Disease Prediction Using Machine Learning

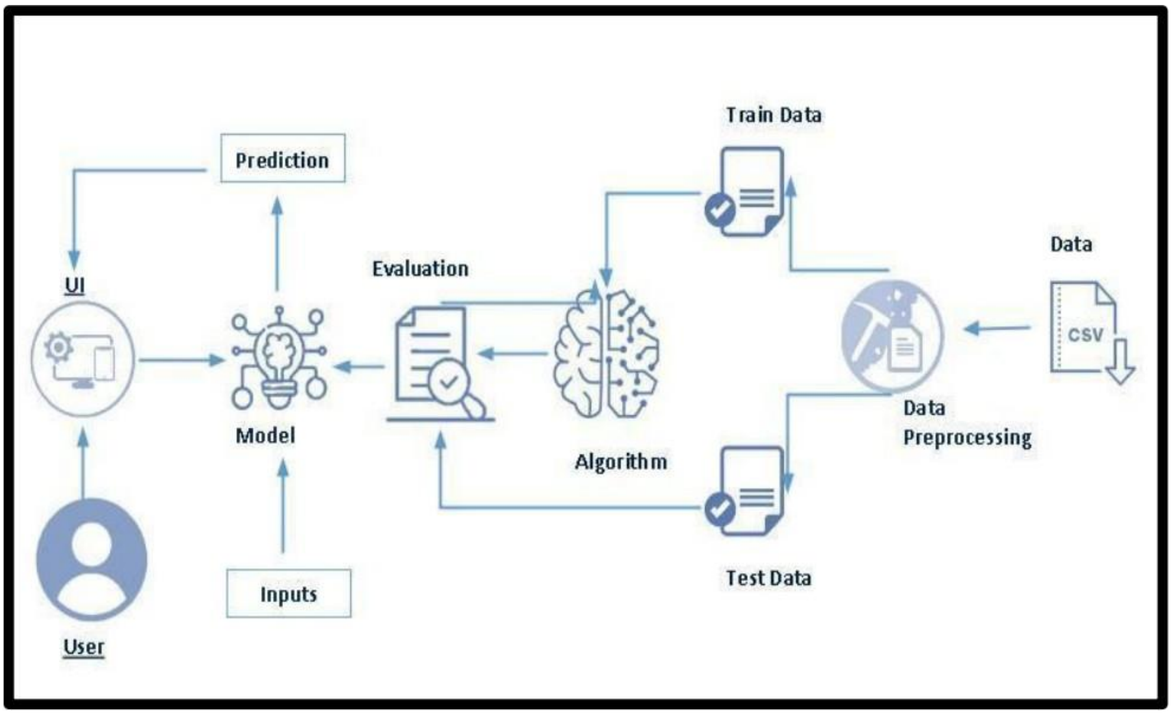
PROJECT MANUAL

In today’s face paced world, there is little to no time spared for healthcare. Even after developing severe symptoms to various diseases patients do not see a doctor. Using Google to type in our symptoms does not lead to good results, it always boils down to one stereotypical disease. So people have stopped using Google to look for their symptoms or the probable disease.

Catering to all the problems stated above, we have developed a model which can predict up to 42 diseases when given symptoms as input. This model can be used by doctors for consulting patients online based on the output of the model or this model can be used by patients for preventive diagnosis and self care as the charges to visit a doctor are high. This model does not ask for personalised data such as name, age, gender, religion, address, etc. You can use this web application anytime you feel you might be suffering from a disease and the model will give the probable disease based on the symptoms. Now you can take a call whether to visit a doctor or no.

The model is right 97 out of 100 times on an average. This model should be used for preventive diagnosis and early intervention by doctors and should not be taken as completely accurate and the final call.

Technical Architecture:



Project Flow:

- User is shown the Home page. The user will browse through Home page and click on the Predict button.
- After clicking the Predict button the user will be directed to the Details page where the user will input the symptoms they are having and click on the Predict button.
- User will be redirected to the Results page. The model will analyse the inputs given by the user and showcase the prediction of the most probable disease on the Results page.

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

- Define problem / Problem understanding
    - Specify the business problem
    - Business Requirements
    - Literature Survey
    - Social or Business Impact
- Data Collection and Preparation
    - Collect the dataset
    - Data Preparation
- Exploratory Data Analysis
    - Descriptive statistical
    - Visual Analysis
- Model Building
    - Creating a function for evaluation
    - Training and testing the Models using multiple algorithms
- Performance Testing & Hyperparameter Tuning
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning
    - Comparing model accuracy for different number of features.
    - Building model with appropriate features.
- Model Deployment
    - Save the best model
    - Integrate with Web Framework

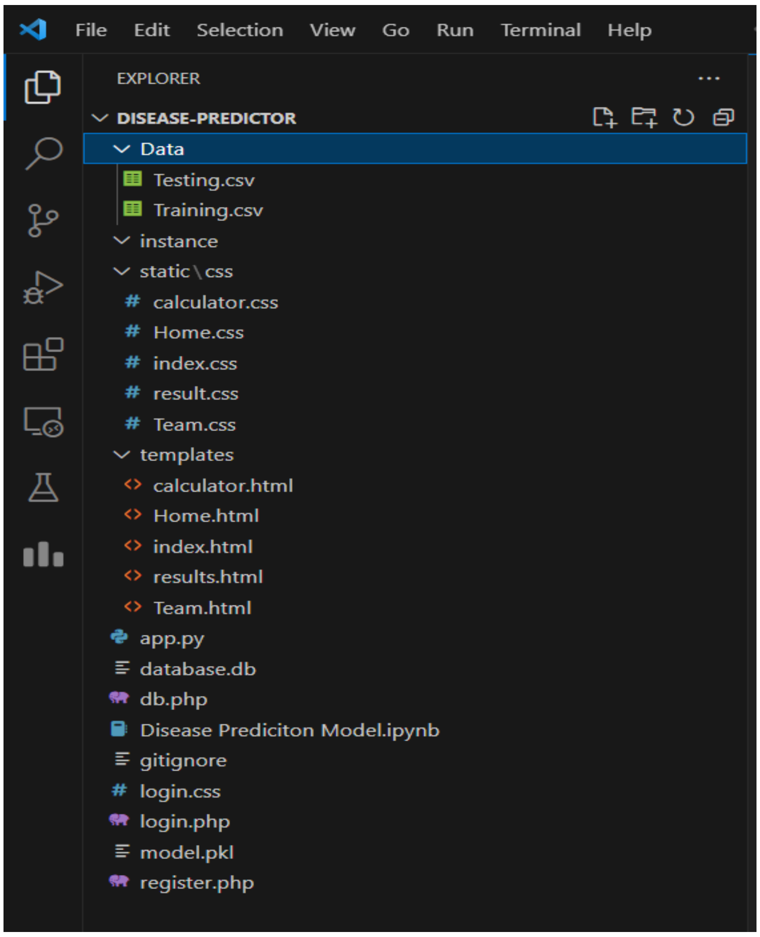
Prior Knowledge:

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

- ML Concepts:
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - K Nearest Neighbours: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - SVM: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classificationalgorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metrics/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create project folder which contains files as shown below:



```
DISEASE-PREDICTOR
  Data
    Testing.csv
    Training.csv
  instance
  static \ css
    calculator.css
    Home.css
    index.css
    result.css
    Team.css
  templates
    calculator.html
    Home.html
    index.html
    results.html
    Team.html
  app.py
  database.db
  db.php
  Disease Prediciton Model.ipynb
  gitignore
  login.css
  login.php
  model.pkl
  register.php
```

- The data obtained is in two csv files, one for training and another for testing.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- The css files should be stored in the static folder.
- App.py file is used for routing purposes using scripting.
- Model.pkl is the saved model. This will further be used in the Flask integration.
- Training folder contains a model training file.
- db.php, login.php and register.php handle all login and registering related funtions.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

Disease prediction involves identifying individuals who are at risk of developing a particular disease, based on various risk factors such as medical history and demographic factors. Predictive analytics and machine learning techniques can be used to analyse large amounts of data to identify patterns and risk factors associated with different diseases. Disease prediction using machine learning involves the use of various algorithms to analyse large datasets and identify patterns and risk factors associated with diseases. By analysing this data, machine learning algorithms can help identify individuals who are at risk of developing a particular disease, enabling healthcare professionals to provide personalized preventive care and early intervention.

Activity 2: Business Requirements

A disease classification project can have a variety of business requirements, depending on the specific goals and objectives of the project. Some potential requirements may include:

Accurate and reliable information

The case of disease prediction is critical and no false information can be tolerated since the consequences can be severe. Also the right symptoms should be linked to the right diseases so that the output is inline with all the patients health situations and variations.

Trust

Trust needs to be developed for the users to use the model. It is difficult to create trust among patients while dealing with a healthcare problem.

Compliance

The model should be fit with all the relevant laws and regulations, such as Central Drug Standard Control Organization, Ministry of Health, etc.

User friendly interface

The interface should be easy to use and understand by the user. The model should not ask inputs for which the user does not have answers.

Activity 3: Literature Survey

A literature survey for a disease prediction project would involve researching and reviewing existing studies, articles, and other publications on the topic of disease prediction. The survey would aim to gather information on current classification systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous disease prediction projects, and any relevant data or findings that could inform the design and implementation of the current project.

Activity 4: Social or Business Impact

Social Impact

Improved preventive diagnosis by predicting the likely disease. Users can easily see which is the probable disease for their symptoms and then decide whether to visit a doctor. This will also allow for better online diagnosis by doctors.

Business Impact

Doctors can treat wider range of patients using online consulting. The rush in the hospitals can be decreased and better care can be taken for critical patients. The doctors can suggest the patients to undergo certain tests before coming to visit them.

Milestone 2: Data Collection and Preparation:

After the identification of the problem at hand we start the project building process by finding publicly available datasets that satisfy our needs This data should be reliable and have information to build our model upon.

Activity 1: Collect the dataset

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

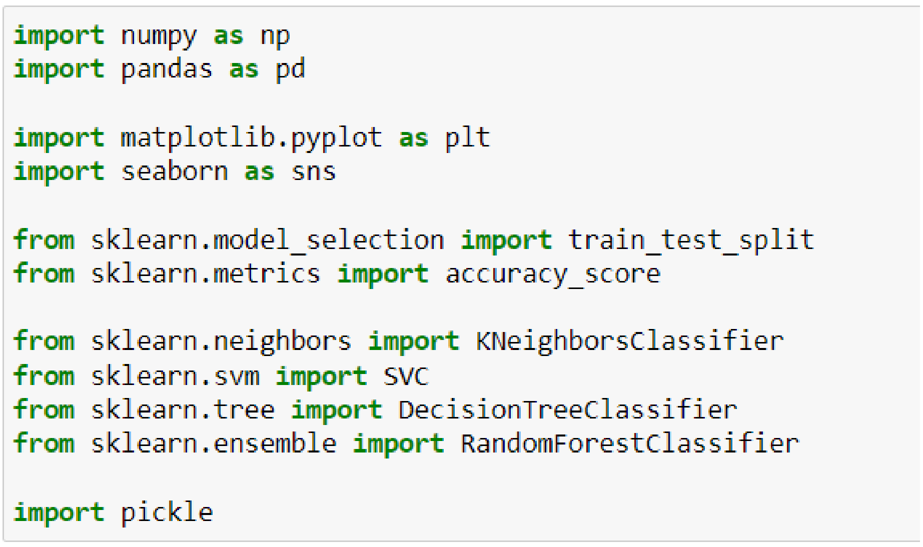
In this project we have used .csv data. This data is downloaded from kaggle.com. The dataset is downloaded from the link provided below,

Link:
https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning

Now that we have acquired the data we now need to explore the data and clean it as raw datasets contain lot of unwanted or incomplete values. Hence now we will start the data preprocessing and analysis part.

Activity 1.1: Importing the Libraries

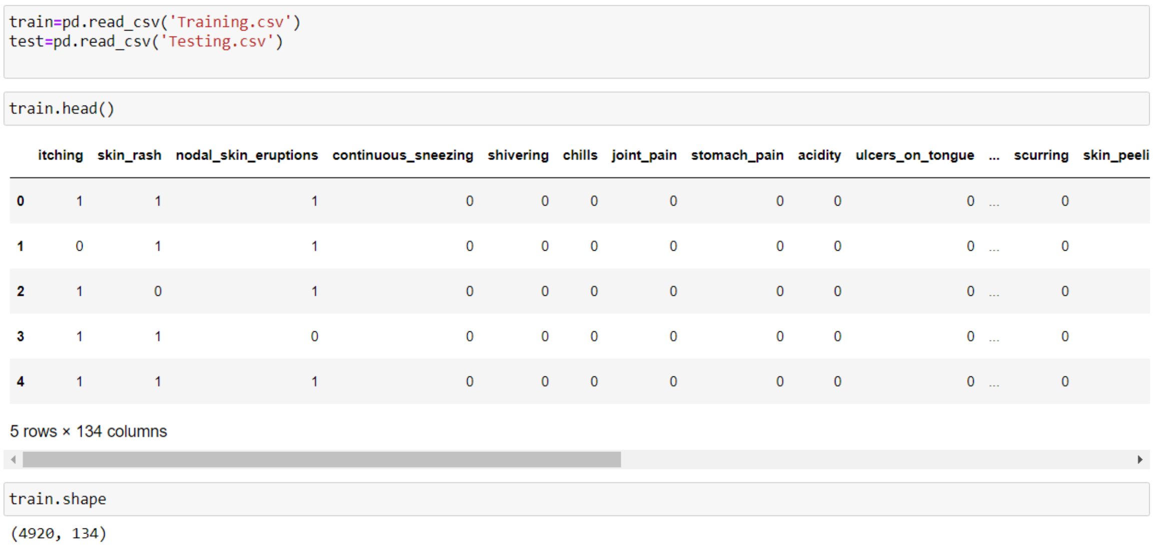
Import the necessary libraries as shown in the image.



```python
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

import pickle
```

Activity 1.2: Read the Dataset

In pandas we have a function called read_csv() to read the dataset. As a parameter we have togive the directory of the csv file. In our case we have the dataset stored in the same directory as the python file, hence we do not need to add the file path instead we can directly add the file name and it will be imported.



```
train=pd.read_csv('Training.csv')
test=pd.read_csv('Testing.csv')
```

```
train.head()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	scurring	skin_peeli
0	1	1	1	0	0	0	0	0	0	0	...	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	
2	1	0	1	0	0	0	0	0	0	0	...	0	
3	1	1	0	0	0	0	0	0	0	0	...	0	
4	1	1	1	0	0	0	0	0	0	0	...	0	

5 rows × 134 columns

```
train.shape
```

```
(4920, 134)
```

As we have two datasets, one for training and other for testing we will import both the csv files.

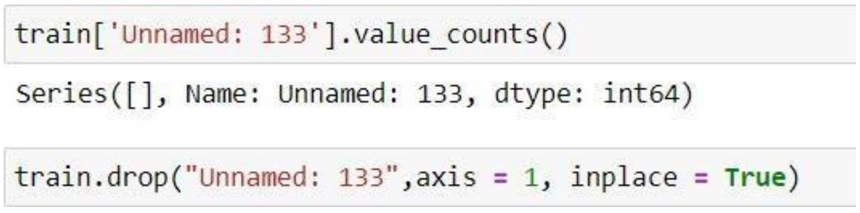
Activity 2: Data Preparation

As we have understood how the data is, let us preprocess the collected data.

The Machine Learning model cannot be trained on the imported data directly. The dataset might have randomness, we might have to clean the dataset and bring it in the right form. This activity involves the following steps:

- Removing Redundant Columns
- Handling Missing Values

Activity 2.1: Removing Redundant Columns



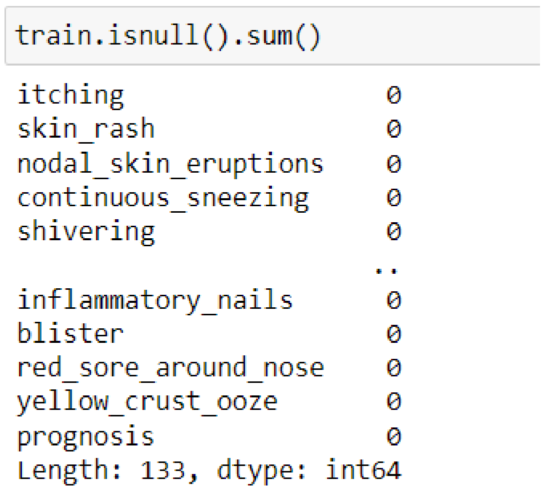
```
train['Unnamed: 133'].value_counts()
```

```
Series([], Name: Unnamed: 133, dtype: int64)
```

```
train.drop("Unnamed: 133",axis = 1, inplace = True)
```

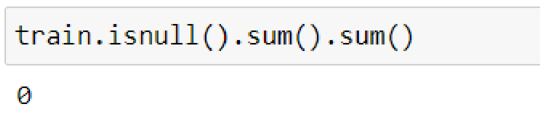
Unnamed: 133 is the redundant column which does not have any values.

Activity 2.2: Handling Missing Values



```
train.isnull().sum()
```

```
itching                 0
skin_rash               0
nodal_skin_eruptions    0
continuous_sneezing     0
shivering               0
                       ..
inflammatory_nails      0
blister                 0
red_sore_around_nose    0
yellow_crust_ooze       0
prognosis               0
Length: 133, dtype: int64
```



```
train.isnull().sum().sum()
```

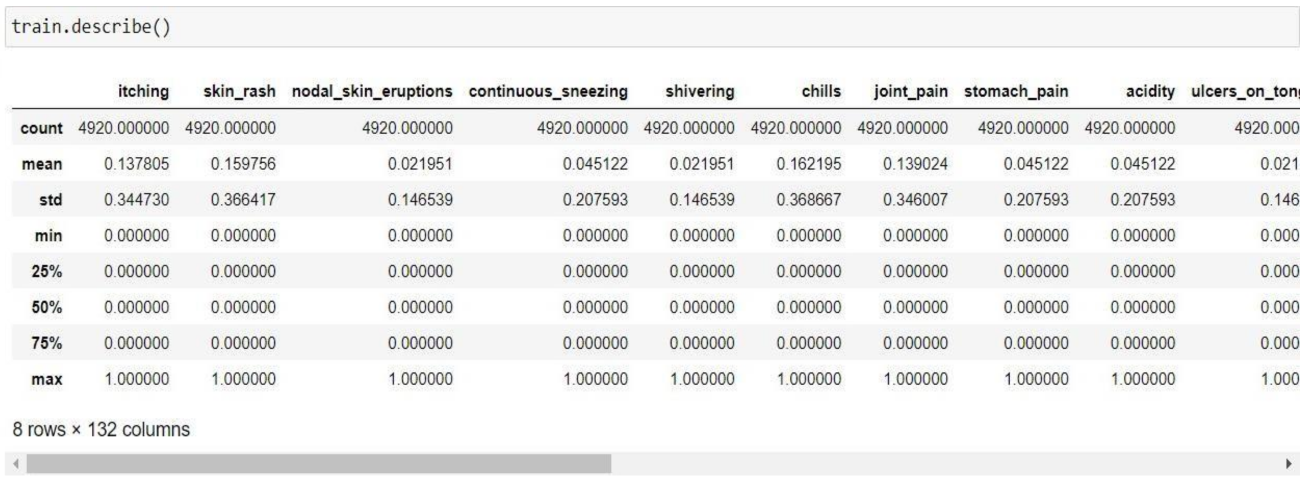
```
0
```

There are no missing values in the dataset.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

The describe function in pandas is a useful tool for exploring the basic characteristics of data using statistical methods. It can show us the distinct, most common and most frequent values of categorical variables. It can also calculate the mean, standard deviation, minimum, maximum and percentiles of numerical variables.



```
train.describe()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_ton
count	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000
mean	0.137805	0.159756	0.021951	0.045122	0.021951	0.162195	0.139024	0.045122	0.045122	0.021
std	0.344730	0.366417	0.146539	0.207593	0.146539	0.368667	0.346007	0.207593	0.207593	0.146
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

8 rows × 132 columns

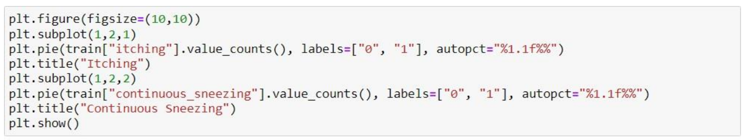
Activity 2: Visual Analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

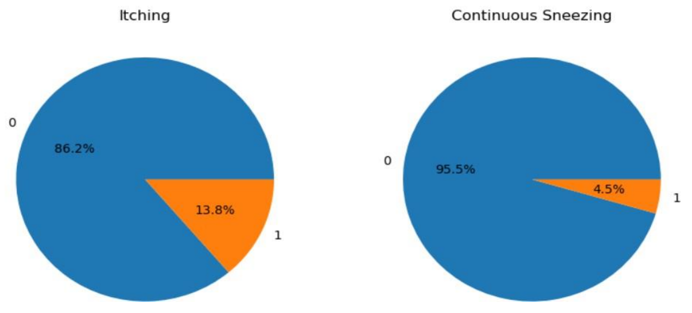
Activity 2.1: Univariate Analysis:

Univariate analysis is understanding the data with a single feature. We havedisplayed three different types of graphs and plots.

For simple visualizations we can use the matplotlib.pyplot library.



```
plt.figure(figsize=(10,10))
plt.subplot(1,2,1)
plt.pie(train["itching"].value_counts(), labels=["0", "1"], autopct="%1.1f%%")
plt.title("Itching")
plt.subplot(1,2,2)
plt.pie(train["continuous_sneezing"].value_counts(), labels=["0", "1"], autopct="%1.1f%%")
plt.title("Continuous Sneezing")
plt.show()
```

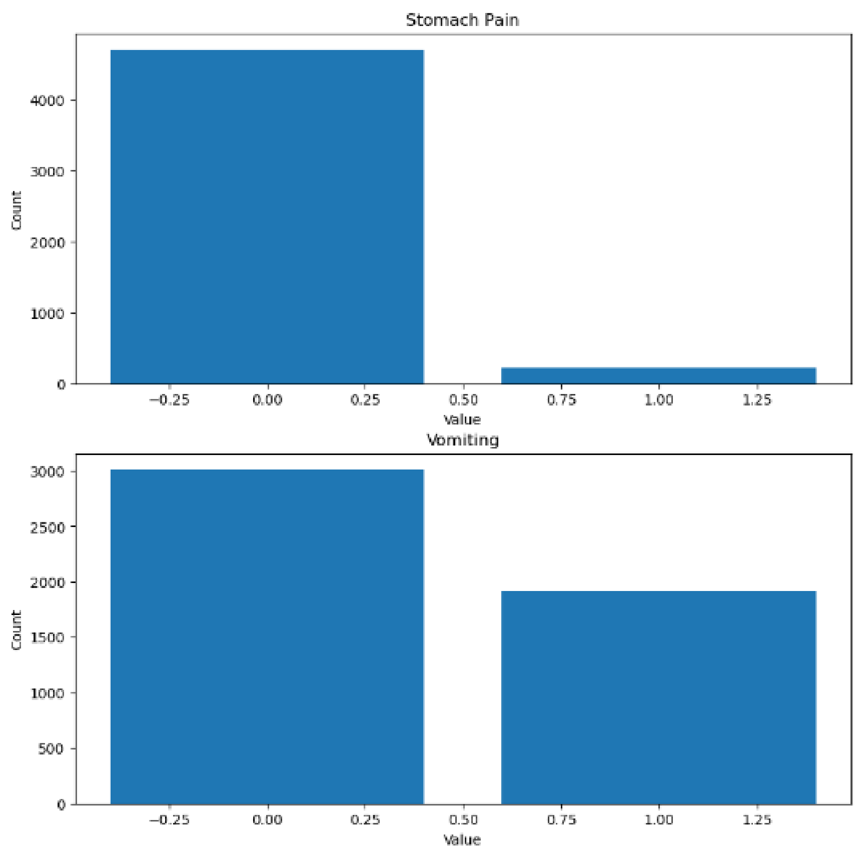


Here the plt.figure() command is used to determine the size of the plot.

The pie plot on the left shows the different values distribution in the Itching column. It shows that there are 86.2% observations where the itching symptom has value 0 and there are 13% observations where the itching symptom has value 1.

The pie plot on the right shows the different values distribution in the continuous_sneezing column. It shows that there are 95.5% observations where the continuous_sneezing symptom has value 0 and there are 4.5% observations where the continuous_sneezing symptom has value 1.

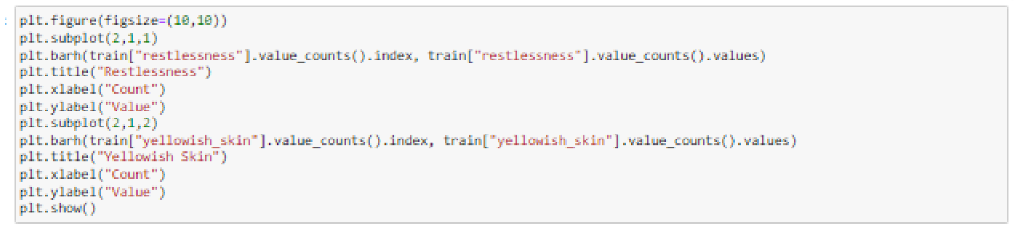
```
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
plt.bar(train["stomach_pain"].value_counts().index, train["stomach_pain"].value_counts().values)
plt.title("Stomach Pain")
plt.xlabel("Value")
plt.ylabel("Count")
plt.subplot(2,1,2)
plt.bar(train["vomiting"].value_counts().index, train["vomiting"].value_counts().values)
plt.title("Vomiting")
plt.xlabel("Value")
plt.ylabel("Count")
plt.show()
```



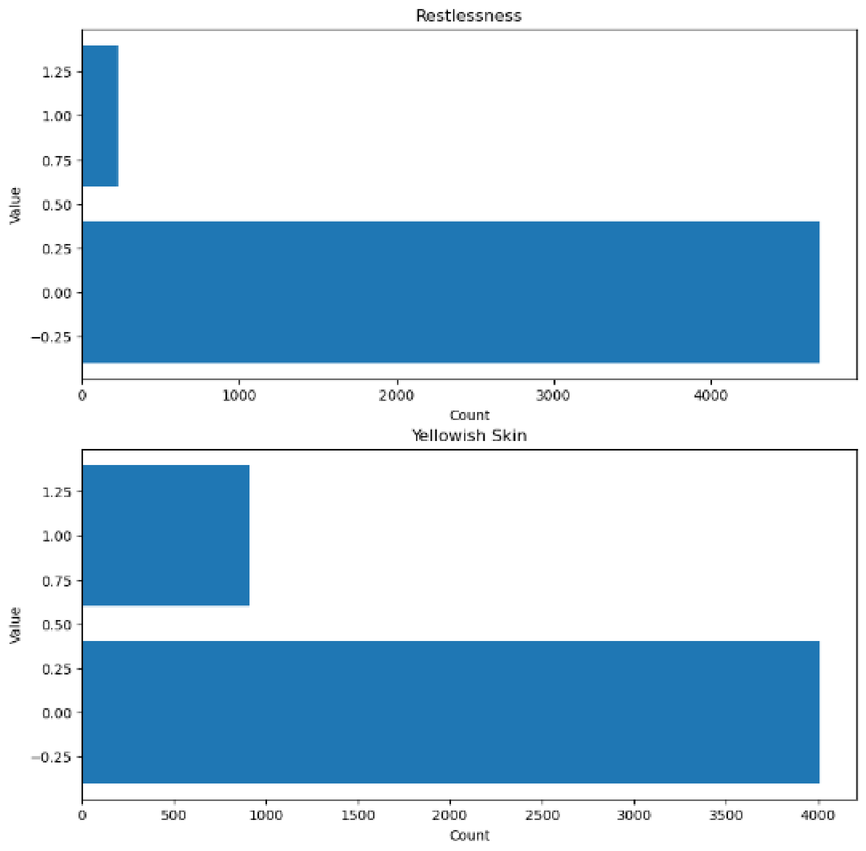
Here we have plotted 2 bar graphs. These bar graphs can be plotted without using any external library. We divide the plot into two subplots using subplot function from the matplotlib.pyplot library. We have plotted the bar graph using the inbuilt plot function in python.

The bar graph on the top shows the distribution of stomach pain symptom values. We can see that the 0 value has count of above 4500 and the 1 value has count of below 500.

The graph on the bottom shows the distribution of vomiting symptom values. We can see thatthe 0 value has count of around 3000 and the 1 value has count of around 2000.



```
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
plt.barh(train["restlessness"].value_counts().index, train["restlessness"].value_counts().values)
plt.title("Restlessness")
plt.xlabel("Count")
plt.ylabel("Value")
plt.subplot(2,1,2)
plt.barh(train["yellowish_skin"].value_counts().index, train["yellowish_skin"].value_counts().values)
plt.title("Yellowish Skin")
plt.xlabel("Count")
plt.ylabel("Value")
plt.show()
```



Here we have plotted 2 horizontal bar graphs. These bar graphs can be plotted without using any external library. We divide the plot into two subplots using subplot function from the matplotlib.pyplot library. We have plotted the bar graph using the inbuilt plot function in python.

The bar graph on the top shows the distribution of restlessness symptom values. We can see that the 0 value has count of around 4800 and the 1 value has count of around 300.

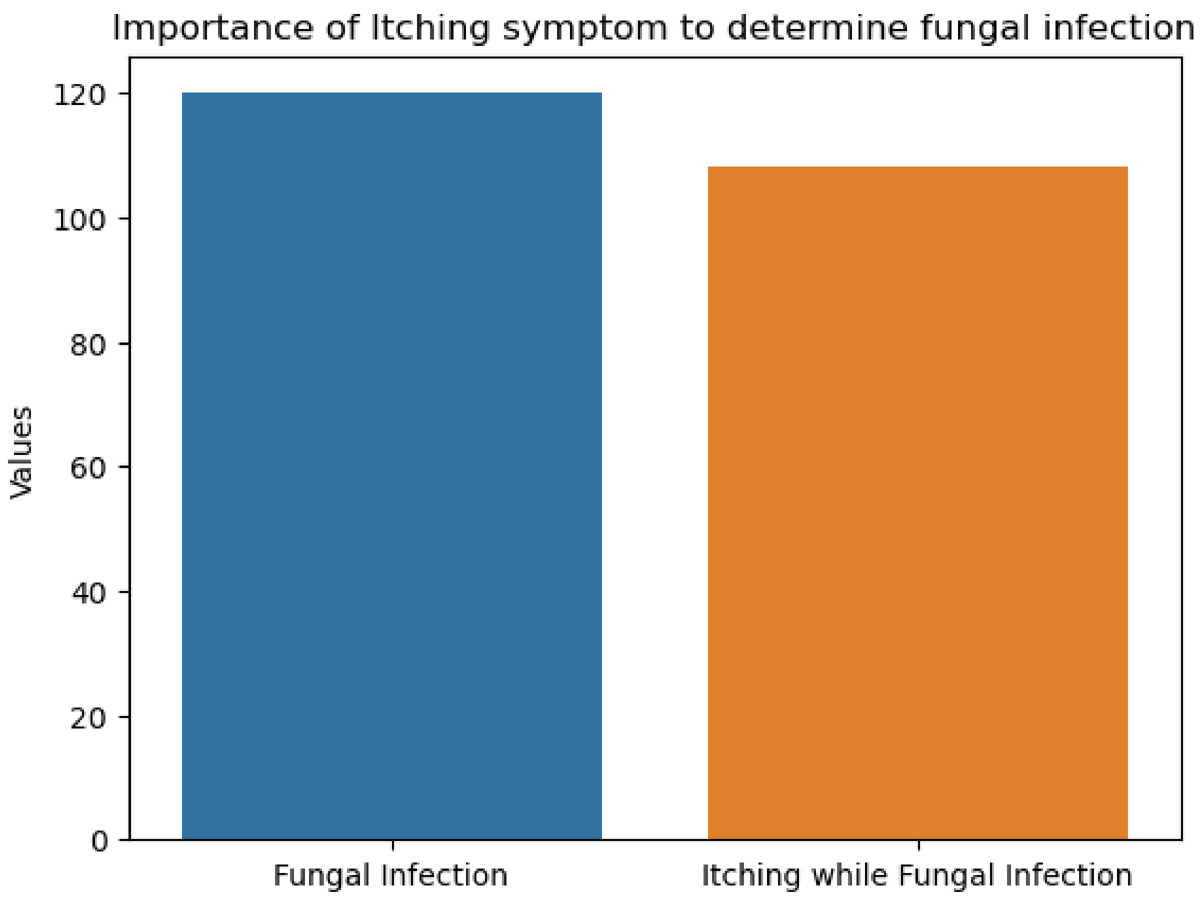
The graph on the bottom shows the distribution of vomiting symptom values. We can see thatthe 0 value has count of around 4500 and the 1 value has count of around 400.

Activity 2.2: Bivariate Analysis:

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between prognosis where the values are Fungal Infection and Itching symptom.

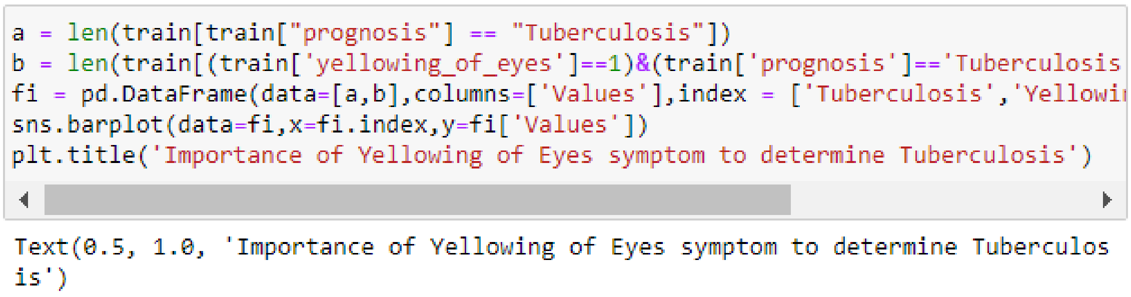
```
a = len(train[train["prognosis"] == "Fungal infection"])
b = len(train[(train['itching']==1)&(train['prognosis']=='Fungal infection')])
fi = pd.DataFrame(data=[a,b],columns=['Values'],index = ['Fungal Infection','Itc
sns.barplot(data=fi,x=fi.index,y=fi['Values'])
plt.title('Importance of Itching symptom to determine fungal infection')
```

```
Text(0.5, 1.0, 'Importance of Itching symptom to determine fungal infection')
```



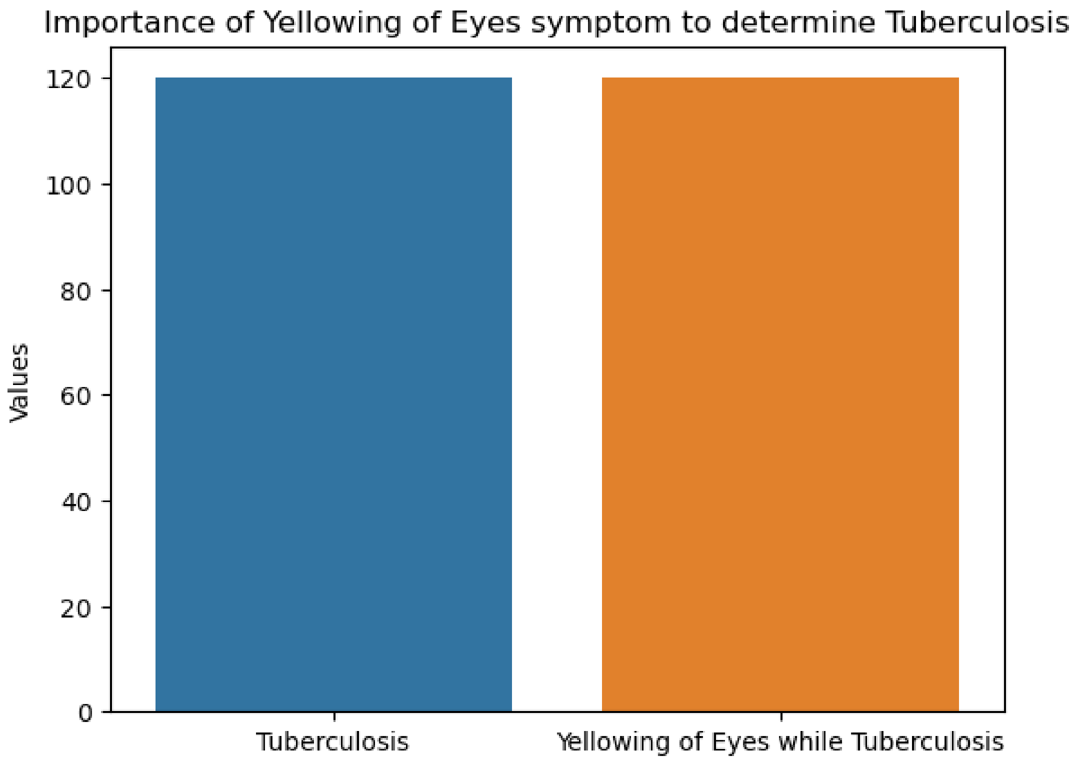
We use a method called Boolean Indexing to select only the rows where the prognosis column has the value ‘Fungal Infection’. We store these rows in a variable called ‘a’. We also select only the rows where the prognosis column has the value ‘Fungal Infection’ and the Itching column has the value 1. From the plot, we can see that when someone has Fungal Infection, they are very likely to have Itching as a symptom. There are 120 rows where the prognosis is Fungal Infection and there are 104 rows where the Itching is 1. This shows that Fungal Infection and Itching are strongly related.

We also looked at how the prognosis and the symptom yellowing_of_eyes are related when the disease is Tuberculosis. From the plot, we can see that when someone has Tuberculosis, they are very likely to have yellowing_of_eyes as a symptom. There are 120 rows where the prognosis is Tuberculosis and there are 119 rows where the yellowing_of_eyes is 1. This shows that Tuberculosis and yellowing_of_eyes are strongly related.



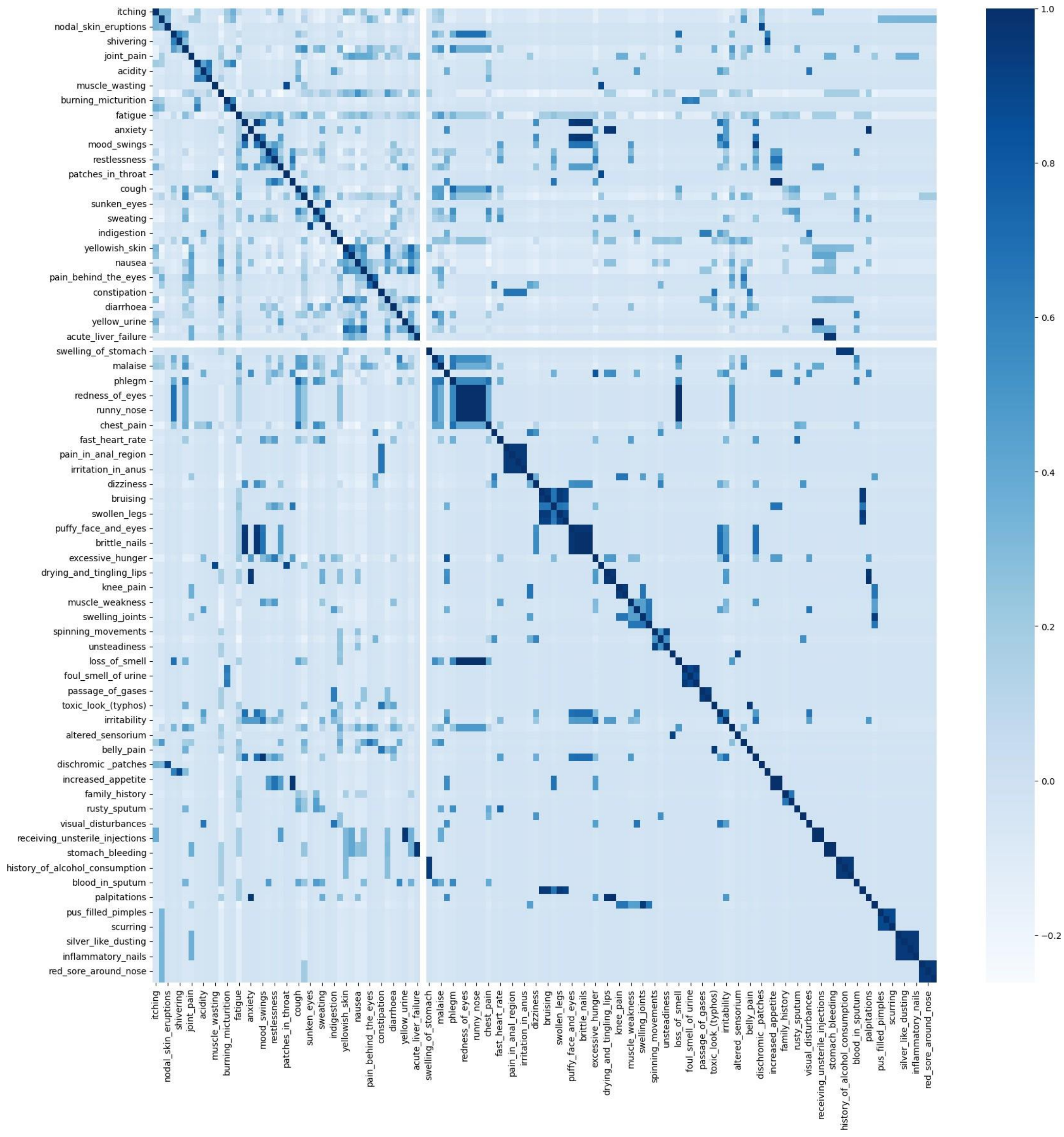
```
a = len(train[train["prognosis"] == "Tuberculosis"])
b = len(train[(train['yellowing_of_eyes']==1)&(train['prognosis']=='Tuberculosis
fi = pd.DataFrame(data=[a,b],columns=['Values'],index = ['Tuberculosis','Yellowi
sns.barplot(data=fi,x=fi.index,y=fi['Values'])
plt.title('Importance of Yellowing of Eyes symptom to determine Tuberculosis')
```

```
Text(0.5, 1.0, 'Importance of Yellowing of Eyes symptom to determine Tuberculos
is')
```



Activity 2.3: Multivariate Analysis

In multivariate analysis we try to find the relation between multiple features. This can be done primarily with the help of Correlation matrix.

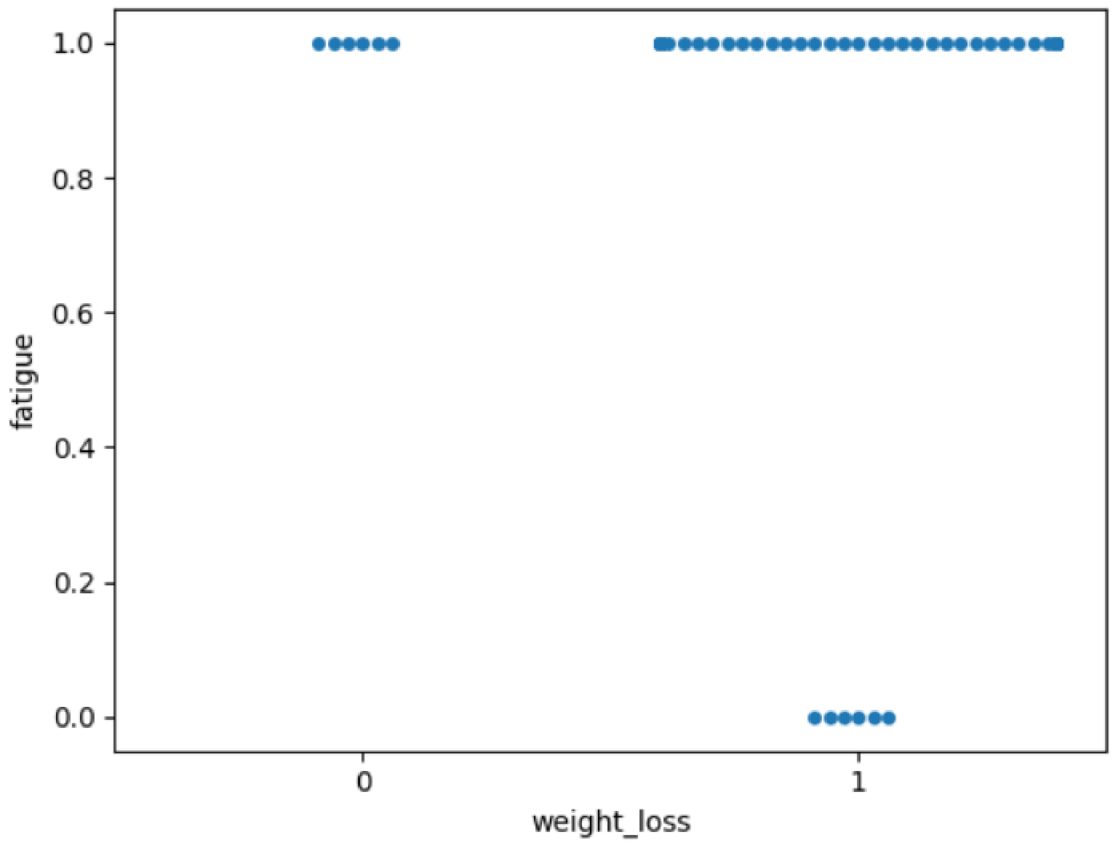


As we have 131 columns which have numerical values the correlation matrix is of dimensions 131 X 131. These many features can only be parsed by scrolling. From the correlation matrix we try to remove the values which are highly correlated with each other. When 2 values are highly correlated with each other, we can only remove one of them. We remove columns where the correlation between the columns is above 0.9

```
train.drop(['weight_gain','cold_hands_and_feets','anxiety','irregular_sugar_level',
            'yellow_urine','acute_liver_failure','swelling_of_stomach',
            'drying_and_tingling_lips','continuous_feel_of_urine',
           'internal_itching','polyuria','mood_swings','receiving_unsterile_injections',
           'stomach_bleeding','prominent_veins_on_calf','loss_of_smell','throat_irritation',
           'redness_of_eyes','sinus_pressure','runny_nose','pain_during_bowel_movements',
           'pain_in_anal_region','cramps','bruising','enlarged_thyroid','brittle_nails',
           'swollen_extremeties','slurred_speech','distention_of_abdomen','fluid_overload.1',
           'skin_peeling','silver_like_dusting','small_dents_in_nails','blister',
           'red_sore_around_nose','bloody_stool','swollen_blood_vessels','hip_joint_pain',
           'painful_walking','spinning_movements','altered_sensorium','toxic_look_(typhos)'],axis =1, inplace = True)
```

We can see the columns that are removed due to high correlation.

For multivariate analysis we will also plot a swarmplot of weightloss and fatigue column where the prognosis column is Tuberculosis.



From this swarm plot we can see that for Tuberculosis disease, there is no observation when the fatigue and weight loss is 0. There are some cases when there is only either of the two, but for Tuberculosis there is a high chance that the patient will have fatigue and weight loss as symptoms.

Preprocessing of Test data

The preprocessing needs to be done for the test data. We can create a function for test data preprocessing which will only leave us with the required features. This function will contain all the steps which we have done for the training data

```
def data_preprocessing(data):
    data.drop(['fluid_overload','weight_gain','cold_hands_and_feets','anxiety','irregular_sugar_level',
           'yellow_urine','acute_liver_failure','swelling_of_stomach',
           'drying_and_tingling_lips','continuous_feel_of_urine',
          'internal_itching','polyuria','mood_swings','receiving_unsterile_injections',
          'stomach_bleeding','prominent_veins_on_calf','loss_of_smell','throat_irritation',
          'redness_of_eyes','sinus_pressure','runny_nose','pain_during_bowel_movements',
          'pain_in_anal_region','cramps','bruising','enlarged_thyroid','brittle_nails',
          'swollen_extremeties','slurred_speech','distention_of_abdomen','fluid_overload.1',
          'skin_peeling','silver_like_dusting','small_dents_in_nails','blister',
          'red_sore_around_nose','bloody_stool','swollen_blood_vessels','hip_joint_pain',
          'painful_walking','spinning_movements','altered_sensorium','toxic_look_(typhos)'],axis =1, inplace = True)
    return data
```

This function drops all the columns which needs to be dropped.

```
test = data_preprocessing(test)
```

Here we call the function for the test data.

Activity 2.5: Split data into training, validation and testing data

We have training and testing data given separately. We further split the training data into training and validation data. This validation data can be used for hyper parameter tuning.

We first need to separate the features and the target variable. The features are used to predict the target variable.

```
X = train.drop('prognosis',axis = 1)
y = train.prognosis
```

We split the training data into features(X) and target variable(y).

```
X_test = test.drop('prognosis',axis = 1)
y_test = test.prognosis
```

Here we split the test data into features(X_test) and the corresponding target variables(y_test)

Now we need to split the training data into training and validation data. It can be done using the following command.

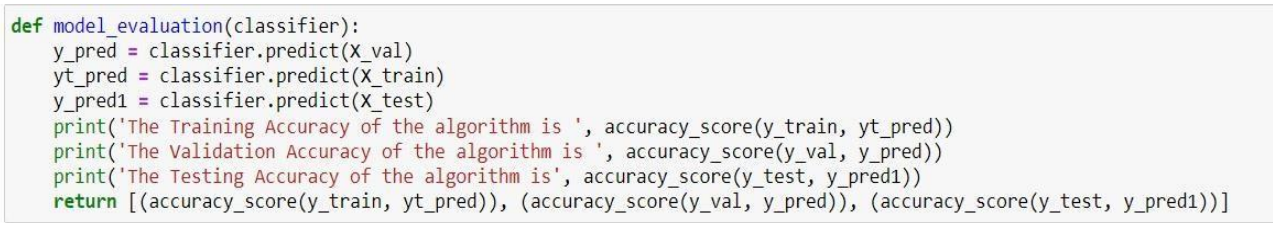
```
X_train, X_val, y_train, y_val = train_test_split(X,y,test_size = 0.2)
```

We have kept 80 % data for training and 20% is used for validation

<u>**Milestone 4: Model Building**</u>

Activity 1: Creating a function for model evaluation

We will be creating four different models using four different machine learning algorithm and choose the most accurate model amongst them.



```
def model_evaluation(classifier):
    y_pred = classifier.predict(X_val)
    yt_pred = classifier.predict(X_train)
    y_pred1 = classifier.predict(X_test)
    print('The Training Accuracy of the algorithm is ', accuracy_score(y_train, yt_pred))
    print('The Validation Accuracy of the algorithm is ', accuracy_score(y_val, y_pred))
    print('The Testing Accuracy of the algorithm is', accuracy_score(y_test, y_pred1))
    return [(accuracy_score(y_train, yt_pred)), (accuracy_score(y_val, y_pred)), (accuracy_score(y_test, y_pred1))]
```

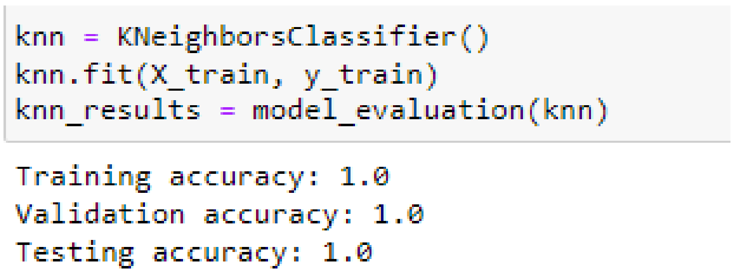
This function will show the accuracies of prediction of model for training, validation and testing data. It will also the return those accuracies.

Activity 2: Training and testing the models using multiple algorithms

Now that we have clean data, a function for evaluation it is time to build models to train the data. For this project we will be using 4 different classification algorithms to build our models. The best model will be used for prediction.

Activity 2.1: K Nearest Neighbors Model

We create a variable called knn and assign it the KNeighborsClassifier() algorithm. We use the .fit() function to train the knn model on the X_train and y_train data, which are the training features and training target variables. Then, we use the model_evaluation function to check how well the model performs.



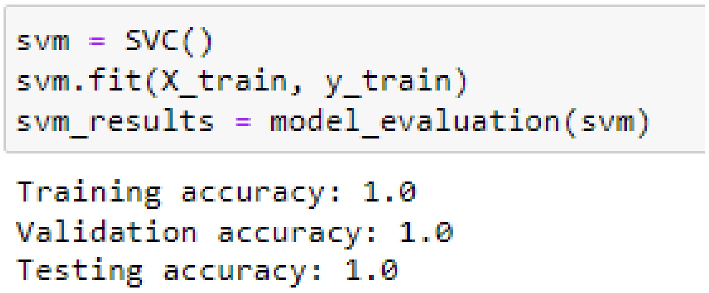
```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_results = model_evaluation(knn)
```

```
Training accuracy: 1.0
Validation accuracy: 1.0
Testing accuracy: 1.0
```

Here we can see that the model has achieved 100% accuracies on training, validation as well as testing data. As the accuracies are high, there is no need for hyperparametertuning. The results are stored in a variable named knn_results.

Activity 2.2: SVM Model

We make a variable named svm and give it the SVC() algorithm. We train the svm model on the X_train and y_train data, which have the training features and training target variables. Then, we check how well the model does by using the model_evaluation function.



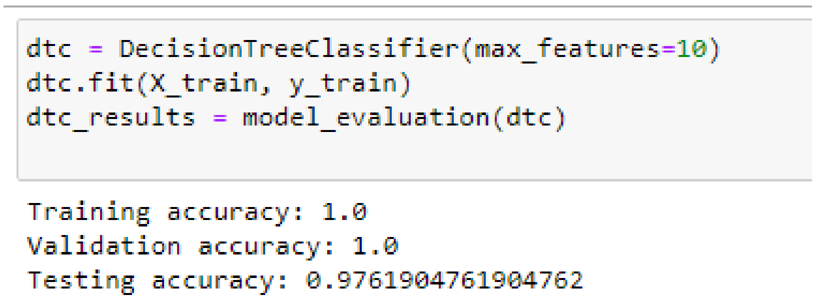
```
svm = SVC()
svm.fit(X_train, y_train)
svm_results = model_evaluation(svm)
```

```
Training accuracy: 1.0
Validation accuracy: 1.0
Testing accuracy: 1.0
```

Here we can see that the model has achieved 100% accuracies on training, validation as well as testing data. As the accuracies are high, there is no need for hyperparametertuning. The results are stored in a variable named svm_results.

Activity 2.3: Decision Tree Model

We create a variable called svm and assign it the SVC() algorithm. We use the .fit() function to train the svm model on the X_train and y_train data, which are the training features and training target variables. Then, we use the model_evaluation function to see how the model performs,



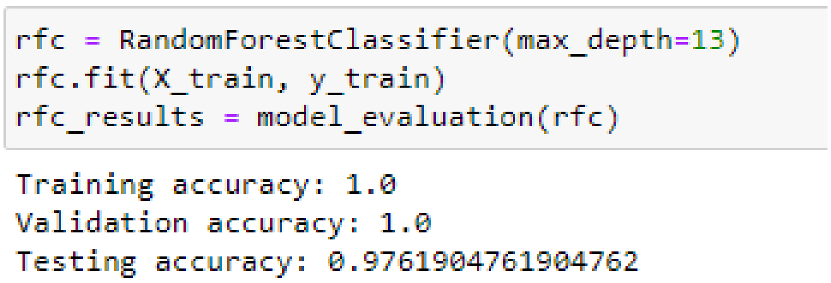
```
dtc = DecisionTreeClassifier(max_features=10)
dtc.fit(X_train, y_train)
dtc_results = model_evaluation(dtc)
```

```
Training accuracy: 1.0
Validation accuracy: 1.0
Testing accuracy: 0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training and validation data .
It has achieved 97.6% accuracy for testing data. As the accuracies arehigh, there is no need for hyperparameter tuning. The results are stored in a variable named dtc_results.

Activity 2.4: Random Forest Model

Random Forest Classifier is a type of Bagging model that uses many decision trees and combines their results to make a prediction. We make a variable called rfc and give it the RandomForestClassifier() algorithm with a parameter max_depth of 13. We use the .fit() function to train the rfc model on the X_train and y_train data, which have the training features and training target variables. Then, we see how the model does by using the model_evaluation function.



```
rfc = RandomForestClassifier(max_depth=13)
rfc.fit(X_train, y_train)
rfc_results = model_evaluation(rfc)
```

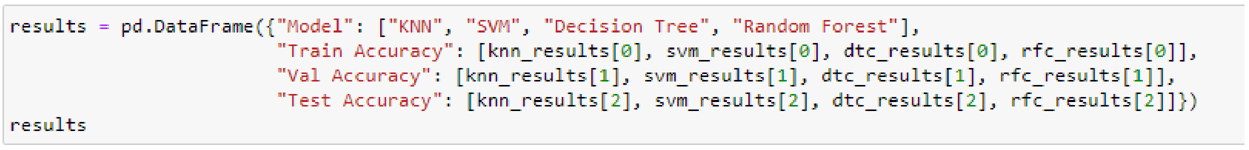
```
Training accuracy: 1.0
Validation accuracy: 1.0
Testing accuracy: 0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training and validation data .
It has achieved 97.6% accuracy for testing data. As the accuracies arehigh, there is no need for hyperparameter tuning. The results are stored in a variable named rfc_results.

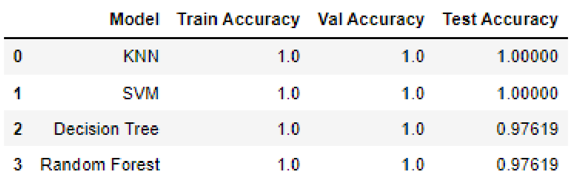
Milestone 5 : Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with Multiple Evaluation metrics

The data has 41 features, hence it is difficult to make confusion matrix as the dimensions of confusion matrix will be 41 X 41. We can check accuracy to test the model. We have already values of the training, validation, and test accuracies of various models. We can put them in a table and then check for the best model.



```
results = pd.DataFrame({"Model": ["KNN", "SVM", "Decision Tree", "Random Forest"],
                        "Train Accuracy": [knn_results[0], svm_results[0], dtc_results[0], rfc_results[0]],
                        "Val Accuracy": [knn_results[1], svm_results[1], dtc_results[1], rfc_results[1]],
                        "Test Accuracy": [knn_results[2], svm_results[2], dtc_results[2], rfc_results[2]]})
results
```



	Model	Train Accuracy	Val Accuracy	Test Accuracy
0	KNN	1.0	1.0	1.00000
1	SVM	1.0	1.0	1.00000
2	Decision Tree	1.0	1.0	0.97619
3	Random Forest	1.0	1.0	0.97619

From the table we can see that KNN and SVM models perform the best.

Activity 2: Comparing model accuracy before and after applying hyperparameter tuning

As the accuracies are already so high, we need not do hyperparameter tuning for the models.

Activity 3: Comparing Model accuracy for different number of features.

Currently the training data has 90 features, which are a high number. If we need to reduce the number of features, we need to check the accuracies for various number of features.

We can check the feature importance using the Random Forest Classifier model.

In the figure below we have created a dictionary with the column names as indexes and the values as their feature importance. Much importance is not assigned to any feature. It is distributed among all the features.

We will keep some number of features for training and check for the accuracy. This process will be repeated a number of times.

```python
a = rfc.feature_importances_
```

```python
col = X.columns
```

```python
feat_imp = {}
for i, j in zip(a,col):
    feat_imp[j] = i
```

```python
feat_imp
```

```
{'itching': 0.014906774548182978,
 'skin_rash': 0.004126193310423079,
 'nodal_skin_eruptions': 0.00506477497511548,
 'continuous_sneezing': 0.011950815998563227,
 'shivering': 0.0148244155625889,
 'chills': 0.008051206993917127,
 'joint_pain': 0.015897929037963693,
 'stomach_pain': 0.010154098478372642,
 'acidity': 0.008624253805092569,
 'ulcers_on_tongue': 0.007019300375694471,
 'muscle_wasting': 0.007044463040109562,
```

We get a dictionary named feat_imp with 90 column names and their feature importance.

We will drop columns which have very less feature importance.

Let us create a for loop which will train the model and give out the accuracy.

```python
rfc_results = []
knn_results = []
```

```python
for main in [0.020,0.018,0.016,0.014,0.012,0.01,0.008]:
    to_drop = []
    for i,j in zip(feat_imp.keys(),feat_imp.values()):
        if j < main:
            to_drop.append(i)

    X_new = X.drop(to_drop,axis = 1)
    y_new = y
    X1_train, X1_val, y1_train, y1_val = train_test_split(X_new, y_new, test_size=0.2)
    X1_test = X_test.drop(to_drop,axis = 1)
    y1_test = y_test
    rfc_new = RandomForestClassifier()
    rfc_new.fit(X1_train, y1_train)
    temp1 = model_evaluation1(X1_train.shape[1], rfc_new)
    rfc_results.append(temp1)
    knn_new = KNeighborsClassifier()
    knn_new.fit(X1_train, y1_train)
    temp2 = model_evaluation1(X1_train.shape[1],knn_new)
    knn_results.append(temp2)
```

Here we create 2 lists for 2 models, knn and random forest classifier.

The for loop will iterate over values given in the list one be one. The first value will be 0.020 and the last will be 0.008

There is a to_drop list created. If the feature_importance is below threshold then the column name will be added to theto_drop list. The columns whose name is in the to_drop list will be dropped.

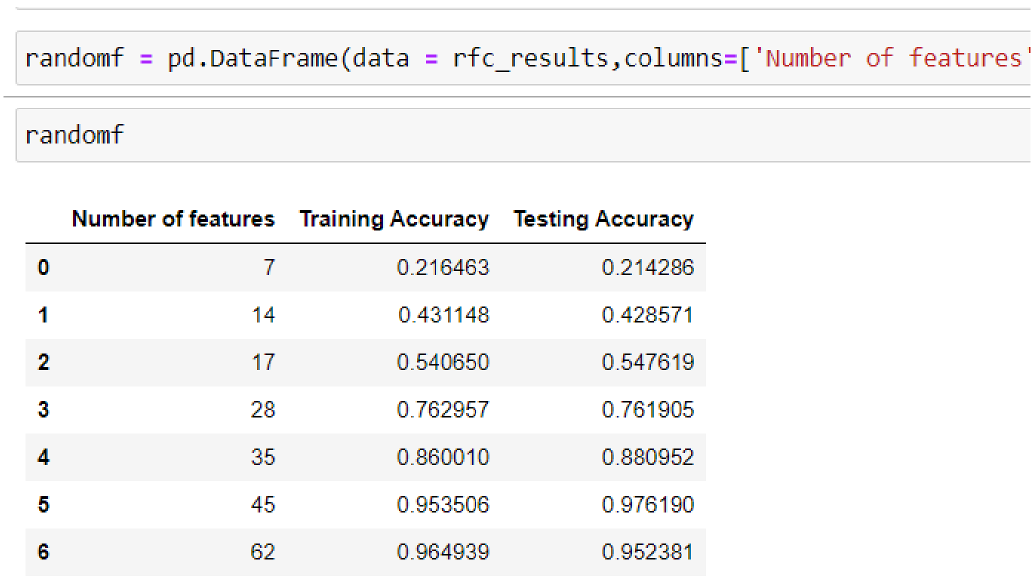
The new data will be split into features and target variable. Further they will be split into training, validation and testing data.

Random Forest Classifier model will be trained and its accuracy will be stored in the list.

Knn model will be trained and its accuracy will be stored in the list.

This process will go on till all the values for i are iterated.

We then plot a table using the number of features and accuracies.

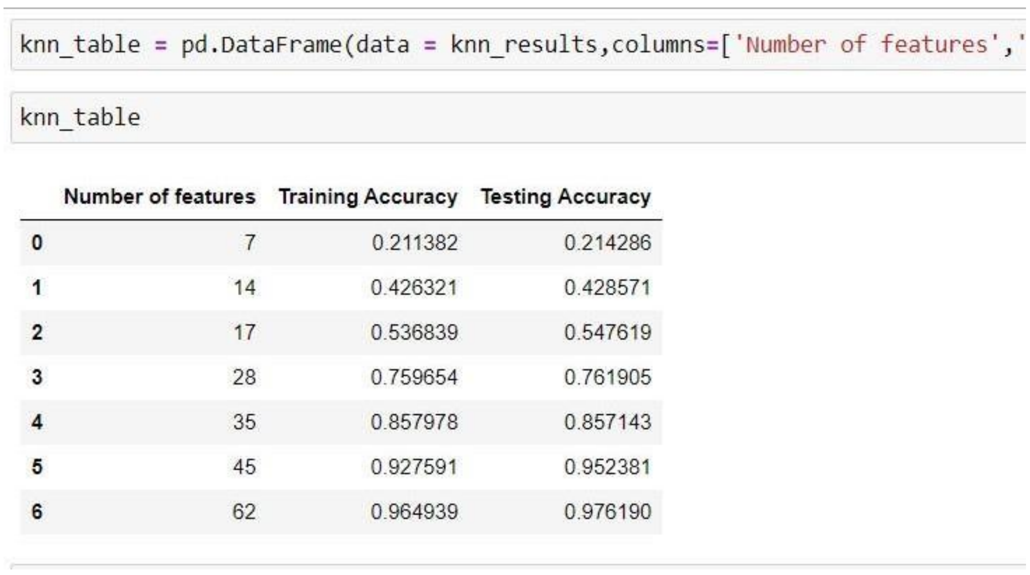


```
randomf = pd.DataFrame(data = rfc_results,columns=['Number of features'
```

```
randomf
```

	Number of features	Training Accuracy	Testing Accuracy
0	7	0.216463	0.214286
1	14	0.431148	0.428571
2	17	0.540650	0.547619
3	28	0.762957	0.761905
4	35	0.860010	0.880952
5	45	0.953506	0.976190
6	62	0.964939	0.952381

This is the table for random forest Classifier for various features. We can see that as the number of features go on increasing, the accuracies increase.



```
knn_table = pd.DataFrame(data = knn_results,columns=['Number of features','
```

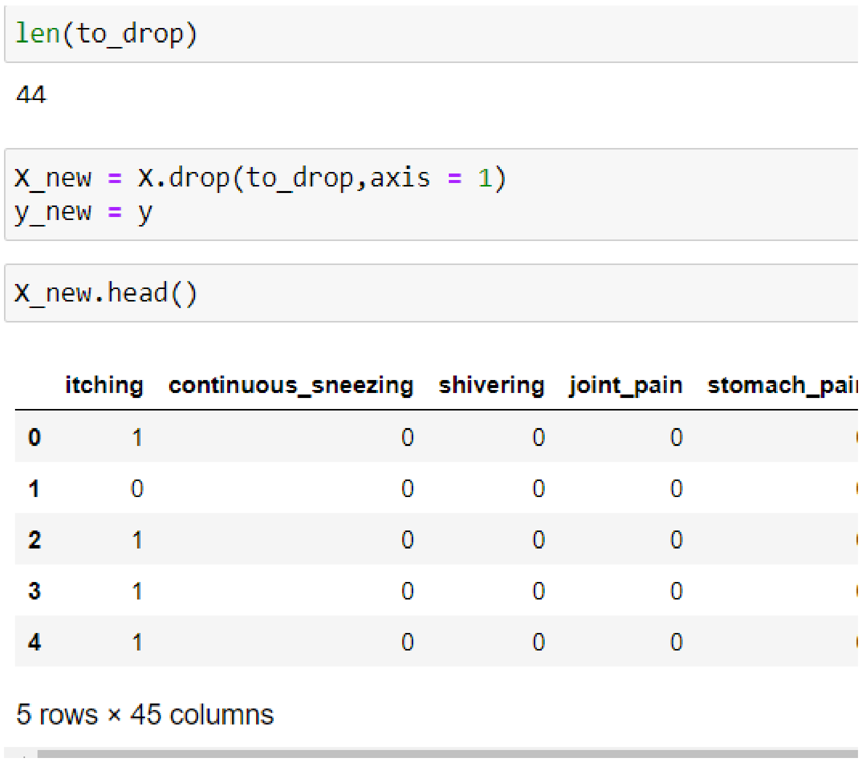
```
knn_table
```

	Number of features	Training Accuracy	Testing Accuracy
0	7	0.211382	0.214286
1	14	0.426321	0.428571
2	17	0.536839	0.547619
3	28	0.759654	0.761905
4	35	0.857978	0.857143
5	45	0.927591	0.952381
6	62	0.964939	0.976190

This is the table for knn model for various number of features. We can see that as the number of features go on increasing, the accuracies increase.

Activity 4: Building Model with appropriate features

From the above result tables, we can see that the accuracy does not change much from 45 features to 62 features. Hence we will choose 45 features for our training.



```
len(to_drop)
```

```
44
```

```
X_new = X.drop(to_drop,axis = 1)
y_new = y
```

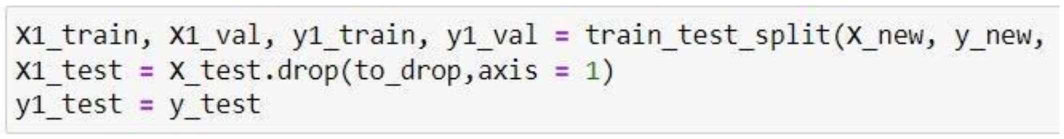
```
X_new.head()
```

	itching	continuous_sneezing	shivering	joint_pain	stomach_pai
0	1	0	0	0	
1	0	0	0	0	
2	1	0	0	0	
3	1	0	0	0	
4	1	0	0	0	

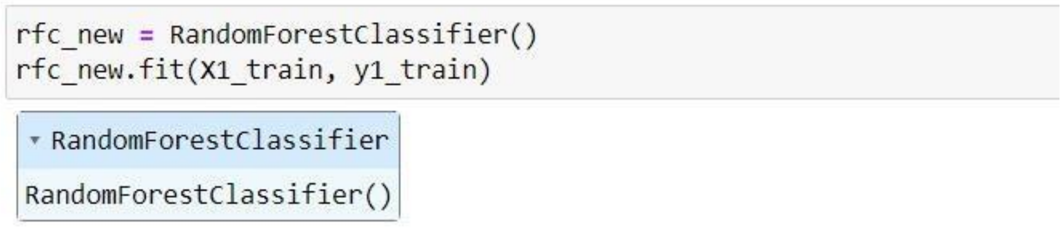
5 rows × 45 columns

We drop 44 features. Create new datasets for features and their labels. As we can see in the figure above we are left with 45 features now. We will build a Random Fores Classifier and KNN model on the new data and check for the accuracies.

As we can see in the figure below our model has achieved test accuracy of 97.6 % which is quite good for the number of features. Previouslyfor 90 features we had similar accuracy for Random Forest Classifier. This states that there were many features which were not contributing much to our model.

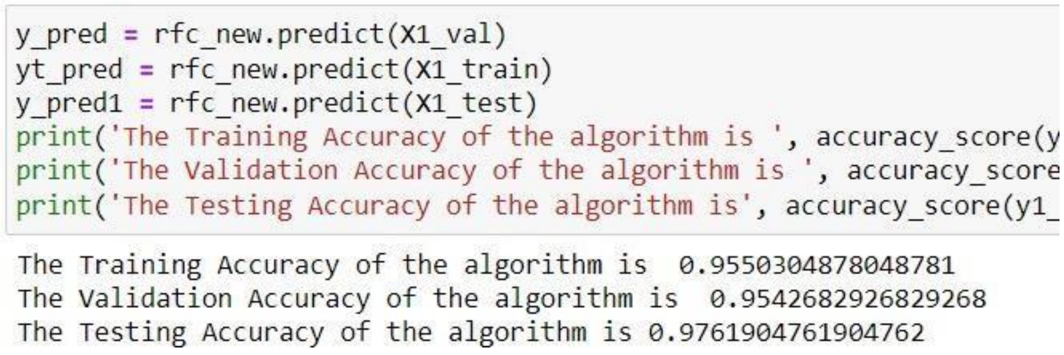


```
X1_train, X1_val, y1_train, y1_val = train_test_split(X_new, y_new,
X1_test = X_test.drop(to_drop,axis = 1)
y1_test = y_test
```



```
rfc_new = RandomForestClassifier()
rfc_new.fit(X1_train, y1_train)
```

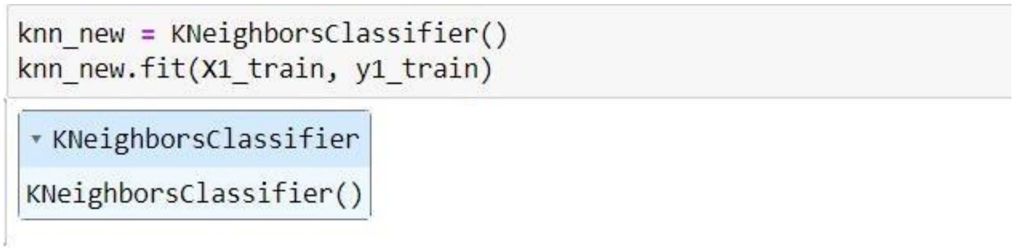
```
▾ RandomForestClassifier
RandomForestClassifier()
```



```
y_pred = rfc_new.predict(X1_val)
yt_pred = rfc_new.predict(X1_train)
y_pred1 = rfc_new.predict(X1_test)
print('The Training Accuracy of the algorithm is ', accuracy_score(y
print('The Validation Accuracy of the algorithm is ', accuracy_score
print('The Testing Accuracy of the algorithm is', accuracy_score(y1_
```

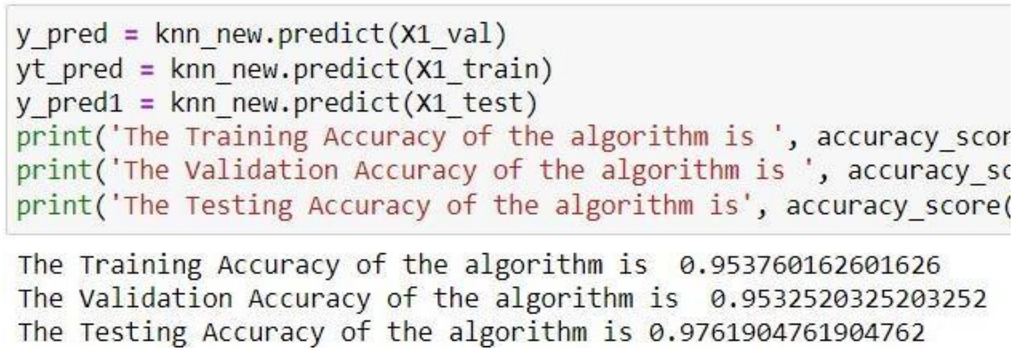
```
The Training Accuracy of the algorithm is  0.9550304878048781
The Validation Accuracy of the algorithm is  0.9542682926829268
The Testing Accuracy of the algorithm is 0.9761904761904762
```

We will also train the model for KNN algorithm as KNN algorithm tends to perform better in such cases.



```
knn_new = KNeighborsClassifier()
knn_new.fit(X1_train, y1_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

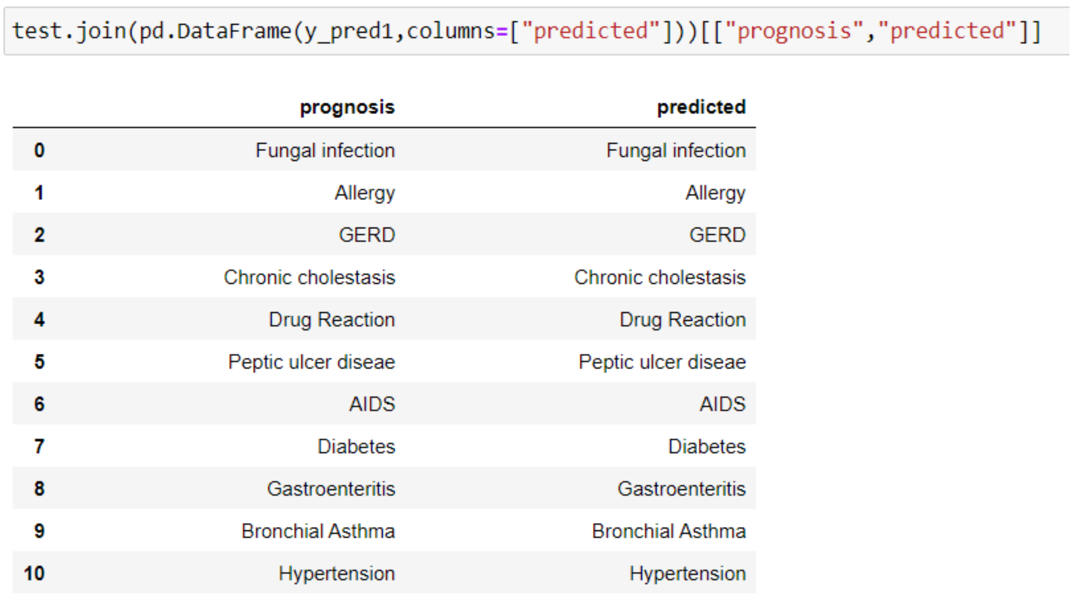


```
y_pred = knn_new.predict(X1_val)
yt_pred = knn_new.predict(X1_train)
y_pred1 = knn_new.predict(X1_test)
print('The Training Accuracy of the algorithm is ', accuracy_scor
print('The Validation Accuracy of the algorithm is ', accuracy_sc
print('The Testing Accuracy of the algorithm is', accuracy_score(
```

```
The Training Accuracy of the algorithm is  0.953760162601626
The Validation Accuracy of the algorithm is  0.9532520325203252
The Testing Accuracy of the algorithm is 0.9761904761904762
```

After training the knn model we check the accuracies. Our model has achieved 97.6 % accuracy for the test data.

To confirm let us check the compare our predicted results with the actual values.



```
test.join(pd.DataFrame(y_pred1,columns=["predicted"]))[["prognosis","predicted"]]
```

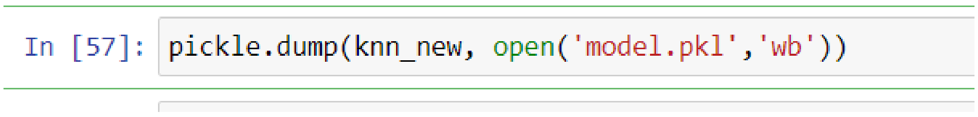
	prognosis	predicted
0	Fungal infection	Fungal infection
1	Allergy	Allergy
2	GERD	GERD
3	Chronic cholestasis	Chronic cholestasis
4	Drug Reaction	Drug Reaction
5	Peptic ulcer diseae	Peptic ulcer diseae
6	AIDS	AIDS
7	Diabetes	Diabetes
8	Gastroenteritis	Gastroenteritis
9	Bronchial Asthma	Bronchial Asthma
10	Hypertension	Hypertension

As we can see above that the values our model has predicted are same as the actual values. This shows that our model is performing good.

Milestone 6: Model Deployment

Activity 1: Save the best model

To save the best model after comparing how well it does using different ways to measure its performance, we choose the model that does the best and save its settings and structure. This can help us to avoid training the model again every time we need it and also to use it later.After checking the performance, we decide to save the knn model built with 45 features.



```
In [57]: pickle.dump(knn_new, open('model.pkl','wb'))
```

We save the model using the pickle library into a file named model.pkl

Activity 2: Integrate with Web Framework

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

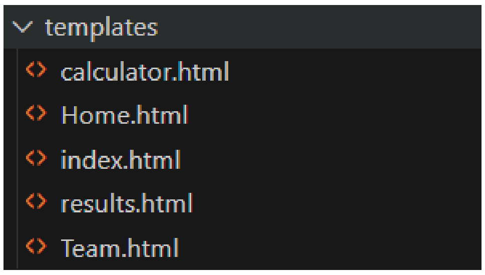
- Building HTML Pages
- Building server-side script for Machine Learning and login/registration purposes.
- Run the web application

Activity 2.1: Building HTML pages:

For this project we create three HTML files namely

- calculator.html
- Home.html
- index.html
- results.html
- team.html

And we will save them in the templates folder.

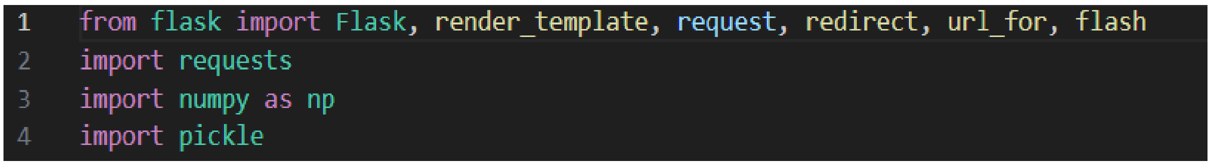


```
templates
  calculator.html
  Home.html
  index.html
  results.html
  Team.html
```

Activity 2.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

Import the necessary Libraries.

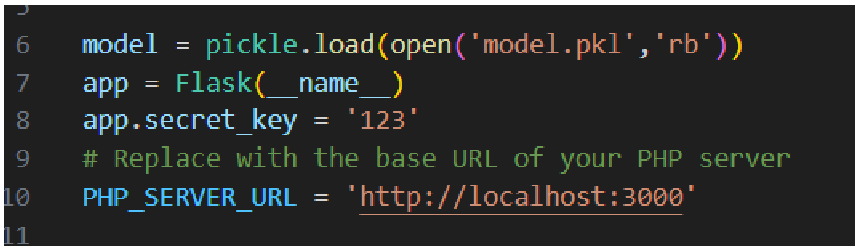


```python
from flask import Flask, render_template, request, redirect, url_for, flash
import requests
import numpy as np
import pickle
```

This code first loads the saved KNN model from the "model.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should beopened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

The PHP_SERVER_URL is used as base to redirect us from the flask app to php login/register page.

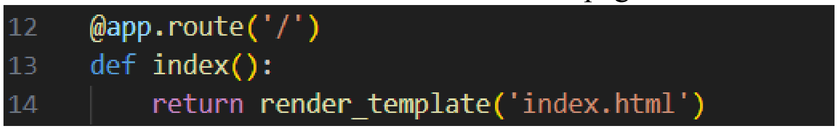


```python
model = pickle.load(open('model.pkl','rb'))
app = Flask(__name__)
app.secret_key = '123'
# Replace with the base URL of your PHP server
PHP_SERVER_URL = 'http://localhost:3000'
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.

The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

The "render_template()" method is used to render an HTML template named "index.html". The "index.html" is the home page.

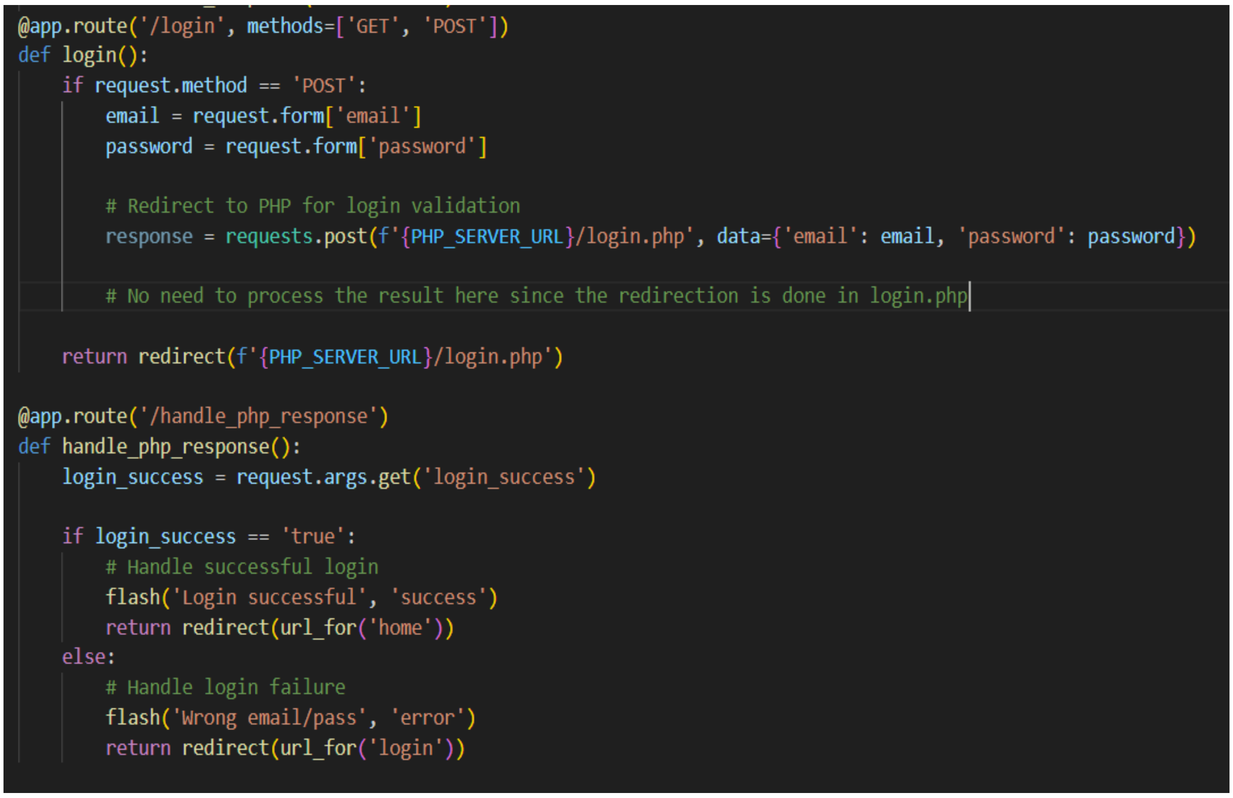


```python
@app.route('/')
def index():
    return render_template('index.html')
```

This code sets up a new route for the Flask web application using the "@app.route()"decorator. The route in this case is the root route "/login".

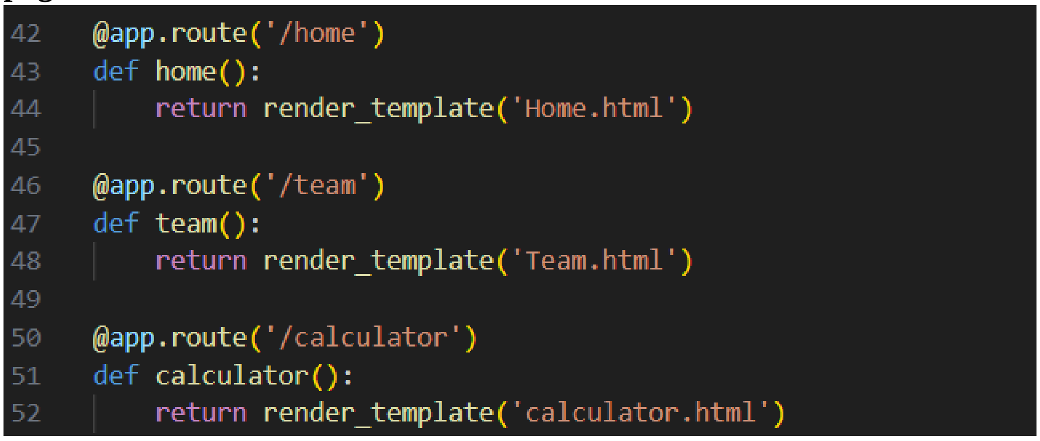
The route in this case is "/login". When a user accesses the "/login" route of the website, this function is "login ()" called. This takes us to the PHP server containing login and register page.

The function that follows soon after handles the success message of the login page by redirecting us to the Home page of our website.



```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        # Redirect to PHP for login validation
        response = requests.post(f'{PHP_SERVER_URL}/login.php', data={'email': email, 'password': password})

        # No need to process the result here since the redirection is done in login.php

    return redirect(f'{PHP_SERVER_URL}/login.php')

@app.route('/handle_php_response')
def handle_php_response():
    login_success = request.args.get('login_success')

    if login_success == 'true':
        # Handle successful login
        flash('Login successful', 'success')
        return redirect(url_for('home'))
    else:
        # Handle login failure
        flash('Wrong email/pass', 'error')
        return redirect(url_for('login'))
```

The route in this case is "/home". When a user accesses the "/home" route of the website, this function is “index()” called. The "render_template()" method is used to render an HTML template named "Home.html". **Similarly its done for the other pages too.**



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

@app.route('/team')
def team():
    return render_template('Team.html')

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

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/predict", and the method is set to GET and POST.

The function "predict()" is then associated with this route. In this function we create a list named col which has all the 45 column names that we have used in our model. In the details.html page we are going to take inputs from the user, which will be in the form of text.

The values are stored in request.form.values()

These values are stored in a list named inputt in the form of strings.

A list is created with 45 0s and stored in variable b.

In the for loop x will take values from 0 to 45.

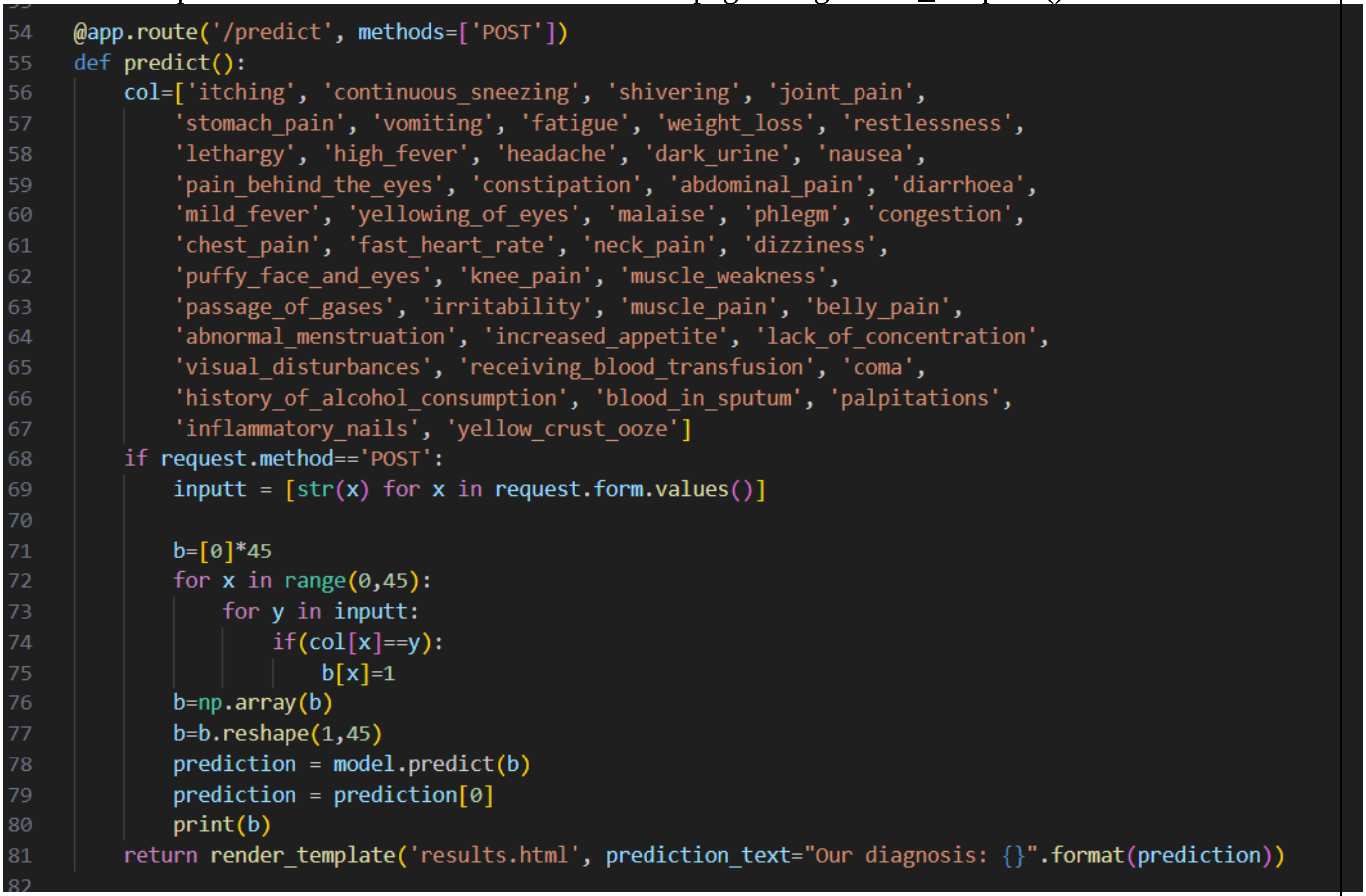
Another for loop is written where y will iterate over the values given by the user as inputs.

If the name of the column which is at the x index in col list matches with the y from the inputt list then a 1 is stored at that index in the list b.

This list b is converted into an array and the shape of the array is changed to (1,45).

Then this array b is given to the model for prediction.

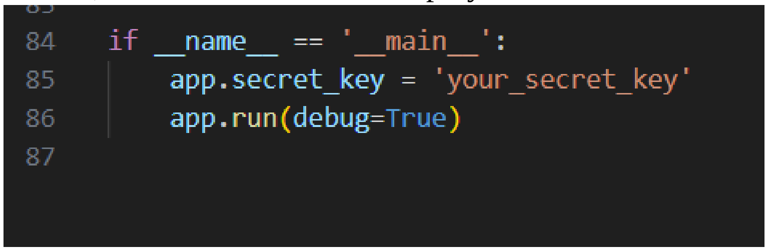
This prediction is returned to the results.html page using render_template()



```
@app.route('/predict', methods=['POST'])
def predict():
    col=['itching', 'continuous_sneezing', 'shivering', 'joint_pain',
        'stomach_pain', 'vomiting', 'fatigue', 'weight_loss', 'restlessness',
        'lethargy', 'high_fever', 'headache', 'dark_urine', 'nausea',
        'pain_behind_the_eyes', 'constipation', 'abdominal_pain', 'diarrhoea',
        'mild_fever', 'yellowing_of_eyes', 'malaise', 'phlegm', 'congestion',
        'chest_pain', 'fast_heart_rate', 'neck_pain', 'dizziness',
        'puffy_face_and_eyes', 'knee_pain', 'muscle_weakness',
        'passage_of_gases', 'irritability', 'muscle_pain', 'belly_pain',
        'abnormal_menstruation', 'increased_appetite', 'lack_of_concentration',
        'visual_disturbances', 'receiving_blood_transfusion', 'coma',
        'history_of_alcohol_consumption', 'blood_in_sputum', 'palpitations',
        'inflammatory_nails', 'yellow_crust_ooze']
    if request.method=='POST':
        inputt = [str(x) for x in request.form.values()]

        b=[0]*45
        for x in range(0,45):
            for y in inputt:
                if(col[x]==y):
                    b[x]=1
        b=np.array(b)
        b=b.reshape(1,45)
        prediction = model.predict(b)
        prediction = prediction[0]
        print(b)
    return render_template('results.html', prediction_text="Our diagnosis: {}".format(prediction))
```

Main Function:

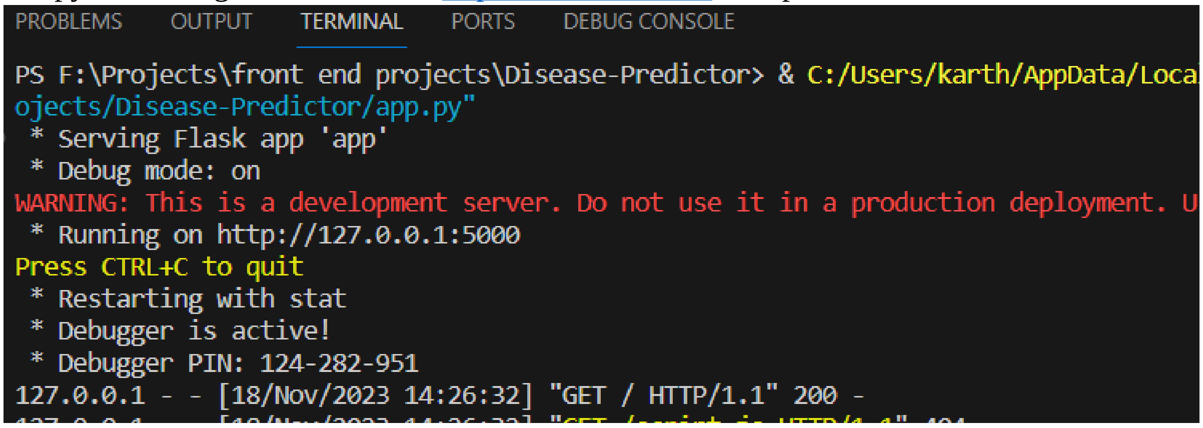
This code sets the entry point of the Flask application. The function “app.run()” is called, which starts the Flask deployment server.



```python
if __name__ == '__main__':
    app.secret_key = 'your_secret_key'
    app.run(debug=True)
```

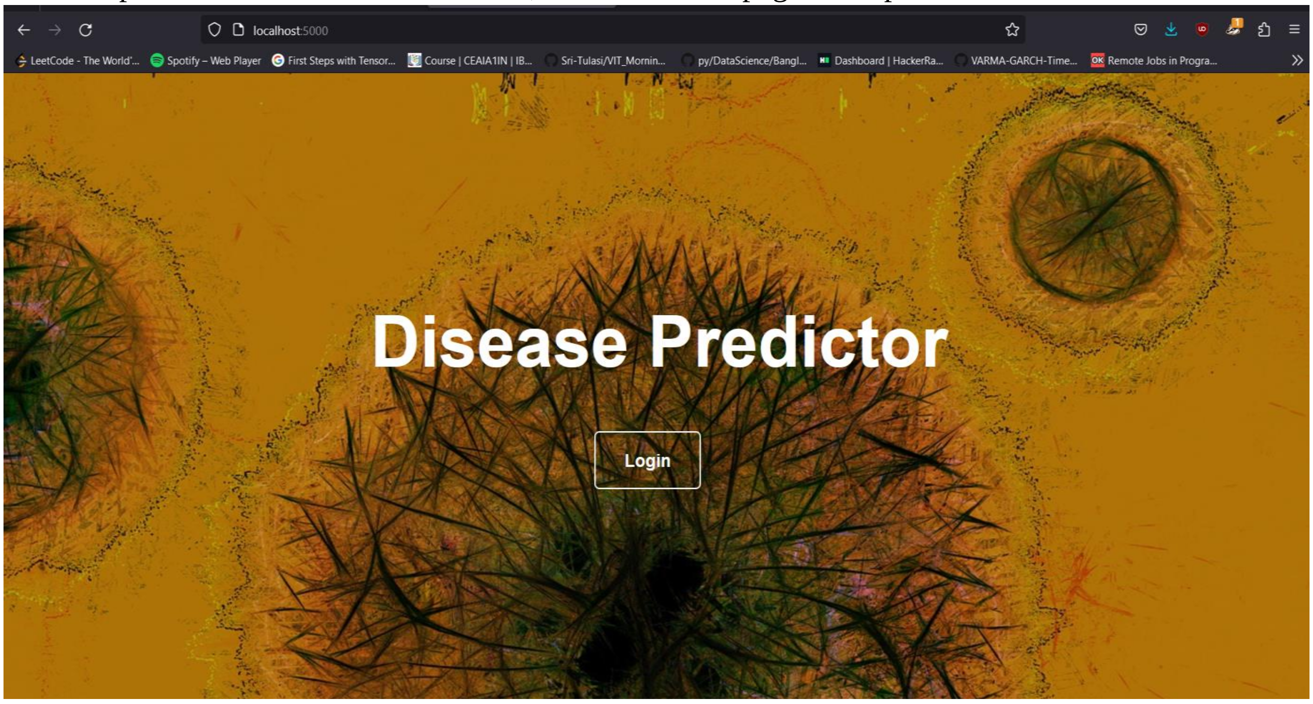
Activity 2.3: Run the Web Application

When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser.

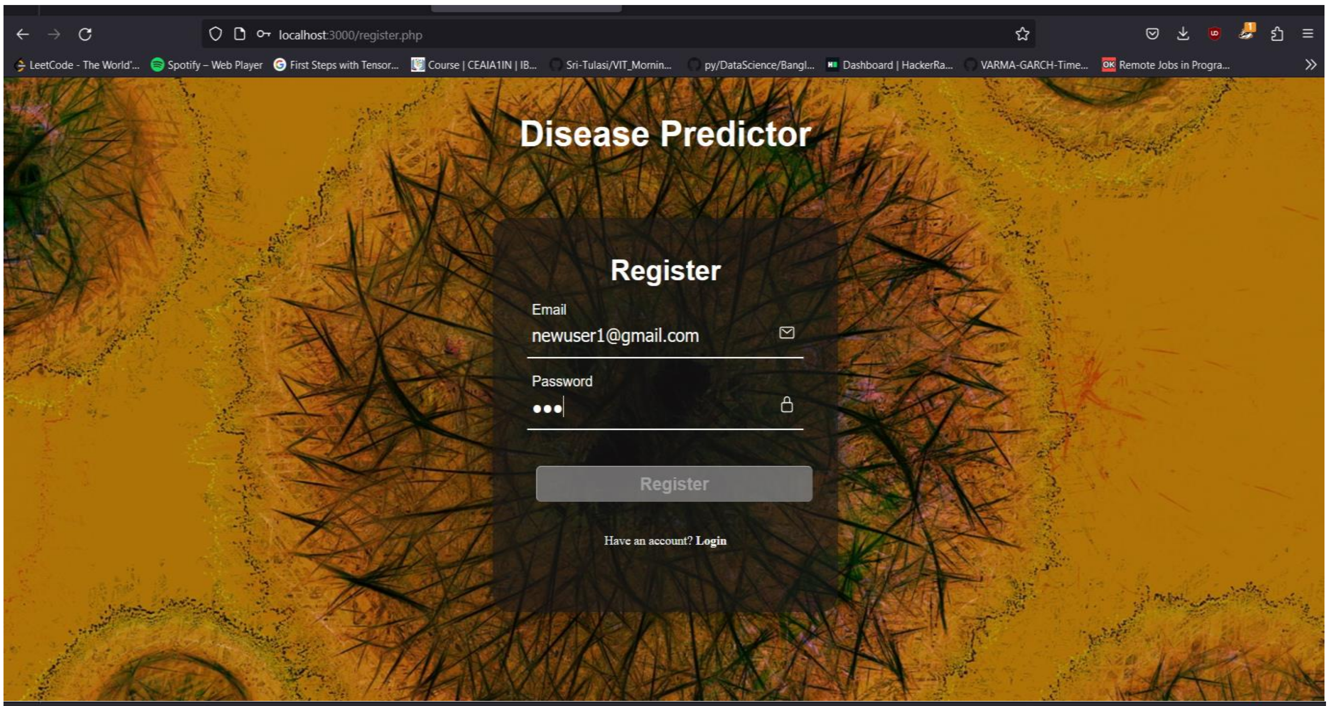


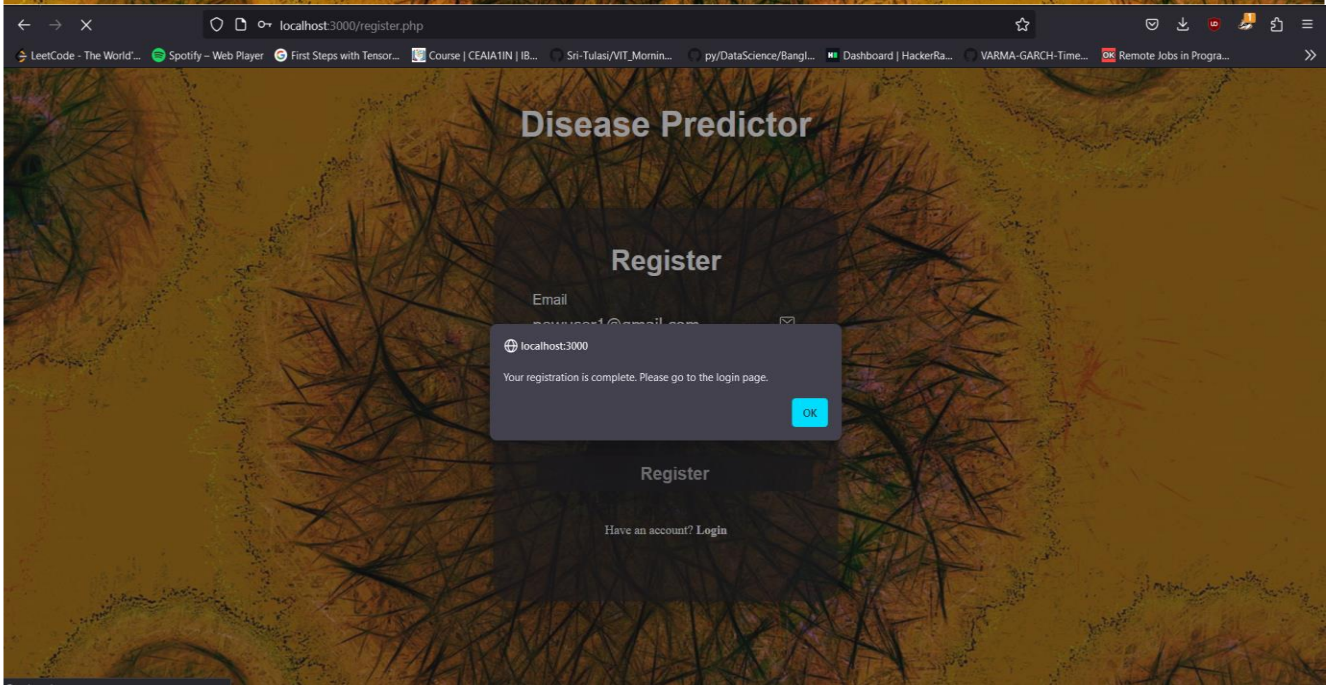
```
PS F:\Projects\front end projects\Disease-Predictor> & C:/Users/karth/AppData/Loca
ojects/Disease-Predictor/app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. U
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 124-282-951
127.0.0.1 - - [18/Nov/2023 14:26:32] "GET / HTTP/1.1" 200 -
```

When we paste the URL in a web browser, our index.html page will open.

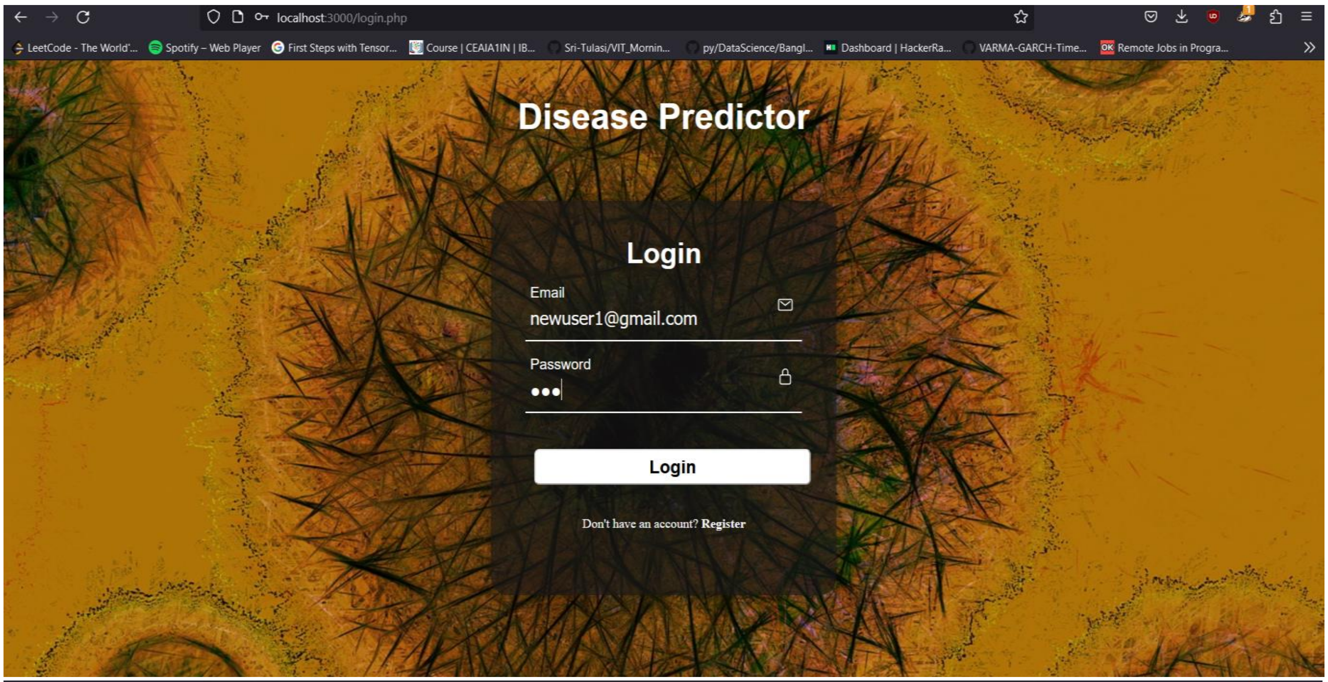


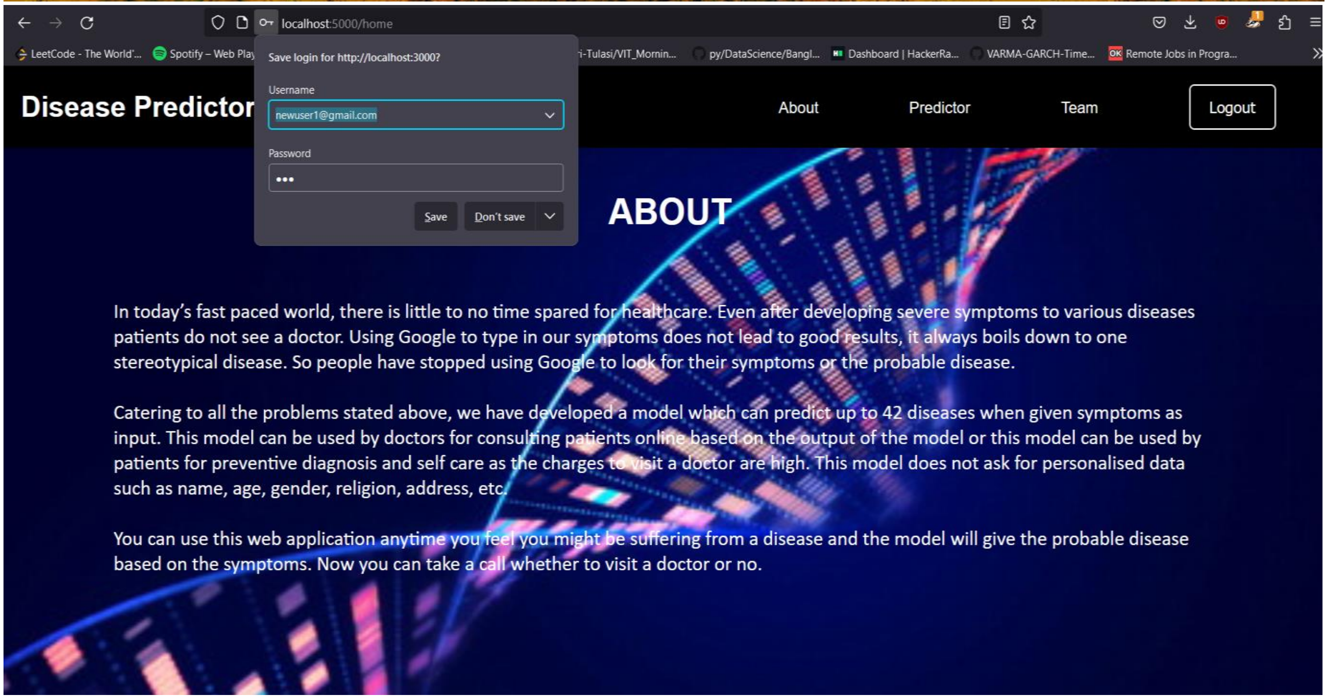
When we click login, we are directed to a login page, if we don't have an account, we can register and then login, like so.





Your registration is complete. Please go to the login page.





In today's fast paced world, there is little to no time spared for healthcare. Even after developing severe symptoms to various diseases patients do not see a doctor. Using Google to type in our symptoms does not lead to good results, it always boils down to one stereotypical disease. So people have stopped using Google to look for their symptoms or the probable disease.

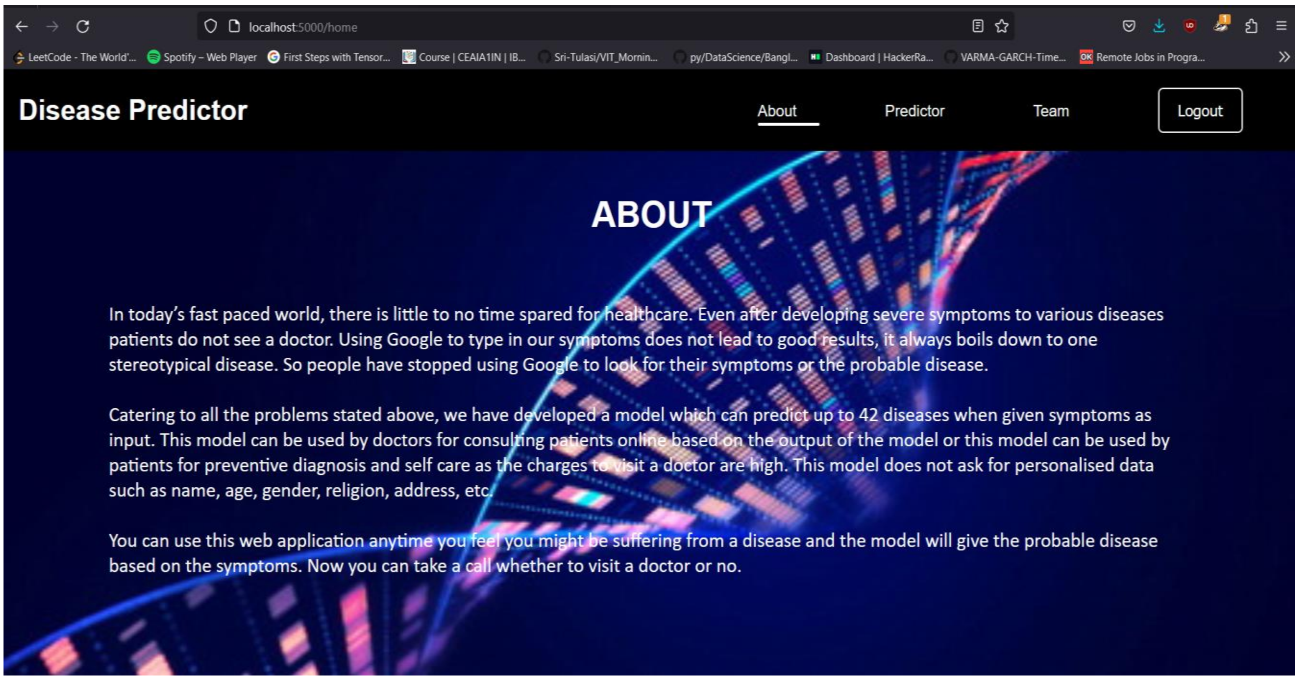
Catering to all the problems stated above, we have developed a model which can predict up to 42 diseases when given symptoms as input. This model can be used by doctors for consulting patients online based on the output of the model or this model can be used by patients for preventive diagnosis and self care as the charges to visit a doctor are high. This model does not ask for personalised data such as name, age, gender, religion, address, etc.

You can use this web application anytime you feel you might be suffering from a disease and the model will give the probable disease based on the symptoms. Now you can take a call whether to visit a doctor or no.

Browser also detects login and asks if it should remember your credentials.

After login you are directed to the About/Home page containing various sections in the header bar such as About, Predict, Team. There is some information given on the web page about our model.

If you click on the About button on home page or in the header bar you will be redirected to the Home.html page.



localhost:5000/home

LeetCode - The World'... Spotify – Web Player First Steps with Tensor... Course | CEAIA1IN | IB... Sri-Tulasi/VIT_Mornin... py/DataScience/Bangl... Dashboard | HackerRa... VARMA-GARCH-Time... Remote Jobs in Progra...

Disease Predictor

About Predictor Team Logout

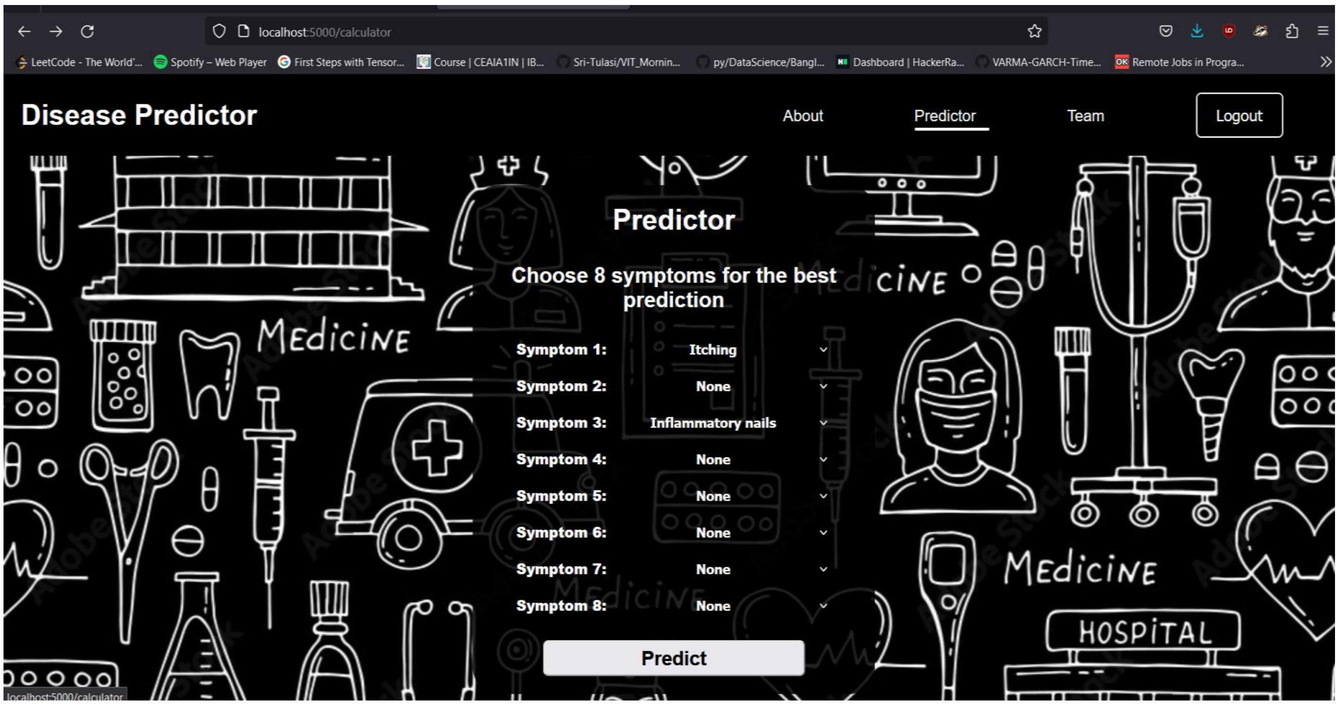
ABOUT

In today's fast paced world, there is little to no time spared for healthcare. Even after developing severe symptoms to various diseases patients do not see a doctor. Using Google to type in our symptoms does not lead to good results, it always boils down to one stereotypical disease. So people have stopped using Google to look for their symptoms or the probable disease.

Catering to all the problems stated above, we have developed a model which can predict up to 42 diseases when given symptoms as input. This model can be used by doctors for consulting patients online based on the output of the model or this model can be used by patients for preventive diagnosis and self care as the charges to visit a doctor are high. This model does not ask for personalised data such as name, age, gender, religion, address, etc.

You can use this web application anytime you feel you might be suffering from a disease and the model will give the probable disease based on the symptoms. Now you can take a call whether to visit a doctor or no.

If you click on the predictor option you will be directed the prediction page where you will be required To enter the symptoms, you are experiencing and you will be given a prediction based on your input. No name, age, gender, etc. necessary!



localhost:5000/calculator

LeetCode - The World'... Spotify – Web Player First Steps with Tensor... Course | CEAIA1IN | IB... Sri-Tulasi/VIT_Mornin... py/DataScience/Bangl... Dashboard | HackerRa... VARMA-GARCH-Time... Remote Jobs in Progra...

Disease Predictor

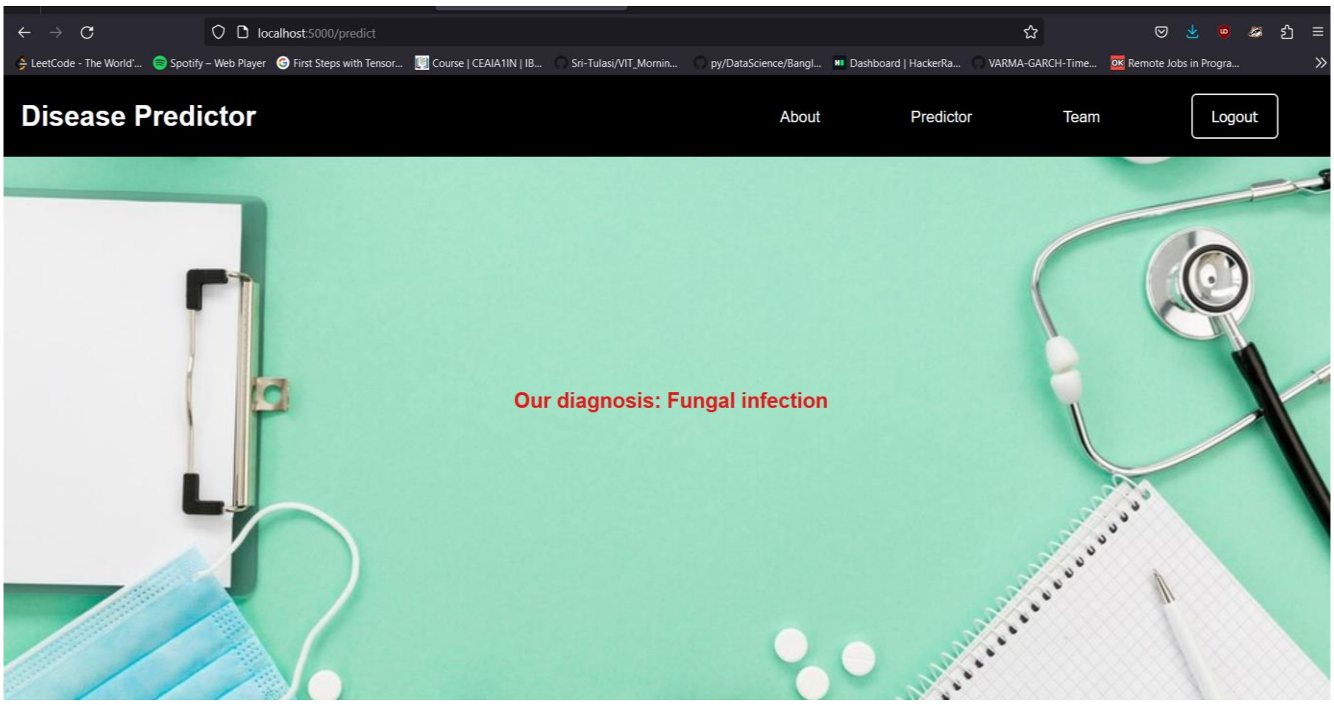
About Predictor Team Logout

Predictor

Choose 8 symptoms for the best prediction

Symptom 1:	Itching
Symptom 2:	None
Symptom 3:	Inflammatory nails
Symptom 4:	None
Symptom 5:	None
Symptom 6:	None
Symptom 7:	None
Symptom 8:	None

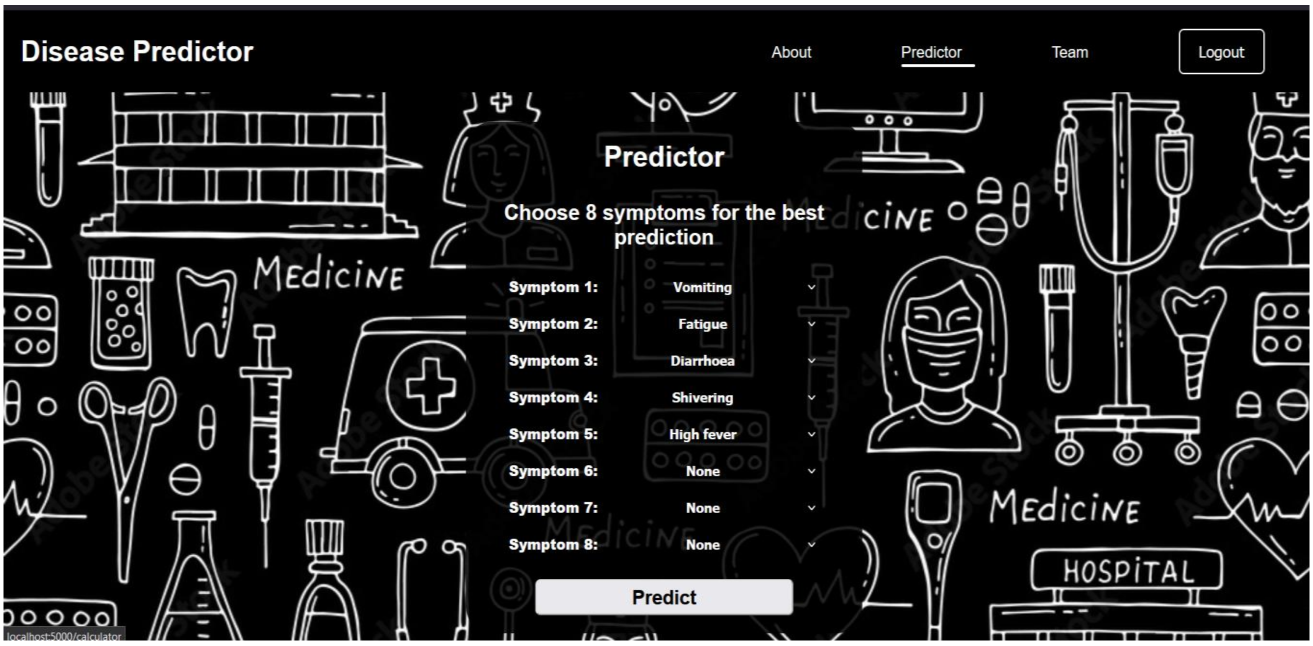
Predict



Our diagnosis: Fungal infection

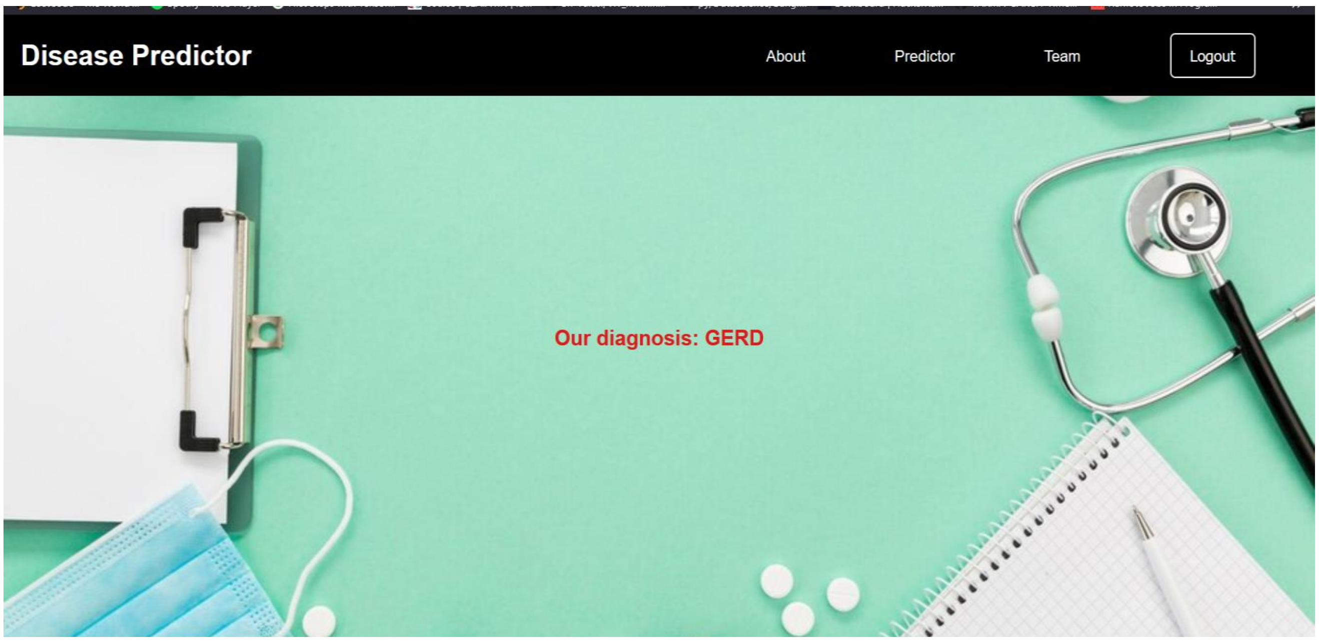
We provide the user with 8 input fields which contain a drop-down list of symptoms to choose from. You can fill all 8 or just 1.

We will input some symptoms such as vomiting, fatigue, diarrhea, shivering, high fever and click on Predict button to get the output.


Disease Predictor
About
Predictor
Team
Logout
Predictor
Choose 8 symptoms for the best prediction
Predict
localhost:5000/calculator

Symptom 1:	Vomiting
Symptom 2:	Fatigue
Symptom 3:	Diarrhoea
Symptom 4:	Shivering
Symptom 5:	High fever
Symptom 6:	None
Symptom 7:	None
Symptom 8:	None

We will be redirected to the Results.html page once we click the Predict button.

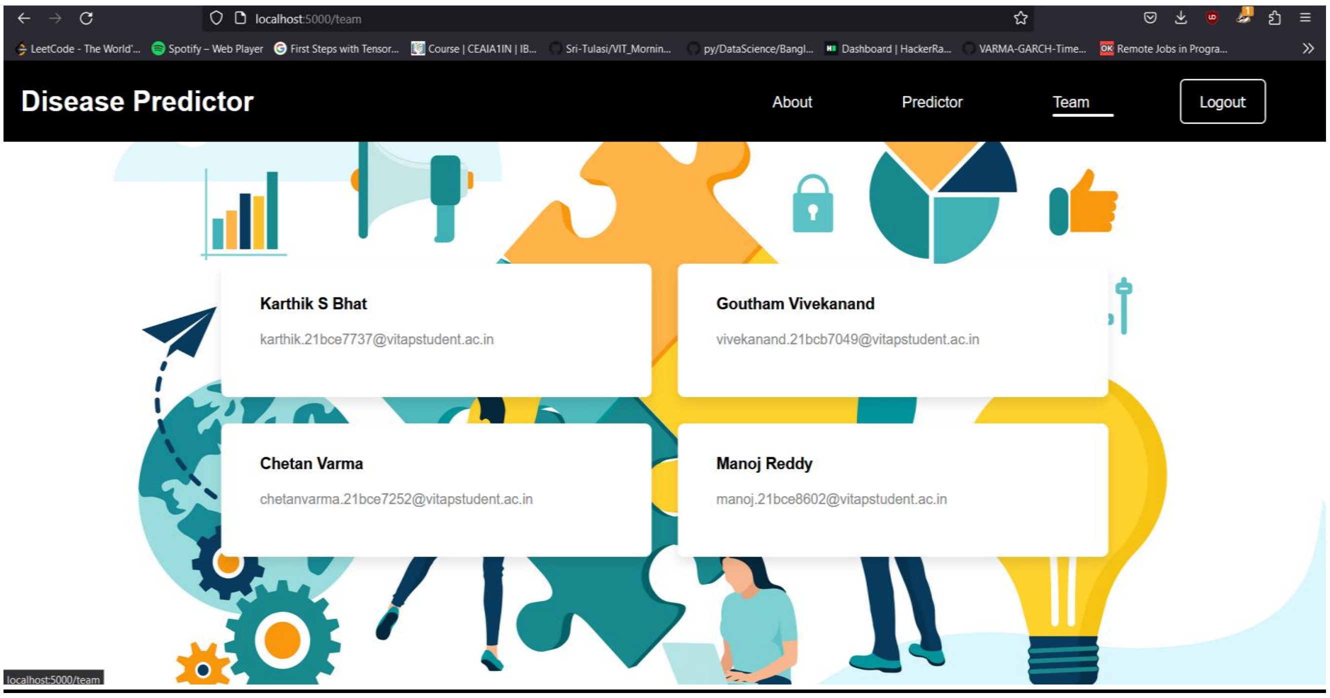


Our diagnosis: GERD

The results say that there are high chances that patient has GERD (Gastroesophageal reflux disease) based on the symptoms received.

After finishing diagnosis, you can input other symptoms, visit our team or logout.

Visiting Team Page



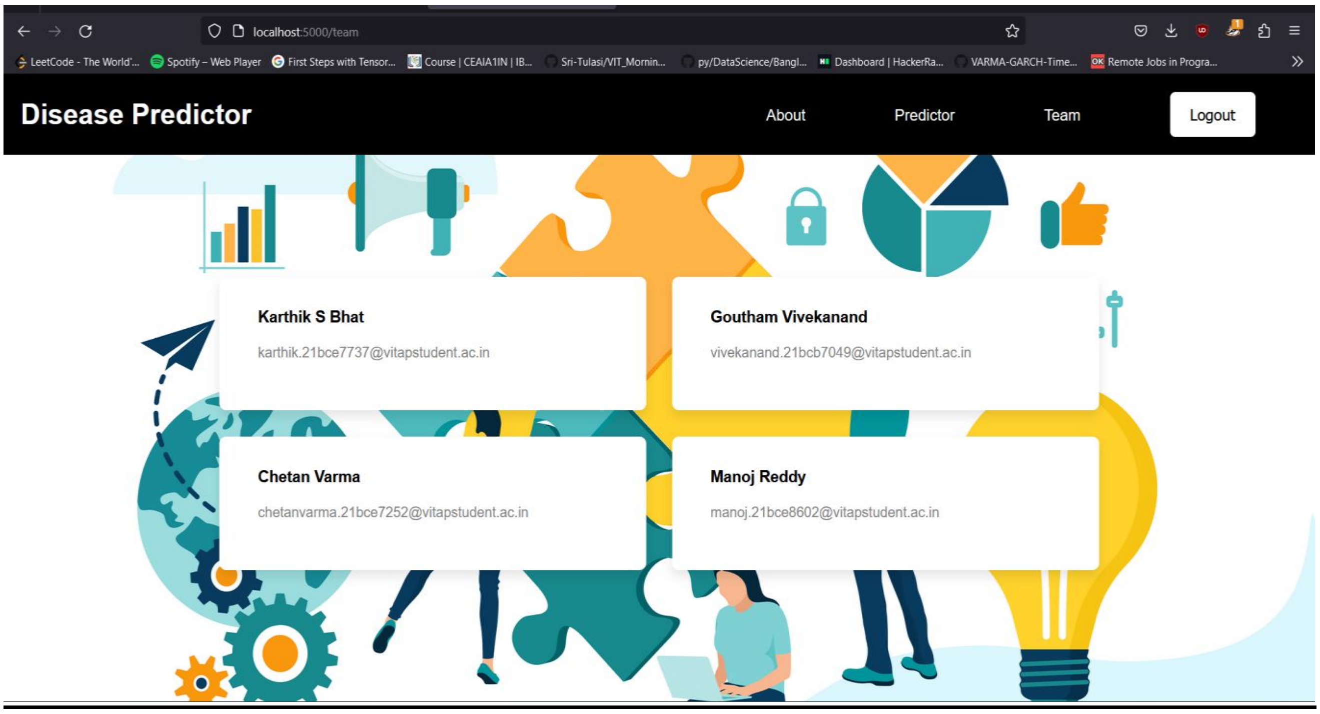
Karthik S Bhat
karthik.21bce7737@vitapstudent.ac.in

Goutham Vivekanand
vivekanand.21bcb7049@vitapstudent.ac.in

Chetan Varma
chetanvarma.21bce7252@vitapstudent.ac.in

Manoj Reddy
manoj.21bce8602@vitapstudent.ac.in

Logging out



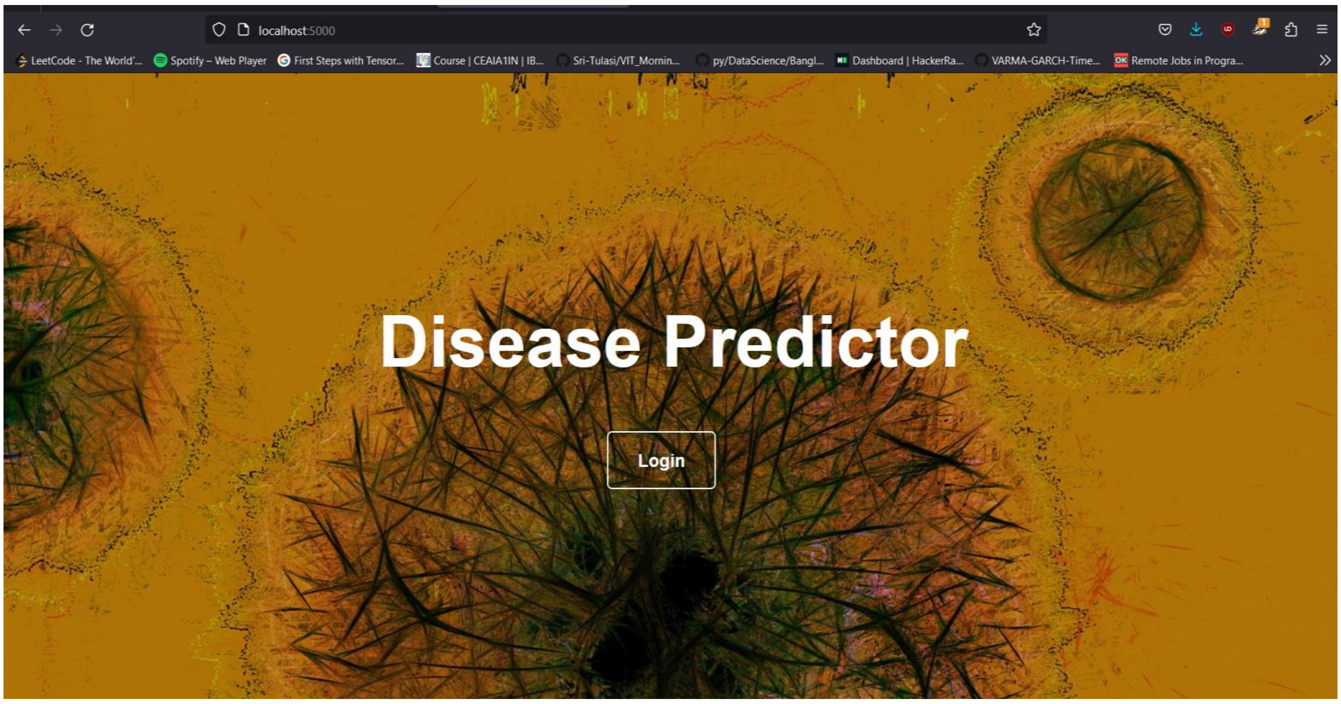
Karthik S Bhat
karthik.21bce7737@vitapstudent.ac.in

Goutham Vivekanand
vivekanand.21bcb7049@vitapstudent.ac.in

Chetan Varma
chetanvarma.21bce7252@vitapstudent.ac.in

Manoj Reddy
manoj.21bce8602@vitapstudent.ac.in

After logging out you will be redirected to our welcome page.



Link to GitHub Repository:

https://github.com/smartinternz02/SI-GuidedProject-612639-1699329275/tree/main/Disease-Predictor