Disease Prediction Using Machine Learning

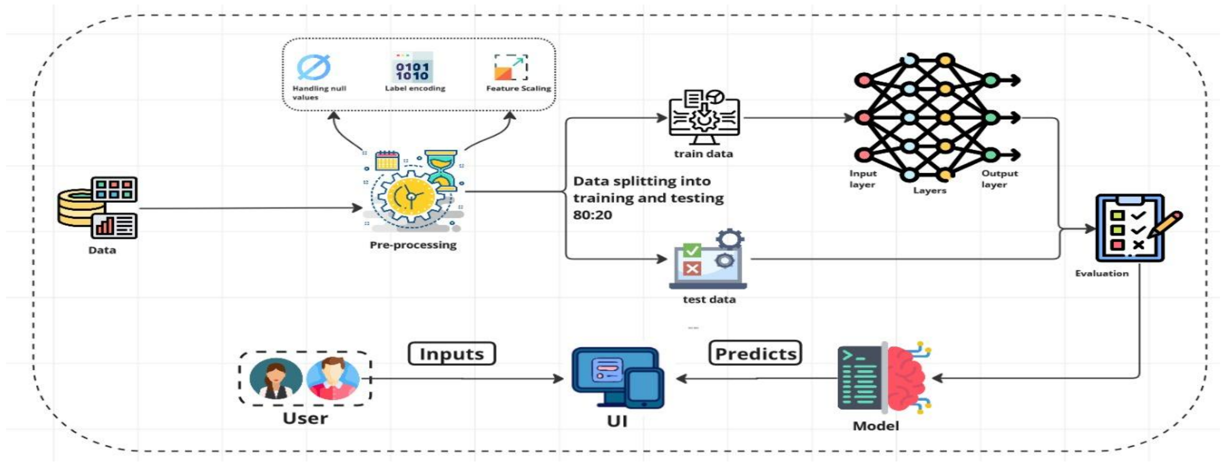
Introduction

In today's fast-paced world, where time is a precious commodity, prioritizing our health often takes a backseat. Many individuals tend to ignore or delay seeking medical attention, even when they experience severe symptoms of various diseases. Resorting to a simple Google search for symptom-checking often leads to generic or stereotypical results, leaving people frustrated and uncertain.Addressing these challenges, we've introduced a groundbreaking solution – a predictive model capable of identifying up to 42 different diseases when provided with a list of symptoms. This innovative model can be a valuable tool for both healthcare professionals and individuals.

For medical professionals, our model can serve as a powerful diagnostic aid, enabling them to consult with patients remotely based on the model's predictions. This is particularly beneficial for telemedicine services and online consultations. For individuals, this model offers a preventive and self-care solution. The high costs associated with visiting a doctor, coupled with the inconvenience of making appointments, can deter many from seeking medical advice. Our model, however, empowers individuals to make informed decisions about their health without the need to share personal data such as their name, age, gender, or address.

Feel free to use our user-friendly web application whenever you suspect you might be experiencing a health issue. Our model will provide you with a list of probable diseases based on the symptoms you provide and you can make an informed decision about whether it's necessary to consult a healthcare professional.Our goal is to make healthcare more accessible, efficient, and user-centric in today's fast-paced world.

Solution Architecture



Project Flow:

- ➔ User is shown the Home page. The user will browse through the 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.
- ➔ Users will be redirected to the Results page. The model will analyze 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
    1. Business Requirements
    2. Literature Survey
    3. Social or Business Impact

- ➔ Data Collection and Preparation o Collect the dataset
    1. Data Preparation

- ➔ Exploratory Data Analysis
    1. Descriptive statistical
    2. Visual Analysis

- ➔ Model Building
    1. Creating a function for evaluation
    2. Training and testing the Models using multiple algorithms

- ➔ Performance Testing & Hyperparameter Tuning
    1. Testing model with multiple evaluation metrics
    2. Comparing model accuracy before & after applying hyperparameter tuning
    3. Comparing model accuracy for different number of features.
    4. Building model with appropriate features.

- ➔ Model Deployment
    1. Save the best model
    2. Integrate with Web Framework

Prior Knowledge:

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

➔ ML Concepts:

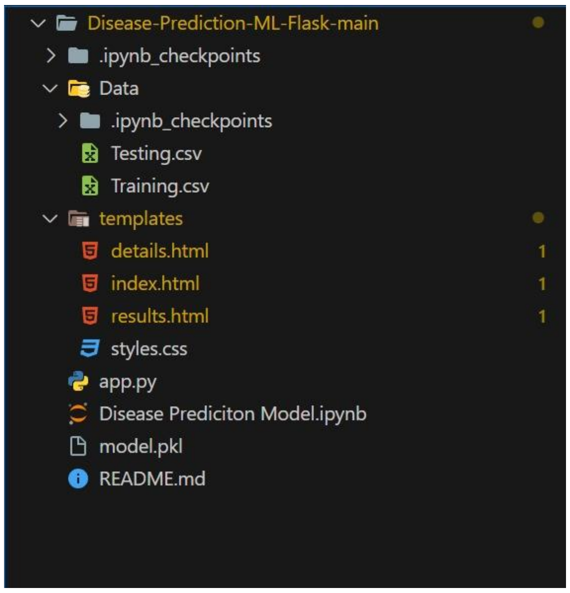
1. Supervised learning: https://www.javatpoint.com/supervised-machine-learning
2. K Nearest Neighbours: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
3. SVM: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
4. Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classificationalgorithm
5. Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm

➔ Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

➔ Frontend Web development: https://www.w3schools.com/html/html_css.asp

Project Structure:

Create project folder which contains files as shown below:



```
Disease-Prediction-ML-Flask-main
  .ipynb_checkpoints
  Data
    .ipynb_checkpoints
    Testing.csv
    Training.csv
  templates
    details.html          1
    index.html            1
    results.html          1
    styles.css
  app.py
  Disease Prediciton Model.ipynb
  model.pkl
  README.md
```

1. The data obtained is in two csv files, one for training and another for testing.

2. We are building a Flask application which will require the html files to be stored in the templates folder.
3. The css and html files should be stored in the static and template folder.
4. App.py file is used for routing purposes using scripting.
5. Model.pkl is the saved model. This will further be used in the Flask integration.
6. Training folder contains a model training file.

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 analyze 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 analyze large datasets and identify patterns and risk factors associated with diseases. By analyzing 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:

1. 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.
2. 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.
3. Compliance:

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

4. 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

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

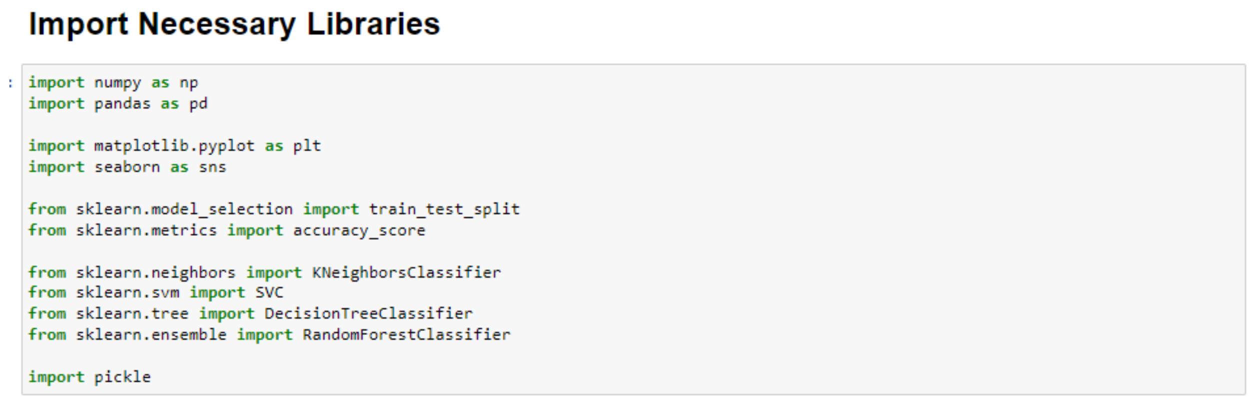
In this project we have used .csv data.

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.a: Importing the Libraries

Import the necessary libraries as shown in the image. Some of them are optional and can be skipped according to your usage.



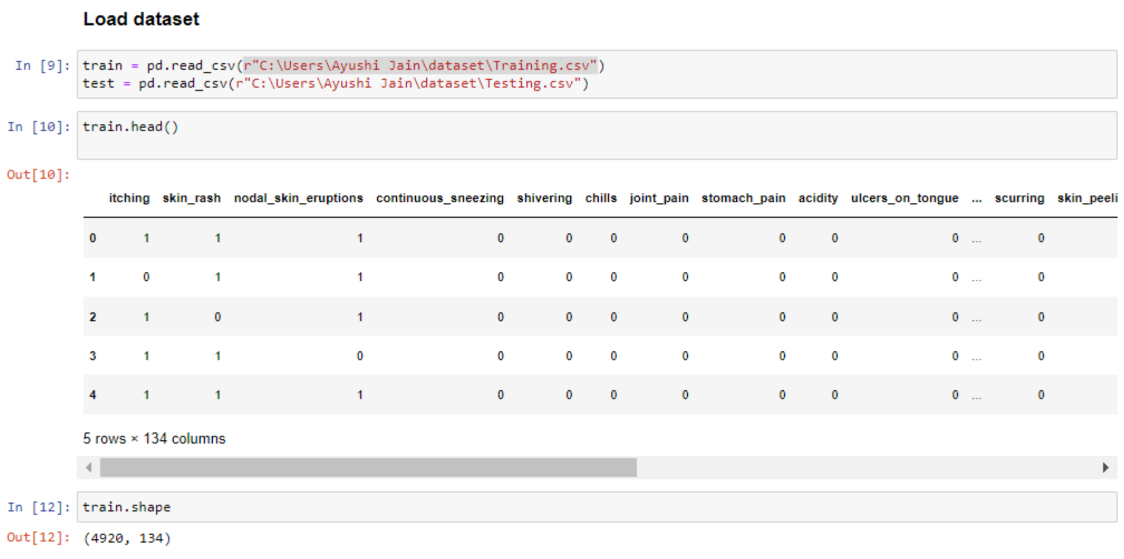
Import Necessary Libraries

```
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.b: Reading the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.
As we have two datasets, one for training and other for testing we will import both the csv files.



Load dataset

```
In [9]: train = pd.read_csv(r"C:\Users\Ayushi Jain\dataset\Training.csv")
        test = pd.read_csv(r"C:\Users\Ayushi Jain\dataset\Testing.csv")

In [10]: train.head()

Out[10]:
```

	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

```
In [12]: train.shape

Out[12]: (4920, 134)
```

Activity 2: Data Preparation

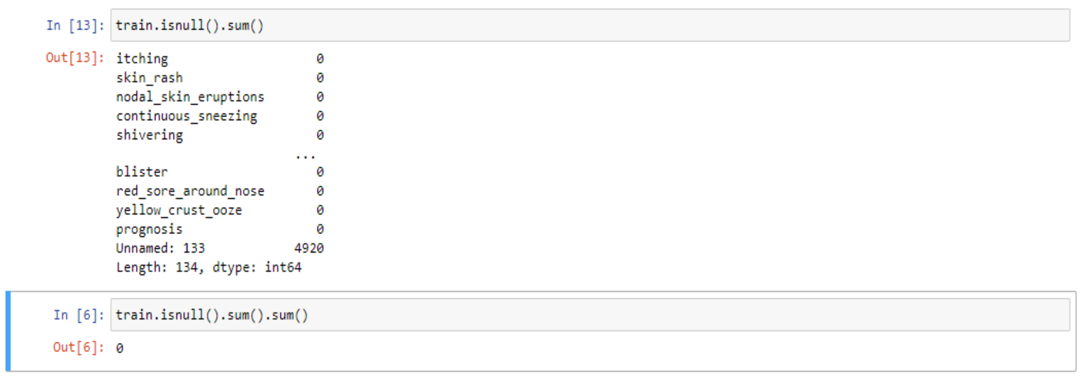
As we have understood how the data is, let us preprocess the collected data.

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: Handling Missing Values

There are no missing values in the dataset.

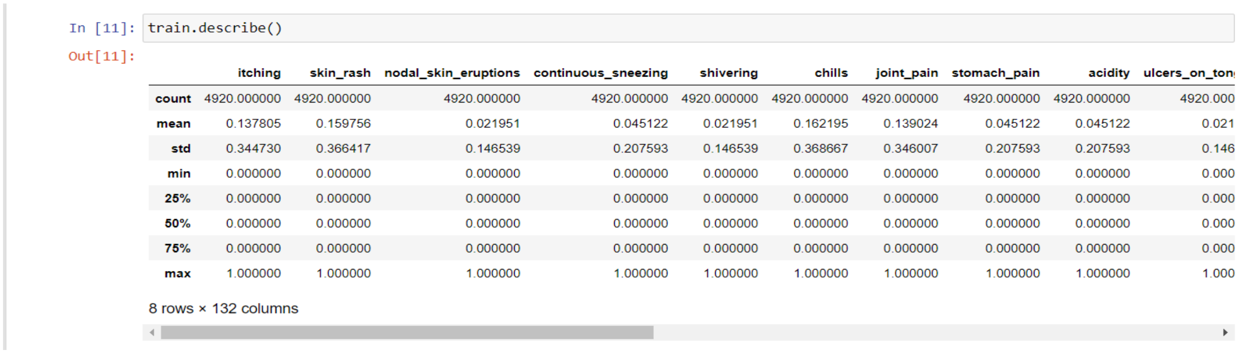


```
In [13]: train.isnull().sum()

Out[13]: itching                    0
         skin_rash                  0
         nodal_skin_eruptions       0
         continuous_sneezing        0
         shivering                  0
                                 ...
         blister                    0
         red_sore_around_nose       0
         yellow_crust_ooze          0
         prognosis                  0
         Unnamed: 133            4920
         Length: 134, dtype: int64

In [6]: train.isnull().sum().sum()

Out[6]: 0
```

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

The goal of descriptive analysis is to use statistics to examine the fundamental characteristics of data. Pandas has a useful function called describe in this case. We can comprehend the distinct, highest, and most common values of categorical features with the help of this describe function. Additionally, we may obtain the mean, standard deviation, minimum, values for the continuous features' maximum and percentiles.



```
In [11]: train.describe()
Out[11]:
```

	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

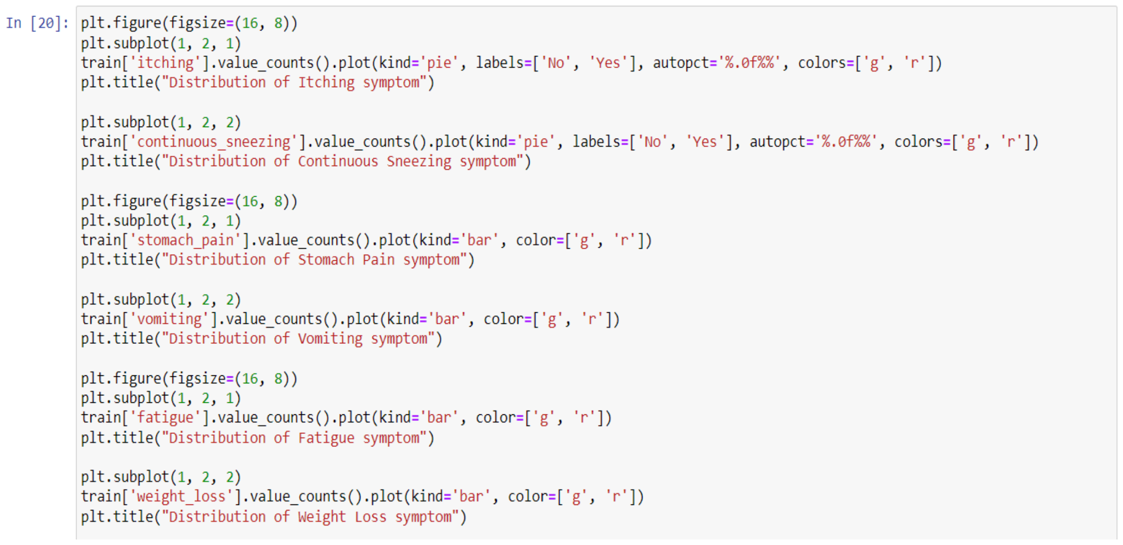
8 rows × 132 columns

Activity 2: Visual Analysis

Using visual aids to examine and comprehend data, such graphs, charts, and plots, is known as visual analysis. It is a method for rapidly seeing trends, patterns, and outliers in the data, which can be useful for gaining understanding and coming to wise conclusions.

Activity 2.1: Univariate Analysis

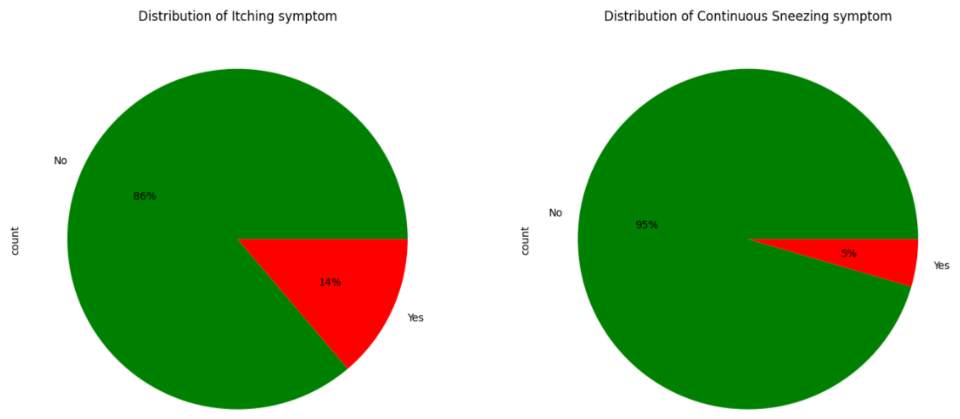
To put it simply, univariate analysis involves using only one feature to comprehend the data. Three distinct kinds of plots and graphs have been shown.
It is possible to utilise the matplotlib.pyplot package for basic visualisations.



```
In [20]: plt.figure(figsize=(16, 8))
         plt.subplot(1, 2, 1)
         train['itching'].value_counts().plot(kind='pie', labels=['No', 'Yes'], autopct='%.0f%%', colors=['g', 'r'])
         plt.title("Distribution of Itching symptom")

         plt.subplot(1, 2, 2)
         train['continuous_sneezing'].value_counts().plot(kind='pie', labels=['No', 'Yes'], autopct='%.0f%%', colors=['g', 'r'])
         plt.title("Distribution of Continuous Sneezing symptom")

         plt.figure(figsize=(16, 8))
         plt.subplot(1, 2, 1)
         train['stomach_pain'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Stomach Pain symptom")

         plt.subplot(1, 2, 2)
         train['vomiting'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Vomiting symptom")

         plt.figure(figsize=(16, 8))
         plt.subplot(1, 2, 1)
         train['fatigue'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Fatigue symptom")

         plt.subplot(1, 2, 2)
         train['weight_loss'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Weight Loss symptom")
```

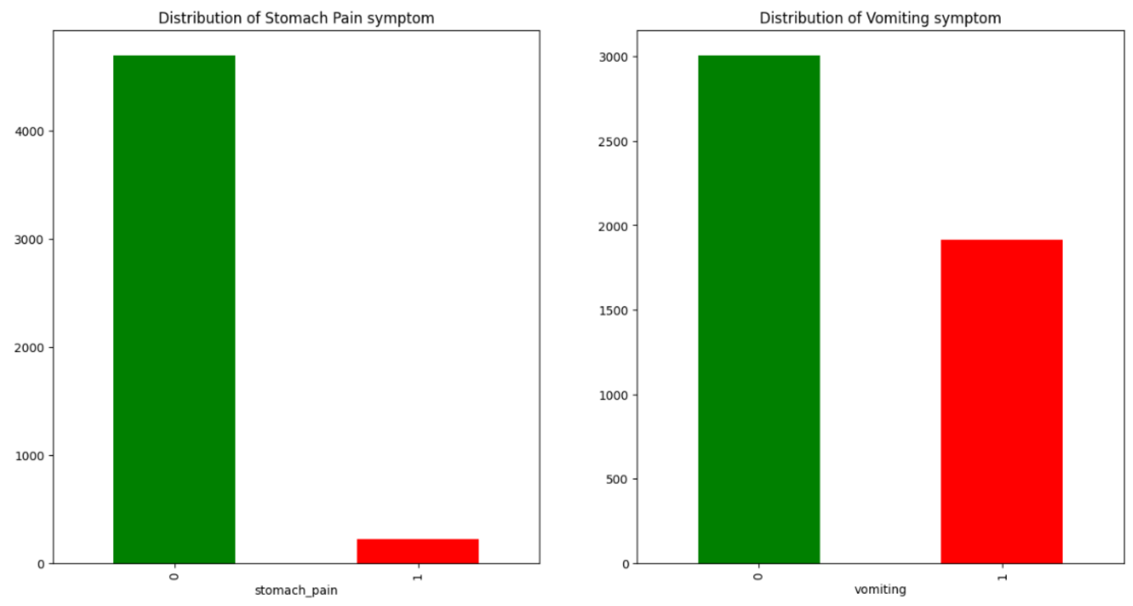
The plot's size is here determined by the plt.figure() command. Next, we split the area into two pie charts.
The various value distributions in the Itching column are displayed in the pie plot on the left. It demonstrates that in 86% of observations, the itching symptom has a value of 0, while in 14% of data, the value is 1.
The continuous_sneezing column's various value distributions are displayed in the pie plot on the right. The data indicates that 95% of the observations have a continuous_sneezing symptom value of 0, and 5% have a continuous_sneezing symptom value of 1.

```
Out[20]: Text(0.5, 1.0, 'Distribution of Headache symptom')
```



Two bar graphs are presented here. Plotting these bar graphs doesn't require the use of any outside libraries. Using the subplot function from the matplotlib.pyplot library, we split the plot into two subplots. We used Python's built-in plot function to plot the bar graph.
The distribution of data for symptoms of stomach pain is displayed in the bar graph on the left. It is evident that the 1 value has a count of approximately 400, while the 0 value has a count of approximately 4700.

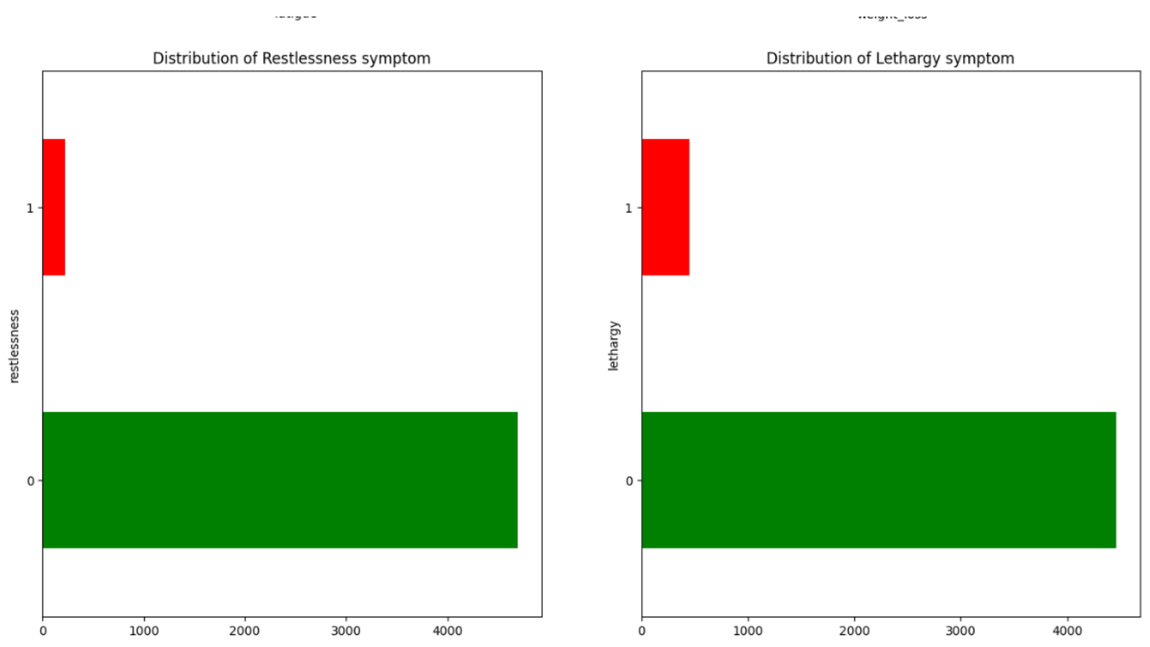
The distribution of vomiting symptom values is displayed on the graph to the right. It is evident that the 1 value has a count of approximately 2000, whereas the 0 value has a count of approximately 3000.



Here, two horizontal bar graphs have been drawn. Plotting these bar graphs doesn't require the use of any additional libraries.

Using the subplot function from the matplotlib.pyplot library, we split the plot into two subplots.

We used Python's built-in plot function to plot the bar graph.

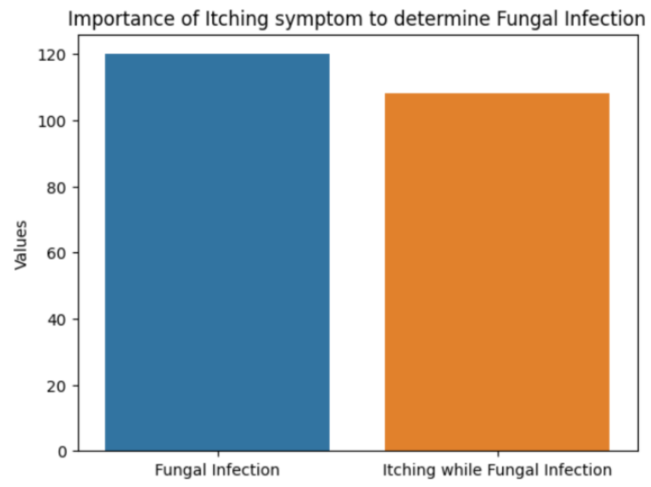


The distribution of restlessness symptom values is displayed in the bar graph on the left. It is evident that the 1 value has a count of approximately 300, whereas the 0 value has a count of approximately 4800.
The distribution of vomiting symptom values is displayed on the graph to the right. It is evident that the 1 value has a count of approximately 400, while the 0 value has a count of approximately 4500.

Activity 2.2: Bivariate Analysis:

Bivariate analysis is used to determine the relationship between two features. The association between prognosis and the levels of fungal infection and itching symptom is visualised here.

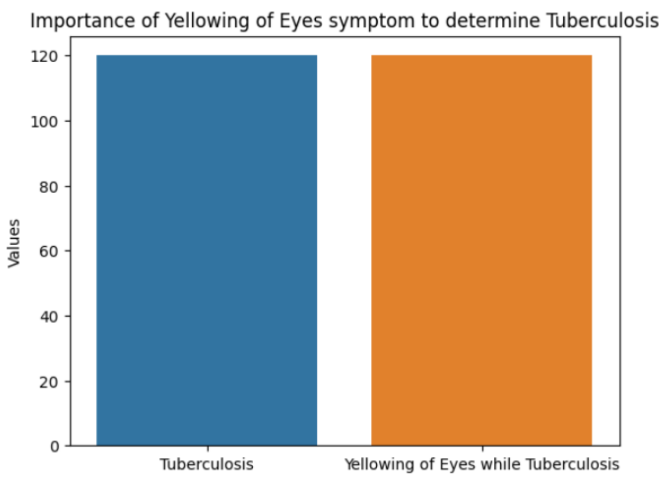
```
In [23]: a = len(train[train['prognosis'] == 'Fungal infection'])
         b = len(train[(train['itching'] == 1) & (train['prognosis'] == 'Fungal infection')])
         fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Fungal Infection','Itching while Fungal Infection'])

         sns.barplot(data = fi, x = fi.index, y = fi['Values'])
         plt.title('Importance of Itching symptom to determine Fungal Infection')

Out[23]: Text(0.5, 1.0, 'Importance of Itching symptom to determine Fungal Infection')
```



Here, we remove values from the prognosis column where the values are "Fungal Infection" by using Boolean indexing. The variable "a" has the data from these observations. Additionally, we eliminate values from the prognosis when they pertain to "Fungal Infection" and when the Itching variable has a value of 1. The graphic indicates that there is a high probability of a value of 1 in the Itching column when there is a fungal infection. There are 104 values with a value of 1 for the itching column and 120 ones with a value of fungal infection. This indicates that there is a good likelihood that itching will occur as a sign of a fungal infection.

Next, we observed the correlation between the prognosis and yellowing_of_the_eyes, a sign of tuberculosis. The figure indicates that there is a substantial probability that the yellowing of the eyes column will have a value of 1 in cases of tuberculosis. Tuberculosis is present in 120 values, while yellowing of the eyes is present in 119 values with a value of 1. This demonstrates the increased likelihood of yellowing of the eyes as a symptom of tuberculosis.\

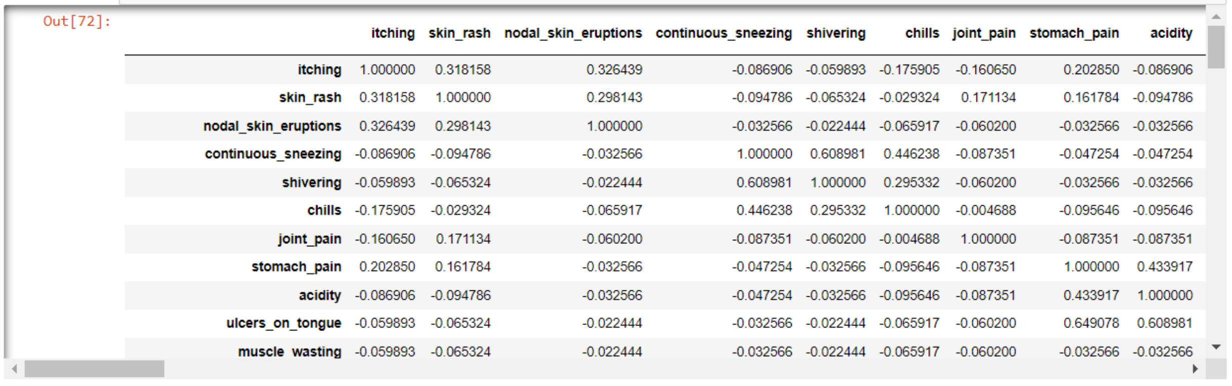
```
In [25]: a = len(train[train['prognosis'] == 'Tuberculosis'])
         b = len(train[(train['yellowing_of_eyes'] == 1) & (train['prognosis'] == 'Tuberculosis')])
         fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Tuberculosis','Yellowing of Eyes while Tuberculosis'])

         sns.barplot(data = fi, x = fi.index, y = fi['Values'])
         plt.title('Importance of Yellowing of Eyes symptom to determine Tuberculosis')

Out[25]: Text(0.5, 1.0, 'Importance of Yellowing of Eyes symptom to determine Tuberculosis')
```



Activity 2.3: Multivariate Analysis

We look for relationships between various characteristics through multivariate analysis. The correlation matrix is the main tool that may be used for this.

```
In [72]: corr_train = train.corr(numeric_only=True)
         corr_train.style.background_gradient('coolwarm')
```

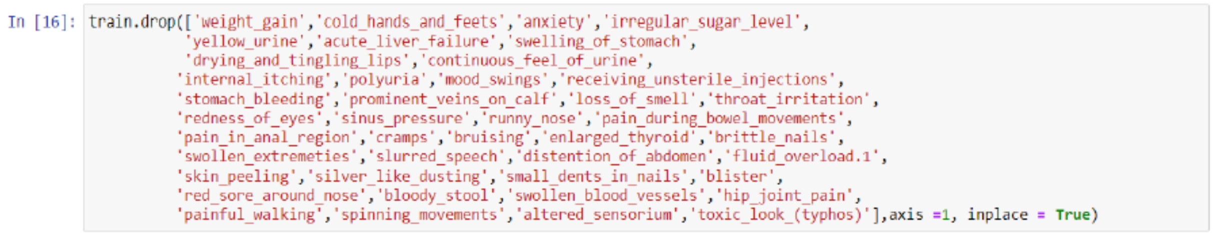


Out[72]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity
itching	1.000000	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.160650	0.202850	-0.086906
skin_rash	0.318158	1.000000	0.298143	-0.094786	-0.065324	-0.029324	0.171134	0.161784	-0.094786
nodal_skin_eruptions	0.326439	0.298143	1.000000	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566
continuous_sneezing	-0.086906	-0.094786	-0.032566	1.000000	0.608981	0.446238	-0.087351	-0.047254	-0.047254
shivering	-0.059893	-0.065324	-0.022444	0.608981	1.000000	0.295332	-0.060200	-0.032566	-0.032566
chills	-0.175905	-0.029324	-0.065917	0.446238	0.295332	1.000000	-0.004688	-0.095646	-0.095646
joint_pain	-0.160650	0.171134	-0.060200	-0.087351	-0.060200	-0.004688	1.000000	-0.087351	-0.087351
stomach_pain	0.202850	0.161784	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	1.000000	0.433917
acidity	-0.086906	-0.094786	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	0.433917	1.000000
ulcers_on_tongue	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.649078	0.608981
muscle_wasting	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566

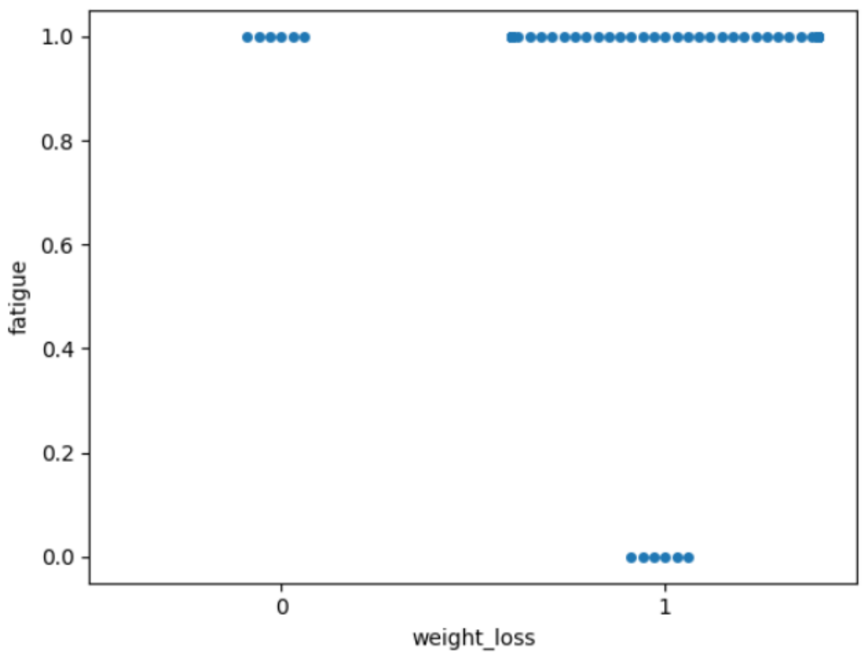
Given that there are 131 columns with numerical values, the correlation matrix has 131 X 131 dimensions. The only way to understand these numerous features is to browse. We attempt to exclude the values that have a strong correlation with one another from the correlation matrix.

Only one value can be eliminated when two values have a strong correlation with one another. When there is a correlation of more than 0.9 between the columns, we eliminate those columns.



```
In [16]: 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)
```

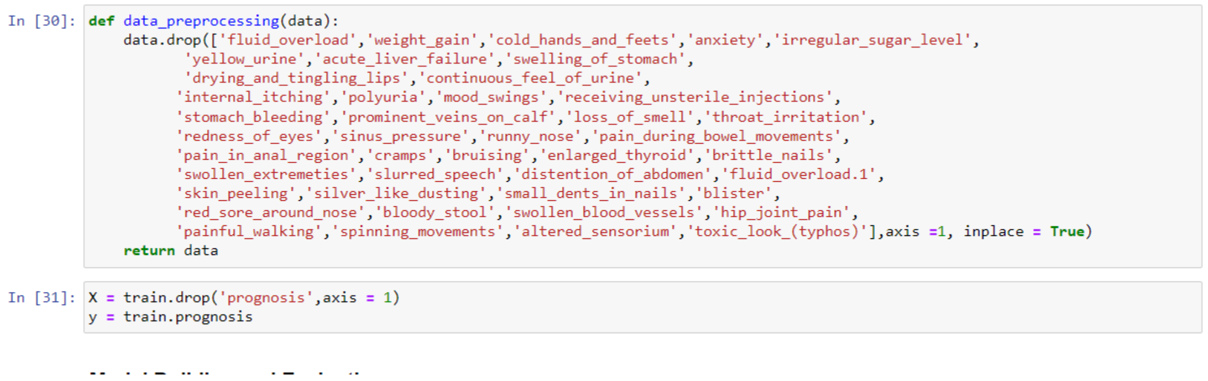
The columns that are eliminated because of strong association are visible.
We will also create a swarmplot with a column for tiredness and weight loss and a column for tuberculosis prognosis for multivariate analysis.



This swarm plot shows that when fatigue and weight loss are both zero, there is no observation for tuberculosis sickness. There are instances where a patient has just one of the two symptoms, although exhaustion and weight loss are frequently seen by patients with tuberculosis.

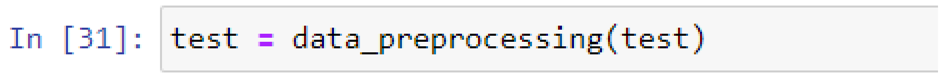
Preprocessing of Test data

For the test data, preparation must be completed. We can write a preprocessing function for test data that will leave us with just the features we need. Every action we took for the training set will be included in this function.



```
In [30]: 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

In [31]: X = train.drop('prognosis',axis = 1)
         y = train.prognosis
```

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

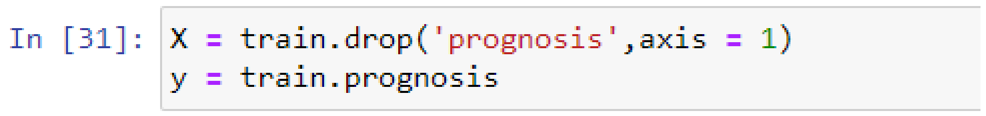


```
In [31]: test = data_preprocessing(test)
```

Here we call the function for the test data.

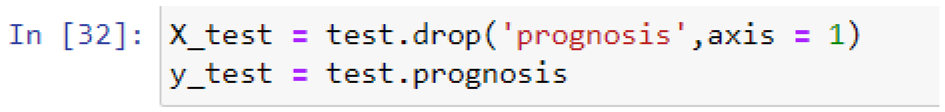
Activity 2.5: Split data into training, validation and testing data

Data for testing and training have been provided separately. The training data was further divided into training and validation sets. Hyperparameter adjustment can be performed using this validation data.
The characteristics and the target variable must first be separated. The target variable is predicted using the features.



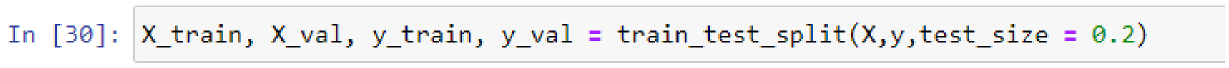
```
In [31]: X = train.drop('prognosis',axis = 1)
         y = train.prognosis
```

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



```
In [32]: X_test = test.drop('prognosis',axis = 1)
         y_test = test.prognosis
```

In this case, the test data was divided into features (X_test) and the associated target variables (y_test).
The training data must now be divided into training and validation data. You can use the following command to accomplish it.



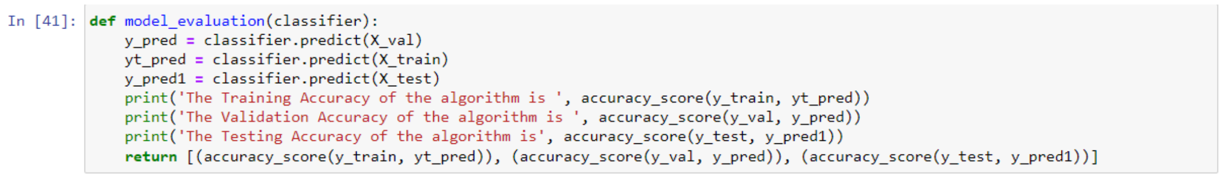
```
In [30]: X_train, X_val, y_train, y_val = train_test_split(X,y,test_size = 0.2)
```

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

Milestone 4: Model Building

Activity 1: Creating a function for model evaluation

We'll be building and testing a number of models. If we construct a function for model assessment directly, it will lessen our tedious effort.



```
In [41]: 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))]
```

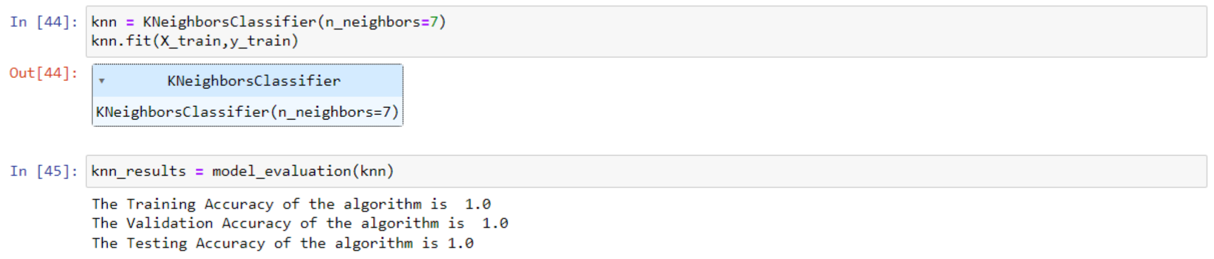
This function will show the accuracies of prediction of model for training, validation and testing data. It will also the return those accuracies.

Activity 2: Training and testing the models using multiple algorithms

Having a function for assessment and clean data, it's time to create models to train the data. Four distinct categorization algorithms will be used in this study to construct our models. For prediction, the best model will be applied.

Activity 2.1: K Nearest Neighbors Model

Initialization of the KNeighborsClassifier() algorithm is done in a variable called knn. The.fit() function is utilised to train the knn model. The training features and training target variables, or X_train and y_train data, are used to train the model. The model evaluation function is then given this model to see how well it performs.

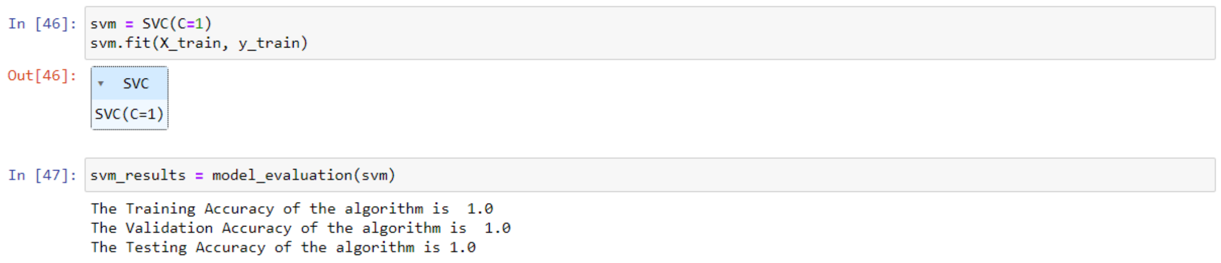


```
In [44]: knn = KNeighborsClassifier(n_neighbors=7)
         knn.fit(X_train,y_train)

Out[44]: KNeighborsClassifier
         KNeighborsClassifier(n_neighbors=7)

In [45]: 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
```

This illustrates that the model has 100% accuracy on testing, validation, and training data. The great accuracy eliminates the necessity for hyperparameter adjustment. A variable called knn_results holds the results.

Activity 2.2: SVM Model

Initialization of the SVC() algorithm is done in a variable called svm. The.fit() function is utilised to train the SVM model. The training features and training target variables, or X_train and Y_train data, are used to train the model. Next, the model_evaluation function is used to assess the model's performance.

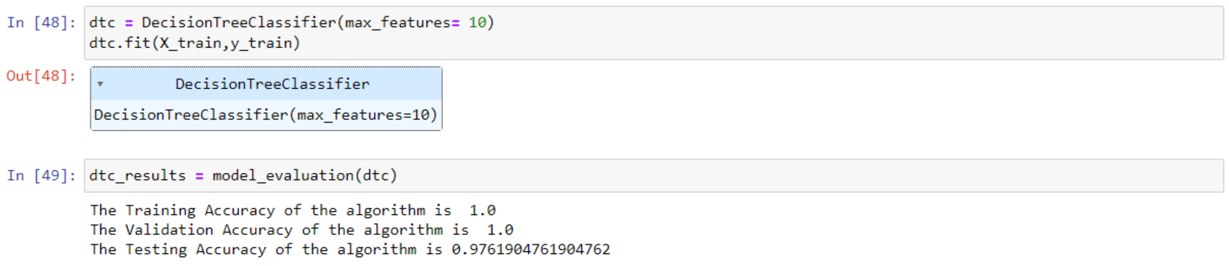


```
In [46]: svm = SVC(C=1)
         svm.fit(X_train, y_train)

Out[46]: SVC
         SVC(C=1)

In [47]: 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
```

This illustrates that the model has 100% accuracy on testing, validation, and training data. The great accuracy eliminates the necessity for hyperparameter adjustment. A variable called svm_results holds the results.

Activity 2.3: Decision Tree Model

The DecisionTreeClassifier() algorithm is initialised in a variable called dtc, with the parameter max_features set to 10. The.fit() function is used to train the DTC model. The X_train and y_train data, which represent the training features and training target variables, are used to train the model. Next, the model_evaluation function is used to assess the model's performance.

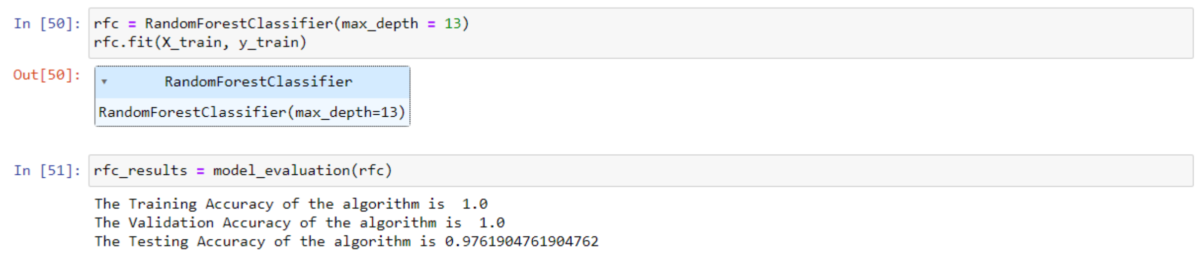


```
In [48]: dtc = DecisionTreeClassifier(max_features= 10)
         dtc.fit(X_train,y_train)

Out[48]: DecisionTreeClassifier
         DecisionTreeClassifier(max_features=10)

In [49]: 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
```

The model has achieved 100% accuracy on both training and validation data, as can be seen below. With testing data, it has a 97.6% accuracy rate. The great accuracy eliminates the necessity for hyperparameter adjustment. A variable called dtc_results holds the results.

Activity 2.4: Random Forest Model

A bagging technique called the Random Forest Classifier uses several decision trees and aggregates them to get a forecast. The RandomForestClassifier() algorithm is initialised in a variable called rfc, with the parameter max_depth set to 13. The.fit() function is used to train the rfc model. The training features and training target variables, or X_train and y_train data, are used to train the model. The model evaluation function is then given this model to see how well it performs.

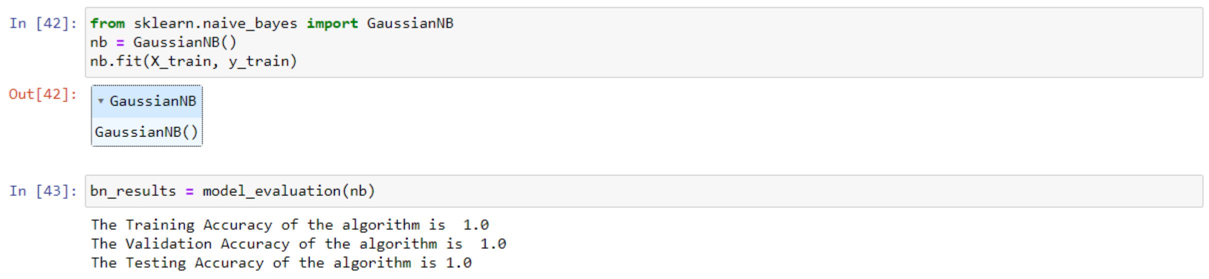


```
In [50]: rfc = RandomForestClassifier(max_depth = 13)
         rfc.fit(X_train, y_train)

Out[50]: ▾      RandomForestClassifier
         RandomForestClassifier(max_depth=13)

In [51]: 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
```

The model has achieved 100% accuracy on both training and validation data, as can be seen below. With testing data, it has a 97.6% accuracy rate. The great accuracy eliminates the necessity for hyperparameter adjustment. A variable called rfc_results holds the results.

Activity 2.5: Gaussian Naïve Bayes

Naive Bayes is a classification algorithm based on Bayes' theorem. It assumes independence between features, which simplifies calculations. It calculates the probability of a class given a set of features by multiplying the probabilities of each feature given that class. Despite its simplicity, Naive Bayes is efficient and widely used in tasks like spam filtering and sentiment analysis.



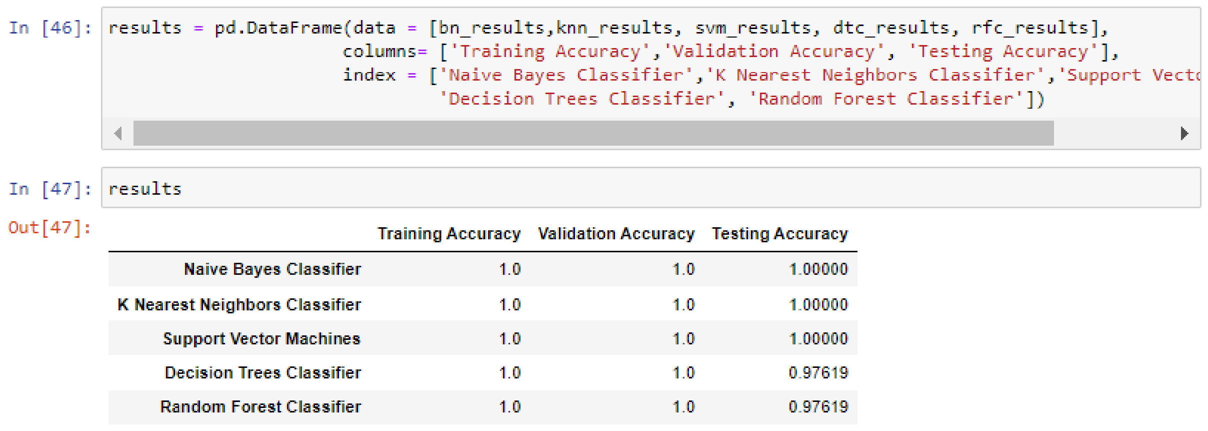
```
In [42]: from sklearn.naive_bayes import GaussianNB
         nb = GaussianNB()
         nb.fit(X_train, y_train)

Out[42]: ▾ GaussianNB
         GaussianNB()

In [43]: bn_results = model_evaluation(nb)

         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
```

The model has achieved 100% accuracy on both training and validation data, as can be seen below. With testing data, it has a 100% accuracy rate. The great accuracy eliminates the necessity for hyperparameter adjustment. A variable called bn_results holds the 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 a confusion matrix as the dimensions of the confusion matrix will be 41 X 41. We can check accuracy to test the model. We already have 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.



```
In [46]: results = pd.DataFrame(data = [bn_results,knn_results, svm_results, dtc_results, rfc_results],
                      columns= ['Training Accuracy','Validation Accuracy', 'Testing Accuracy'],
                      index = ['Naive Bayes Classifier','K Nearest Neighbors Classifier','Support Vecto
                               'Decision Trees Classifier', 'Random Forest Classifier'])
```

```
In [47]: results
```

Out[47]:

	Training Accuracy	Validation Accuracy	Testing Accuracy
Naive Bayes Classifier	1.0	1.0	1.00000
K Nearest Neighbors Classifier	1.0	1.0	1.00000
Support Vector Machines	1.0	1.0	1.00000
Decision Trees Classifier	1.0	1.0	0.97619
Random Forest Classifier	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.

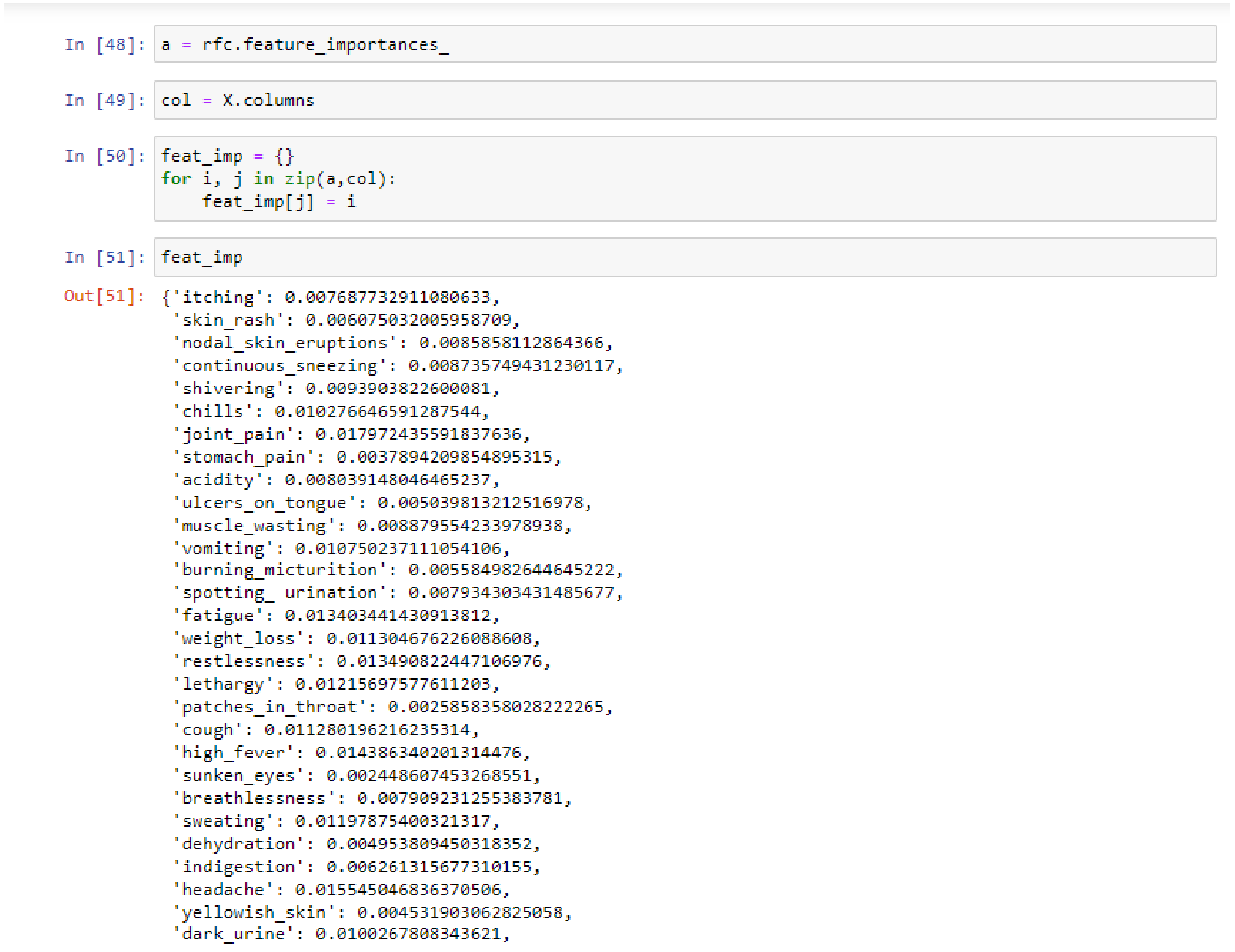
Activity 3: Comparing Model accuracy for different number of features.

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.

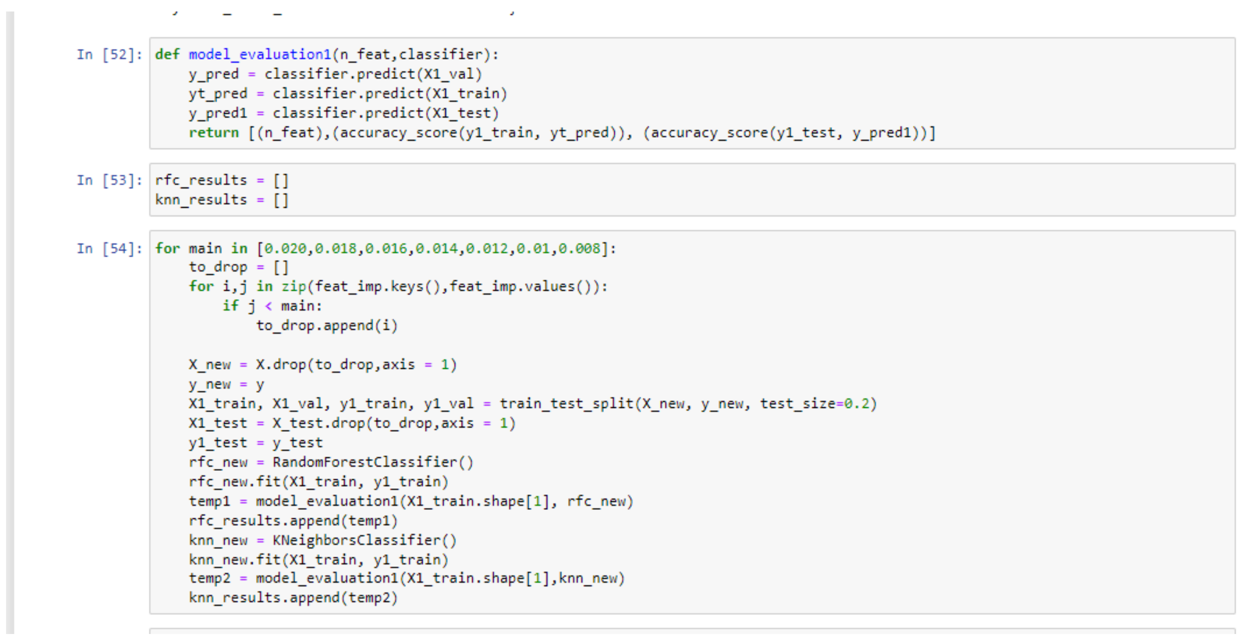


```
In [48]: a = rfc.feature_importances_

In [49]: col = X.columns

In [50]: feat_imp = {}
         for i, j in zip(a,col):
             feat_imp[j] = i

In [51]: feat_imp

Out[51]: {'itching': 0.007687732911080633,
          'skin_rash': 0.006075032005958709,
          'nodal_skin_eruptions': 0.0085858112864366,
          'continuous_sneezing': 0.008735749431230117,
          'shivering': 0.0093903822600081,
          'chills': 0.010276646591287544,
          'joint_pain': 0.017972435591837636,
          'stomach_pain': 0.0037894209854895315,
          'acidity': 0.008039148046465237,
          'ulcers_on_tongue': 0.005039813212516978,
          'muscle_wasting': 0.008879554233978938,
          'vomiting': 0.010750237111054106,
          'burning_micturition': 0.005584982644645222,
          'spotting_ urination': 0.007934303431485677,
          'fatigue': 0.013403441430913812,
          'weight_loss': 0.011304676226088608,
          'restlessness': 0.013490822447106976,
          'lethargy': 0.01215697577611203,
          'patches_in_throat': 0.0025858358028222265,
          'cough': 0.011280196216235314,
          'high_fever': 0.014386340201314476,
          'sunken_eyes': 0.002448607453268551,
          'breathlessness': 0.007909231255383781,
          'sweating': 0.011978754003213 17,
          'dehydration': 0.004953809450318352,
          'indigestion': 0.006261315677310155,
          'headache': 0.015545046836370506,
          'yellowish_skin': 0.004531903062825058,
          'dark_urine': 0.0100267808343621,
```

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.



```
In [52]: def model_evaluation1(n_feat,classifier):
             y_pred = classifier.predict(X1_val)
             yt_pred = classifier.predict(X1_train)
             y_pred1 = classifier.predict(X1_test)
             return [(n_feat),(accuracy_score(y1_train, yt_pred)), (accuracy_score(y1_test, y_pred1))]

In [53]: rfc_results = []
         knn_results = []

In [54]: 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 the to_drop list.

The columns whose name is in the to_drop list will be dropped.

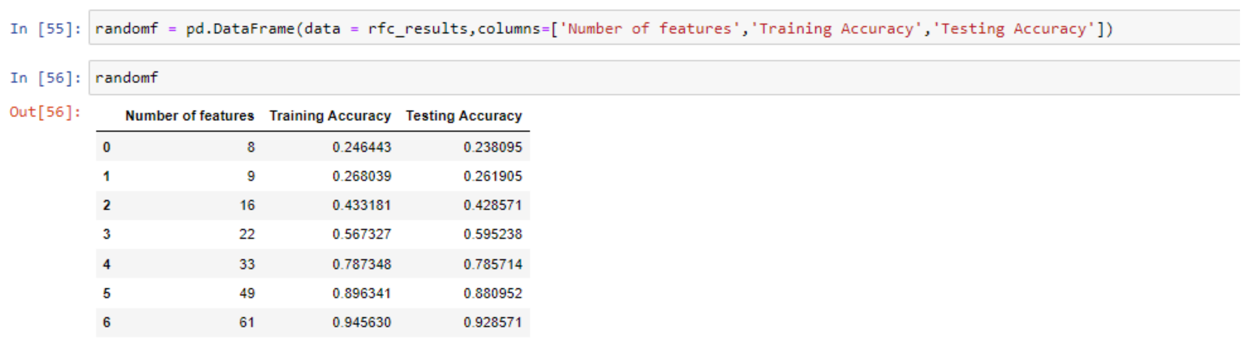
The new data will be split into features and target variable. Further they will be split into training, validation and testing data.

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.

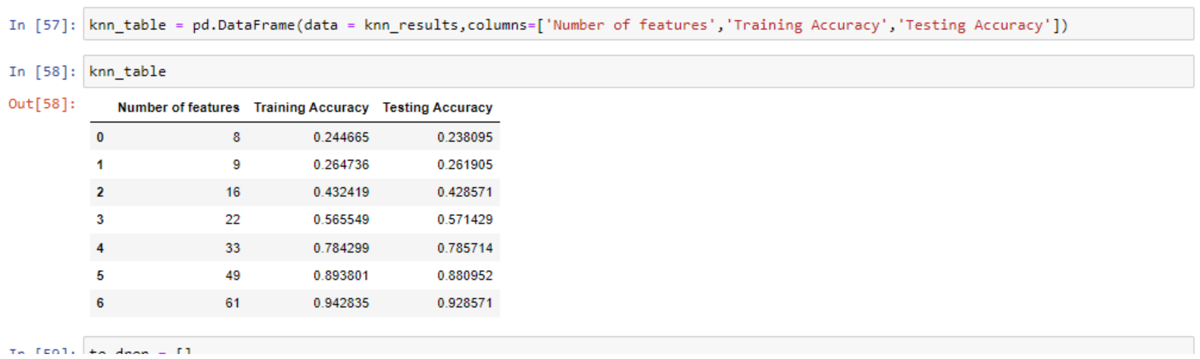
We then plot a table using the number of features and accuracies.



```
In [55]: randomf = pd.DataFrame(data = rfc_results,columns=['Number of features','Training Accuracy','Testing Accuracy'])

In [56]: randomf
Out[56]:
```

	Number of features	Training Accuracy	Testing Accuracy
0	8	0.246443	0.238095
1	9	0.268039	0.261905
2	16	0.433181	0.428571
3	22	0.567327	0.595238
4	33	0.787348	0.785714
5	49	0.896341	0.880952
6	61	0.945630	0.928571

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.



```
In [57]: knn_table = pd.DataFrame(data = knn_results,columns=['Number of features','Training Accuracy','Testing Accuracy'])

In [58]: knn_table
Out[58]:
```

	Number of features	Training Accuracy	Testing Accuracy
0	8	0.244665	0.238095
1	9	0.264736	0.261905
2	16	0.432419	0.428571
3	22	0.565549	0.571429
4	33	0.784299	0.785714
5	49	0.893801	0.880952
6	61	0.942835	0.928571

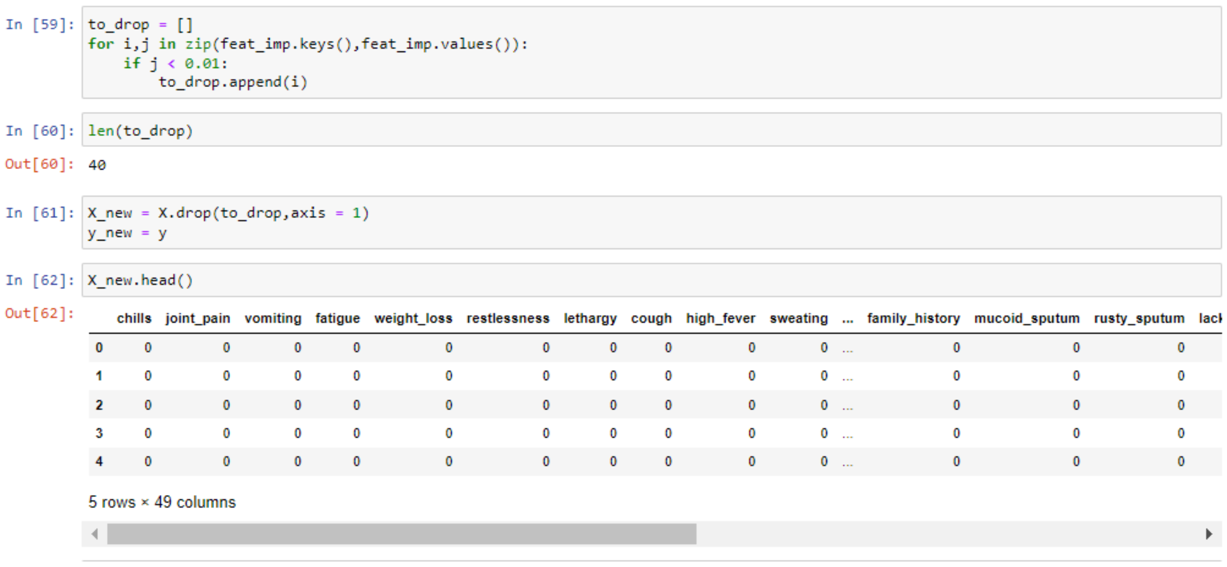
```
In [59]: to_drop = []
```

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 61 features.
Hence we will choose 45 features for our training.

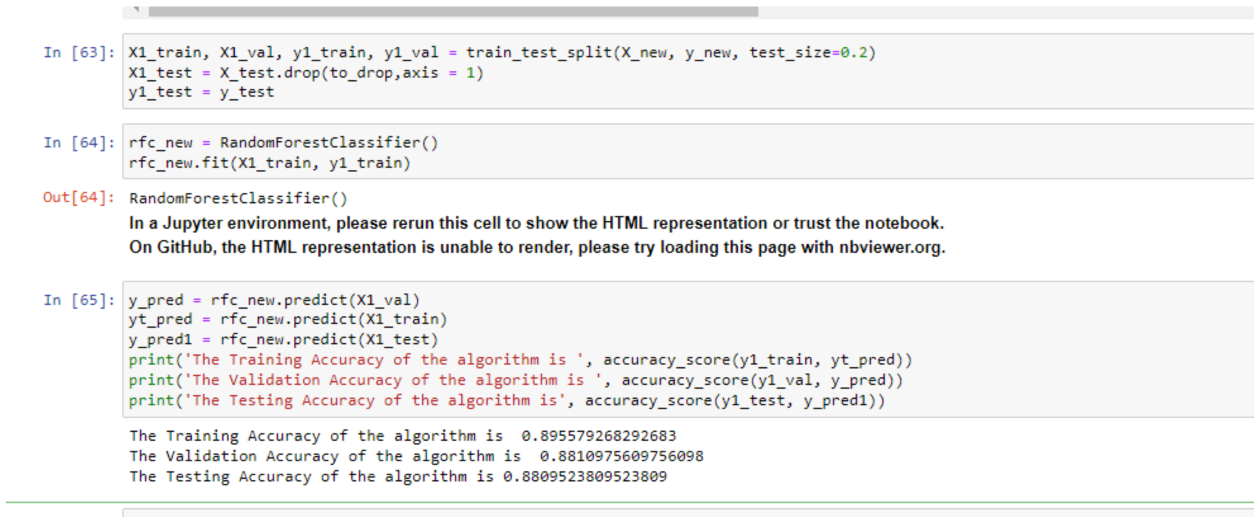


```
In [59]: to_drop = []
         for i,j in zip(feat_imp.keys(),feat_imp.values()):
             if j < 0.01:
                 to_drop.append(i)

In [60]: len(to_drop)

Out[60]: 40

In [61]: X_new = X.drop(to_drop,axis = 1)
         y_new = y

In [62]: X_new.head()

Out[62]:
```

	chills	joint_pain	vomiting	fatigue	weight_loss	restlessness	lethargy	cough	high_fever	sweating	...	family_history	mucoid_sputum	rusty_sputum	lack
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

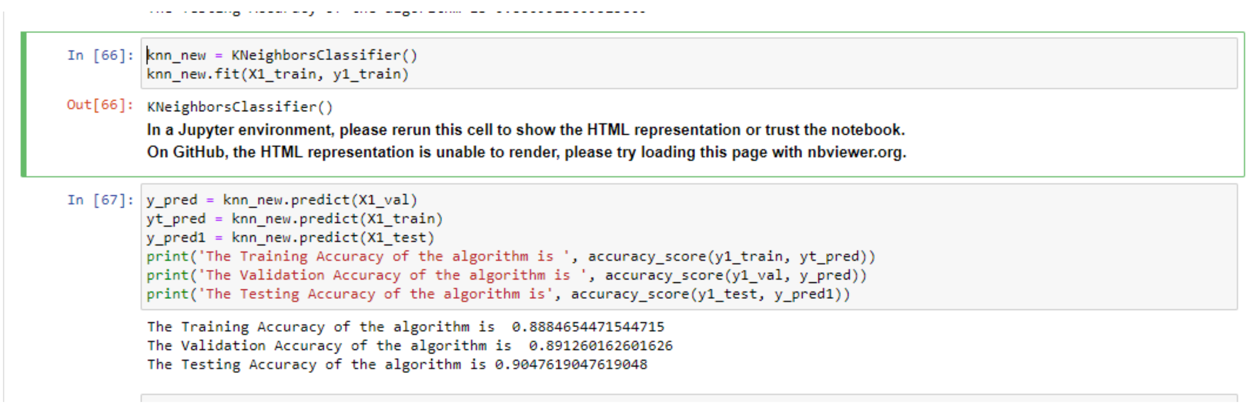
5 rows × 49 columns

We drop 40 features. Create new datasets for features and their labels. As we can see in the figure above we are left with 49 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 88.09% which is quite good for the number of features and training accuracy of 89.55% . 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.



```
In [63]: 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

In [64]: rfc_new = RandomForestClassifier()
         rfc_new.fit(X1_train, y1_train)

Out[64]: RandomForestClassifier()
         In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
         On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [65]: 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(y1_train, yt_pred))
         print('The Validation Accuracy of the algorithm is ', accuracy_score(y1_val, y_pred))
         print('The Testing Accuracy of the algorithm is', accuracy_score(y1_test, y_pred1))

         The Training Accuracy of the algorithm is  0.895579268292683
         The Validation Accuracy of the algorithm is  0.8810975609756098
         The Testing Accuracy of the algorithm is 0.8809523809523809
```

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



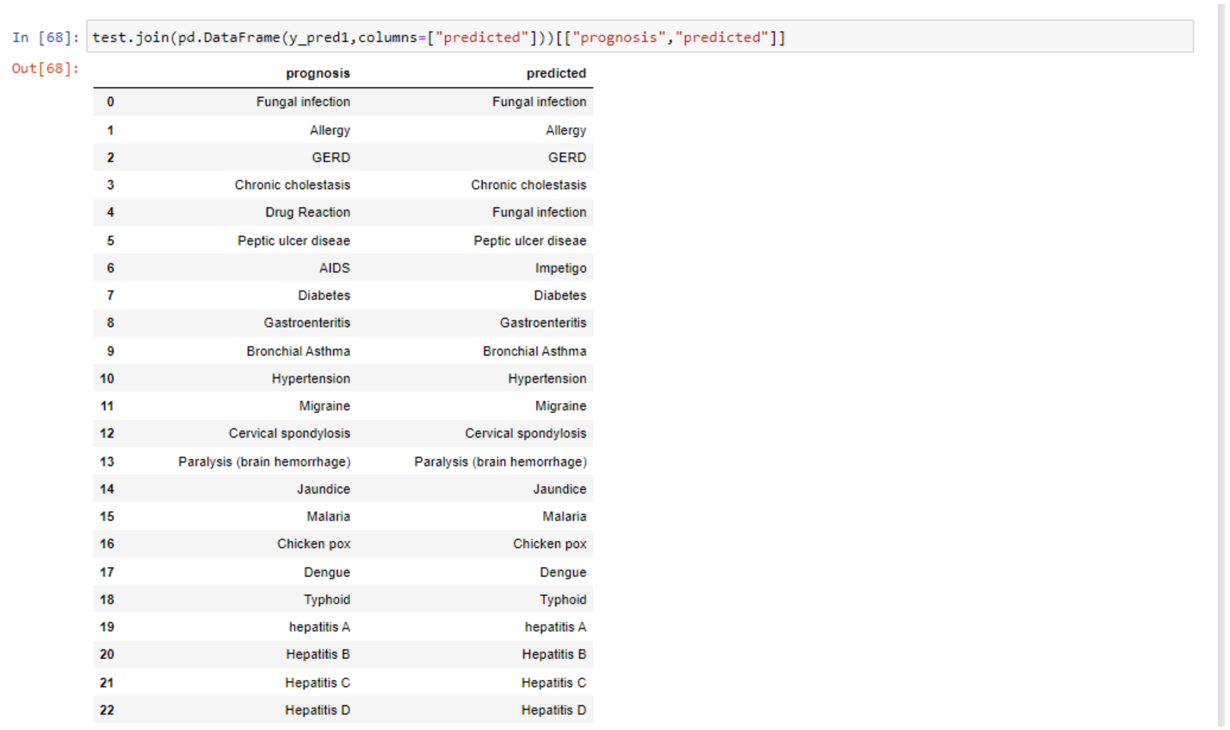
```
In [66]: knn_new = KNeighborsClassifier()
         knn_new.fit(X1_train, y1_train)

Out[66]: KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [67]: 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(y1_train, yt_pred))
         print('The Validation Accuracy of the algorithm is ', accuracy_score(y1_val, y_pred))
         print('The Testing Accuracy of the algorithm is', accuracy_score(y1_test, y_pred1))

         The Training Accuracy of the algorithm is  0.8884654471544715
         The Validation Accuracy of the algorithm is  0.891260162601626
         The Testing Accuracy of the algorithm is 0.9047619047619048
```

After training the knn model we check the accuracies. Our model has achieved 90.4 % accuracy for the test data.

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



```
In [68]: test.join(pd.DataFrame(y_pred1,columns=["predicted"]))[["prognosis","predicted"]]

Out[68]:
```

	prognosis	predicted
0	Fungal infection	Fungal infection
1	Allergy	Allergy
2	GERD	GERD
3	Chronic cholestasis	Chronic cholestasis
4	Drug Reaction	Fungal infection
5	Peptic ulcer diseae	Peptic ulcer diseae
6	AIDS	Impetigo
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
16	Chicken pox	Chicken pox
17	Dengue	Dengue
18	Typhoid	Typhoid
19	hepatitis A	hepatitis A
20	Hepatitis B	Hepatitis B
21	Hepatitis C	Hepatitis C
22	Hepatitis D	Hepatitis D

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.

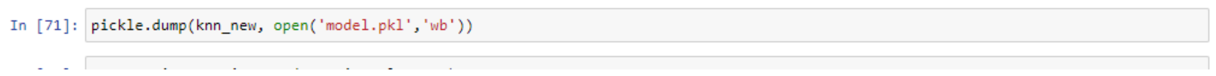
Milestone 6: Model Deployment

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 41 features.



```
In [71]: 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

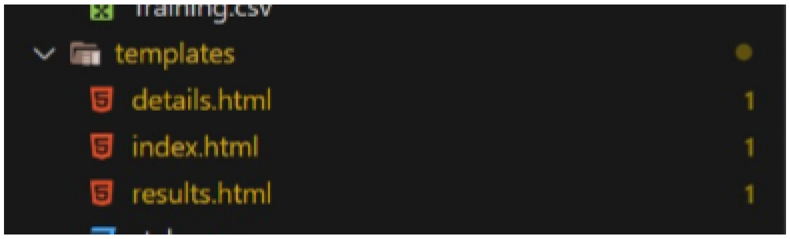
- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building HTML pages:

For this project we create three HTML files namely

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

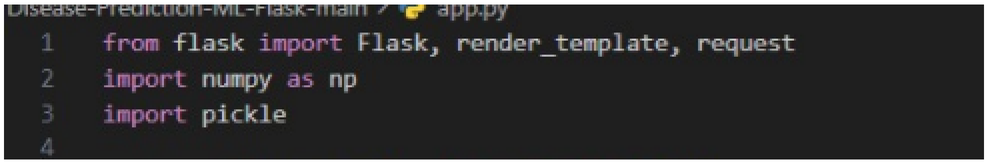
And we will save them in the templates folder.


training.csv
templates
details.html 1
index.html 1
results.html 1

Activity 2.2: Build Python code

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

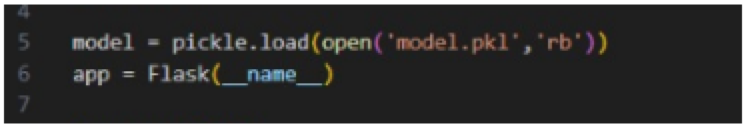
Import the necessary Libraries.


Disease-Prediction-ML-Flask-main > app.py

```python
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.

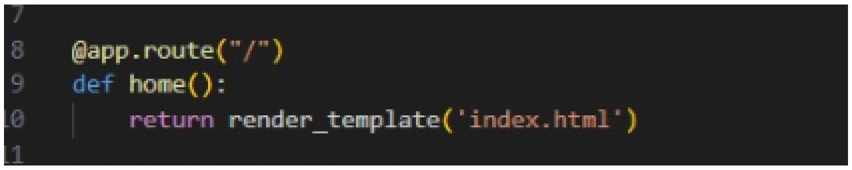


```python
model = pickle.load(open('model.pkl','rb'))
app = Flask(__name__)
```

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.

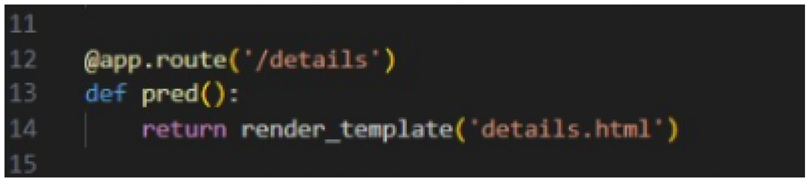
The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

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



```python
@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".



```python

@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 input in the form of strings.

A list is created with 49 0s and stored in variable b.

In the for loop x will take values from 0 to 49.

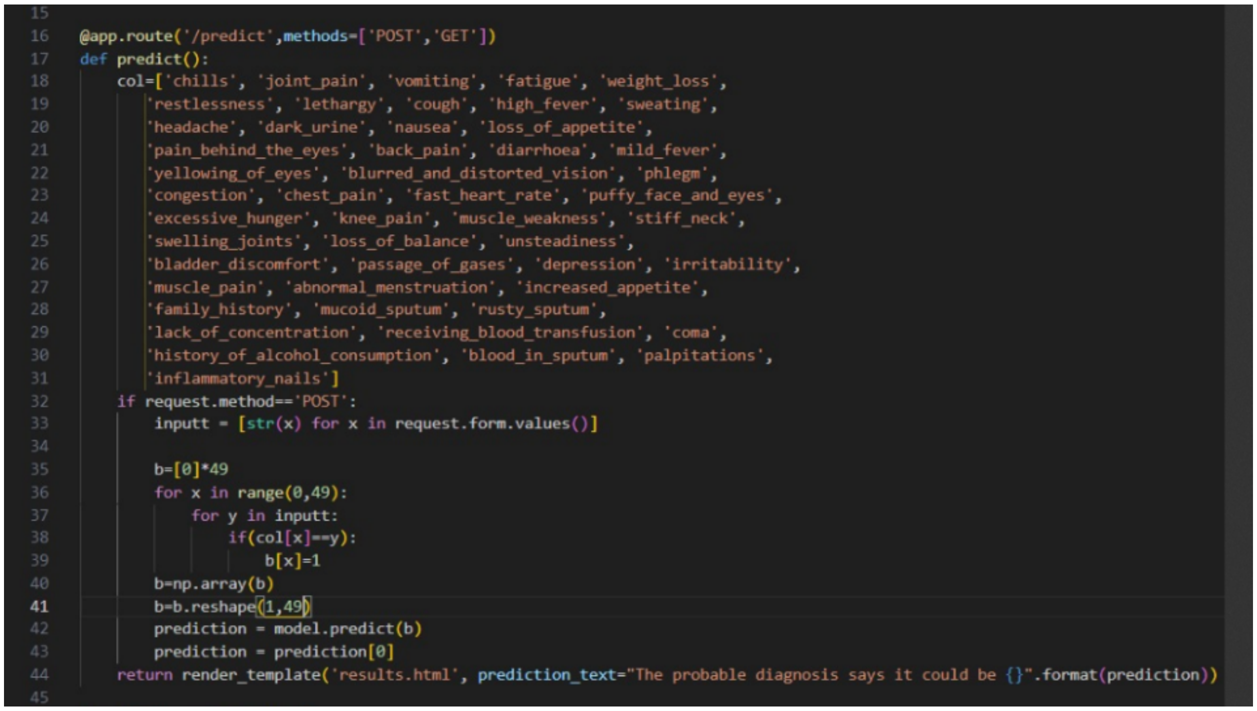
Another for loop is written where y will iterate over the values given by the user as inputs.

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 (1,49).

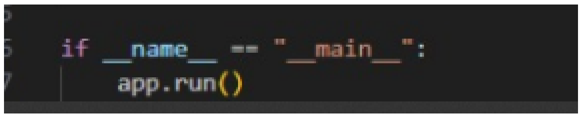
Then this array b is given to the model for prediction.

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



```python
@app.route('/predict',methods=['POST','GET'])
def predict():
    col=['chills', 'joint_pain', 'vomiting', 'fatigue', 'weight_loss',
        'restlessness', 'lethargy', 'cough', 'high_fever', 'sweating',
        'headache', 'dark_urine', 'nausea', 'loss_of_appetite',
        'pain_behind_the_eyes', 'back_pain', 'diarrhoea', 'mild_fever',
        'yellowing_of_eyes', 'blurred_and_distorted_vision', 'phlegm',
        'congestion', 'chest_pain', 'fast_heart_rate', 'puffy_face_and_eyes',
        'excessive_hunger', 'knee_pain', 'muscle_weakness', 'stiff_neck',
        'swelling_joints', 'loss_of_balance', 'unsteadiness',
        'bladder_discomfort', 'passage_of_gases', 'depression', 'irritability',
        'muscle_pain', 'abnormal_menstruation', 'increased_appetite',
        'family_history', 'mucoid_sputum', 'rusty_sputum',
        'lack_of_concentration', 'receiving_blood_transfusion', 'coma',
        'history_of_alcohol_consumption', 'blood_in_sputum', 'palpitations',
        'inflammatory_nails']
    if request.method=='POST':
        inputt = [str(x) for x in request.form.values()]

        b=[0]*49
        for x in range(0,49):
            for y in inputt:
                if(col[x]==y):
                    b[x]=1
        b=np.array(b)
        b=b.reshape(1,49)
        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:



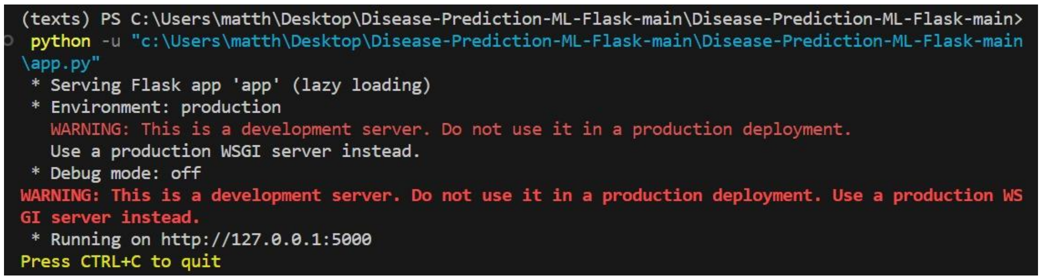
```python
if __name__ == "__main__":
    app.run()
```

This code sets the entry point of the Flask application.

The function "app.run()" is called, which starts the Flask deployment server.

Activity 2.3: Run the Web Application

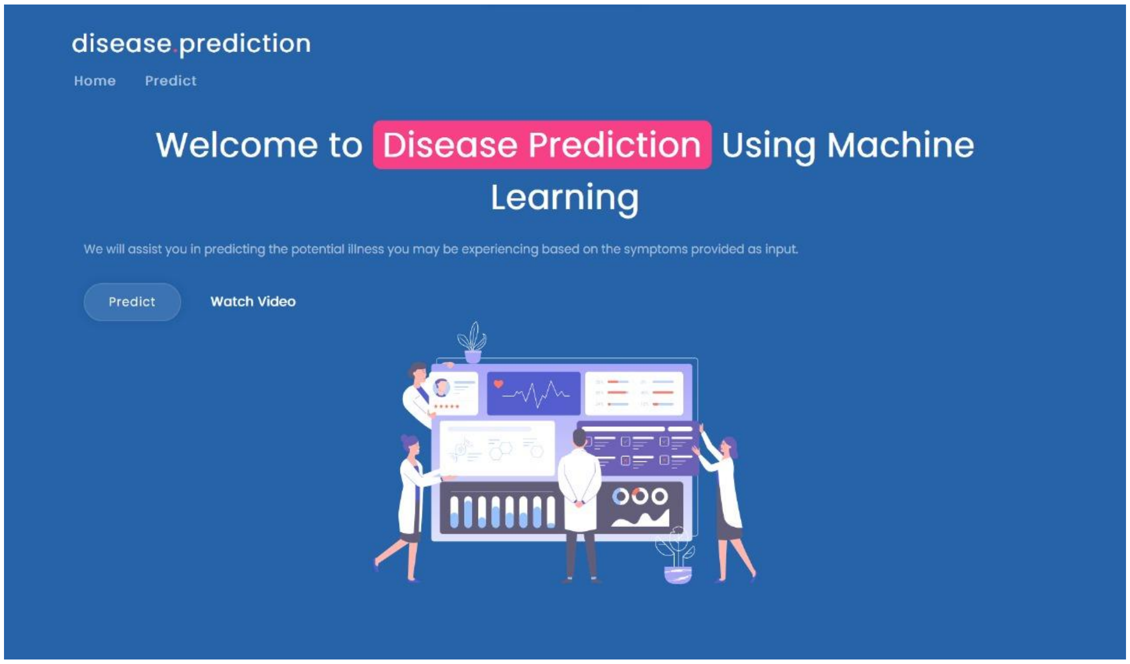
When you run the "app.py" file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser.



```
(texts) PS C:\Users\matth\Desktop\Disease-Prediction-ML-Flask-main\Disease-Prediction-ML-Flask-main>
 python -u "c:\Users\matth\Desktop\Disease-Prediction-ML-Flask-main\Disease-Prediction-ML-Flask-main
\app.py"
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WS
GI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

When we paste the URL in a web browser, our index.html page will open. It contains various sections in the header bar such as Home, Predict, About Model, Testimonials, FAQ, Contact. There is some information given on the web page about our model.

If you click on the Predict button on home page or in the header bar you will be redirected to the details.html page.
Our Detials.html looks as shown below.

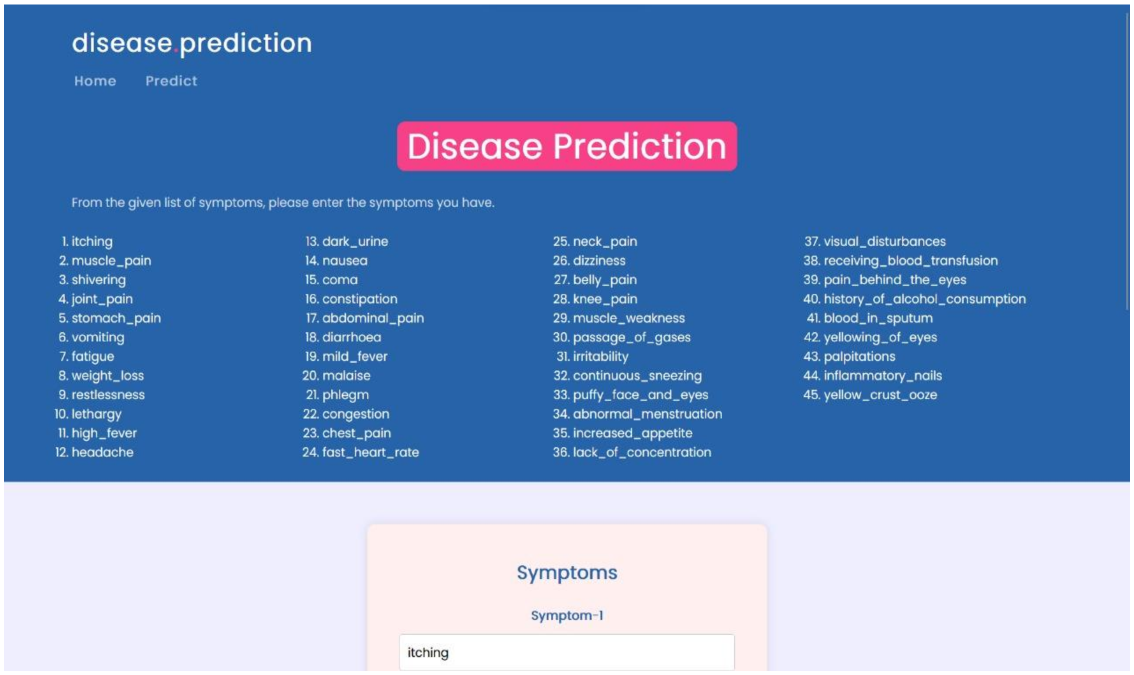

disease.prediction

Home Predict

Welcome to Disease Prediction Using Machine Learning

We will assist you in predicting the potential illness you may be experiencing based on the symptoms provided as input.

Predict Watch Video


disease.prediction

Home Predict

Disease Prediction

From the given list of symptoms, please enter the symptoms you have.

1. itching
2. muscle_pain
3. shivering
4. joint_pain
5. stomach_pain
6. vomiting
7. fatigue
8. weight_loss
9. restlessness
10. lethargy
11. high_fever
12. headache
13. dark_urine
14. nausea
15. coma
16. constipation
17. abdominal_pain
18. diarrhoea
19. mild_fever
20. malaise
21. phlegm
22. congestion
23. chest_pain
24. fast_heart_rate
25. neck_pain
26. dizziness
27. belly_pain
28. knee_pain
29. muscle_weakness
30. passage_of_gases
31. irritability
32. continuous_sneezing
33. puffy_face_and_eyes
34. abnormal_menstruation
35. increased_appetite
36. lack_of_concentration
37. visual_disturbances
38. receiving_blood_transfusion
39. pain_behind_the_eyes
40. history_of_alcohol_consumption
41. blood_in_sputum
42. yellowing_of_eyes
43. palpitations
44. inflammatory_nails
45. yellow_crust_ooze

Symptoms

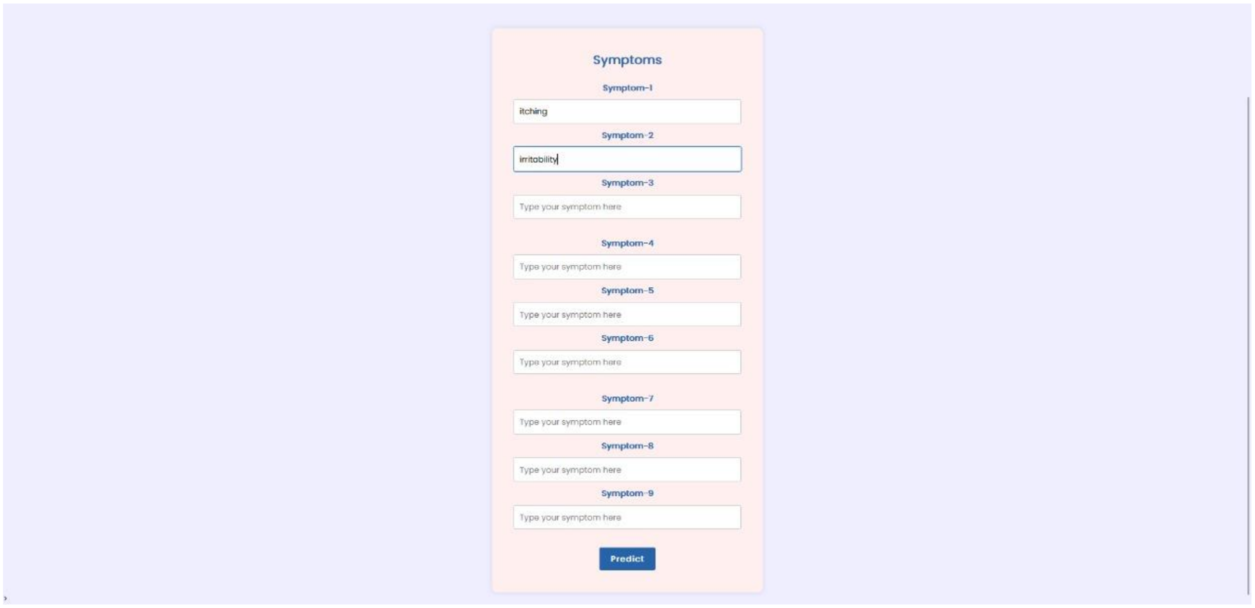
Symptom-1

itching

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.



Symptoms

Symptom-1: itching

Symptom-2: irritability

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

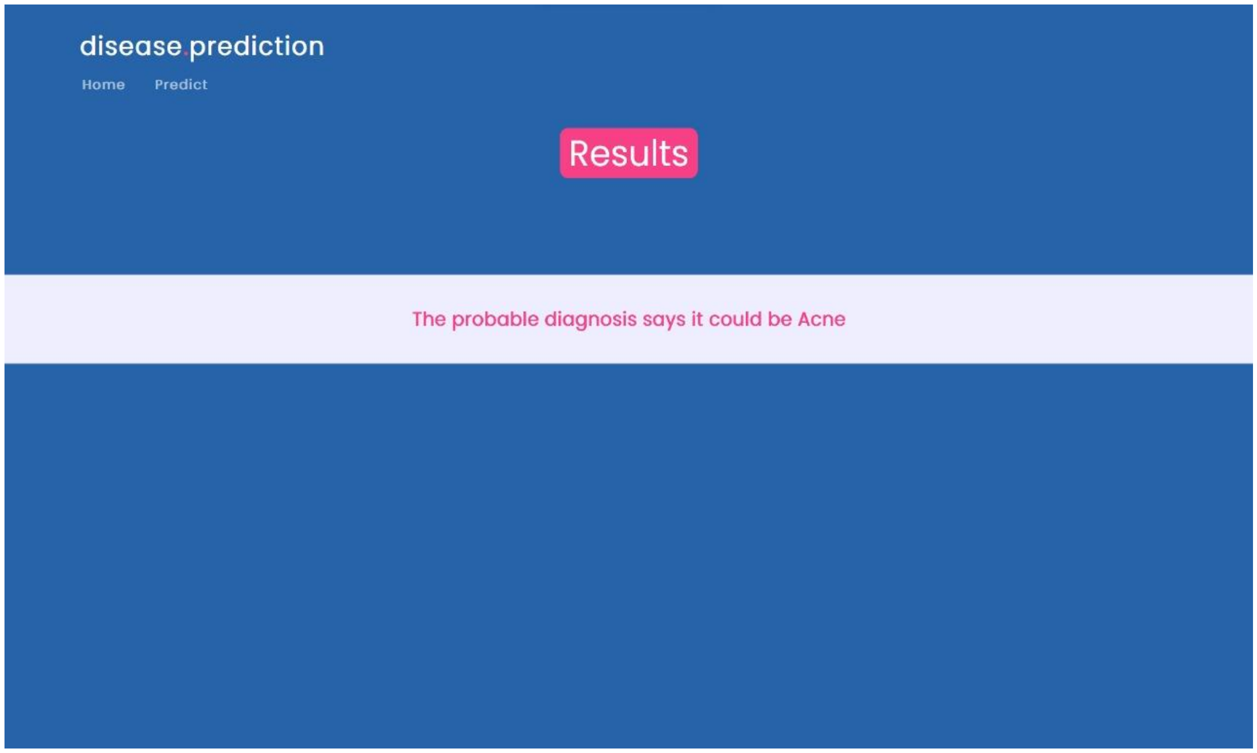
Symptom-8: Type your symptom here

Symptom-9: Type your symptom here

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

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

For example, we have added the symptoms of the patients suffering from Acne that are itching and irritability



The probable diagnosis says it could be Acne

The results say that there are high chances that a patient has Acne based on the various symptoms we have given as input.