In [2]:
```
# Numpy and pandas for mathematical operations
import numpy as np
import pandas as pd

# To read csv dataset files
import csv

# Regular expression, for pattern matching
import re

# The preprocessing module provides functions for data preprocessing tasks suc
from sklearn import preprocessing

# For Visualization
import seaborn as sns
import matplotlib.pyplot as plt

# train-test split
from sklearn.model_selection import train_test_split

# For building decision tree models, and _tree to access low-level decision of
from sklearn.tree import DecisionTreeClassifier, _tree

# For evaluating model performance using cross_validation
from sklearn.model_selection import cross_val_score

# Import Support Vector Classification from sklearn library for model deployme
from sklearn.svm import SVC

# Remove unecessary warnings
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [3]:
```
!pip install pyttsx3
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Requirement already satisfied: pyttsx3 in c:\users\priya\appdata\roaming\pyt
hon\python311\site-packages (2.98)
Requirement already satisfied: comtypes in c:\users\priya\appdata\roaming\py
thon\python311\site-packages (from pyttsx3) (1.4.10)
Requirement already satisfied: pypiwin32 in c:\users\priya\appdata\roaming\p
ython\python311\site-packages (from pyttsx3) (223)
Requirement already satisfied: pywin32 in c:\programdata\anaconda3\lib\site-
packages (from pyttsx3) (305.1)
```

In [*]:
```
!pip show pyttsx3
```

In [5]:
```
import pyttsx3
```

In [11]:
```
pip install pyttsx3
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Requirement already satisfied: pyttsx3 in c:\users\priya\appdata\roaming\pyt
hon\python311\site-packages (2.98)
Requirement already satisfied: comtypes in c:\users\priya\appdata\roaming\py
thon\python311\site-packages (from pyttsx3) (1.4.10)
Requirement already satisfied: pypiwin32 in c:\users\priya\appdata\roaming\p
ython\python311\site-packages (from pyttsx3) (223)
Requirement already satisfied: pywin32 in c:\programdata\anaconda3\lib\site-
packages (from pyttsx3) (305.1)
Note: you may need to restart the kernel to use updated packages.
```

In [12]:
```
engine = pyttsx3.init()
```

In [13]:
```
!python -m pip install pyttsx3
```

```
Defaulting to user installation because normal site-packages is not writeabl
e
Requirement already satisfied: pyttsx3 in c:\users\priya\appdata\roaming\pyt
hon\python311\site-packages (2.98)
Requirement already satisfied: comtypes in c:\users\priya\appdata\roaming\py
thon\python311\site-packages (from pyttsx3) (1.4.10)
Requirement already satisfied: pypiwin32 in c:\users\priya\appdata\roaming\p
ython\python311\site-packages (from pyttsx3) (223)
Requirement already satisfied: pywin32 in c:\programdata\anaconda3\lib\site-
packages (from pyttsx3) (305.1)
```

In [14]:
```
import pyttsx3
engine = pyttsx3.init()
```

In [15]:
```
def text_to_speech(text):
    # Set properties (optional)
    engine.setProperty('rate', 150)    # Speed percent (can go over 100)
    engine.setProperty('volume', 0.9)  # Volume 0-1

    # Convert text to speech
    engine.say(text)
    engine.runAndWait()
```

In [16]:
```
import pandas as pd
training = pd.read_csv("Training.csv")
testing= pd.read_csv("Testing.csv")
```

In [17]:
```
shape = training.shape
print("Shape of Training dataset: ", shape)
```

```
Shape of Training dataset:  (4920, 133)
```

In [18]:
```
description = training.describe()
description
```

Out[18]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	
count	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.
mean	0.137805	0.159756	0.021951	0.045122	0.021951	0.
std	0.344730	0.366417	0.146539	0.207593	0.146539	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.

8 rows × 132 columns

In [19]:
```
# Information about Dataset
info_df = training.info()
info_df
```

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

In [20]:
```
# To find total number of null values in dataset
null_values_count = training.isnull().sum()
null_values_count
```

Out[20]:
```
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 [21]:
```
# Print First eight rows of the Dataset
training.head(8)
```

Out[21]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	s
0	1	1	1	0	0	0	0	
1	0	1	1	0	0	0	0	
2	1	0	1	0	0	0	0	
3	1	1	0	0	0	0	0	
4	1	1	1	0	0	0	0	
5	0	1	1	0	0	0	0	
6	1	0	1	0	0	0	0	
7	1	1	0	0	0	0	0	

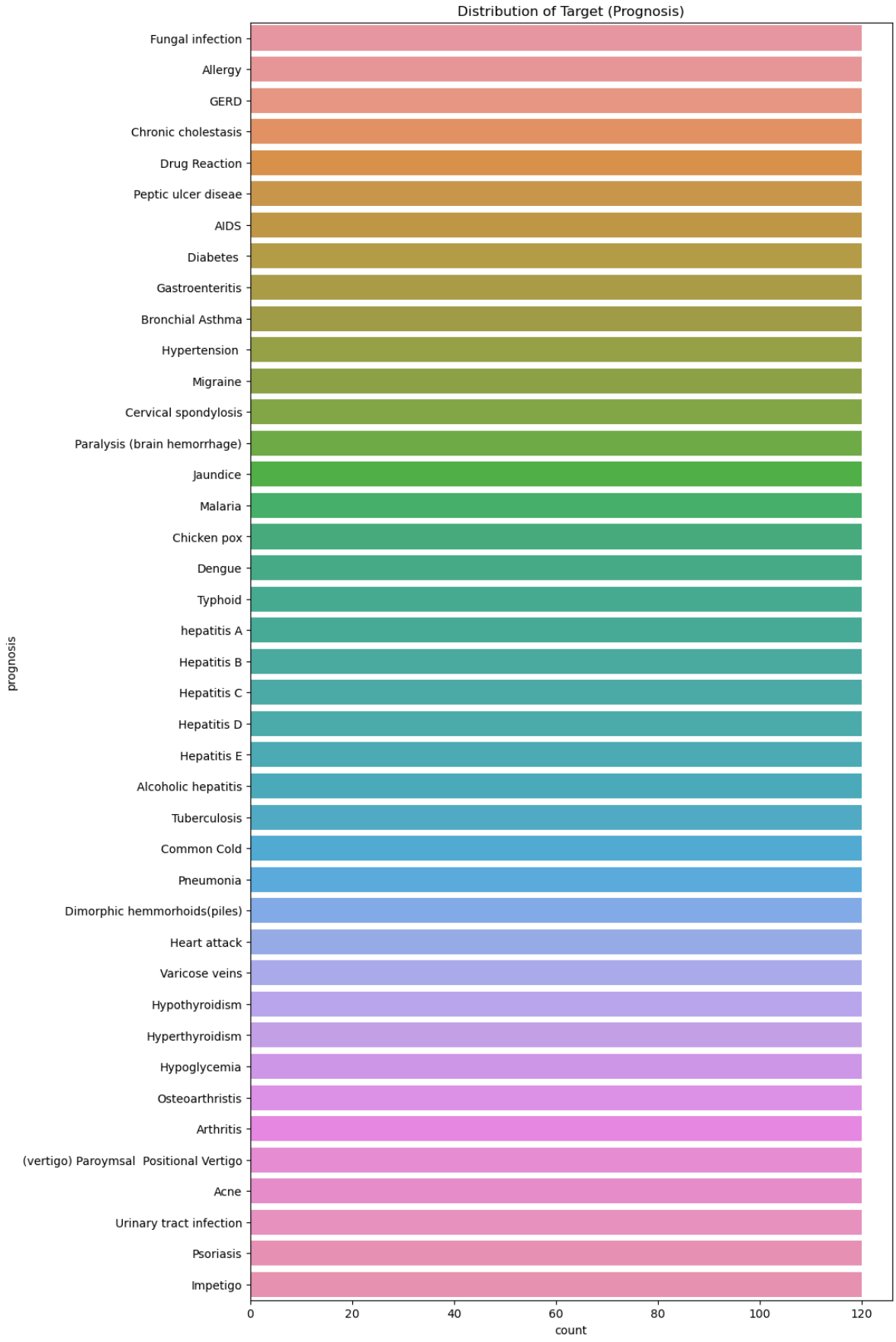
8 rows × 133 columns

In [22]:
```
cols= training.columns
cols= cols[:-1]

# x stores every column data except the last one
x = training[cols]

# y stores the target variable for disease prediction
y = training['prognosis']
```

In [23]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 20))
sns.countplot(y='prognosis', data=training)
plt.title('Distribution of Target (Prognosis)')
plt.show()
```



In [24]:
```
# Grouping Data by Prognosis and Finding Maximum Values
reduced_data = training.groupby(training['prognosis']).max()

# Display the first five rows of the reduced data
reduced_data.head()
```

Out[24]:

prognosis	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	join
(vertigo) Paroymsal Positional Vertigo	0	0	0	0	0	0	
AIDS	0	0	0	0	0	0	
Acne	0	1	0	0	0	0	
Alcoholic hepatitis	0	0	0	0	0	0	
Allergy	0	0	0	1	1	1	

5 rows × 132 columns

In [25]:
```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()


le.fit(y)
y = le.transform(y)
```

In [26]:
```
# Splitting the dataset into training and testing
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, rand

# Features for testing except the last variable
testx    = testing[cols]

# Target variable for Testing
testy    = testing['prognosis']

# Transforming categorical value into numerical labels
testy    = le.transform(testy)
```

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score


# Decision Tree Model Implementation
clf1  = DecisionTreeClassifier()

# Fitting the Training Data
clf = clf1.fit(x_train,y_train)

# Cross-Validation for Model Evaluation
scores = cross_val_score(clf, x_test, y_test, cv=3)

# Print the Mean Score
print("Mean Score: ",scores.mean())
```

```
Mean Score:  0.9759874770651589
```

In [28]:
```
from sklearn.svm import SVC
import numpy as np

# Creating Support Vector Machine Model
model=SVC()

# Train the model on Training Data
model.fit(x_train,y_train)

# Print accuracy for SVM Model on the training set
print("Accuracy score for svm: ", model.score(x_test,y_test))

# Calculate feature importance using the trained Decision tree classifier
importances = clf.feature_importances_

# Sort indices in descending order based on feature importance
indices = np.argsort(importances)[::-1]

# Get feature names corresponding to their importance score
features = cols
```

```
Accuracy score for svm:  1.0
```

In [29]:
```
# Initialize dictionaries to store symptom severity, description, and precauti

severityDictionary=dict()
description_list = dict()
precautionDictionary=dict()

# Dictionary to map symptoms to their indices
symptoms_dict = {}

# Populate symptoms dictionary with indices
for index, symptom in enumerate(x):
       symptoms_dict[symptom] = index

# Function to calculate the overall severity of the symptom
def calc_condition(exp,days):
    sum=0
    for item in exp:
         sum=sum+severityDictionary[item]
    if((sum*days)/(len(exp)+1)>13):
        print("You should take the consultation from doctor. ")
    else:
        print("It might not be that bad but you should take precautions.")

        # Function to read and store symptom descriptions from a CSV file
def getDescription():
    global description_list
    with open("symptom_Description.csv") as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        for row in csv_reader:
            _description={row[0]:row[1]}
            description_list.update(_description)



# Function to read and store symptom severity information from a CSV file
def getSeverityDict():
    global severityDictionary
    with open("Symptom_severity.csv") as csv_file:

        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        try:
            for row in csv_reader:
                _diction={row[0]:int(row[1])}
                severityDictionary.update(_diction)
        except:
            pass
        # Function to read and store symptom precaution information from a CSV
def getprecautionDict():
    global precautionDictionary
    with open("symptom_precaution.csv") as csv_file:

        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        for row in csv_reader:
            _prec={row[0]:[row[1],row[2],row[3],row[4]]}
```

```
            precautionDictionary.update(_prec)
```

In [30]:
```
def getInfo():
    print("-----------------------------------HealthCare ChatBot--------------
    print("\nYour Name? \t\t\t\t",end="->")
    name=input("")
    print("Hello", name)
```

In [31]:
```
def check_pattern(dis_list,inp):
    pred_list=[]
    inp=inp.replace(' ','_')
    patt = f"{inp}"
    regexp = re.compile(patt)
    pred_list=[item for item in dis_list if regexp.search(item)]
    if(len(pred_list)>0):
        return 1,pred_list
    else:
        return 0,[]
```

In [32]:
```
def sec_predict(symptoms_exp):
    df = pd.read_csv('Training.csv')
    X = df.iloc[:, :-1]
    y = df['prognosis']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
    rf_clf = DecisionTreeClassifier()
    rf_clf.fit(X_train, y_train)

    symptoms_dict = {symptom: index for index, symptom in enumerate(X)}
    input_vector = np.zeros(len(symptoms_dict))
    for item in symptoms_exp:
      input_vector[[symptoms_dict[item]]] = 1

    return rf_clf.predict([input_vector])
```

In [34]:
```
def print_disease(node):
    node = node[0]
    val  = node.nonzero()
    disease = le.inverse_transform(val[0])
    return list(map(lambda x:x.strip(),list(disease)))
```

In [ ]:

```
import pyttsx3
from sklearn.tree import _tree
import re
engine = pyttsx3.init()

def tree_to_code(tree, feature_names):
    tree_ = tree.tree_
    feature_name = [
        feature_names[i] if i != _tree.TREE_UNDEFINED else "undefined!"
        for i in tree_.feature
    ]

    chk_dis=",".join(feature_names).split(",")
    symptoms_present = []

    while True:

      # Prompt the user to enter the symptom
        engine.say("\n Enter the symptom you are experiencing \t\t\t",)
        engine.runAndWait()
        print("\nEnter the symptom you are experiencing  \t\t",end="->")
        disease_input = input("")

        conf,cnf_dis=check_pattern(chk_dis,disease_input)
        if conf==1:
            print("searches related to input: ")
            for num,it in enumerate(cnf_dis):
                print(num,")",it)
            if num!=0:
                print(f"Select the one you meant (0 - {num}):  ", end="")
                conf_inp = int(input(""))
            else:
                conf_inp=0

            disease_input=cnf_dis[conf_inp]
            break
        else:
            print("Enter valid symptom.")

    while True:
        try:
            num_days=int(input("Okay. From how many days ? : "))
            break
        except:
            print("Enter valid input.")
    def recurse(node, depth):
        indent = "  " * depth
        if tree_.feature[node] != _tree.TREE_UNDEFINED:
            name = feature_name[node]
            threshold = tree_.threshold[node]

            if name == disease_input:
                val = 1
            else:
                val = 0
            if  val <= threshold:
                recurse(tree_.children_left[node], depth + 1)
            else:
                symptoms_present.append(name)
                recurse(tree_.children_right[node], depth + 1)
        else:
```

```
            present_disease = print_disease(tree_.value[node])

            red_cols = reduced_data.columns
            symptoms_given = red_cols[reduced_data.loc[present_disease].values

            engine.say("Are you experiencing any")
            engine.runAndWait()
            print("Are you experiencing any ")
            symptoms_exp=[]
            for syms in list(symptoms_given):
                inp=""
                engine.say(f"{syms}, are you experiencing it?")
                engine.runAndWait()
                print(syms,"? : ",end='')
                while True:
                    inp=input("")
                    if(inp=="yes" or inp=="no"):
                        break
                    else:
                        print("provide proper answers i.e. (yes/no) : ",end=""
                if(inp=="yes"):
                    symptoms_exp.append(syms)

            second_prediction=sec_predict(symptoms_exp)
            # print(second_prediction)
            calc_condition(symptoms_exp,num_days)
            if(present_disease[0]==second_prediction[0]):
                engine.say("You may have ", present_disease[0])
                engine.runAndWait()
                print("You may have ", present_disease[0])
                print(description_list[present_disease[0]])


            else:
                engine.say(f"You may have {present_disease[0]} or {second_prec
                engine.runAndWait()
                print("You may have ", present_disease[0], "or ", second_predi
                print(description_list[present_disease[0]])
                print(description_list[second_prediction[0]])

            # print(description_list[present_disease[0]])
            precution_list=precautionDictionary[present_disease[0]]
            print("Take following measures : ")
            for  i,j in enumerate(precution_list):
                print(i+1,")",j)


    recurse(0, 1)
getSeverityDict()
getDescription()
getprecautionDict()
getInfo()
tree_to_code(clf,cols)
print("-----------------------------------------------------------------------
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: