DISEASE PREDICTION USING MACHINE LEARNING

In the midst of today's fast-paced world, where time is a precious commodity, healthcare often takes a backseat. Even when experiencing severe symptoms of various ailments, individuals often delay seeking medical attention. The ease of resorting to self-diagnosis through online symptom searches often leads to misleading results, frequently pointing towards stereotypical diseases. This has led to a growing skepticism towards online medical diagnosis.

To address these prevalent concerns, we have developed a groundbreaking model capable of predicting up to 42 diseases based on symptom inputs. This innovative tool empowers both healthcare professionals and individuals seeking preventive diagnosis and self-care. For doctors, this model serves as a valuable asset for online consultations, providing preliminary insights based on symptom-driven analysis. Patients can also utilize this model to make informed decisions regarding their health, particularly considering the rising costs of traditional medical consultations.

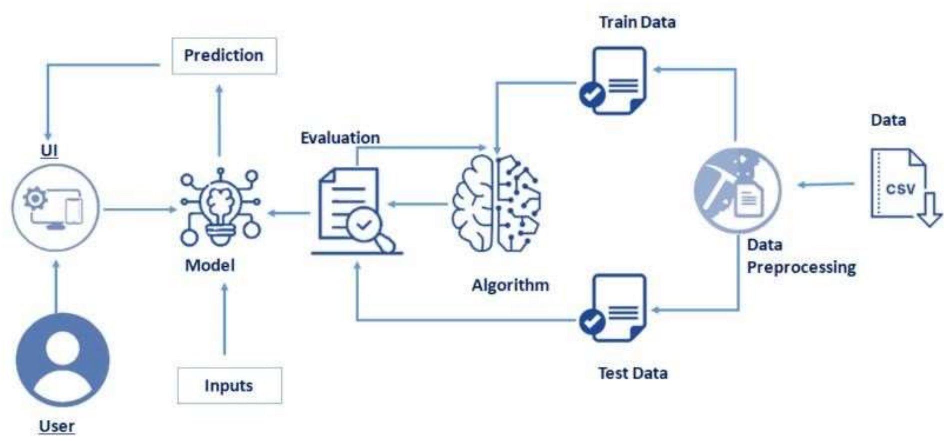
The model operates without requiring any personalized data, such as name, age, gender, religion, or address. This ensures user privacy and eliminates concerns about data breaches or misuse. Individuals can access this web application anytime they suspect a potential health issue, and the model will provide a list of probable diseases based on the symptoms provided. This empowers users to make informed decisions about seeking further medical attention.

The model's accuracy stands at an impressive 97 out of 100 times on average, a testament to its effectiveness in disease prediction. However, it is crucial to emphasize that this model should be primarily utilized for preventive diagnosis and early intervention by healthcare professionals. It should not be considered a substitute for a comprehensive medical evaluation and diagnosis.

The introduction of this innovative model represents a significant advancement in the realm of healthcare. Its ability to provide prompt and accurate symptom-based disease prediction empowers individuals to take charge of their health, while simultaneously assisting healthcare professionals in delivering timely and effective care. As technology continues to evolve, such advancements hold the potential to revolutionize healthcare, making it more accessible, efficient, and personalized.

Let us look at the Technical Architecture of the project.

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

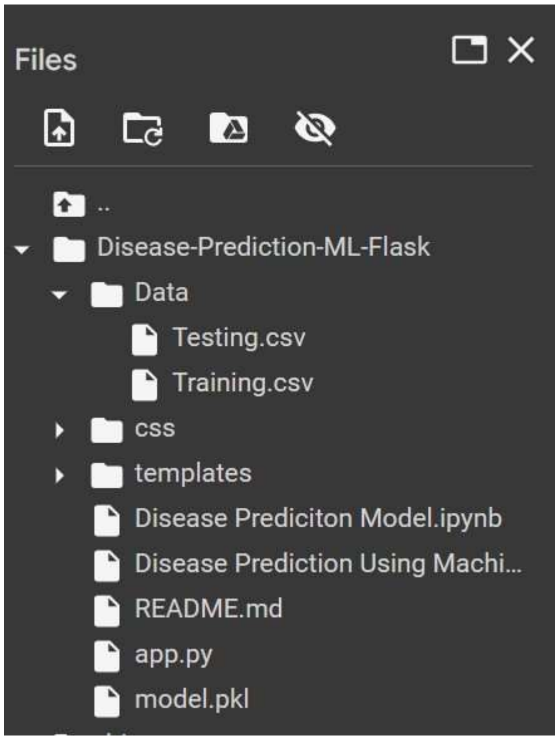
- **Model Building** o Creating a function for evaluation
  - o Training and testing the Models using multiple algorithms

- **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:

Create the Project folder which contains files as shown below:



```
Files
..
Disease-Prediction-ML-Flask
    Data
        Testing.csv
        Training.csv
    css
    templates
    Disease Prediciton Model.ipynb
    Disease Prediction Using Machi...
    README.md
    app.py
    model.pkl
```

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

Define Problem / Problem Understanding

Activity 1: Specify the business problem:

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

Activity 2: Business requirements:

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

- Accurate and reliable information: The case of disease prediction is critical and no false information can be tolerated since the consequences can be severe. Also the right symptoms should be linked to the right diseases so that the output is inline with all the patients health situations and variations.
- Trust: Trust needs to be developed for the users to use the model. It is difficult to create trust among patients while dealing with a healthcare problem.
- Compliance: The model should be fit with all the relevant laws and regulations, such as Central Drug Standard Control Organization, Ministry of Health, etc.
- User friendly interface: The interface should be easy to use and understand by the user. The model should not ask inputs for which the user does not have answers.

Activity 3: Literature Survey (Student Will Write)

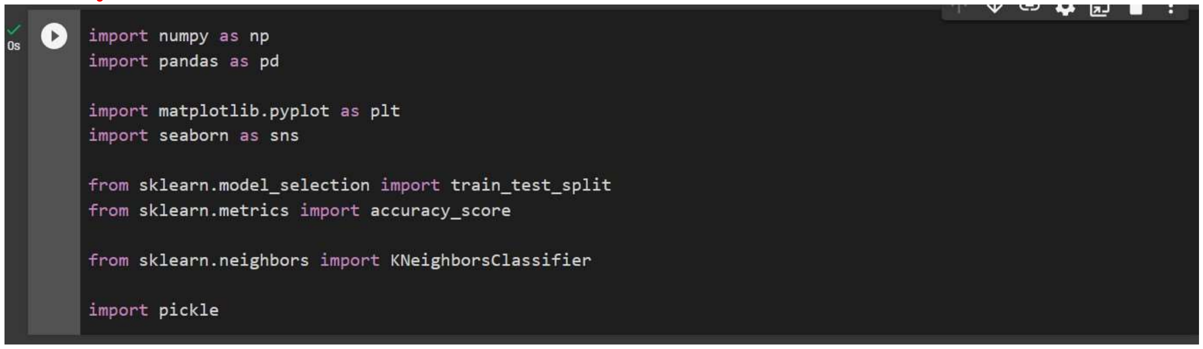
A literature survey would involve researching and reviewing existing studies, articles, and other publications on the topic of project. The survey would aim to gather information on current systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous 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

Activity 1: IMPORTING THE LIBRARIES:



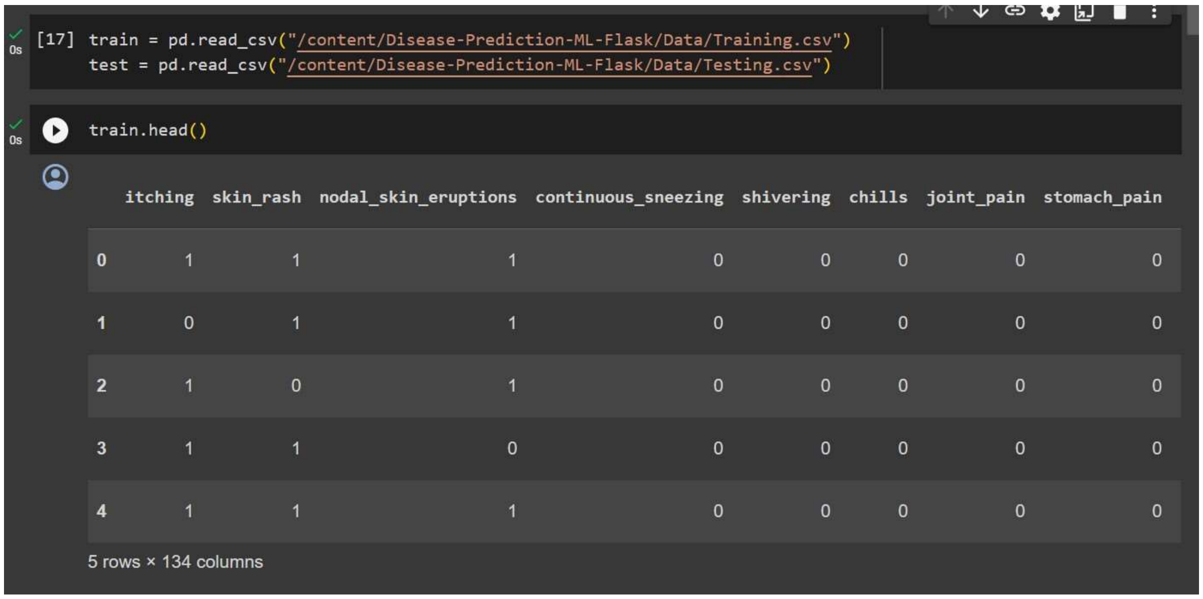
```
import numpy as np
import pandas as pd

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

import pickle
```

Here we are importing the pandas library as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns, and various modules from scikit-learn and imbalancedlearn libraries for the tasks such as model selection, preprocessing, and machine learning.

Activity 1.1: 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

1) The train.head() command is used to display the first few rows of Pandas DataFrame names train.



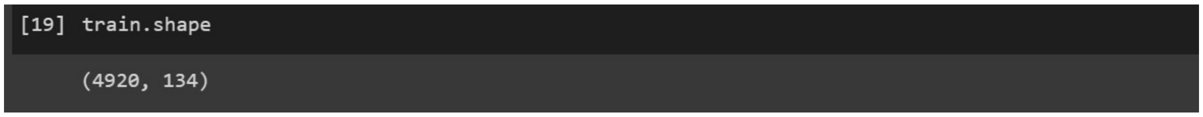
```
[17] train = pd.read_csv("/content/Disease-Prediction-ML-Flask/Data/Training.csv")
     test = pd.read_csv("/content/Disease-Prediction-ML-Flask/Data/Testing.csv")
```

```
train.head()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain
0	1	1	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0
2	1	0	1	0	0	0	0	0
3	1	1	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0

5 rows × 134 columns

2)train.shape is used to display the number of rows and columns.

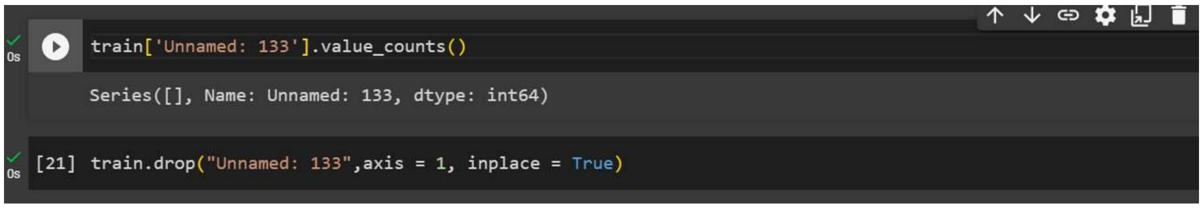


```
[19] train.shape

     (4920, 134)
```

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

Activity 2: DATA PREPARATION

Activity 2.1: Removing Redundant Columns

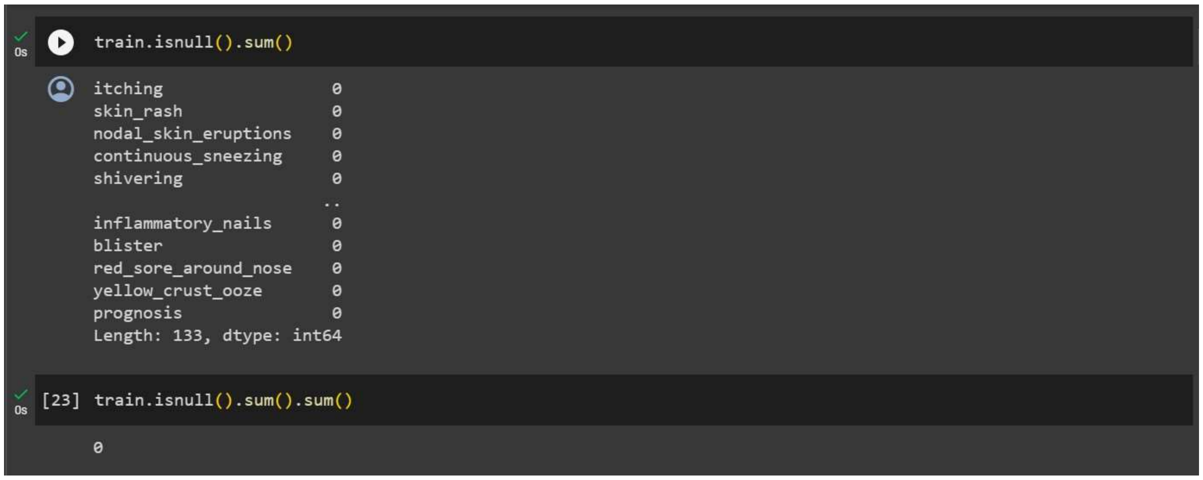


```
train['Unnamed: 133'].value_counts()

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

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

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

Activity 2.2: Handling Missing Values

train.isnull().sum(): The train.isnull().sum() expression is used to calculate the total number of null (missing) values in each column of the DataFrame train.



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

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

[23] train.isnull().sum().sum()

     0
```

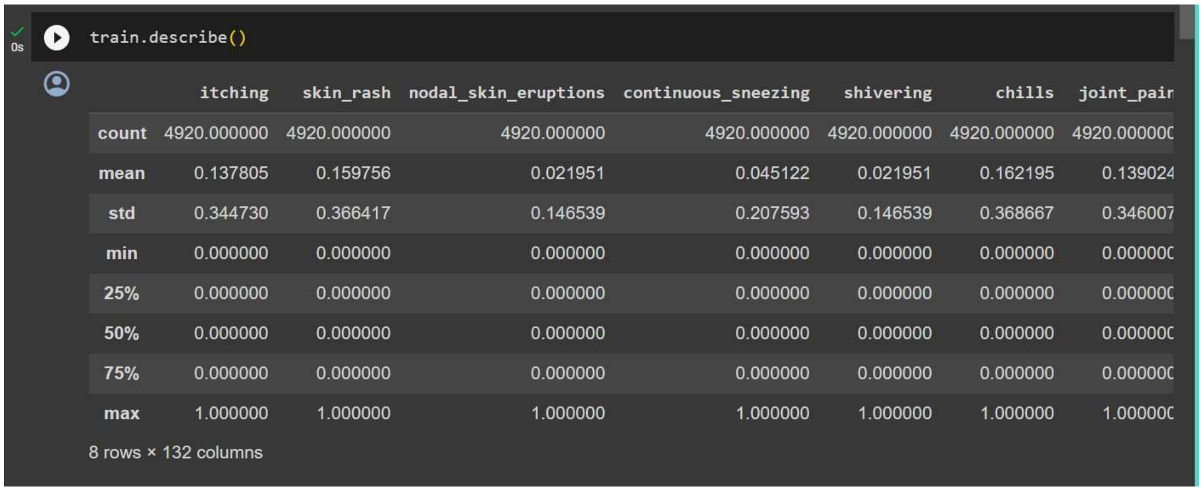
There are no missing values in this dataset. That is why we can skip this step

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.

The train.describe() is a Pandas DataFrame method that provides statistical summaries of the numerical columns in the DataFrame.



```
train.describe()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pair
count	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.00000C
mean	0.137805	0.159756	0.021951	0.045122	0.021951	0.162195	0.139024
std	0.344730	0.366417	0.146539	0.207593	0.146539	0.368667	0.346007
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000C
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000C
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000C
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000C
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000C

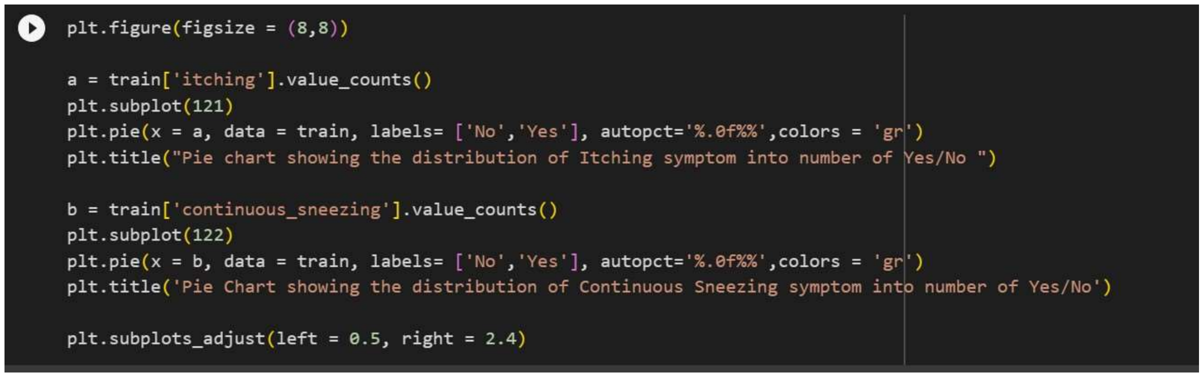
8 rows × 132 columns

Activity 2: Visual Analysis

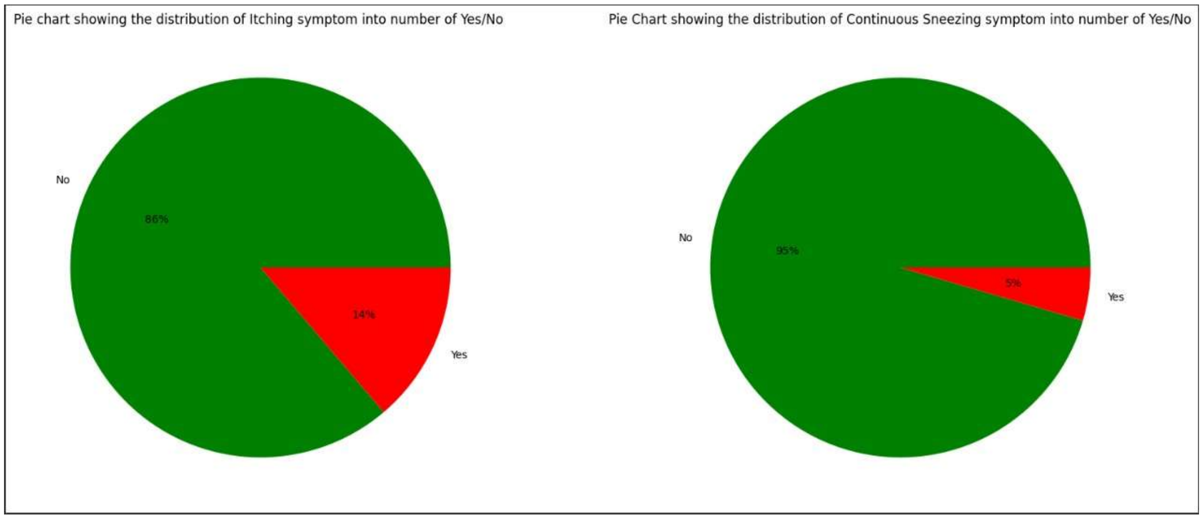
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate Analysis:

In simple words, univariate analysis is understanding the data with a single feature. We have displayed three different types of graphs and plots. For simple visualizations we can use the matplotlib.pyplot library



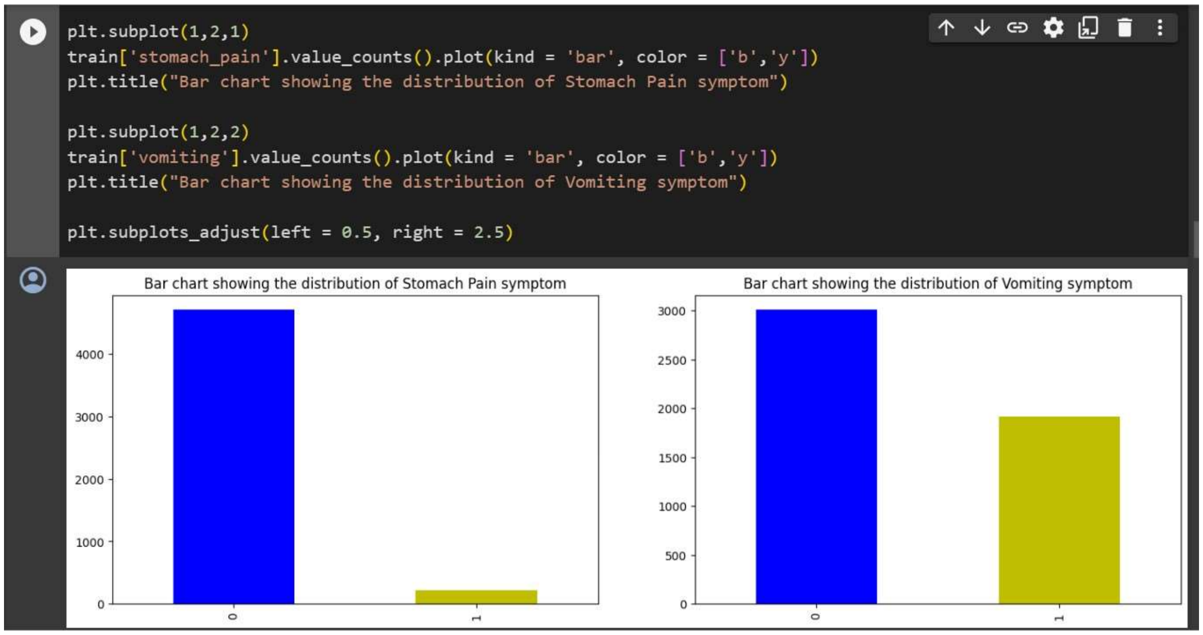
```
plt.figure(figsize = (8,8))

a = train['itching'].value_counts()
plt.subplot(121)
plt.pie(x = a, data = train, labels= ['No','Yes'], autopct='%.0f%%',colors = 'gr')
plt.title("Pie chart showing the distribution of Itching symptom into number of Yes/No ")

b = train['continuous_sneezing'].value_counts()
plt.subplot(122)
plt.pie(x = b, data = train, labels= ['No','Yes'], autopct='%.0f%%',colors = 'gr')
plt.title('Pie Chart showing the distribution of Continuous Sneezing symptom into number of Yes/No')

plt.subplots_adjust(left = 0.5, right = 2.4)
```



Here the plt.figure() command is used to determine the size of the plot. Then we divide the space for 2 pie plots.

The pie plot on the left 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 right 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.

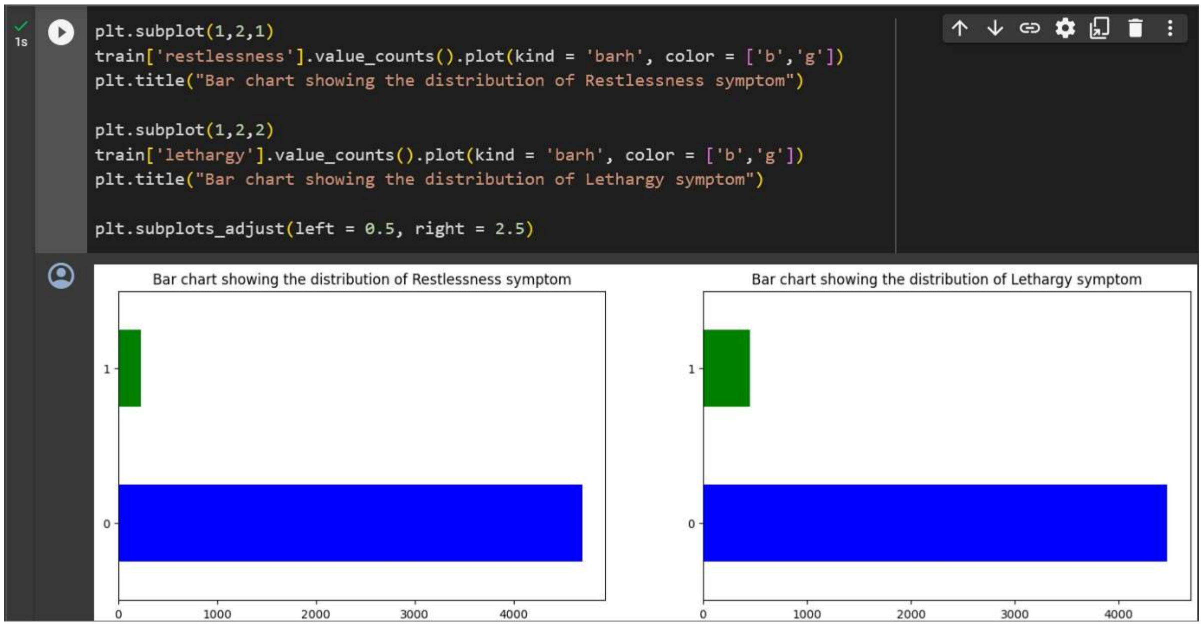


```
plt.subplot(1,2,1)
train['stomach_pain'].value_counts().plot(kind = 'bar', color = ['b','y'])
plt.title("Bar chart showing the distribution of Stomach Pain symptom")

plt.subplot(1,2,2)
train['vomiting'].value_counts().plot(kind = 'bar', color = ['b','y'])
plt.title("Bar chart showing the distribution of Vomiting symptom")

plt.subplots_adjust(left = 0.5, right = 2.5)
```

Bar chart showing the distribution of Stomach Pain symptom
4000
3000
2000
1000
0
0
1
Bar chart showing the distribution of Vomiting symptom
3000
2500
2000
1500
1000
500
0
0
1

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

The bar graph on the left shows the distribution of stomach pain symptom values. We can see that the 0 value has count of around 4700 and the 1 value has count of around 400. The graph on the right shows the distribution of vomiting symptom values. We can see that the 0 value has count of around 3000 and the 1 value has count of around 2000.



```
plt.subplot(1,2,1)
train['restlessness'].value_counts().plot(kind = 'barh', color = ['b','g'])
plt.title("Bar chart showing the distribution of Restlessness symptom")

plt.subplot(1,2,2)
train['lethargy'].value_counts().plot(kind = 'barh', color = ['b','g'])
plt.title("Bar chart showing the distribution of Lethargy symptom")

plt.subplots_adjust(left = 0.5, right = 2.5)
```

Bar chart showing the distribution of Restlessness symptom
Bar chart showing the distribution of Lethargy symptom
1
0
0
1000
2000
3000
4000

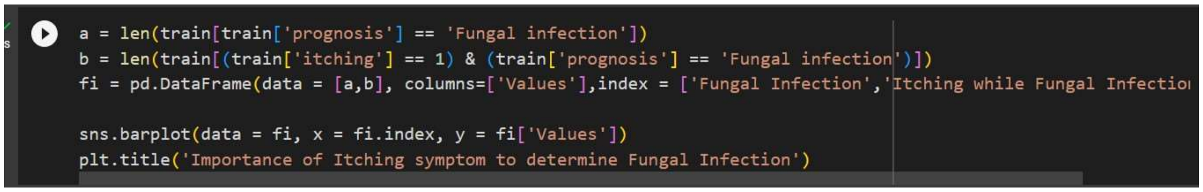
Here we have plotted 2 horizontal bar graphs. These bar graphs can be plotted without using any external library. We divide the plot into two subplots using subplot function from the matplotlib.pyplot library. We have plotted the bar graph using the inbuilt plot function in python.

The bar graph on the left 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.

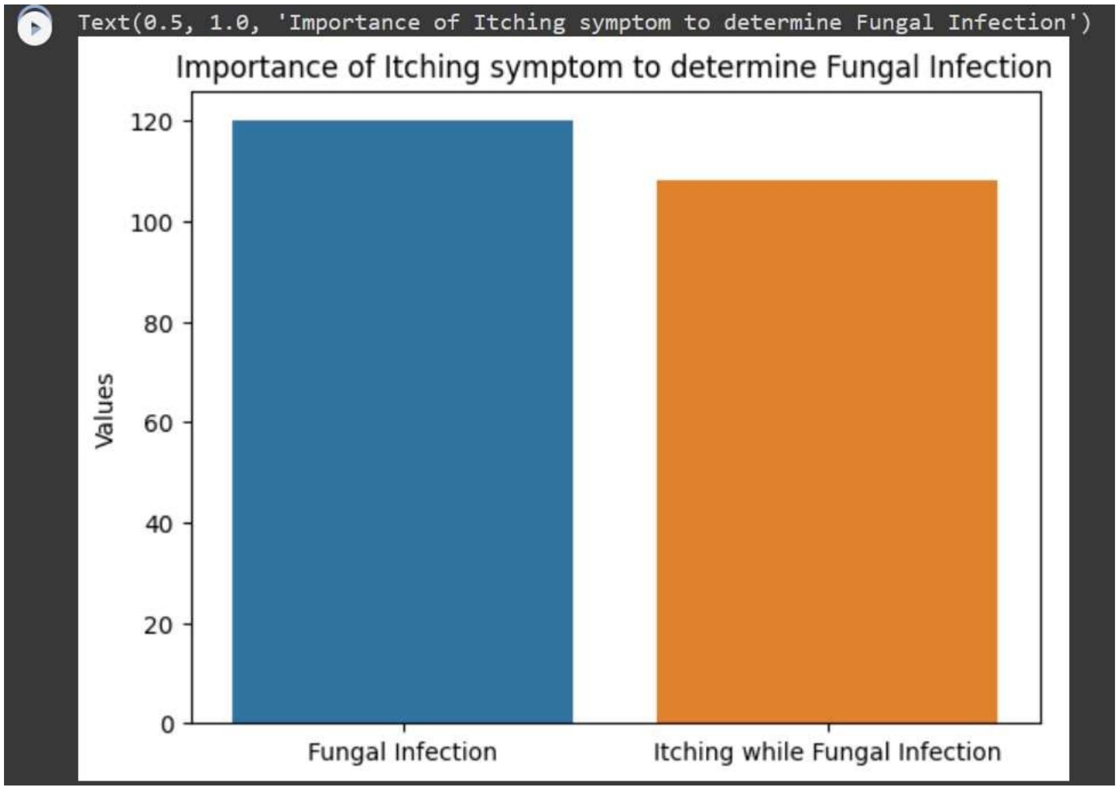
The graph on the right shows the distribution of vomiting symptom values. We can see that the 0 value has count of around 4500 and the 1 value has count of around 400.

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



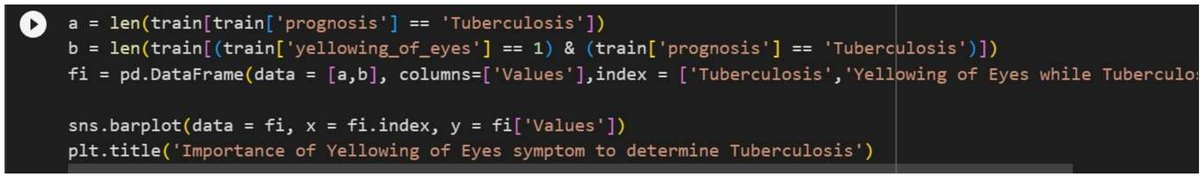
```
a = len(train[train['prognosis'] == 'Fungal infection'])
b = len(train[(train['itching'] == 1) & (train['prognosis'] == 'Fungal infection')])
fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Fungal Infection','Itching while Fungal Infectio

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')
```

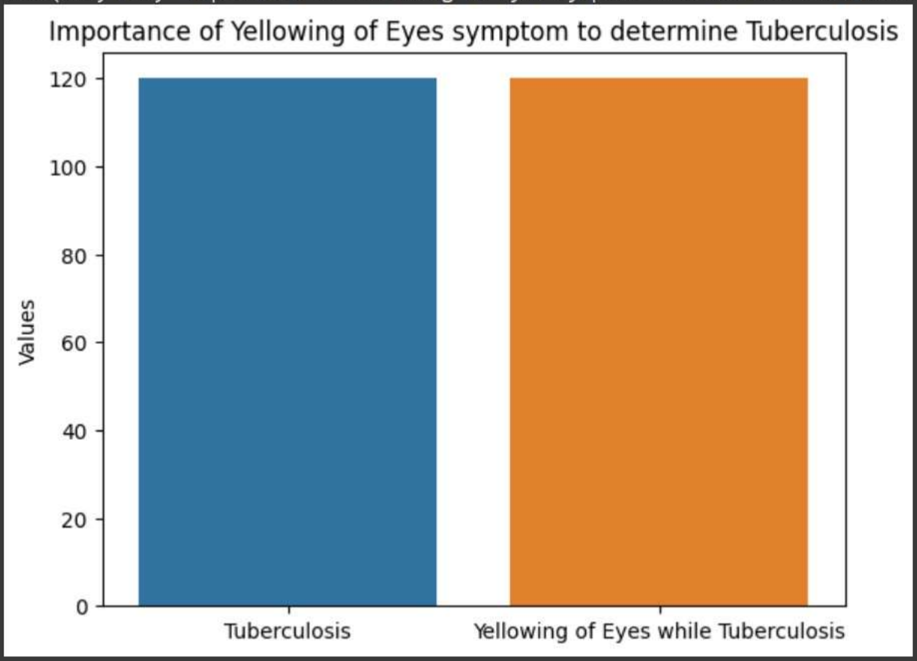
Here we use Boolean Indexing to filter out values from the prognosis column where the values are ‘Fungal Infection’. These observations are stored in variable ‘a’. Alse we filter out values from the prognosis where values are ‘Fungal Infection’ and also the values of Itching variable are 1. From the 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.

Next 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



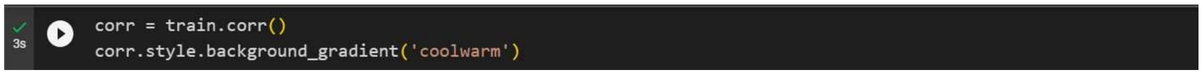
```
a = len(train[train['prognosis'] == 'Tuberculosis'])
b = len(train[(train['yellowing_of_eyes'] == 1) & (train['prognosis'] == 'Tuberculosis')])
fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Tuberculosis','Yellowing of Eyes while Tuberculo

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

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

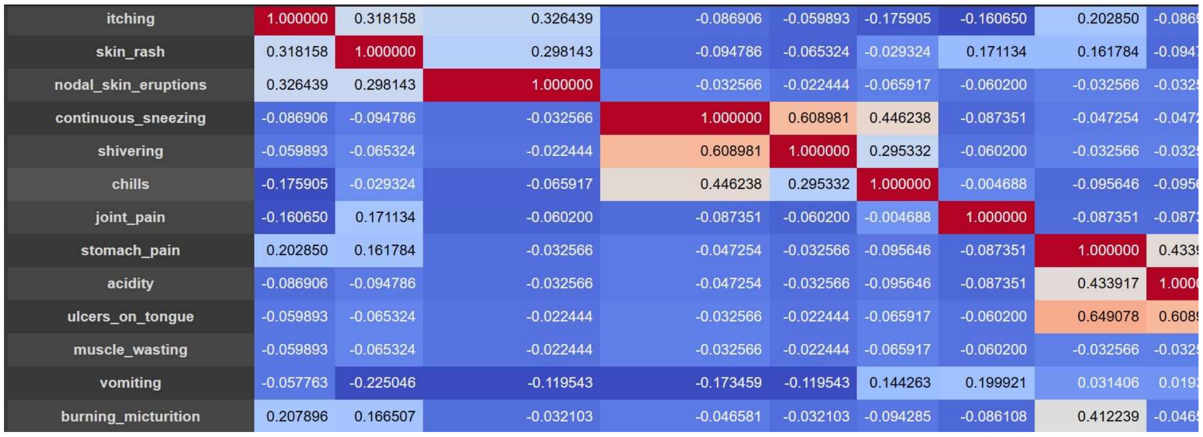


Activity 2.3: Multivariate Analysis

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



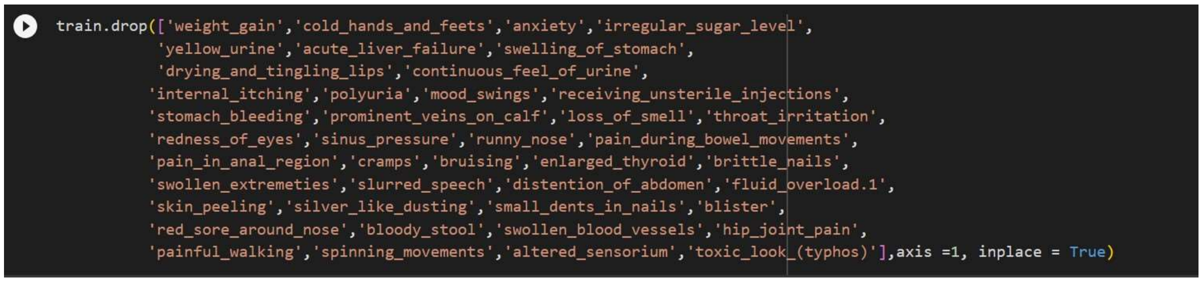
```
corr = train.corr()
corr.style.background_gradient('coolwarm')
```



itching	1.000000	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.160650	0.202850	-0.0869
skin_rash	0.318158	1.000000	0.298143	-0.094786	-0.065324	-0.029324	0.171134	0.161784	-0.0947
nodal_skin_eruptions	0.326439	0.298143	1.000000	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.0325
continuous_sneezing	-0.086906	-0.094786	-0.032566	1.000000	0.608981	0.446238	-0.087351	-0.047254	-0.0472
shivering	-0.059893	-0.065324	-0.022444	0.608981	1.000000	0.295332	-0.060200	-0.032566	-0.0325
chills	-0.175905	-0.029324	-0.065917	0.446238	0.295332	1.000000	-0.004688	-0.095646	-0.0956
joint_pain	-0.160650	0.171134	-0.060200	-0.087351	-0.060200	-0.004688	1.000000	-0.087351	-0.0873
stomach_pain	0.202850	0.161784	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	1.000000	0.4339
acidity	-0.086906	-0.094786	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	0.433917	1.0000
ulcers_on_tongue	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.649078	0.6089
muscle_wasting	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.0325
vomiting	-0.057763	-0.225046	-0.119543	-0.173459	-0.119543	0.144263	0.199921	0.031406	0.0193
burning_micturition	0.207896	0.166507	-0.032103	-0.046581	-0.032103	-0.094285	-0.086108	0.412239	-0.0465

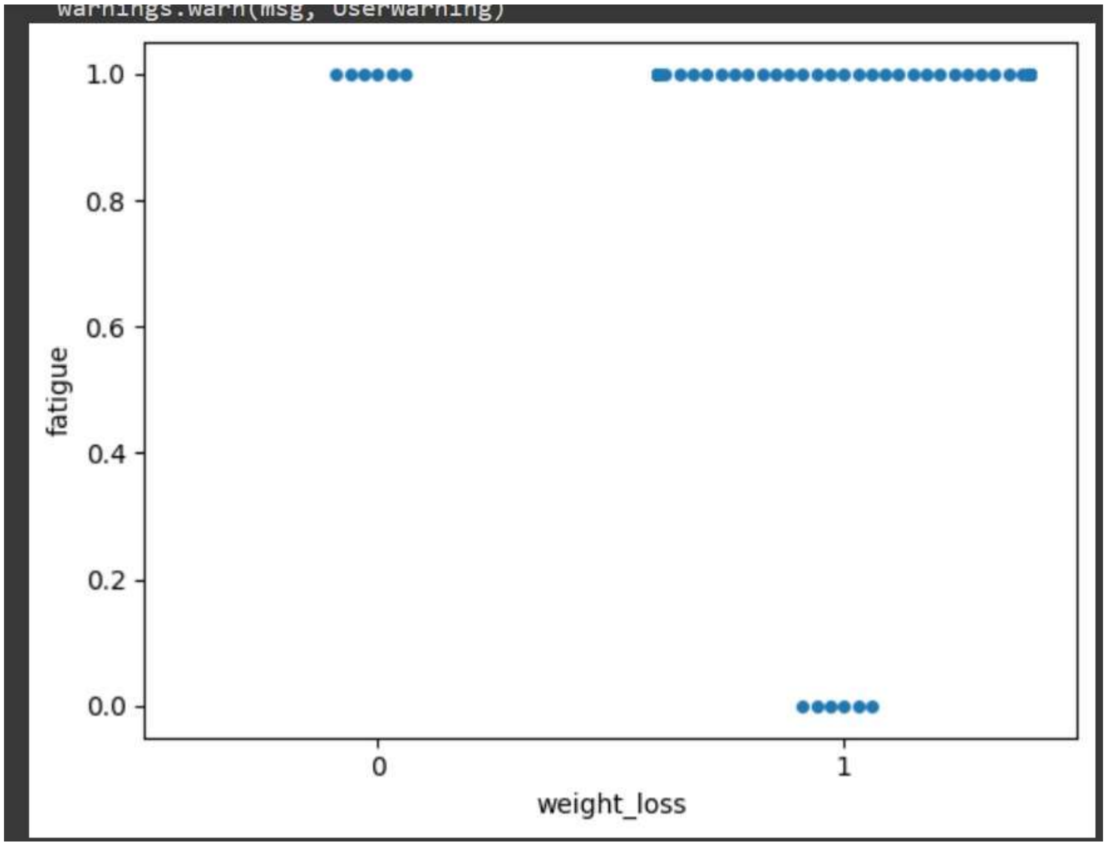
As we have 131 columns which have numerical values the correlation matrix is of dimensions 131 X 131. These many features can only be parsed by scrolling.

From the correlation matrix we try to remove the values which are highly correlated with each other. When 2 values are highly correlated with each other, we can only remove one of them. We remove columns where the correlation between the columns is above 0.9



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

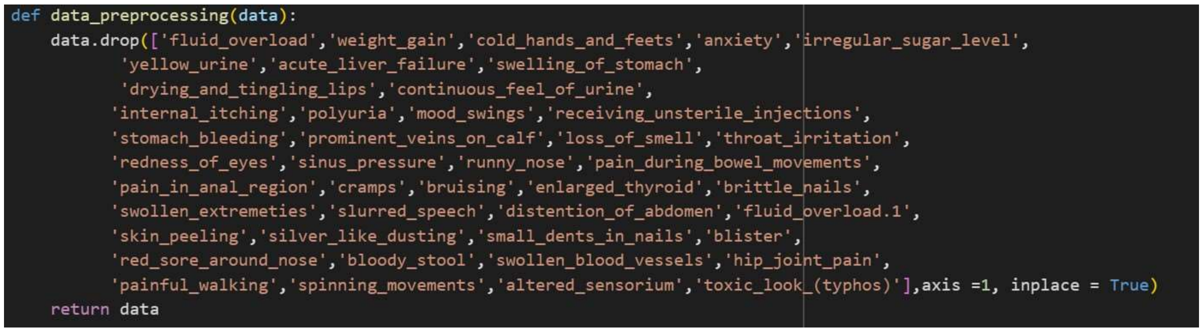
We can see the columns that are removed due to high correlation. For multivariate analysis we will also plot a swarmplot of weightloss and fatigue column where the prognosis column is Tuberculosis.


```
warnings.warn(msg, UserWarning)
```

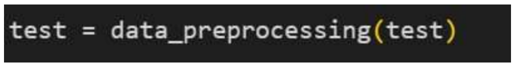
From this swarm plot we can see that for Tuberculosis disease, there is no observation when the fatigue and weight loss is 0. There are some cases when there is only either of the two, but for Tuberculosis there is a high chance that the patient will have fatigue and weight loss as symptoms.

Preprocessing of Test data

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


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

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



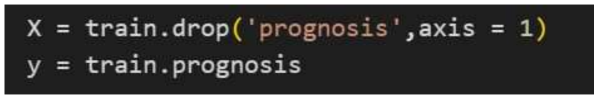
```
test = data_preprocessing(test)
```

Here we call the function for the test data

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

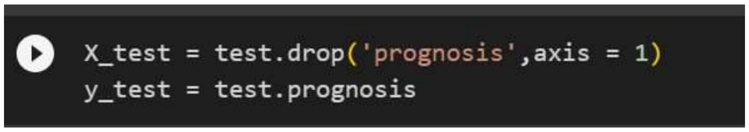
We have training and testing data given separately. We further split the training data into training and validation data. This validation data can be used for hyper parameter tuning.

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



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

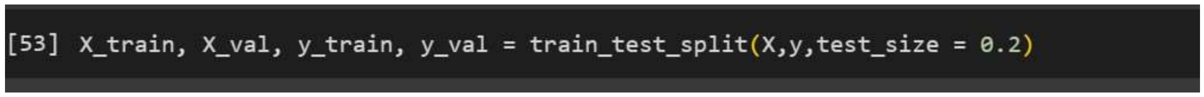
We split the training data into features(X) and target variable(y).



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

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

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



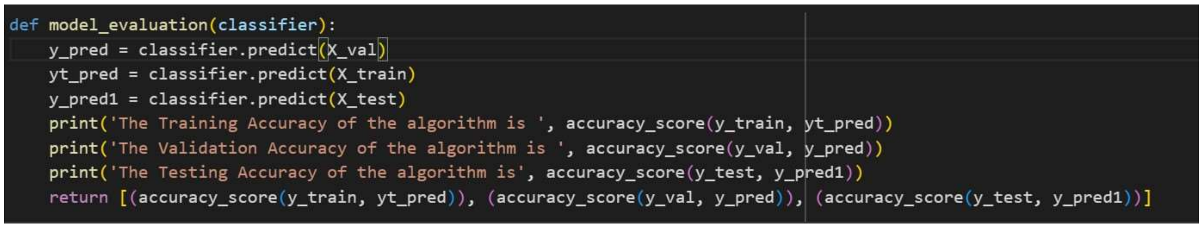
```
[53] X_train, X_val, y_train, y_val = train_test_split(X,y,test_size = 0.2)
```

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

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



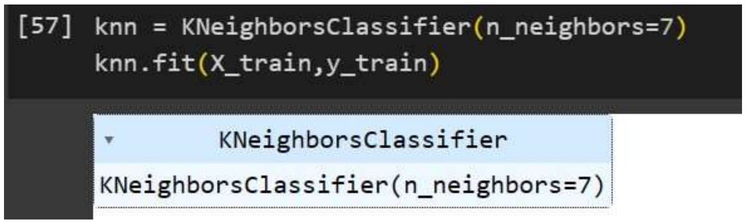
```
def model_evaluation(classifier):
    y_pred = classifier.predict(X_val)
    yt_pred = classifier.predict(X_train)
    y_pred1 = classifier.predict(X_test)
    print('The Training Accuracy of the algorithm is ', accuracy_score(y_train, yt_pred))
    print('The Validation Accuracy of the algorithm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

Activity 2: Training and testing the models using multiple algorithms

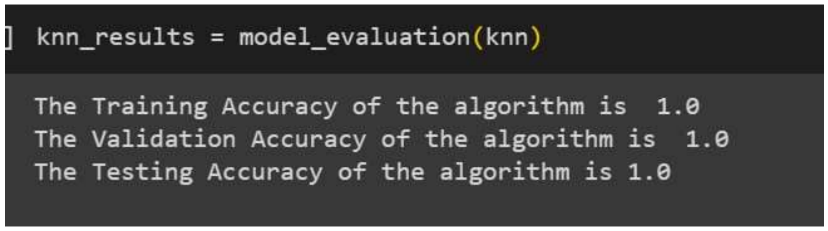
Now that we have clean data, a function for evaluation it is time to build models to train the data. For this project we will be using 4 different classification algorithms to build our models. The best model will be used for prediction.

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



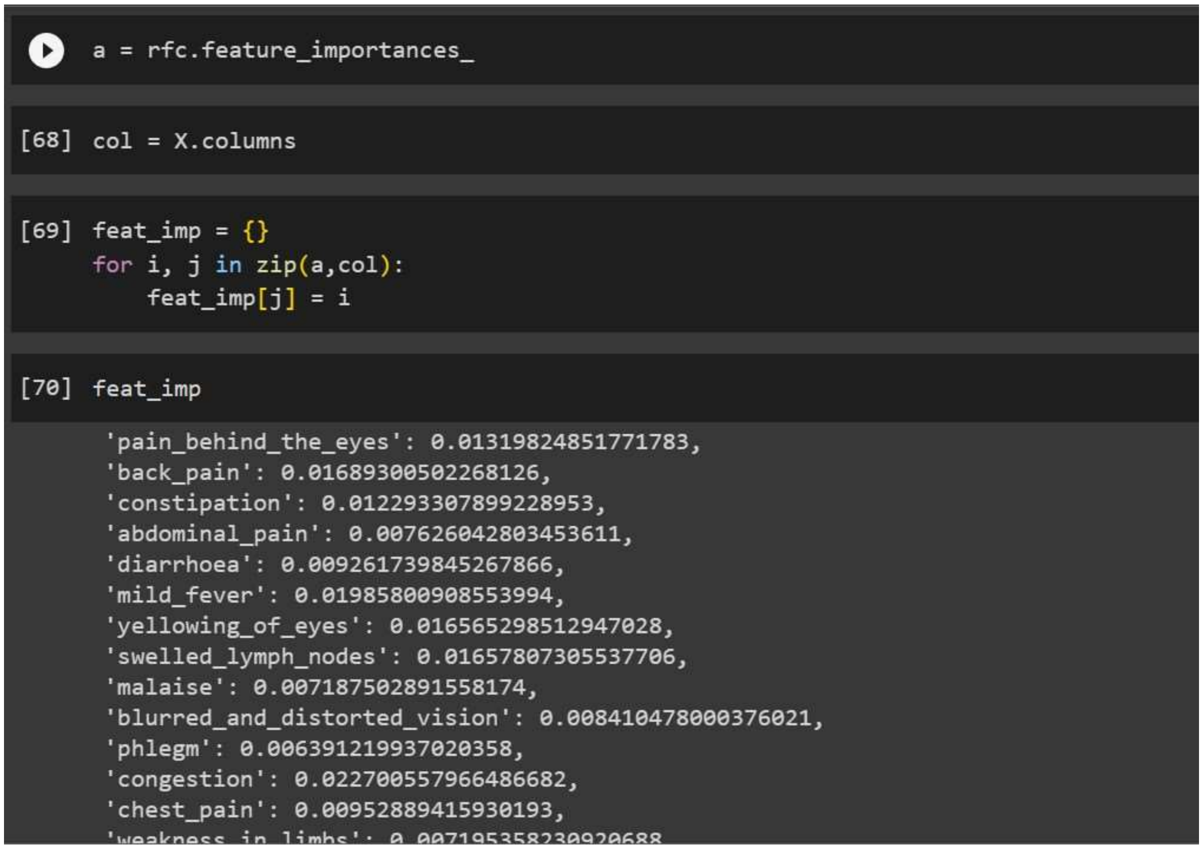
```
[57] knn = KNeighborsClassifier(n_neighbors=7)
     knn.fit(X_train,y_train)

     KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```



```
] 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.

Milestone 5 : Performance Testing & Hyperparameter Tuning

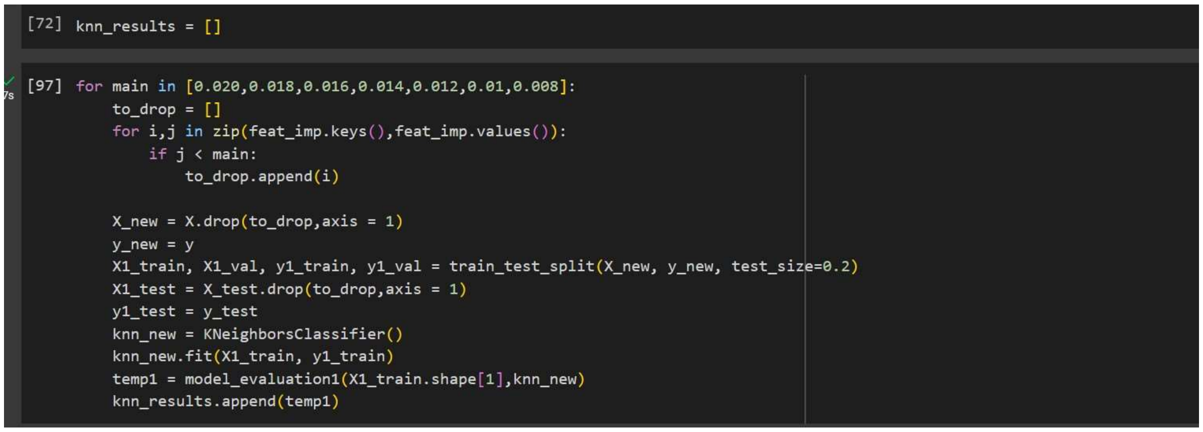


```
a = rfc.feature_importances_

[68] col = X.columns

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

[70] feat_imp

       'pain_behind_the_eyes': 0.01319824851771783,
       'back_pain': 0.01689300502268126,
       'constipation': 0.012293307899228953,
       'abdominal_pain': 0.007626042803453611,
       'diarrhoea': 0.009261739845267866,
       'mild_fever': 0.01985800908553994,
       'yellowing_of_eyes': 0.016565298512947028,
       'swelled_lymph_nodes': 0.01657807305537706,
       'malaise': 0.007187502891558174,
       'blurred_and_distorted_vision': 0.008410478000376021,
       'phlegm': 0.006391219937020358,
       'congestion': 0.022700557966486682,
       'chest_pain': 0.00952889415930193,
       'weakness_in_limbs': 0.007195358230920688
```

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.



```
[72] knn_results = []

[97] 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
         knn_new = KNeighborsClassifier()
         knn_new.fit(X1_train, y1_train)
         temp1 = model_evaluation1(X1_train.shape[1],knn_new)
         knn_results.append(temp1)
```

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.

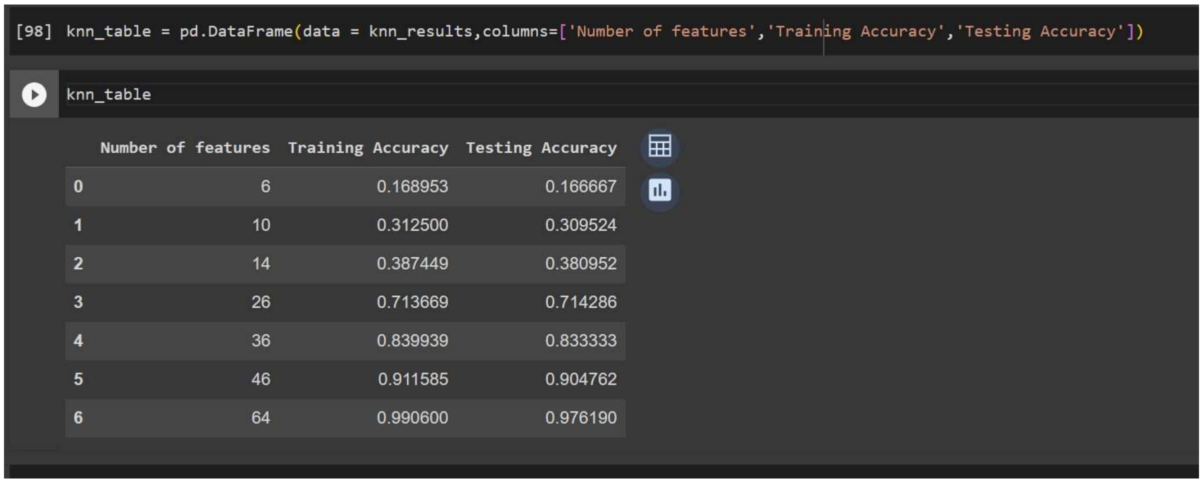
The columns whose name is in the to_drop list will be dropped.

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

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.



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

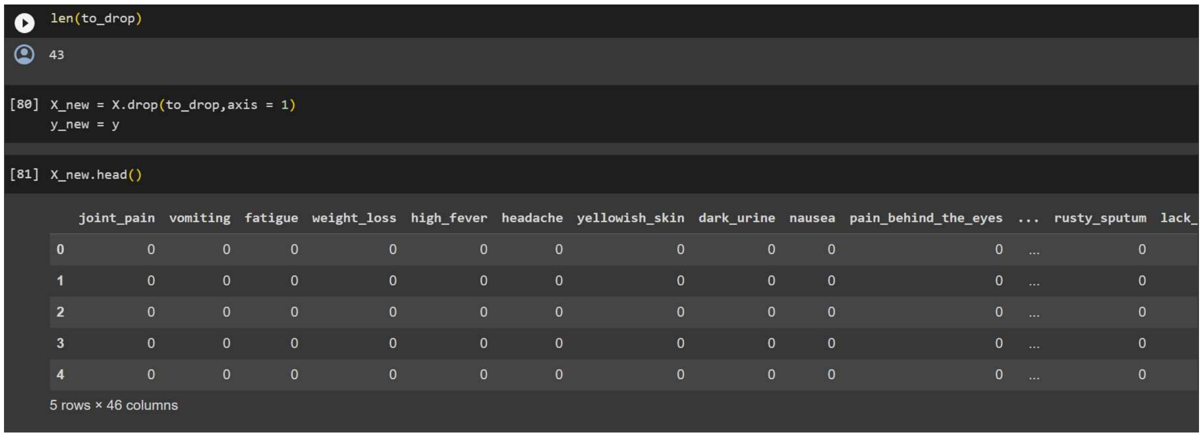
```
knn_table
```

	Number of features	Training Accuracy	Testing Accuracy
0	6	0.168953	0.166667
1	10	0.312500	0.309524
2	14	0.387449	0.380952
3	26	0.713669	0.714286
4	36	0.839939	0.833333
5	46	0.911585	0.904762
6	64	0.990600	0.976190

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

Activity 4: Building Model with appropriate features

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



```
len(to_drop)
```

```
43
```

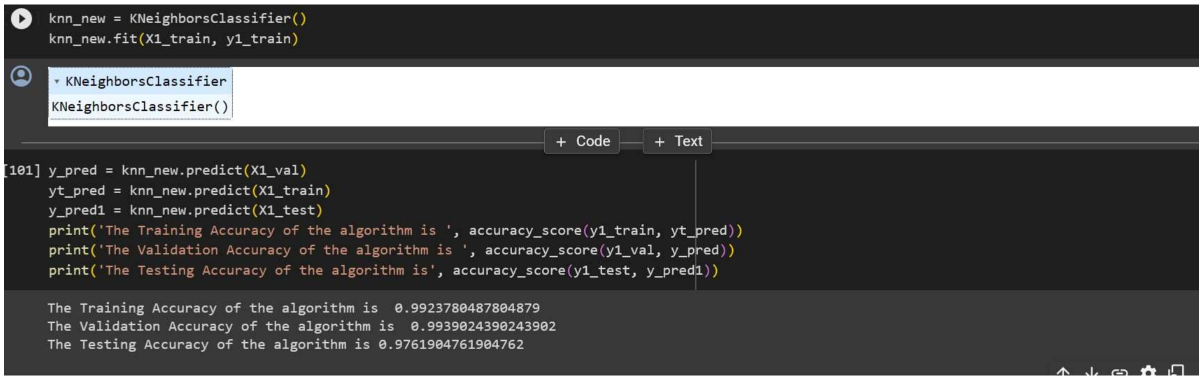
```
[80] X_new = X.drop(to_drop,axis = 1)
     y_new = y
```

```
[81] X_new.head()
```

	joint_pain	vomiting	fatigue	weight_loss	high_fever	headache	yellowish_skin	dark_urine	nausea	pain_behind_the_eyes	...	rusty_sputum	lack_
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 46 columns

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



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

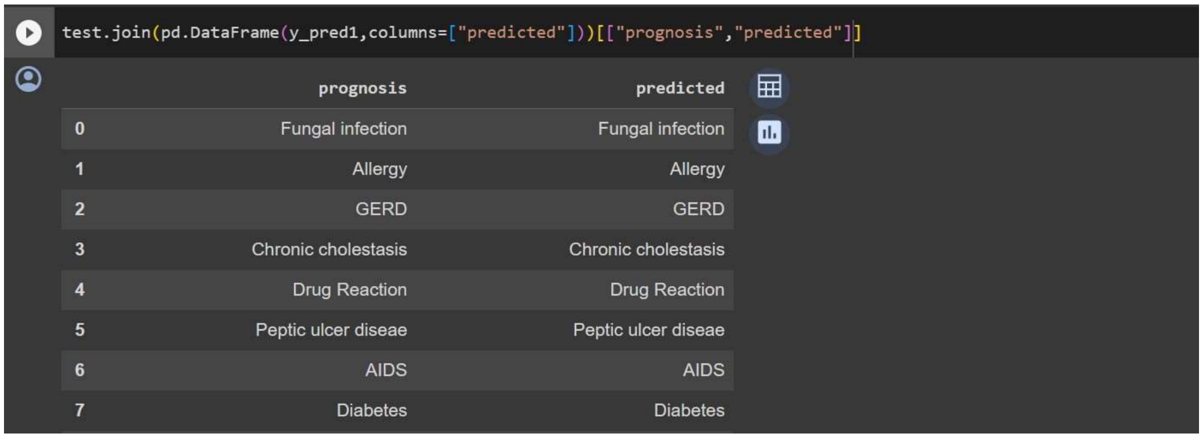
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
[101] 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  0.9923780487804879
The Validation Accuracy of the algorithm is  0.9939024390243902
The Testing Accuracy of the algorithm is 0.9761904761904762
```

AFTER TRAINING THE KNN MODEL WE CHECK THE ACCURACIES. OUR MODEL HAS ACHIEVED 97.6 % ACCURACY FOR THE TEST DATA

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



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

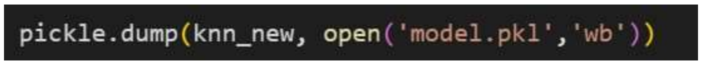
	prognosis	predicted
0	Fungal infection	Fungal infection
1	Allergy	Allergy
2	GERD	GERD
3	Chronic cholestasis	Chronic cholestasis
4	Drug Reaction	Drug Reaction
5	Peptic ulcer diseae	Peptic ulcer diseae
6	AIDS	AIDS
7	Diabetes	Diabetes

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.



```
pickle.dump(knn_new, open('model.pkl','wb'))
```

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

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:

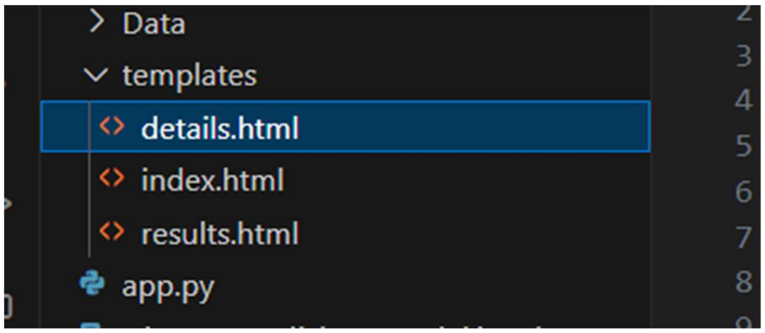
For this project we create three HTML files namely

● Index.html

● Details.html

● Results.html

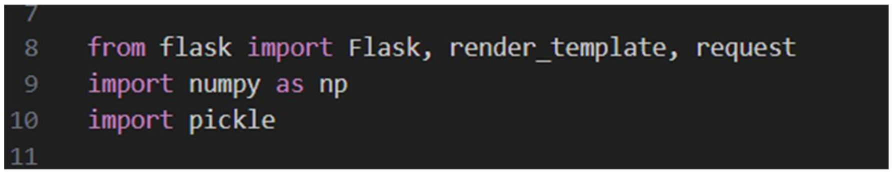
And we will save them in the templates folder.



Activity 2.2: Build Python code

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

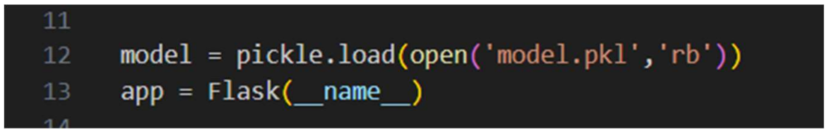
Import the necessary Libraries.



```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.

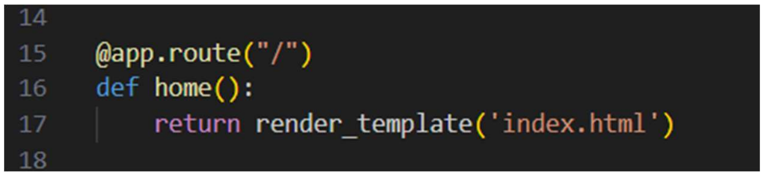


```python
model = pickle.load(open('model.pkl','rb'))
app = Flask(__name__)
```

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.

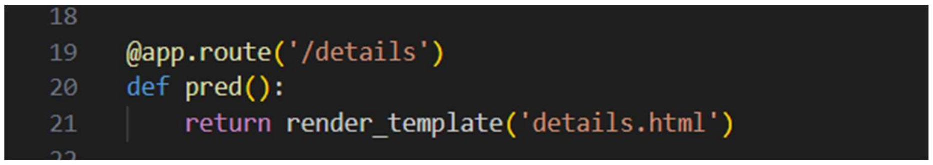
The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

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 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.

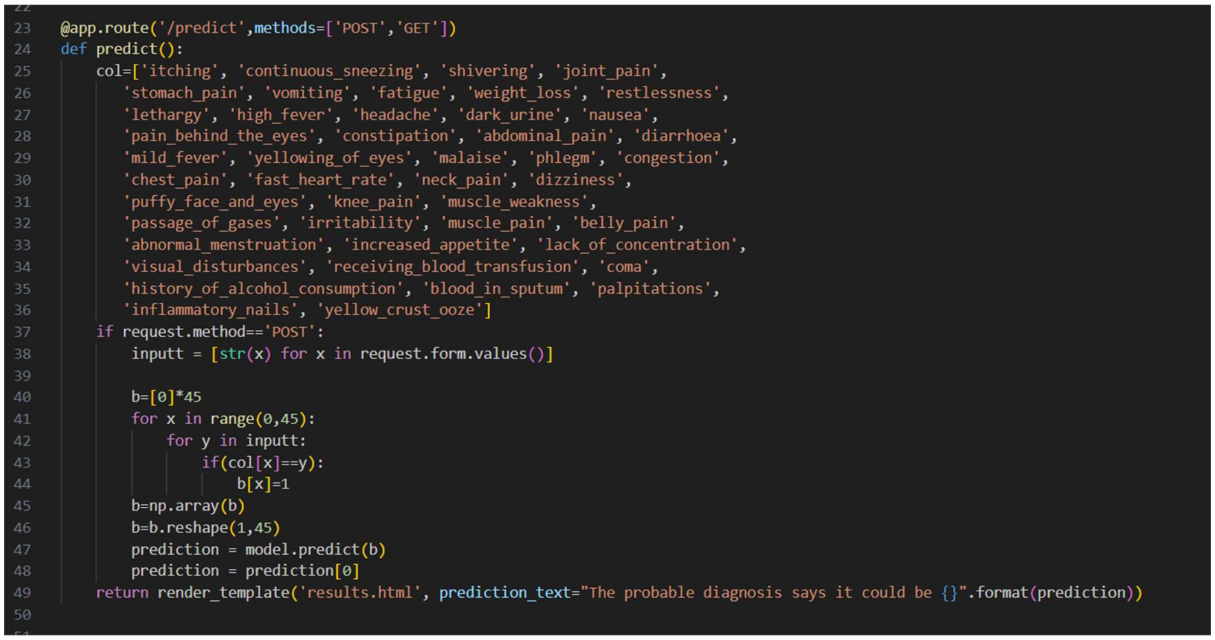
Another for loop is written where y will iterate over the values given by the user as inputs.

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.

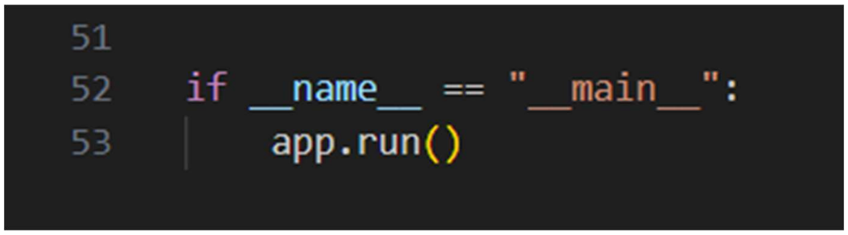
This prediction is returned to the results.html page using render_



```python
@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:

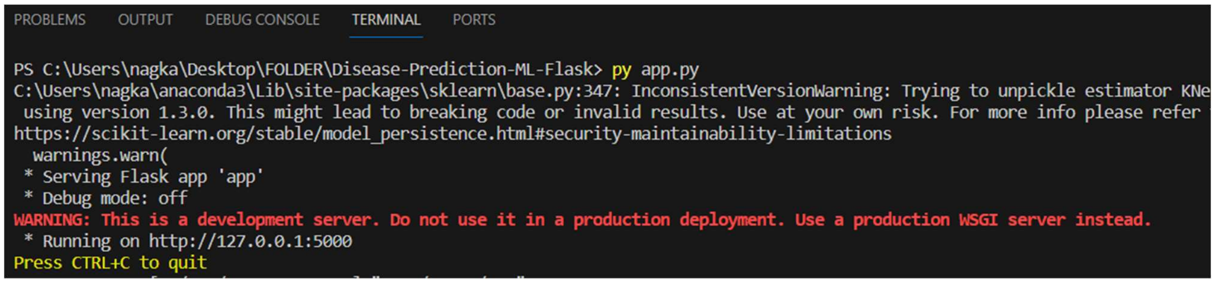
This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask deployment server.



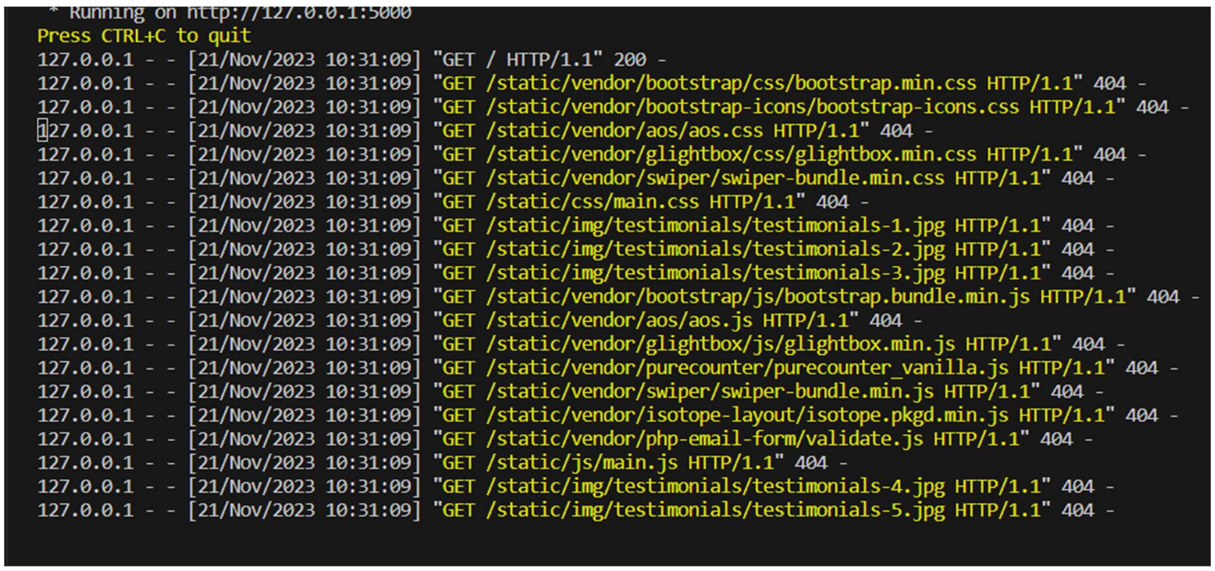
```python
if __name__ == "__main__":
    app.run()
```

Activity 2.3: Run the Web Application

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



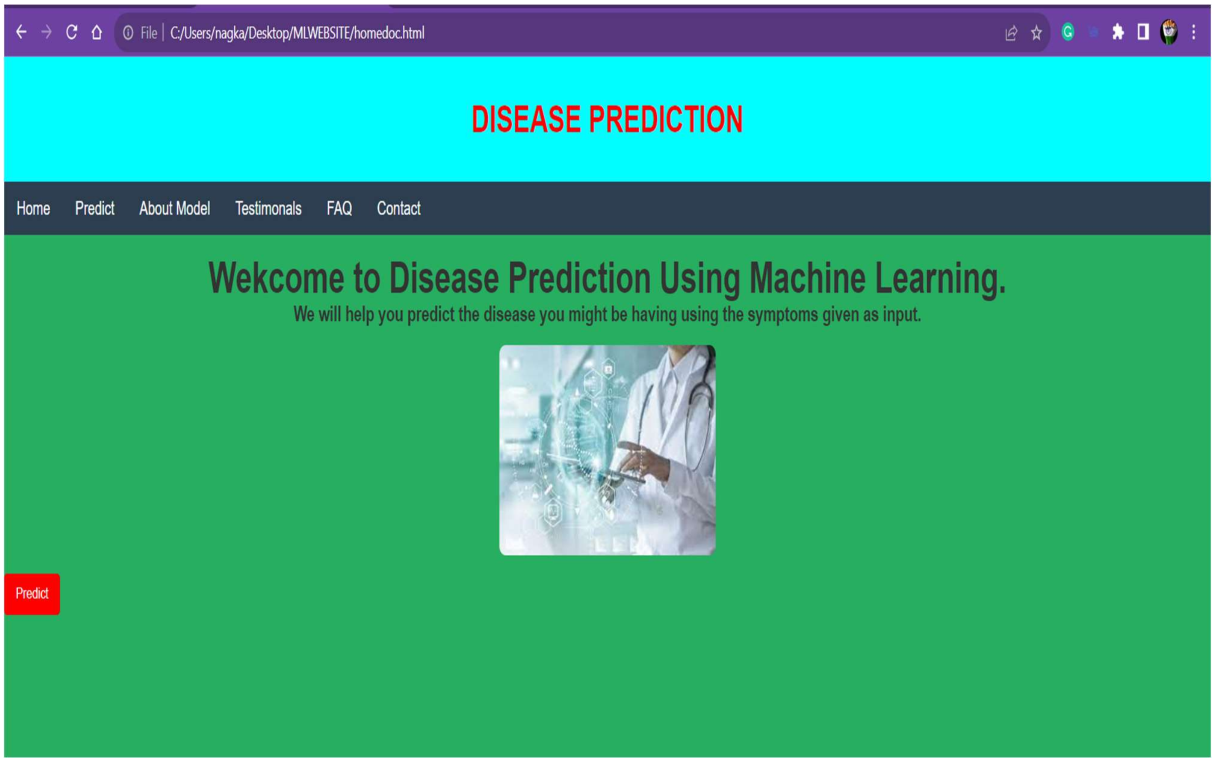
```
PS C:\Users\nagka\Desktop\FOLDER\Disease-Prediction-ML-Flask> py app.py
C:\Users\nagka\anaconda3\Lib\site-packages\sklearn\base.py:347: InconsistentVersionWarning: Trying to unpickle estimator KNe
 using version 1.3.0. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```



```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/bootstrap/css/bootstrap.min.css HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/bootstrap-icons/bootstrap-icons.css HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/aos/aos.css HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/glightbox/css/glightbox.min.css HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/swiper/swiper-bundle.min.css HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/css/main.css HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/img/testimonials/testimonials-1.jpg HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/img/testimonials/testimonials-2.jpg HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/img/testimonials/testimonials-3.jpg HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/bootstrap/js/bootstrap.bundle.min.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/aos/aos.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/glightbox/js/glightbox.min.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/purecounter/purecounter_vanilla.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/swiper/swiper-bundle.min.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/isotope-layout/isotope.pkgd.min.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/vendor/php-email-form/validate.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/js/main.js HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/img/testimonials/testimonials-4.jpg HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 10:31:09] "GET /static/img/testimonials/testimonials-5.jpg HTTP/1.1" 404 -
```

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.

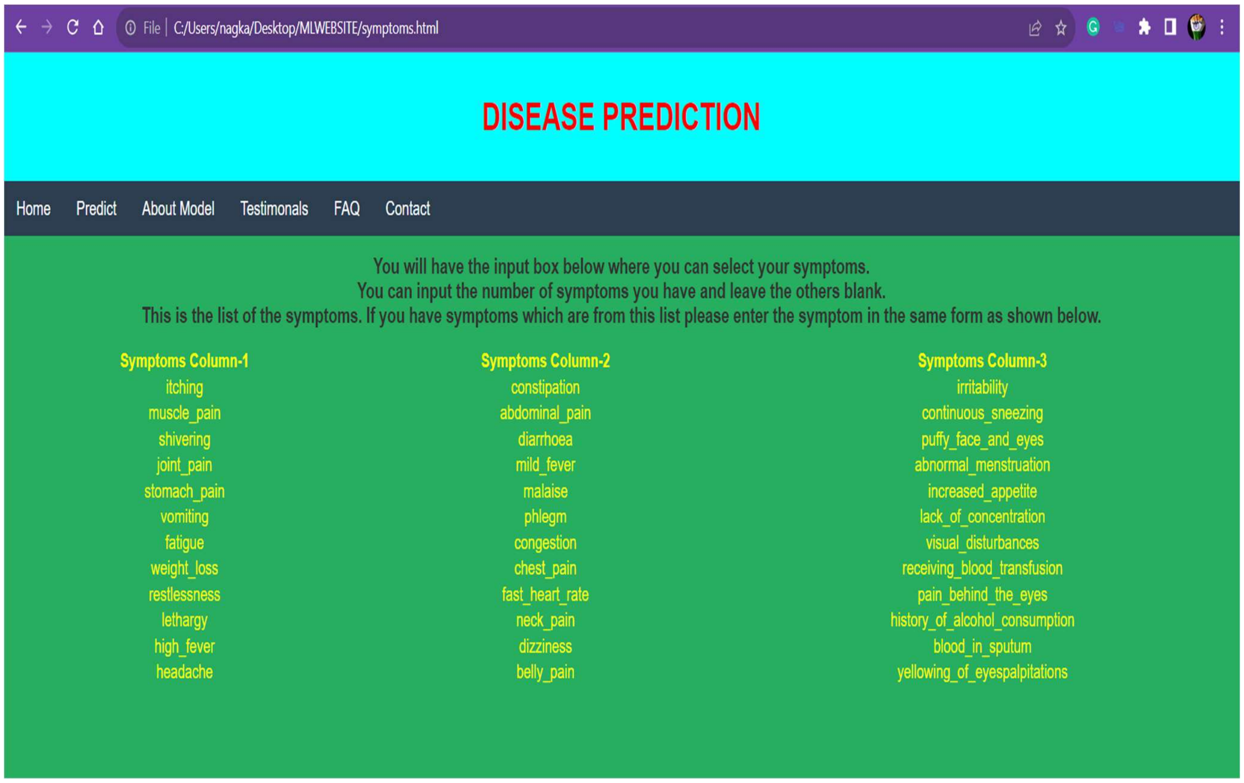
If you click on the Predict button on home page or in the header bar you will be redirected to the details.html page (homedoc.html)



Wekcome to Disease Prediction Using Machine Learning.

We will help you predict the disease you might be having using the symptoms given as input.

Our details.html will be shown as(symptoms.html)

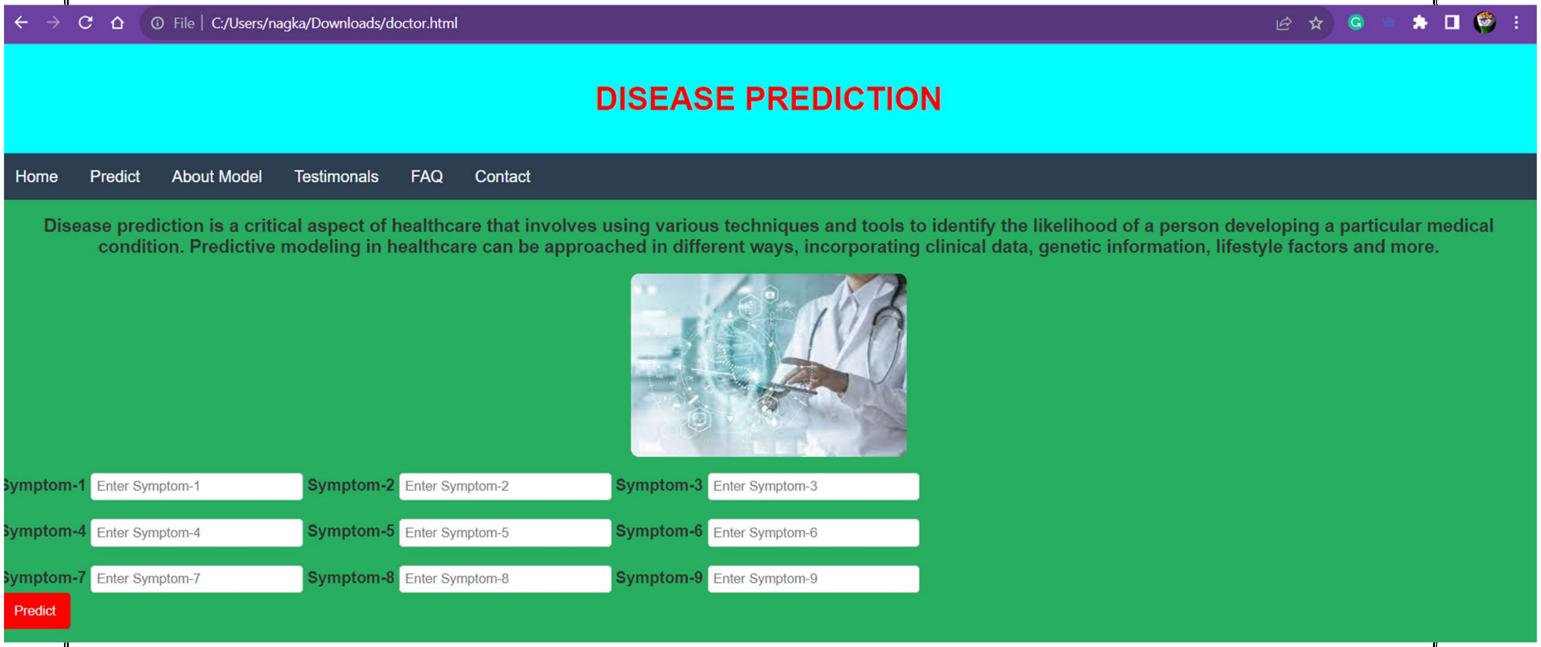


DISEASE PREDICTION

Home Predict About Model Testimonals FAQ Contact

You will have the input box below where you can select your symptoms.
You can input the number of symptoms you have and leave the 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.

Symptoms Column-1	Symptoms Column-2	Symptoms Column-3
itching	constipation	irritability
muscle_pain	abdominal_pain	continuous_sneezing
shivering	diarrhoea	puffy_face_and_eyes
joint_pain	mild_fever	abnormal_menstruation
stomach_pain	malaise	increased_appetite
vomiting	phlegm	lack_of_concentration
fatigue	congestion	visual_disturbances
weight_loss	chest_pain	receiving_blood_transfusion
restlessness	fast_heart_rate	pain_behind_the_eyes
lethargy	neck_pain	history_of_alcohol_consumption
high_fever	dizziness	blood_in_sputum
headache	belly_pain	yellowing_of_eyespalpitations



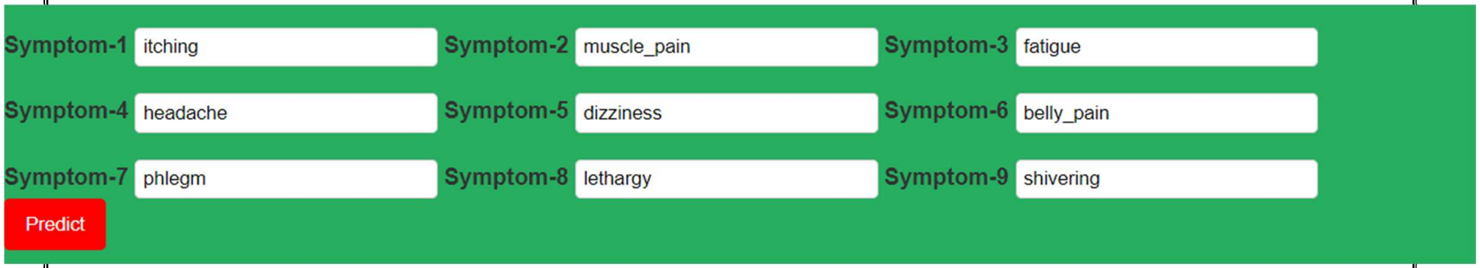
DISEASE PREDICTION

Home Predict About Model Testimonals FAQ Contact

Disease prediction is a critical aspect of healthcare that involves using various techniques and tools to identify the likelihood of a person developing a particular medical condition. Predictive modeling in healthcare can be approached in different ways, incorporating clinical data, genetic information, lifestyle factors and more.

ymptom-1 Enter Symptom-1	Symptom-2 Enter Symptom-2	Symptom-3 Enter Symptom-3
ymptom-4 Enter Symptom-4	Symptom-5 Enter Symptom-5	Symptom-6 Enter Symptom-6
ymptom-7 Enter Symptom-7	Symptom-8 Enter Symptom-8	Symptom-9 Enter Symptom-9

Input-1:

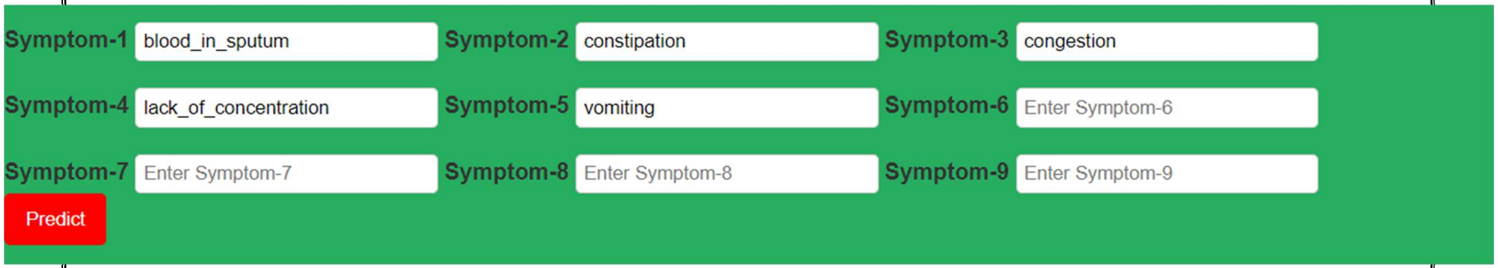


Symptom-1	itching	Symptom-2	muscle_pain	Symptom-3	fatigue
Symptom-4	headache	Symptom-5	dizziness	Symptom-6	belly_pain
Symptom-7	phlegm	Symptom-8	lethargy	Symptom-9	shivering

Output-1:

The probable diagnosis says it could be Chicken pox

Input-2:



Symptom-1	blood_in_sputum	Symptom-2	constipation	Symptom-3	congestion
Symptom-4	lack_of_concentration	Symptom-5	vomiting	Symptom-6	Enter Symptom-6
Symptom-7	Enter Symptom-7	Symptom-8	Enter Symptom-8	Symptom-9	Enter Symptom-9

Output-2:

The probable diagnosis says it could be Dimorphic hemmorhoids(piles)