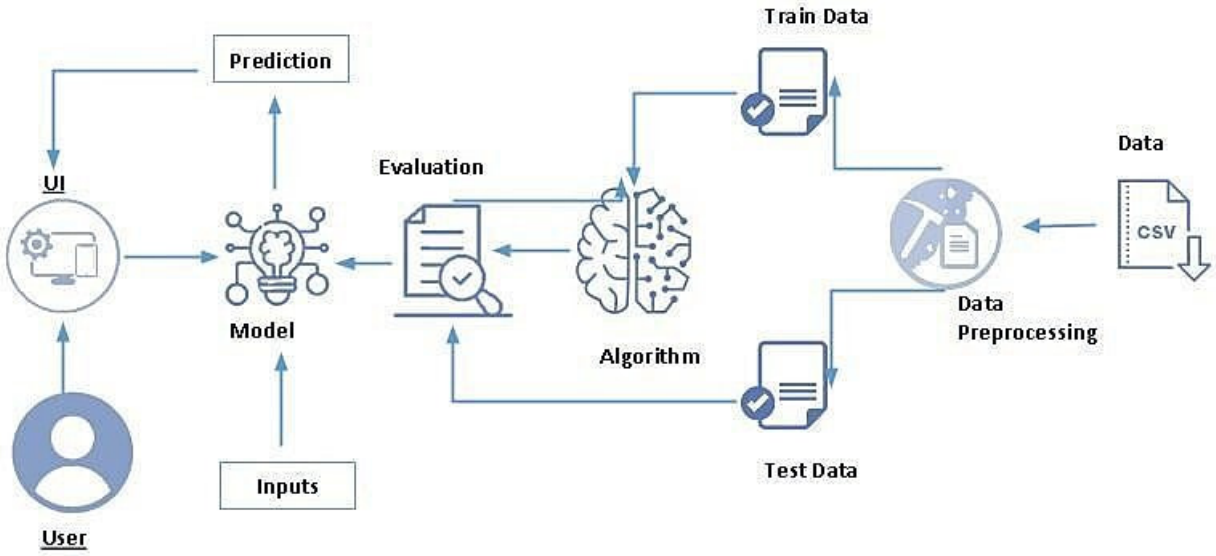
Milk Grading System Using Machine Learning

Project Description:

The purpose of grading milk is to separate the available supply of por table 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.

We will be 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.

Technical Architecture:



Project Objectives

By the end of this project you will:

- 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.

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

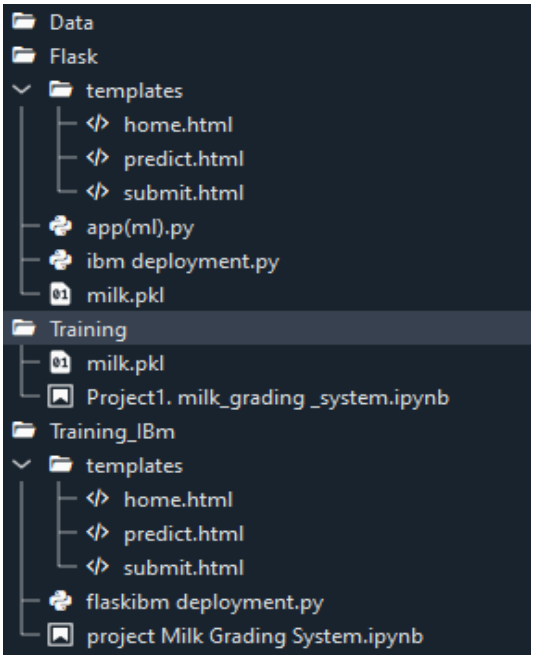
To accomplish this, we have to complete all the activities listed below,

- Data collection
    - Collect the dataset or create the dataset
- Visualising and analysing data
    - Univariate analysis
    - Bivariate analysis
    - Multivariate analysis
    - Descriptive analysis
- Data pre-processing
    - Checking for null values

    - Handling outlier
    - Handling categorical data
    - Splitting data into train and test
- Model building
    - Import the model building libraries
    - Initialising the model
    - Training and testing the model
    - Evaluating performance of model
    - Save the model
- Application Building
    - Create an HTML file
    - Build python code
    - Run the application

Project Structure

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Model.pkl is our saved model.

Further we will use this model for flask integration.

Pre Requisites

To complete this project, you must required following software's, concepts and packages

Anaconda Navigator

Refer the link below to download anaconda navigator

Link

Python Packages

Open anaconda prompt as administrator and install the required libraries by following the below instructions.

- Type "pip install numpy" and click enter.
- Type "pip install pandas" and click enter.
- Type "pip install scikit-learn" and click enter.
- Type "pip install matplotlib" and click enter.
- Type "pip install scipy" and click enter.
- Type "pip install pickle-mixin" and click enter.
- Type "pip install seaborn" and click enter.
- Type "pip install Flask" and click enter.

Prior Knowledge

You must have prior knowledge of following topics to complete this project

- ML Concepts

- Supervisedlearning
- Unsupervisedlearning
- Regression and classification
- Decisiontree
- Randomforest
- KNN
- Svm
- Extra tree classifier
- Evaluationmetrics

- Falsk Basics

Data Collection

- ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

- There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Dataset

In this project we have used Milk Grading (1).csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset

https://www.kaggle.com/datasets/prudhvignv/milk-grading

Data Preprocessing /Analysis

In this milestone, you need to complete all the below activities to build the model

1. 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
```

by use the info method to knowing the information about the dataset.

Visualization

- Univariate analysis
- Bivariate analysis
- Multivariate analysis
- Descriptive analysis

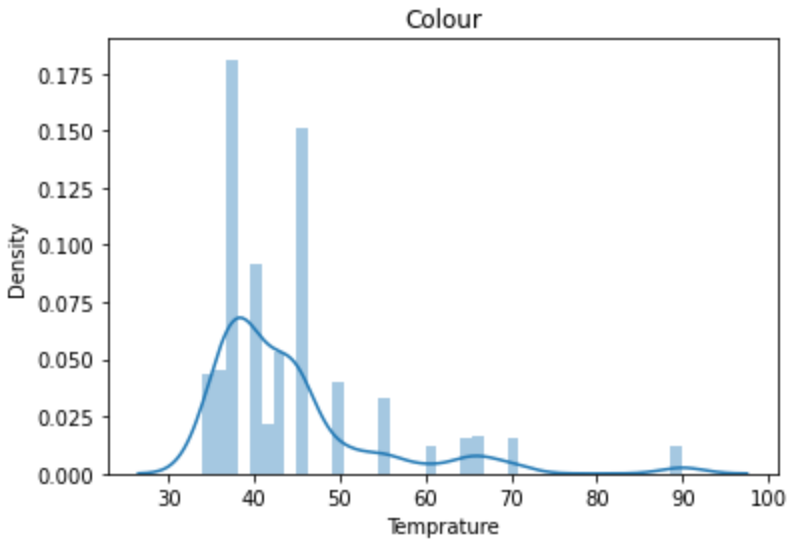
Univariate Analysis

In simple words, univariate analysis is understanding the data with a single feature. Here I have displayed the graph such as distplot .

The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature.

```
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)
```



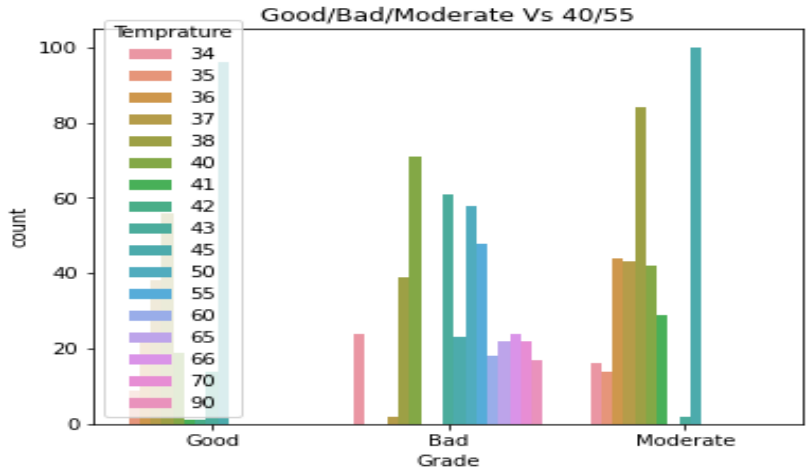
Bivariate Analysis

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between Grade and Temperature.

Countplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
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()
```

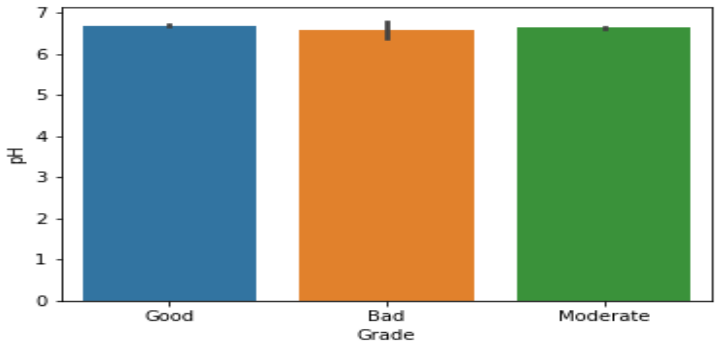
```
countplot_of_2('Grade','Temprature','Good/Bad/Moderate Vs 40/55')
```



barplot for pH and Grade

```
# create barplot Grade Vs pH
sns.barplot(x = 'Grade',
            y = 'pH',
            data = data)

# Show the plot
plt.show()
```

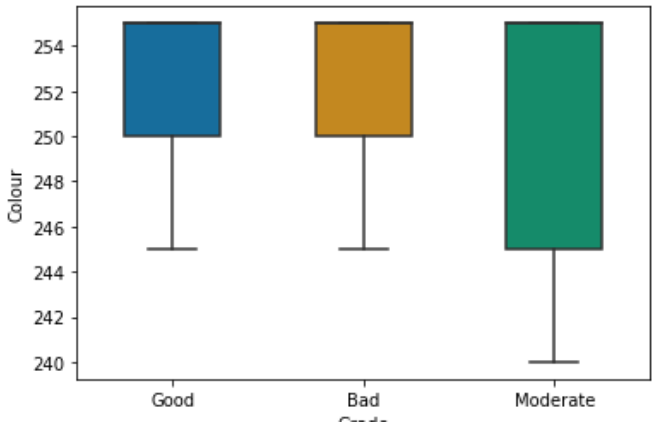


Here we are visualising the relationship between Grade and pH.

Barplot is used here. As a 1st parameter we are passing x(Grade) value and as a 2nd parameter we are passing y(pH) value.

boxplot for Colour and Grade

```
# create boxplot Colour Vs Grade
bplot = sns.boxplot(y='Colour', x='Grade',
                    data=data,
                    width=0.5,
                    palette="colorblind")
```



Here we are visualising the relationship between Grade and Colour.

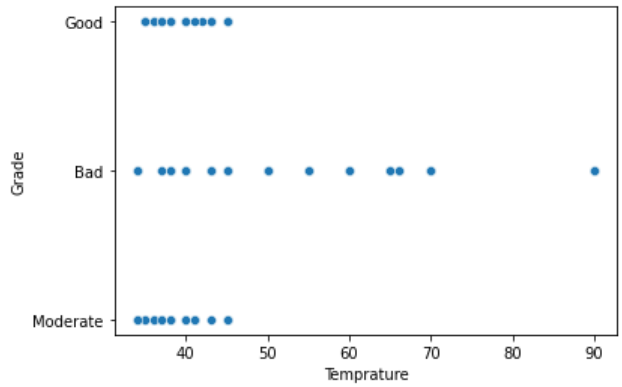
Boxplot is used here. As a 1st parameter we are passing x(Grade) value and as a 2nd parameter we are passing y(Colour)value.

scatterplot for Temprature and Grade

```
# create scatterplot Temprature Vs Grade

sns.scatterplot(data = data, x = "Temprature", y = "Grade")

plt.show()
```



In the sameway here we are visualising the relationship between Grade and Temperature.

Barplot is used here. As a 1st parameter we are passing x(Temperature) value and as a 2nd parameter we are passing y(Grade)value.

Multivariate Analysis

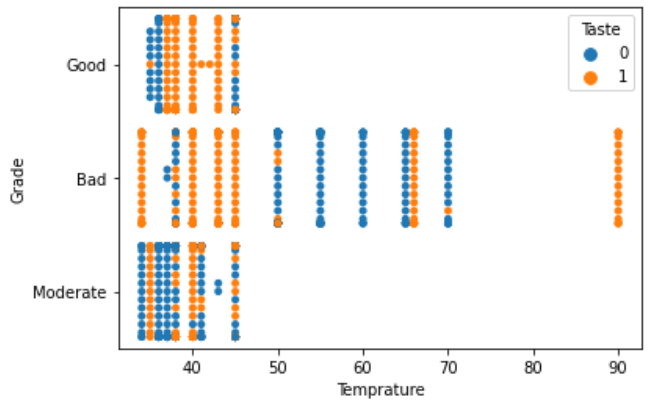
In simple words, multivariate analysis is to find the relation between multiple features. Here we have used swarmplot from the seaborn package.

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'>
```



Here we are visualising the relationship between Temperature ,Grade and Taste.
Swarmplot is used here. As a 1st parameter we are passing x(Temperature) value and as a 2nd parameter we are passing y(Grade)value and huge(Taste) values.

Descriptive Analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
data.describe()
```

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
count	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000	1059.000000
mean	6.630123	44.226629	0.546742	0.432483	0.671388	0.491029	251.840415	0.418319
std	1.399679	10.098364	0.498046	0.495655	0.469930	0.500156	4.307424	0.393934
min	3.000000	34.000000	0.000000	0.000000	0.000000	0.000000	240.000000	0.000000
25%	6.500000	38.000000	0.000000	0.000000	0.000000	0.000000	250.000000	0.000000
50%	6.700000	41.000000	1.000000	0.000000	1.000000	0.000000	255.000000	0.500000
75%	6.800000	45.000000	1.000000	1.000000	1.000000	1.000000	255.000000	0.500000
max	9.500000	90.000000	1.000000	1.000000	1.000000	1.000000	255.000000	1.000000

```
data['Grade'].value_counts()
```

```
0.0    429
0.5    374
1.0    256
Name: Grade, dtype: int64
```

Here using the data.value_counts() to know whether the data set is overfitted or under fitted. In my data it is overfitted

```
: data.loc[data['Grade']==0.0,'Grade'] = 'Bad'
  data.loc[data['Grade']==0.5,'Grade']='Moderate'
  data.loc[data['Grade']==1.0,'Grade']='Good'
```

Using data.loc method to convert the continuous values to categorical values.

Outlier Detection

Checking for null values

Let's find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

```
: 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
```

For checking the null values, data.isnull() function is used. To sum those null values we use the .sum() function to it. From the below image we found that there are no null values present in our dataset.So we can skip handling of missing values step.

```
data.isnull().sum()
```

```
pH            0
Temprature    0
Taste         0
Odor          0
Fat           0
Turbidity     0
Colour        0
Grade         0
dtype: int64
```

Let’s look for any outliers in the dataset

Activity 2: Handling outliers

With the help of boxplot, outliers are visualised. And here we are going to find the upper bound and lower bound of the pH feature with some mathematical formula.

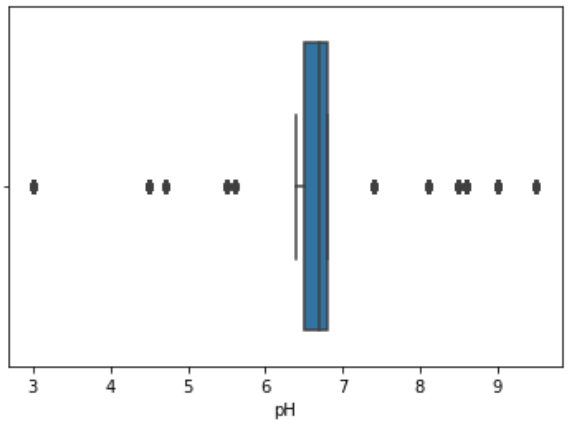
From the below diagram, we could visualise that pH feature has outliers. Boxplot from seaborn library is used here.

Detecting the Outliers

```
sns.boxplot(data['pH'])
```

```
E:\Users\anaconda3\lib\site-packages\seaborn\_decorators.
g: x. From version 0.12, the only valid positional argume
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='pH'>
```



From the above image we found that there are no outliers values present in our dataset. So we can skip handling of outliers values step.

Train Test Split

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)
```

Oversampling Technique

The above is using the Oversampling technique.why because my data is imbalanced that is the reason i'm importing the Oversampling.

Oversampling is a technique to balance uneven datasets by keeping all of the data in the majority class and increasing the size of the minority class. It is one of several techniques data scientists can use to extract more accurate information from originally imbalanced datasets.

over sampling

```
from imblearn import over_sampling
```

```
os = over_sampling.RandomOverSampler(random_state=0)
```

```
x,y = os.fit_resample(x,y)
```

```
y.value_counts()
```

```
Good        429
Bad         429
Moderate    429
Name: Grade, dtype: int64
```

Model Building

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

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
```

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
```

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
```

Extra Tree Classifier Model

A function named Extra Tree is created and train and test data are passed as the parameters. Inside the function, the Extra Tree Classifier 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
```

Parameter Tuning

Hyper parameters

Hyperparameters are the variables that the user specifies usually while building the Machine Learning model. thus, hyperparameters are specified before specifying the parameters or we can say that hyperparameters are used to evaluate optimal parameters of the model

--For example, max depth in Random Forest Algorithms, k in KNN Classifier.

Grid Search CV

1. Search process in sequential order
2. iterate through all combinations
3. computationally cost
4. chances of overfit

Grid Search CV

Grid Search CV

1. Search process in sequential order
2. iterate through all combinations
3. computationally cost
4. chances of overfit

Hyper parameter tuning using GridSearchCV

1.Hyper parameter tuning using GridSearchCV for SVC

```
from sklearn.model_selection import GridSearchCV
```

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

```
from sklearn.model_selection import KFold
svc1=SVC()
gdcv = GridSearchCV(estimator=svc1,param_grid=parameters)
```

```
gdcv.fit(x_train,y_train)
```

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

```
gdcv.best_params_
```

```
{'break_ties': 'bool', 'gamma': 'auto', 'kernel': 'rbf'}
```

```
from sklearn.metrics import accuracy_score
svc1=SVC(kernel='rbf',gamma='auto',break_ties='bool')
svc1.fit(x_train,y_train)
y_train_pred1=svc1.predict(x_train)
y_test_pred1=svc1.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred1,y_train))
print("test accuracy",accuracy_score(y_test_pred1,y_test))
```

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

2.Hyper parameter tuning using GridSearchCV for RFC

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

```
rfc2=RandomForestClassifier()
gdcv1 = GridSearchCV(estimator=rfc2,param_grid=parameters)
```

```
gdcv1.fit(x_train,y_train)
```

```
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']})
```

```
gdcv1.best_params_
```

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

```
from sklearn.metrics import accuracy_score
rfc2=RandomForestClassifier(criterion='entropy',min_samples_split=2,n_estimators=5,warm_start='False',random_state=111)
rfc2.fit(x_train,y_train)
y_train_pred2=rfc2.predict(x_train)
y_test_pred2=rfc2.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred2,y_train))
print("test accuracy",accuracy_score(y_test_pred2,y_test))
```

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

3.Hyper parameter tuning using GridSearchCV for ETC

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

```
etc3=ExtraTreesClassifier()
gdcv2 = GridSearchCV(estimator=etc3,param_grid=parameters)
```

```
gdcv2.fit(x_train,y_train)
```

```
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]})
```

```
gdcv2.best_params_
```

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

```
from sklearn.metrics import accuracy_score
etc3=ExtraTreesClassifier(min_samples_leaf=1,min_samples_split=2,n_estimators=5,criterion='entropy',random_state= 111)
etc3.fit(x_train,y_train)
y_train_pred3=etc3.predict(x_train)
y_test_pred3=etc3.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred3,y_train))
print("test accuracy",accuracy_score(y_test_pred3,y_test))
```

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

4.Hyper parameter tuning using GridSearchCV for DTC

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

```
dtc4=DecisionTreeClassifier()
gdcv3 = GridSearchCV(estimator=dtc4,param_grid=parameters)
```

```
gdcv3.fit(x_train,y_train)
```

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

```
gdcv3.best_params_
```

```
{'criterion': 'entropy',
 'min_samples_split': 2,
 'random_state': 111,
 'splitter': 'best'}
```

```
from sklearn.metrics import accuracy_score
dtc4=DecisionTreeClassifier(splitter='best',min_samples_split=2,criterion='entropy', random_state=111)
dtc4.fit(x_train,y_train)
y_train_pred4=dtc4.predict(x_train)
y_test_pred4=dtc4.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred4,y_train))
print("test accuracy",accuracy_score(y_test_pred4,y_test))
```

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

Now let’s see the performance of all the models and save the best model

Comparison Of Models

For comparing the above four models, the compare Model function is defined.

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.

```
def compareModel():
    print("train accuracy for svc1",accuracy_score(y_train_pred1,y_train))
    print("test accuracy for svc1",accuracy_score(y_test_pred1,y_test))
    print("train accuracy for rfc1",accuracy_score(y_train_pred2,y_train))
    print("test accuracy for rfc1",accuracy_score(y_test_pred2,y_test))
    print("train accuracy for dtc1",accuracy_score(y_train_pred4,y_train))
    print("test accuracy for dtc1",accuracy_score(y_test_pred4,y_test))
    print("train accuracy for etc1",accuracy_score(y_train_pred3,y_train))
    print("test accuracy for etc1",accuracy_score(y_test_pred3,y_test))
```

```
compareModel()
```

```
train accuracy for svc1 0.961038961038961
test accuracy for svc1 0.9433962264150944
train accuracy for rfc1 1.0
test accuracy for rfc1 0.9905660377358491
train accuracy for dtc1 1.0
test accuracy for dtc1 0.9905660377358491
train accuracy for etc1 1.0
test accuracy for etc1 0.9905660377358491
```

Evaluation Of The Model & Save The Model

From sklearn, accuracy score is used to evaluate the score of the model. On the parameters, we have given svc1 (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model is svc1 by pickle.dump()..

```
from sklearn.metrics import accuracy_score
svc1=SVC(kernel='rbf',gamma='auto',break_ties='bool')
svc1.fit(x_train,y_train)
y_train_pred1=svc1.predict(x_train)
y_test_pred1=svc1.predict(x_test)
print("train accuracy",accuracy_score(y_train_pred1,y_train))
print("test accuracy",accuracy_score(y_test_pred1,y_test))
```

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

```
import pickle
pickle.dump(svc1,open('milk.pkl','wb'))
```

Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
This section has the following tasks
Building HTML Pages
Building server side script

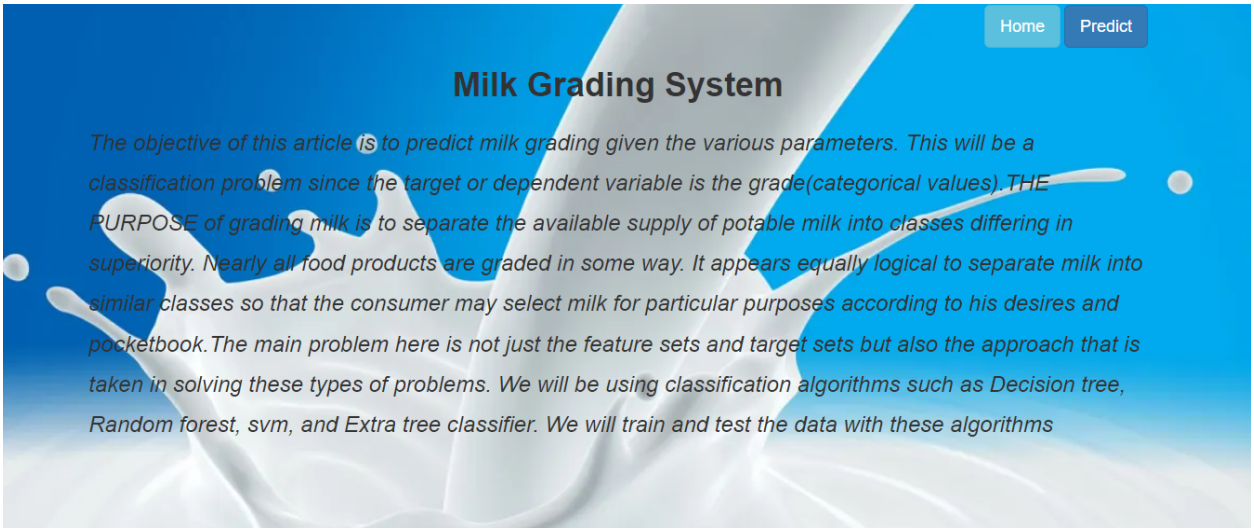
Building Html Pages

For this project create three HTML files namely

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

and save them in the templates folder.

Let’s see how our home.html page looks like:



Now when you click on predict button from top right corner you will get redirected to predict.html

Let's look how our predict.html file looks like:

Now when you click on submit button from left bottom corner you will get redirected to submit.html

Let's look how our submit.html file looks like:

Build Python Code

Import the libraries

```
from flask import Flask, render_template, request
import numpy as np
import pickle
import pandas as pd

model = pickle.load(open(r"C:/Users/user/milk.pkl",'rb'))
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
model = pickle.load(open(r"C:/Users/user/milk.pkl",'rb'))

app = Flask(__name__)
```

Render HTML page:

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

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@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()]]
   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))
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

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

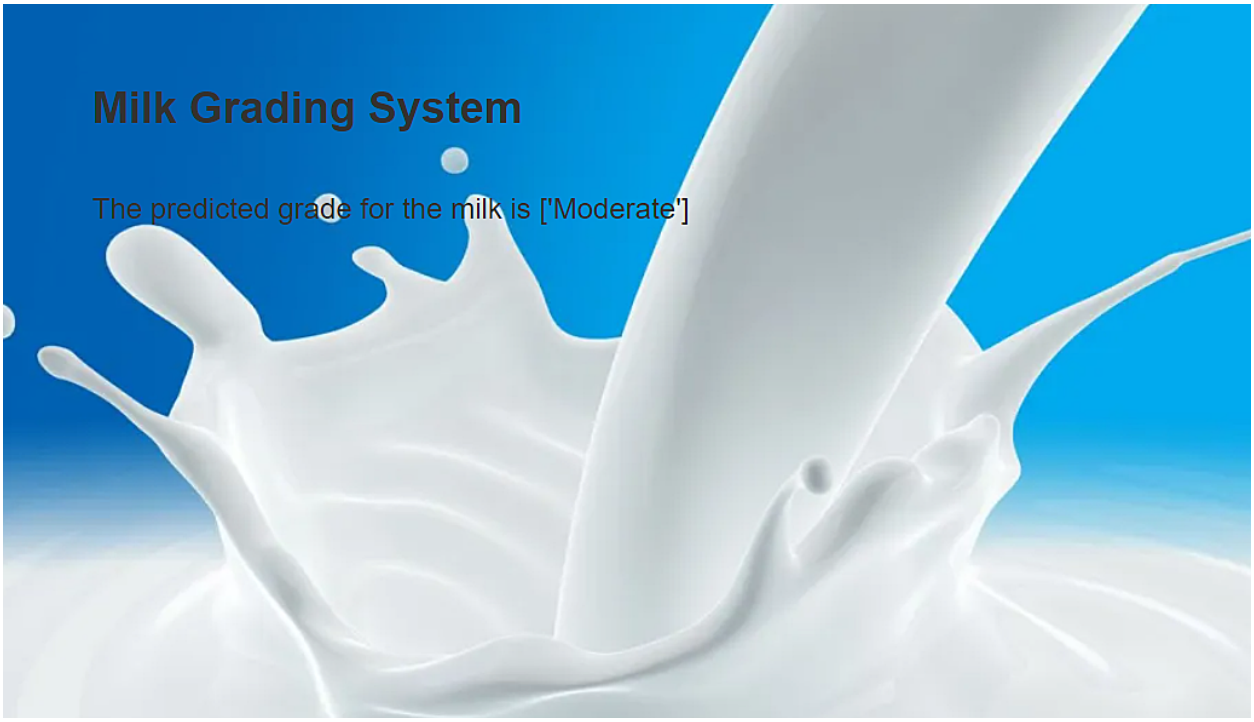
Run The Application

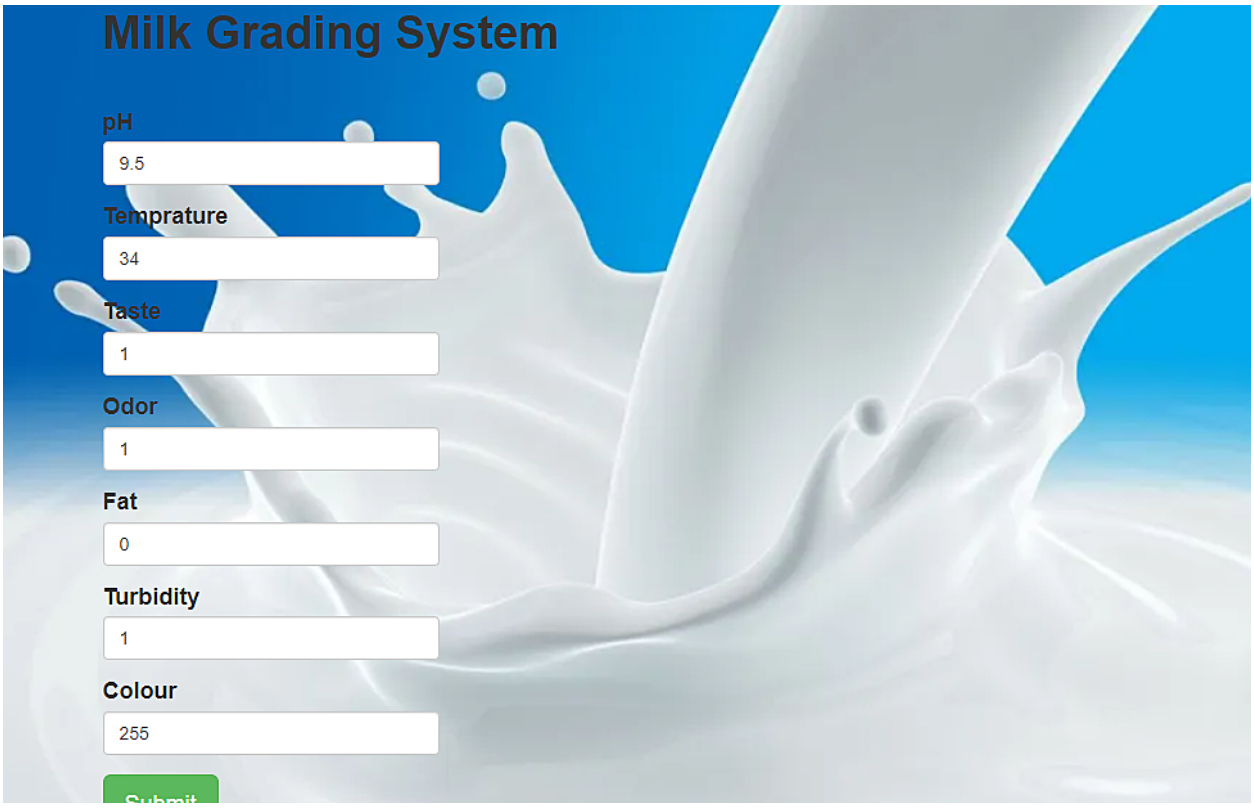
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and 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)
```

Output screenshots:


Milk Grading System
pH
6.6
Temprature
37
Taste
0
Odor
0
Fat
0
Turbidity
0
Colour
255
Submit


Milk Grading System
The predicted grade for the milk is ['Moderate']


Milk Grading System
pH
9.5
Temprature
34
Taste
1
Odor
1
Fat
0
Turbidity
1
Colour
255


Milk Grading System
The predicted grade for the milk is ['Bad']