Milk Grading System Using IBM Watson

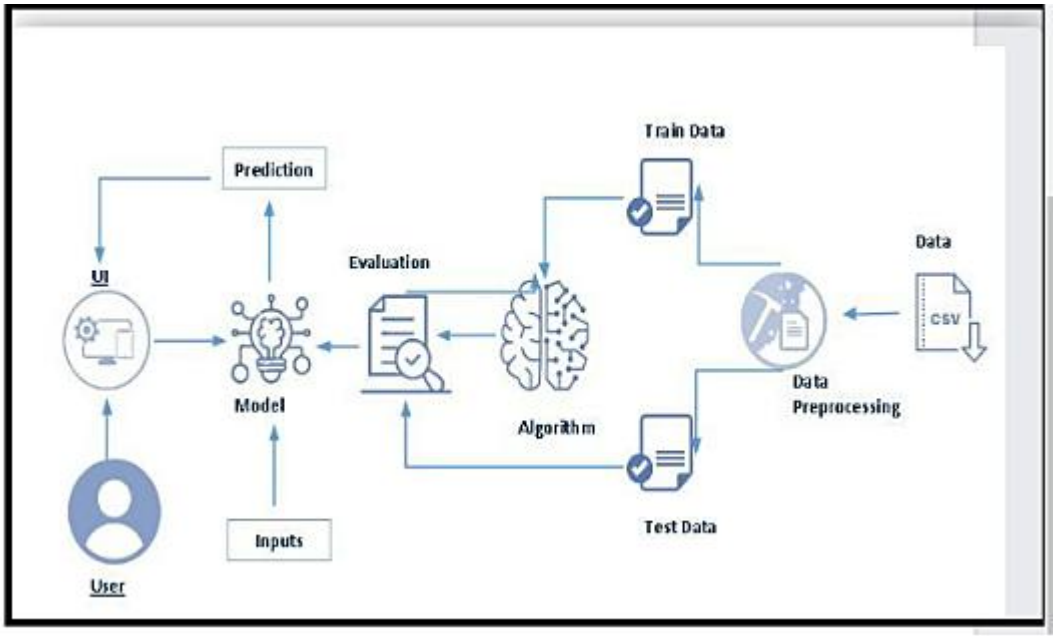
1. INTRODUCTION

1.1 Project Description:

The purpose of grading milk is to separate the available supply of potable milk into classes differing in superiority. Nearly all food products are graded in some way. so that the consumer may select milk for particular purposes according to his desires and pocketbook. Certified milk is practically the only stable grade, rules for production being laid down by the American Association of Medical Milk Commissions. There is a serious question as to whether or not it is possible in the present state of the industry to enforce uniform grades universally. The main problem here is not just the feature sets and target sets but also the approach that is taken in solving these types of problems. Also milk is part of our diet in many ways. With different varities and quality of milk in our environent, it is important to have a good grading system.

We are using classification algorithms such as Decision tree, Random forest, svm, and Extra tree classifier. We will train and test the data with these algorithms. From this the best model is selected and saved in pkl format.

1.2 Technical Architecture

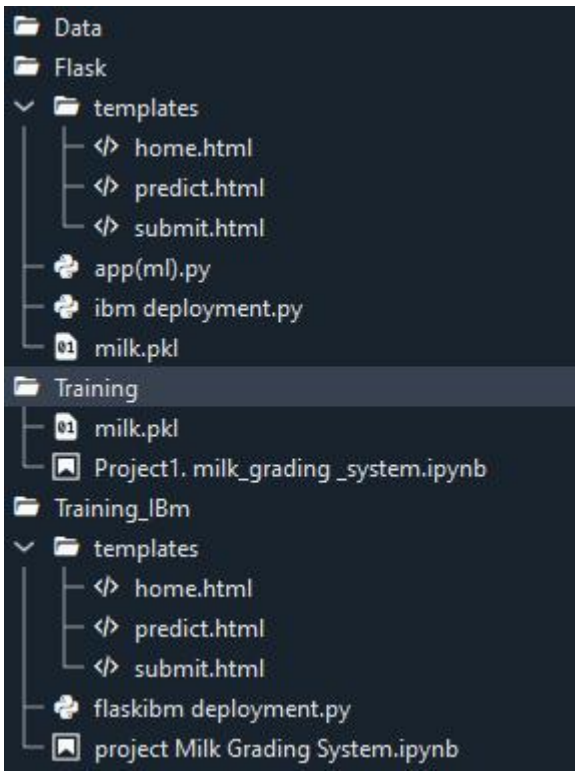


1.3 Project Objectives

- The main objectives behind this project is to;
- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.

1.4 Project Structure

Create the Project folder which contains files as shown below



2 . Project Flow

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI
- To accomplish this, we have to complete all the activities listed below

2.1 Data collection

- Collect the dataset or create the dataset

ML depends heavily on data, without data, it is impossible to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. This particular data set is taken from kaggle.com.

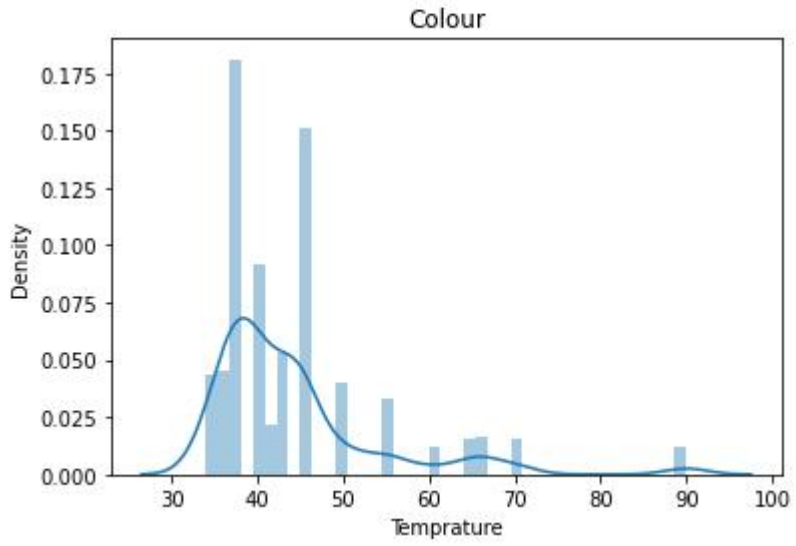
2.2 Visualising and analysing data

Univariate analysis

Univariate analysis is the technique of comparing and analyzing the dependency of a single predictor and a response variable. The prefix "uni" means one, emphasizing the fact that the analysis only accounts for one variable's effect on a dependent variable. Univariate Analysis is thought to be one of the simplest forms of data analysis as it doesn't deal with causes or relationships, like a regression would. Primarily, Univariate Analysis simply takes data and provides a summary and associated patterns.

```
sns.distplot(data['Temprature'])
plt.title('Colour')
plt.show()
```

```
E:\Users\anaconda3\lib\site-packages\seaborn\distribu
will be removed in a future version. Please adapt you
flexibility) or `histplot` (an axes-level function fo
  warnings.warn(msg, FutureWarning)
```



Bivariate analysis

This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship among the two variables.Thus bivariate data analysis involves comparisons, relationships, causes and explanations. These variables are often plotted on X and Y axis on the graph for better understanding of data and one of these variables is independent while the other is dependent.

Multivariate analysis

When the data involves **three or more variables**, it is categorized under multivariate. Example of this type of data is suppose an advertiser wants to compare the popularity of

four advertisements on a website, then their click rates could be measured for both men and women and relationships between variables can then be examined.

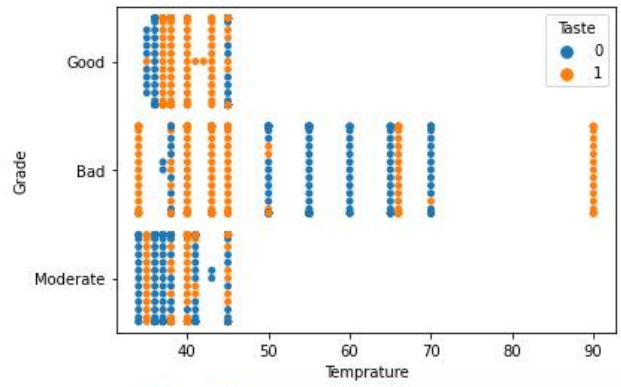
It is similar to bivariate but contains more than one dependent variable. The ways to perform analysis on this data depends on the goals to be achieved.Some of the techniques are regression analysis,path analysis,factor analysis and multivariate analysis of variance (MANOVA)

create swarmplot

```
import seaborn as sns
sns.swarmplot(x="Temprature", y="Grade", hue="Taste", data=data)
```

```
E:\Users\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 69.9% of the points
ay want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
E:\Users\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 68.8% of the points
ay want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
E:\Users\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 75.9% of the points
ay want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

```
<AxesSubplot:xlabel='Temprature', ylabel='Grade'>
```



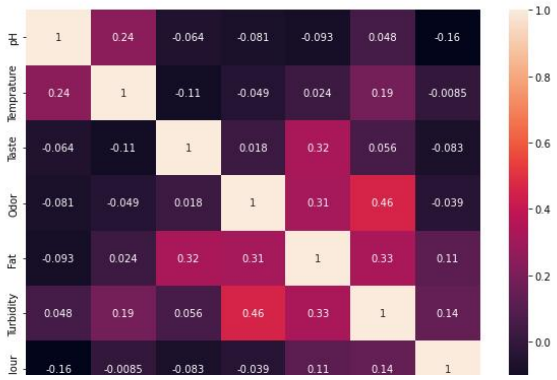
Heatmap

A Heat map is a graphical representation of multivariate data that is structured as a matrix of columns and rows.Heat maps are very useful in describing correlation among several numerical variables, visualizing patterns and anomalies.

Heatmap

```
In [13]: plt.figure(figsize=(10,7))
         sns.heatmap(data.corr(),annot=True)
```

```
Out[13]: <AxesSubplot:>
```



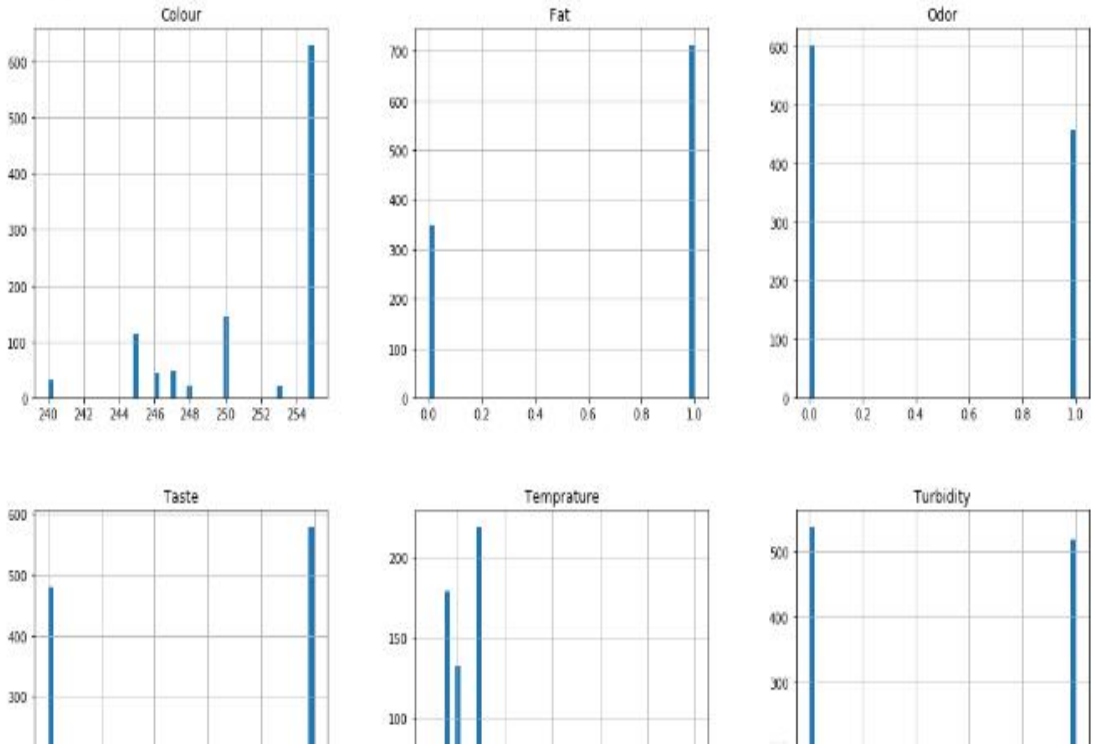
Histograms

Histograms group the data in bins and is the fastest way to get idea about the distribution of each attribute in dataset. The following are some of the characteristics of histograms −

- It provides us a count of the number of observations in each bin created for visualization.
- From the shape of the bin, we can easily observe the distribution i.e. weather it is Gaussian, skewed or exponential.
- Histograms also help us to see possible outliers.

```
In [15]: data.hist(bins=50, figsize=(18,18))

Out[15]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000027E341487F0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000027E344CB400>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000027E3427C980>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x0000027E342AEF60>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000027E342ED550>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000027E3431AB00>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x0000027E3435C0F0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000027E3438E6D8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000027E3438E710>]],
               dtype=object)
```

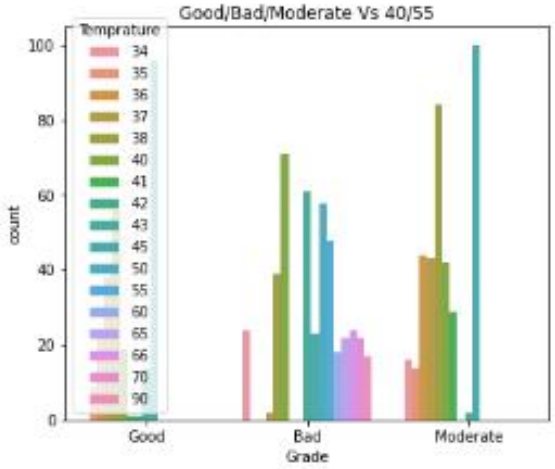


Count plot

countplot() method is used to Show the counts of observations in each categorical bin using bars.

```
In [16]: def countplot_of_2(x,hue,title=None,figsize=(6,5)):
             plt.figure(figsize=figsize)
             sns.countplot(data=data[[x,hue]],x=x,hue=hue)
             plt.title(title)
             plt.show()
```

```
In [17]: countplot_of_2('Grade','Temprature','Good/Bad/Moderate Vs 40/55')
```



2.3 Descriptive analysis

Descriptive Analysis in Machine Learning is all about perspective to understand the data and its different existing patterns. Basically, it is part of four types of Data Analysis concepts. Descriptive Analysis mainly tends more towards unsupervised learning for outlining, classifying, and drawing out information to get the answers for what had happened in past. It also helps people in the proper understanding of certain scenarios and its outcome.

3. Data pre-processing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model. Following are the syeps involved in data preprocessing.

- **Checking for null values**

Checking for null values

```
In [5]: data.isnull().sum()
```

```
Out[5]: pH            0
        Temprature    0
        Taste         0
        Odor          0
        Fat           0
        Turbidity     0
        Colour        0
        Grade         0
        dtype: int64
```

- **Importing the libraries**

Import the necessary libraries as shown in the image.

```
import numpy as nm
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from imblearn import over_sampling
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
import pickle
```

- **Read the Dataset**

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
data=pd.read_csv('Milk Grading (1).csv')
data.head()
```

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	1.0
1	6.6	36	0	1	0	1	253	1.0
2	8.5	70	1	1	1	1	246	0.0
3	9.5	34	1	1	0	1	255	0.0
4	6.6	37	0	0	0	0	255	0.5

- **Checking file size**

Checking file size

```
data.shape
```

```
(1059, 8)
```

- **Data info**

Data Analysis & Cleaning

```
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   float64
dtypes: float64(2), int64(6)
memory usage: 66.3 KB
```

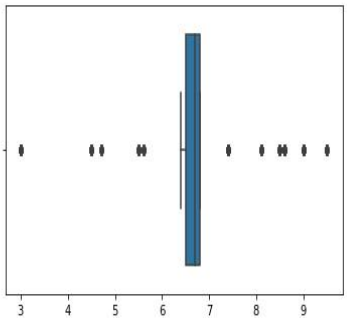
3.1 Handling outlier

Detecting the Outliers

```
In [25]: sns.boxplot(data['pH'])
```

```
E:\Users\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x.
From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword w
ill result in an error or misinterpretation.
  warnings.warn(
```

```
Out[25]: <AxesSubplot:xlabel='pH'>
```



3.2 Splitting data into train and test

Now let's split the Dataset into train and test sets
Changes: first split the dataset into x and y and then split the data set
Here x and y variables are created. On x variable, df is passed with dropping the target variable. And my target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

Train test split¶

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)
```

4. Model building

The major steps involved in model building are:

- Import the model building libraries
- Initialising the model
- Training and testing the model
- Evaluating performance of model
- Save the model

4.1 The SVC Model

A function named SVC is created and train and test data are passed as the parameters. Inside the function, the SupportVectorClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc= SVC()
svc.fit(x_train,y_train)
y_predict=svc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_predict)
test_accuracy
```

```
0.5283018867924528
```

```
y_train_predict=svc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict)
train_accuracy
```

```
0.5560802833530106
```

4.2 Random Forest Model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)

y_predict1=rfc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_predict1)
test_accuracy
```

```
0.9905660377358491
```

```
y_train_predict1=rfc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict1)
train_accuracy
```

```
1.0
```

4.3 Decision Tree Model

A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_train, y_train)

y_predict2=dtc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_predict2)
test_accuracy
```

```
0.9905660377358491
```

```
y_train_predict2=dtc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict2)
train_accuracy
```

```
1.0
```

4.4 Extra Tree Classifier Model

A function named ExtraTree is created and train and test data are passed as the parameters. Inside the function, the ExtraTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, confusion matrix and classification report is done

```
from sklearn.ensemble import ExtraTreesClassifier
etc=ExtraTreesClassifier()
etc.fit(x_train,y_train)

y_predict3=etc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_predict3)
test_accuracy
```

```
0.9905660377358491
```

```
y_train_predict3=etc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict3)
train_accuracy
```

```
1.0
```

5. Parameter Tuning

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

Hyper parameter tuning using GridSearchCV

Hyper parameter tuning using GridSearchCV for SVC

In [51]:
```
from sklearn.model_selection import GridSearchCV
```

In [52]:
```
parameters = {
              "kernel":['linear', 'rbf', 'sigmoid'],"gamma":['scale', 'auto'],
              "break_ties":['bool', 'default=False']
              }
```

In [53]:
```
from sklearn.model_selection import KFold
svc=SVC()
gdcv = GridSearchCV(estimator=svc,param_grid=parameters)
```

In [54]:
```
gdcv.fit(x_train,y_train)
```

Out[54]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'break_ties': ['bool', 'default=False'],
                         'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'rbf', 'sigmoid']})
```

In [55]:
```
gdcv.best_params_
```

Out[55]:
```
{'break_ties': 'bool', 'gamma': 'auto', 'kernel': 'rbf'}
```

In [56]:
```
from sklearn.metrics import accuracy_score
svc=SVC(kernel='rbf',gamma='auto',break_ties='bool')
svc.fit(x_train,y_train)
y_train_pred=svc.predict(x_train)
y_test_pred=svc.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred,y_train))
print("test accuracy",accuracy_score(y_test_pred,y_test))
```

```
train accuracy 0.961038961038961
test accuracy 0.9433962264150944
```

Hyper parameter tuning using GridSearchCV for RFC

In [57]:
```
parameters={"n_estimators" : [2,5,10,15,20,25],
            "warm_start":['False'],"min_samples_split":[2],"criterion":['entropy'],"random_state":[111]
    }
```

In [58]:
```
rfc=RandomForestClassifier()
gdcv1 = GridSearchCV(estimator=rfc,param_grid=parameters)
```

In [59]:
```
gdcv1.fit(x_train,y_train)
```

Out[59]:
```
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'criterion': ['entropy'], 'min_samples_split': [2],
                         'n_estimators': [2, 5, 10, 15, 20, 25],
                         'random_state': [111], 'warm_start': ['False']})
```

In [60]:
```
gdcv1.best_params_
```

Out[60]:
```
{'criterion': 'entropy',
 'min_samples_split': 2,
 'n_estimators': 5,
 'random_state': 111,
 'warm_start': 'False'}
```

In [61]:
```
from sklearn.metrics import accuracy_score
rfc=RandomForestClassifier(criterion='entropy',min_samples_split=2,n_estimators=5,warm_start='False',random_state=111)
rfc.fit(x_train,y_train)
y_train_pred=rfc.predict(x_train)
y_test_pred=rfc.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred,y_train))
print("test accuracy",accuracy_score(y_test_pred,y_test))
```

```
train accuracy 1.0
test accuracy 0.9905660377358491
```

Hyper parameter tuning using GridSearchCV for ETC

In [62]:
```
parameters={"n_estimators":[2,5,10,15,20,25],"criterion":['entropy'],
            "min_samples_split":[2],
             "min_samples_leaf":[1],"random_state":[111]}
```

In [63]:
```
etc=ExtraTreesClassifier()
gdcv2 = GridSearchCV(estimator=etc,param_grid=parameters)
```

In [64]:
```
gdcv2.fit(x_train,y_train)
```

Out[64]:
```
GridSearchCV(estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['entropy'], 'min_samples_leaf': [1],
                         'min_samples_split': [2],
                         'n_estimators': [2, 5, 10, 15, 20, 25],
                         'random_state': [111]})
```

In [65]:
```
gdcv2.best_params_
```

Out[65]:
```
{'criterion': 'entropy',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 5,
 'random_state': 111}
```

In [66]:
```
from sklearn.metrics import accuracy_score
etc=ExtraTreesClassifier(min_samples_leaf=1,min_samples_split=2,n_estimators=5,criterion='entropy',random_state= 111)
etc.fit(x_train,y_train)
y_train_pred=etc.predict(x_train)
y_test_pred=etc.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred,y_train))
print("test accuracy",accuracy_score(y_test_pred,y_test))
```

```
train accuracy 1.0
test accuracy 0.9905660377358491
```

Hyper parameter tuning using GridSearchCV for DTC

```
In [67]: parameters={"criterion":['entropy'],
             "splitter":['best'],
             "min_samples_split":[2],"random_state":[111]}

In [68]: dtc=DecisionTreeClassifier()
         gdcv3 = GridSearchCV(estimator=dtc,param_grid=parameters)

In [69]: gdcv3.fit(x_train,y_train)

Out[69]: GridSearchCV(estimator=DecisionTreeClassifier(),
                      param_grid={'criterion': ['entropy'], 'min_samples_split': [2],
                                  'random_state': [111], 'splitter': ['best']})

In [70]: gdcv3.best_params_

Out[70]: {'criterion': 'entropy',
          'min_samples_split': 2,
          'random_state': 111,
          'splitter': 'best'}

In [71]: from sklearn.metrics import accuracy_score
         dtc=DecisionTreeClassifier(splitter='best',min_samples_split=2,criterion='entropy', random_state=111)
         dtc.fit(x_train,y_train)
         y_train_pred=dtc.predict(x_train)
         y_test_pred=dtc.predict(x_test)
         print("train accuracy",accuracy_score(y_train_pred,y_train))
         print("test accuracy",accuracy_score(y_test_pred,y_test))

         train accuracy 1.0
         test accuracy 0.9905660377358491

In [72]: import pickle
         pickle.dump(svc,open('milk.pkl','wb'))

In [73]: pwd

Out[73]: 'C:\\Users\\user'
```

Comparison of Model

After calling the function, the results of models are displayed as output. From the four models, the svc1 is performing well. From the below image, We can see the accuracy of the model is 94% accuracy.From the above models used, the model that gives us the best training set value is choosen as the accepted value.The pickle file is then downloadedand run in the application.

```
In [71]: from sklearn.metrics import accuracy_score
         dtc=DecisionTreeClassifier(splitter='best',min_samples_split=2,criterion='entropy', random_state=111)
         dtc.fit(x_train,y_train)
         y_train_pred=dtc.predict(x_train)
         y_test_pred=dtc.predict(x_test)
         print("train accuracy",accuracy_score(y_train_pred,y_train))
         print("test accuracy",accuracy_score(y_test_pred,y_test))

         train accuracy 1.0
         test accuracy 0.9905660377358491

In [72]: import pickle
         pickle.dump(svc,open('milk.pkl','wb'))

In [73]: pwd

Out[73]: 'C:\\Users\\user'
```

6 Application Building]

6.1 Create an HTML file

For this project create three HTML files namely:

- home.html
- predict.html
- submit.html

These html files are saved in the **Templates** Folder

7. Build python code

the python code is:

app.py

```
from flask import Flask, render_template, request
import numpy as np
import pickle
import pandas as pd
model = pickle.load(open(r'D:\MILK_GRADING\Flask\milk.pkl','rb'))
app = Flask(__name__)
```

```
@app.route("/")
def about():
    return render_template('home.html')
@app.route("/predict")
def home1():
    return render_template('predict.html')
@app.route("/pred", methods=['POST','GET'])
def predict():
   x = [[x for x in request.form.values()]] #x is our input variables
   print(x)
   x = np.array(x)
   print(x.shape)



   print(x)
   pred = model.predict(x)
   print(pred[0])
   return render_template('submit.html', prediction_text=str(pred))
if __name__ == "__main__":
app.run(debug=False)
```

Run the application

Open anaconda prompt from the start menu

Navigate to the folder where your python script is.

Now type “python app.py” command

Navigate to the localhost where you can view your web page.

Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
In [2]: runfile('C:/Users/user/Desktop/connected to ml project/milk grading system files/
Flask/app(ml).py', wdir='C:/Users/user/Desktop/connected to ml project/milk grading system
files/Flask')
 * Serving Flask app "app(ml)" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

IBM Deployment

Training file:

Ibm_training.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 17 20:12:45 2022

@author: HP
"""

from flask import Flask, render_template, request
import numpy as np
import requests
import json

# NOTE: you must manually set API_KEY below using information retrieved from your IBM Cloud account.
API_KEY = "_uQJaQRra2V-2lHNbQY1-q314HMfQ3Rkw1ZhkFeVDtWN"
token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey":
 API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})
mltoken = token_response.json()["access_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}

# NOTE: manually define and pass the array(s) of values to be scored in the next line
```

```
#payload_scoring = {"input_data": [{"field": [["pH","Temprature", "Taste","Odor",
"Fat","Turbidity","Colour"]], "values": [[8.5,70,1,1,1,1,246]]}]}


#response_scoring = requests.post('https://us-
south.ml.cloud.ibm.com/ml/v4/deployments/f7293f6f-67be-4c43-9eb5-
917a520bac11/predictions?version=2022-06-07', json=payload_scoring,
 #headers={'Authorization': 'Bearer ' + mltoken})


app = Flask(__name__)
@app.route("/")
def about():
    return render_template('home.html')


@app.route("/predict")
def Predict():
    return render_template('predict.html')



@app.route("/pred", methods=['POST','GET'])
def predict():
    pH = request.form['pH']
    Temprature= request.form['Temprature']
    Taste = request.form['Taste']
    Odor= request.form['Odor']
    Fat = request.form['Fat']
    Turbidity = request.form['Turbidity']
    Colour = request.form['Colour']
    total = [[float(pH), int(Temprature), int(Taste), int(Odor),int(Fat),int(Turbidity), int(Colour)]]
    print(total)
    # NOTE: manually define and pass the array(s) of values to be scored in the next line
    payload_scoring = {"input_data": [{"field": [["pH","Temprature", "Taste","Odor",
"Fat","Turbidity","Colour"]], "values": [[8.5,70,1,1,1,1,246]]}]}


    response_scoring = requests.post('https://us-
south.ml.cloud.ibm.com/ml/v4/deployments/f7293f6f-67be-4c43-9eb5-
917a520bac11/predictions?version=2022-06-07', json=payload_scoring,
    headers={'Authorization': 'Bearer ' + mltoken})
    print("Scoring response")
```

```
    print(response_scoring.json())
    predictions=response_scoring.json()

    print(predictions)
    pred=predictions['predictions'][0]['values'][0][0]


    return render_template('submit.html', prediction_text=pred)

if __name__ == "__main__":
    app.run(debug=False)
```

SCOPE AND RELEVANCE OF THE PROJECT

The purpose of grading milk is to separate the available supply of potable milk into classes differing in superiority. Nearly all food products are graded in some way. Butter and cheese, for instance, are graded according to palatability and other factors which appeal to the consumer.

It appears equally logical to separate milk into similar classes so that the consumer may select milk for particular purposes according to his desires and pocketbook.

Many grading systems have been worked out and a number of them are now in operation. These systems have varied over a wide range; for instance, the bacterial requirements for Grade A milk range all the way from 10,000 to 200,000 per c.c. Certified milk is practically the only stable grade, rules for production being laid down by the American Association of Medical Milk Commissions. From the ethical standpoint this variation in requirements for similar grades is extremely undesirable, and the trend should be toward a similarity of requirements in the same grades. There enters into the problem, however, the practicability of the wliole matter, and there is a serious question as to whether or not it is possible in the present state of the industry to enforce uniform grades universally.

SYSTEM STUDY AND ANALYSIS

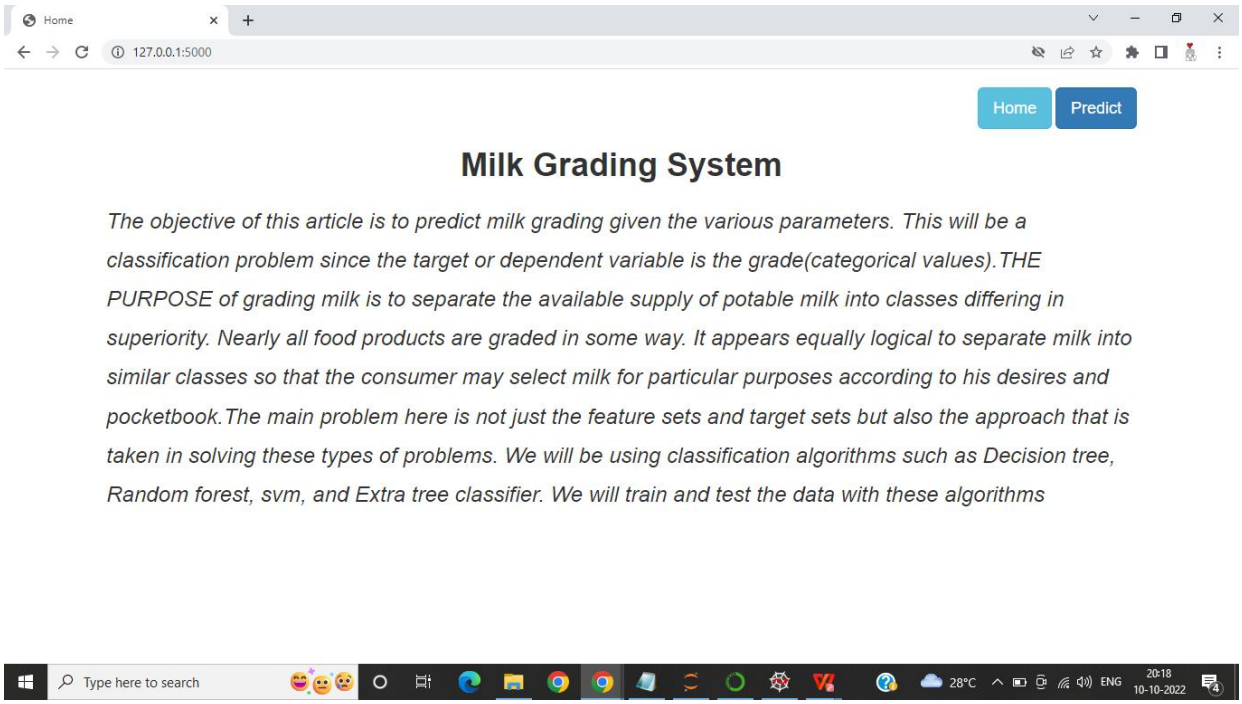
System Analysis is the process of gathering and interpreting facts, diagnosing the problems and using the information to recommend improvements. System study is a general term that refers to an orderly, structured process for identifying and solving problems. The first phase of software development is system study.

The importance of system study phase is the establishment of the requirements for the system to be acquired, developed and installed. Analysing the project to understand the complexity forms the vital part of the system study. Problematic areas are identified and information is collected. Fact finding or gathering is essential to any analysis of requirements. It is also highly essential that the analyst familiarize himself with the objectives, activities and functions of organizations in which the system is to be implemented.

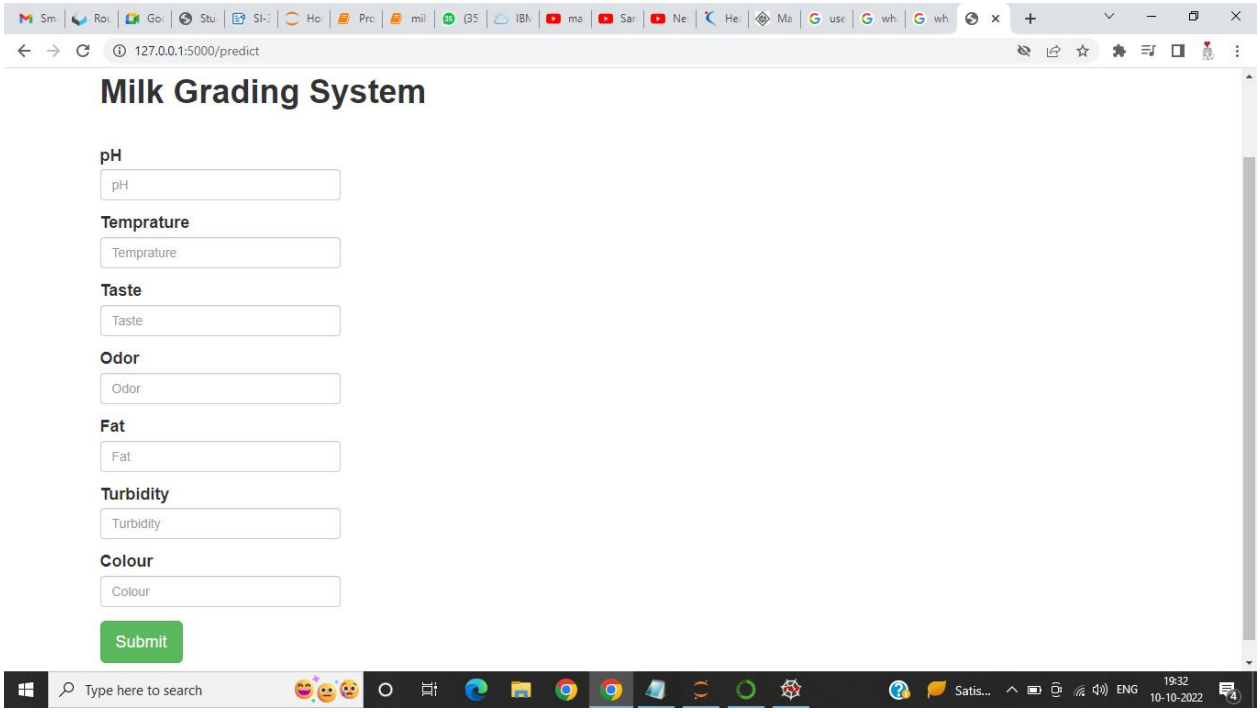
In system study, a detailed study of these operations performed by a system and their relationships within and outside the system is done. A key question considered here is, “What must be done to solve the problem?” One aspect of system study is defining the boundaries of the application and determining whether or not the candidate application should be considered.

OUTPUT SCREEN SHOTS

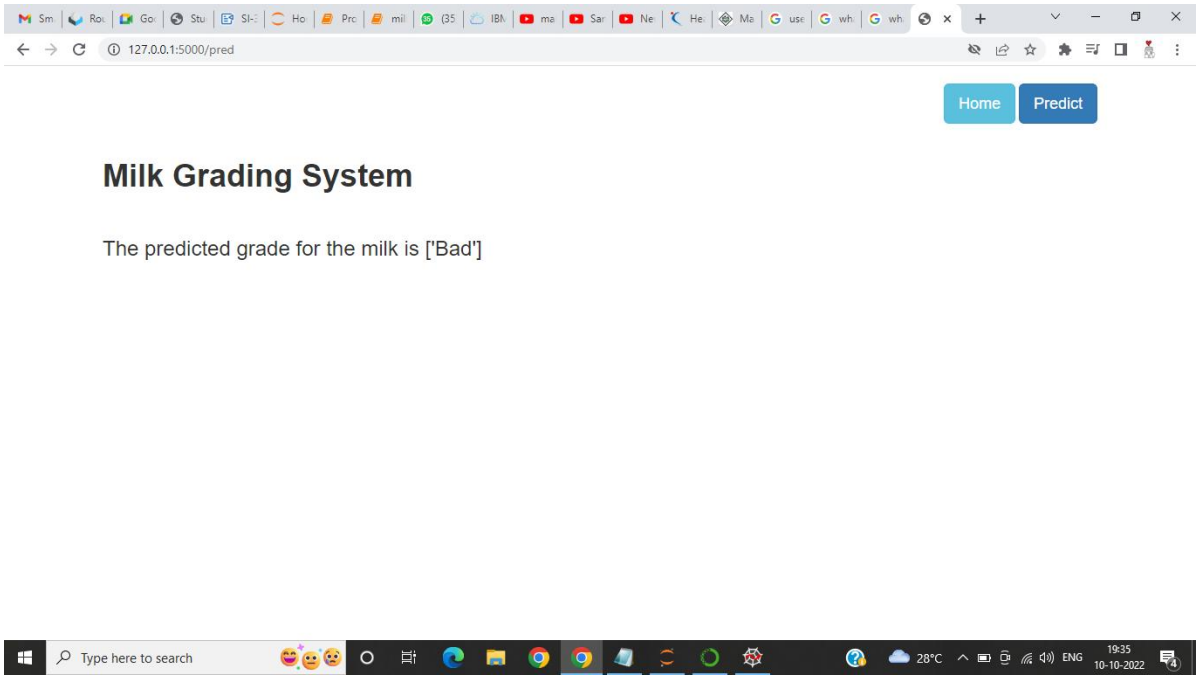
Home.html



Predict.html



Submit.html


127.0.0.1:5000/pred
Home
Predict
Milk Grading System
The predicted grade for the milk is ['Bad']
Type here to search
28°C
ENG
19:35
10-10-2022