Date	17-11-2023
ProjectName	Disease Prediction Using Machine Learning
Team Id	Team-592083

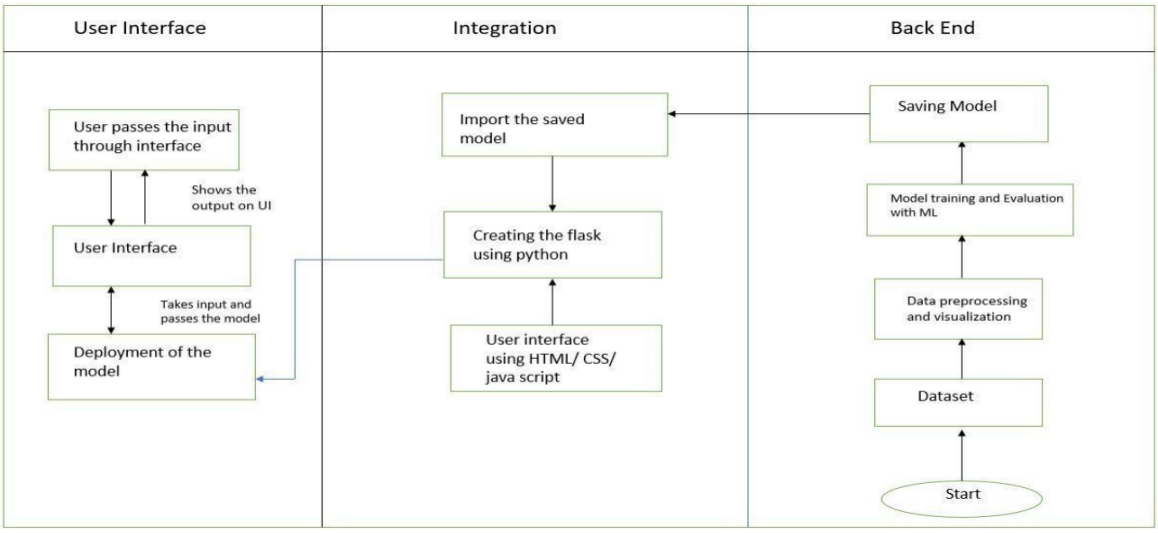
Disease Prediction Using Machine Learning

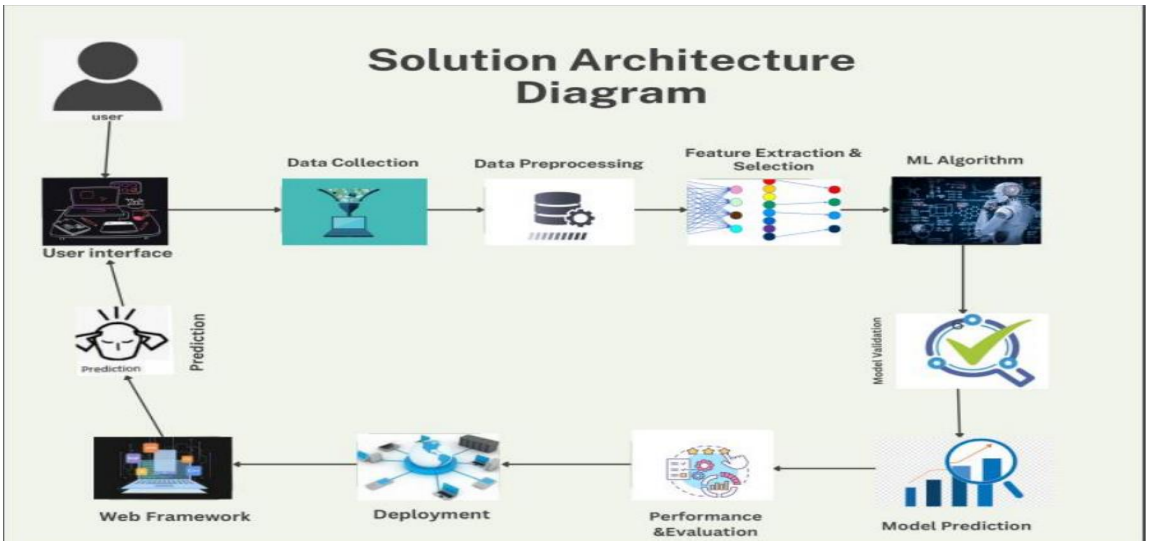
Disease prediction models are crucial for early intervention, enabling timely treatment and improved outcomes. They help preventative healthcare by identifying high-risk individuals and enabling for focused interventions and lifestyle changes. By focusing on high-risk groups, these models optimise healthcare resources, resulting in more efficient resource allocation. Early detection and intervention reduce the economic burden of advanced-stage illnesses, resulting in cost savings. Disease prediction helps with personalised medicine by adapting therapies to specific risk variables. Furthermore, aggregated data assists public health planning by directing targeted actions and policy development.

In today's face paced world, there is little to no time spared for healthcare. Even after developing severe symptoms to various diseases patients do not see a doctor. Using Google to type in our symptoms does not lead to good results, it always boils down to one stereotypical disease. Some people have stopped using Google to look for their symptoms or the probable disease. Late disease detection limits treatment options.Resource constraints in healthcare necessitate efficient allocation through predictive models. Rising healthcare costs .Predictive models address public health challenges by informing targeted campaigns and strategies.

The model is right 95 out of 100 times on an average. This model should be used for preventive diagnosis and early intervention by doctors and should not be taken as completely accurate and the final call.

Technical Architecture:





Project Flow:

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

- **Importing the libraries**

  All the required libraries are to be imported like numpy,[pandas,seaborn

- **Collect or Create Dataset:**
  - Acquire relevant data either by collecting from existing sources or creating a dataset that aligns with the objectives of the analysis or modeling.

- **Data Preprocessing:**
  a. **Import Libraries:**
    - Import necessary programming libraries (e.g., in Python, using **import** statements) for data manipulation, analysis, and modeling.

  b. **Importing the Dataset:**
    - Load the dataset into the chosen programming environment or tool (e.g., using Pandas in Python).

  c. **Checking for Null Values:**
    - Examine the dataset for missing values to ensure data quality and decide on appropriate strategies for handling them.

  d. **Data Visualization:**

- Utilize visualization techniques (e.g., matplotlib or seaborn in Python) to gain insights into the dataset's distribution, patterns, and relationships between variables.

e. **Outlier Detection:**
- Identify and handle outliers, data points significantly different from others, to prevent them from disproportionately influencing the model.

f. **Splitting Dependent and Independent Variables:**
- Distinguish between the target variable (dependent) and the features (independent variables) to prepare for model training.

g. **Encoding:**
- Convert categorical variables into a numerical format suitable for machine learning algorithms, often using techniques like one-hot encoding.

h. **Feature Scaling:**
- Normalize or scale numerical features to ensure uniformity in their impact on the model.

i. **Splitting Data into Train and Test:**
- Divide the dataset into training and testing sets to assess the model's performance on unseen data.

- **Model Building:**

a. **Import Model Building Libraries:**
- Bring in libraries specific to machine learning algorithms, such as scikit-learn for Python.

b. **Initializing the Model:**
- Create an instance of the chosen machine learning model, defining its structure and initial parameters.

c. **Performance Testing & Hyperparameter Tuning**
- Testing model with multiple evaluation metrics
- Comparing model accuracy before & after applying hyperparameter tuning
- Comparing model accuracy for different number of features.
- Building model with appropriate features.

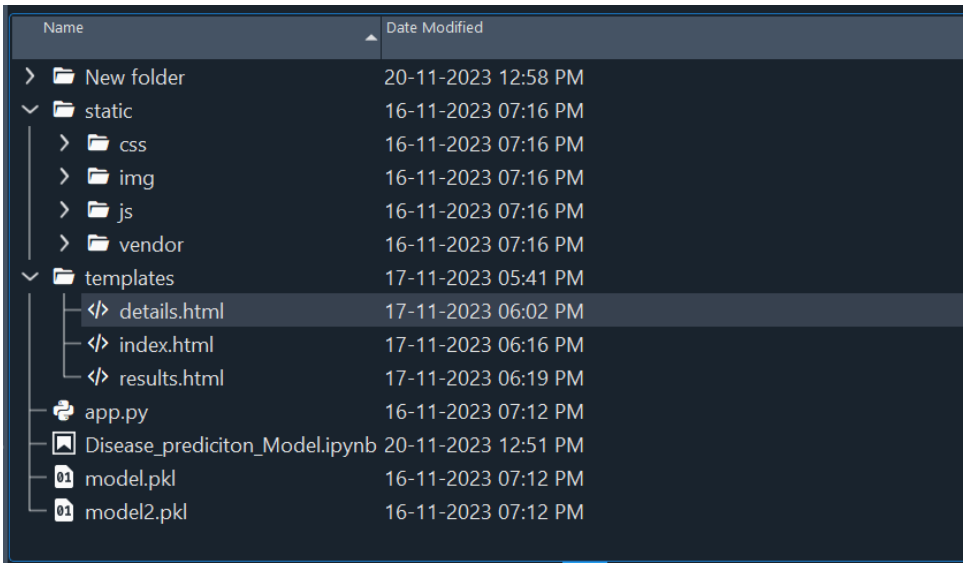
d. **Training and Testing the Model:**
- Use the training dataset to teach the model patterns and relationships, then assess its performance on the testing set to ensure generalization.

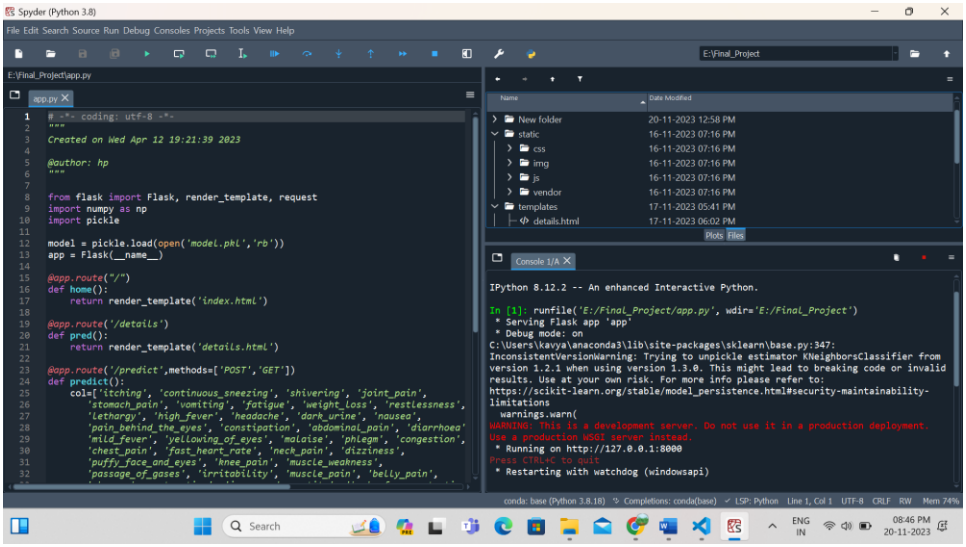
e. **Evaluation of Model:**
- Assess the model's performance using appropriate metrics, considering accuracy, precision, recall, and other relevant measures based on the problem at hand.

f. **Save the Model:**
- Preserve the trained model for future use or deployment, allowing for consistency and efficiency in application scenarios.

Project Structure:

We use spyder for flask integration process





Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

Disease prediction involves identifying individuals who are at risk of developing a particular disease, based on various risk factors such as medical history and demographic factors. Predictive analytics and machine learning techniques can be used to analyse large amounts of data to identify patterns and risk factors associated with different diseases.

Disease prediction using machine learning involves the use of various algorithms to analyse large datasets and identify patterns and risk factors associated with diseases. By analysing this data, machine learning algorithms can help identify individuals who are at risk of developing a particular disease, enabling healthcare professionals to provide personalized preventive care and early intervention.

Activity 2: Business Requirements

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

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

Activity 3: Literature Survey

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

Activity 4: Social or Business Impact.

Social Impact:

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

Business Impact:

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

Milestone 2: Data Collection and Preparation:

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.

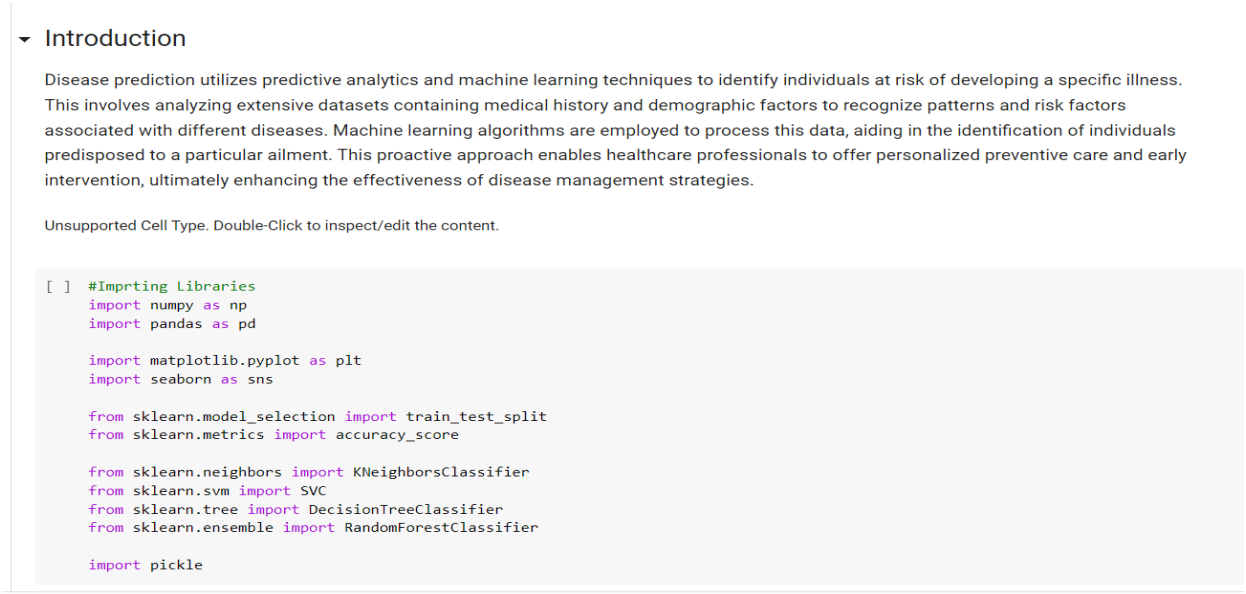
Activity 1: Collect the dataset

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

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

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

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 1.1: 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.

Activity 1.2: Read the Dataset

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

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

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

```
#Importing the dataset
train = pd.read_csv("E:/final p/Training.csv")
test = pd.read_csv("E:/final p/Testing.csv")
```

```
train.head()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	a
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

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

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

Activity 2.1: Removing Redundant Columns

```
[ ] #Removing Redundant Columns
    train['Unnamed: 133'].value_counts()
```

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

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

```
[ ] #checking for null values if any
    train.isnull().sum()
```

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

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

```
0
```

here we dont have any null values so no need to handle anything we can skip this part

```
train.columns
```

```
Index(['itching', 'skin_rash', 'nodal_skin_eruptions', 'continuous_sneezing',
       'shivering', 'chills', 'joint_pain', 'stomach_pain', 'acidity',
       'ulcers_on_tongue',
       ...
       'blackheads', 'scurring', 'skin_peeling', 'silver_like_dusting',
       'small_dents_in_nails', 'inflammatory_nails', 'blister',
       'red_sore_around_nose', 'yellow_crust_ooze', 'prognosis'],
      dtype='object', length=133)
```

Unnamed: 133 is the redundant column which does not have any values.There are no missing values in the dataset. That is why we can skip this step.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

```
#Descriptive analysis--- used to study the basic features of data with the statistical process.
train.describe()
```

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

8 rows × 132 columns

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.

```
test.shape
```

```
(42, 133)
```

```
len(train.prognosis.unique())
```

```
41
```

```
train.prognosis.value_counts()
```

```
Pneumonia                          120
Bronchial Asthma                   120
Dengue                             120
Varicose veins                     120
Fungal infection                   120
Hypothyroidism                     120
Osteoarthristis                    120
Acne                               120
Urinary tract infection            120
Chicken pox                        120
Hepatitis D                        120
hepatitis A                        120
GERD                               120
Malaria                            120
Paralysis (brain hemorrhage)       120
Common Cold                        120
Hepatitis C                        120
Migraine                           120
Typhoid                            120
Hypertension                       120
```

```
#multivariate analysis-comparing more features at a time
corr = train.corr()
corr.style.background_gradient('coolwarm')
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acid
itching	1.000000	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.160650	0.202850	-0.086
skin_rash	0.318158	1.000000	0.298143	-0.094786	-0.065324	-0.029324	0.171134	0.161784	-0.094
nodal_skin_eruptions	0.326439	0.298143	1.000000	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032
continuous_sneezing	-0.086906	-0.094786	-0.032566	1.000000	0.608981	0.446238	-0.087351	-0.047254	-0.047
shivering	-0.059893	-0.065324	-0.022444	0.608981	1.000000	0.295332	-0.060200	-0.032566	-0.032
chills	-0.175905	-0.029324	-0.065917	0.446238	0.295332	1.000000	-0.004688	-0.095646	-0.095
joint_pain	-0.160650	0.171134	-0.060200	-0.087351	-0.060200	-0.004688	1.000000	-0.087351	-0.087
stomach_pain	0.202850	0.161784	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	1.000000	0.433
acidity	-0.086906	-0.094786	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	0.433917	1.000
ulcers_on_tongue	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.649078	0.608
muscle_wasting	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032
vomiting	-0.057763	-0.225046	-0.119543	-0.173459	-0.119543	0.144263	0.199921	0.031406	0.019
burning_micturition	0.207896	0.166507	-0.032103	-0.046581	-0.032103	-0.094285	-0.086108	0.412239	-0.046
spotting_ urination	0.350585	0.298143	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.608981	-0.032
fatigue	0.069744	-0.105248	-0.120465	0.041755	-0.120465	0.269437	0.066652	-0.174797	-0.174
weight_gain	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033
anxiety	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033
cold_hands_and_feets	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033
mood_swings	-0.088129	-0.096120	-0.033025	-0.047919	-0.033025	-0.096992	-0.088581	-0.047919	-0.047
weight_loss	0.091830	-0.139363	-0.047882	-0.069477	-0.047882	0.064721	-0.128431	-0.069477	-0.069
restlessness	-0.088129	-0.096120	-0.033025	-0.047919	-0.033025	-0.096992	-0.088581	-0.047919	-0.047
lethargy	0.311436	0.067246	-0.047882	-0.069477	-0.047882	-0.140627	-0.128431	-0.069477	-0.069
patches_in_throat	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032

```
#we drop the some columns due to high correlation
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)
```

```
#new correlation matrix
corr = train.corr()
corr.style.background_gradient('coolwarm')
```

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

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

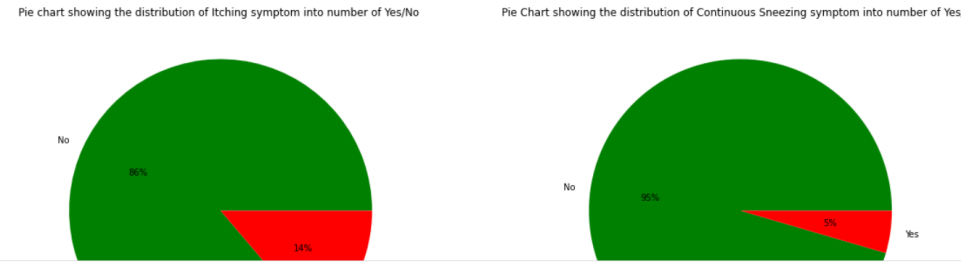
Here the plt.figure() command is used to determine the size of the plot. Then we divide the space for 2 pie plots.

```
#data Visualization
#Univariate Analysis -- understanding the data with a single feature.
#creating a side-by-side comparison of pie charts for distribution of 'itching' and 'continuous_sneezing' symptoms
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)
```



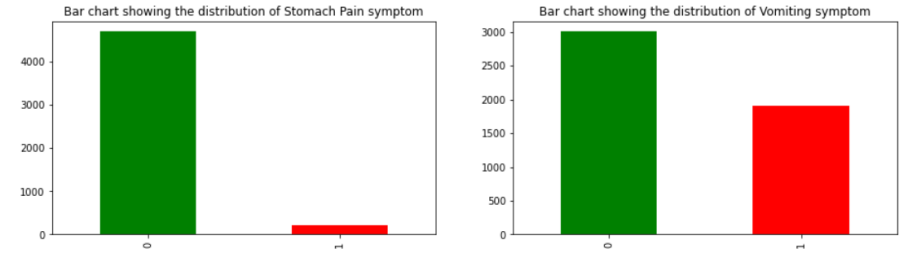
The pie plot on the left shows the different values distribution in the Itching column. It shows that there are 86% observations where the itching symptom has value 0 and there are 14% observations where the itching symptom has value 1.

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.

```
#creating a side-by-side comparison of bar graphs for below distribution
plt.subplot(1,2,1)
train['stomach_pain'].value_counts().plot(kind = 'bar', color = ['g','r'])
plt.title("Bar chart showing the distribution of Stomach Pain symptom")

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

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



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.

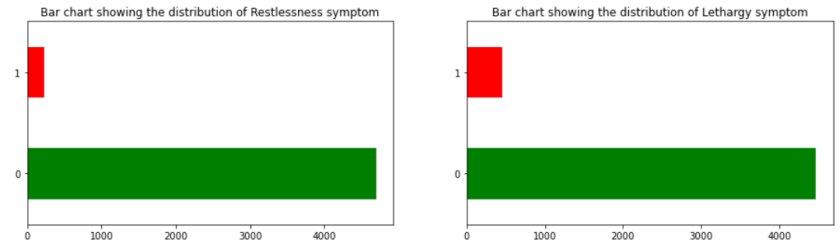
The graph on the right shows the distribution of vomiting symptom values. We can

see that the 0 value has count of around 3000 and the 1 value has count of around 2000.

```
#creating a side-by-side comparison of bar graphs horizontally for below distribution
plt.subplot(1,2,1)
train['restlessness'].value_counts().plot(kind = 'barh', color = ['g','r'])
plt.title("Bar chart showing the distribution of Restlessness symptom")

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

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



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.

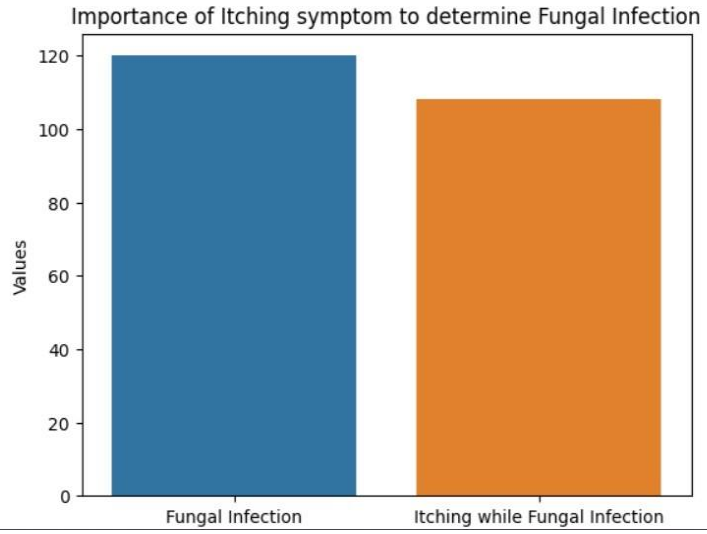
Activity 2.2: Bivariate Analysis:

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.

```
In [26]: 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[26]: Text(0.5, 1.0, 'Importance of Itching symptom to determine Fungal Infection')
```

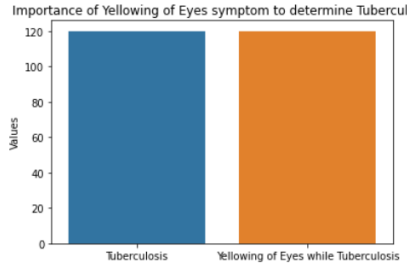


Here we use Boolean Indexing to filter out values from the prognosis column where the values are ‘Fungal Infection’. These observations are stored in variable ‘a’. Alse we filter out values from the prognosis where values are ‘Fungal Infection’ and also the values of Itching variable are 1. From the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

```
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')
```

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



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.

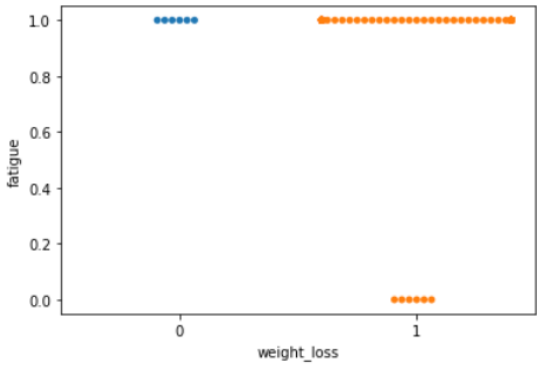
```
a = train[train['prognosis'] == 'Tuberculosis']
a.head()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_p
250	0	0	0	0	0	1	
251	0	0	0	0	0	1	
252	0	0	0	0	0	0	
253	0	0	0	0	0	1	
254	0	0	0	0	0	1	

5 rows × 90 columns

```
sns.swarmplot(x = a['weight_loss'], y = a['fatigue'])
```

```
C:\Users\kavya\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 7
  warnings.warn(msg, UserWarning)
<AxesSubplot:xlabel='weight_loss', ylabel='fatigue'>
```



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.

Activity 2.4: 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.

```
In [18]: 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.

```
In [21]: 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.

```
In [19]: X = train.drop('prognosis',axis = 1)
         y = train.prognosis
```

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

```
In [22]: 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.

```
In [20]: 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

```
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))]
```

```
### Training and testing the models using multiple algorithms steps for model building
KNN
SVM
Decision
Tree Random Forest

#Steps for model building
o Import the model building Libraries
o Initializing the model
o Training and testing the model
o Evaluation of Model
o Save the Model
```

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.

Activity 2: Training and testing the models using multiple algorithms

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

Activity 2.1: K Nearest Neighbors Model

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

```
In [24]: knn = KNeighborsClassifier(n_neighbors=7)
         knn.fit(X_train,y_train)
```

```
Out[24]: KNeighborsClassifier
         KNeighborsClassifier(n_neighbors=7)
```

```
In [25]: knn_results = model_evaluation(knn)
```

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

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

Activity 2.2: SVM Model

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

its performance.

```
In [26]: svm = SVC(C=1)
         svm.fit(X_train, y_train)
```

```
Out[26]: SVC
         SVC(C=1)
```

```
In [27]: svm_results = model_evaluation(svm)
```

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

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

Activity 2.3: Decision Tree Model

A variable is created with name dtc which has DecisionTreeClassifier() algorithm initialised in it with a parameter max_features set to 10. The dtc model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

```
In [28]: dtc = DecisionTreeClassifier(max_features= 10)
         dtc.fit(X_train,y_train)
```

```
Out[28]: DecisionTreeClassifier
         DecisionTreeClassifier(max_features=10)
```

```
In [29]: 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
```

```
[ ] #Testing model with Multiple Evaluatiton metrics
    results = pd.DataFrame(data = [knn_results, svm_results, dtc_results, rfc_results],
                           columns= ['Training Accuracy','Validation Accuracy', 'Testing Accuracy'],
                           index = ['K Nearest Neighbors Classifier','Support Vector Machines',
                                    'Decision Trees Classifier', 'Random Forest Classifier'])
```

```
results
```

	Training Accuracy	Validation Accuracy	Testing Accuracy
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

```
[ ] #We can check the feature importance using the Random Forest Classifier model.
    a = rfc.feature_importances_
```

```
[ ] col = X.columns
```

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

```
[ ] feat_imp
```

```
{'itching': 0.011774032425580673,
 'skin_rash': 0.007241202608728456,
 'nodal_skin_eruptions': 0.0027358866421039784,
 'continuous_sneezing': 0.009910955596113775,
 'shivering': 0.00971557795997768,
 'chills': 0.009165960313556449,
 'joint_pain': 0.013366839582351518,
 'stomach_pain': 0.005419558423438437,
 'acidity': 0.0055385087453628966,
 'ulcers_on_tongue': 0.007516397534192122,
 'muscle_wasting': 0.007133396221666308,
 'vomiting': 0.01076990965711016,
 'burning_micturition': 0.0037559264185494275,
 'spotting_ urination': 0.0021202740595620793,
 'fatigue': 0.010136474381609821,
 'weight_loss': 0.014953223166354796,
 'restlessness': 0.014022958923694 74,
 'lethargy': 0.01169987391710434,
```

```
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))]
```

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

```
#We will drop columns which have very less feature importance.
#we have created a for loop which will train the model and give out the accuracy.
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)
```

Activity 2.4: Random Forest Model

Random Forest Classifier is a Bagging model which utilises multiple decision

trees and takes their aggregate to give a prediction. A variable is created with name rfc which has RandomForestClassifier() algorithm initialised in it with a parameter max_depth set to 13. The rfc model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

Milestone 5 : Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with Multiple Evaluatiton metrics

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

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.

#CONFUSION MATRIX

```
[81] confusion_matrix(y1_test, y_pred1)

array([[1, 0, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 0, 0, 1]])
```

```
pd.crosstab(y1_test, y_pred1)
```

col_0	(vertigo) Paroymsal Positional Vertigo	AIDS	Arthritis	Bronchial Asthma	Cervical spondylosis	Chicken pox	Chronic cholestasis	Common Cold	Dengue	Diabetes	...	Osteoarthristis	h
prognosis													
(vertigo) Paroymsal Positional Vertigo	1	0	0	0	0	0	0	0	0	0	...	0	
AIDS	0	1	0	0	0	0	0	0	0	0	...	0	
Acne	0	1	0	0	0	0	0	0	0	0	...	0	
Alcoholic hepatitis	0	1	0	0	0	0	0	0	0	0	...	0	
Allergy	0	1	0	0	0	0	0	0	0	0	...	0	
Arthritis	0	0	1	0	0	0	0	0	0	0	...	0	
Bronchial Asthma	0	0	0	1	0	0	0	0	0	0	...	0	
Cervical													

GERD	0	0	0	0	0	0	0	0	0	0	...	0	0
Gastroenteritis	0	0	0	0	0	0	0	0	0	0	...	0	0
Heart attack	0	0	0	0	0	0	0	0	0	0	...	0	0
Hepatitis B	0	0	0	0	0	0	0	0	0	0	...	0	0
Hepatitis C	0	0	0	0	0	0	0	0	0	0	...	0	0
Hepatitis D	0	0	0	0	0	0	0	0	0	0	...	0	0
Hepatitis E	0	0	0	0	0	0	0	0	0	0	...	0	0
Hypertension	0	0	0	0	0	0	0	0	0	0	...	0	0
Hyperthyroidism	0	0	0	0	0	0	0	0	0	0	...	0	0
Hypoglycemia	0	0	0	0	0	0	0	0	0	0	...	0	0
Hypothyroidism	0	0	0	0	0	0	0	0	0	0	...	0	0
Impetigo	0	0	0	0	0	0	0	0	0	0	...	0	0
Jaundice	0	0	0	0	0	0	0	0	0	0	...	0	0
Malaria	0	0	0	0	0	0	0	0	0	0	...	0	0
Migraine	0	0	0	0	0	0	0	0	0	0	...	0	0
Osteoarthristis	0	0	0	0	0	0	0	0	0	0	...	1	0
Paralysis (brain hemorrhage)	0	0	0	0	0	0	0	0	0	0	...	0	1
Peptic ulcer diseae	0	0	0	0	0	0	0	0	0	0	...	0	0
Pneumonia	0	0	0	0	0	0	0	0	0	0	...	0	0
Psoriasis	0	0	0	0	0	0	0	0	0	0	...	0	0

✓ Connected to Python 3 Google Compute Engine backend

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.

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

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

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

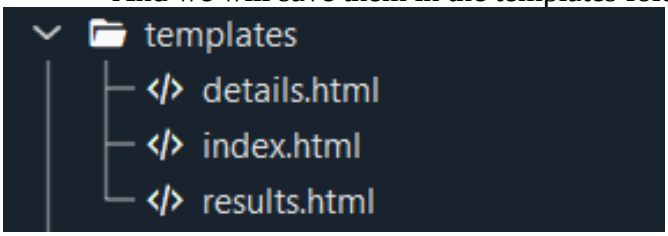
- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building HTML pages:

For this project we create three HTML files namely

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

And we will save them in the templates folder.



Activity 2.2: Build Python code

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

folder. Import the necessary Libraries.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

This list b is converted into an array and the shape of the array is changed to

(1,45). Then this array b is given to the model for prediction.

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

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

        b=[0]*45
        for x in range(0,45):
            for y in inputt:
                if(col[x]==y):
                    b[x]=1
        b=np.array(b)
        b=b.reshape(1,45)
        prediction = model.predict(b)
        prediction = prediction[0]
    return render_template('results.html', prediction_text="The probable diagnosis says it cou
```

Main Function:

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

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

Activity 2.3: Run the Web Application

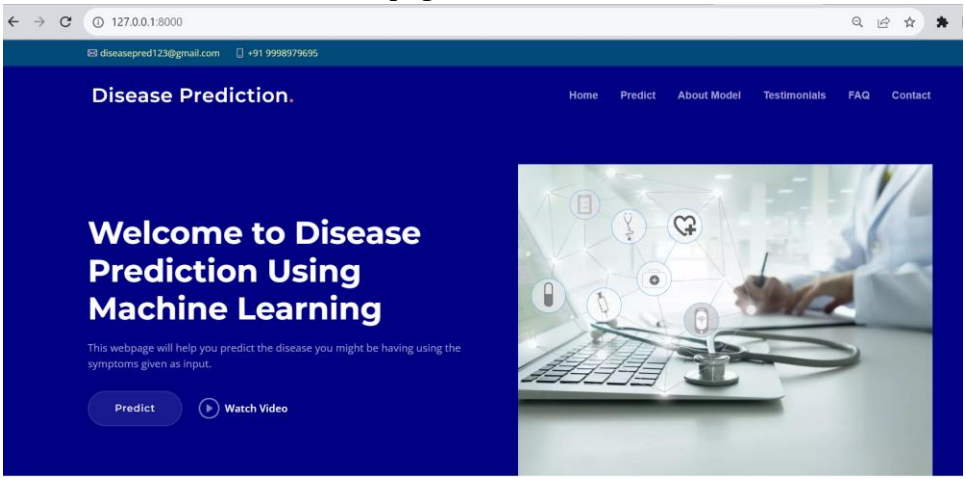
When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the

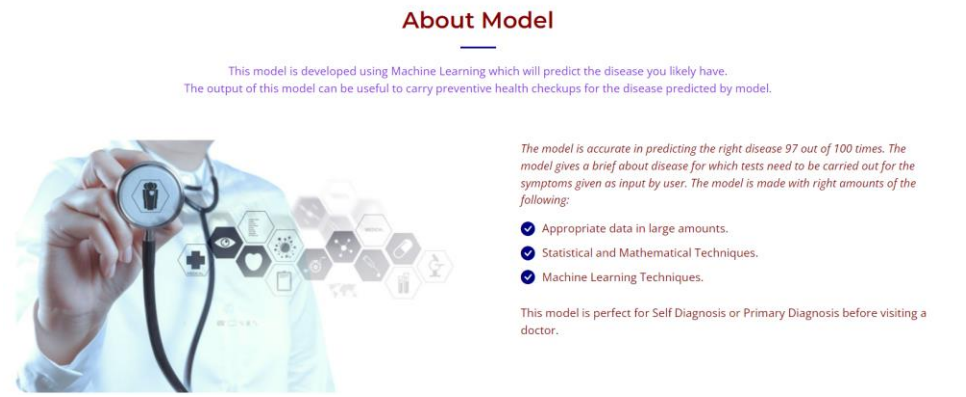
browser.

```
In [1]: runfile('C:/Users/hp/SB/Disease Prediction/Flask/app.py', wdir='C:/Users/hp/SB/Disease
Prediction/Flask')
 * 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
```

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.





Testimonials

Some testimonials about the experience of using this model.

Minal
Journalist
★★★★★

This is a great invention which is helpful for doctors as well as general patients round the world to check if there is a need to visit the doctor.

Bhavya
Designer
★★★★★

My doctor recommended me to use this model once when he was not sure of his availability. Since then I've been using this model whenever I feel a bit low.

Frequently Asked Questions

There are some questions that are very common while using the Disease Prediction Model which can come to anybody's mind. Lets us try and answer in detail regarding your questions.

1. What level of trust is shown by doctors for this Disease Prediction model?
2. Are there any timings as to when we can use this model?
3. Will this model suggest tests for full diagnosis?

   This model does not suggest any test. This model only predicts the disease which is likely for the symptoms you have given as input. You can use the information for preventive diagnosis or can consult a doctor regarding the same.
4. How many dieases can this model predict based on symptoms?

Contact

In case of any queries, please feel free to type in a message below or email us on diseasepred123@gmail.com
Also check the FAQ section for your question.

Location:
Vijayawada,Andhra Pradesh

Email:
diseasepred123@gmail.com

Call:
+91 9998979695

Open Hours:
Mon-Sat: 9AM - 5PM

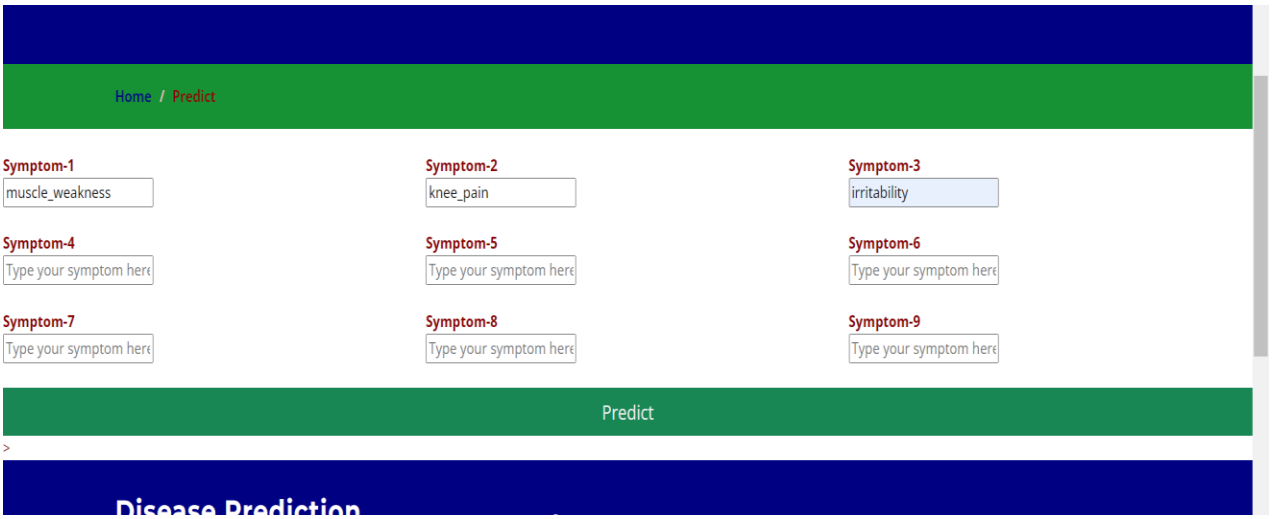
Your Name

Your Email

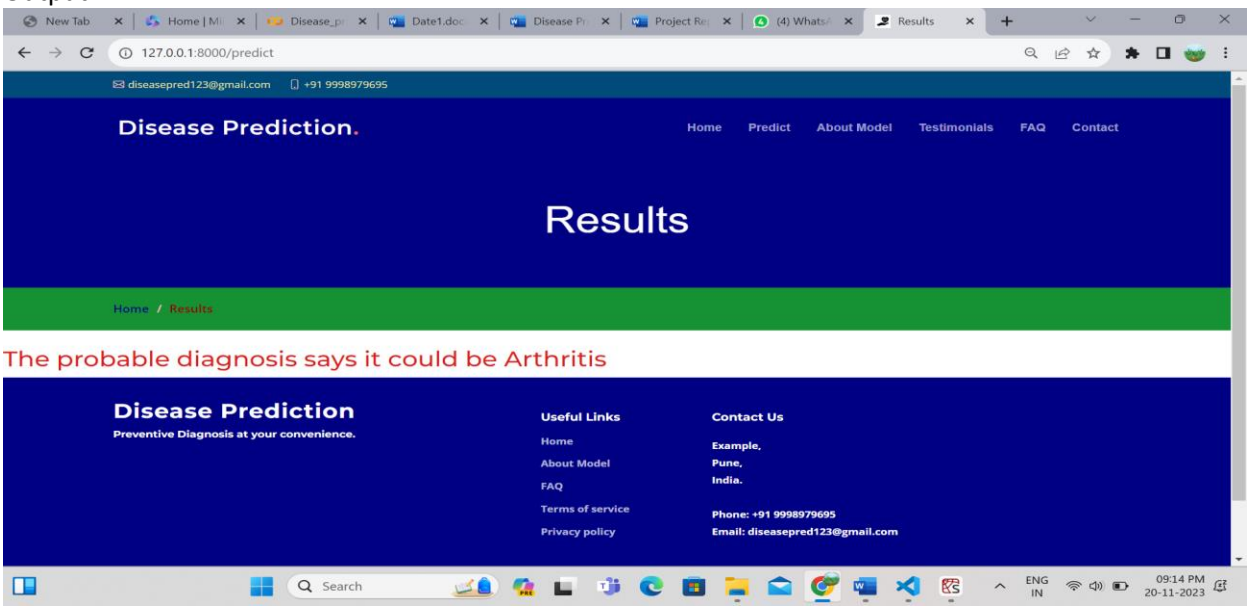
Subject

Message

Send Message


Home / Predict
Symptom-1
muscle_weakness
Symptom-2
knee_pain
Symptom-3
irritability
Symptom-4
Type your symptom here
Symptom-5
Type your symptom here
Symptom-6
Type your symptom here
Symptom-7
Type your symptom here
Symptom-8
Type your symptom here
Symptom-9
Type your symptom here
Predict
Disease Prediction

Output


127.0.0.1:8000/predict
diseasepred123@gmail.com
+91 9998979695
Disease Prediction.
Home
Predict
About Model
Testimonials
FAQ
Contact
Results
Home / Results
The probable diagnosis says it could be Arthritis
Disease Prediction
Preventive Diagnosis at your convenience.
Useful Links
Home
About Model
FAQ
Terms of service
Privacy policy
Contact Us
Example,
Pune,
India.
Phone: +91 9998979695
Email: diseasepred123@gmail.com
Search
ENG
IN
09:14 PM
20-11-2023