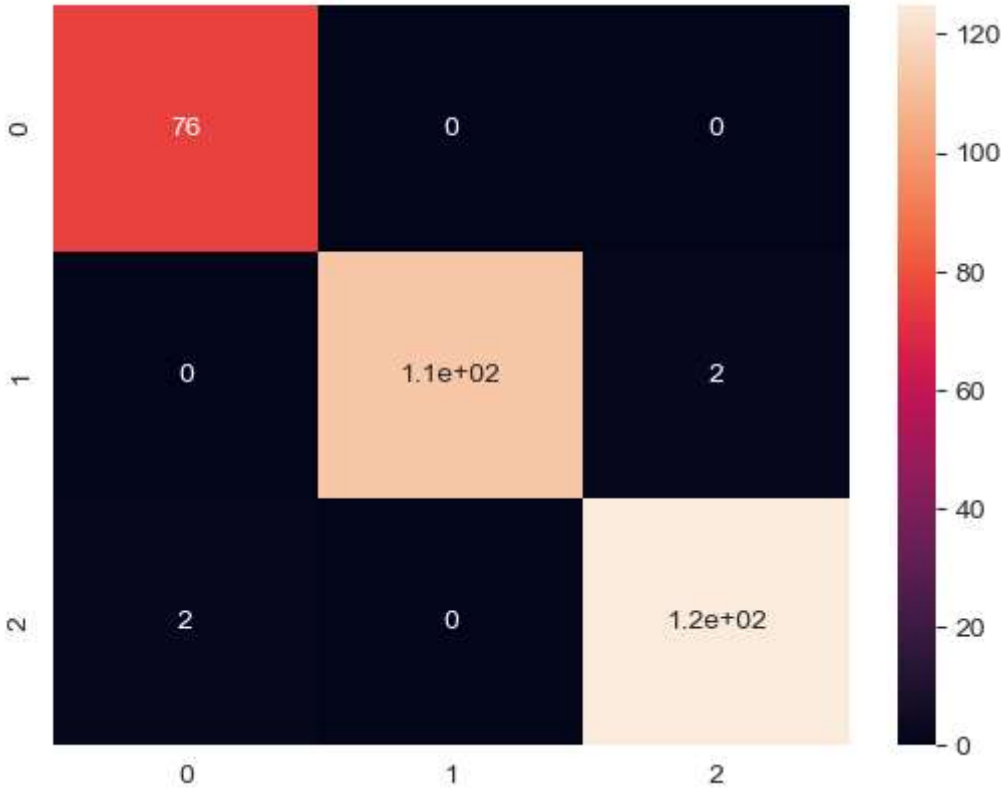
Out[78]:
```
array([[ 76,   0,   0],
       [  0, 113,   2],
       [  2,   0, 125]], dtype=int64)
```

In [79]:
```
print(classification_report(y_test,knn_pred))
```

```
              precision    recall  f1-score   support

        high       0.97      1.00      0.99        76
         low       1.00      0.98      0.99       115
      medium       0.98      0.98      0.98       127

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318
```

In [80]:
```
sns.heatmap(confusion_matrix(y_test,knn_pred),annot=True)
```

Out[80]: <AxesSubplot:>



In [81]:
```
print(dtc.predict(X_test))
```

```
['medium' 'low' 'low' 'medium' 'medium' 'low' 'low' 'medium' 'low'
 'medium' 'low' 'medium' 'medium' 'medium' 'low' 'low' 'medium' 'medium'
 'medium' 'medium' 'low' 'medium' 'medium' 'medium' 'medium' 'medium'
 'medium' 'medium' 'low' 'low' 'medium' 'medium' 'medium' 'medium'
 'medium' 'medium' 'medium' 'medium' 'medium' 'medium' 'low' 'low' 'low'
 'medium' 'medium' 'low' 'medium' 'medium' 'medium' 'medium' 'medium'
 'low' 'low' 'medium' 'medium' 'low' 'medium' 'low' 'low' 'medium'
 'medium' 'low' 'medium' 'medium' 'medium' 'low' 'medium' 'medium' 'low'
 'medium' 'low' 'low' 'medium' 'medium' 'low' 'low' 'low' 'medium' 'low'
 'medium' 'medium' 'medium' 'medium' 'medium' 'medium' 'medium' 'medium'
 'low' 'medium' 'medium' 'low' 'low' 'medium' 'medium' 'low' 'medium'
 'medium' 'medium' 'low' 'medium' 'low' 'low' 'low' 'low' 'medium'
 'medium' 'low' 'low' 'medium' 'medium' 'medium' 'medium' 'medium' 'low'
 'medium' 'low' 'medium' 'low' 'medium' 'medium' 'medium' 'low' 'low'
 'low' 'medium' 'medium' 'medium' 'medium' 'medium' 'medium' 'low'
 'medium' 'low' 'medium' 'low' 'low' 'medium' 'low' 'medium' 'medium'
 'medium' 'medium' 'low' 'medium' 'low' 'low' 'medium' 'low' 'low'
 'medium' 'medium' 'medium' 'medium' 'low' 'medium' 'low' 'low' 'medium'
 'low' 'medium' 'medium' 'medium' 'medium' 'low' 'medium' 'medium' 'low'
 'low' 'medium' 'low' 'medium' 'medium' 'low' 'low' 'medium' 'low'
 'medium' 'medium' 'medium' 'medium' 'low' 'medium' 'medium' 'low'
 'medium' 'low' 'medium' 'medium' 'medium' 'low' 'low' 'medium' 'medium'
 'low' 'medium' 'low' 'low' 'medium' 'medium' 'medium' 'medium' 'medium'
 'medium' 'medium' 'medium' 'low' 'medium' 'medium' 'medium' 'medium'
 'medium' 'medium' 'medium' 'medium' 'medium' 'low' 'low' 'medium'
 'medium' 'low' 'medium' 'medium' 'medium' 'medium' 'low' 'medium' 'low'
 'low' 'medium' 'medium' 'medium' 'low' 'medium' 'low' 'medium' 'low'
 'medium' 'low' 'medium' 'low' 'low' 'medium' 'medium' 'low' 'low'
 'medium' 'low' 'medium' 'low' 'low' 'low' 'medium' 'medium' 'medium'
 'medium' 'medium' 'low' 'medium' 'low' 'low' 'medium' 'medium' 'medium'
 'medium' 'medium' 'low' 'medium' 'medium' 'medium' 'medium' 'medium'
 'medium' 'low' 'medium' 'medium' 'low' 'low' 'low' 'medium' 'medium'
 'medium' 'medium' 'medium' 'medium' 'low' 'low' 'low' 'medium' 'medium'
 'medium' 'low' 'low' 'medium' 'medium' 'medium' 'medium' 'medium'
 'medium' 'medium' 'low' 'medium' 'low' 'medium' 'medium' 'low' 'low'
 'low' 'medium' 'medium' 'medium' 'medium' 'medium' 'medium' 'medium'
 'medium' 'medium' 'low' 'medium']
```

In [ ]:
```
```