Milk Quality Prediction

Our goal in this project is to design a machine learning model and to predict quality of milk using the data we have, ensuring the safety and excellence of dairy products.

Data Set - Variables

- pH: This Column defines PH alus of the milk which ranges from 3 to 9.5 max : 6.25 to 6.90
- Temperature: This Column defines Temprature of the milk which ranges from 34'C to 90'C max : 34'C to 45.20'C
- Taste: This Column defines Taste of the milk which is categorical data 0 (Bad) or 1 (Good) max : 1 (Good)
- Odor: This Column defines Odor of the milk which is categorical data 0 (Bad) or 1 (Good) max : 0 (Bad)
- Fat: This Column defines Fat of the milk which is categorical data 0 (Low) or 1 (High) max : 1 (High)
- Turbidity: This Column defines Turbidity of the milk which is categorical data 0 (Low) or 1 (High) max : 1 (High)
- Colour: This Column defines Colour of the milk which ranges from 240 to 255 max : 255
- Grade: This Column defines Grade (Target) of the milk which is categorical data Where Low(Bad), Medium(Moderate), High(Good)

Import libraries

In [151…

```
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
```

Read the data

In [152…

```
df = pd.read_csv("milkpred2.csv")
df.head()
```

Out[152]:

	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35.0	1.0	0.0	1.0	0.0	254.0	high
1	6.6	36.0	0.0	1.0	0.0	1.0	253.0	high
2	8.5	70.0	1.0	1.0	1.0	1.0	246.0	medium
3	9.5	34.0	1.0	1.0	0.0	1.0	255.0	low
4	6.6	37.0	0.0	0.0	0.0	0.0	255.0	low

In [153…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1074 entries, 0 to 1073
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   pH           1067 non-null   float64
 1   Temperature  1070 non-null   float64
 2   Taste        1073 non-null   float64
 3   Odor         1067 non-null   float64
 4   Fat          1070 non-null   float64
 5   Turbidity    1064 non-null   float64
 6   Colour       1065 non-null   float64
 7   Grade        1074 non-null   object 
dtypes: float64(7), object(1)
memory usage: 67.3+ KB
```

In [154…
```
df.shape
```

Out[154]:
```
(1074, 8)
```

Data preprocessing

Label Encoding

We have 7 numeric values and 1 categorical value. Converting the categorical variable (grade) into numerical.

In [155…
```
# Importing LabelEncoder
from sklearn.preprocessing import LabelEncoder
LE=LabelEncoder()
# Encoding Transmission with the help of Label Encoder
df['Grade']=LE.fit_transform(df['Grade']).astype(int)
df.head()
```

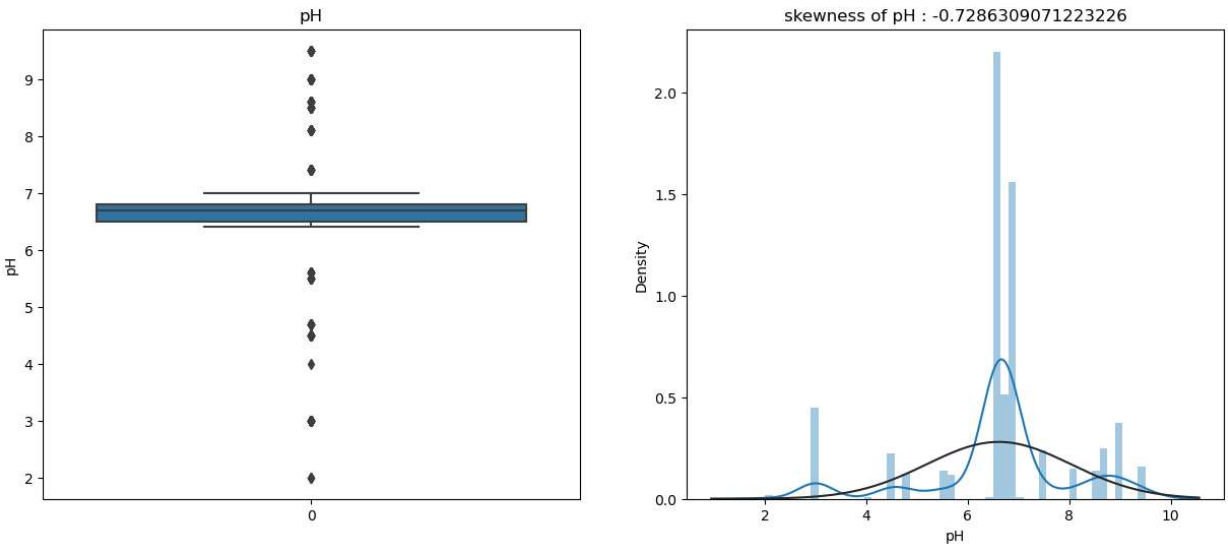
Out[155]:

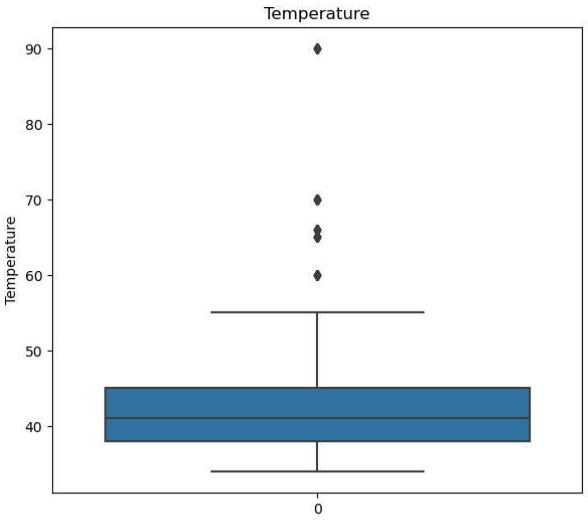
	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35.0	1.0	0.0	1.0	0.0	254.0	0
1	6.6	36.0	0.0	1.0	0.0	1.0	253.0	0
2	8.5	70.0	1.0	1.0	1.0	1.0	246.0	2
3	9.5	34.0	1.0	1.0	0.0	1.0	255.0	1
4	6.6	37.0	0.0	0.0	0.0	0.0	255.0	1

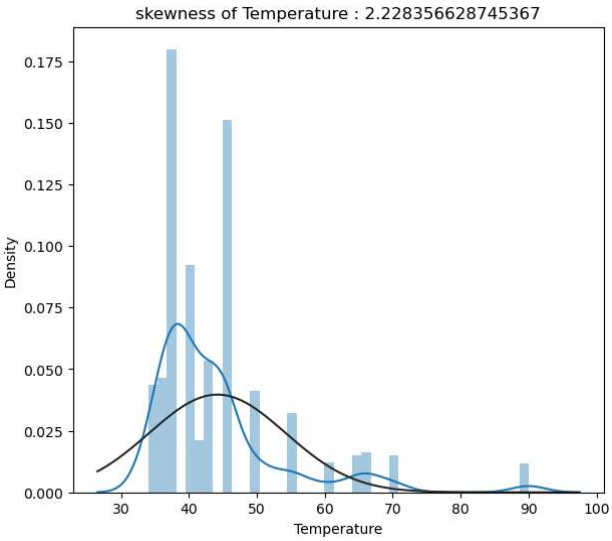
Graphical Analysis

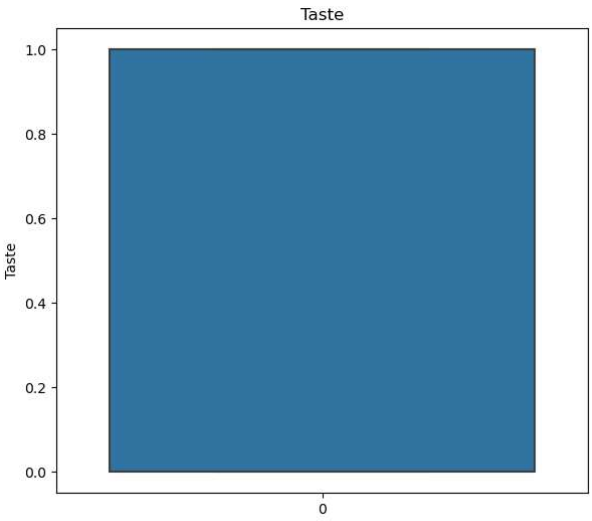
In [156…

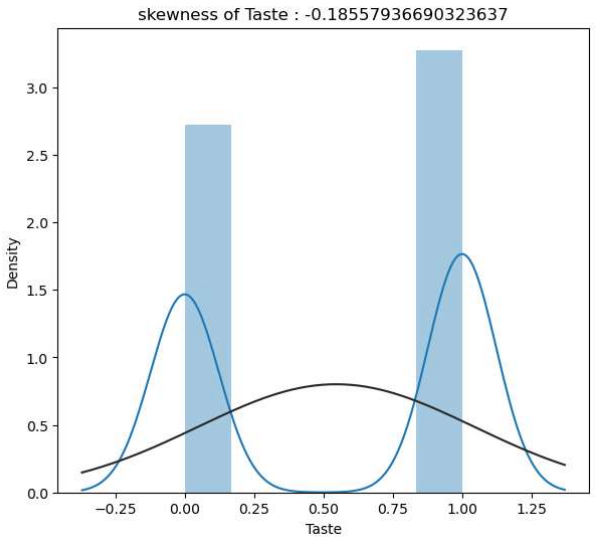
```
# Checking for outliers in numerical variables using boxplot
from scipy.stats import norm
num_var=[var for var in df.columns]

# Plotting Box and Distribution plot
for var in num_var:
    plt.figure(figsize=(15,6))
    plt.subplot(1,2,1)
    ax=sns.boxplot(data=df[var])
    ax.set_title(f'{var}')
    ax.set_ylabel(var)

    plt.subplot(1,2,2)
    ax=sns.distplot(df[var], fit=norm)
    ax.set_title(f'skewness of {var} : {df[var].skew()}')
    ax.set_xlabel(var)
    print('__'*50)
    plt.show()
```

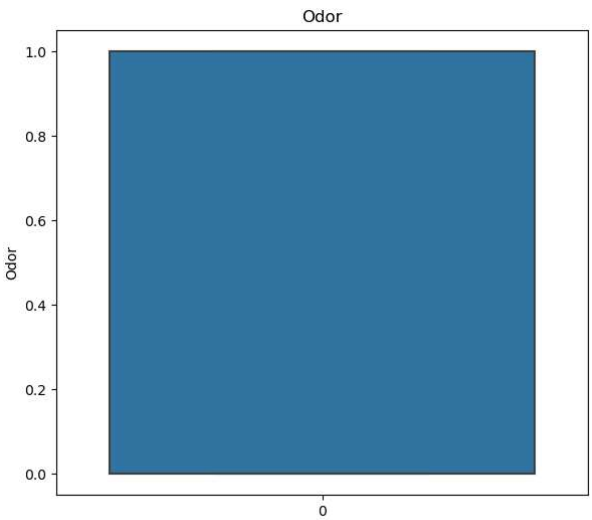


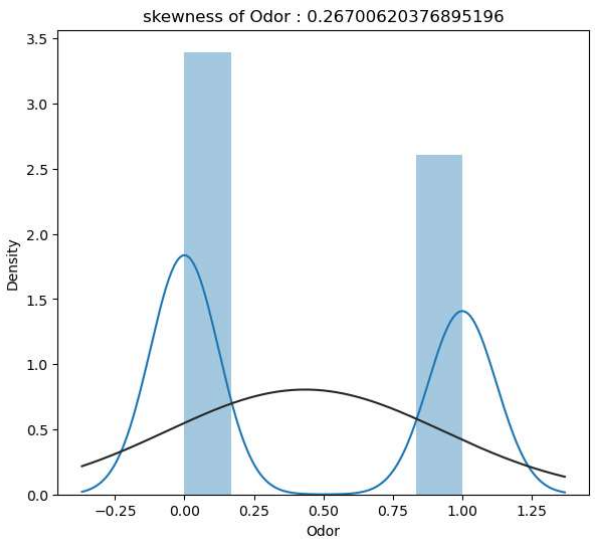

Temperature
Temperature
0

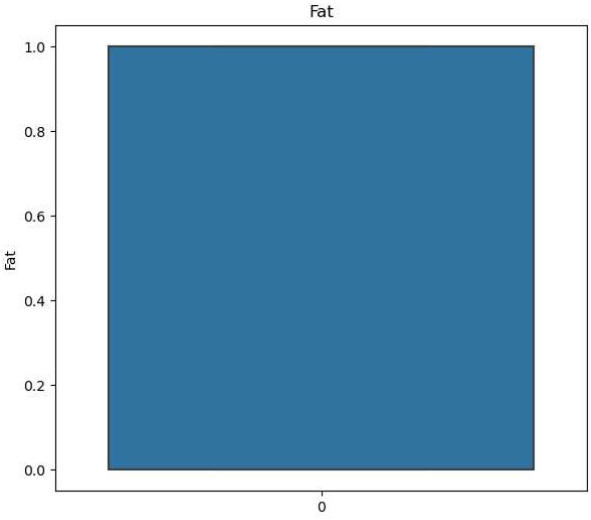

skewness of Temperature : 2.228356628745367
Density
Temperature

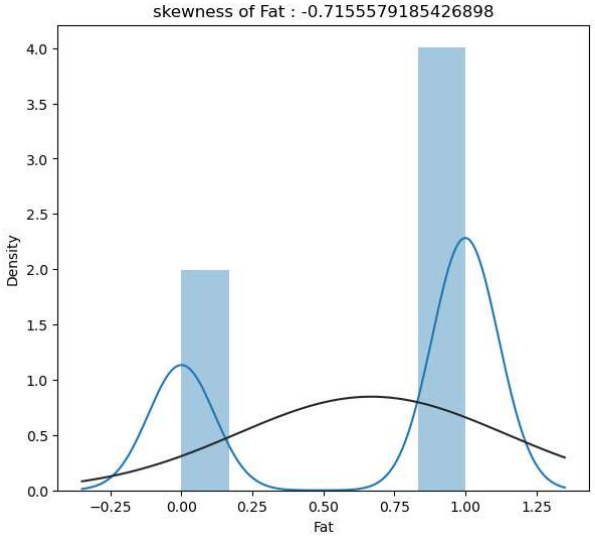

Taste
Taste
0

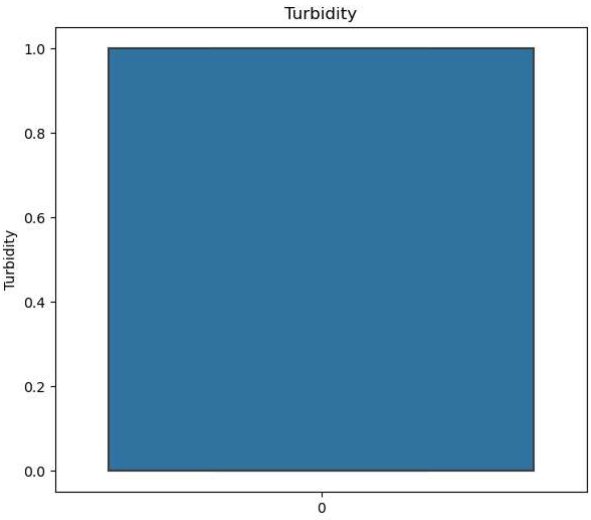

skewness of Taste : -0.18557936690323637
Density
Taste

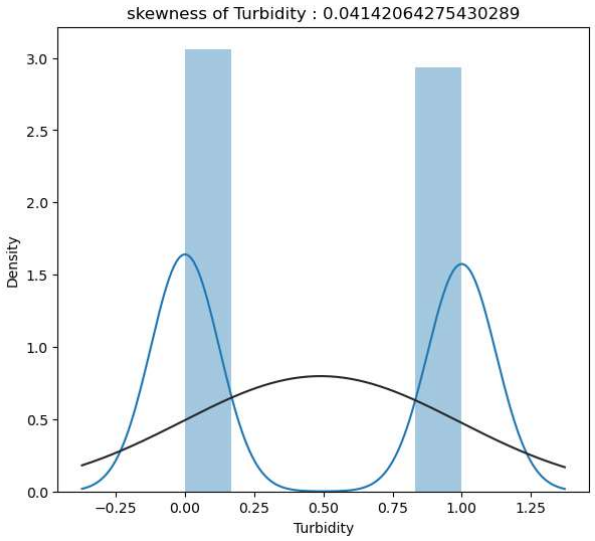

Odor
Odor
0

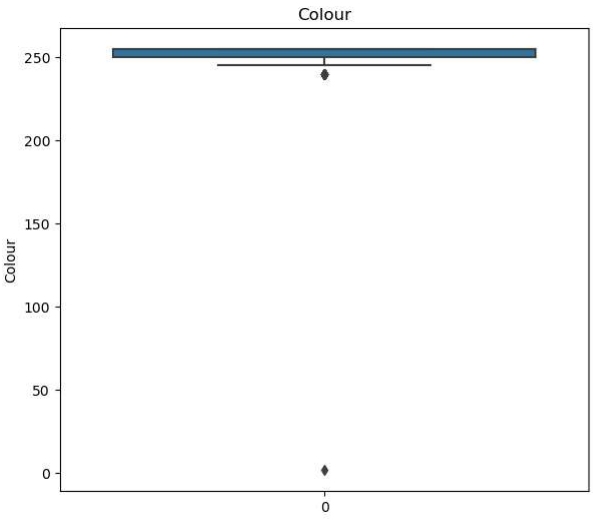

skewness of Odor : 0.26700620376895196
Density
Odor

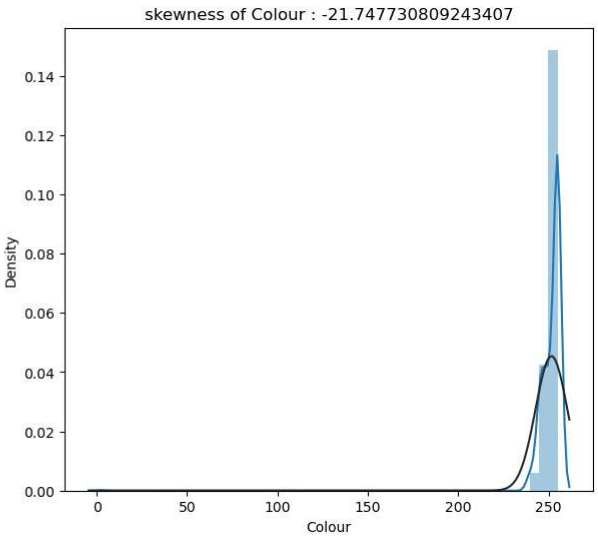

Fat
Fat
0

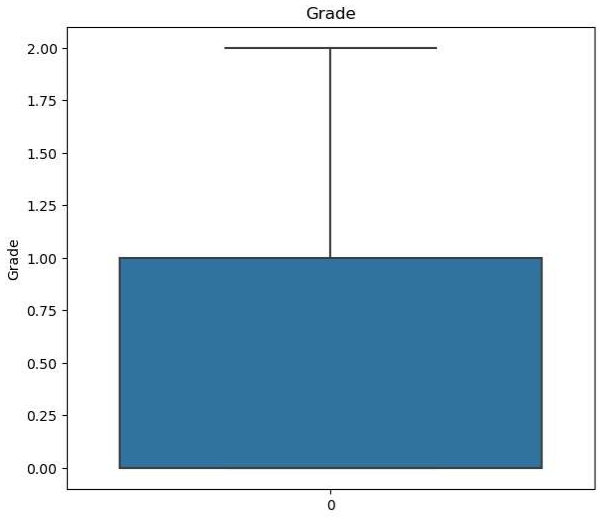

skewness of Fat : -0.7155579185426898
Density
Fat

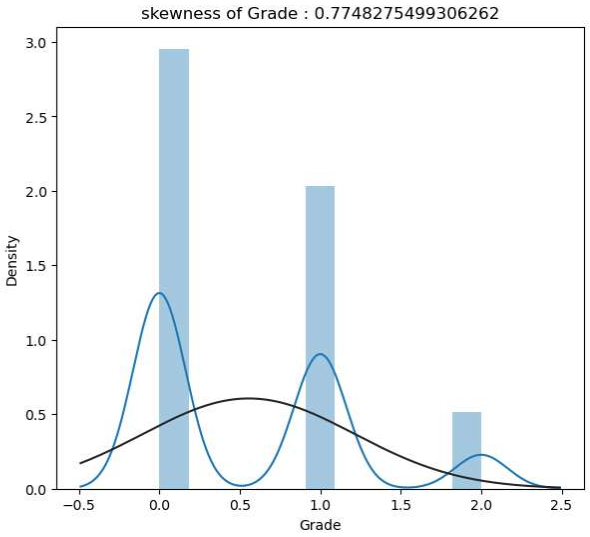

Turbidity
Turbidity
0


skewness of Turbidity : 0.04142064275430289
Density
Turbidity


Colour
Colour
0

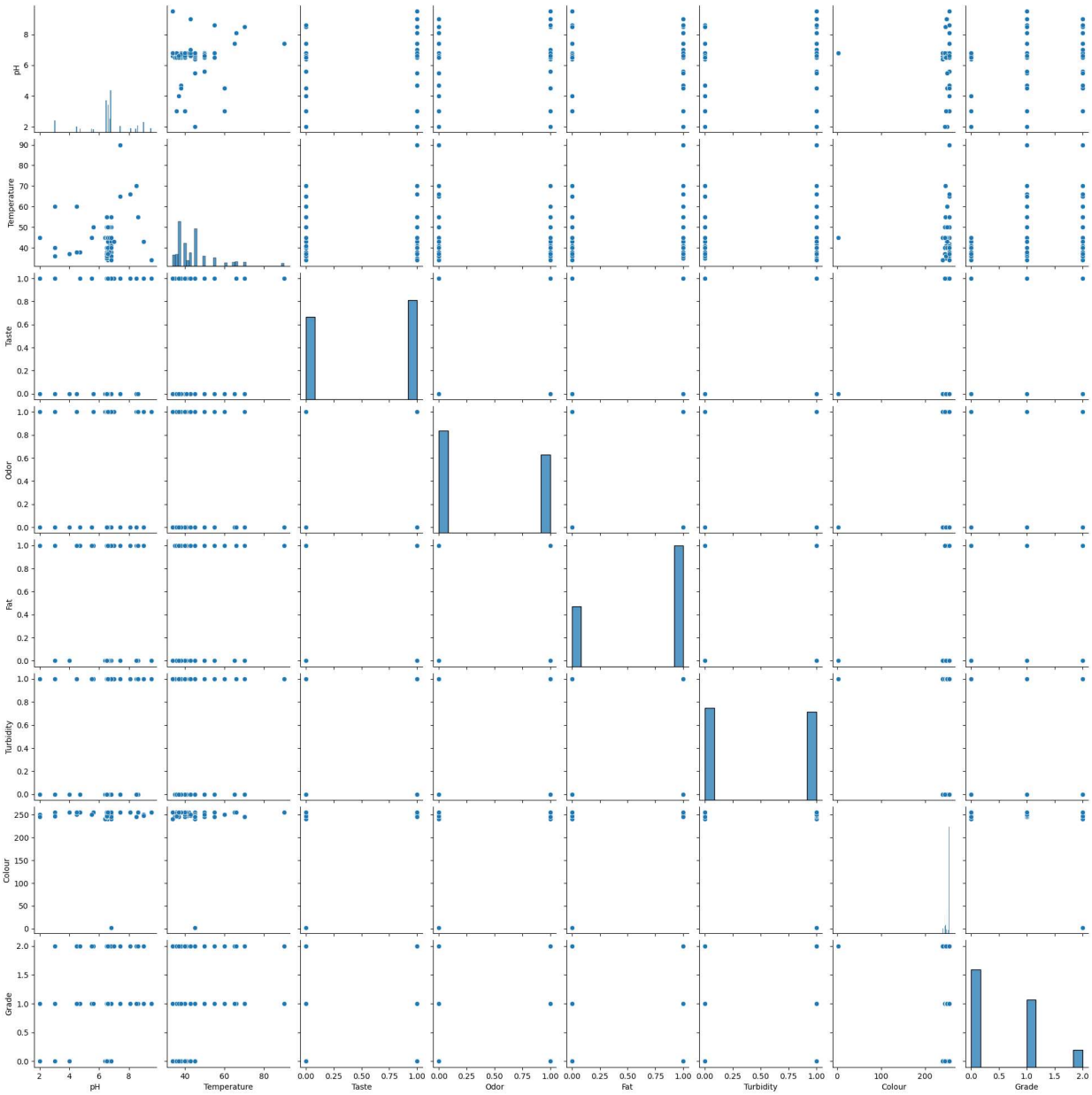

skewness of Colour : -21.747730809243407
Density
Colour



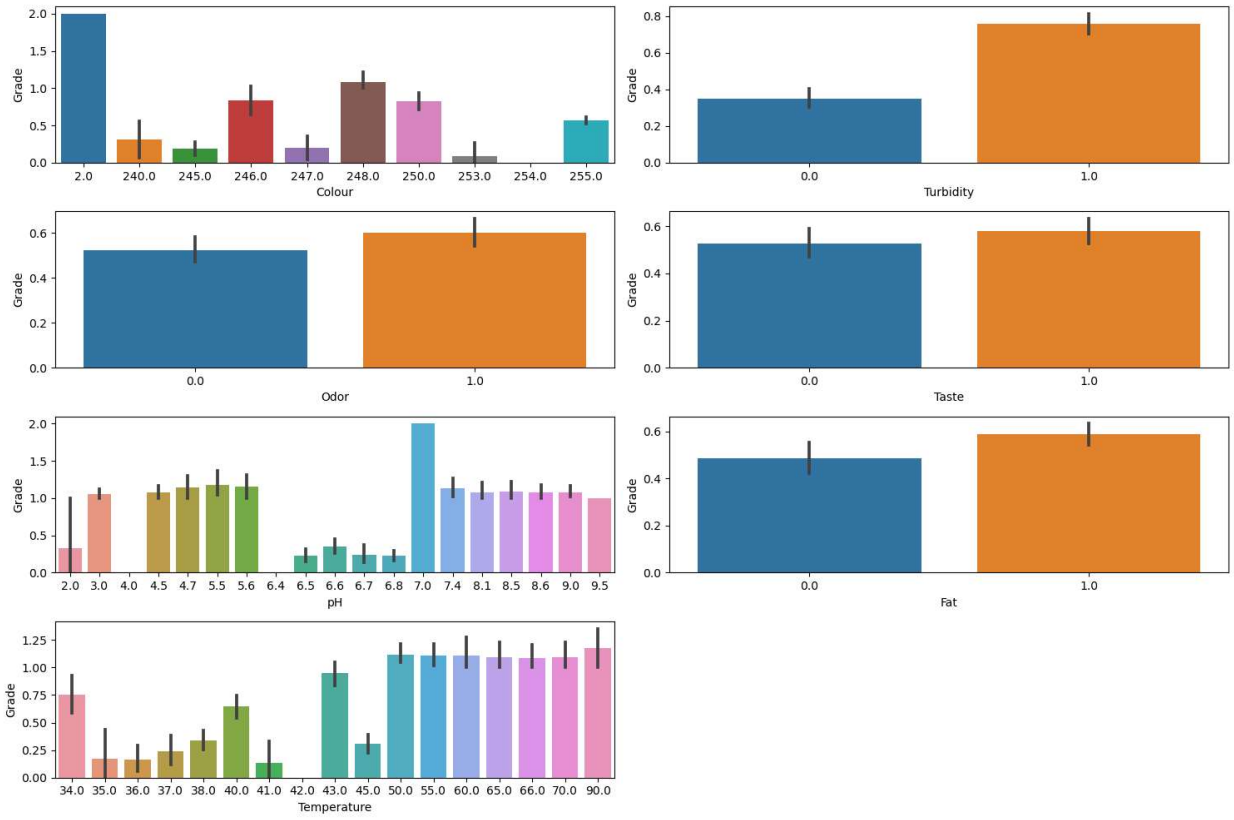


In [157…
```
sns.pairplot(df)
```

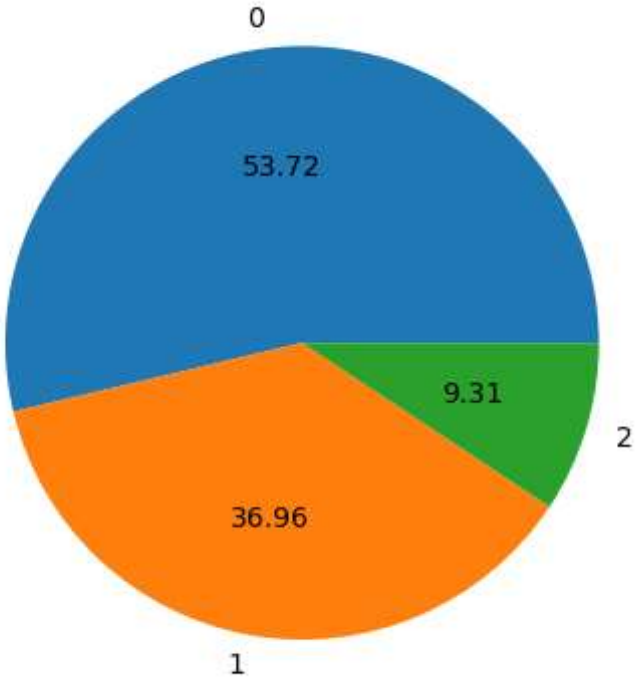
Out[157]:
```
<seaborn.axisgrid.PairGrid at 0x235dafe3b50>
```



In [158…
```
countplot_list = ["Colour","Turbidity","Odor","Taste","pH","Fat","Temperature"]
plt.figure(figsize=(15,10))
x=1
for i in countplot_list:
    plt.subplot(4,2,x)
    sns.barplot(x=i,y='Grade',data=df)
    x = x + 1
plt.tight_layout()
```



In [159…
```
palette_color = sns.color_palette('bright')

# plotting data on chart
df.groupby('Grade').size().plot(kind='pie', autopct='%.2f')

# displaying chart
plt.show()
```



Handling Outliers

In [160…

```
#Checking for outliers value in pH,Colour,Temperature,Taste,Odor,Fat,Turbidity,Grade
att=['pH','Colour','Temperature','Taste','Odor','Fat','Turbidity','Grade']
q1, q3 = np.percentile(df[att], [25, 75])
iqr = q3 - q1
lower_bound = q1 - 1.5*iqr
upper_bound = q3 + 1.5*iqr
print(upper_bound)
outliers = df[(df[att] < lower_bound) | (df[att] > upper_bound)]
outliers
```

```
nan
```

Out[160]:

	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
1069	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1070	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1071	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1072	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1073	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1074 rows × 8 columns

Missing values

In [161...
```
df.isnull().sum()
```

Out[161]:
```
pH             7
Temperature    4
Taste          1
Odor           7
Fat            4
Turbidity     10
Colour         9
Grade          0
dtype: int64
```

In [162...
```
# pH
value_to_fill=df['pH'].mean().round(1)
df['pH'].fillna(value_to_fill,inplace=True)
```

In [163...
```
# Temperature
value_to_fill=df['Temperature'].mean().round(1)
df['Temperature'].fillna(value_to_fill,inplace=True)
```

In [164...
```
# Odor
value_to_fill=df['Odor'].mean().round(1)
df['Odor'].fillna(value_to_fill,inplace=True)
```

In [165...
```
# Fat
value_to_fill=df['Fat'].mean().round(1)
df['Fat'].fillna(value_to_fill,inplace=True)
```

In [166...
```
# Taste
value_to_fill=df['Taste'].mean().round(1)
```

```
df['Taste'].fillna(value_to_fill,inplace=True)
```

In [167…
```
# Turbidity
value_to_fill=df['Turbidity'].mean().round(1)
df['Turbidity'].fillna(value_to_fill,inplace=True)
```

In [168…
```
# Colour
value_to_fill=df['Colour'].mean().round(1)
df['Colour'].fillna(value_to_fill,inplace=True)
```

In [169…
```
att=["pH","Temperature","Taste","Odor","Fat","Turbidity","Colour"]
miss_bool = df[att].isnull()
miss_bool
```

Out[169]:

	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
…	…	…	…	…	…	…	…
1069	False	False	False	False	False	False	False
1070	False	False	False	False	False	False	False
1071	False	False	False	False	False	False	False
1072	False	False	False	False	False	False	False
1073	False	False	False	False	False	False	False

1074 rows × 7 columns

In [170…
```
df.isnull().sum()
```

Out[170]:
```
pH             0
Temperature    0
Taste          0
Odor           0
Fat            0
Turbidity      0
Colour         0
Grade          0
dtype: int64
```

Statistical description of the data

In [171…
```
df.describe()
```

Out[171]:

	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour	1
count	1074.000000	1074.000000	1074.000000	1074.000000	1074.000000	1074.000000	1074.000000	
mean	6.610801	44.184171	0.546089	0.433706	0.668343	0.489758	251.613966	
std	1.420878	10.041328	0.497869	0.494235	0.470196	0.497793	8.741839	
min	2.000000	34.000000	0.000000	0.000000	0.000000	0.000000	2.000000	
25%	6.500000	38.000000	0.000000	0.000000	0.000000	0.000000	250.000000	
50%	6.700000	41.000000	1.000000	0.000000	1.000000	0.000000	255.000000	
75%	6.800000	45.000000	1.000000	1.000000	1.000000	1.000000	255.000000	
max	9.500000	90.000000	1.000000	1.000000	1.000000	1.000000	255.000000	

In [172…

```
df.corr()
```

Out[172]:

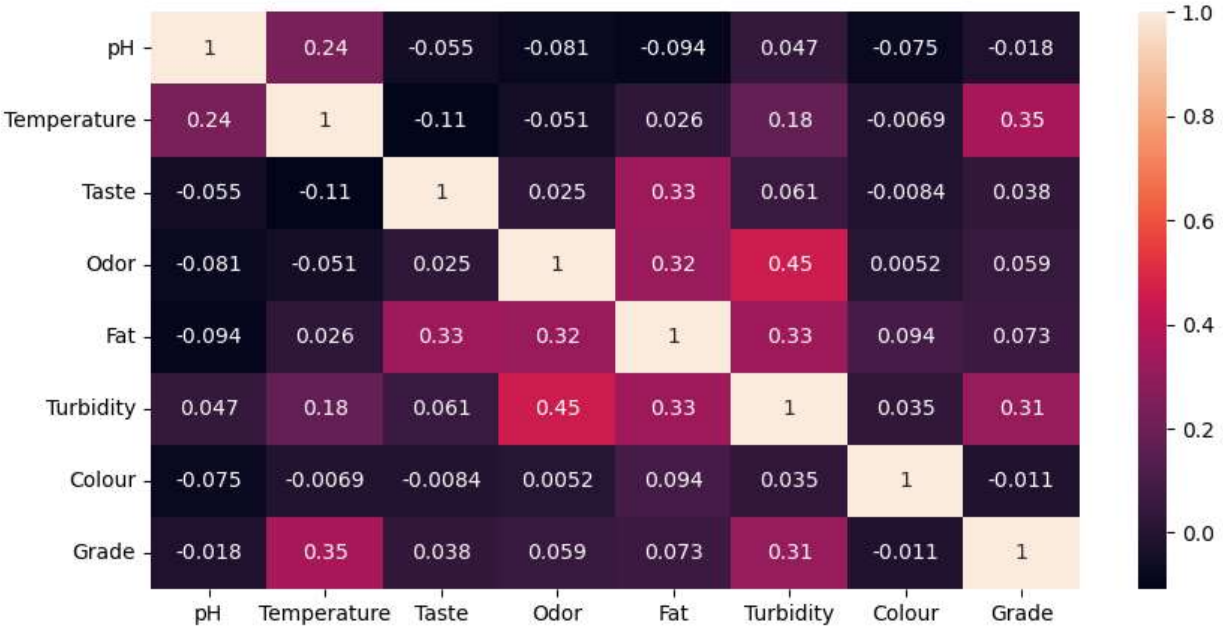
	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour	Grade
pH	1.000000	0.235884	-0.054983	-0.080677	-0.094234	0.046669	-0.075259	-0.017681
Temperature	0.235884	1.000000	-0.108947	-0.050927	0.026471	0.177691	-0.006857	0.352252
Taste	-0.054983	-0.108947	1.000000	0.025496	0.327316	0.061133	-0.008371	0.038338
Odor	-0.080677	-0.050927	0.025496	1.000000	0.316765	0.453321	0.005240	0.059221
Fat	-0.094234	0.026471	0.327316	0.316765	1.000000	0.327903	0.094026	0.072549
Turbidity	0.046669	0.177691	0.061133	0.453321	0.327903	1.000000	0.035135	0.308847
Colour	-0.075259	-0.006857	-0.008371	0.005240	0.094026	0.035135	1.000000	-0.011033
Grade	-0.017681	0.352252	0.038338	0.059221	0.072549	0.308847	-0.011033	1.000000

In [173…

```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),annot=True)
```

Out[173]:

```
<Axes: >
```



We can see that 'Temperature' and 'Taste' have strong positive correlation.

And 'Temperature' and 'Grade' have strong negative correlation

Modelling

In [174…
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

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

In [175…
```
x=df.iloc[:,:7]
x.head()
```

Out[175]:

	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour
0	6.6	35.0	1.0	0.0	1.0	0.0	254.0
1	6.6	36.0	0.0	1.0	0.0	1.0	253.0
2	8.5	70.0	1.0	1.0	1.0	1.0	246.0
3	9.5	34.0	1.0	1.0	0.0	1.0	255.0
4	6.6	37.0	0.0	0.0	0.0	0.0	255.0

In [176…
```
y=df.iloc[:,7]
y.head()
```

Out[176]:
```
0    0
1    0
2    2
3    1
4    1
Name: Grade, dtype: int32
```

In [177…
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.head()
y_train.head()
```

Out[177]:
```
308    1
154    2
827    0
750    0
894    0
Name: Grade, dtype: int32
```

Logistic Regression

In [178…
```
LR = LogisticRegression()
LR.fit(x_train,y_train)
```

Out[178]:
```
LogisticRegression()
```

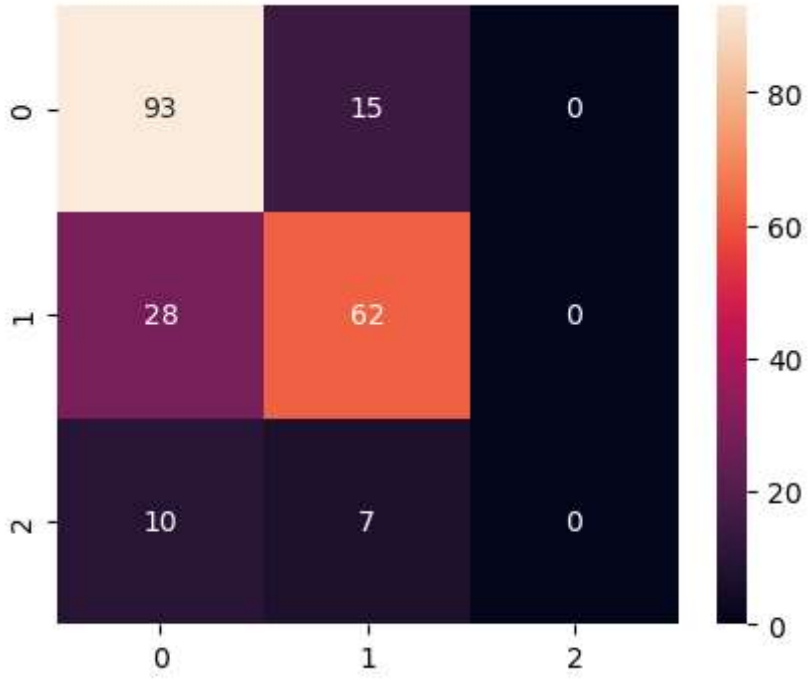
In [179…
```
y_predicts =LR.predict(x_test)
y_predicts
```

Out[179]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1])
```

In [180…
```
print(confusion_matrix(y_test, y_predicts))
```

```
[[93 15  0]
 [28 62  0]
 [10  7  0]]
```

In [181…
```
plt.figure(figsize=(5,4))
sns.heatmap(confusion_matrix(y_test, y_predicts), annot=True);
```



In [182…
```
LR.score(x_train,y_train)
```

Out[182]:
```
0.7066356228172294
```

In [183…
```
LR.score(x_test,y_test)
```

Out[183]:
```
0.7209302325581395
```

Classification report for RF model

In [184…
```
LR_Predict = LR.predict(x_train)
LR_Accuracy = accuracy_score(y_train, LR_Predict)
print("Accuracy: " + str(LR_Accuracy))
```

```
Accuracy: 0.7066356228172294
```

In [185…
```
resultLR = classification_report(y_test, y_predicts)
print(resultLR)
```

```
              precision    recall  f1-score   support

           0       0.71      0.86      0.78       108
           1       0.74      0.69      0.71        90
           2       0.00      0.00      0.00        17

    accuracy                           0.72       215
   macro avg       0.48      0.52      0.50       215
weighted avg       0.67      0.72      0.69       215
```

Random Forest Classifier

In [186…
```
x_train,x_test,y_train, y_test=train_test_split(x,y,test_size=0.33)
x_train.head()
```

Out[186]:

	pH	Temperature	Taste	Odor	Fat	Turbidity	Colour
486	6.8	45.0	1.0	1.0	1.0	1.0	245.0
637	3.0	40.0	1.0	1.0	1.0	1.0	251.6
338	6.7	45.0	1.0	1.0	1.0	0.0	245.0
233	8.5	70.0	0.0	0.0	0.0	0.0	246.0
576	6.6	40.0	1.0	0.0	1.0	1.0	255.0

In [187…
```
RFC = RandomForestClassifier()
RFC.fit(x_train, y_train)
```

Out[187]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [188…
```
y_predicts =RFC.predict(x_test)
y_predicts
```

Out[188]:
```
array([1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 2, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       1, 2, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 1])
```

In [189…
```
print(confusion_matrix(y_test, y_predicts))
```

```
[[187   0   1]
 [  2 131   1]
 [ 19  14   0]]
```

In [190…
```
RFC.score(x_train,y_train)
```

Out[190]:
```
0.9165507649513213
```

In [191…
```
RFC.score(x_test,y_test)
```

Out[191]:
```
0.895774647887324
```

Classification report for RF model

In [192…
```
resultRFC = classification_report(y_test, y_predicts)
print(resultRFC)
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       188
           1       0.90      0.98      0.94       134
           2       0.00      0.00      0.00        33

    accuracy                           0.90       355
   macro avg       0.60      0.66      0.63       355
weighted avg       0.82      0.90      0.85       355
```

In [193…
```
RFC_Predict = RFC.predict(x_train)
RFC_Accuracy = accuracy_score(y_train, RFC_Predict)
print("Accuracy: " + str(RFC_Accuracy))
```

```
Accuracy: 0.9165507649513213
```

In [194…
```
pd.DataFrame({'Actual': y_test,  'Predicted': y_predicts}).head(100)
```

Out[194]:

	Actual	Predicted
371	2	1
656	1	1
728	1	1
827	0	0
489	1	1
…	…	…
478	0	0
653	0	0
516	0	0
37	1	1
841	1	1

100 rows × 2 columns

In [195…
```
model_performance_accuracy = pd.DataFrame({'Model': ['LogisticRegression', 'RandomFore
model_performance_accuracy.sort_values(by = "Accuracy", ascending = False)
```

Out[195]:

	Model	Accuracy
1	RandomForestClassifier	0.916551
0	LogisticRegression	0.706636

After a thorough exploration and evaluation of both machine learning models, it is evident that the Random Forest Classifier stands out as the most efficient and accurate model for predicting milk quality.