Milk Quality Prediction (Classification)

About dataset :

This dataset is manually collected from observations. It helps us to build machine learning models to predict quality of milk.

This dataset consists of 7 independent variables i.e pH, Temperature, Taste, Odor, Fat, Turbidity, Color. Generally, Grade or Quality of the milk depends on these parameters. These parameters plays a vital role in predictive analysis of the milk.

Usage :

Target variable is nothing but Grade of the milk. It can be

Low (Bad)

Medium (Moderate)

High (Good)

We have to perform data preprocessing, data augmentation techniques to build statistical and predictive models to predict the quality of the milk.

Inspiration :

To leverage the benefits of machine learning in the dairy industry.

- pH - This Column defines PH alus of the milk which ranges from 3 to 9.5 max : 6.25 to 6.90
- Temprature - 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) or Medium (Moderate) High (Good)

Importing Packages

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Loading and Evaluating Dataset

In [2]:
```
df=pd.read_csv('milknew.csv')
df.head()
```

Out[2]:

	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 [3]:
```
df.shape

print('No. of rows :',df.shape[0])
print('No. of columns :',df.shape[1])
```

```
No. of rows : 1059
No. of columns : 8
```

In [4]:
```
df.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 [5]:
```
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
pH	1059.0	6.630123	1.399679	3.0	6.5	6.7	6.8	9.5
Temprature	1059.0	44.226629	10.098364	34.0	38.0	41.0	45.0	90.0
Taste	1059.0	0.546742	0.498046	0.0	0.0	1.0	1.0	1.0
Odor	1059.0	0.432483	0.495655	0.0	0.0	0.0	1.0	1.0
Fat	1059.0	0.671388	0.469930	0.0	0.0	1.0	1.0	1.0
Turbidity	1059.0	0.491029	0.500156	0.0	0.0	0.0	1.0	1.0
Colour	1059.0	251.840415	4.307424	240.0	250.0	255.0	255.0	255.0

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
pH            0
Temprature    0
Taste         0
Odor          0
Fat           0
Turbidity     0
Colour        0
Grade         0
dtype: int64
```

- No missing values so we can proceed to the pre-processing part

EDA and Pre-Processing the Data

- Checking the types of grade of milk

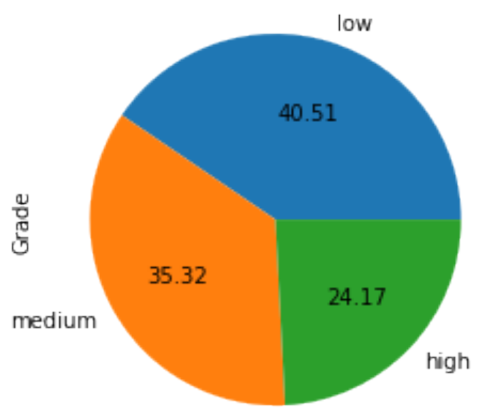
In [7]:
```
df['Grade'].value_counts()
```

Out[7]:
```
low       429
medium    374
high      256
Name: Grade, dtype: int64
```

- Visualising this data in a pie chart form for better understanding :

In [8]:
```
df['Grade'].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[8]:
```
<AxesSubplot:ylabel='Grade'>
```



- Low Quality milk is having the largest proportion around 40.51%
- While High Quality milk is having the smallest proportion around 24.17%

- Converting grade column into numerical, using mapping such as low - 1, medium - 2 and high - 3

In [9]:
```
df['Grade']=df['Grade'].map({'high':3,'medium':2,'low':1})
df.head()
```

Out[9]:

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

Splitting the Data into Features and Target

In [10]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

- Now we need to scale down the features column

In [11]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
sc = StandardScaler()
```

In [13]:
```
x = sc.fit_transform(x)
```

Splitting the Data into Training and Testing

In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,r
andom_state=0)
```

Model Selection

In [16]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
```

In [17]:
```
logreg=LogisticRegression()
knn=KNeighborsClassifier()
svm=SVC()
abc=AdaBoostClassifier()
```

In [18]:
```
models = [logreg, knn, svm, abc]
```

Model Training and Evaluation

In [19]:
```
from sklearn.metrics import accuracy_score, classification_rep
ort, confusion_matrix
```

In [20]:
```
accuracies=[]
for model in models:
    print('Model Name :',model,'\n')
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    accuracies.append(model.score(xtest,ytest).round(4)*100)
    print('Accuracy Score :',accuracy_score(ytest,ypred),'\n')
    print('Classification Report :','\n\n',classification_repo
rt(ytest,ypred))
    print('Confusion Matrix :', '\n\n', confusion_matrix(ytest
,ypred),'\n')
    print('Training Score :',model.score(xtrain,ytrain))
    print('Testing Score :',model.score(xtest,ytest),'\n')
    print(('-'*54),'\n')
```

```
Model Name : LogisticRegression()

Accuracy Score : 0.839622641509434

Classification Report :

              precision    recall  f1-score   support

           1       0.82      0.88      0.85        69
           2       0.91      0.82      0.86        77
           3       0.78      0.82      0.80        66

    accuracy                           0.84       212
   macro avg       0.84      0.84      0.84       212
weighted avg       0.84      0.84      0.84       212
```

```
Confusion Matrix :

 [[61  3  5]
 [ 4 63 10]
 [ 9  3 54]]

Training Score : 0.8571428571428571
Testing Score : 0.839622641509434

------------------------------------------------------

Model Name : KNeighborsClassifier()

Accuracy Score : 0.9858490566037735

Classification Report :

              precision    recall  f1-score   support

           1       0.99      0.97      0.98        69
           2       0.99      0.99      0.99        77
           3       0.99      1.00      0.99        66

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

Confusion Matrix :

 [[67  1  1]
 [ 1 76  0]
 [ 0  0 66]]

Training Score : 0.9976387249114522
Testing Score : 0.9858490566037735

------------------------------------------------------

Model Name : SVC()

Accuracy Score : 0.9292452830188679

Classification Report :

              precision    recall  f1-score   support

           1       1.00      0.94      0.97        69
           2       0.97      0.87      0.92        77
           3       0.83      0.98      0.90        66

    accuracy                           0.93       212
   macro avg       0.93      0.93      0.93       212
weighted avg       0.94      0.93      0.93       212
```

```
Confusion Matrix :

 [[65  1  3]
 [ 0 67 10]
 [ 0  1 65]]

Training Score : 0.9563164108618654
Testing Score : 0.9292452830188679

------------------------------------------------------

Model Name : AdaBoostClassifier()

Accuracy Score : 0.9056603773584906

Classification Report :

               precision    recall  f1-score   support

           1       1.00      1.00      1.00        69
           2       0.83      0.94      0.88        77
           3       0.91      0.77      0.84        66

    accuracy                           0.91       212
   macro avg       0.91      0.90      0.90       212
weighted avg       0.91      0.91      0.90       212

Confusion Matrix :

 [[69  0  0]
 [ 0 72  5]
 [ 0 15 51]]

Training Score : 0.9362455726092089
Testing Score : 0.9056603773584906

------------------------------------------------------
```

In [21]:
```
model_performance_accuracy=pd.DataFrame({'Model Name':['Logistic Regression','KNeighborsClassifier','SVC','AdaBoostClassifier'],'Accuracy (%)':accuracies})
model_performance_accuracy.sort_values(by='Accuracy (%)',ascending=False)
```

Out[21]:

	Model Name	Accuracy (%)
1	KNeighborsClassifier	98.58
2	SVC	92.92
3	AdaBoostClassifier	90.57
0	Logistic Regression	83.96

- From the above result of accuracies it is clear that KNeighborsClassifier (knn) model is the best fit model with accuracy of 98%

- Hyper Tuning KNeighborsClassifier (knn) using n_neighbors parameter

In [28]:
```
for i in range(1,31):
    knn=KNeighborsClassifier(n_neighbors=i)
    print('k :',i,'\n')
    knn.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    print('Accuracy Score :',accuracy_score(ytest,ypred),'\n')
    print('Training Score :',knn.score(xtrain,ytrain))
    print('Testing Score :',knn.score(xtest,ytest),'\n')
```

```
k : 1

Accuracy Score : 0.9056603773584906

Training Score : 1.0
Testing Score : 0.9952830188679245

k : 2

Accuracy Score : 0.9056603773584906

Training Score : 0.9988193624557261
Testing Score : 0.9905660377358491

k : 3

Accuracy Score : 0.9056603773584906
```

```
Training Score : 0.9976387249114522
Testing Score : 0.9858490566037735

k : 4

Accuracy Score : 0.9056603773584906

Training Score : 0.9976387249114522
Testing Score : 0.9858490566037735

k : 5

Accuracy Score : 0.9056603773584906

Training Score : 0.9976387249114522
Testing Score : 0.9858490566037735

k : 6

Accuracy Score : 0.9056603773584906

Training Score : 0.9940968122786304
Testing Score : 0.9669811320754716

k : 7

Accuracy Score : 0.9056603773584906

Training Score : 0.9905548996458088
Testing Score : 0.9622641509433962

k : 8

Accuracy Score : 0.9056603773584906

Training Score : 0.987012987012987
Testing Score : 0.9528301886792453

k : 9

Accuracy Score : 0.9056603773584906

Training Score : 0.987012987012987
Testing Score : 0.9528301886792453

k : 10

Accuracy Score : 0.9056603773584906

Training Score : 0.987012987012987
Testing Score : 0.9528301886792453
```

```
k : 11

Accuracy Score : 0.9056603773584906

Training Score : 0.9811097992916175
Testing Score : 0.9433962264150944

k : 12

Accuracy Score : 0.9056603773584906

Training Score : 0.9811097992916175
Testing Score : 0.9433962264150944

k : 13

Accuracy Score : 0.9056603773584906

Training Score : 0.974025974025974
Testing Score : 0.9386792452830188

k : 14

Accuracy Score : 0.9056603773584906

Training Score : 0.974025974025974
Testing Score : 0.9386792452830188

k : 15

Accuracy Score : 0.9056603773584906

Training Score : 0.974025974025974
Testing Score : 0.9386792452830188

k : 16

Accuracy Score : 0.9056603773584906

Training Score : 0.9645808736717828
Testing Score : 0.9292452830188679

k : 17

Accuracy Score : 0.9056603773584906

Training Score : 0.9456906729634003
Testing Score : 0.9245283018867925

k : 18

Accuracy Score : 0.9056603773584906
```

```
Training Score : 0.9338842975206612
Testing Score : 0.910377358490566

k : 19

Accuracy Score : 0.9056603773584906

Training Score : 0.9338842975206612
Testing Score : 0.910377358490566

k : 20

Accuracy Score : 0.9056603773584906

Training Score : 0.9338842975206612
Testing Score : 0.910377358490566

k : 21

Accuracy Score : 0.9056603773584906

Training Score : 0.9338842975206612
Testing Score : 0.910377358490566

k : 22

Accuracy Score : 0.9056603773584906

Training Score : 0.9338842975206612
Testing Score : 0.910377358490566

k : 23

Accuracy Score : 0.9056603773584906

Training Score : 0.9315230224321134
Testing Score : 0.9198113207547169

k : 24

Accuracy Score : 0.9056603773584906

Training Score : 0.9315230224321134
Testing Score : 0.9198113207547169

k : 25

Accuracy Score : 0.9056603773584906

Training Score : 0.9315230224321134
Testing Score : 0.9198113207547169

k : 26
```

```
Accuracy Score : 0.9056603773584906

Training Score : 0.9315230224321134
Testing Score : 0.9198113207547169

k : 27

Accuracy Score : 0.9056603773584906

Training Score : 0.9327036599763873
Testing Score : 0.9245283018867925

k : 28

Accuracy Score : 0.9056603773584906

Training Score : 0.9362455726092089
Testing Score : 0.9292452830188679

k : 29

Accuracy Score : 0.9056603773584906

Training Score : 0.9161747343565525
Testing Score : 0.9009433962264151

k : 30

Accuracy Score : 0.9056603773584906

Training Score : 0.8842975206611571
Testing Score : 0.8443396226415094
```

- Hyper Tuning Logistic Regression with solvers parameter

In [29]:
```
solvers=['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
```

In [32]:
```
for i in solvers:
    logreg=LogisticRegression(solver=i)
    print('solvers :',i,'\n')
    logreg.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    print('Accuracy Score :',accuracy_score(ytest,ypred),'\n')
    print('Training Score :',logreg.score(xtrain,ytrain))
    print('Testing Score :',logreg.score(xtest,ytest),'\n')
```

```
solvers : newton-cg

Accuracy Score : 0.9056603773584906

Training Score : 0.8571428571428571
Testing Score : 0.839622641509434

solvers : lbfgs

Accuracy Score : 0.9056603773584906

Training Score : 0.8571428571428571
Testing Score : 0.839622641509434

solvers : liblinear

Accuracy Score : 0.9056603773584906

Training Score : 0.8394332939787486
Testing Score : 0.7924528301886793

solvers : sag

Accuracy Score : 0.9056603773584906

Training Score : 0.8571428571428571
Testing Score : 0.839622641509434

solvers : saga

Accuracy Score : 0.9056603773584906

Training Score : 0.8571428571428571
Testing Score : 0.839622641509434
```

Prediction with KNeighborsClassifier (knn)

In [22]:
```python
grade=knn.predict([[6.6,45,1,1,1,1,240]])
print(grade)

if grade == 1:
    print('The milk quality is Low')
elif grade == 2:
    print('The milk quality is Medium')
elif grade == 3:
    print('The milk quality is High')
else:
    print('Unknown milk quality')
```

```
[1]
The milk quality is Low
```

- Lets re-assure the accuracy by comparing predicted grade with the actual grade using a random sample row from the dataset

In [23]:
```python
sample_row = x[1]

sample_row = np.array(sample_row).reshape(1, -1)

grade = knn.predict(sample_row)
print(grade)

if grade == 1:
    print('The milk quality is Low')
elif grade == 2:
    print('The milk quality is Medium')
elif grade == 3:
    print('The milk quality is High')
else:
    print('Unknown milk quality')
```

```
[3]
The milk quality is High
```

In [24]:
```python
df.iloc[1]
```

Out[24]:
```
pH              6.6
Temprature     36.0
Taste           0.0
Odor            1.0
Fat             0.0
Turbidity       1.0
Colour        253.0
Grade           3.0
Name: 1, dtype: float64
```

- The Model is working fine with 98% accuracy using KNeighborsClassifier model.