Milk Quality Prediction

Import Packages

In [1]:
```
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import warnings
warnings.simplefilter('ignore')
%matplotlib inline
import matplotlib.pyplot as plt
```

Loading and Evaluating Dataset

In [2]:
```
#loading the dataset in pandas dataframe
data = pd.read_csv("milknew.csv")
```

In [3]:
```
#check first five rows of the dataset
data.head()
```

Out[3]:

	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 [4]:
```
#check last five rows of the dataset
data.tail()
```

Out[4]:

	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 [5]:
```
#check shape of the dataset
data.shape
```

Out[5]:
```
(1059, 8)
```

In [6]:
```
#check more infomation of the dataset
data.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
```

In [7]:
```
#check mathamtic realtionship of the dataset
data.describe()
```

Out[7]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	105
mean	6.630123	44.226629	0.546742	0.432483	0.671388	0.491029	25
std	1.399679	10.098364	0.498046	0.495655	0.469930	0.500156	4
min	3.000000	34.000000	0.000000	0.000000	0.000000	0.000000	24
25%	6.500000	38.000000	0.000000	0.000000	0.000000	0.000000	25
50%	6.700000	41.000000	1.000000	0.000000	1.000000	0.000000	25
75%	6.800000	45.000000	1.000000	1.000000	1.000000	1.000000	25
max	9.500000	90.000000	1.000000	1.000000	1.000000	1.000000	25

In [8]:
```
#check corr realtionship of the dataset
data.corr()
```

Out[8]:

	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

In [9]:
```
#check missing value of the dataset
data.isnull().sum()
```

Out[9]:
```
pH            0
Temprature    0
Taste         0
Odor          0
Fat           0
Turbidity     0
Colour        0
Grade         0
dtype: int64
```

In [10]:
```
data.loc[data["Grade"] == 'high', "Grade"] = 2
data.loc[data["Grade"] == 'medium', "Grade"] = 1
data.loc[data["Grade"] == 'low', "Grade"] = 0
data.head()
```

Out[10]:

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

In [11]:
```
data.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
```

When we look at it, "Grade" type appears as object. I convert it to integer.

In [12]:
```
data["Grade"] = data["Grade"].astype(str).astype(int)
```

In [13]:
```
data.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   int64  
dtypes: float64(1), int64(7)
memory usage: 66.3 KB
```

In [14]:
```
data.describe()
```

Out[14]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059
mean	6.630123	44.226629	0.546742	0.432483	0.671388	0.491029	251
std	1.399679	10.098364	0.498046	0.495655	0.469930	0.500156	4
min	3.000000	34.000000	0.000000	0.000000	0.000000	0.000000	240
25%	6.500000	38.000000	0.000000	0.000000	0.000000	0.000000	250
50%	6.700000	41.000000	1.000000	0.000000	1.000000	0.000000	255
75%	6.800000	45.000000	1.000000	1.000000	1.000000	1.000000	255
max	9.500000	90.000000	1.000000	1.000000	1.000000	1.000000	255

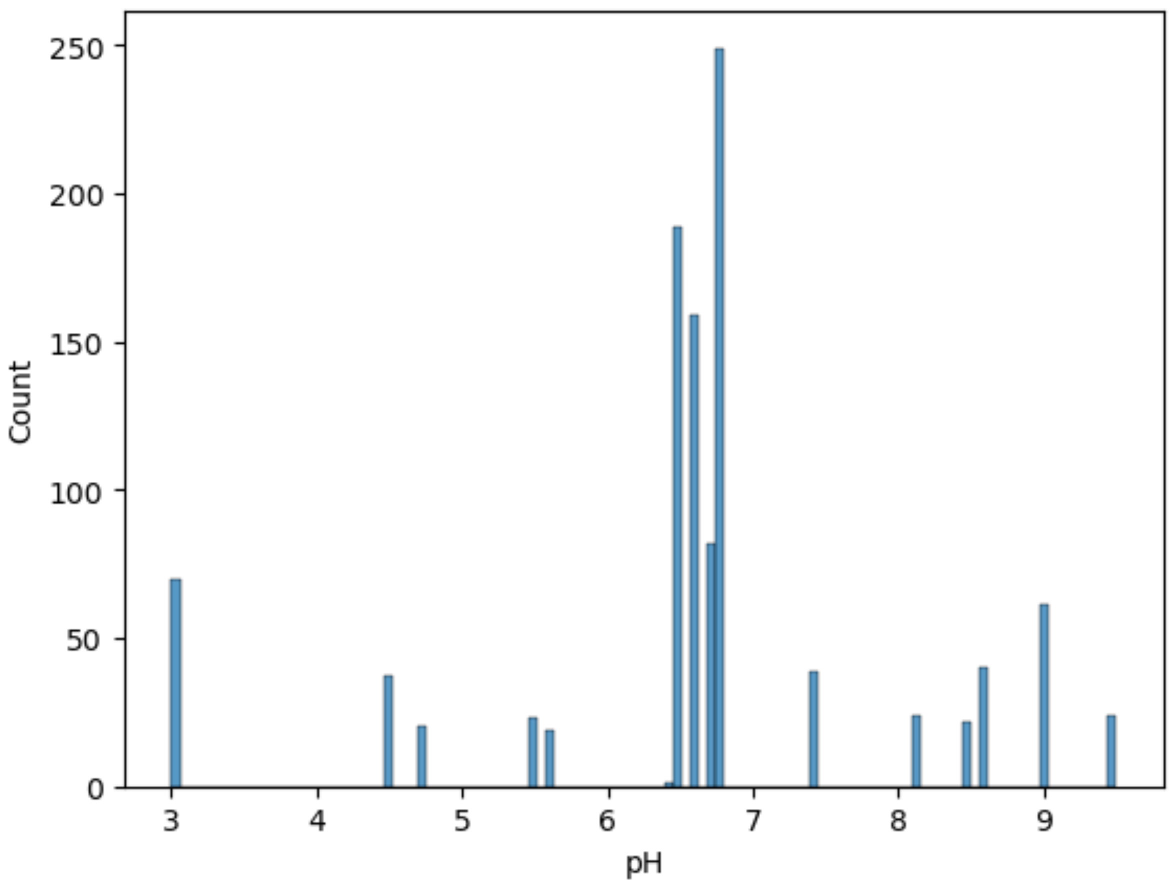
In [15]:
```
data.corr()
```

Out[15]:

	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
Grade	0.028980	-0.417789	0.025500	0.149626	0.151002	-0.153634	-0.056986

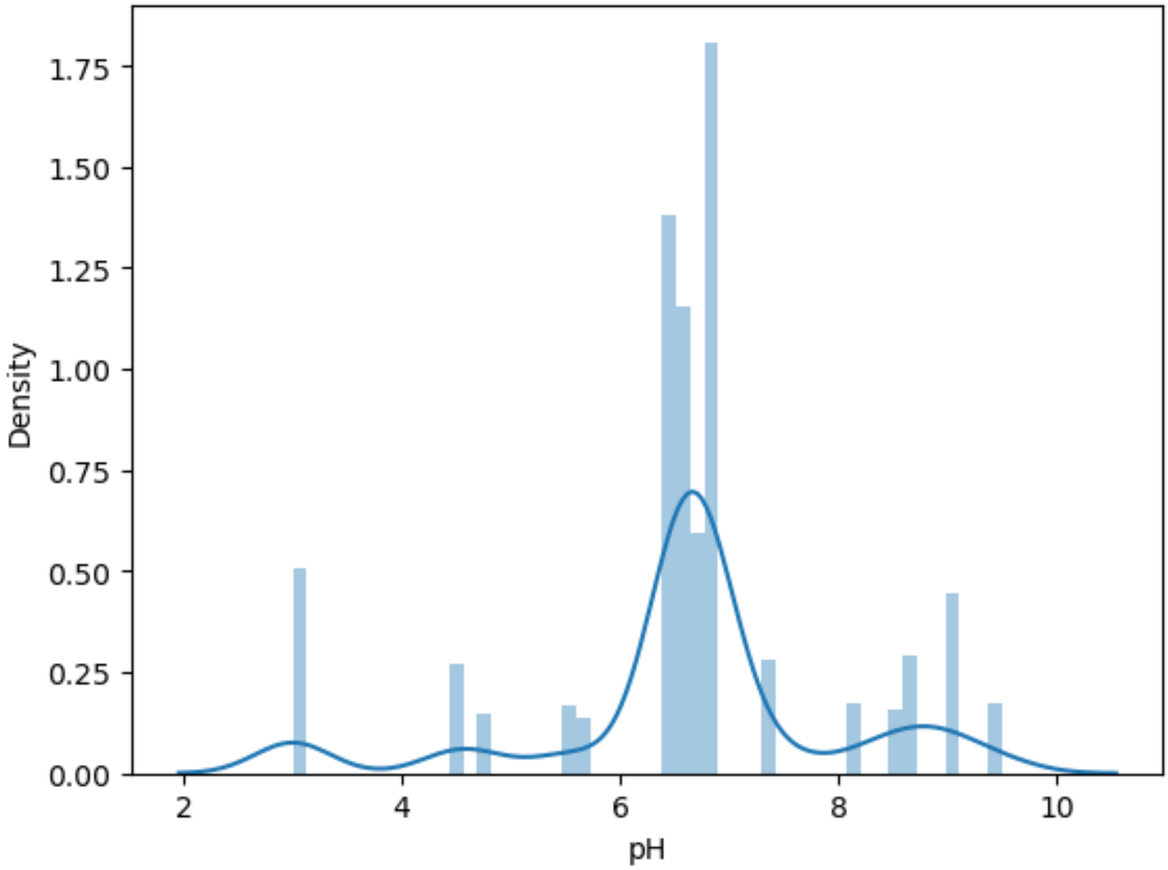
In [16]:
```
sns.histplot(data['pH'])
```

Out[16]: <AxesSubplot:xlabel='pH', ylabel='Count'>



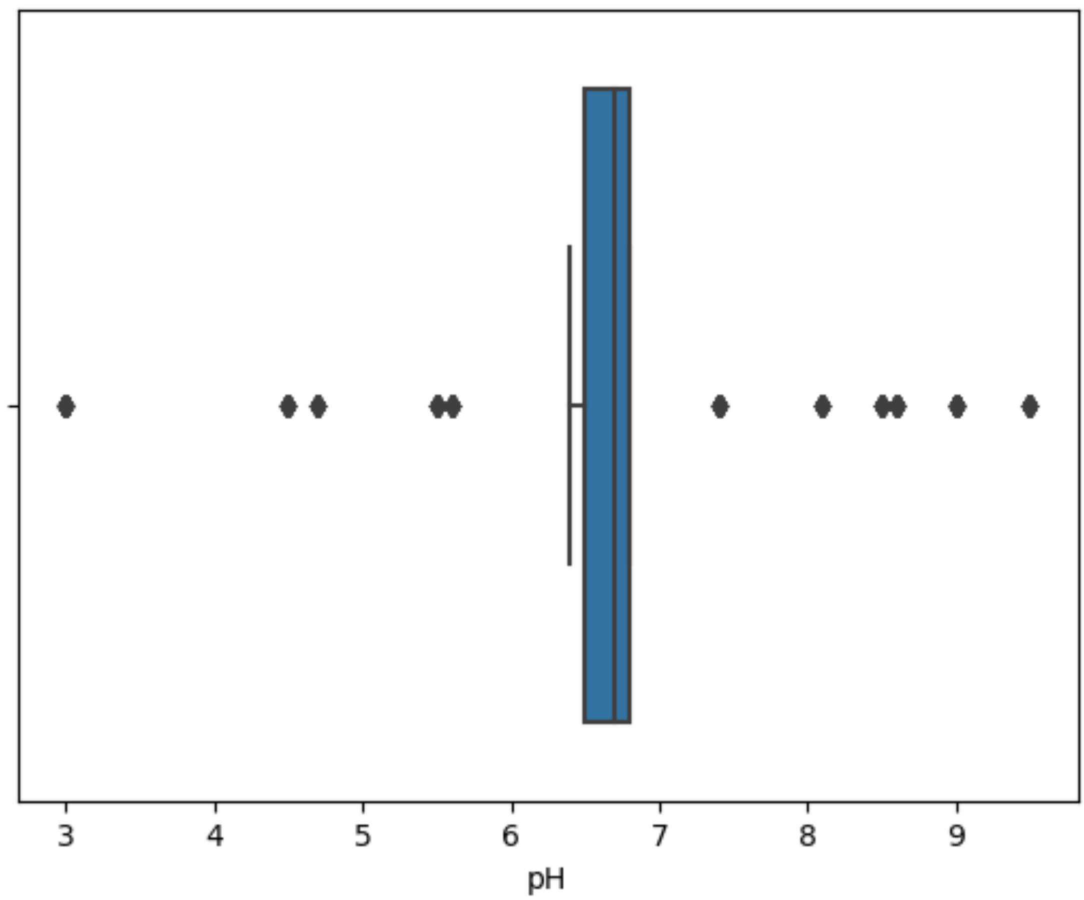
In [17]:
```
sns.distplot(data['pH'])
```

Out[17]: <AxesSubplot:xlabel='pH', ylabel='Density'>



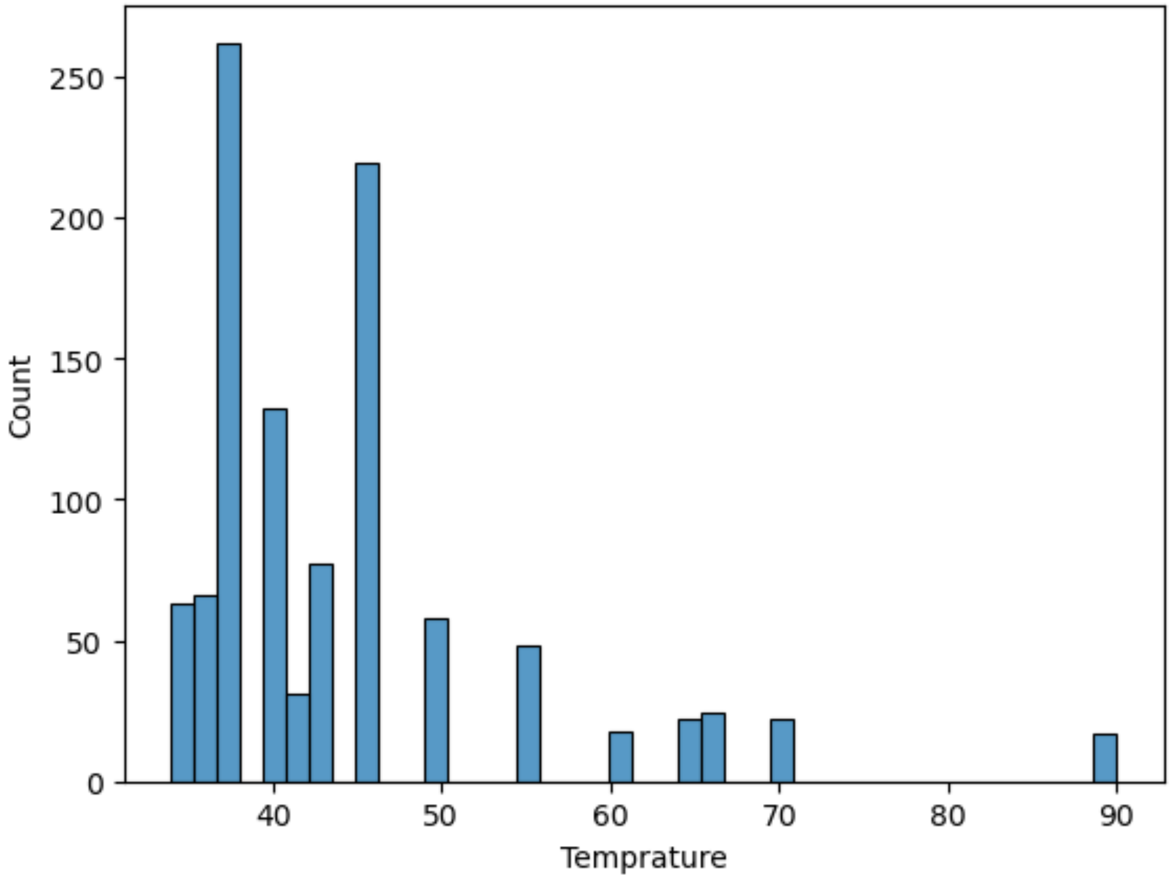
In [18]:
```
sns.boxplot(data['pH'])
```

Out[18]: <AxesSubplot:xlabel='pH'>



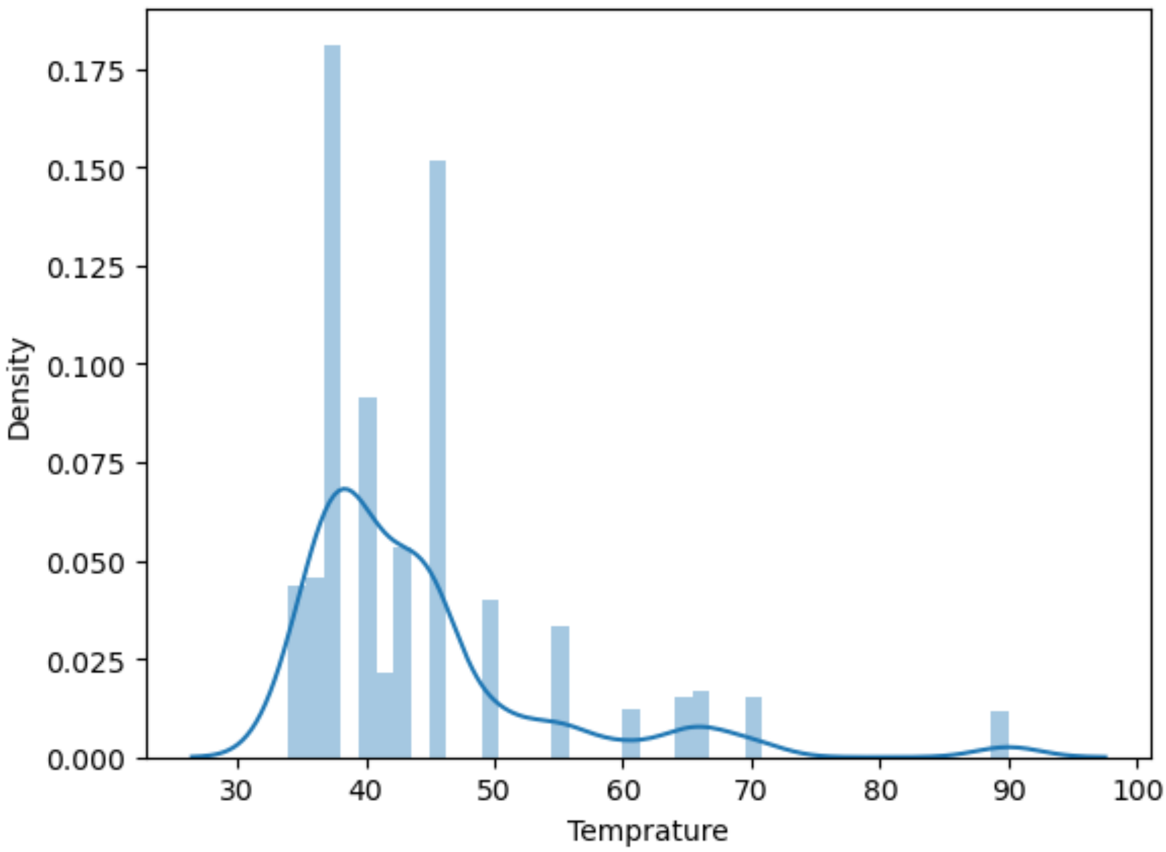
In [19]:
```
sns.histplot(data['Temprature'])
```

Out[19]:
```
<AxesSubplot:xlabel='Temprature', ylabel='Count'>
```



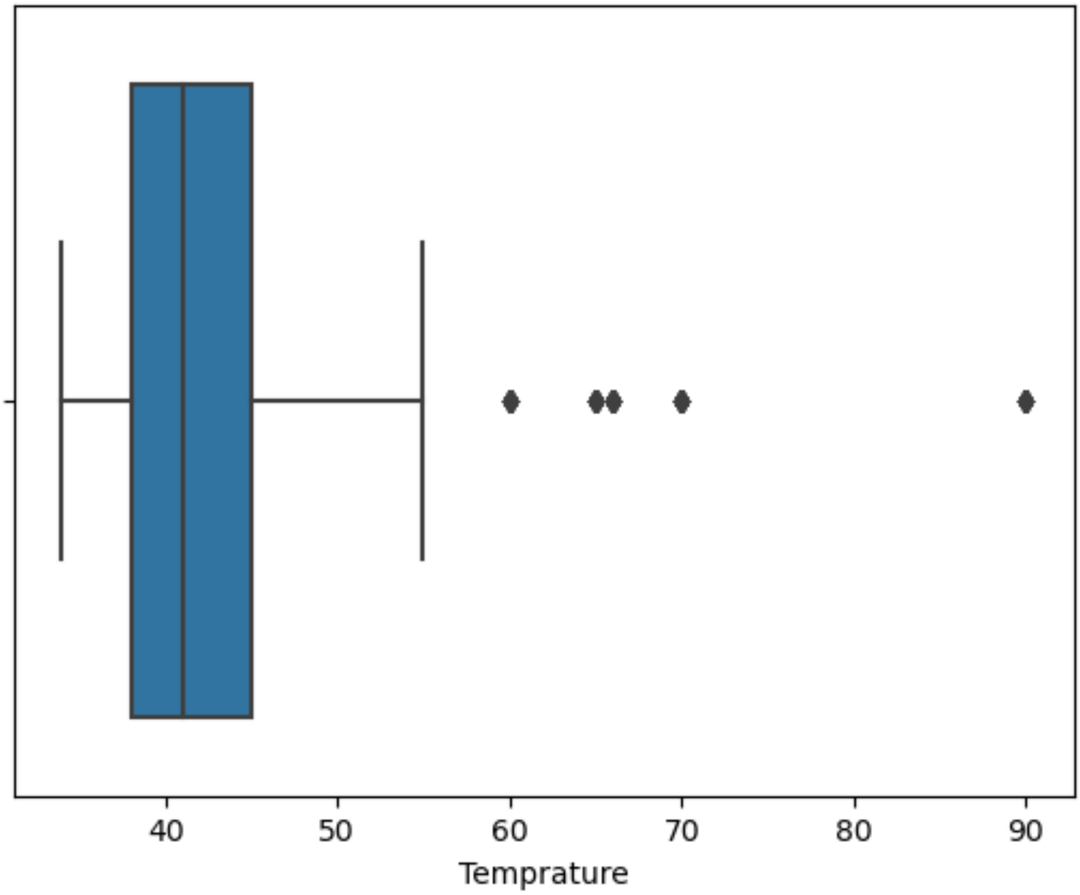
In [20]:
```
sns.distplot(data['Temprature'])
```

Out[20]: <AxesSubplot:xlabel='Temprature', ylabel='Density'>



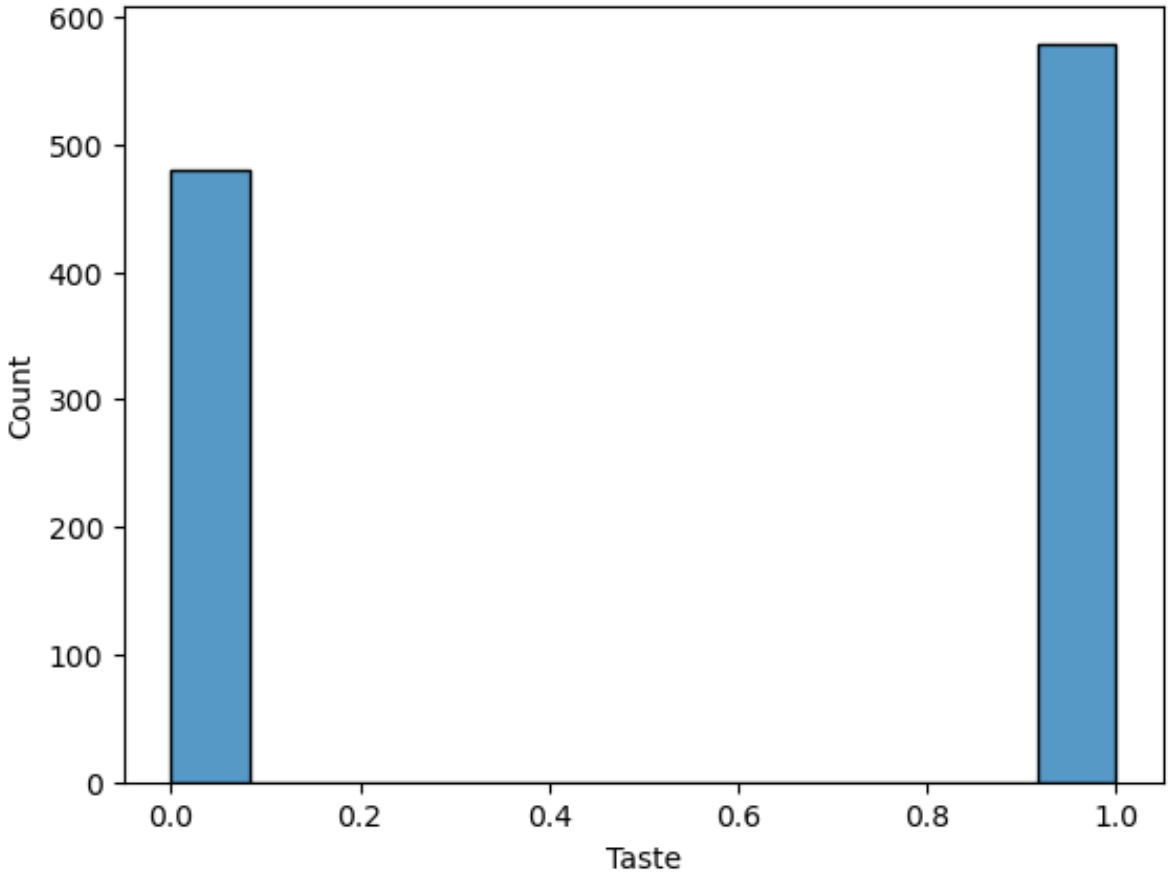
In [21]:
```
sns.boxplot(data['Temprature'])
```

Out[21]: <AxesSubplot:xlabel='Temprature'>



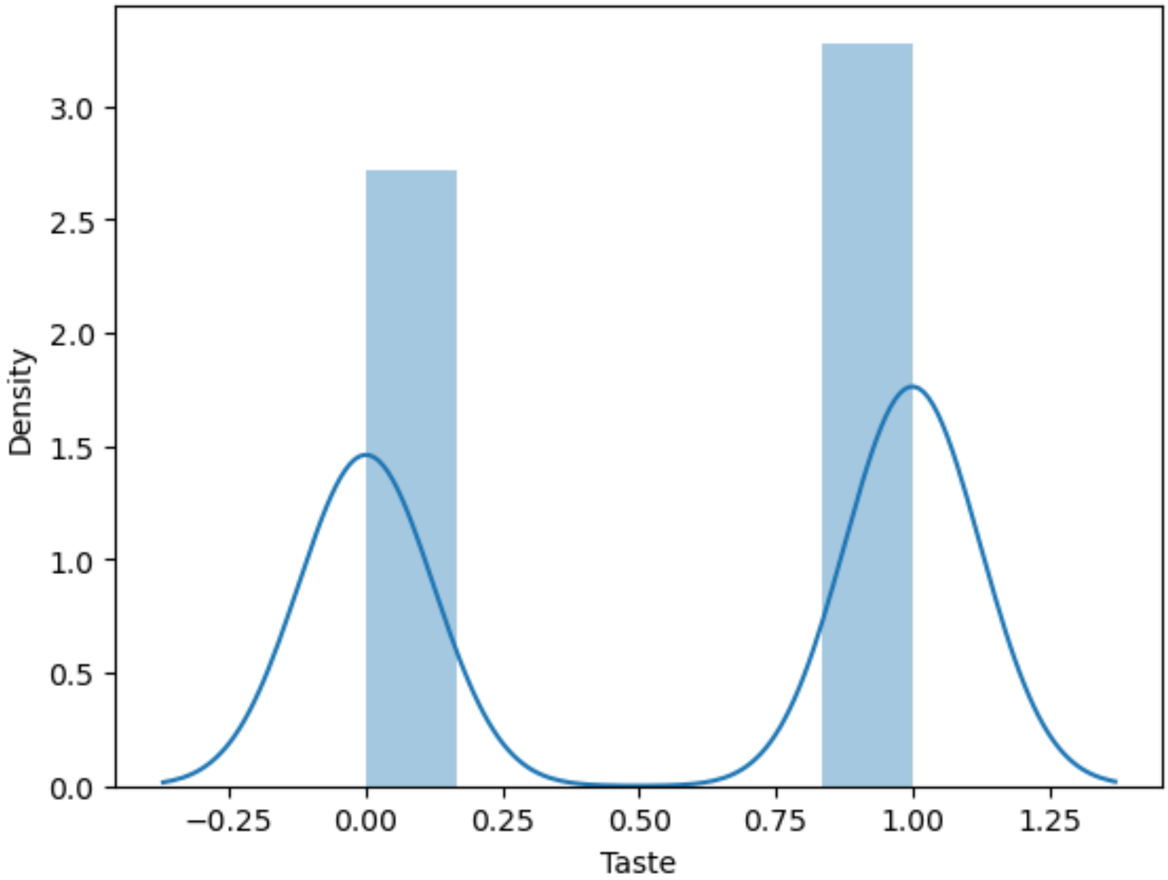
In [22]:
```
sns.histplot(data['Taste'])
```

Out[22]: <AxesSubplot:xlabel='Taste', ylabel='Count'>



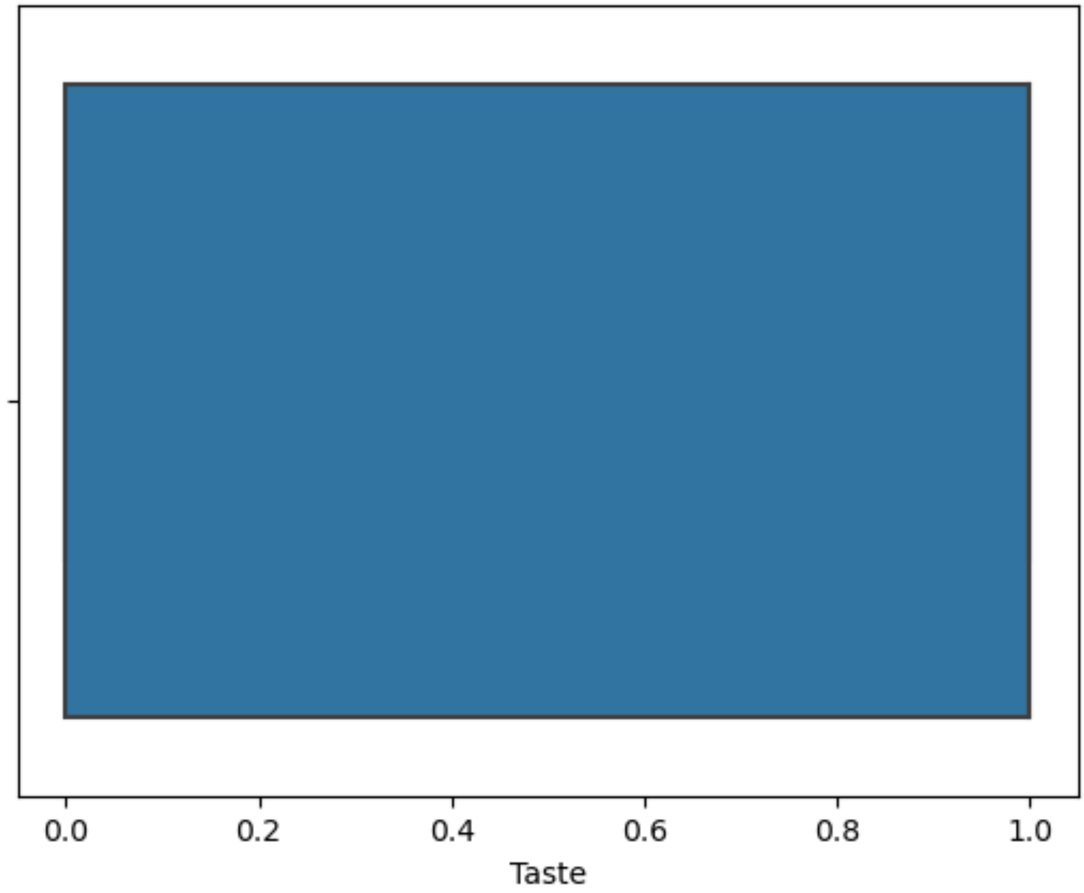
In [23]:
```
sns.distplot(data['Taste'])
```

Out[23]:
```
<AxesSubplot:xlabel='Taste', ylabel='Density'>
```



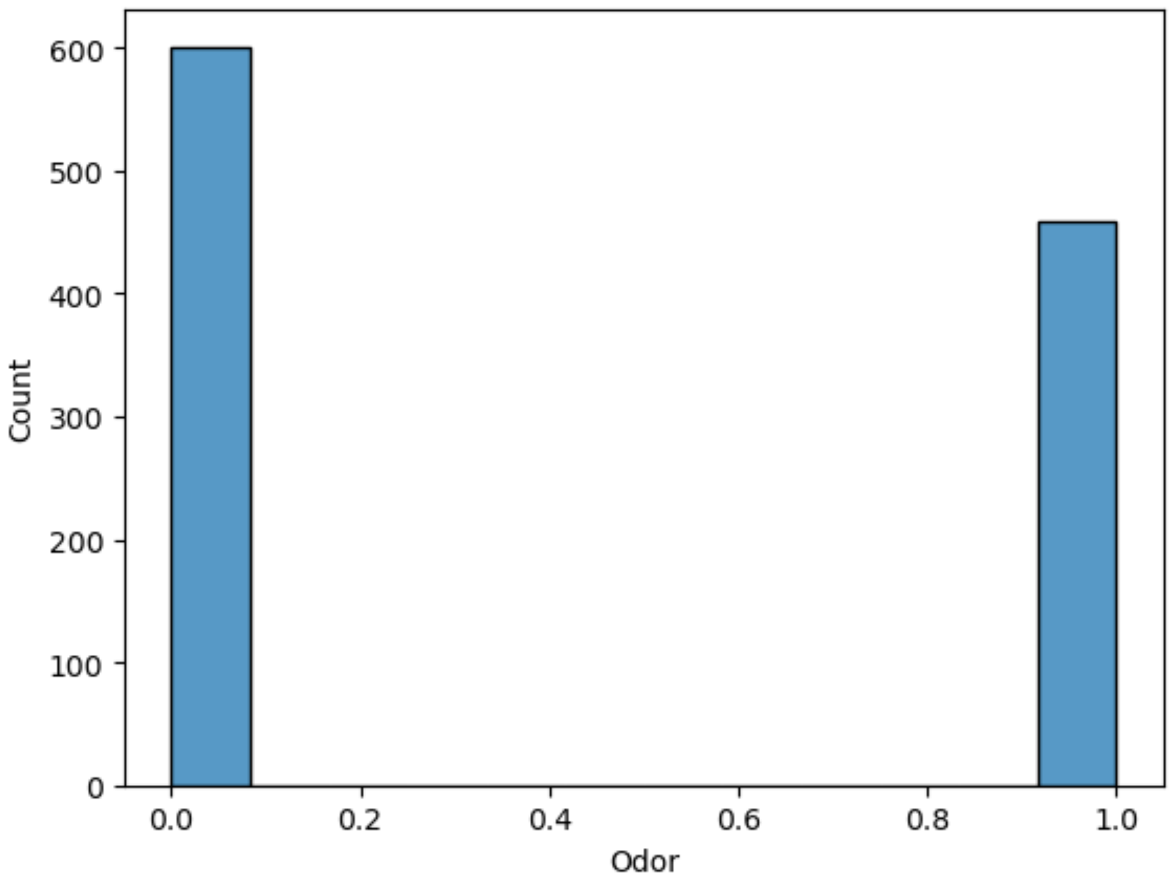
In [24]:
```
sns.boxplot(data['Taste'])
```

Out[24]: <AxesSubplot:xlabel='Taste'>



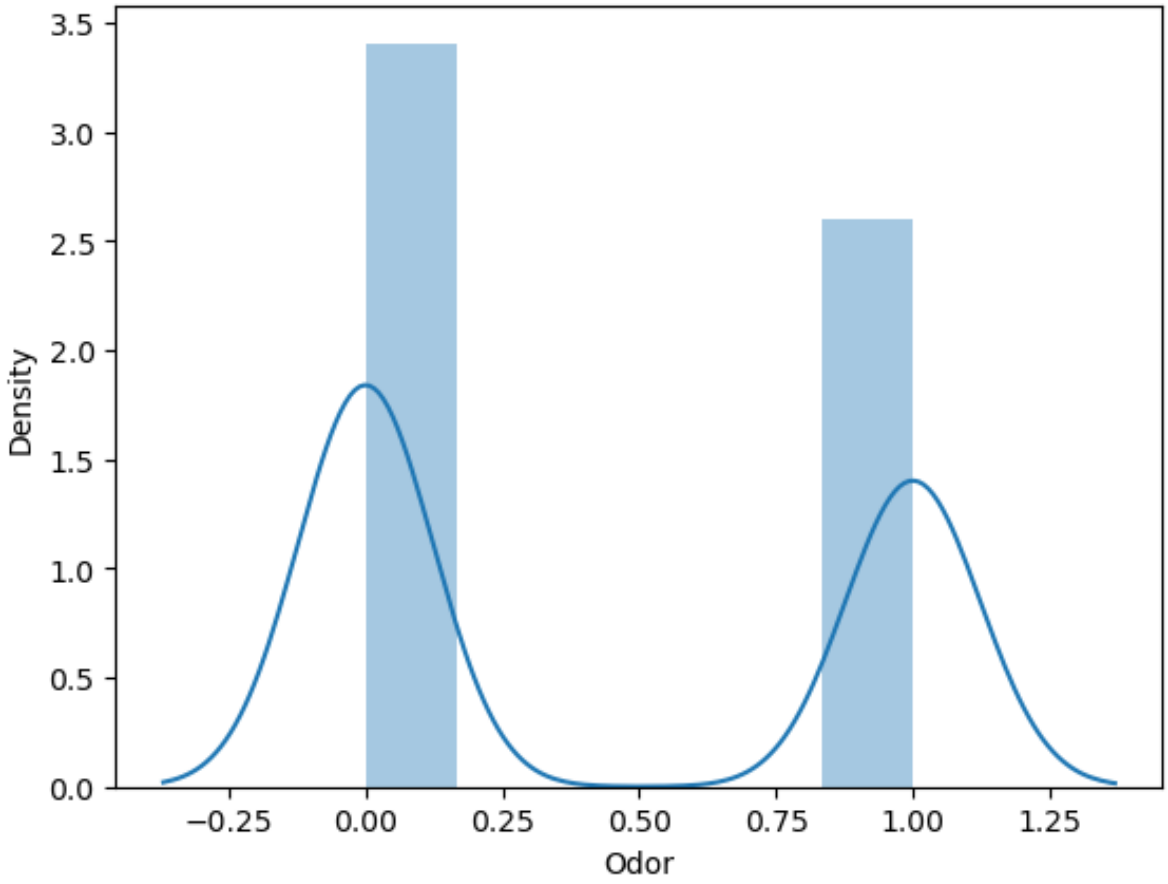
In [25]:
```
sns.histplot(data['Odor'])
```

Out[25]:
```
<AxesSubplot:xlabel='Odor', ylabel='Count'>
```



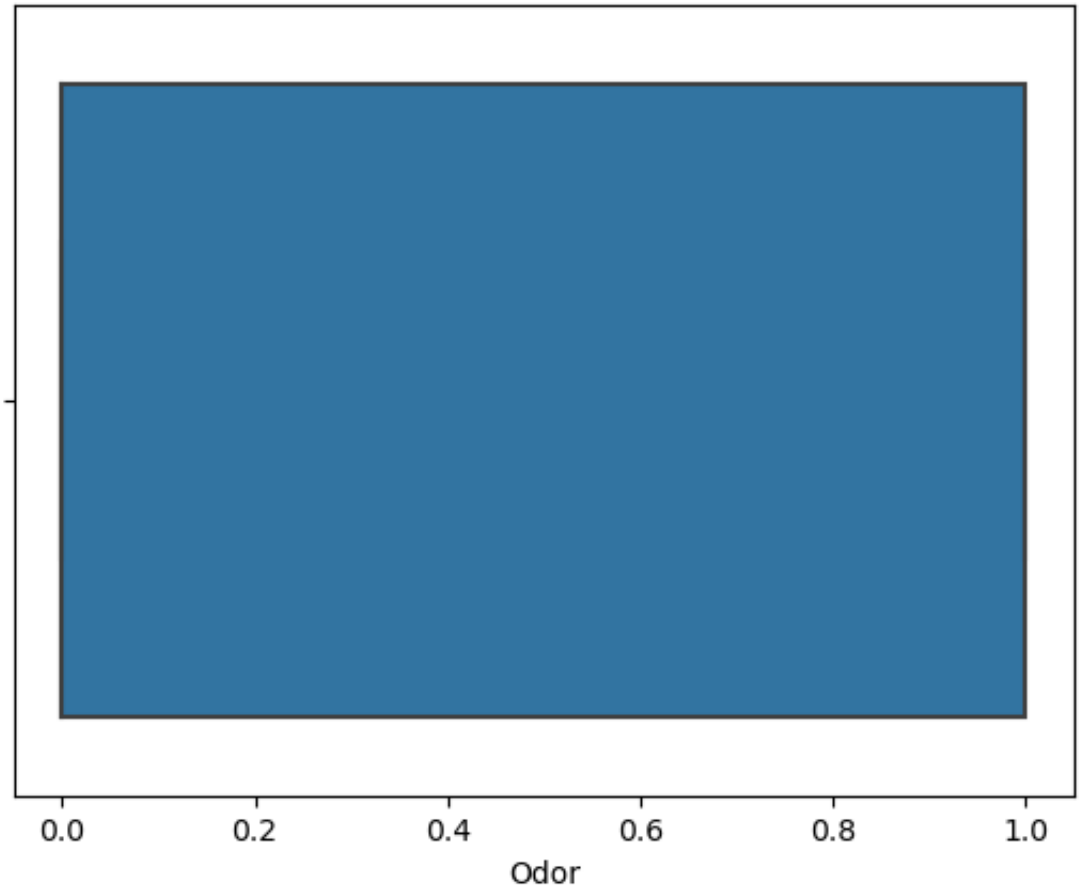
In [26]:
```
sns.distplot(data['Odor'])
```

Out[26]: <AxesSubplot:xlabel='Odor', ylabel='Density'>



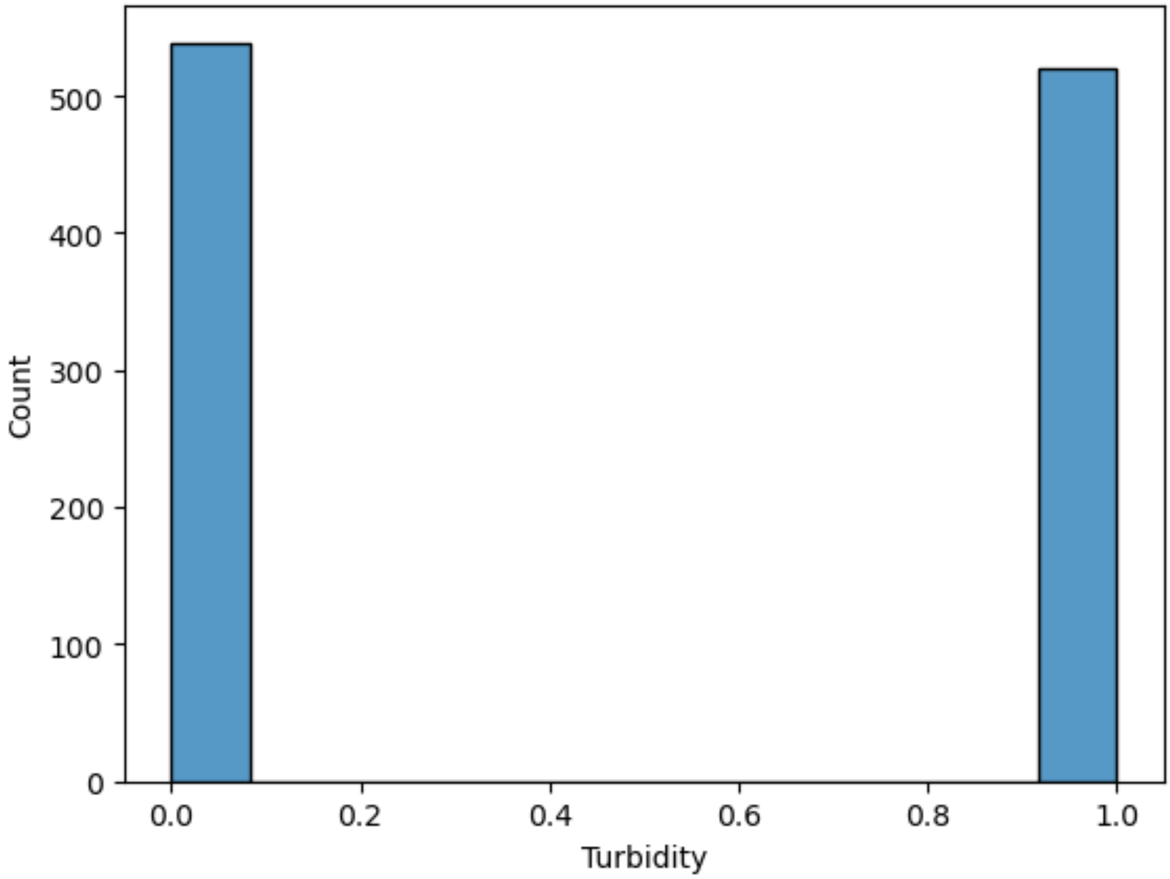
In [27]:
```
sns.boxplot(data['Odor'])
```

Out[27]: <AxesSubplot:xlabel='Odor'>



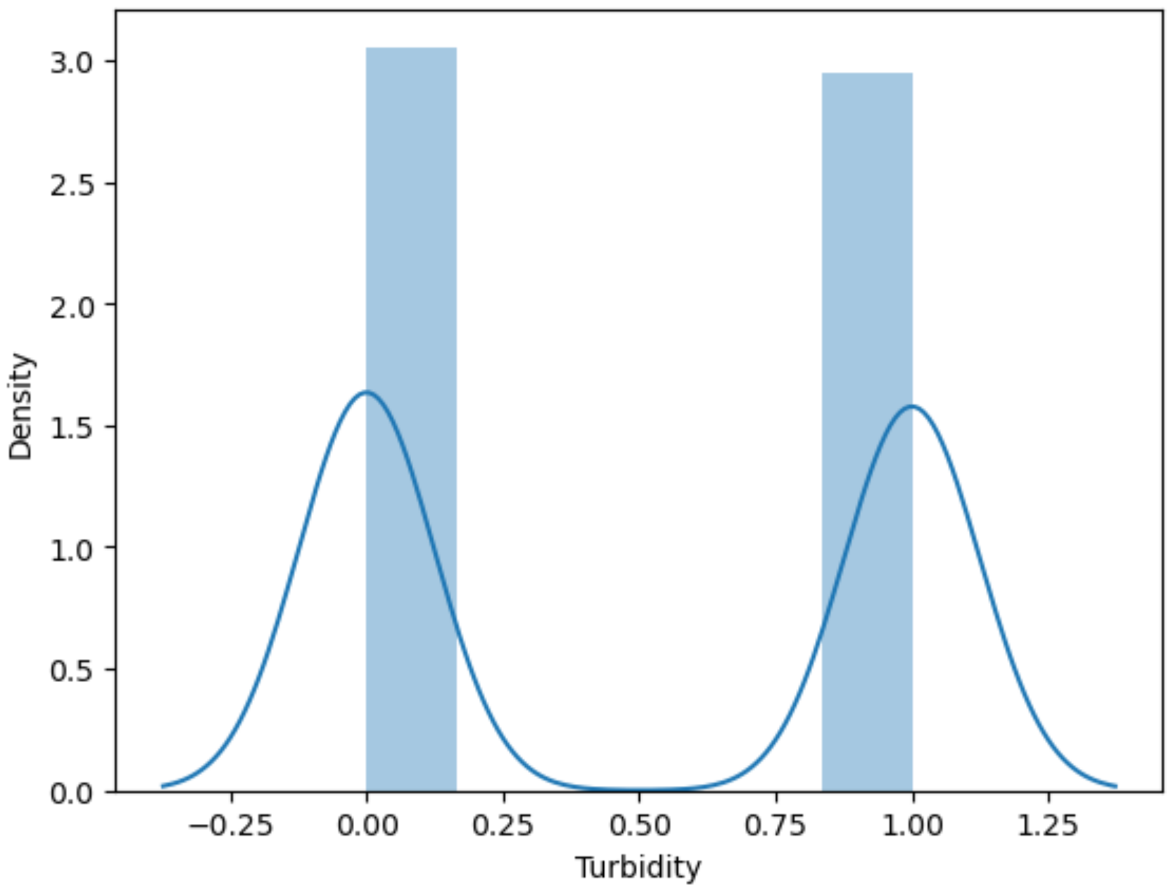
In [28]:
```
sns.histplot(data['Turbidity'])
```

Out[28]: <AxesSubplot:xlabel='Turbidity', ylabel='Count'>



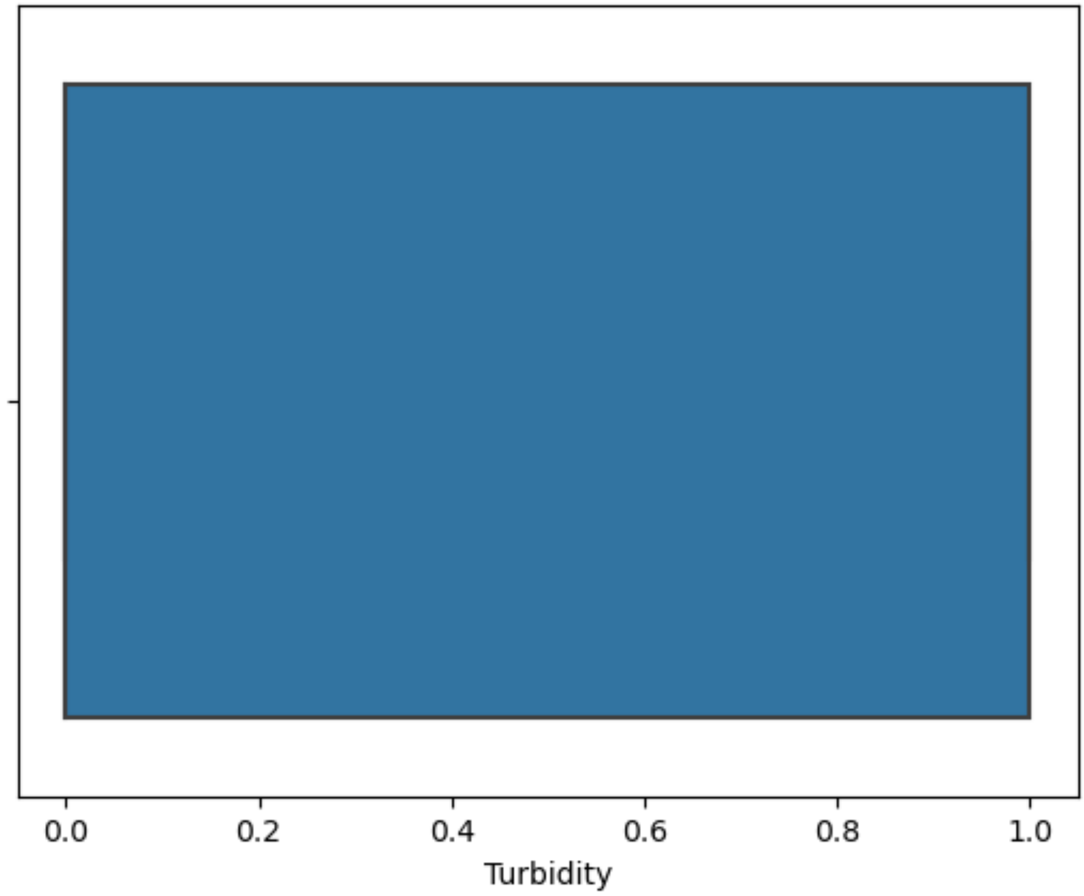
In [29]:
```
sns.distplot(data['Turbidity'])
```

Out[29]: <AxesSubplot:xlabel='Turbidity', ylabel='Density'>



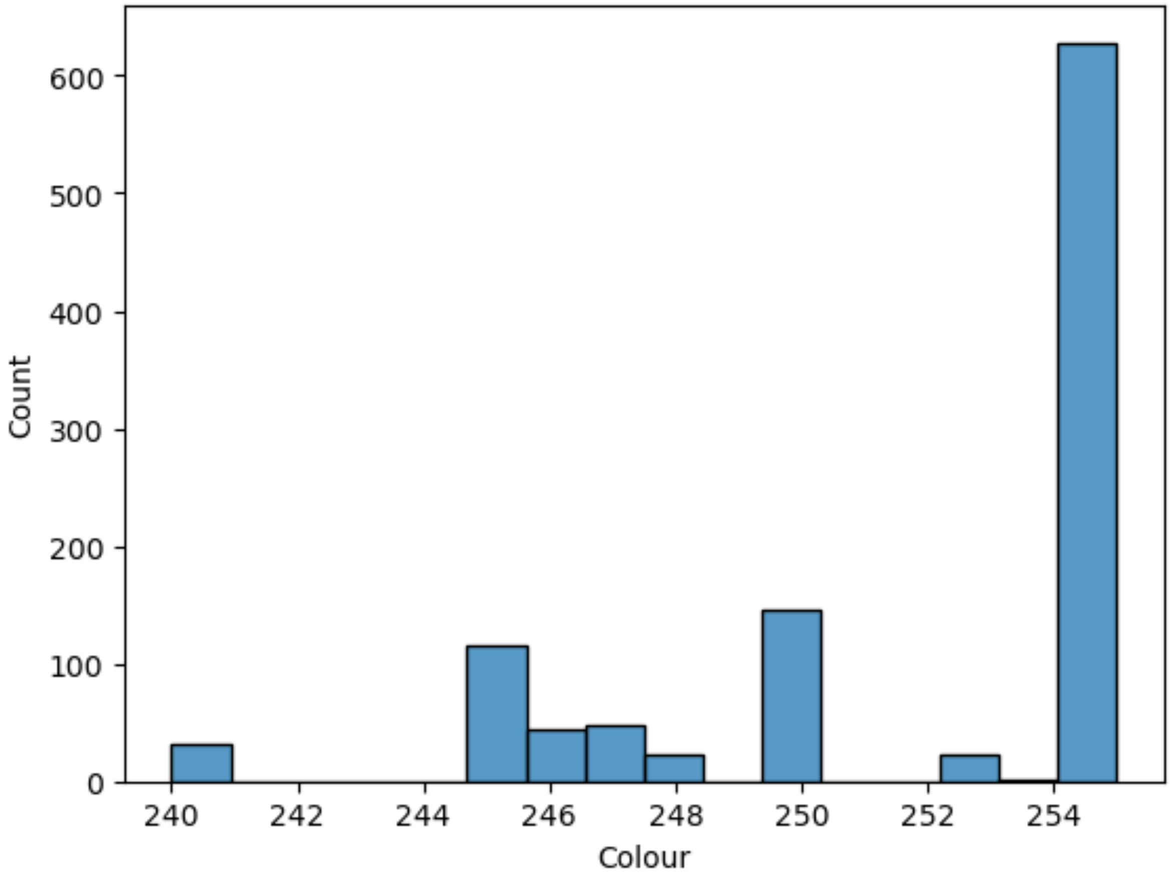
In [30]:
```
sns.boxplot(data['Turbidity'])
```

Out[30]: <AxesSubplot:xlabel='Turbidity'>



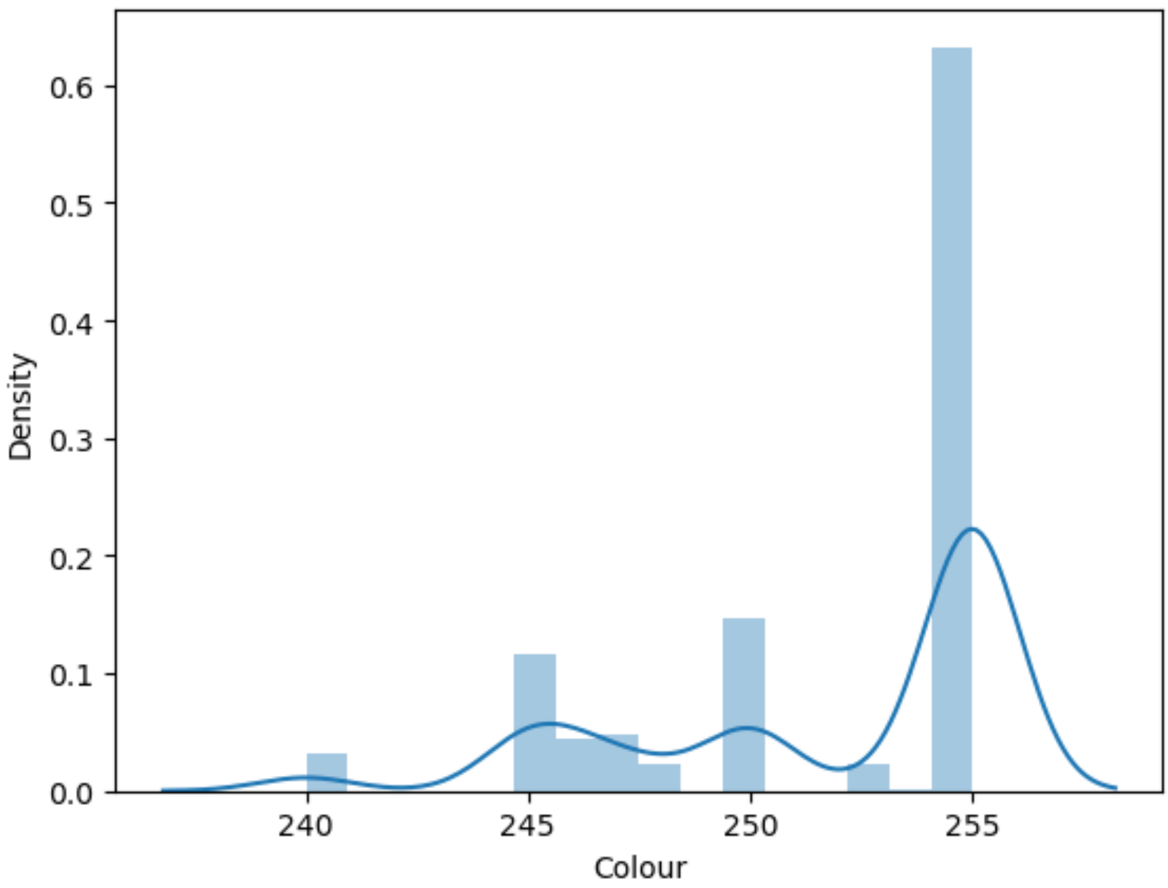
In [31]:
```
sns.histplot(data['Colour'])
```

Out[31]: <AxesSubplot:xlabel='Colour', ylabel='Count'>



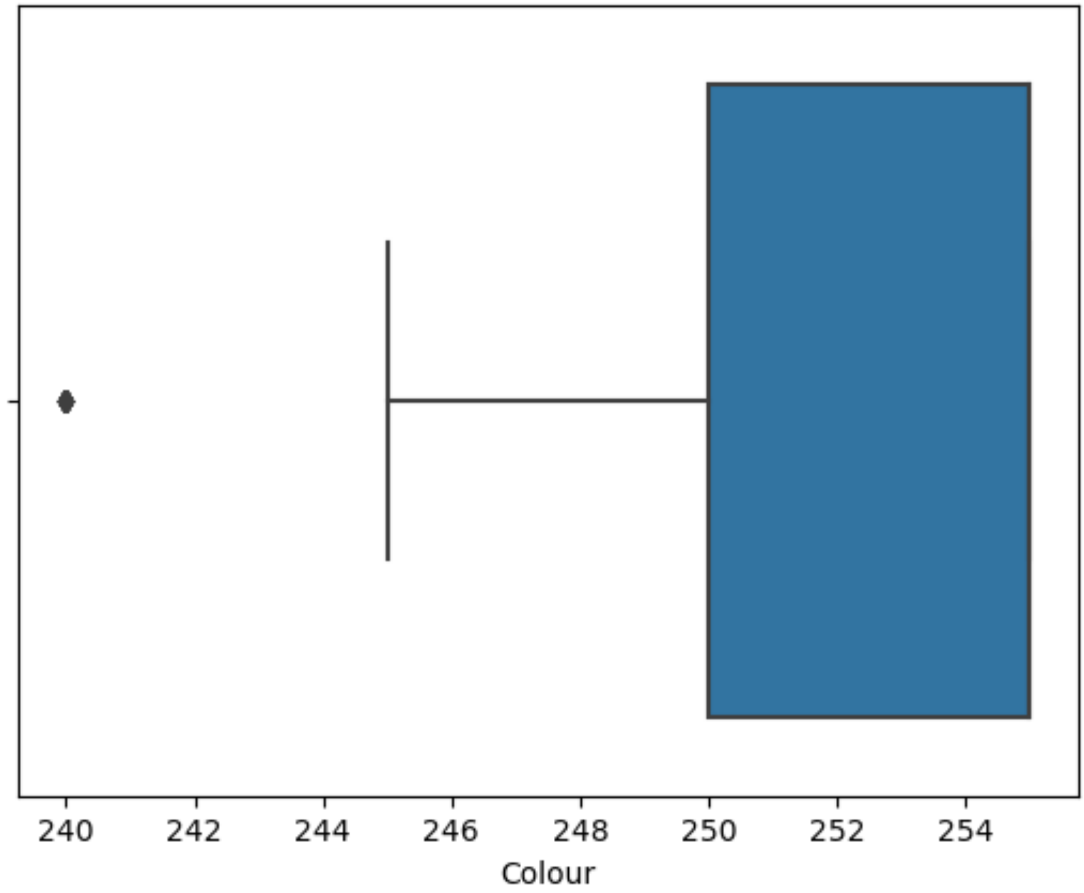
In [32]:
```
sns.distplot(data['Colour'])
```

Out[32]: <AxesSubplot:xlabel='Colour', ylabel='Density'>



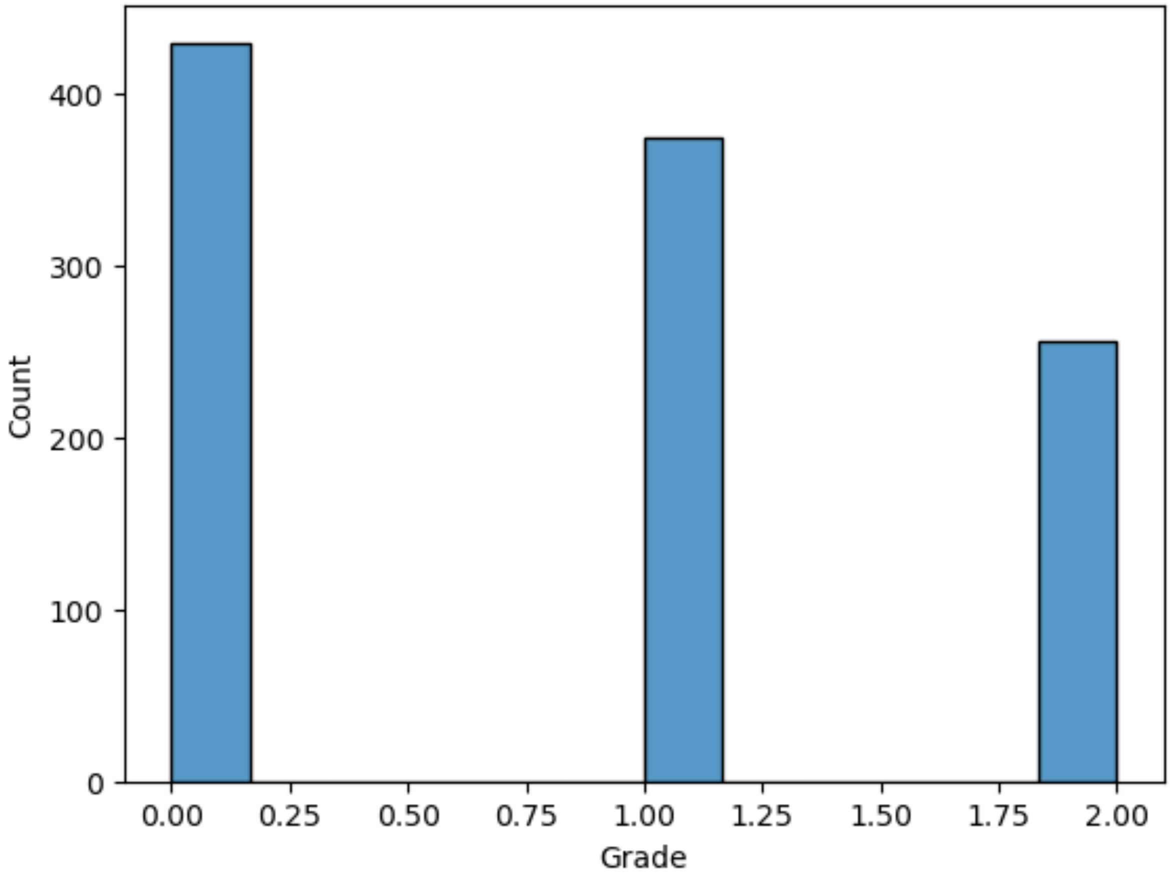
In [33]:
```
sns.boxplot(data['Colour'])
```

Out[33]: <AxesSubplot:xlabel='Colour'>



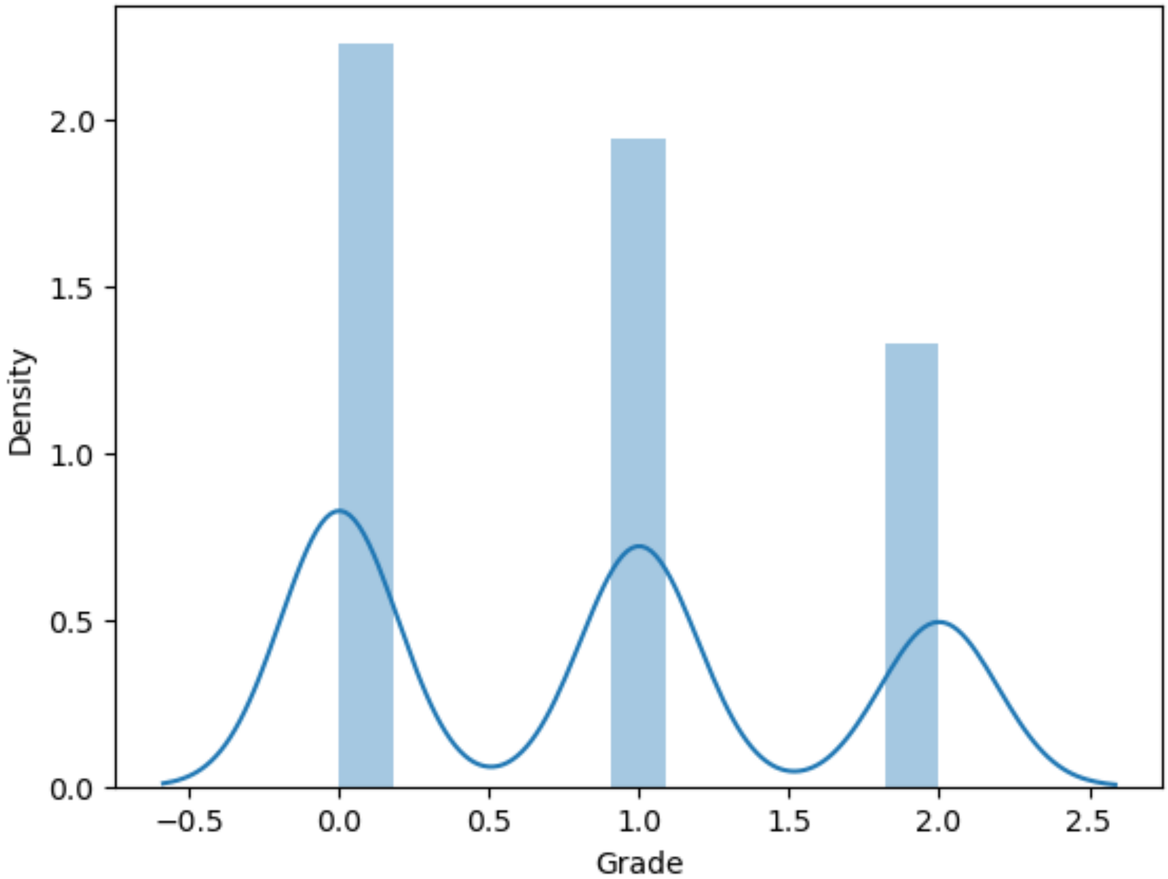
In [34]:
```
sns.histplot(data['Grade'])
```

Out[34]: <AxesSubplot:xlabel='Grade', ylabel='Count'>



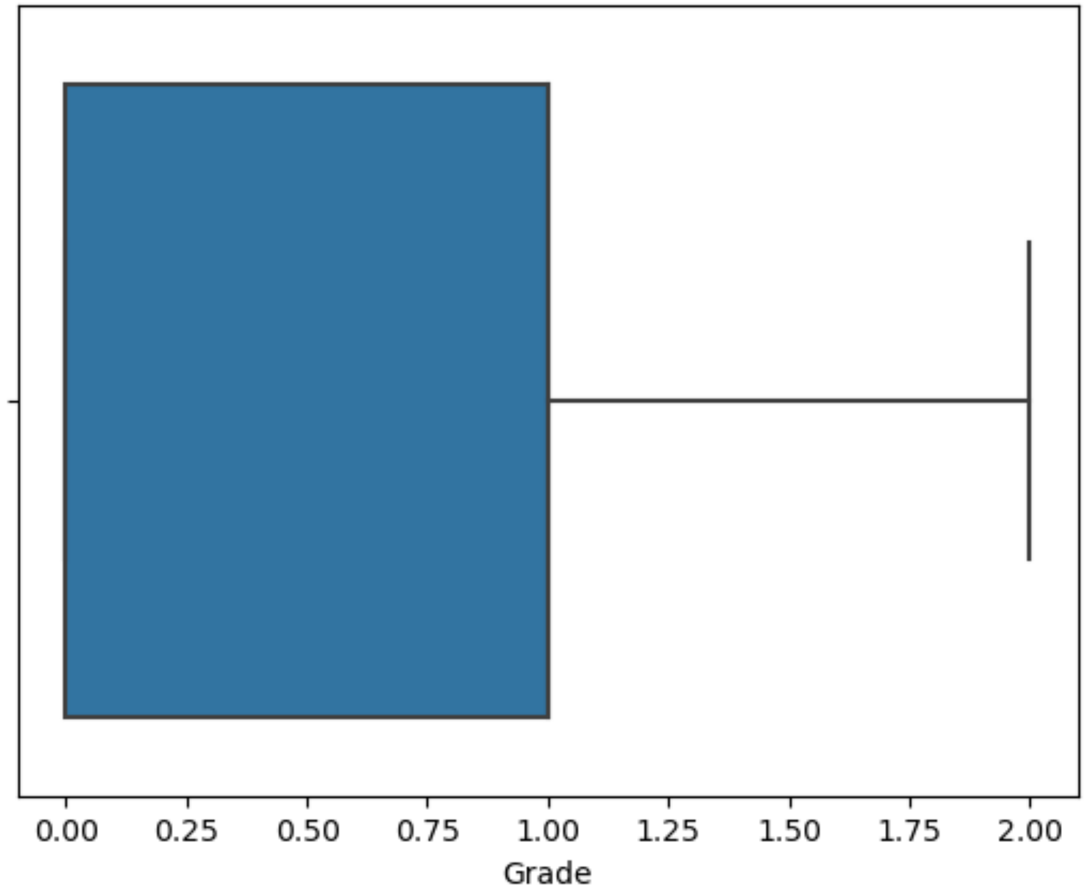
In [35]:
```
sns.distplot(data['Grade'])
```

Out[35]: <AxesSubplot:xlabel='Grade', ylabel='Density'>



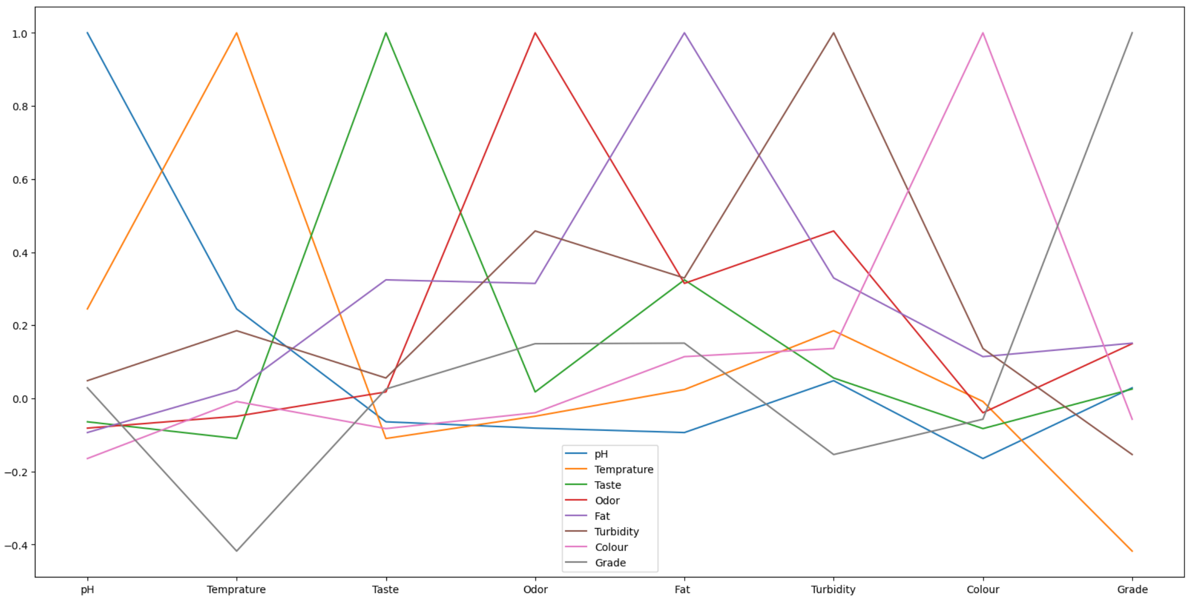
In [36]:
```
sns.boxplot(data['Grade'])
```

Out[36]: <AxesSubplot:xlabel='Grade'>



In [37]:
```
#plot correation
data_corr = data.corr()
data_corr.plot(figsize=(20,10))
```

Out[37]: <AxesSubplot:>



In [38]:
```
#plot heatmap
plt.figure(figsize=(20,10))
sns.heatmap(data.corr(),annot=True)
```

Out[38]: <AxesSubplot:>

I check for null values.

In [39]:
```
pd.isnull(data).sum()
```

Out[39]:
```
pH            0
Temprature    0
Taste         0
Odor          0
Fat           0
Turbidity     0
Colour        0
Grade         0
dtype: int64
```

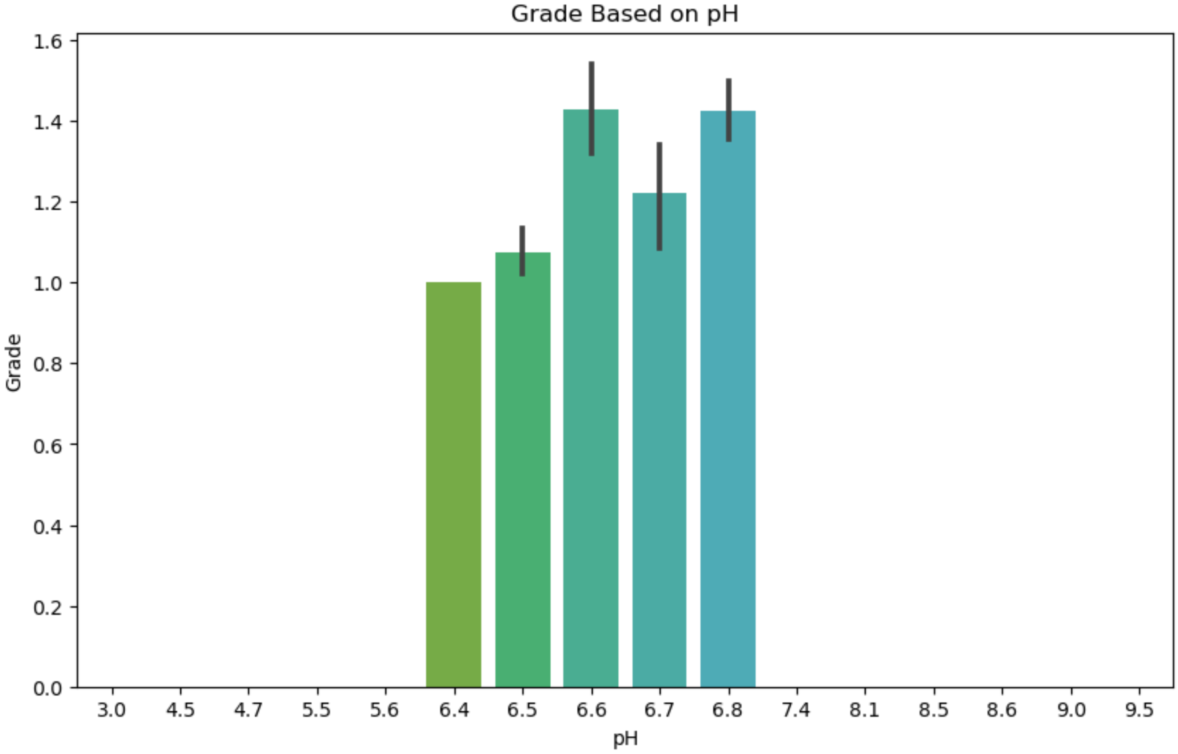
I don't need to fill in missing value as there is no null value.

Value Analysis With Graphs

I'm creating charts to examine how variables relate to "Grade".

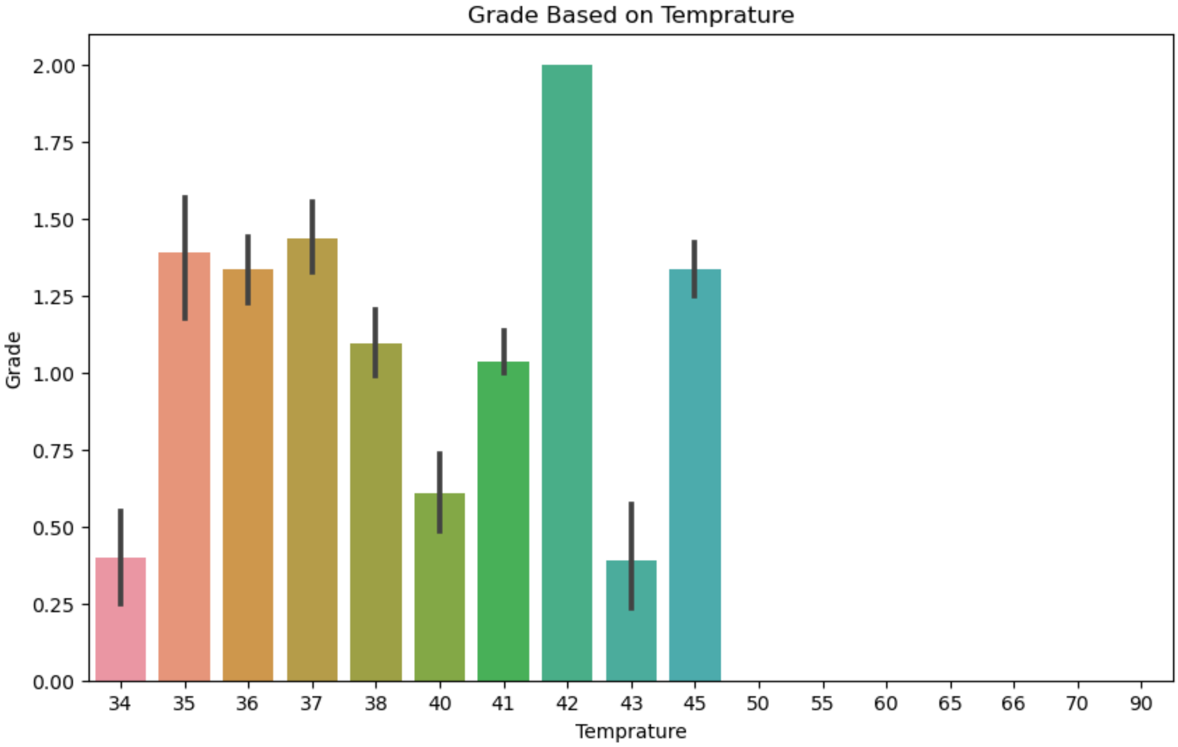
In [40]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=data, x="pH", y="Grade", ax=ax)
plt.title('Grade Based on pH')
```

Out[40]:
```
Text(0.5, 1.0, 'Grade Based on pH')
```



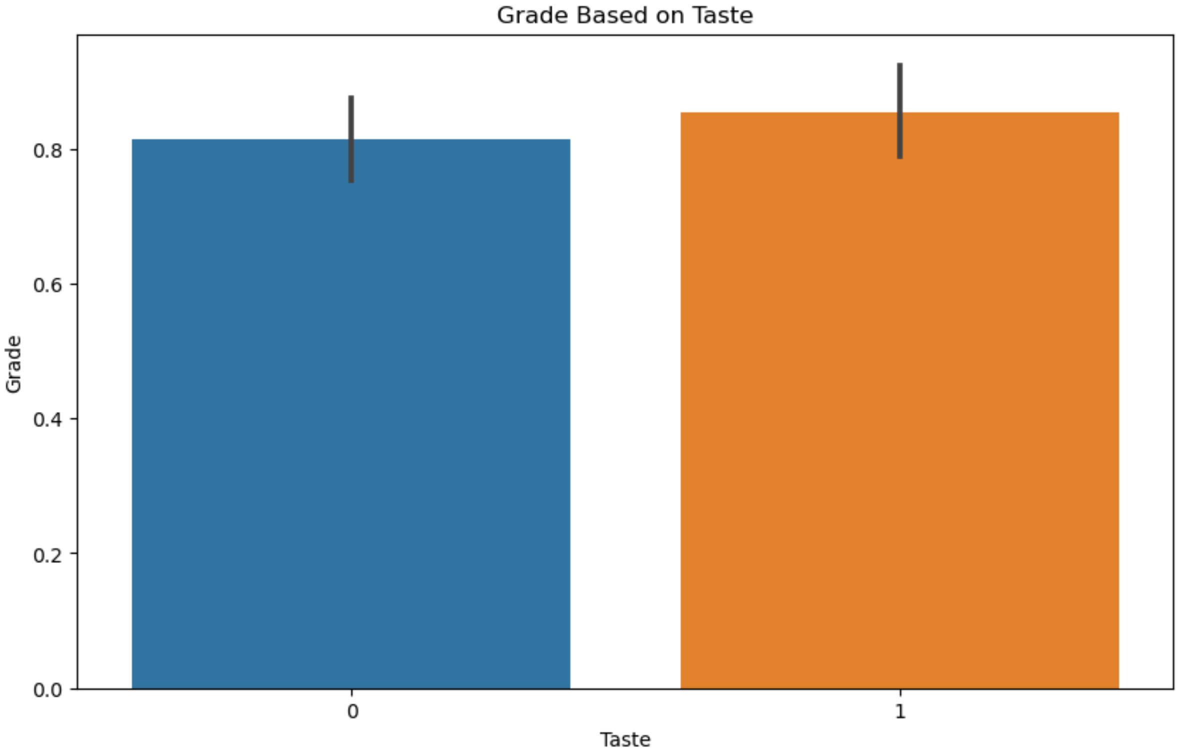
In [41]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=data, x="Temprature", y="Grade", ax=ax)
plt.title('Grade Based on Temprature')
```

Out[41]: Text(0.5, 1.0, 'Grade Based on Temprature')



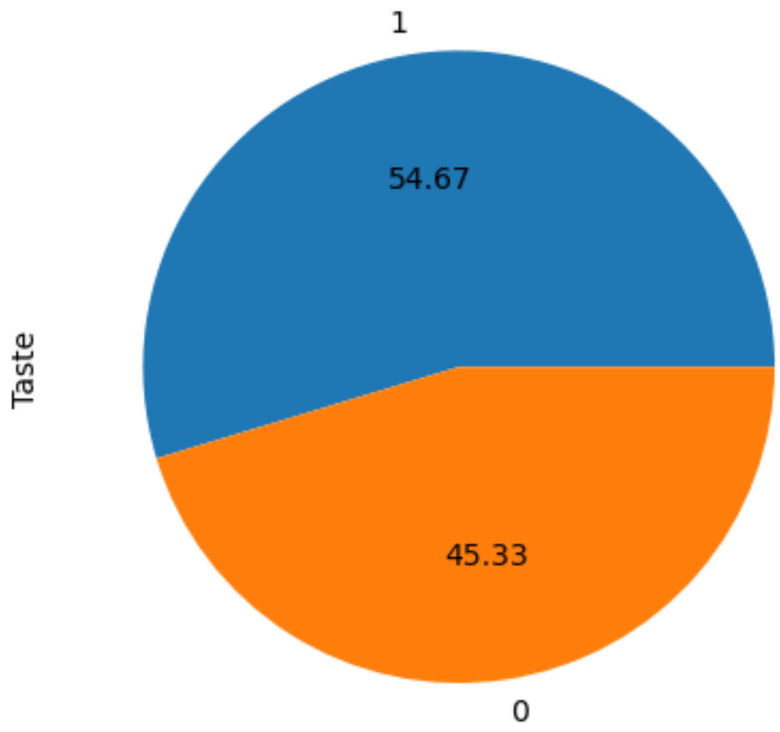
In [42]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=data, x="Taste", y="Grade", ax=ax)
plt.title('Grade Based on Taste')
```

Out[42]:
```
Text(0.5, 1.0, 'Grade Based on Taste')
```



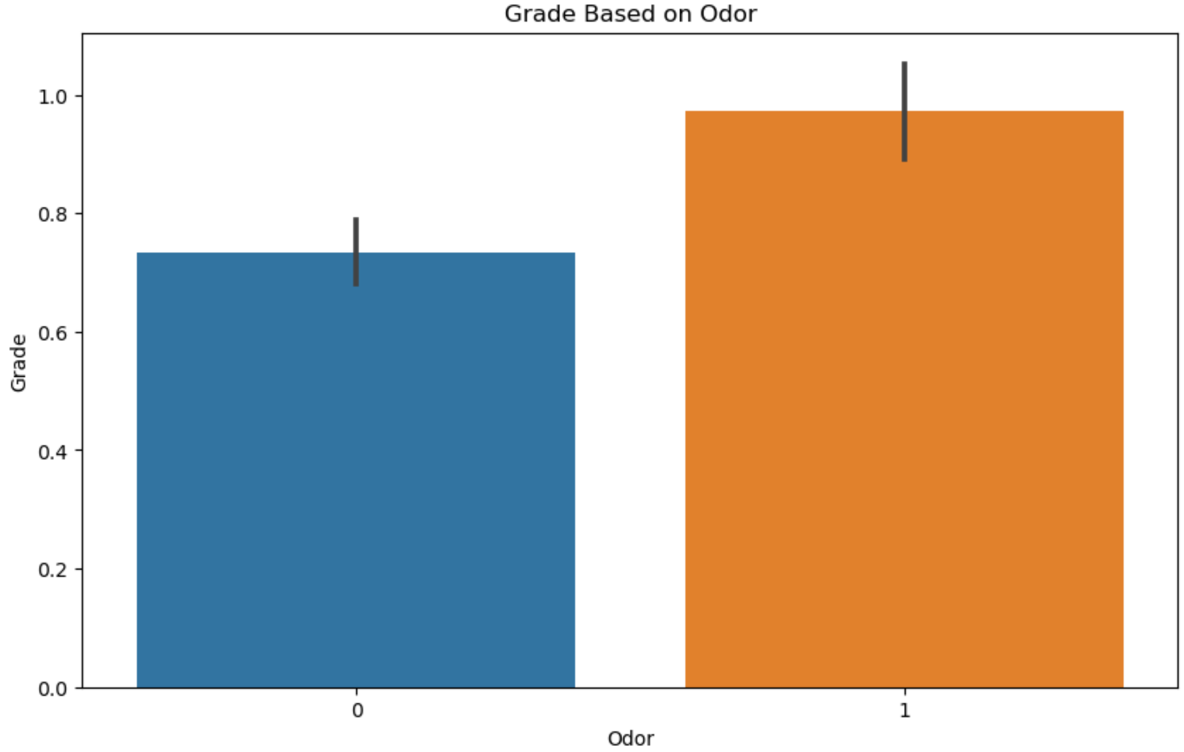
In [43]:
```
data['Taste'].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[43]: <AxesSubplot:ylabel='Taste'>



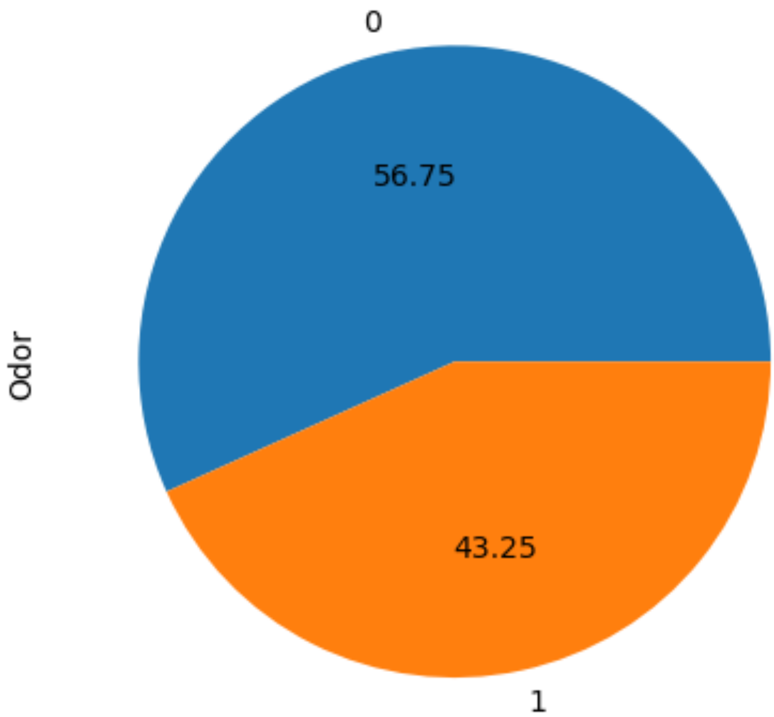
In [44]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=data, x="Odor", y="Grade", ax=ax)
plt.title('Grade Based on Odor')
```

Out[44]:
```
Text(0.5, 1.0, 'Grade Based on Odor')
```



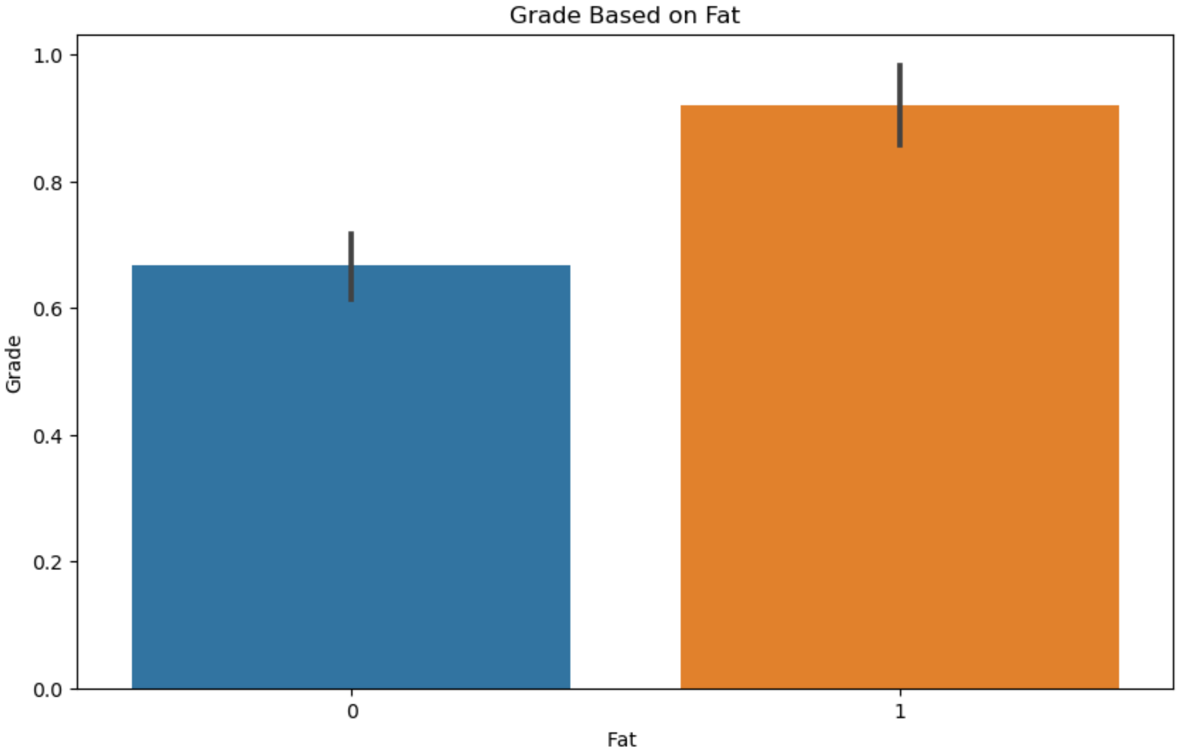
In [45]:
```
data['Odor'].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[45]: <AxesSubplot:ylabel='Odor'>



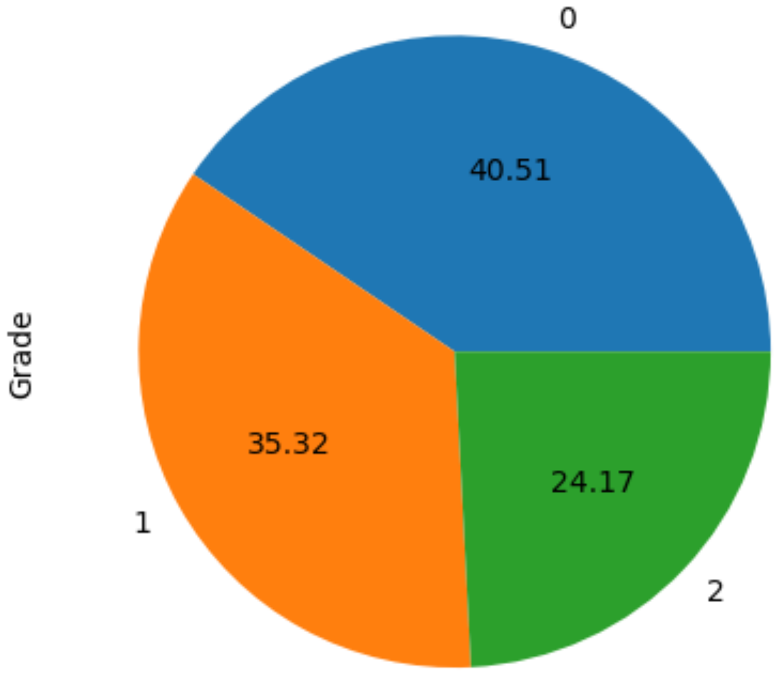
In [46]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=data, x="Fat ", y="Grade", ax=ax)
plt.title('Grade Based on Fat')
```

Out[46]:
```
Text(0.5, 1.0, 'Grade Based on Fat')
```



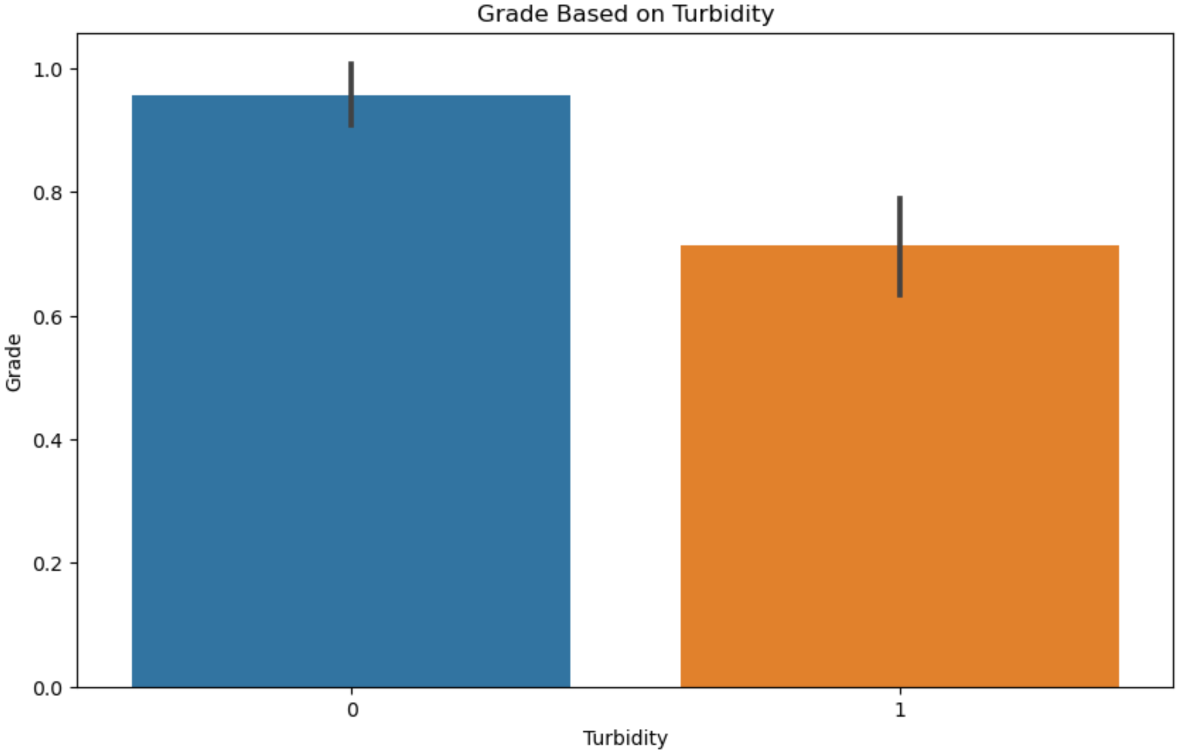
In [47]:
```
data['Grade'].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[47]:
```
<AxesSubplot:ylabel='Grade'>
```



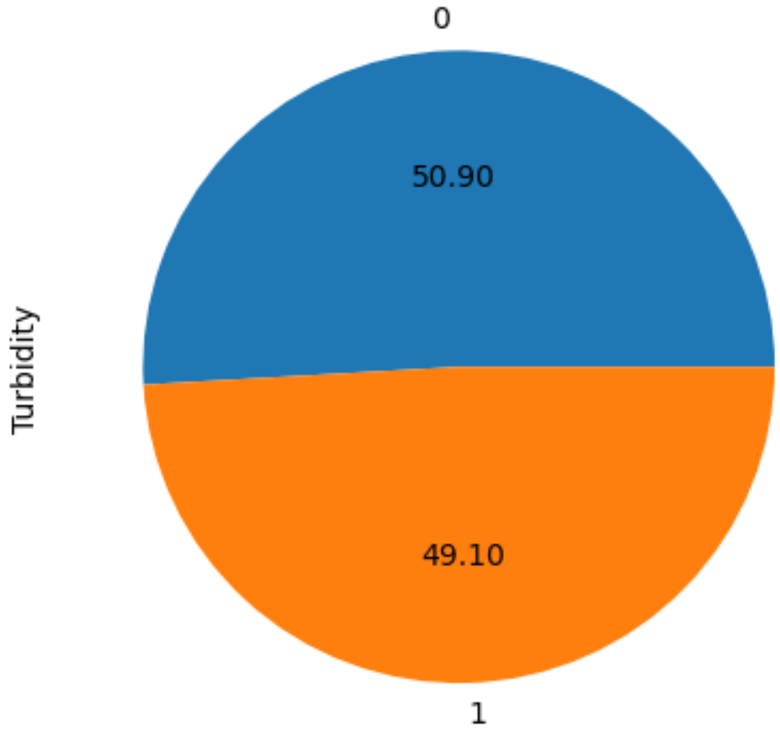
In [48]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=data, x="Turbidity", y="Grade", ax=ax)
plt.title('Grade Based on Turbidity')
```

Out[48]:
```
Text(0.5, 1.0, 'Grade Based on Turbidity')
```



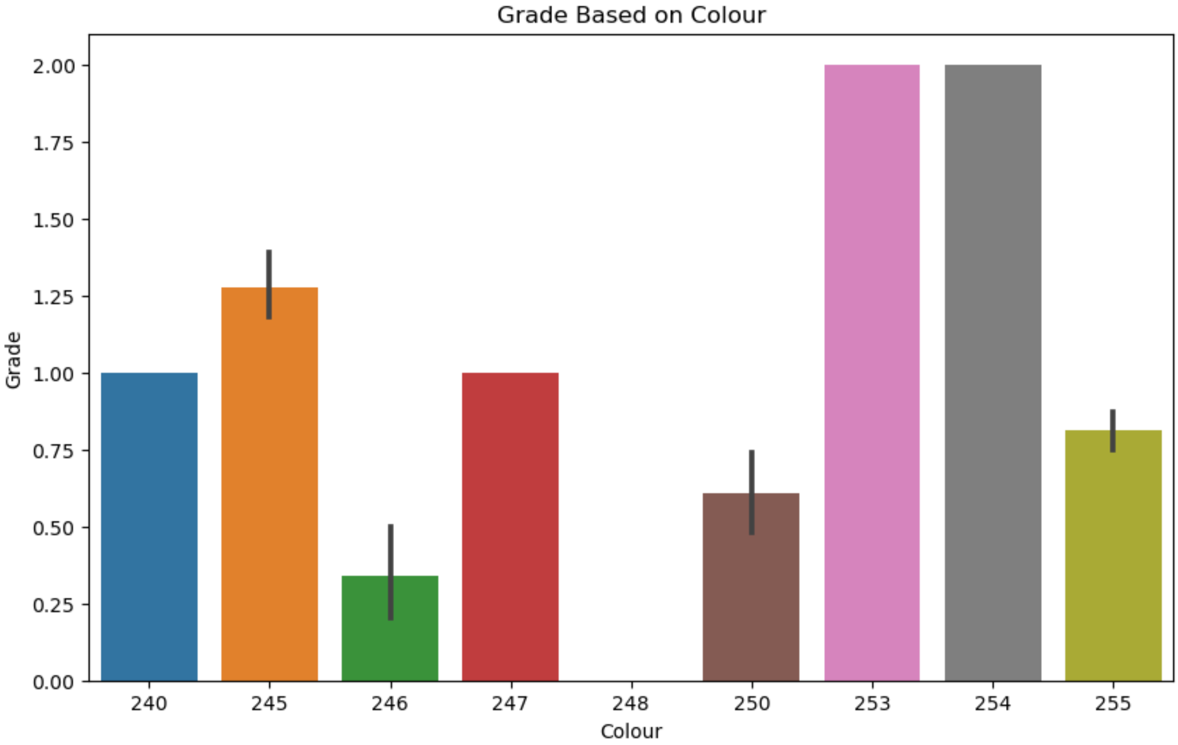
In [49]:
```
data['Turbidity'].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[49]: <AxesSubplot:ylabel='Turbidity'>



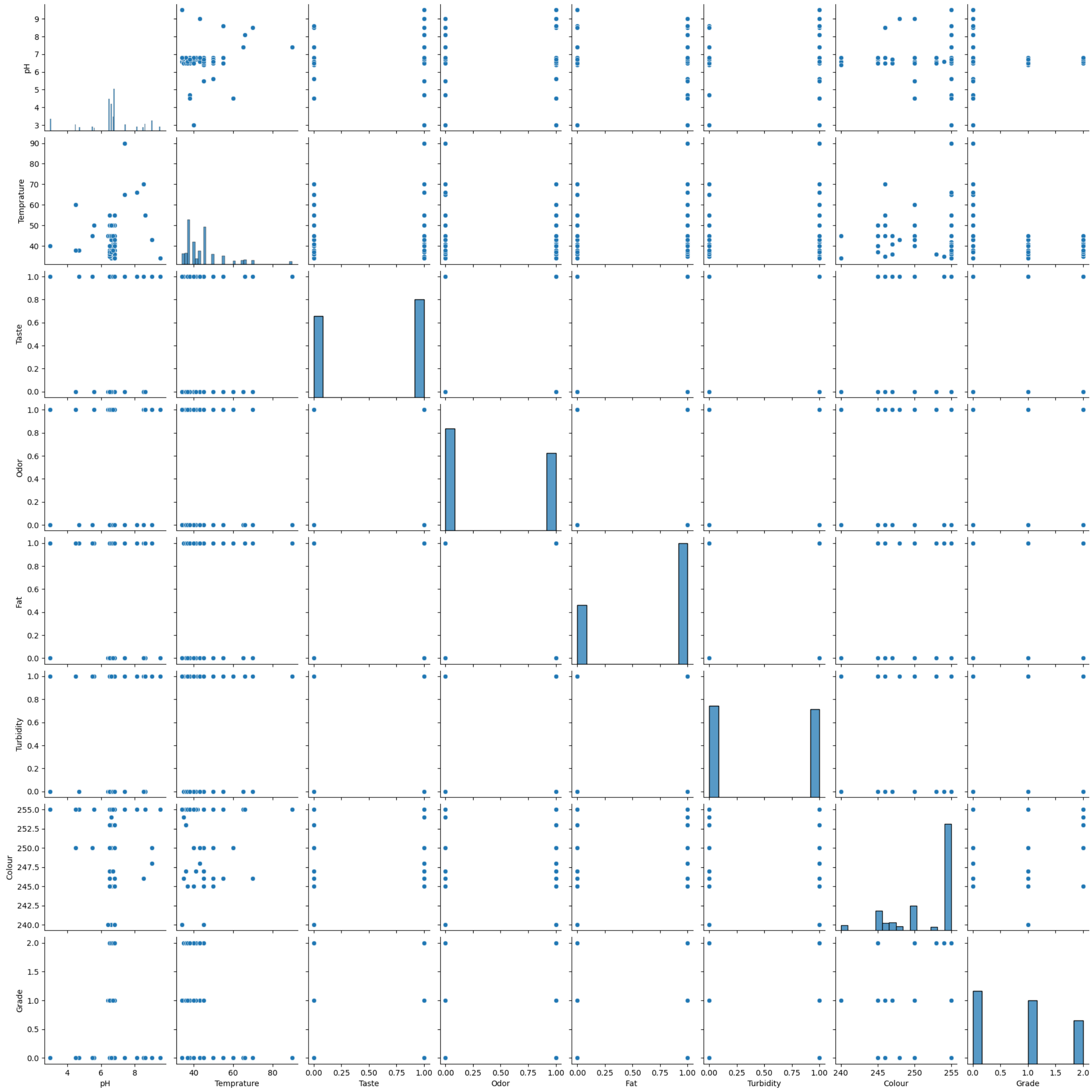
In [50]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(data=data, x="Colour", y="Grade", ax=ax)
plt.title('Grade Based on Colour')
```

Out[50]:
```
Text(0.5, 1.0, 'Grade Based on Colour')
```



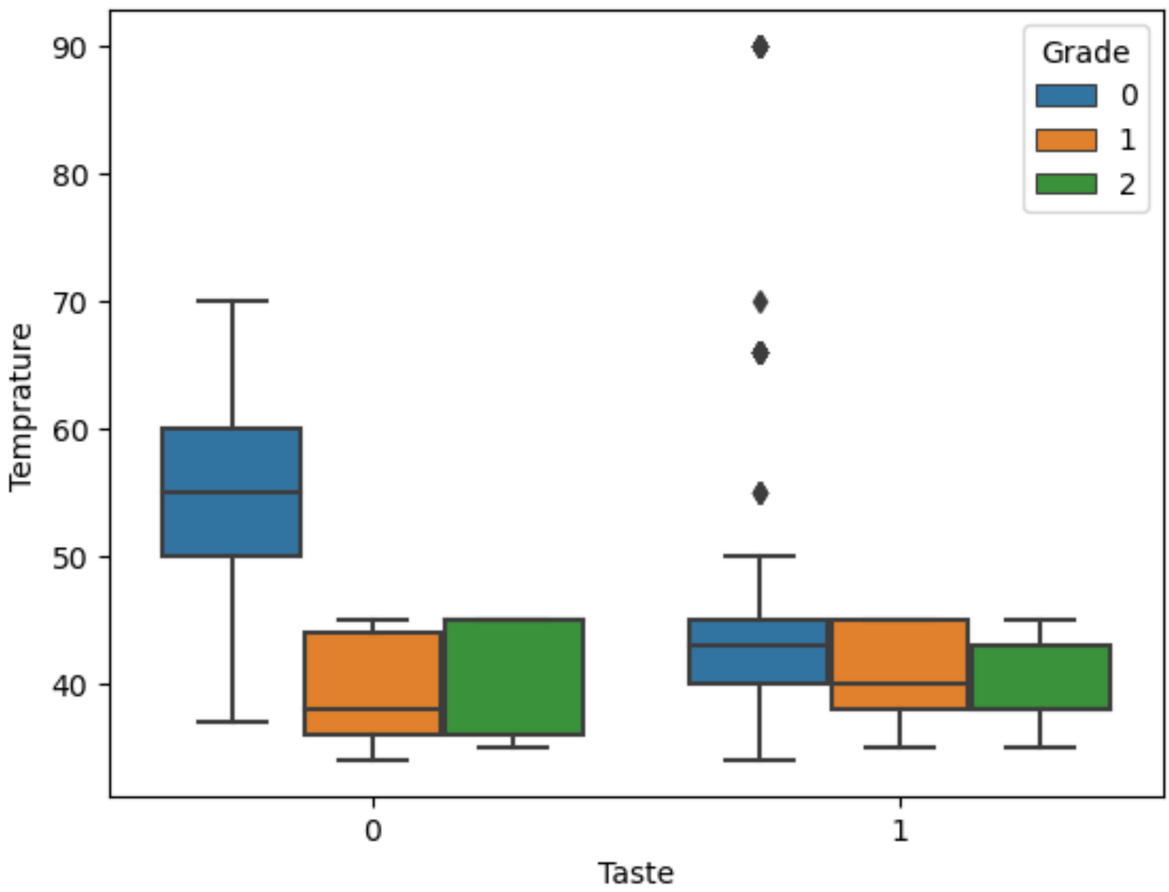
In [51]:
```
sns.pairplot(data)
```

Out[51]: <seaborn.axisgrid.PairGrid at 0x7fbc151596a0>



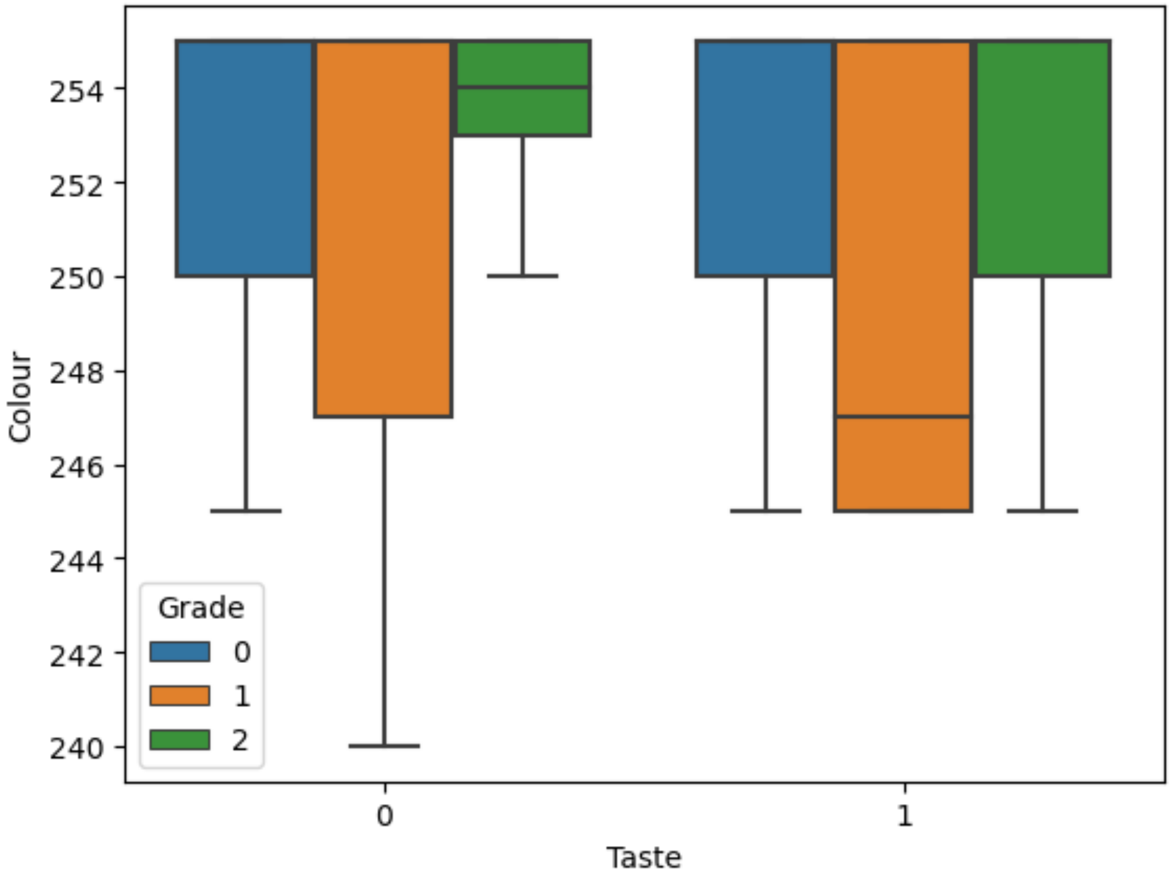
In [52]:
```
sns.boxplot(data['Taste'],data['Temprature'],hue=data['Grade'])
```

Out[52]:
```
<AxesSubplot:xlabel='Taste', ylabel='Temprature'>
```



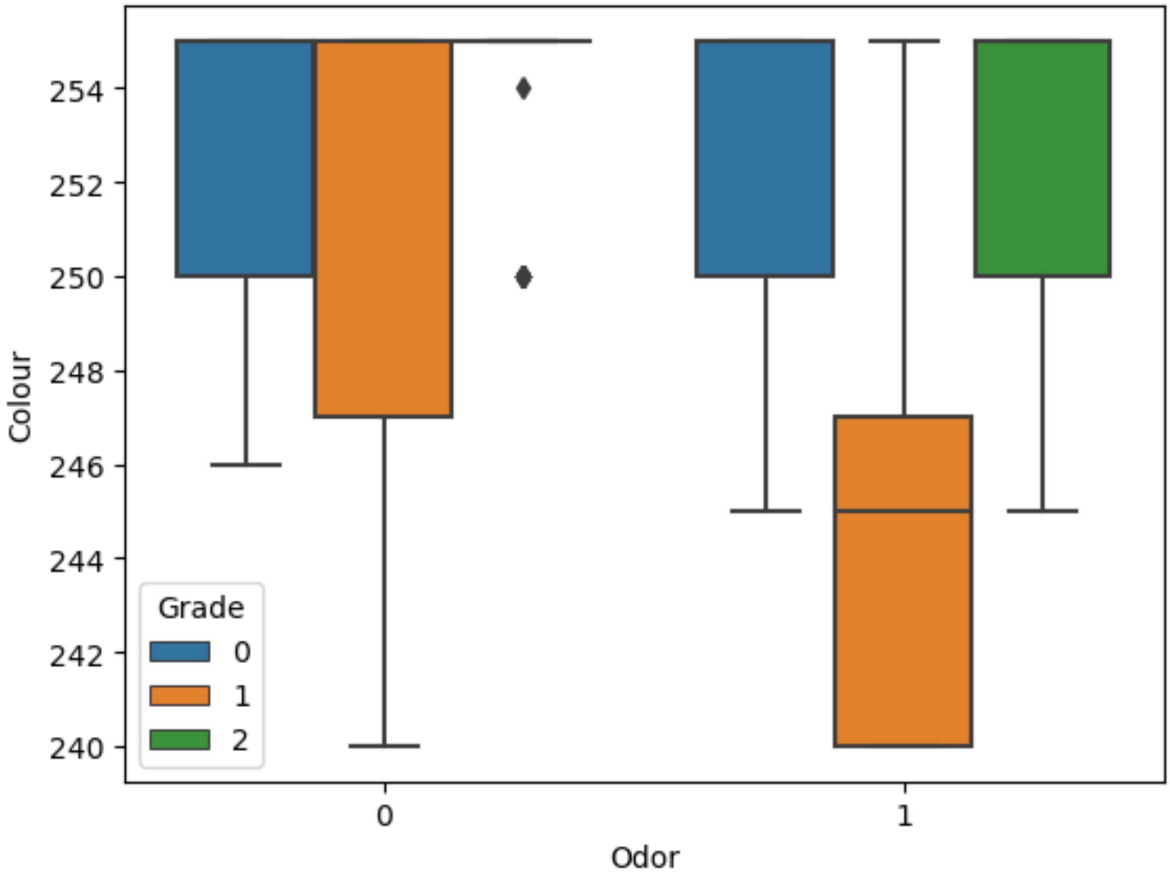
In [53]:
```
sns.boxplot(data['Taste'],data['Colour'],hue=data['Grade'])
```

Out[53]:
```
<AxesSubplot:xlabel='Taste', ylabel='Colour'>
```



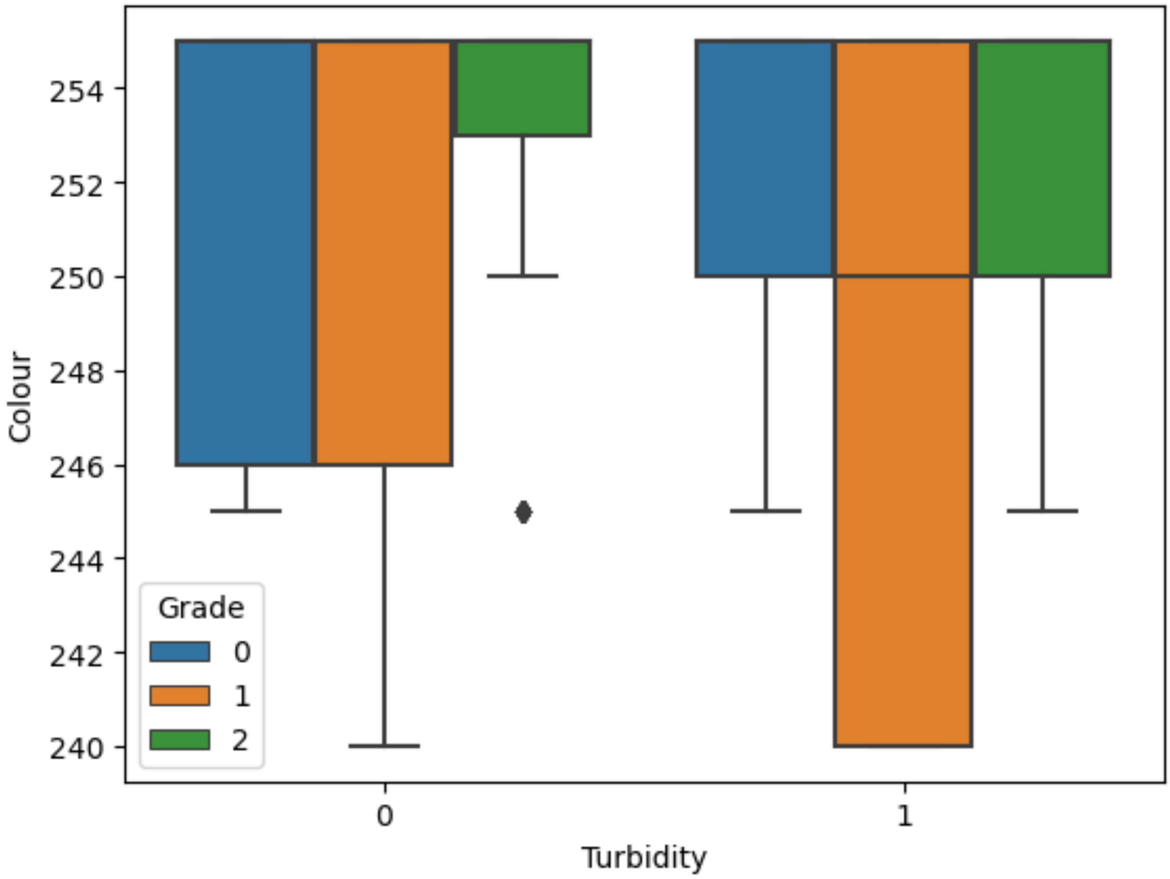
In [55]:
```
sns.boxplot(data['Odor'],data['Colour'],hue=data['Grade'])
```

Out[55]:
```
<AxesSubplot:xlabel='Odor', ylabel='Colour'>
```



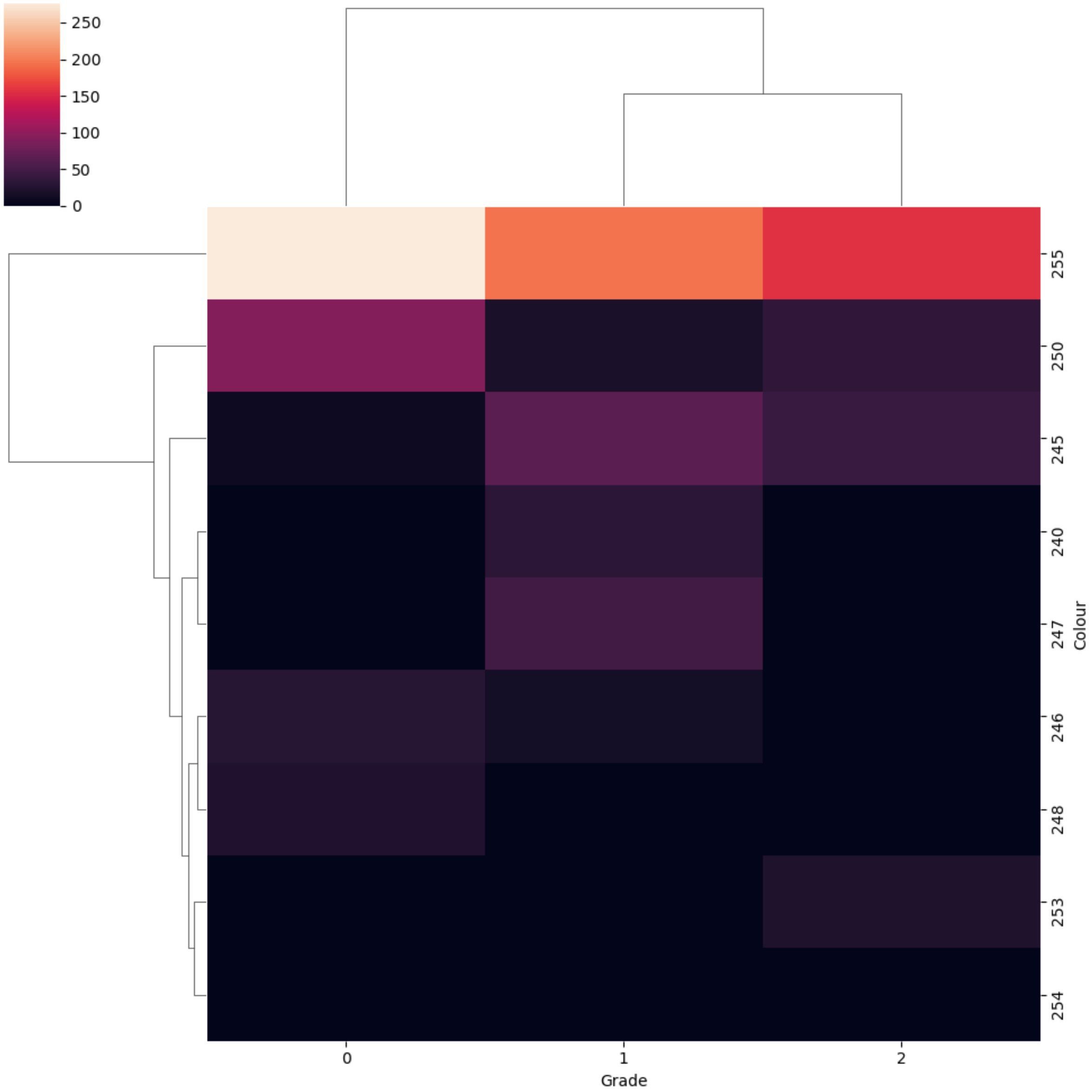
In [56]:
```
sns.boxplot(data['Turbidity'],data['Colour'],hue=data['Grade'])
```

Out[56]: <AxesSubplot:xlabel='Turbidity', ylabel='Colour'>



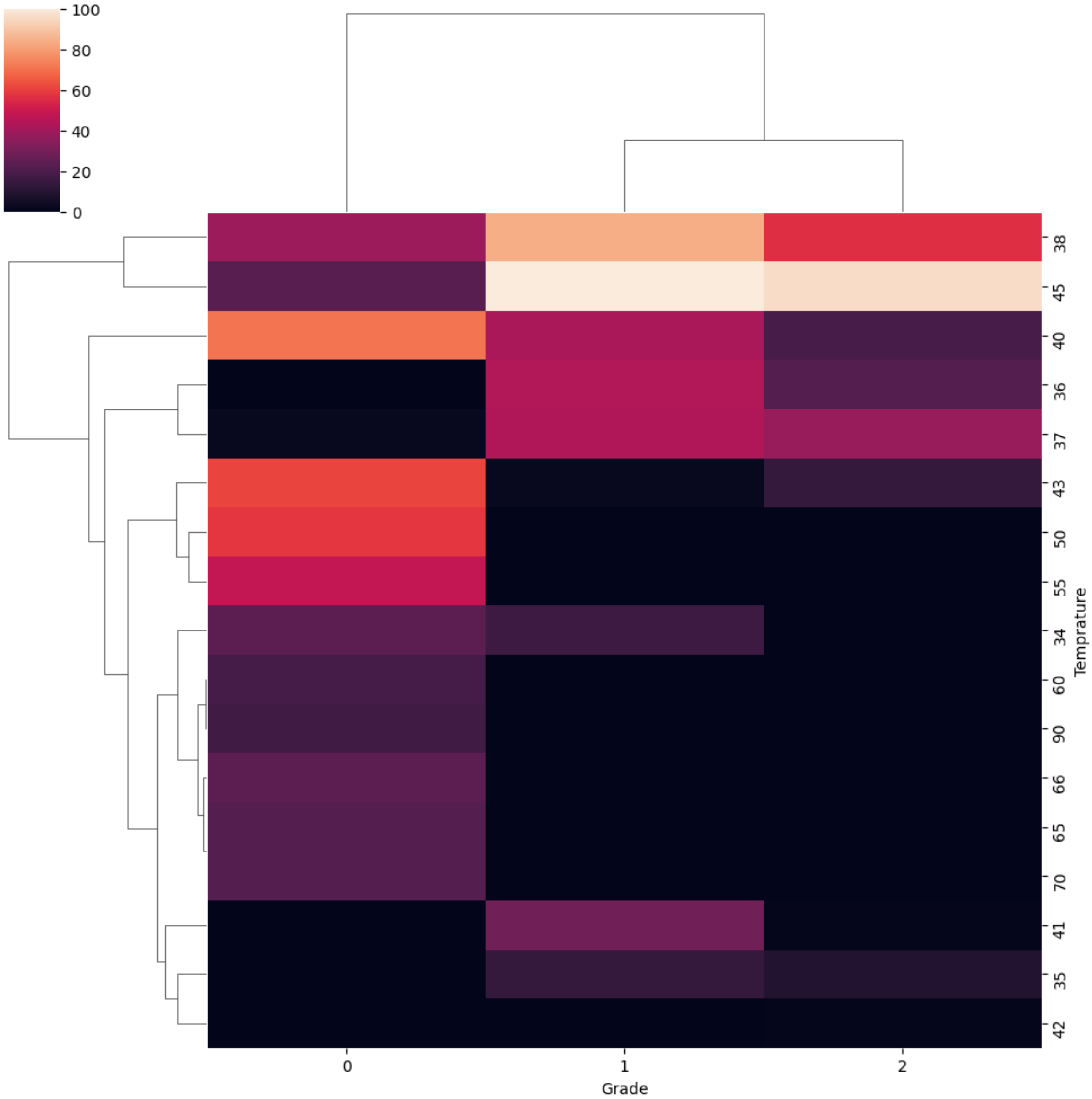
In [57]:
```
sns.clustermap(pd.crosstab(data['Colour'],data['Grade']))
```

Out[57]: <seaborn.matrix.ClusterGrid at 0x7fbbfb1112b0>



In [58]:
```
sns.clustermap(pd.crosstab(data['Temprature'],data['Grade']))
```

Out[58]: <seaborn.matrix.ClusterGrid at 0x7fbbfb0f78b0>



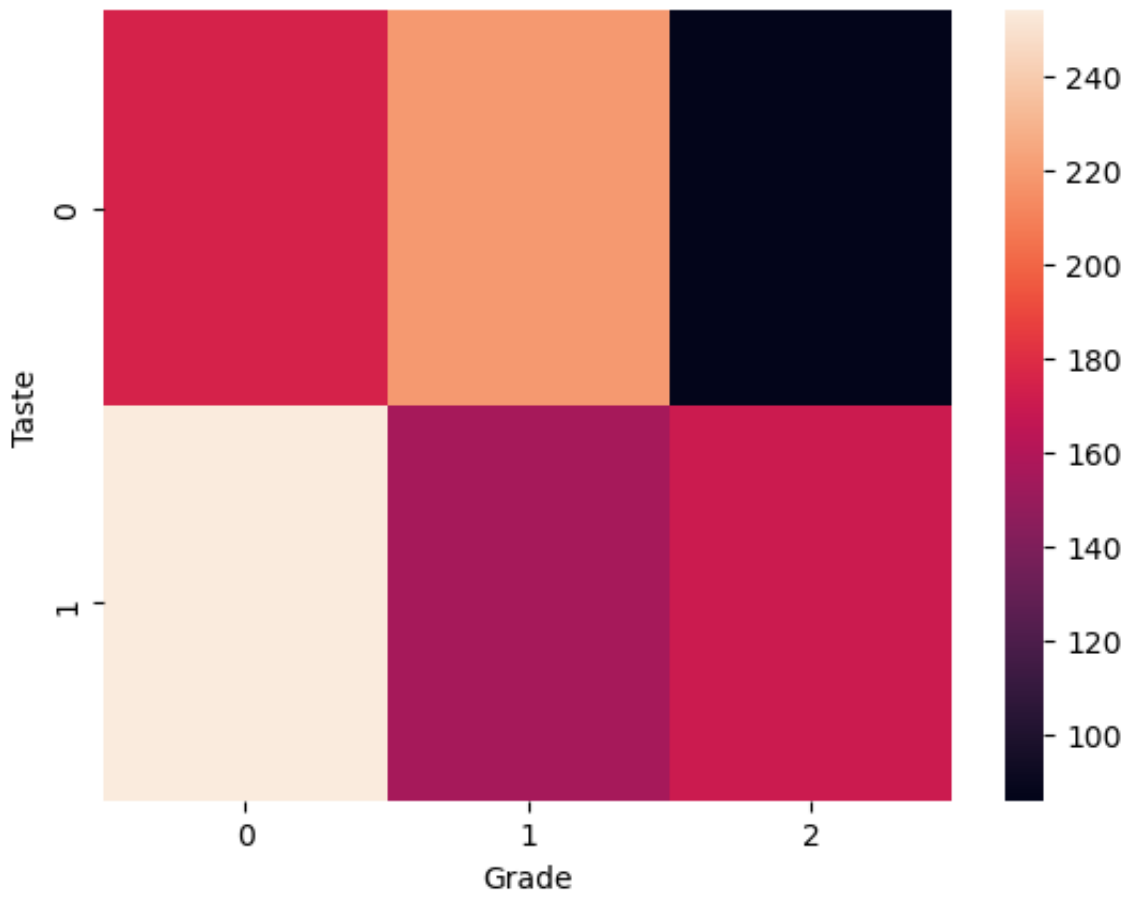
In [54]:
```
data.head()
```

Out[54]:

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

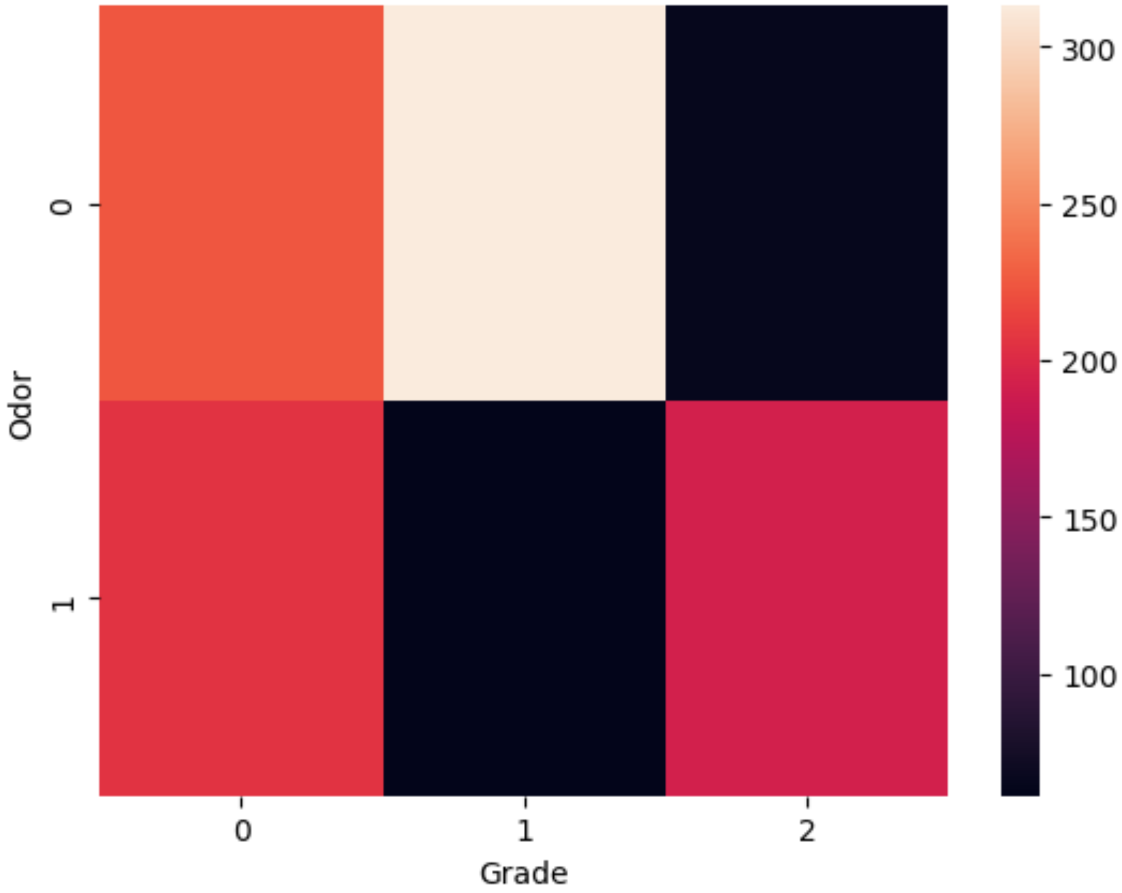
In [59]:
```
sns.heatmap(pd.crosstab(data['Taste'],data['Grade']))
```

Out[59]: <AxesSubplot:xlabel='Grade', ylabel='Taste'>



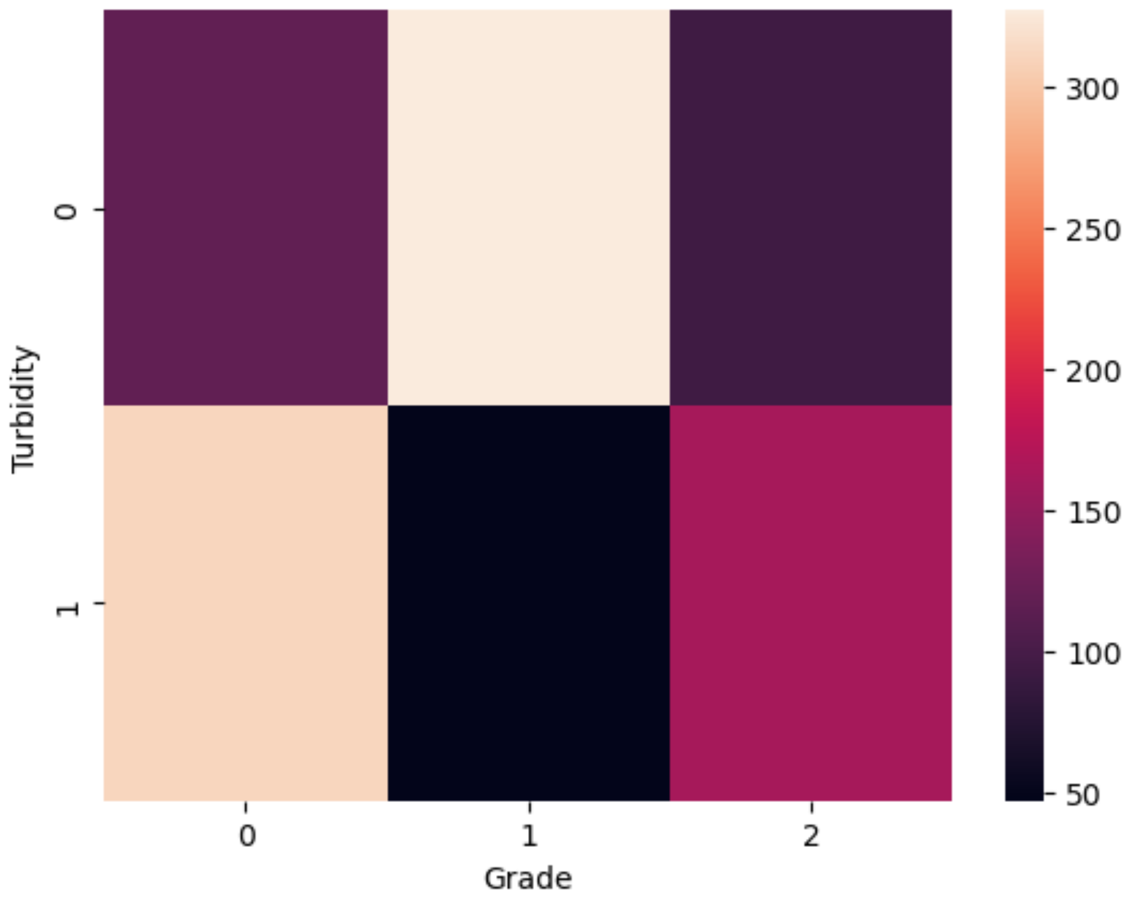
In [60]:
```
sns.heatmap(pd.crosstab(data['Odor'],data['Grade']))
```

Out[60]: <AxesSubplot:xlabel='Grade', ylabel='Odor'>



In [62]:
```
sns.heatmap(pd.crosstab(data['Turbidity'],data['Grade']))
```

Out[62]: <AxesSubplot:xlabel='Grade', ylabel='Turbidity'>



Modelling

I'm installing the necessary packages for estimation.

In [63]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import make_scorer, accuracy_score, roc_auc_sc
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

I split the columns for test and train.

In [64]:
```
x=data.iloc[:,:7]
x.head()
```

Out[64]:

	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

In [65]:
```
y=data.iloc[:,7]
y.head()
```

Out[65]:
```
0    2
1    2
2    0
3    0
4    1
Name: Grade, dtype: int64
```

In [66]:
```
type(data)
```

Out[66]:
```
pandas.core.frame.DataFrame
```

Next I define 80% of the dataframe for training and 20% of the dataframe for testing.

Linear Regression

In [67]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.head()
```

Out[67]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
273	6.8	40	1	1	1	1	255
710	8.6	55	0	1	1	1	255
33	6.7	41	1	0	0	0	247
811	6.6	37	1	0	1	0	255
685	4.7	38	1	0	1	0	255

In [68]:
```
LinReg=LinearRegression()
LinReg.fit(x_train,y_train)
```

Out[68]:
```
▾ LinearRegression
LinearRegression()
```

In [69]:
```
y_predicts =LinReg.predict(x_test)
y_predicts
```

Out[69]:
```
array([ 0.96913698,  0.90655945,  0.96913698,  1.26413904,  0.9691
3698,
        0.05108122,  0.95304397,  0.31928152,  0.38733182,  1.2989
5978,
        0.08517218,  0.31928152,  0.73415691,  1.29895978,  1.0555
0788,
        0.66351468,  1.09647881,  0.88586643,  1.16886361,  1.2989
5978,
        0.2844707 ,  0.95648965,  0.94637422,  0.88046981,  0.1644
5799,
        0.87965597,  1.16349419,  0.77932502,  0.87965597,  0.8055
4381,
        0.68208258,  1.09655274,  0.66351468,  1.60439922,  1.1634
9419,
        0.10013035,  1.02037324,  0.95304397,  0.77932502,  0.8796
5597,
        0.88046981,  0.97567389,  1.02037324,  0.31928152,  0.8858
6643,
        0.2704151 ,  1.16313283,  0.49387323,  1.02037324,  1.0984
6656,
        1.16349419,  0.64660671,  0.88397903,  1.09846656,  1.1631
3283,
        1.09655274,  0.95862752,  0.73824505,  1.16886361,  1.2547
4803,
        0.597758  ,  0.80554381,  1.60439922,  0.96913698,  0.8055
4381,
        0.95304397,  1.15403294,  1.18575701,  1.26413904,  1.1540
3294,
        1.25080036,  1.09655274,  0.73415691,  0.77932502,  1.2547
4803,
        0.80554381,  0.73415691,  1.14471216,  1.15403294,  0.6635
1468,
        0.90655945,  0.73870986,  0.77932502,  1.26413904,  1.1631
3283,
        0.80554381,  1.18575701,  1.09846656,  1.29895978,  0.6635
1468,
        1.09846656,  1.26413904,  1.29895978,  1.09846656,  1.2796
8356,
        1.26413904,  0.90655945,  1.29895978,  0.95304397,  1.0964
7881,
        0.87965597,  0.9019188 ,  0.38733182,  0.66351468,  1.1857
5701,
        1.60439922,  1.09655274,  0.87965597,  1.15403294,  1.1688
```

```
6361,
        1.15410317,  0.898438  ,  1.01191955,  0.9019188 ,  1.0555
0788,
        0.38733182,  0.95648965,  0.16445799,  0.88586643,  1.1634
9419,
        0.2844707 ,  0.10013035,  1.16349419,  0.95862752,  1.2450
4311,
        0.16445799,  1.09846656,  1.09647881,  0.73824505,  1.1447
1216,
        0.88046981,  0.95304397,  1.14471216,  1.15403294,  0.8796
5597,
        0.00274309,  1.14471216,  0.9019188 ,  0.73415691,  1.6043
9922,
        0.73870986,  1.60439922,  1.25474803,  0.73415691,  1.1540
3294,
        0.90655945,  0.08517218,  1.05550788,  0.49387323,  1.0368
437 ,
        0.68208258,  0.73415691,  0.95304397,  0.73415691,  0.6635
1468,
        0.16445799,  1.16349419,  0.9019188 ,  0.88586643,  1.6043
9922,
        1.05550788,  1.60439922,  0.97567389,  0.66351468,  0.8055
4381,
        1.27968356,  0.95304397,  0.95648965,  1.22255708,  1.2508
0036,
        1.01191955,  0.68208258,  1.15403294,  0.3227272 ,  0.9530
4397,
        1.25474803,  0.2844707 ,  0.80487593,  0.08517218,  0.8804
6981,
        0.88586643,  0.80554381,  0.97567389,  0.97567389,  0.9065
5945,
        0.88586643,  1.25080036,  0.16445799,  1.16313283,  0.1644
5799,
        1.15410317,  0.38733182,  0.16445799,  1.04224824,  0.9756
7389,
       -0.7743481 ,  1.09655274,  0.88397903,  0.66351468,  1.1540
3294,
        0.9019188 ,  1.16349419,  1.14471216,  0.49387323,  0.6635
1468,
        0.68208258,  0.3227272 ,  0.97567389,  1.10900161,  0.9691
3698,
        0.95648965,  1.60439922])
```

In [70]:
```
LinReg.score(x_test,y_test)
```

Out[70]:
```
0.291230321538534
```

In [71]:
```
LinReg.score(x_train,y_train)
```

Out[71]:
```
0.2706461051813025
```

Logistic Regression

In [72]:
```
LR = LogisticRegression()
LR.fit(x_train,y_train)
```

Out[72]:
```
▾ LogisticRegression
LogisticRegression()
```

In [73]:
```
y_predicts =LR.predict(x_test)
y_predicts
```

Out[73]:
```
array([1, 0, 1, 2, 1, 0, 2, 0, 0, 1, 0, 0, 1, 1, 2, 0, 0, 0, 1, 1,
0, 0,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 2, 0, 2, 1, 0, 2, 2, 1, 1, 1, 1,
2, 0,
       0, 0, 2, 0, 2, 1, 1, 0, 1, 1, 2, 2, 2, 1, 1, 2, 1, 0, 2, 1,
0, 2,
       1, 1, 2, 1, 2, 2, 1, 1, 2, 0, 1, 1, 1, 0, 0, 1, 1, 2, 2, 0,
1, 1,
       1, 0, 1, 2, 1, 1, 1, 2, 0, 1, 2, 0, 1, 1, 0, 0, 1, 2, 2, 1,
1, 1,
       1, 1, 1, 1, 2, 0, 0, 0, 0, 1, 0, 0, 1, 2, 1, 0, 1, 0, 1, 1,
1, 2,
       1, 1, 1, 0, 1, 1, 1, 2, 1, 2, 2, 1, 1, 0, 0, 2, 0, 1, 0, 1,
2, 1,
       0, 0, 1, 1, 0, 2, 2, 2, 1, 0, 0, 1, 2, 0, 2, 2, 1, 0, 1, 0,
2, 2,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 2, 0, 2, 0, 1, 0, 0, 1, 1, 0,
2, 1,
       0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 2])
```

In [74]:
```
print(confusion_matrix(y_test, y_predicts))
```

```
[[57 14  7]
 [ 5 67  4]
 [ 9 13 36]]
```

In [75]:
```
LR.score(x_train,y_train)
```

Out[75]:
```
0.743801652892562
```

In [76]:
```
LR.score(x_test,y_test)
```

Out[76]:
```
0.7547169811320755
```

In [77]:
```
LR_Predict = LR.predict(x_train)
LR_Accuracy = accuracy_score(y_train, LR_Predict)
print("Accuracy: " + str(LR_Accuracy))
```

```
Accuracy: 0.743801652892562
```

In [78]:
```
resultLR = classification_report(y_test, y_predicts)
print(resultLR)
```

```
              precision    recall  f1-score   support

           0       0.80      0.73      0.77        78
           1       0.71      0.88      0.79        76
           2       0.77      0.62      0.69        58

    accuracy                           0.75       212
   macro avg       0.76      0.74      0.75       212
weighted avg       0.76      0.75      0.75       212
```

Random Forest Classifier

In [79]:
```
x_train,x_test,y_train, y_test=train_test_split(x,y,test_size=0.2)
x_train.head()
```

Out[79]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
803	6.5	36	0	0	1	0	255
216	6.7	38	1	0	1	0	255
110	3.0	40	1	1	1	1	255
1002	6.5	38	1	0	0	0	255
309	6.5	37	0	0	0	0	255

In [80]:
```
RFC = RandomForestClassifier()
RFC.fit(x_train, y_train)
```

Out[80]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [81]:
```
y_predicts =RFC.predict(x_test)
y_predicts
```

Out[81]:
```
array([0, 2, 2, 0, 1, 0, 2, 0, 1, 2, 1, 0, 0, 0, 2, 0, 0, 1, 0, 0,
2, 2,
       0, 1, 1, 1, 0, 1, 0, 1, 0, 2, 1, 0, 0, 0, 1, 2, 1, 1, 2, 1,
0, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 2,
1, 0,
       0, 0, 1, 1, 1, 1, 1, 2, 1, 0, 1, 1, 1, 0, 0, 2, 1, 0, 2, 2,
1, 1,
       0, 1, 2, 0, 1, 1, 0, 0, 1, 1, 0, 2, 0, 1, 1, 0, 1, 0, 1, 0,
2, 0,
       1, 2, 0, 2, 0, 0, 0, 0, 1, 2, 1, 1, 0, 1, 0, 2, 1, 2, 0, 0,
0, 0,
       1, 1, 1, 2, 2, 1, 2, 2, 0, 0, 1, 2, 0, 0, 1, 1, 1, 1, 1, 1,
0, 1,
       0, 1, 1, 2, 1, 0, 0, 1, 0, 0, 0, 2, 2, 0, 1, 0, 2, 0, 2, 0,
1, 2,
       1, 1, 2, 2, 1, 2, 0, 1, 1, 1, 0, 1, 2, 1, 2, 1, 2, 0, 2, 2,
2, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 2, 1, 2])
```

In [82]:
```
print(confusion_matrix(y_test, y_predicts))
```

```
[[82  0  0]
 [ 0 84  0]
 [ 0  0 46]]
```

In [83]:
```
RFC.score(x_train,y_train)
```

Out[83]:
```
1.0
```

In [84]:
```
RFC.score(x_test,y_test)
```

Out[84]:
```
1.0
```

In [85]:
```
resultRFC = classification_report(y_test, y_predicts)
print(resultRFC)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        82
           1       1.00      1.00      1.00        84
           2       1.00      1.00      1.00        46

    accuracy                           1.00       212
   macro avg       1.00      1.00      1.00       212
weighted avg       1.00      1.00      1.00       212
```

In [86]:
```
RFC_Predict = RFC.predict(x_train)
RFC_Accuracy = accuracy_score(y_train, RFC_Predict)
print("Accuracy: " + str(RFC_Accuracy))
```

```
Accuracy: 1.0
```

K-Nearest Neighbors Classifier

In [87]:
```
x_train,x_test,y_train, y_test=train_test_split(x,y,test_size=0.2)
x_train.head()
```

Out[87]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
19	6.8	40	1	0	1	0	245
968	8.1	66	1	0	1	1	255
381	3.0	40	1	1	1	1	255
512	6.8	45	1	1	1	0	245
443	6.8	45	1	1	1	1	245

In [88]:
```
KNN = KNeighborsClassifier()
KNN.fit(x_train, y_train)
```

Out[88]:

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [89]:
```
y_predicts =KNN.predict(x_test)
y_predicts
```

Out[89]:
```
array([0, 0, 2, 1, 1, 0, 1, 1, 1, 2, 2, 0, 1, 1, 1, 0, 0, 0, 1, 1,
0, 1,
       0, 2, 1, 0, 1, 0, 2, 1, 1, 1, 2, 1, 0, 0, 0, 1, 2, 1, 2, 1,
1, 0,
       1, 1, 0, 0, 1, 2, 2, 2, 1, 1, 1, 2, 0, 1, 0, 1, 2, 1, 1, 2,
0, 2,
       0, 0, 1, 0, 0, 1, 0, 2, 0, 1, 2, 0, 0, 1, 1, 1, 1, 2, 0, 1,
2, 0,
       1, 0, 1, 0, 1, 1, 2, 0, 0, 1, 0, 1, 2, 2, 0, 2, 1, 0, 2, 1,
1, 1,
       0, 2, 0, 0, 2, 0, 0, 0, 2, 1, 2, 1, 1, 1, 1, 0, 2, 1, 1, 2,
2, 2,
       2, 1, 1, 0, 1, 2, 0, 0, 0, 1, 0, 1, 0, 0, 2, 0, 2, 2, 1, 2,
1, 1,
       0, 2, 0, 0, 0, 0, 2, 2, 0, 1, 0, 2, 1, 2, 0, 0, 1, 0, 0, 0,
2, 0,
       1, 0, 0, 2, 0, 0, 0, 2, 0, 1, 2, 2, 1, 0, 0, 0, 0, 0, 1, 1,
0, 2,
       0, 2, 1, 0, 1, 1, 2, 1, 2, 0, 2, 2, 2, 1])
```

In [90]:
```
print(confusion_matrix(y_test, y_predicts))
```

```
[[80  1  0]
 [ 0 76  1]
 [ 0  0 54]]
```

In [91]:
```
KNN.score(x_train,y_train)
```

Out[91]: 0.9917355371900827

In [92]:
```
KNN.score(x_test,y_test)
```

Out[92]: 0.9905660377358491

In [93]:
```
resultKNN = classification_report(y_test, y_predicts)
print(resultKNN)
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        81
           1       0.99      0.99      0.99        77
           2       0.98      1.00      0.99        54

    accuracy                           0.99       212
   macro avg       0.99      0.99      0.99       212
weighted avg       0.99      0.99      0.99       212
```

In [94]:
```
KNN_Predict = KNN.predict(x_train)
KNN_Accuracy = accuracy_score(y_train, KNN_Predict)
print("Accuracy: " + str(KNN_Accuracy))
```

```
Accuracy: 0.9917355371900827
```

Decision Tree Classifier

In [95]:
```
x_train,x_test,y_train, y_test=train_test_split(x,y,test_size=0.2)
x_train.head()
```

Out[95]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
483	6.6	45	0	0	0	1	250
321	6.7	38	1	0	1	0	255
69	3.0	40	1	1	1	1	255
133	6.8	36	0	1	1	0	253
937	8.1	66	1	0	1	1	255

In [96]:
```
DT = DecisionTreeClassifier()
DT.fit(x_train, y_train)
```

Out[96]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [97]:
```
y_predicts =DT.predict(x_test)
y_predicts
```

Out[97]:
```
array([1, 1, 0, 2, 0, 1, 1, 1, 1, 2, 0, 0, 0, 1, 0, 2, 1, 0, 1, 2,
0, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 0, 1,
1, 1,
       0, 1, 2, 0, 1, 2, 1, 0, 0, 1, 2, 0, 2, 1, 1, 0, 0, 2, 1, 1,
2, 0,
       1, 2, 0, 0, 1, 2, 0, 0, 0, 2, 1, 1, 1, 2, 0, 2, 0, 0, 1, 1,
0, 0,
       0, 0, 2, 2, 1, 0, 2, 1, 1, 1, 2, 2, 0, 1, 1, 0, 2, 0, 0, 1,
0, 0,
       2, 2, 2, 0, 0, 1, 0, 0, 2, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1,
0, 1,
       2, 1, 1, 1, 2, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 2, 2, 0, 1, 1,
1, 2,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 2, 0, 2, 2, 1, 1, 1, 1, 0, 1, 0,
0, 2,
       1, 2, 1, 0, 0, 2, 2, 0, 2, 0, 1, 2, 2, 0, 2, 2, 2, 0, 1, 2,
1, 1,
       1, 1, 0, 1, 0, 1, 0, 0, 2, 2, 0, 1, 2, 0])
```

In [98]:
```
print(confusion_matrix(y_test, y_predicts))
```

```
[[80  0  2]
 [ 0 81  0]
 [ 1  0 48]]
```

In [99]:
```
DT.score(x_train,y_train)
```

Out[99]:
```
1.0
```

In [100]:
```
DT.score(x_test,y_test)
```

Out[100]:
```
0.9858490566037735
```

In [101]:
```
resultDT = classification_report(y_test, y_predicts)
print(resultDT)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98        82
           1       1.00      1.00      1.00        81
           2       0.96      0.98      0.97        49

    accuracy                           0.99       212
   macro avg       0.98      0.99      0.98       212
weighted avg       0.99      0.99      0.99       212
```

In [102]:
```
DT_Predict = DT.predict(x_train)
DT_Accuracy = accuracy_score(y_train, DT_Predict)
print("Accuracy: " + str(DT_Accuracy))
```

```
Accuracy: 1.0
```

Model Performance Summary

I created a list where I could review model performances and compared them.

In [103]:
```
model_performance_accuracy = pd.DataFrame({'Model': ['LogisticRegre
```

In [104]:
```
model_performance_accuracy.sort_values(by = "Accuracy", ascending =
```

Out[104]:

	Model	Accuracy
1	RandomForestClassifier	1.000000
3	DecisionTreeClassifier	1.000000
2	KNeighborsClassifier	0.991736
0	LogisticRegression	0.743802

In [105]:
```
data.head()
```

Out[105]:

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

Now you can learn the quality of the milk you will obtain by entering the values you want respectively.

In [106]:
```
output=RFC.predict([[6.6,1,1,1,1,1,240]])
```

In [107]:
```
if output == 2:
    print("The milk quality is 'Good'")
if output == 1:
    print("The milk quality is 'Moderate'")
if output == 0:
    print("The milk quality is 'Bad'")
```

```
The milk quality is 'Good'
```

In [ ]: