Disease Prediction Using Machine Learning

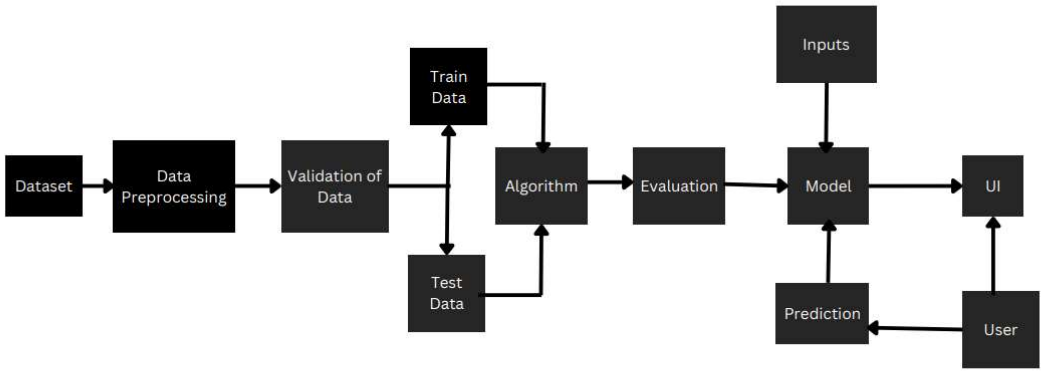
In today's world, there is little to no time spared for healthcare. Even after developing severe symptoms to various diseases patients do not see a doctor. Using Google to type in our symptoms does not lead to good results, it always boils down to one stereotypical disease. So people have stopped using 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 selfcare 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**
  - **o Specify the business problem o Business Requirements**
  - **o Literature Survey o Social or Business Impact**
- **Data Collection and Preparation:**
  - **o Collect the dataset**
  - **o Data Preparation**
- **Exploratory Data Analysis:**
  - **o Descriptive statistical**
  - **o Visual Analysis**
- **Model Building:**
  - **o Creating a function for evaluation**
  - **o Training and testing the Models using multiple algorithms**
- **Performance Testing & Hyperparameter Tuning:**
  - **o Testing model with multiple evaluation metrics**
  - **o Comparing model accuracy before & after applying hyperparameter tuning**

o Comparing model accuracy for different number of features.

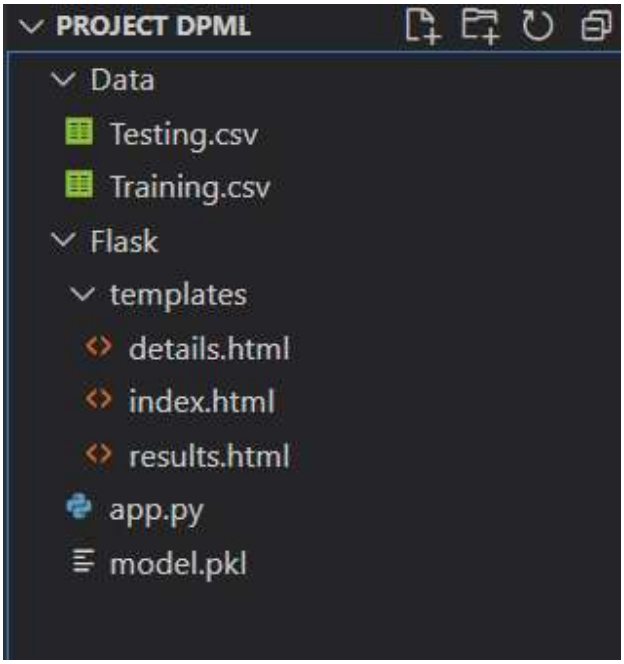
o Building model with appropriate features.

- Model Deployment:

o Save the best model

o Integrate with Web Framework

Project Structure:



- 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 which will further be used in the Flask integration.**

Milestone 1: Define Problem/Problem Understanding

Activity 1: Specify the 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: 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

Machine Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

Activity 1: Collect the Dataset

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

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

As the dataset is downloaded, we need to read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Activity 1.1: Importing the Libraries

```
[1] import numpy as np
    import pandas as pd
```

```
[2] import seaborn as sns
    import matplotlib.pyplot as plt
```

```
[3] from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
    from sklearn.neighbors import KNeighborsClassifier
```

```
[4] from sklearn.svm import SVC
    from sklearn.tree import DecisionTreeClassifier
```

```
[5] from sklearn.ensemble import RandomForestClassifier
    import pickle
```

Activity 1.2: Read the Dataset

Our dataset format is in .csv. We read the dataset with the help of pandas with the help of function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
[6] traind=pd.read_csv('/content/Training.csv')
    traind.head()
```

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

5 rows × 134 columns

```
[7] testd=pd.read_csv('/content/Testing.csv')
    testd.head()
```

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

5 rows × 133 columns

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

```
[8] traind['Unnamed: 133'].value_counts()

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

```
[9] traind.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

```
[10] traind.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
```

```
[11] traind.isnull().sum().sum()

     0
```

There are no missing values in the dataset.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

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.

```
[12] traind.describe()
```

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

8 rows × 132 columns

Activity 2: Visual Analysis

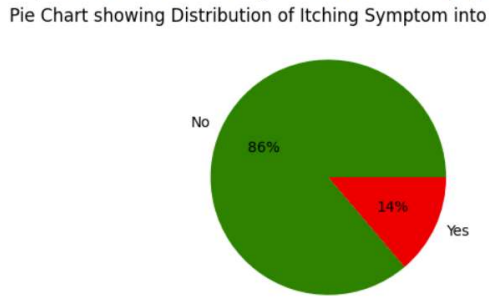
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate Analysis:

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

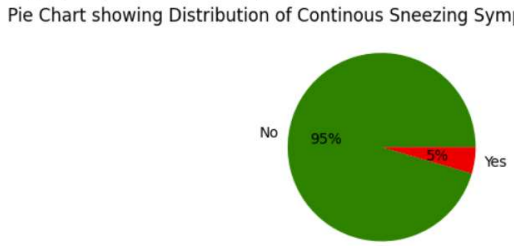
```
[13] plt.figure(figsize=(8,8))
     a=traind['itching'].value_counts()
     plt.subplot(121)
     plt.pie(x=a,data=traind,labels=['No','Yes'],autopct='%.0f%%',colors='gr')
     plt.title("Pie Chart showing Distribution of Itching Symptom into Number of Yes/No")
```

```
Text(0.5, 1.0, 'Pie Chart showing Distribution of Itching Symptom into Number of Yes/No')
```




```
b=traind['continuous_sneezing'].value_counts()
plt.subplot(121)
plt.pie(x=b,data=traind,labels=['No','Yes'],autopct='%.0f%%',colors='gr')
plt.title("Pie Chart showing Distribution of Continous Sneezing Symptom into Number of Yes/No")
```

```
Text(0.5, 1.0, 'Pie Chart showing Distribution of Continous Sneezing Symptom into Number of Yes/No')
```



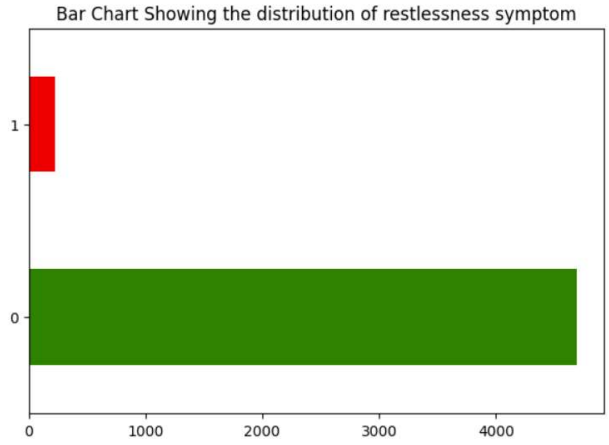
The plt.figure() command is used to determine the size of the plot.

The pie plot on the top shows the different values distribution in the Itching column. It shows that there are 86% observations where the itching symptom has

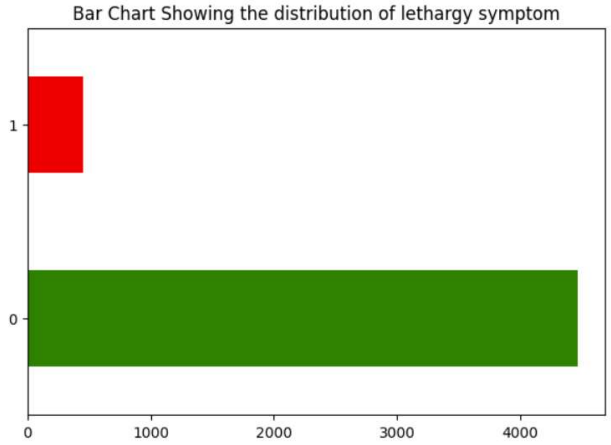
value 0 and there are 14% observations where the itching symptom has value 1.

The pie plot on the bottom shows the different values distribution in the continuous_sneezing column. It shows that there are 95% observations where the continuous_sneezing symptom has value 0 and there are 5% observations where the continuous_sneezing symptom has value 1.

```
[15] plt.subplot(1,2,1)
     traind['restlessness'].value_counts().plot(kind='barh',color=['g','r'])
     plt.title("Bar Chart Showing the distribution of restlessness symptom")
     plt.subplots_adjust(left=0.5,right=2.4)
```



```
[16] plt.subplot(1,2,1)
     traind['lethargy'].value_counts().plot(kind='barh',color=['g','r'])
     plt.title("Bar Chart Showing the distribution of lethargy symptom")
     plt.subplots_adjust(left=0.5,right=2.4)
```



Here we have plotted 2 horizontal bar graphs. These bar graphs can be plotted without using any external 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 that the 0 value has count of around 4500 and the 1 value has count of around 400.

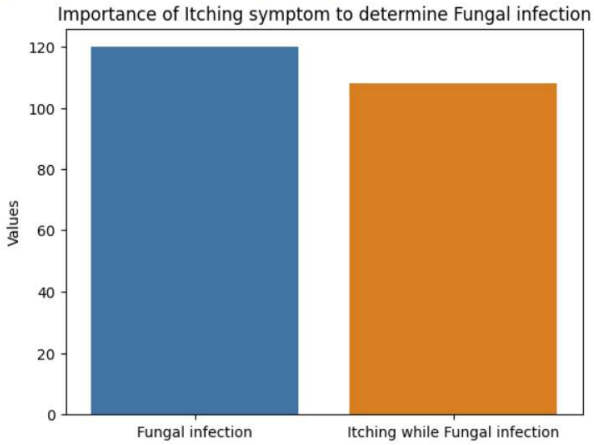
Activity 2.2: Bivariate Analysis:

To find the relation between two features we use bivariate analysis. Here we are visualising the

relationship between prognosis where the values are Fungal Infection and Itching symptom.

```
[17] a=len(traind[traind['prognosis']=='Fungal infection'])
     b=len(traind[(traind['itching']==1) & (traind['prognosis']=='Fungal infection')])
     fi= pd.DataFrame(data=[a,b],columns=['Values'],index=['Fungal infection','Itching while Fungal infection'])
     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')
```

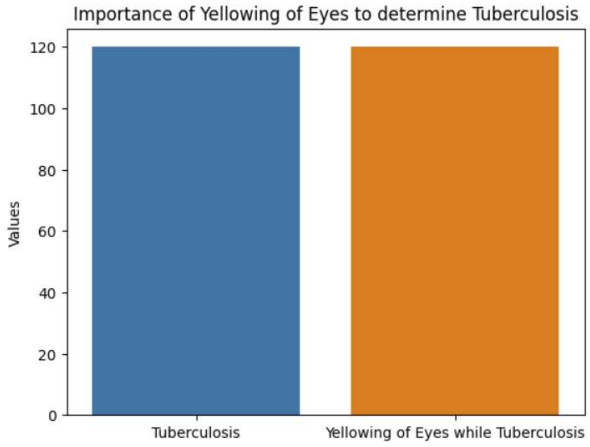


Here we use Boolean Indexing to filter out values from the prognosis column where the values are 'Fungal Infection'. These observations are stored in variable 'a'. Alse we filter out values from the prognosis where values are 'Fungal Infection' and also the values of Itching variable are 1. From the plot we can see that when there is Fungal Infection there is a high chance that the Itching column has value 1. There are 120 values where the value are fungal infection and there are 104 values where the value of itching column is 1. This shows that when there is a fungal infection there is a high chance that there is itching as a symptom.

We have also seen the relationship between prognosis when the disease is Tuberculosis and the symptom yellowing_of_eyes. . From the plot we can see that when there is Tuberculosis there is a high chance that the yellowing of eyes column has value 1. There are 120 values where the value is Tuberculosis and there are 119 values where the value of yellowing_of_eyes column is 1. This shows that when there is tuberculosis there is a high chance that there is yellowing_of_eyes as a symptom.

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

```
Text(0.5, 1.0, 'Importance of Yellowing of Eyes to determine Tuberculosis')
```



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.

```
[19] corr=traind.corr()
     corr.style.background_gradient('coolwarm')
```

```
<ipython-input-19-e37d892fa4b4>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it w
  corr=traind.corr()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/style.py:3931: RuntimeWarning: All-NaN slice encountered
  smin = np.nanmin(gmap) if vmin is None else vmin
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/style.py:3932: RuntimeWarning: All-NaN slice encountered
  smax = np.nanmax(gmap) if vmax is None else vmax
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	
itching	1.000000	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.160650	0.202850	-0
skin_rash	0.318158	1.000000	0.298143	-0.094786	-0.065324	-0.029324	0.171134	0.161784	-0
nodal_skin_eruptions	0.326439	0.298143	1.000000	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0
continuous_sneezing	-0.086906	-0.094786	-0.032566	1.000000	0.608981	0.446238	-0.087351	-0.047254	-0
shivering	-0.059893	-0.065324	-0.022444	0.608981	1.000000	0.295332	-0.060200	-0.032566	-0
chills	-0.175905	-0.029324	-0.065917	0.446238	0.295332	1.000000	-0.004688	-0.095646	-0
joint_pain	-0.160650	0.171134	-0.060200	-0.087351	-0.060200	-0.004688	1.000000	-0.087351	-0
stomach_pain	0.202850	0.161784	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	1.000000	0
acidity	-0.086906	-0.094786	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	0.433917	1
ulcers_on_tongue	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.649078	0
muscle_wasting	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0
vomiting	-0.057763	-0.225046	-0.119543	-0.173459	-0.119543	0.144263	0.199921	0.031406	0

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.

```
[20] traind.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)
```

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.

```
[21] def data_preprocessing(data):
       data.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)
       return data
```

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

```
[22] testd=data_preprocessing(testd)
```

Here we call the function for the test data.

Activity 2.5: Split the 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.

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

We split the test data into features(x_test) and the corresponding target variables(y_test) and split the training data into training and validation data. We have kept 80 % data for training and 20% is used for validation.

```
[23] x=traind.drop('prognosis',axis=1)
     y=traind.prognosis
     x_test=testd.drop('prognosis',axis=1)
     y_test=testd.prognosis
     x_train,x_val,y_train,y_val=train_test_split(x,y,test_size=0.2)
```

Milestone 4: Model Building

Activity 1: Creating a function for model evaluation

We will be creating multiple models and then testing them. It will reduce our monotonous task if we directly write a function for model evaluation.

```
[24] 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 shows 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

A variable is created with name knn which has KNeighborsClassifier() algorithm initialised in it. The knn model is trained using the .fit() function. The model is trained on the x_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
[25] knn=KNeighborsClassifier(n_neighbors=7)
     knn.fit(x_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
[26] knn_results=model_evaluation(knn)
```

```
The Training Accuracy of the algorithm is 1.0
The Validation Accuracy of the algorithm is 1.0
The Testing Accuracy of the algorithm is 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 hyperparameter tuning. The results are stored in a variable named knn_results.

Activity 2.2: SVM Model

A variable is created with name svm which has SVC() algorithm initialised in it. The svm model is trained using the .fit() function. The model is trained on the x_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
[27] svm=SVC(C=1)
     svm.fit(x_train,y_train)
```

```
SVC(C=1)
```

```
[28] svm_results=model_evaluation(svm)
```

```
The Training Accuracy of the algorithm is 1.0
The Validation Accuracy of the algorithm is 1.0
The Testing Accuracy of the algorithm is 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 hyperparameter tuning. The results are stored in a variable named svm_results.

Activity 2.3: Decision Tree Model

A variable is created with name dtc which has DecisionTreeClassifier() algorithm initialised in it with a parameter max_features set to 10. The dtc model is

trained using the .fit() function. The model is trained on the x_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
[29] dtc=DecisionTreeClassifier(max_features=10)
     dtc.fit(x_train,y_train)
```

```
DecisionTreeClassifier(max_features=10)
```

```
[30] dtc_results=model_evaluation(dtc)
```

```
The Training Accuracy of the algorithm is 1.0
The Validation Accuracy of the algorithm is 1.0
The Testing Accuracy of the algorithm is 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 are high, 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 Bagging model which utilises multiple decision trees and takes their aggregate to give a prediction. A variable is created with name rfc which has RandomForestClassifier() algorithm initialised in it with a parameter max_depth set to 13. The rfc model is trained using the .fit() function. The model is trained on the x_train and y_train data that is the training features and training

target variables. This model is then given to the model_evaluation function to check its performance.

```
[31] rfc=RandomForestClassifier(max_depth=13)
     rfc.fit(x_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=13)
```

```
[32] rfc_results=model_evaluation(rfc)
```

```
The Training Accuracy of the algorithm is 1.0
The Validation Accuracy of the algorithm is 1.0
The Testing Accuracy of the algorithm is 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 are high, there is no need for hyperparameter tuning. The results are stored in a variable named rfc_results.

Milestone 5: Performance Testing and 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.

```
[33] results=pd.DataFrame(data=[knn_results,svm_results,dtc_results,rfc_results],
                          columns=['Training Accuracy','Validation Accuracy','Testing Accuracy'],
                          index=['K Nearest Neighbors Classfier','Support Vector Machines',
                                 'Decision Trees Classfier','Random Forest Classfier'])
     results
```

	Training Accuracy	Validation Accuracy	Testing Accuracy
K Nearest Neighbors Classfier	1.0	1.0	1.00000
Support Vector Machines	1.0	1.0	1.00000
Decision Trees Classfier	1.0	1.0	0.97619
Random Forest Classfier	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 aren't applying the hyperparameter tuning

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.

```
[34] a=rfc.feature_importances_
     col=x.columns
     feat_imp={}
     for i,j in zip(a,col):
       feat_imp[j]=i
     feat_imp
```

```
'back_pain': 0.015828616220892464,
'constipation': 0.008702320037466538,
'abdominal_pain': 0.013266296439988072,
'diarrhoea': 0.006143911187138672,
'mild_fever': 0.01827480269032994,
'yellowing_of_eyes': 0.014813690696650728,
'fluid_overload': 0.0,
'swelled_lymph_nodes': 0.008913232690224615,
'malaise': 0.010924358665339893,
'blurred_and_distorted_vision': 0.011624488726916385,
'phlegm': 0.01177301262215603,
'congestion': 0.01649100518575724,
'chest_pain': 0.014718449500033842,
'weakness_in_limbs': 0.009365301405656959,
'fast_heart_rate': 0.008702356162783555,
'irritation_in_anus': 0.02141148228039585,
'neck_pain': 0.015362707112067364,
'dizziness': 0.013579156907297685,
'obesity': 0.011268323337894341,
'swollen_legs': 0.014188129636759817,
'puffy_face_and_eyes': 0.00879460368699282,
'excessive_hunger': 0.011360187518931685,
'extra_marital_contacts': 0.004420661394733149,
'knee_pain': 0.01138463423603362,
'muscle_weakness': 0.01014864691931197,
'stiff_neck': 0.010908019446195764,
```

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.

```
[35] rfc_results=[]
     knn_results=[]
```

```
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

    def model_evaluation1(num_features, classifier, x_val, y_val):
      y_train_pred = classifier.predict(x1_train)
      train_accuracy = accuracy_score(y1_train, y_train_pred)
      y_val_pred = classifier.predict(x_val)
      test_accuracy = accuracy_score(y_val, y_val_pred)

      return train_accuracy, test_accuracy
    rfc_new = RandomForestClassifier()
    rfc_new.fit(x1_train, y1_train)
    temp1 = model_evaluation1(x1_train.shape[1], rfc_new, x1_val, y1_val)
    rfc_results.append([x1_train.shape[1], temp1[0], temp1[1]])
```

```
    knn_new = KNeighborsClassifier()
    knn_new.fit(x1_train, y1_train)
    temp2 = model_evaluation1(x1_train.shape[1], knn_new, x1_val, y1_val)
    knn_results.append([x1_train.shape[1], temp2[0], temp2[1]])
```

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 the to_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','Training Accuracy','Testing Accuracy'])
randomf
```

	Number of features	Training Accuracy	Testing Accuracy
0	6	0.171748	0.142276
1	8	0.244919	0.203252
2	15	0.459350	0.418699
3	24	0.646850	0.619919
4	37	0.865600	0.848577
5	48	0.933943	0.928862
6	62	0.983740	0.985772

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.

```
[38] knn_table=pd.DataFrame(data=knn_results,columns=['Number of features','Training Accuracy','Testing Accuracy'])
knn_table
```

	Number of features	Training Accuracy	Testing Accuracy
0	6	0.166921	0.161585
1	8	0.237805	0.237805
2	15	0.449441	0.452236
3	24	0.644563	0.622967
4	37	0.861535	0.858740
5	48	0.930894	0.928862
6	62	0.983740	0.985772

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.

```
[39] len(to_drop)
     x_new=x.drop(to_drop,axis=1)
     y_new=y
     x_new.head()
```

	itching	shivering	joint_pain	stomach_pain	acidity	vomiting	fatigue	weight_loss	lethargy	high_fever	...	family_history	rusty_sputum
0	1	0	0	0	0	0	0	0	0	0	...	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0
4	1	0	0	0	0	0	0	0	0	0	...	0	0

5 rows × 62 columns

We drop 44 features which creates 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 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. Previously for 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.

```
[40] 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)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
[41] 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_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))
```

```
The Training Accuracy of the algorithm is 0.023373983739837397
The Validation Accuracy of the algorithm is 0.027439024390243903
The Testing Accuracy of the algorithm is 1.0
```

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

```
[42] knn_new=KNeighborsClassifier()
     knn_new.fit(x1_train,y1_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
[43] 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_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))
```

```
The Training Accuracy of the algorithm is 0.023373983739837397
The Validation Accuracy of the algorithm is 0.027439024390243903
The Testing Accuracy of the algorithm is 1.0
```

After training the knn model we check the accuracies. Our model has achieved 100 % accuracy for the test data.

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

```
[44] testd.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
11	Migraine	Migraine
12	Cervical spondylosis	Cervical spondylosis
13	Paralysis (brain hemorrhage)	Paralysis (brain hemorrhage)
14	Jaundice	Jaundice
15	Malaria	Malaria

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

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future. After checking the performance, we decide to save the knn model built with 45 features.

```
[45] 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:

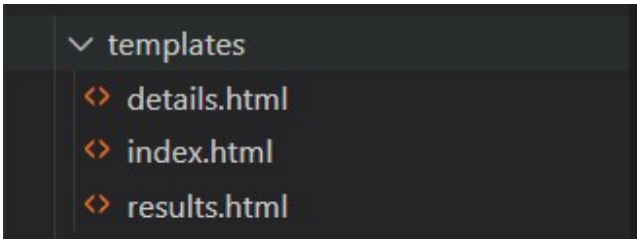
- **Building HTML Pages**
- **Building server-side script**
- **Run the web application**

Activity 2.1: Building HTML pages:

For this project we create three HTML files namely

- **Index.html**
- **Details.html**
- **Results.html**

And we will save them in the templates folder



Activity 2.2: Build Python code

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

Import the necessary Libraries.

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

This code first loads the saved Linear Regression model from the "bodyfat.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened 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.

```
model = pickle.load(open('model.pkl','rb'))
app = Flask(__name__)
```

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.

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

The route in this case is "/details". When a user accesses the "/predict" route of the website, this function is “index()” called. The "render_template()" method is used to render an HTML template named "details.html".

```
@app.route('/details')
def pred():
    return render_template('details.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','GET'])
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]
    return render_template('results.html', prediction_text="The probable diagnosis says it could be {}".format(prediction))
```

Main Function:

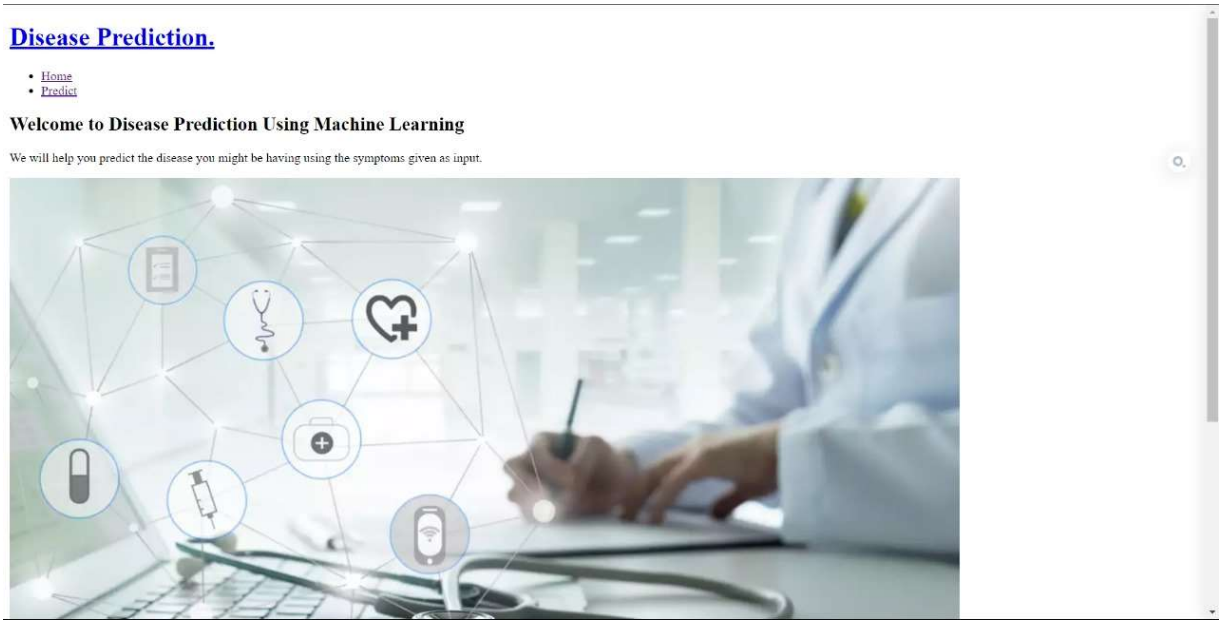
This code sets the entry point of the Flask application. The function “app.run()” is called, which starts the Flask deployment server.

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

Activity 2.3: Run the Web Application

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

When we paste the URL in a web browser, our index.html page will open. It contains various sections in the header bar.



Detials.html shown below

Disease Prediction

You will have the input box below where you can select your symptoms.

You can input the number of symptoms you have and leave others blank.

This is the list of the symptoms . If you have symptoms which are from this list please enter the symptom in the same form as shown below.

- itching
- muscle_pain
- shivering
- joint_pain
- stomach_pain
- vomiting
- fatigue
- weight_loss
- restlessness
- lethargy
- high_fever
- headache
- dark_urine
- nausea
- coma
- constipation
- abdominal_pain
- diarrhoea
- mild_fever
- malaise
- phlegm
- congestion
- chest_pain
- fast_heart_rate
- neck_pain
- dizziness
- belly_pain
- knee_pain
- muscle_weakness
- passage_of_gases
- irritability
- continuous_sneezing
- puffy_face_and_eyes
- abnormal_menstruation
- increased_appetite
- lack_of_concentration
- visual_disturbances
- receiving_blood_transfusion
- pain_behind_the_eyes
- history_of_alcohol_consumption
- blood_in_sputum
- yellowing_of_eyespalpitations
- inflammatory_nails
- yellow_crust_ooze

Symptom-1
itching
Symptom-2
skin_rash
Symptom-3
Type your symptom here
Symptom-4
Type your symptom here
Symptom-5
Type your symptom here
Symptom-6
Type your symptom here
Symptom-7
Type your symptom here
Symptom-8
Type your symptom here
Symptom-9
Type your symptom here

Predict

>

Disease Prediction

Preventive Diagnosis at your convenience.

We are provided with 9 input fields where we can input our symptoms. We can fill all 9 boxes or can just fill 1. We are provided with a list of symptoms in the form of bullet points above. We can input symptoms from above but they have to be in the same form as given in the list since they are the column names. After filling the boxes and click on Predict button, we get the output. We will be redirected to the Results.html page once we click the Predict button.

Disease Prediction.

- Home
- Predict

The probable diagnosis says it could be Fungal infection