

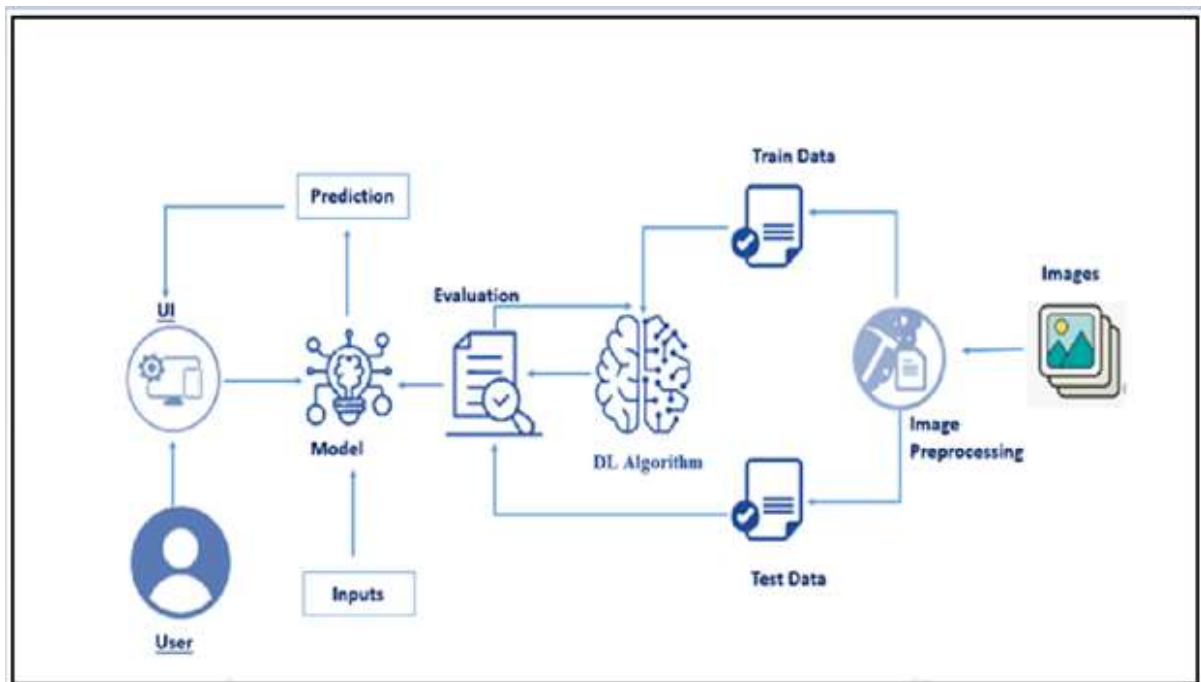
# Plant Seeding Classification By IBM Watson

## Project Description:

Agriculture is vital for human survival and remains a major driver of several economies around the world; more so in underdeveloped and developing economies. With increasing demand for food and cash crops, due to a growing global population and the challenges posed by climate change, there is a pressing need to increase farm outputs while incurring minimal costs. One major reason for reduction in crop yield is weed invasion on farmlands. Weeds generally have no useful value in terms of food, nutrition or medicine yet they have accelerated growth and parasitically compete with actual crops for nutrients and space. Inefficient processes such as hand weeding has led to significant losses and increasing costs due to manual labor. Plants continue to serve as a source of food and oxygen for all life on earth. In continents like Africa, where agriculture is predominant, proper automation of the farming process would help optimize crop yield and ensure continuous productivity and sustainability. In recent times, the state of agriculture and the amount of work people need to put in to check if plants/food is growing correctly is phenomenal, because it is 2019 and workers still need to organize and recognize the difference between different plants and weeds. People who are working in the agriculture field still have to have the ability to sort and recognize different plants and weeds, which does take a lot of time and a lot of effort in the long term.

This is where Artificial Intelligence can actually help benefit those workers, as the time and energy to identify plant seedlings will be much shortened. The ability to do so effectively can mean better crop yields and in the long term will result in better care for the environment. As by identifying the difference among the plants and weeds in a timely manner where it is highly accurate can positively impact agriculture.

## Technical Architecture:



## Pre requisites:

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

- **Anaconda navigator and pycharm:**
  - Refer the link below to download anaconda navigator
  - Link : <https://youtu.be/1ra4zH2G4o0>
- **Python packages:**
  - Open anaconda prompt as administrator
  - 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**
  - Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
  - Unsupervised learning: <https://www.javatpoint.com/unsupervised-machine-learning>

- Regression and classification
- Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
- Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
- KNN: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
- Xgboost: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
- Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
- **Flask Basics** : [https://www.youtube.com/watch?v=Ij4I\\_CvBnt0](https://www.youtube.com/watch?v=Ij4I_CvBnt0)

## Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- Know basics of Transfer Learning and build Xception model.
- Know how to build a web application using the Flask framework.

## Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- Xception Model analyzes the image, then prediction is showcased on the Flask UI.

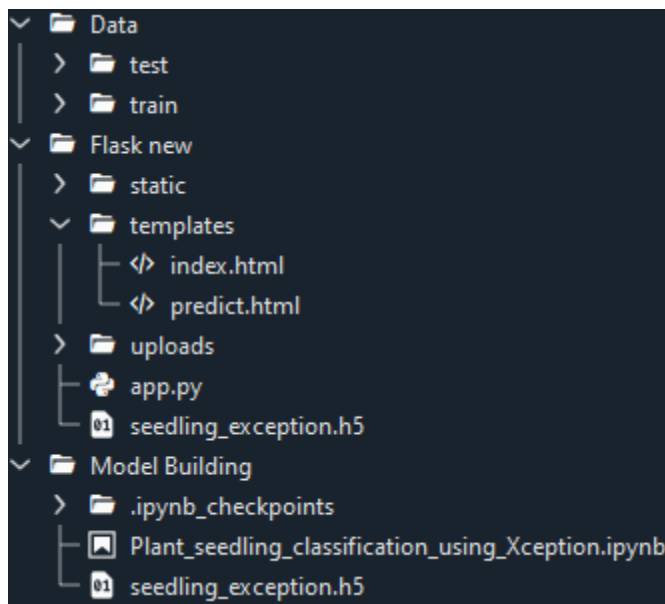
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
  - Create Train and Test Folders.
- Model Building
  - Importing the Model Building Libraries
  - Loading the model
  - Adding Flatten Layers
  - Adding Output Layer
  - Creating a Model object:

- Configure the Learning Process
  - Import the ImageDataGenerator library
  - Configure ImageDataGenerator class
  - Apply ImageDataGenerator functionality to Trainset and Testset
- Training
  - Train the Model
  - Save the Model
- Testing
  - Test the model
- Application Building
  - Create an HTML file
  - Build Python Code
  - Run the application
  - Final Output

## Project Structure:

- Create a Project folder which contains files as shown below



- The Dataset folder contains the training and testing images for training our model.

- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server side scripting.
- we need the model which is saved and the saved model in this content is a seedling\_exception.h5
- Templates folder contains about.html, index.html, predict.html pages.
- Model building folder that contains model building file (.ipynb file) and saved model.

## Milestone 1: Data Collection

You can download the dataset used in this project using the below link [Dataset](#)

Note: For better accuracy, train on more images, which can be downloaded from google and place it in respective folders.

### Activity 1: Download the dataset

Please download the dataset from the link. [dataset](#).

## Milestone 2: Model Building

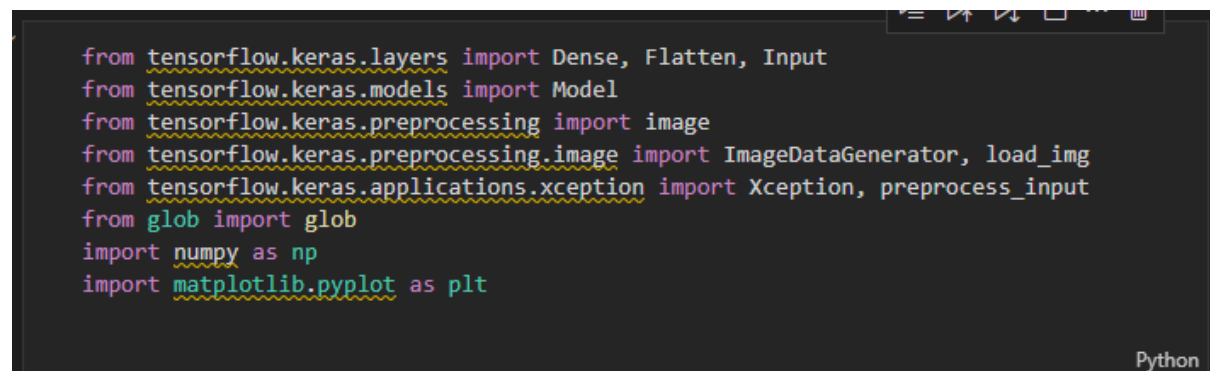
Now it's time to Build input and output layers for Xception model Hidden layers freeze because they have trained sequence, so changing the input and output layers.

### Activity 1: Importing Model Building Libraries

Importing the necessary libraries

#### : Importing The Model Building Libraries

Importing the necessary libraries



```
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.applications.xception import Xception, preprocess_input
from glob import glob
import numpy as np
import matplotlib.pyplot as plt
```

### Activity 2: 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 csv file.

```
# Reading the csv data
df = pd.read_csv(r'D:\TheSmartBridge\Project\dataset\heart.csv')
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC

### Activity 3: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

- Seaborn package provides a wonderful function `distplot`. With the help of `distplot`, we can find the distribution of the feature. To make multiple graphs in a single plot, we use `subplot`.



- In our dataset we have some categorical features. With the `countplot` function, we are

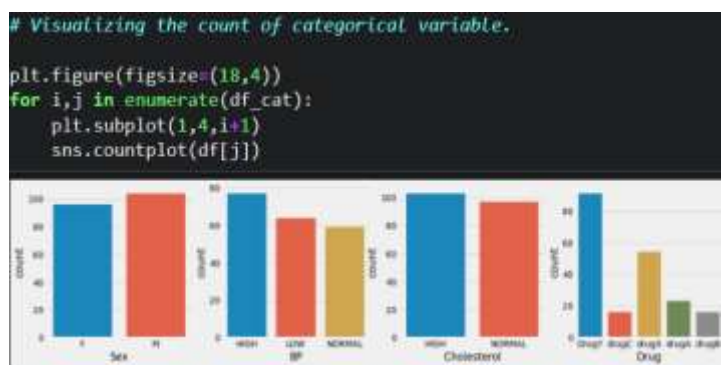
going to count the unique category in those features. We have created a dummy data frame with categorical features. With for loop and subplot we have plotted this below graph.

- From the plot we came to know, Most of the patients are using drugY and drugX. And most of the patients have high BP and high Cholesterol.

```
# Creating a data frame with categorical features
```

```
df_cat = df.select_dtypes(include='object')
df_cat.head()
```

	Sex	BP	Cholesterol	Drug
0	F	HIGH	HIGH	DrugY
1	M	LOW	HIGH	drugC
2	M	LOW	HIGH	drugC
3	F	NORMAL	HIGH	drugX
4	F	LOW	HIGH	DrugY



#### Activity 4: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between drug & BP, drug & sex and drug & cholesterol.

- Countplot is used here. As a 1<sup>st</sup> parameter we are passing x value and as a 2<sup>nd</sup> parameter we are passing hue value.
- From the below plot you can understand that drugA and drugB is not preferred to low and normal BP patients. DrugC is preferred only to low BP patients.
- By third graph we can understand, drugC is not preferred to normal cholesterol patients.



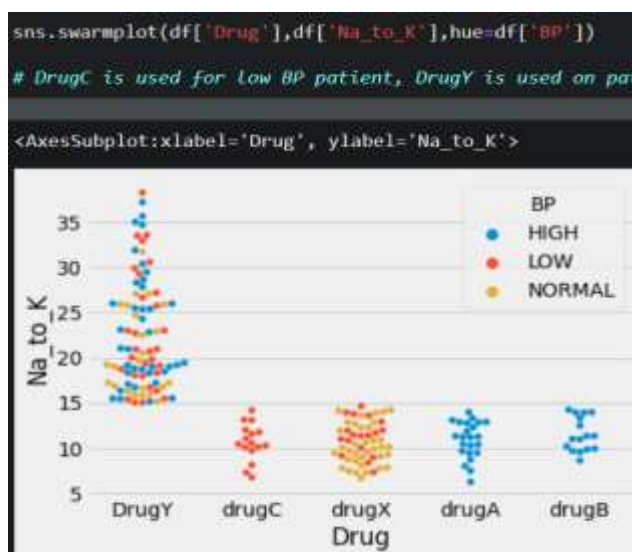
- With the help of age feature we are creating an age interval and finding the relation between drug feature and age interval feature. Function crosstab is used to find the relationship. From the below image we get a clear understanding, DrugB is preferred only for patients above age 50 years. And drugA is not preferred for patients above age 50 years.



## Activity 5: Multivariate analysis

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

- From the below image, we came to a conclusion that DrugY is used by most of patients who has different BP levels. But It is preferred only for patients having Na\_to\_K > 15 (Na\_to\_K – Sodium to potassium ratio on blood).





## Activity 6: 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.

```
df.describe(include='all')
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
count	200.000000	200	200	200	200.000000	200
unique	NaN	2	3	2	NaN	5
top	NaN	M	HIGH	HIGH	NaN	DrugY
freq	NaN	104	77	103	NaN	91
mean	44.315000	NaN	NaN	NaN	16.084485	NaN
std	16.544315	NaN	NaN	NaN	7.223956	NaN
min	15.000000	NaN	NaN	NaN	6.269000	NaN
25%	31.000000	NaN	NaN	NaN	10.445500	NaN
50%	45.000000	NaN	NaN	NaN	13.936500	NaN
75%	58.000000	NaN	NaN	NaN	19.380000	NaN
max	74.000000	NaN	NaN	NaN	38.247000	NaN

## Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

### Activity 1: Checking for null values

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

```
# Shape of csv data
df.shape

(200, 6)

# Checking the information of features
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  -
 0   Age             200 non-null   int64  
 1   Sex             200 non-null   object  
 2   BP              200 non-null   object  
 3   Cholesterol      200 non-null   object  
 4   Na_to_K         200 non-null   float64  
 5   Drug            200 non-null   object  
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

- For checking the null values, df.isnull() function is used. To sum those null values we use .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.

```
# Finding null values
df.isnull().sum()

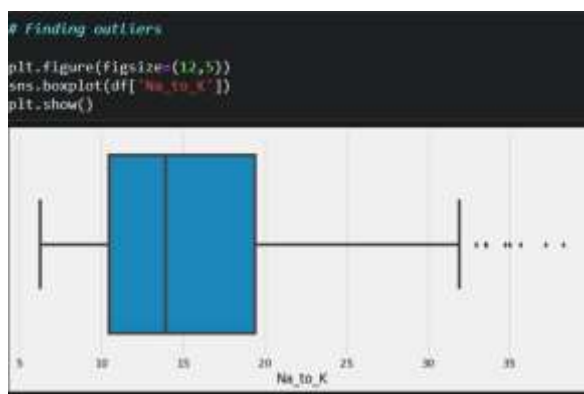
Age      0
Sex      0
BP       0
Cholesterol 0
Na_to_K  0
Drug     0
dtype: int64
```

Let's look for any outliers in the dataset

## Activity 2: Handling outliers

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of Na\_to\_K feature with some mathematical formula.

- From the below diagram, we could visualize that Na\_to\_K feature has outliers. Boxplot from seaborn library is used here.



- To find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3<sup>rd</sup> quantile. To find lower bound instead of adding, subtract it with 1<sup>st</sup> quantile. Take image attached below as your reference.

```
# Na_to_K has 8 outliers. In this project we are not going to remove them.

q1 = np.quantile(df['Na_to_K'],0.25)
q3 = np.quantile(df['Na_to_K'],0.75)

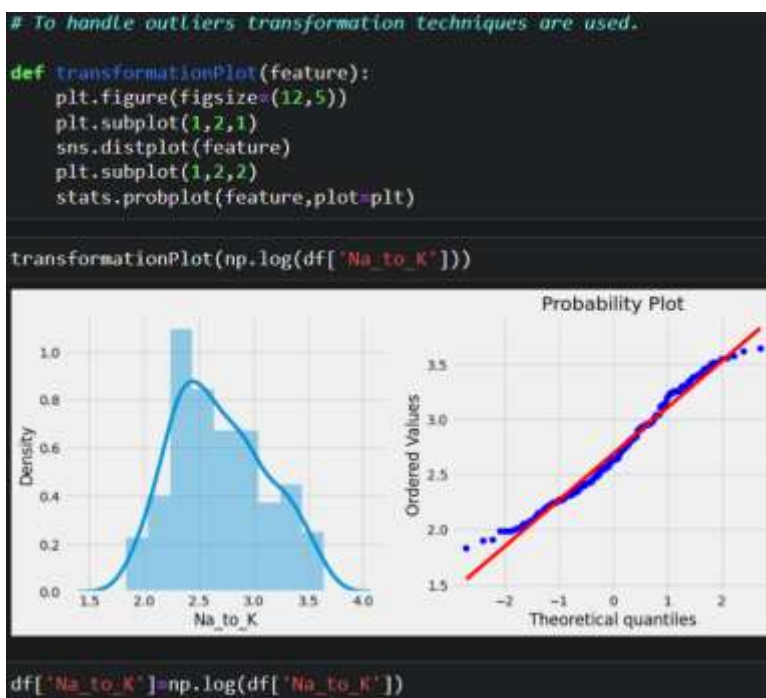
IQR = q3-q1

upper_bound = q3+(1.5*IQR)
lower_bound = q1-(1.5*IQR)

print('q1 : ',q1)
print('q3 : ',q3)
print('IQR : ',IQR)
print('Upper Bound : ',upper_bound)
print('Lower Bound : ',lower_bound)
print('Skewed data : ',len(df[df['Na_to_K']>upper_bound]))
print('Skewed data : ',len(df[df['Na_to_K']<lower_bound]))

q1 : 18.4455
q3 : 19.38
IQR : 8.9345
Upper Bound : 32.78175
Lower Bound : -2.9562500000000007
Skewed data : 8
Skewed data : 0
```

- To handle the outliers transformation technique is used. Here log transformation is used. We have created a function to visualize the distribution and probability plot of Na\_to\_K feature.



### Activity 3: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques.

There are several techniques but in our project we are using manual encoding with the help of list comprehension.

- In our project, categorical features are BP, Cholesterol and sex. With list comprehension encoding is done.

```
# Replacing low, normal & high with 0, 1 & 2...
df['BP'] = [0 if x=='LOW' else 1 if x=='NORMAL' else 2 for x in df['BP']]

# Replacing normal and high cholesterol with 0 & 1
df['Cholesterol'] = [0 if x=='NORMAL' else 1 for x in df['Cholesterol']]

# Replacing female and male with 0 & 1
df['Sex'] = [0 if x=='F' else 1 for x in df['Sex']]
```

#### Activity 4: 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 on y target variable is passed. For splitting training and testing data we are using train\_test\_split() function from sklearn. As parameters, we are passing x, y, test\_size, random\_state.

```
x = df.drop('Drug',axis=1)
y = df['Drug']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

print('Shape of x_train {}'.format(x_train.shape))
print('Shape of y_train {}'.format(y_train.shape))
print('Shape of x_test {}'.format(x_test.shape))
print('Shape of y_test {}'.format(y_test.shape))

Shape of x_train (140, 5)
Shape of y_train (140,)
Shape of x_test (60, 5)
Shape of y_test (60,)
```

## Milestone 4: 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.

### Activity 1: 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 initialized and training data is passed to the model with `.fit()` function. Test data is predicted with `.predict()` function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
def decisionTree(x_train, x_test, y_train, y_test):
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train)
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

### Activity 2: Random forest model

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

```
def randomForest(x_train, x_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train)
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

### Activity 3: KNN model

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

```
def KNN(x_train, x_test, y_train, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train)
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

#### Activity 4: Xgboost model

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

```
def xgboost(x_train, x_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train)
    yPred = xg.predict(x_test)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

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

#### Activity 5: Compare the model

For comparing the above four models compareModel function is defined.

```
def compareModel(x_train, x_test, y_train, y_test):
    decisionTree(x_train, x_test, y_train, y_test)
    print('-'*100)
    randomForest(x_train, x_test, y_train, y_test)
    print('-'*100)
    KNN(x_train, x_test, y_train, y_test)
    print('-'*100)
    xgboost(x_train, x_test, y_train, y_test)
```

After calling the function, the results of models are displayed as output. From the four model random forest and decision tree is performing well. From the below image, We can see the accuracy of the model. Both models have 97% accuracy. Even confusion matrix also have same results. Training time of decision tree is faster than random forest. In such case we have to select decision tree model (time saving & cost wise profitable). But, here random forest is



selected and evaluated with cross validation. Additionally, we can tune the model with hyper parameter tuning techniques.

```
compareModel(x_train, x_test, y_train, y_test)
```

```
***DecisionTreeClassifier***
Confusion matrix
[[25  0  0  0  0]
 [ 0  7  0  0  0]
 [ 0  2  4  0  0]
 [ 0  0  0  7  0]
 [ 0  0  0  0 15]]
Classification report
              precision    recall  f1-score   support

   DrugY       1.00        1.00        1.00         25
   drugA       0.78        1.00        0.88          7
   drugB       1.00        0.67        0.80          6
   drugC       1.00        1.00        1.00          7
   drugX       1.00        1.00        1.00         15

 accuracy      0.96
macro avg      0.96        0.93        0.93         60
weighted avg   0.97        0.97        0.97         60
```

```
***RandomForestClassifier***
Confusion matrix
[[25  0  0  0  0]
 [ 0  7  0  0  0]
 [ 0  2  4  0  0]
 [ 0  0  0  7  0]
 [ 0  0  0  0 15]]
Classification report
              precision    recall  f1-score   support

   DrugY       1.00        1.00        1.00         25
   drugA       0.78        1.00        0.88          7
   drugB       1.00        0.67        0.80          6
   drugC       1.00        1.00        1.00          7
   drugX       1.00        1.00        1.00         15

 accuracy      0.96
macro avg      0.96        0.93        0.93         60
weighted avg   0.97        0.97        0.97         60
```

```
***KNeighborsClassifier***
Confusion matrix
[[18  2  1  0  4]
 [ 6  0  0  0  1]
 [ 3  0  2  0  1]
 [ 5  0  0  0  2]
 [10  1  1  1  2]]
Classification report
              precision    recall  f1-score   support

   DrugY       0.43        0.72        0.54         25
   drugA       0.00        0.00        0.00          7
   drugB       0.50        0.33        0.40          6
   drugC       0.00        0.00        0.00          7
   drugX       0.20        0.13        0.16         15

 accuracy      0.23
macro avg      0.23        0.24        0.22         60
weighted avg   0.28        0.37        0.30         60
```

```
***GradientBoostingClassifier***
Confusion matrix
[[25  0  0  0  0]
 [ 0  7  0  0  0]
 [ 0  2  3  0  1]
 [ 0  0  0  6  1]
 [ 0  0  0  0 15]]
Classification report
              precision    recall  f1-score   support

   DrugY       1.00        1.00        1.00         25
   drugA       0.78        1.00        0.88          7
   drugB       1.00        0.50        0.67          6
   drugC       1.00        0.86        0.92          7
   drugX       0.88        1.00        0.94         15

 accuracy      0.93
macro avg      0.93        0.87        0.88         60
weighted avg   0.94        0.93        0.93         60
```

## Activity 6: Evaluating performance of the model and saving the model

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

Note: To understand cross validation, refer this link. <https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85>.

```

# Decision tree and Random forest performs well

from sklearn.model_selection import cross_val_score

# Random forest model is selected
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
yPred = rf.predict(x_test)

f1_score(yPred,y_test,average='weighted')
0.9679166666666668

cv = cross_val_score(rf,x,y,cv=5)

np.mean(cv)
0.985

pickle.dump(rf,open('model.pkl','wb'))

```

## Milestone 5: 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 serverside script

### Activity1: Building Html Pages:

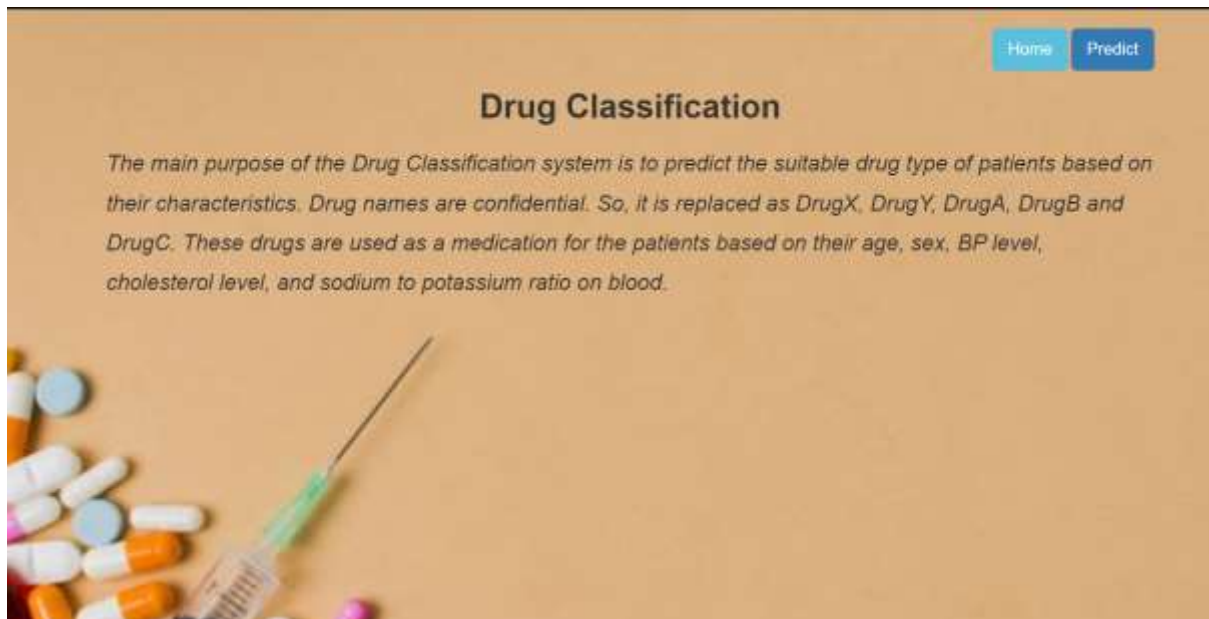
For this project create three HTML files namely

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

and save them in templates folder.

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





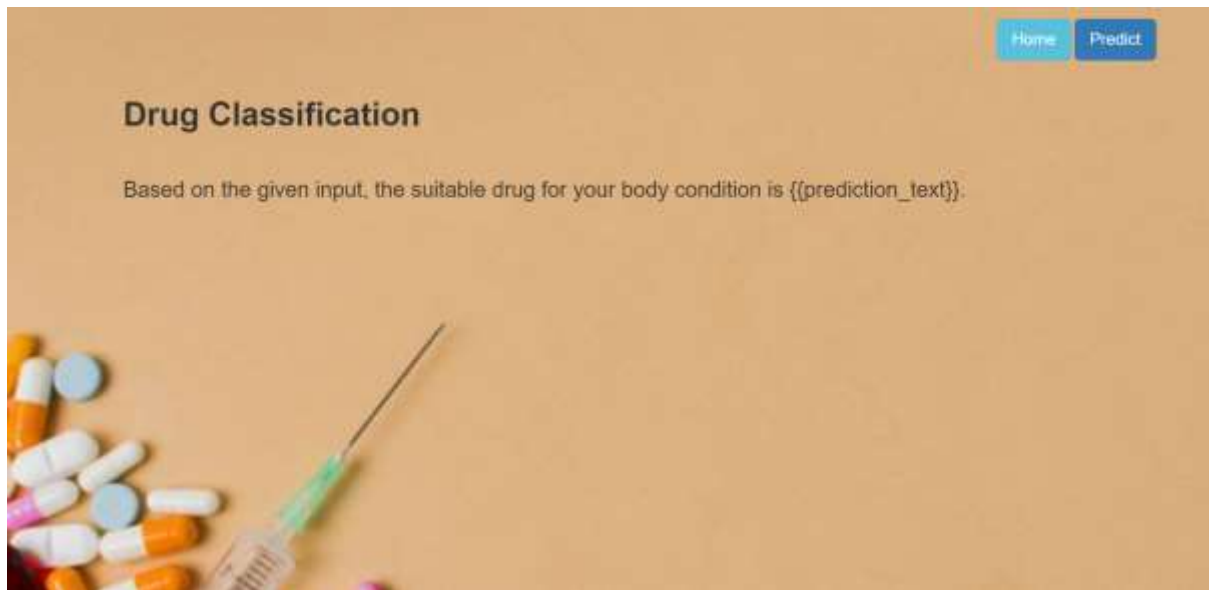
Now when you click on predict button from top right corner you will get redirected to predict.html

Lets look how our predict.html file looks like:

The image shows the "predict.html" page of the "Drug Classification" application. It features the same header with "Home" and "Predict" buttons. The main heading "Drug Classification" is followed by a form with the following fields: "Age" (text input), "Sex" (dropdown menu with "Male" selected), "BP" (dropdown menu with "Low" selected), "Cholesterol" (dropdown menu with "Normal" selected), and "Na\_to\_K" (text input). A green "Submit" button is located at the bottom left of the form. The background image of pills and a syringe is consistent with the home page.

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

Lets look how our submit.html file looks like:



## Activity 2: Build Python code:

Import the libraries

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

Load the saved model. Importing 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('model.pkl', 'rb'))
app = Flask(__name__)
```

Render HTML page:

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

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

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in 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("/pred", methods=['POST'])
def predict():
    age = request.form['Age']
    print(age)
    sex = request.form['Sex']
    if sex == '1':
        sex = 1
    if sex == '0':
        sex = 0
    bp = request.form['BP']
    if bp == '0':
        bp = 0
    if bp == '1':
        bp = 1
    if bp == '2':
        bp = 2
    cholesterol = request.form['Cholesterol']
    if cholesterol == '0':
        cholesterol = 0
    if cholesterol == '1':
        cholesterol = 1
    na_to_k = request.form['Na_to_K']
    total = [[int(age), int(sex), int(bp), int(cholesterol), float(na_to_k)]]
    print(total)
    prediction = model.predict(total)
    print(prediction)

    return render_template('submit.html', prediction_text=prediction)
```

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)
```

### Activity 3: 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.

```
base) D:\TheSmartBridge\Projects\2. DrugClassification\Drug c
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a p
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

