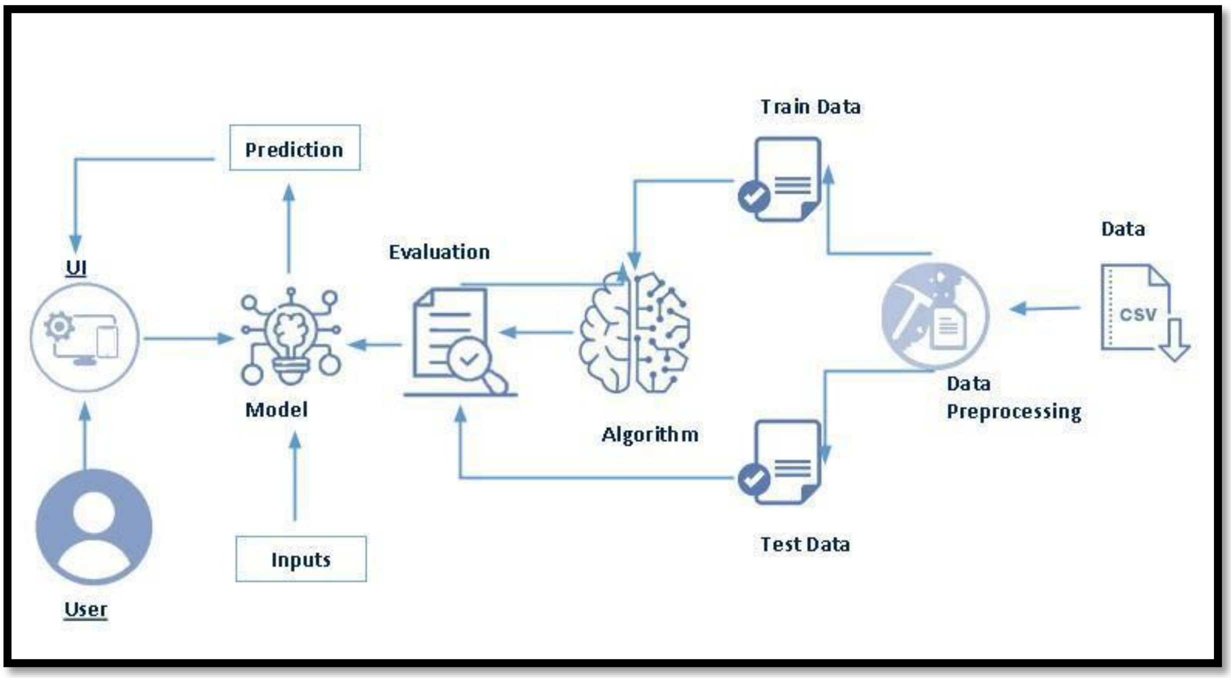
Disease Prediction Using Machine Learning

In the fast-paced world of today, finding time for healthcare has become quite challenging. Many individuals, even when experiencing severe symptoms of different illnesses, hesitate to consult a doctor. Resorting to Google for typing in symptoms often yields less-than-accurate results, typically suggesting a stereotypical disease. Consequently, people are now avoiding relying on Google to search for their symptoms or potential illnesses.

Addressing the issues mentioned earlier, we've developed a model capable of predicting up to 42 diseases when provided with symptoms as input. This model serves a dual purpose: doctors can use it for online consultations with patients based on its output, and individuals can utilize it for preventive diagnosis and self-care, especially considering the high costs associated with visiting a doctor. Importantly, the model doesn't request personal data such as name, age, gender, religion, or address. You can access this web application whenever you suspect you might be experiencing symptoms of a disease, and the model will provide a likely diagnosis based on the symptoms. It's then up to you to decide whether to seek a doctor's advice.

On average, the model accurately identifies the correct disease 97 out of 100 times. While valuable for preventive diagnosis and early intervention by doctors, it's crucial to recognize that the model isn't infallible, and the final decision should be made with caution.

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

Prior Knowledge:

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

● ML Concepts:

o Supervised learning

o K Nearest Neighbours

o SVM

o Decision Tree

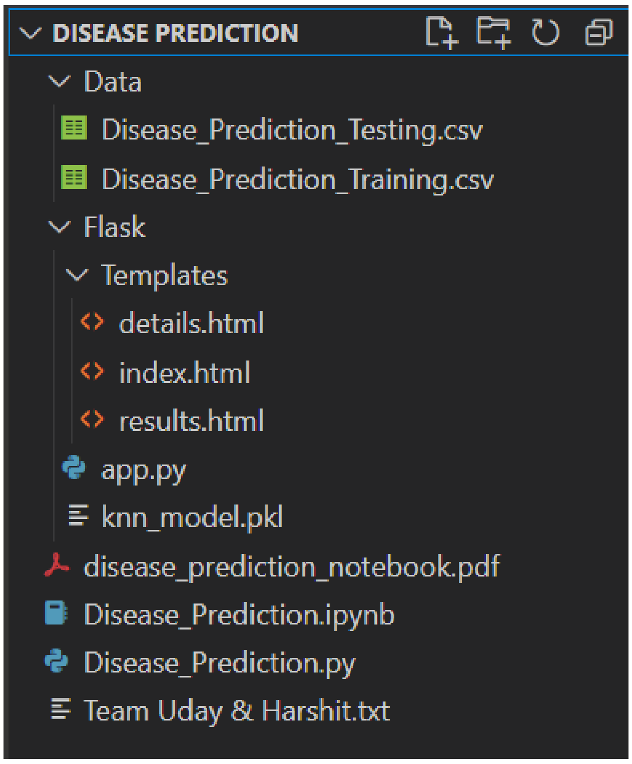
o Random Forest:

o Evaluation metrics:

● Flask Basics

Project Structure:

Create project folder which contains files as shown below:



```
DISEASE PREDICTION
  Data
    Disease_Prediction_Testing.csv
    Disease_Prediction_Training.csv
  Flask
    Templates
      details.html
      index.html
      results.html
    app.py
    knn_model.pkl
  disease_prediction_notebook.pdf
  Disease_Prediction.ipynb
  Disease_Prediction.py
  Team Uday & Harshit.txt
```

- The data obtained is in two csv files, one for training and another for testing.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- The css files should be stored in the static folder.
- App.py file is used for routing purposes using scripting.

- Model.pkl is the saved model. This will further be used in the Flask integration.
- Training folder contains a model training file.

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.

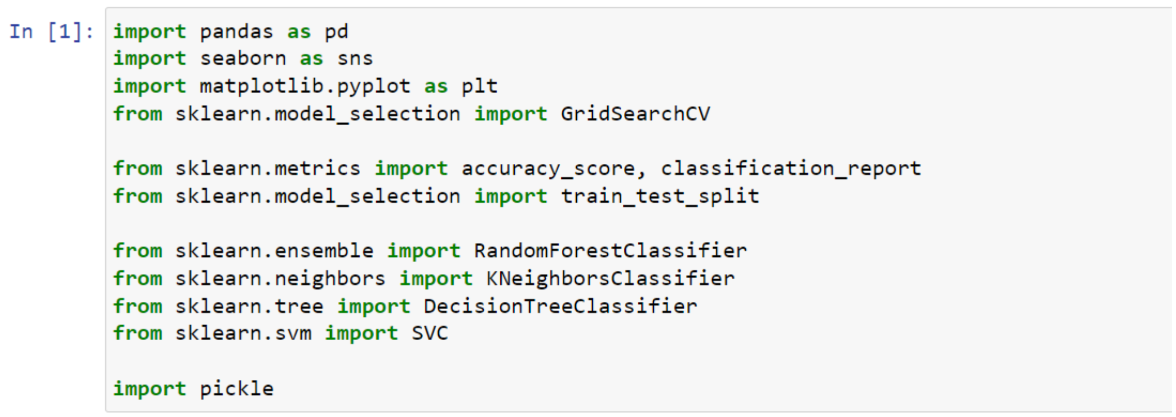
Link: https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning

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

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

Activity 1.1: 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.

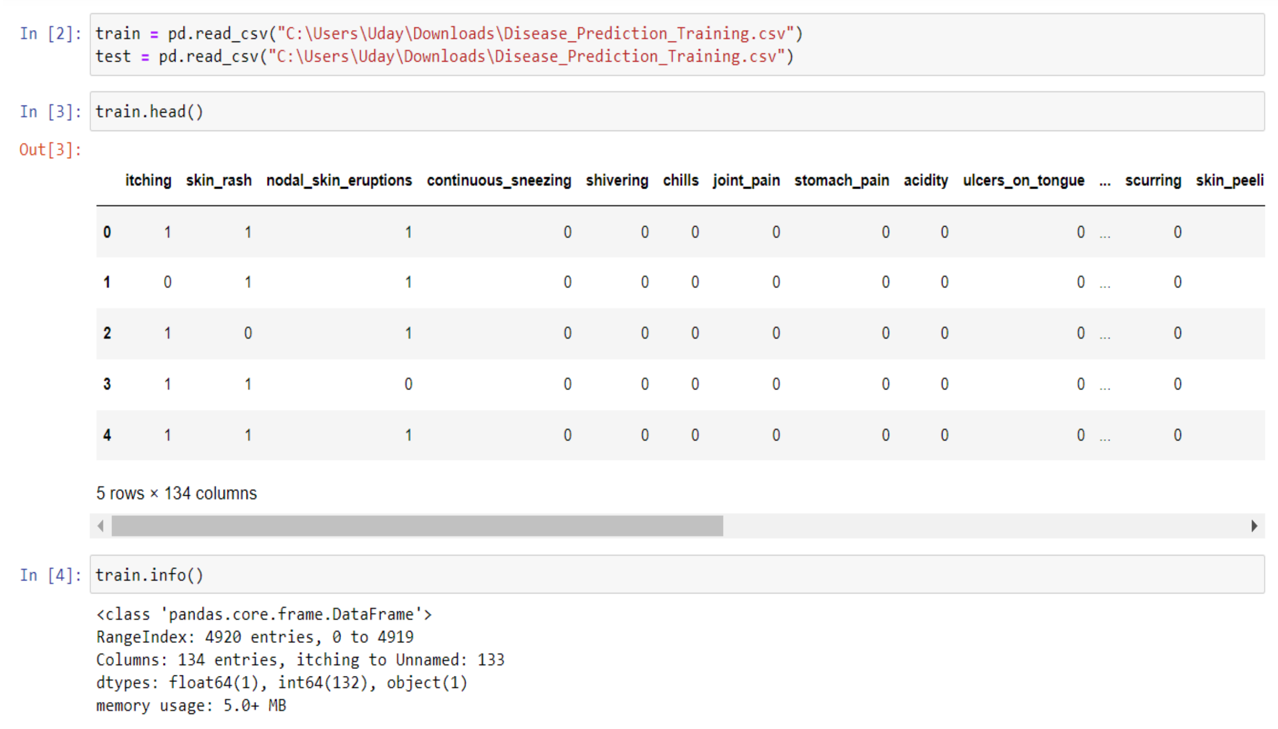


```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.model_selection import GridSearchCV

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

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

        import pickle
```

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.



```
In [2]: train = pd.read_csv("C:\Users\Uday\Downloads\Disease_Prediction_Training.csv")
        test = pd.read_csv("C:\Users\Uday\Downloads\Disease_Prediction_Training.csv")
```

```
In [3]: train.head()
```

Out[3]:

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

5 rows × 134 columns

```
In [4]: train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4920 entries, 0 to 4919
Columns: 134 entries, itching to Unnamed: 133
dtypes: float64(1), int64(132), object(1)
memory usage: 5.0+ MB
```

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

Activity 2: Data Preparation

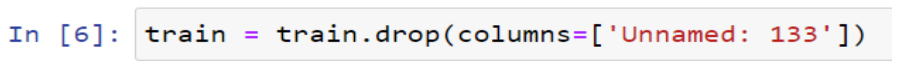
As we have understood how the data is, let us preprocess the collected data.

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

- Removing Redundant Columns
- Handling Missing Values

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.

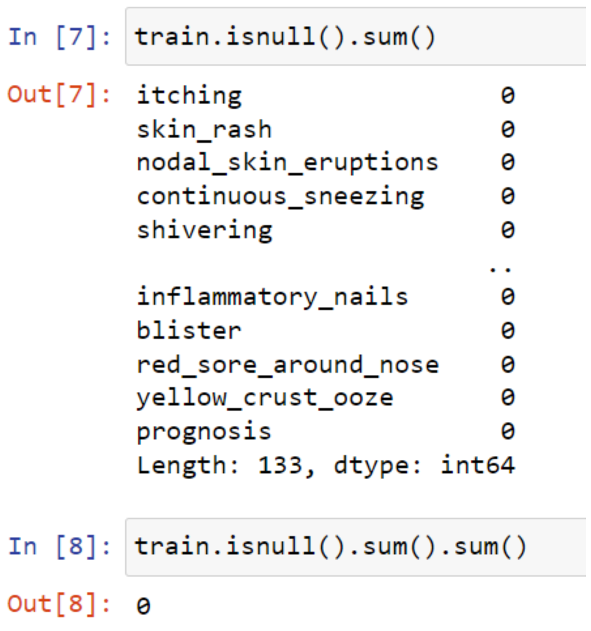
Activity 2.1: Removing Redundant Columns



```
In [6]: train = train.drop(columns=['Unnamed: 133'])
```

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

Activity 2.2: Handling Missing Values



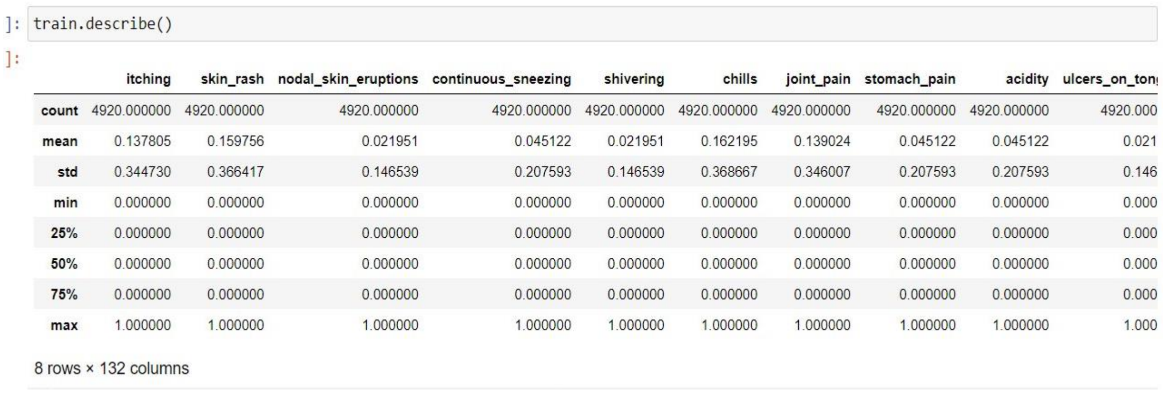
```
In [7]: train.isnull().sum()

Out[7]: 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

In [8]: train.isnull().sum().sum()

Out[8]: 0
```

There are nomissing values in the dataset. That is why we can skip this step.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.



```
]: train.describe()
]:
```

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

8 rows × 132 columns

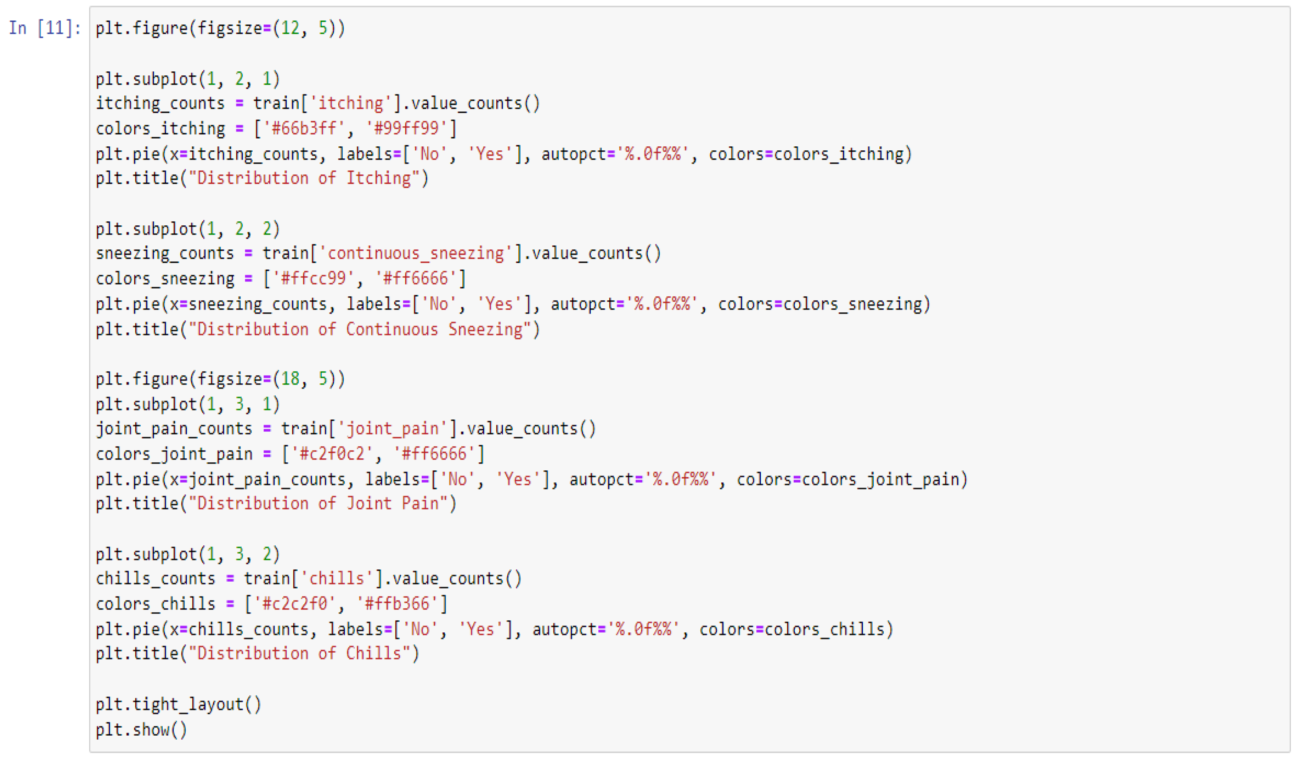
Activity 2: Visual Analysis

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

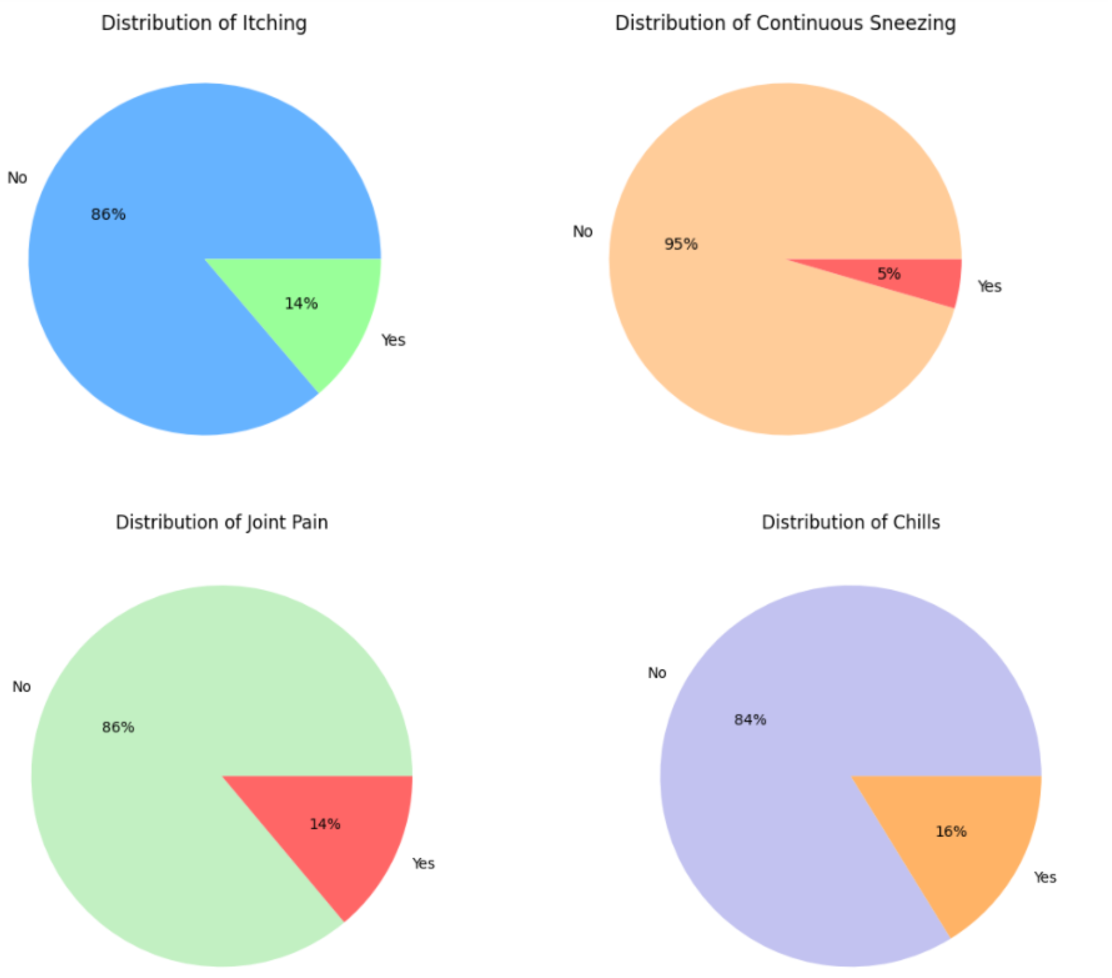
Activity 2.1: Univariate Analysis:

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.



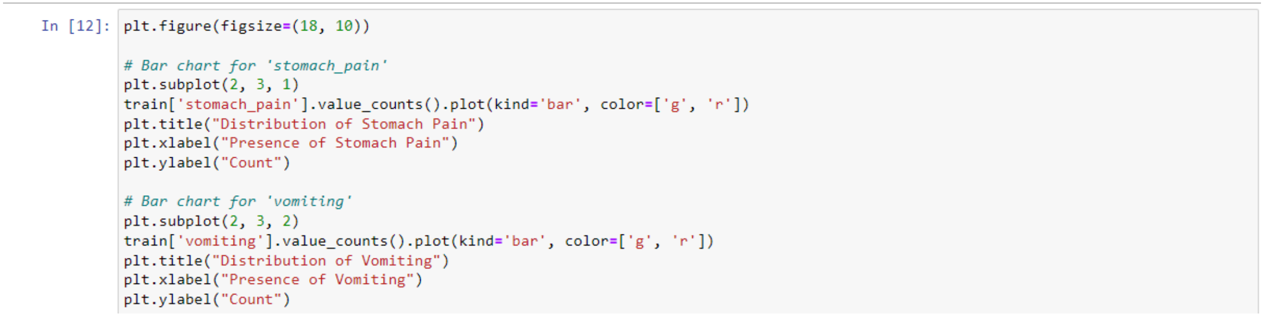
```
In [11]: plt.figure(figsize=(12, 5))

         plt.subplot(1, 2, 1)
         itching_counts = train['itching'].value_counts()
         colors_itching = ['#66b3ff', '#99ff99']
         plt.pie(x=itching_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_itching)
         plt.title("Distribution of Itching")

         plt.subplot(1, 2, 2)
         sneezing_counts = train['continuous_sneezing'].value_counts()
         colors_sneezing = ['#ffcc99', '#ff6666']
         plt.pie(x=sneezing_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_sneezing)
         plt.title("Distribution of Continuous Sneezing")

         plt.figure(figsize=(18, 5))
         plt.subplot(1, 3, 1)
         joint_pain_counts = train['joint_pain'].value_counts()
         colors_joint_pain = ['#c2f0c2', '#ff6666']
         plt.pie(x=joint_pain_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_joint_pain)
         plt.title("Distribution of Joint Pain")

         plt.subplot(1, 3, 2)
         chills_counts = train['chills'].value_counts()
         colors_chills = ['#c2c2f0', '#ffb366']
         plt.pie(x=chills_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_chills)
         plt.title("Distribution of Chills")

         plt.tight_layout()
         plt.show()
```



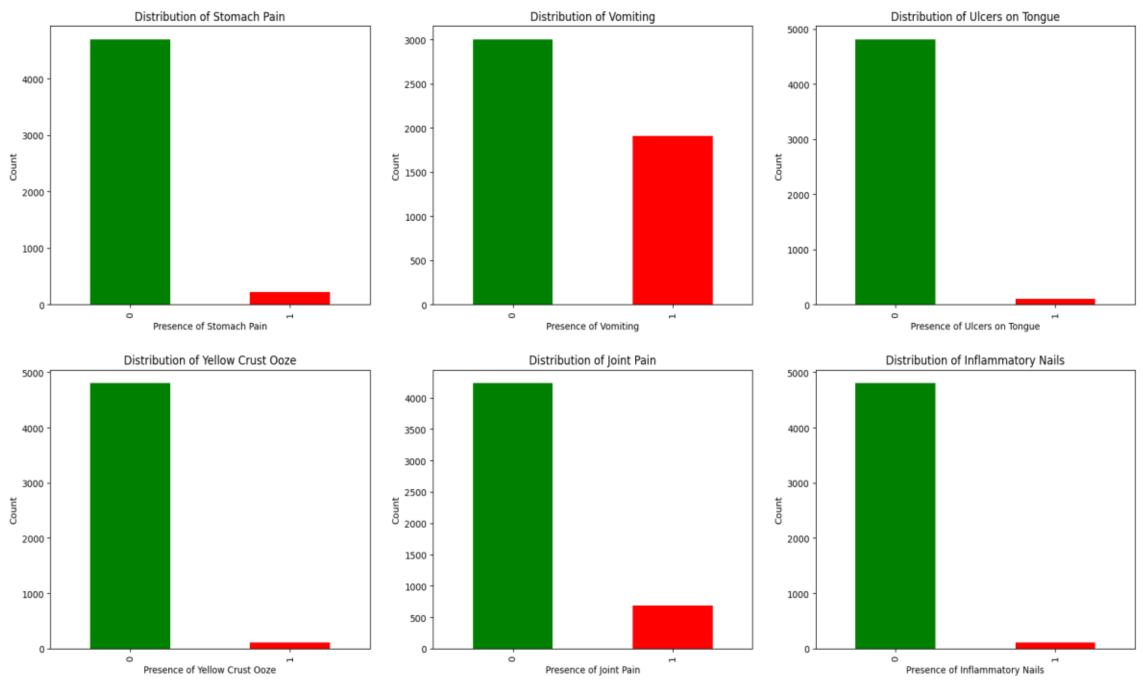
Here the plt.figure() command is used to determine the size of the plot. Then we divide the space for 4 pie plots.

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.



```
In [12]: plt.figure(figsize=(18, 10))

         # Bar chart for 'stomach_pain'
         plt.subplot(2, 3, 1)
         train['stomach_pain'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Stomach Pain")
         plt.xlabel("Presence of Stomach Pain")
         plt.ylabel("Count")

         # Bar chart for 'vomiting'
         plt.subplot(2, 3, 2)
         train['vomiting'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Vomiting")
         plt.xlabel("Presence of Vomiting")
         plt.ylabel("Count")
```



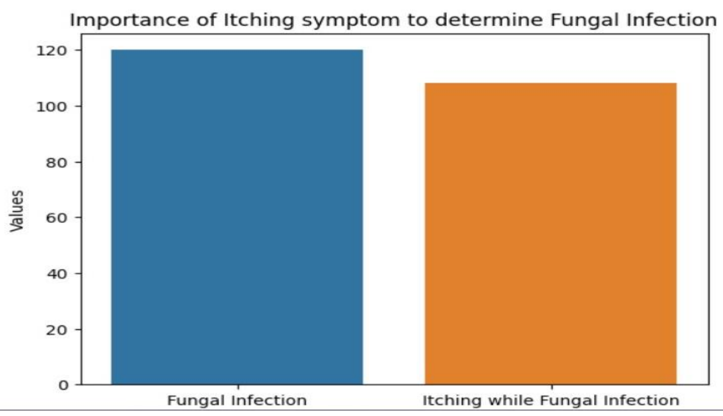
Here we have plotted 2 bar graphs. These bar graphs can be plotted without using any external library. We divide the plot into two subplots using subplot function from the matplotlib.pyplot library. We have plotted the bar graph using the inbuilt plot function in python.

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.

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 Infection'])

   sns.barplot(data = fi, x = fi.index, y = fi['Values'])
   plt.title('Importance of Itching symptom to determine Fungal Infection')
]: Text(0.5, 1.0, 'Importance of Itching symptom to determine Fungal Infection')
```

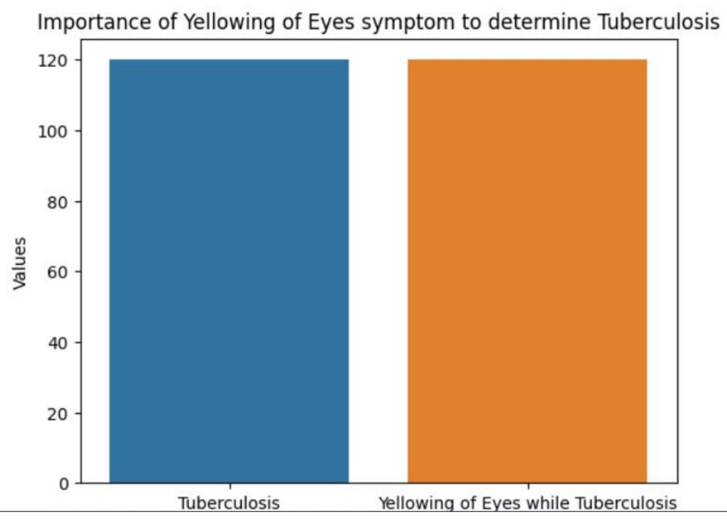


Here we use Boolean Indexing to filter out values from the prognosis column where the values are ‘Fungal Infection’. These observations are stored in variable ‘a’. Alse we filter out values from the prognosis where values are ‘Fungal Infection’ and also the values of Itching variable are 1. From the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

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.

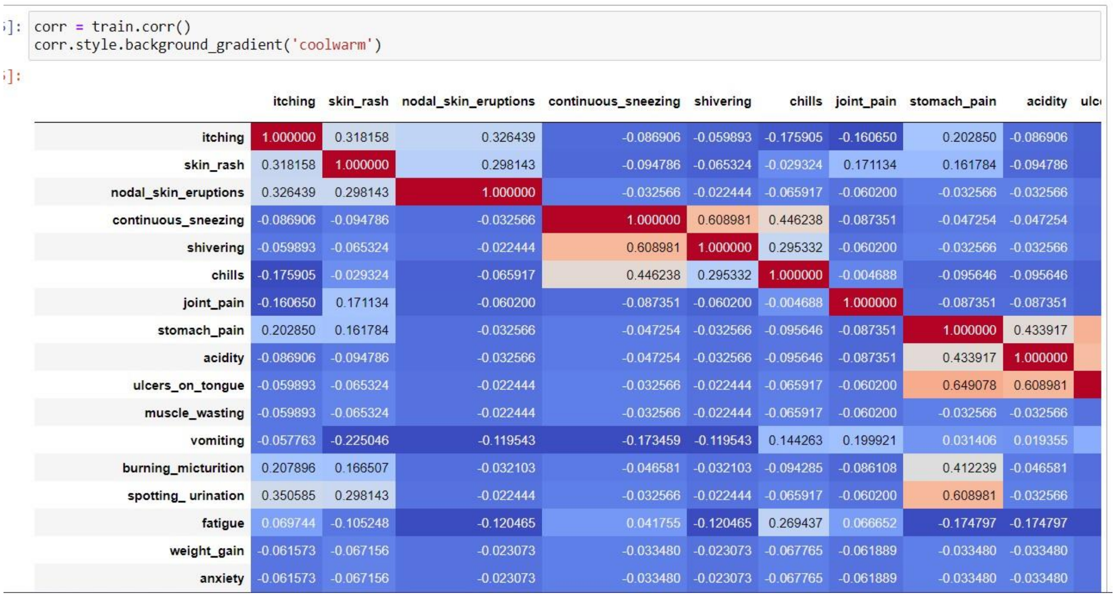
```
In [28]:
a = len(train[train['prognosis'] == 'Tuberculosis'])
b = len(train[(train['yellowing_of_eyes'] == 1) & (train['prognosis'] == 'Tuberculosis')])
fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Tuberculosis','Yellowing of Eyes while Tuberculosis'])

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

```
Out[28]: 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	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulc
itching	1.000000	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.160650	0.202850	-0.086906	
skin_rash	0.318158	1.000000	0.298143	-0.094786	-0.065324	-0.029324	0.171134	0.161784	-0.094786	
nodal_skin_eruptions	0.326439	0.298143	1.000000	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566	
continuous_sneezing	-0.086906	-0.094786	-0.032566	1.000000	0.608981	0.446238	-0.087351	-0.047254	-0.047254	
shivering	-0.059893	-0.065324	-0.022444	0.608981	1.000000	0.295332	-0.060200	-0.032566	-0.032566	
chills	-0.175905	-0.029324	-0.065917	0.446238	0.295332	1.000000	-0.004688	-0.095646	-0.095646	
joint_pain	-0.160650	0.171134	-0.060200	-0.087351	-0.060200	-0.004688	1.000000	-0.087351	-0.087351	
stomach_pain	0.202850	0.161784	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	1.000000	0.433917	
acidity	-0.086906	-0.094786	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	0.433917	1.000000	
ulcers_on_tongue	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.649078	0.608981	
muscle_wasting	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566	
vomiting	-0.057763	-0.225046	-0.119543	-0.173459	-0.119543	0.144263	0.199921	0.031406	0.019355	
burning_micturition	0.207896	0.166507	-0.032103	-0.046581	-0.032103	-0.094285	-0.086108	0.412239	-0.046581	
spotting_ urination	0.350585	0.298143	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.608981	-0.032566	
fatigue	0.069744	-0.105248	-0.120465	0.041755	-0.120465	0.269437	0.066652	-0.174797	-0.174797	
weight_gain	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033480	
anxiety	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033480	

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

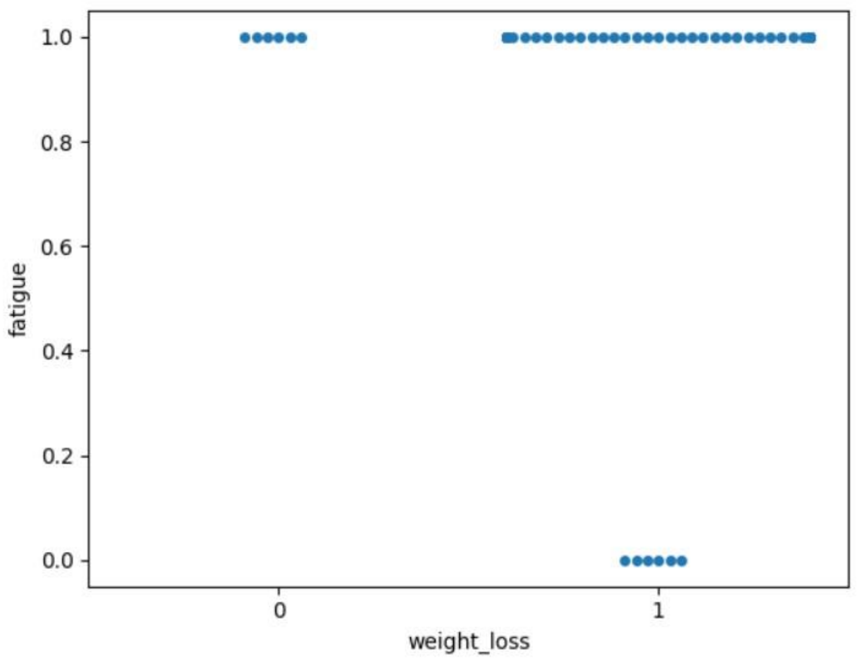
```
In [17]: #This code identifies and removes highly correlated features (columns) from a numeric DataFrame
         correlation_threshold = 0.9

         numeric_df = train.drop('prognosis', axis=1)
         corr_matrix = numeric_df.corr().abs()
         upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
         to_drop = [column for column in upper_triangle.columns if any(upper_triangle[column] > correlation_threshold)]


         df_filtered = numeric_df.drop(to_drop, axis=1)
```

```
In [18]: print("Dropped columns With high correlation:")
         print(to_drop)

         print("\n\nFiltered columns:")
         print(df_filtered.columns.tolist())
```

```
Dropped columns With high correlation:
['cold_hands_and_feets', 'redness_of_eyes', 'sinus_pressure', 'runny_nose', 'congestion', 'pain_in_anal_region', 'bloody_stoo
l', 'irritation_in_anus', 'bruising', 'swollen_legs', 'swollen_blood_vessels', 'puffy_face_and_eyes', 'enlarged_thyroid', 'brit
tle_nails', 'swollen_extremeties', 'drying_and_tingling_lips', 'slurred_speech', 'hip_joint_pain', 'unsteadiness', 'loss_of_sme
ll', 'continuous_feel_of_urine', 'internal_itching', 'altered_sensorium', 'belly_pain', 'abnormal_menstruation', 'increased_app
etite', 'polyuria', 'receiving_blood_transfusion', 'receiving_unsterile_injections', 'coma', 'stomach_bleeding', 'distention_of
_abdomen', 'history_of_alcohol_consumption', 'fluid_overload.1', 'prominent_veins_on_calf', 'palpitations', 'painful_walking',
'silver_like_dusting', 'small_dents_in_nails', 'inflammatory_nails', 'red_sore_around_nose', 'yellow_crust_ooze']
```

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

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



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

Preprocessing of Test data

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

```
In [20]: def preprocess_test_data(test, to_drop):
             test.drop(columns=to_drop, inplace=True)
             return test
```

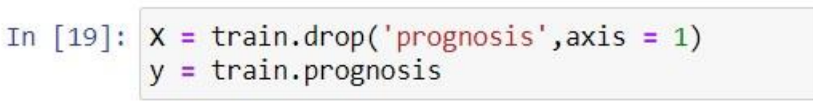
```
In [21]: test_data = preprocess_test_data(test, to_drop)
```

This function drops all the columns which needs to be dropped.
Here we call the function for the test data.

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

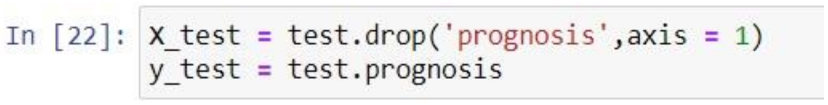
We have training and testing data given separately. We further split the training data into training and validation data. This validation data can be used for hyper parameter tuning.

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
```

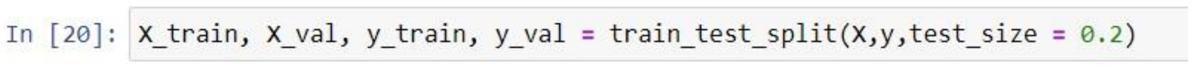
We split the training data into features(X) and target variable(y).



```
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.



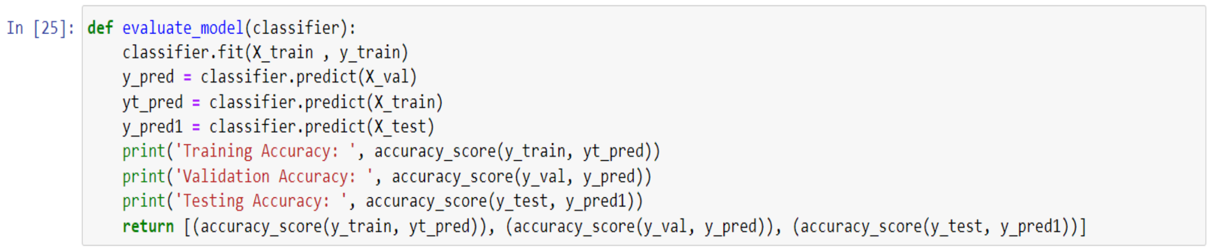
```
In [20]: X_train, X_val, y_train, y_val = train_test_split(X,y,test_size = 0.2)
```

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

Milestone 4: Model Building

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.



```
In [25]: def evaluate_model(classifier):
             classifier.fit(X_train , y_train)
             y_pred = classifier.predict(X_val)
             yt_pred = classifier.predict(X_train)
             y_pred1 = classifier.predict(X_test)
             print('Training Accuracy: ', accuracy_score(y_train, yt_pred))
             print('Validation Accuracy: ', accuracy_score(y_val, y_pred))
             print('Testing Accuracy: ', accuracy_score(y_test, y_pred1))
             return [(accuracy_score(y_train, yt_pred)), (accuracy_score(y_val, y_pred)), (accuracy_score(y_test, y_pred1))]
```

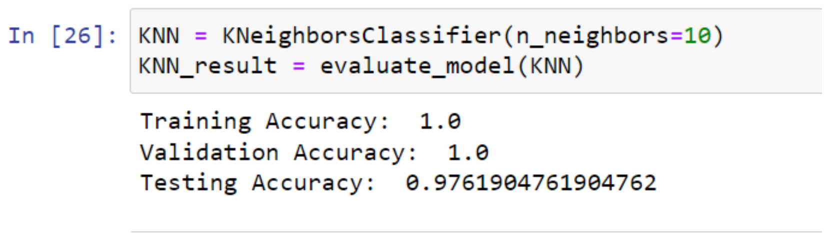
This function will show the accuracies of prediction of model for training, validation and testing data. It will also the return those accuracies.

Activity 2: Training and testing the models using multiple algorithms

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

Activity 2.1: K Nearest Neighbors Model

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.

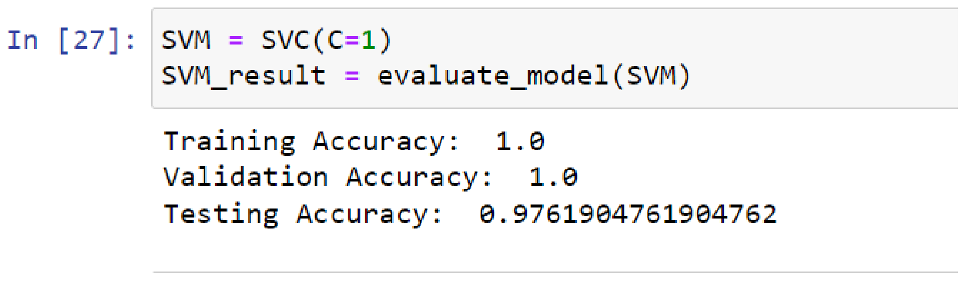


```
In [26]: KNN = KNeighborsClassifier(n_neighbors=10)
         KNN_result = evaluate_model(KNN)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training, 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.

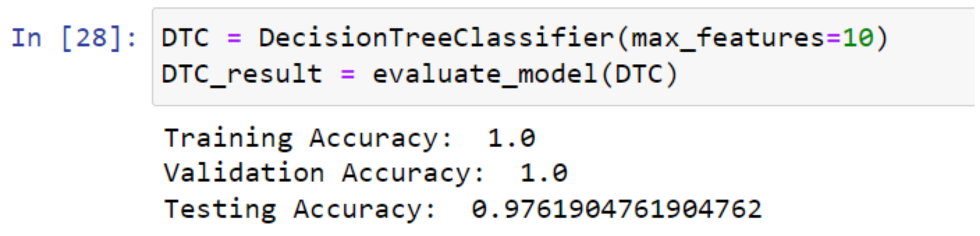


```
In [27]: SVM = SVC(C=1)
         SVM_result = evaluate_model(SVM)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training, 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.

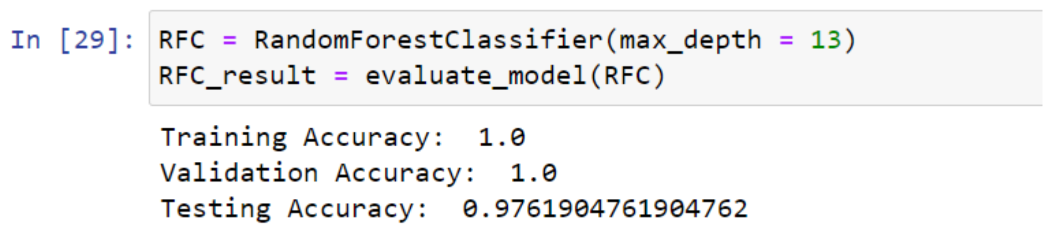


```
In [28]: DTC = DecisionTreeClassifier(max_features=10)
         DTC_result = evaluate_model(DTC)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

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

Activity 2.4: Random Forest Model

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



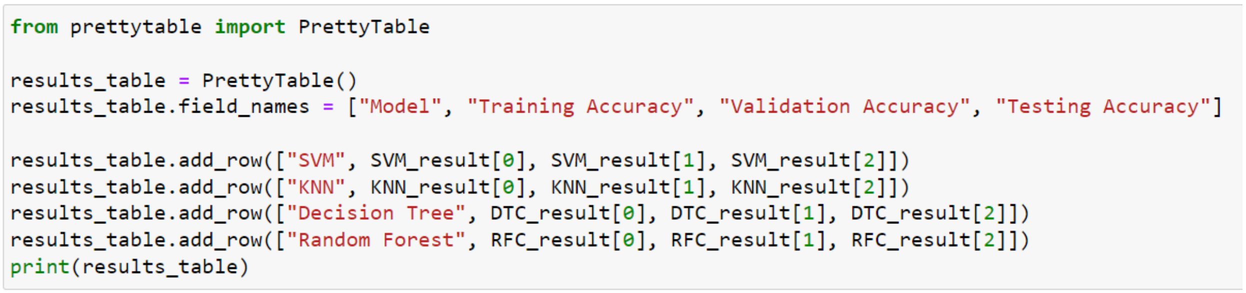
```
In [29]: RFC = RandomForestClassifier(max_depth = 13)
         RFC_result = evaluate_model(RFC)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

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

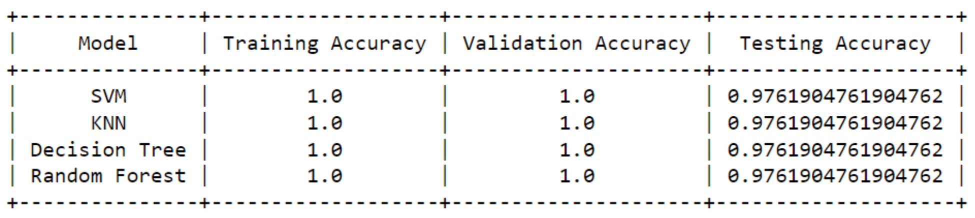
Milestone 5 : Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with Multiple 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.



```
from prettytable import PrettyTable

results_table = PrettyTable()
results_table.field_names = ["Model", "Training Accuracy", "Validation Accuracy", "Testing Accuracy"]

results_table.add_row(["SVM", SVM_result[0], SVM_result[1], SVM_result[2]])
results_table.add_row(["KNN", KNN_result[0], KNN_result[1], KNN_result[2]])
results_table.add_row(["Decision Tree", DTC_result[0], DTC_result[1], DTC_result[2]])
results_table.add_row(["Random Forest", RFC_result[0], RFC_result[1], RFC_result[2]])
print(results_table)
```



```
+---------------+-------------------+---------------------+--------------------+
|     Model     | Training Accuracy | Validation Accuracy |  Testing Accuracy  |
+---------------+-------------------+---------------------+--------------------+
|      SVM      |        1.0        |         1.0         | 0.9761904761904762 |
|      KNN      |        1.0        |         1.0         | 0.9761904761904762 |
| Decision Tree |        1.0        |         1.0         | 0.9761904761904762 |
| Random Forest |        1.0        |         1.0         | 0.9761904761904762 |
+---------------+-------------------+---------------------+--------------------+
```

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

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

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

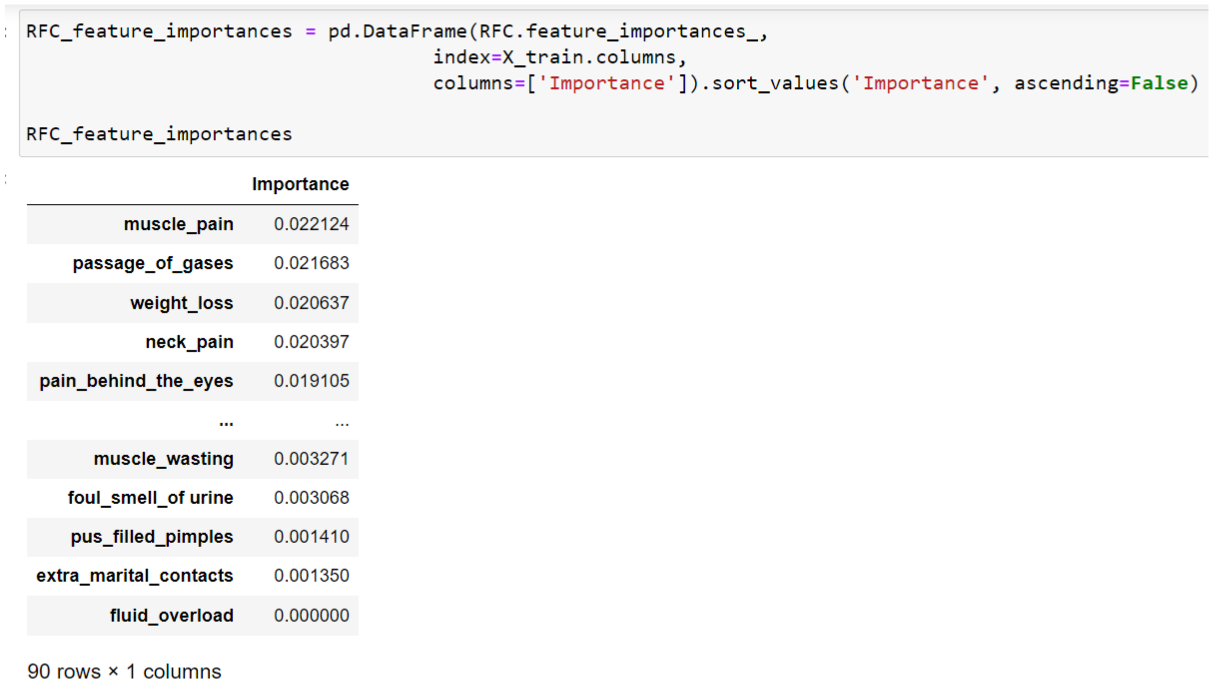
Activity 3: Comparing Model accuracy for different number of features.

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

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

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

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



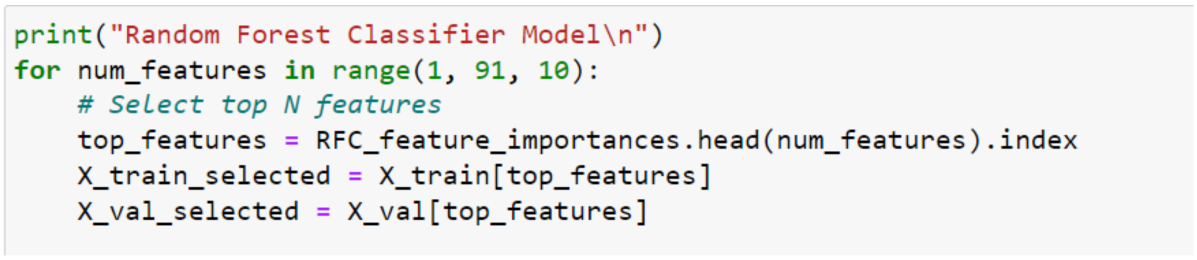
```
RFC_feature_importances = pd.DataFrame(RFC.feature_importances_,
                                       index=X_train.columns,
                                       columns=['Importance']).sort_values('Importance', ascending=False)

RFC_feature_importances
```

	Importance
muscle_pain	0.022124
passage_of_gases	0.021683
weight_loss	0.020637
neck_pain	0.020397
pain_behind_the_eyes	0.019105
...	...
muscle_wasting	0.003271
foul_smell_of urine	0.003068
pus_filled_pimples	0.001410
extra_marital_contacts	0.001350
fluid_overload	0.000000

90 rows × 1 columns

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.



```
print("Random Forest Classifier Model\n")
for num_features in range(1, 91, 10):
    # Select top N features
    top_features = RFC_feature_importances.head(num_features).index
    X_train_selected = X_train[top_features]
    X_val_selected = X_val[top_features]
```

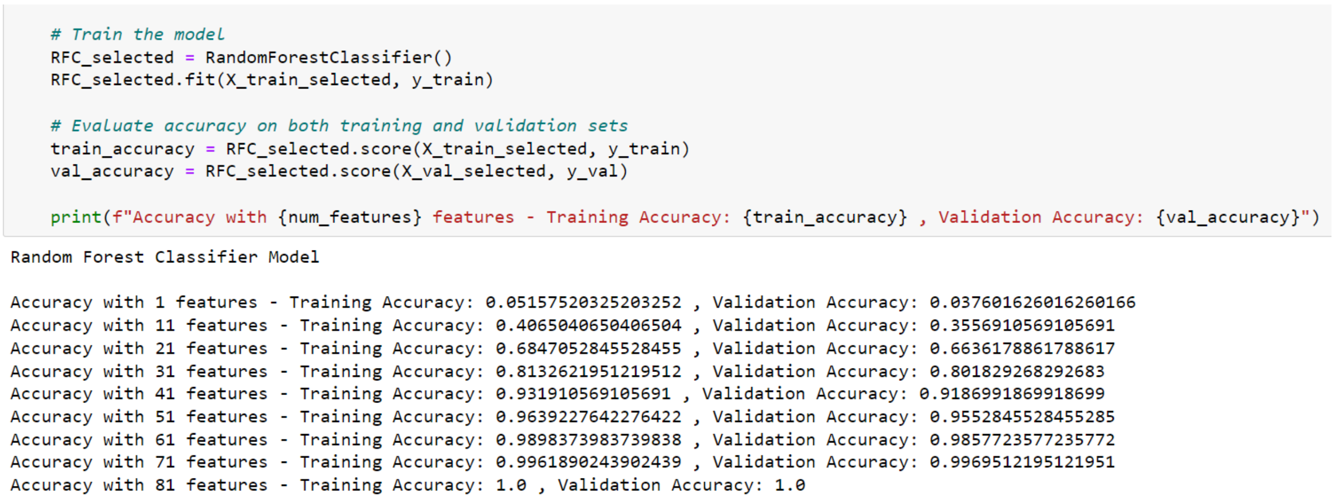
Here we create 2 lists for 2 models, knn and random forest classifier.

The for loop will iterate over values given in the list one be one. The first value will be 1 and the last will be 81
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.

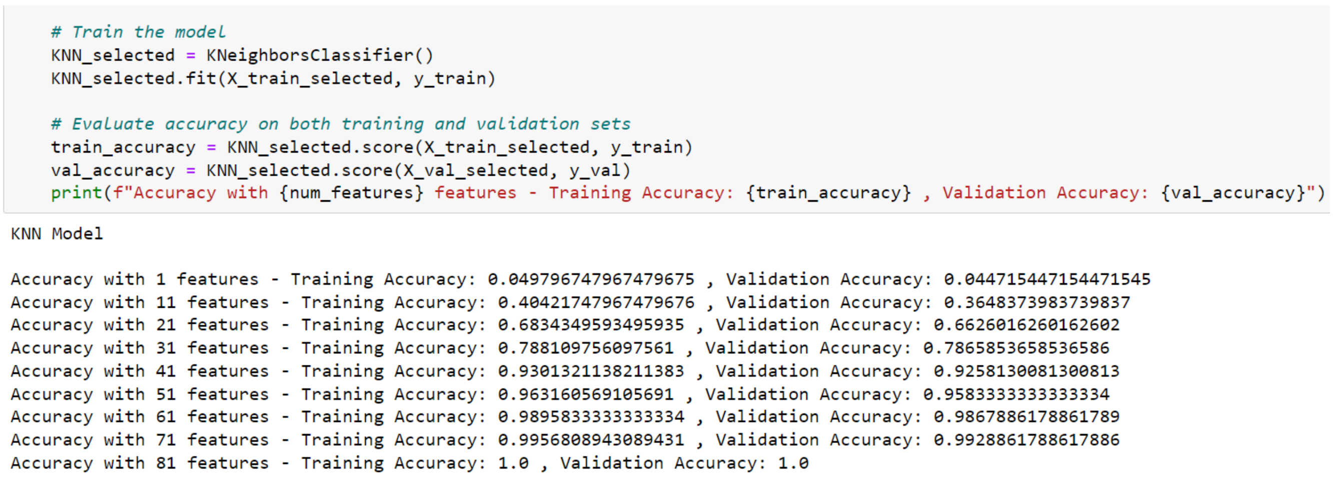
The new data will be split into features and target variable. Further they will be split into training, validation and testing data.



```
# Train the model
RFC_selected = RandomForestClassifier()
RFC_selected.fit(X_train_selected, y_train)

# Evaluate accuracy on both training and validation sets
train_accuracy = RFC_selected.score(X_train_selected, y_train)
val_accuracy = RFC_selected.score(X_val_selected, y_val)

print(f"Accuracy with {num_features} features - Training Accuracy: {train_accuracy} , Validation Accuracy: {val_accuracy}")
```

```
Random Forest Classifier Model

Accuracy with 1 features - Training Accuracy: 0.05157520325203252 , Validation Accuracy: 0.037601626016260166
Accuracy with 11 features - Training Accuracy: 0.4065040650406504 , Validation Accuracy: 0.3556910569105691
Accuracy with 21 features - Training Accuracy: 0.6847052845528455 , Validation Accuracy: 0.6636178861788617
Accuracy with 31 features - Training Accuracy: 0.8132621951219512 , Validation Accuracy: 0.801829268292683
Accuracy with 41 features - Training Accuracy: 0.931910569105691 , Validation Accuracy: 0.9186991869918699
Accuracy with 51 features - Training Accuracy: 0.9639227642276422 , Validation Accuracy: 0.9552845528455285
Accuracy with 61 features - Training Accuracy: 0.9898373983739838 , Validation Accuracy: 0.9857723577235772
Accuracy with 71 features - Training Accuracy: 0.9961890243902439 , Validation Accuracy: 0.9969512195121951
Accuracy with 81 features - Training Accuracy: 1.0 , Validation Accuracy: 1.0
```

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



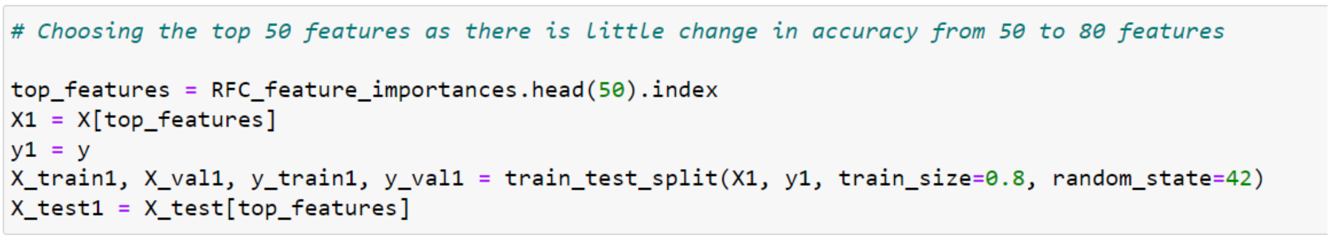
```
# Train the model
KNN_selected = KNeighborsClassifier()
KNN_selected.fit(X_train_selected, y_train)

# Evaluate accuracy on both training and validation sets
train_accuracy = KNN_selected.score(X_train_selected, y_train)
val_accuracy = KNN_selected.score(X_val_selected, y_val)
print(f"Accuracy with {num_features} features - Training Accuracy: {train_accuracy} , Validation Accuracy: {val_accuracy}")
```

```
KNN Model

Accuracy with 1 features - Training Accuracy: 0.049796747967479675 , Validation Accuracy: 0.044715447154471545
Accuracy with 11 features - Training Accuracy: 0.40421747967479676 , Validation Accuracy: 0.3648373983739837
Accuracy with 21 features - Training Accuracy: 0.6834349593495935 , Validation Accuracy: 0.6626016260162602
Accuracy with 31 features - Training Accuracy: 0.788109756097561 , Validation Accuracy: 0.7865853658536586
Accuracy with 41 features - Training Accuracy: 0.9301321138211383 , Validation Accuracy: 0.9258130081300813
Accuracy with 51 features - Training Accuracy: 0.963160569105691 , Validation Accuracy: 0.9583333333333334
Accuracy with 61 features - Training Accuracy: 0.9895833333333334 , Validation Accuracy: 0.9867886178861789
Accuracy with 71 features - Training Accuracy: 0.9956808943089431 , Validation Accuracy: 0.9928861788617886
Accuracy with 81 features - Training Accuracy: 1.0 , Validation Accuracy: 1.0
```

Above KNN model for various number of features. We can see that as the number of features go on increasing, the accuracies increase.

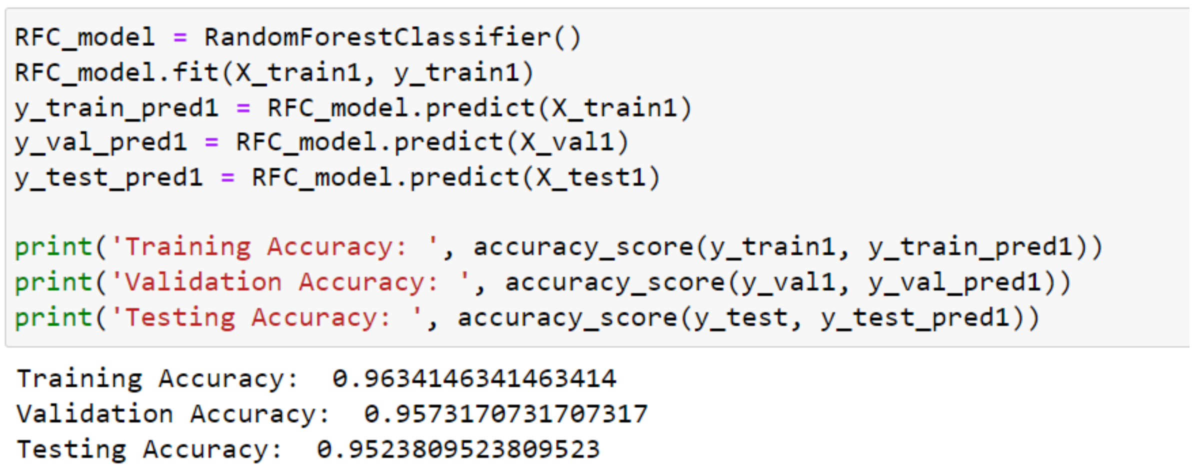
Activity 4: Building Model with appropriate features

From the above result tables, we can see that the accuracy does not change much from 51 features to 81 features. Hence we will choose 50 features for our training.



```
# Choosing the top 50 features as there is little change in accuracy from 50 to 80 features

top_features = RFC_feature_importances.head(50).index
X1 = X[top_features]
y1 = y
X_train1, X_val1, y_train1, y_val1 = train_test_split(X1, y1, train_size=0.8, random_state=42)
X_test1 = X_test[top_features]
```

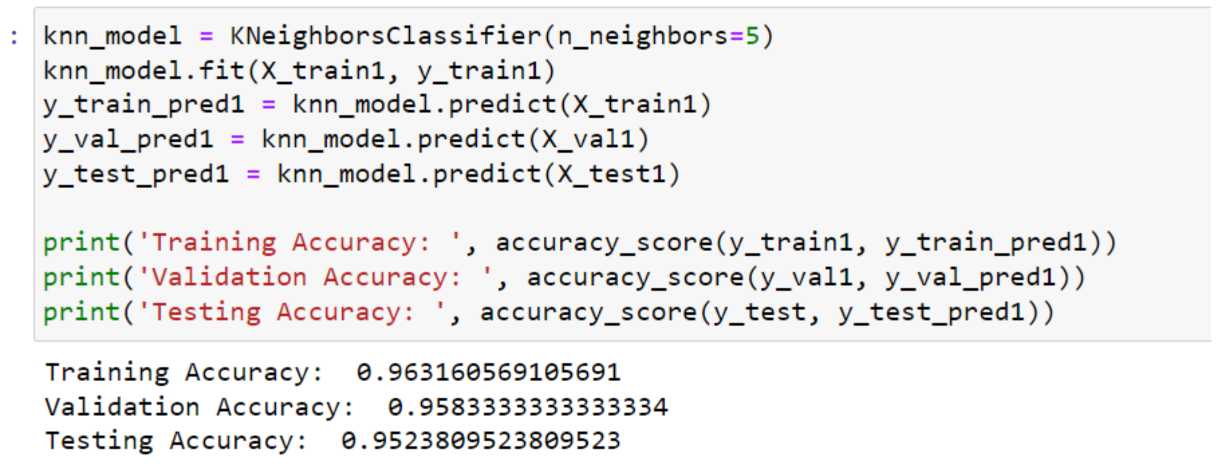
We dropped some features. Create new datasets for features and their labels. As we can see in the figure above we are left with 50 features now. We will build a Random Fores Classifier on the new data and check for the accuracies. As we can see in the figure below our model has achieved test accuracy of 95.2 % which is quite good for the number of features. Previously for 90 features we had similar accuracy for Random Forest Classifier. This states that there were many features which were not contributing much to our model.



```
RFC_model = RandomForestClassifier()
RFC_model.fit(X_train1, y_train1)
y_train_pred1 = RFC_model.predict(X_train1)
y_val_pred1 = RFC_model.predict(X_val1)
y_test_pred1 = RFC_model.predict(X_test1)

print('Training Accuracy: ', accuracy_score(y_train1, y_train_pred1))
print('Validation Accuracy: ', accuracy_score(y_val1, y_val_pred1))
print('Testing Accuracy: ', accuracy_score(y_test, y_test_pred1))
```

```
Training Accuracy:  0.9634146341463414
Validation Accuracy:  0.9573170731707317
Testing Accuracy:  0.9523809523809523
```

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



```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train1, y_train1)
y_train_pred1 = knn_model.predict(X_train1)
y_val_pred1 = knn_model.predict(X_val1)
y_test_pred1 = knn_model.predict(X_test1)

print('Training Accuracy: ', accuracy_score(y_train1, y_train_pred1))
print('Validation Accuracy: ', accuracy_score(y_val1, y_val_pred1))
print('Testing Accuracy: ', accuracy_score(y_test, y_test_pred1))
```

```
Training Accuracy:  0.963160569105691
Validation Accuracy:  0.9583333333333334
Testing Accuracy:  0.9523809523809523
```

After training the KNN model we check the accuracies. Our model has achieved 95.2 % accuracy for the test data.
To confirm let us check the compare our predicted results with the actual values.

```
In [48]: test_predictions = pd.DataFrame(y_test_pred1, columns=["predicted"])
         result_df = test.join(test_predictions)[["prognosis", "predicted"]]
         result_df.head(10)
```

Out[48]:

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

As we can see above that the values our model has predicted are same as the actual values. This shos 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_model , open('knn_model.pkl' ,'wb'))
```

```
pwd
```

```
'C:\\Users\\Uday'
```

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

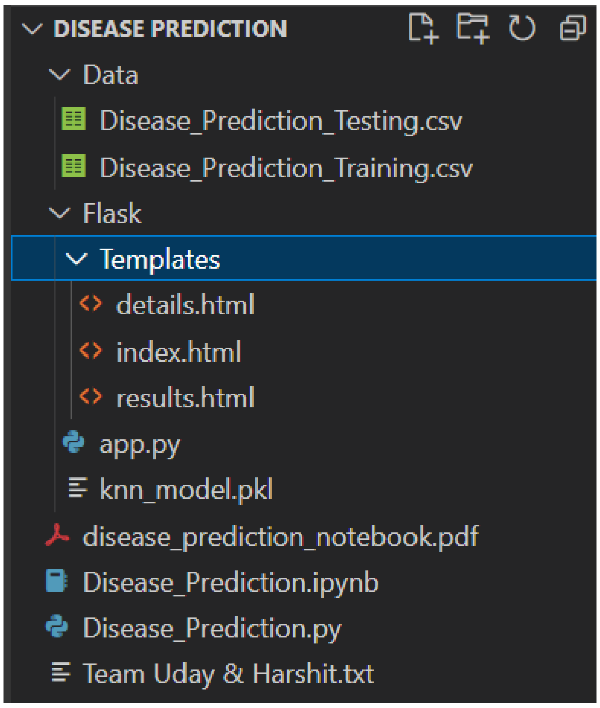
- Building HTML Pages
- Building server-side script
- Run the web application

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.

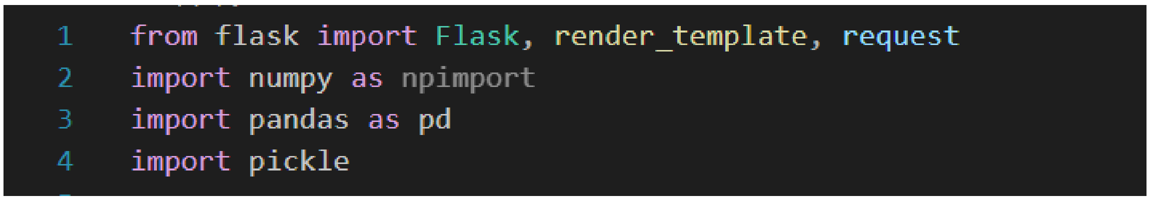


```
DISEASE PREDICTION
  Data
    Disease_Prediction_Testing.csv
    Disease_Prediction_Training.csv
  Flask
    Templates
      details.html
      index.html
      results.html
    app.py
    knn_model.pkl
  disease_prediction_notebook.pdf
  Disease_Prediction.ipynb
  Disease_Prediction.py
  Team Uday & Harshit.txt
```

Activity 2.2: Build Python code

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

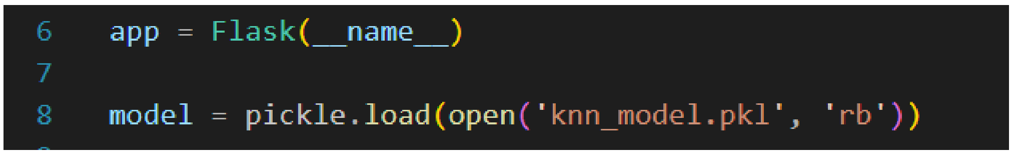
Import the necessary Libraries.



```python
from flask import Flask, render_template, request
import numpy as npimport
import pandas as pd
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.

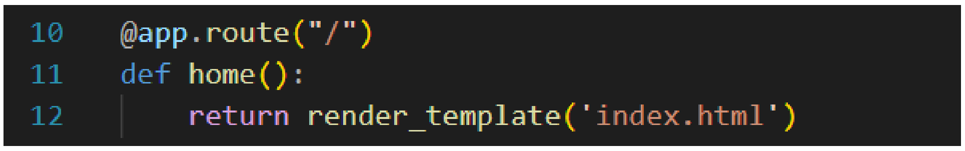


```python
app = Flask(__name__)

model = pickle.load(open('knn_model.pkl', 'rb'))
```

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.

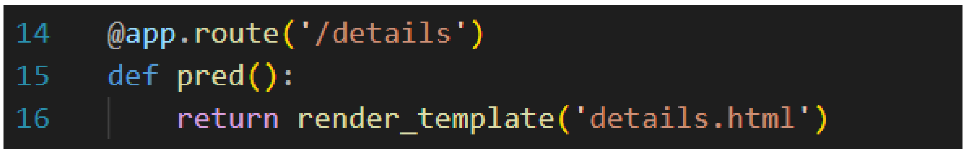
The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

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 Flask web application establishes a new route, "/predict", which accepts both GET and POST methods. The associated function, "predict()", handles the prediction process.

The variable 'col' is a list containing the names of 50 symptoms used in the model.

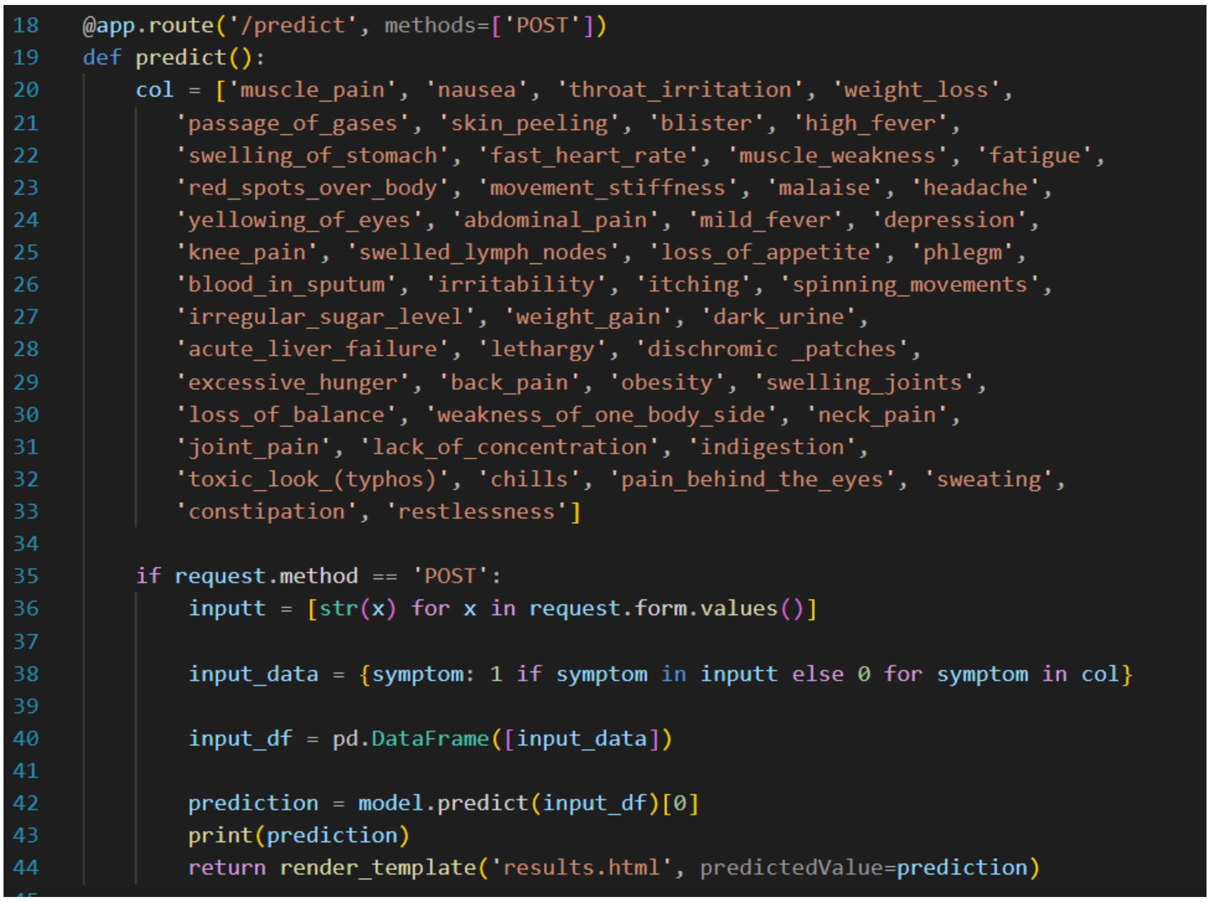
The function retrieves user inputs from a form in the details.html page, and these inputs are stored in the 'inputt' list as strings.

A list 'input_data' is created, initialized with 0s, and iterated through. For each symptom in 'col', if it matches a symptom in 'inputt', a corresponding 1 is set in 'input_data'.

The 'input_data' list is then converted into a DataFrame and used for making predictions with the pre-trained model.

The prediction result is printed and passed to the 'results.html' page using the 'render_template()' function.

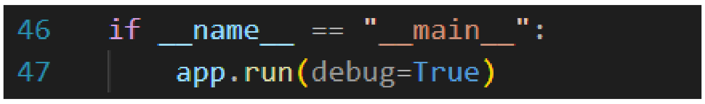
The code establishes a Flask route for predicting disease based on user-input symptoms. It efficiently processes user input, transforms it into a format suitable for the model, obtains predictions, and displays the result on a web page.



```python
@app.route('/predict', methods=['POST'])
def predict():
    col = ['muscle_pain', 'nausea', 'throat_irritation', 'weight_loss',
        'passage_of_gases', 'skin_peeling', 'blister', 'high_fever',
        'swelling_of_stomach', 'fast_heart_rate', 'muscle_weakness', 'fatigue',
        'red_spots_over_body', 'movement_stiffness', 'malaise', 'headache',
        'yellowing_of_eyes', 'abdominal_pain', 'mild_fever', 'depression',
        'knee_pain', 'swelled_lymph_nodes', 'loss_of_appetite', 'phlegm',
        'blood_in_sputum', 'irritability', 'itching', 'spinning_movements',
        'irregular_sugar_level', 'weight_gain', 'dark_urine',
        'acute_liver_failure', 'lethargy', 'dischromic _patches',
        'excessive_hunger', 'back_pain', 'obesity', 'swelling_joints',
        'loss_of_balance', 'weakness_of_one_body_side', 'neck_pain',
        'joint_pain', 'lack_of_concentration', 'indigestion',
        'toxic_look_(typhos)', 'chills', 'pain_behind_the_eyes', 'sweating',
        'constipation', 'restlessness']

    if request.method == 'POST':
        inputt = [str(x) for x in request.form.values()]

        input_data = {symptom: 1 if symptom in inputt else 0 for symptom in col}

        input_df = pd.DataFrame([input_data])

        prediction = model.predict(input_df)[0]
        print(prediction)
        return render_template('results.html', predictedValue=prediction)
```

Main Function:

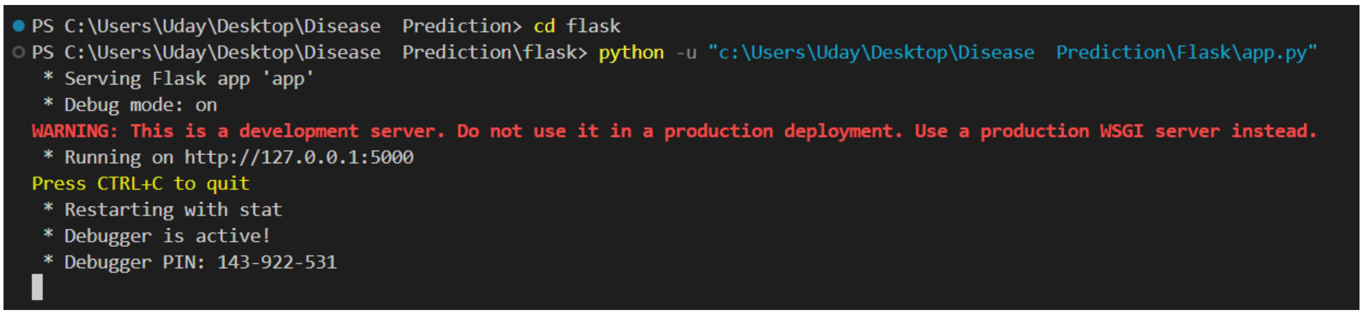
This code sets the entry point of the Flask application. The function “app.run()” is called, which starts the Flask deployment server.



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

Activity 2.3: Run the Web Application

When you execute the "app.py" file, a console or output terminal will display, and a window will open. Copy the URL provided, typically in the format http://127.0.0.1:5000 and paste it into your web browser's address bar. This will allow you to access and interact with the Flask web application locally.



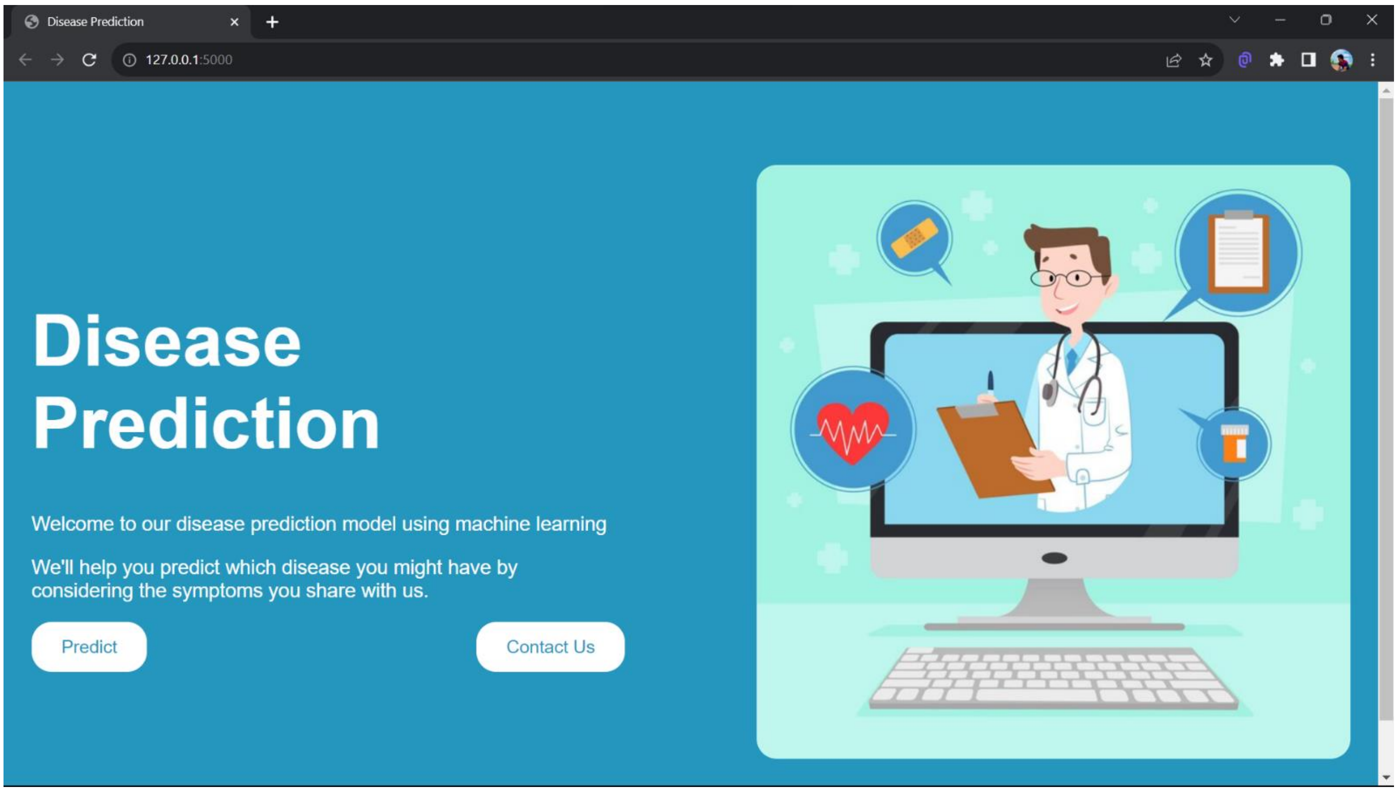
```
● PS C:\Users\Uday\Desktop\Disease  Prediction> cd flask
○ PS C:\Users\Uday\Desktop\Disease  Prediction\flask> python -u "c:\Users\Uday\Desktop\Disease  Prediction\Flask\app.py"
  * Serving Flask app 'app'
  * Debug mode: on
 WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://127.0.0.1:5000
 Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 143-922-531
```

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

It contains some sections such as Predict and 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.

The structure of our "Index.html" is shown as follows.

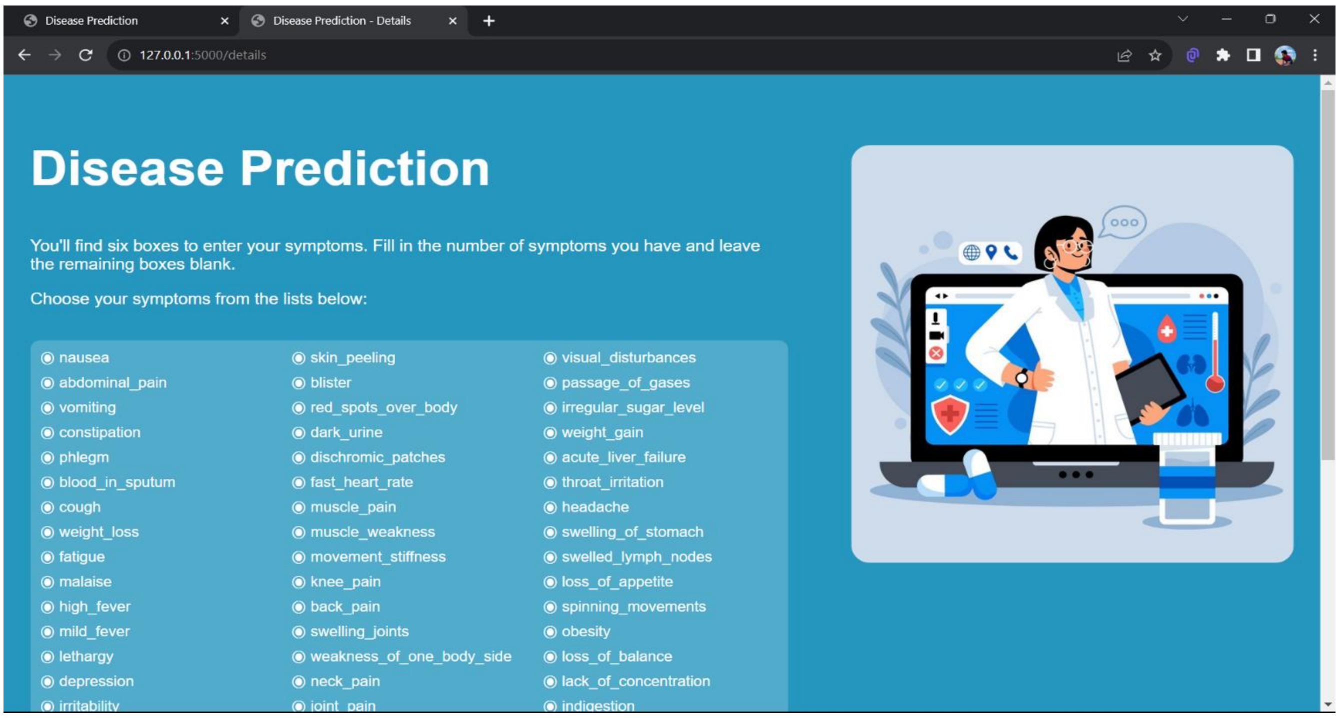


Disease Prediction

Welcome to our disease prediction model using machine learning

We'll help you predict which disease you might have by considering the symptoms you share with us.

Structure of our "Details.html" is shown as follows.

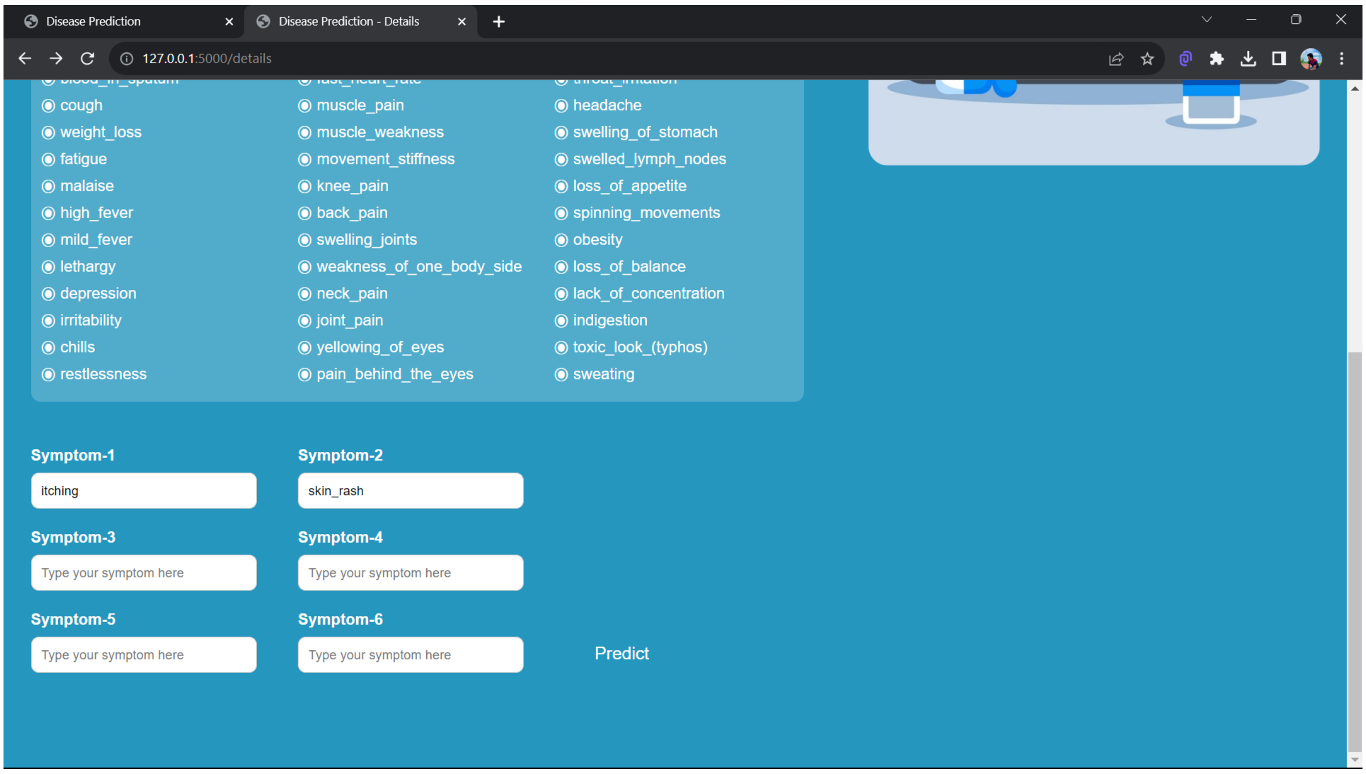


Disease Prediction

You'll find six boxes to enter your symptoms. Fill in the number of symptoms you have and leave the remaining boxes blank.

Choose your symptoms from the lists below:

◉ nausea	◉ skin_peeling	◉ visual_disturbances
◉ abdominal_pain	◉ blister	◉ passage_of_gases
◉ vomiting	◉ red_spots_over_body	◉ irregular_sugar_level
◉ constipation	◉ dark_urine	◉ weight_gain
◉ phlegm	◉ dischromic_patches	◉ acute_liver_failure
◉ blood_in_sputum	◉ fast_heart_rate	◉ throat_irritation
◉ cough	◉ muscle_pain	◉ headache
◉ weight_loss	◉ muscle_weakness	◉ swelling_of_stomach
◉ fatigue	◉ movement_stiffness	◉ swelled_lymph_nodes
◉ malaise	◉ knee_pain	◉ loss_of_appetite
◉ high_fever	◉ back_pain	◉ spinning_movements
◉ mild_fever	◉ swelling_joints	◉ obesity
◉ lethargy	◉ weakness_of_one_body_side	◉ loss_of_balance
◉ depression	◉ neck_pain	◉ lack_of_concentration
◉ irritability	◉ joint_pain	◉ indigestion



cough	muscle_pain	headache
weight_loss	muscle_weakness	swelling_of_stomach
fatigue	movement_stiffness	swelled_lymph_nodes
malaise	knee_pain	loss_of_appetite
high_fever	back_pain	spinning_movements
mild_fever	swelling_joints	obesity
lethargy	weakness_of_one_body_side	loss_of_balance
depression	neck_pain	lack_of_concentration
irritability	joint_pain	indigestion
chills	yellowing_of_eyes	toxic_look_(typhos)
restlessness	pain_behind_the_eyes	sweating

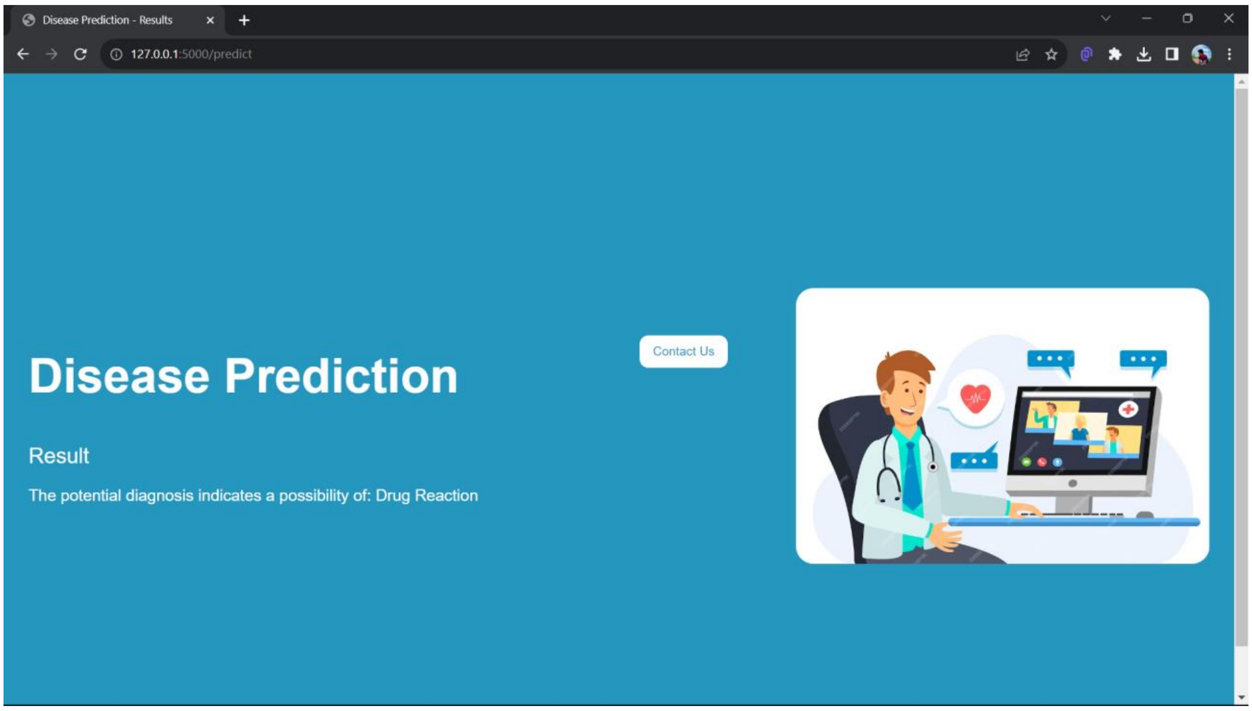
Symptom-1: itching	Symptom-2: skin_rash
Symptom-3: Type your symptom here	Symptom-4: Type your symptom here
Symptom-5: Type your symptom here	Symptom-6: Type your symptom here

We have six input fields available for entering symptoms, and users have the flexibility to fill all six boxes or just one.

The list of symptoms is presented in bullet points above these input fields, and it's important to input symptoms exactly as listed since they correspond to the column names.

Users can enter symptoms like itching, skin rash, and more into the input fields. After entering the symptoms, clicking the "Predict" button initiates the prediction process.

Upon doing so, users will be redirected to the "Results.html" page, where the output will be displayed.


Disease Prediction - Results

127.0.0.1:5000/predict

Disease Prediction

Result

The potential diagnosis indicates a possibility of: Drug Reaction

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

Link to GitHub Repository:

https://github.com/smartinternz02/SI-GuidedProject-611642-1700551037

contact:

Uday Suggula – https://www.linkedin.com/in/uday-suggula-51b614261/

Harshit Bogineni – https://www.linkedin.com/in/harshit-bogineni-b0b481220/