

Project Report
On
“Predicting Disease by Symptoms”
Spring 2023 CPSC 531-03 22145
Advanced Database Management
Spring, 2023
Under Guidance of
Prof. Tseng-Ching James Shen
Department of Computer Science



California State University
Fullerton, CA - 92831 May 2023

Prepared By:

Sri Datta Venga S (885176917) sridattavenga14@csu.fullerton.edu

Manish Reddy Kambalapally (885175679) manish.reddy@csu.fullerton.edu

Table of Contents

1) Introduction.....	3
2) Functionalities.....	3
3) Dataset.....	4
4) Architecture & Design (Technology and Tools Used)	6
5) Minimum System Requirements.....	6
6) GitHub Location of Code.....	6
7) Deployment Instructions.....	7
8) Steps to Run the Application.....	15
9) Test Results.....	15
10) Conclusion.....	17

1) Introduction

People presently suffer from a variety of ailments as a result of their surroundings and lifestyle choices. As a result, disease prediction at an early stage has become a key obligation. Doctors, on the other hand, find it difficult to make precise forecasts based on symptoms. The most difficult challenge is precisely predicting illness. The accurate and prompt investigation of any health-related concern is crucial for disease prevention and treatment. The normal way of diagnosis may not be sufficient in the case of a serious illness.

The creation of a Disease Symptoms Prediction based on machine learning (ML) algorithms for sickness prediction could aid in more accurate diagnosis than present approaches. We developed a disease prediction system using Supervised machine learning techniques.

2) Functionalities

- Cluster Creation with 3 Server Nodes:

Using the Google Cloud Service creating a Dataproc cluster with the desired number of worker nodes. Set the appropriate machine types and configurations based on your workload requirements.

- Storage Bucket Creation:

Using the Google Cloud create a storage bucket. Ensuring that the bucket name is globally unique.

- Importing Data to the Cluster and bucket:

To import dataset from internet on to our Hadoop cluster from Hadoop cluster to google storage bucket. We use two commands.

1, `wget https://d37ci6vzurychx.cloudfront.net/misc/taxi+_zone_lookup.csv`;
2, `gsutil cp taxi+_zone_lookup.csv gs://rs-bucket1-dataproc/Data`;

- Jupyter Notebook Integration:

Launching a Jupyter Notebook server on the master node of the Dataproc cluster.

Interacting with the Jupyter Notebook interface to write and execute PySpark code.

- Data Processing using PySpark and Visualization Libraries:

Utilizing PySpark powerful distributed computing capabilities to process the data. Applying transformations and actions on the DataFrame to manipulate, filter, aggregate, join, or transform the data as per requirements.

Use visualization libraries like Matplotlib and Seaborn to create meaningful visualizations of the data.

we're using the k-nearest neighbors (KNN) algorithm to classify the diseases based on the symptoms provided. First , we import the KNeighborsClassifier class from the sklearn.neighbors module. Then, we create an instance of this class with a parameter n_neighbors set to 5. This means that the KNN algorithm will classify each data point based on the 5 closest neighbors to it.

3) Dataset

Reference Link -<https://www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset>

Raw Data-

	A	B	C	D	E	F	G	H
1	Disease	Count of D	Symptom					
2	UMLS:C00	3363	UMLS:C0008031_pain chest					
3			UMLS:C0392680_shortness of breath					
4			UMLS:C0012833_dizziness					
5			UMLS:C0004093_asthenia					
6			UMLS:C0085639_fall					
7			UMLS:C0039070_syncope					
8			UMLS:C0042571_vertigo					
9			UMLS:C0038990_sweat^UMLS:C0700590_sweating increased					
10			UMLS:C0030252_palpitation					
11			UMLS:C0027497_nausea					
12			UMLS:C0002962_angina pectoris					
13			UMLS:C0438716_pressure chest					
14	UMLS:C00	1421	UMLS:C0032617_polyuria					
15			UMLS:C0085602_polydypsia					
16			UMLS:C0392680_shortness of breath					
17			UMLS:C0008031_pain chest					
18			UMLS:C0004093_asthenia					
19			UMLS:C0027497_nausea					
20			UMLS:C0085619_orthopnea					
21			UMLS:C0034642_rale					
22			UMLS:C0038990_sweat^UMLS:C0700590_sweating increased					
23			UMLS:C0241526_unresponsiveness					
24			UMLS:C0856054_mental status changes					
25			UMLS:C0042571_vertigo					
26			UMLS:C0042963_vomiting					
27			UMLS:C0553668_labored breathing					
28	UMLS:C00	1337	UMLS:C0424000_feeling suicidal					
29			UMLS:C0438696_suicidal					

By splitting the data into separate train and test datasets, the train dataset is used to train or "teach" the model by adjusting its parameters based on the input data and known output values. The test dataset, on the other hand, is used to assess how well the trained model performs on unseen data.

Train data-

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	itching	skin_rash	nodal_skin_tender	continuous_sweating	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	muscle_wasting	vomiting	burning_mouth	spotting	fatigue	weight_gain	anxiety
2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0
23	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0
24	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0

Test data-

[illegible]

4) Architecture & Design

Tools and technologies used –

- ❖ Google Cloud Service (Dataproc)
- ❖ Hadoop Cluster
- ❖ Google storage bucket
- ❖ Jupyter notebook
- ❖ Pyspark
- ❖ ML algorithm
- ❖ Matplotlib
- ❖ Seaborn
- ❖ Vector assembler
- ❖ NumPy
- ❖ pandas

5) Minimum System Requirements

Every system that is planned to be a part of the cluster must satisfy the following hardware requirements:

- ❖ Google cloud service Account
- ❖ 1.5 GB RAM (2GB recommended)
- ❖ 20 GB Disk Space
- ❖ Hypervisor to support virtualization

6) GitHub Location of Code



[Disease prediction using symptoms ADBMS-final-project](#)

7) Deployment Instructions

Dataproc is fully managed big data cluster or big data as service provider.

The screenshot shows the Google Cloud Dataproc Clusters page. The top navigation bar includes the Google Cloud logo, the project name 'My First Project', and a search bar with the text 'dataproc'. The left sidebar contains a navigation menu with options: Clusters (selected), Jobs, Workflows, Autoscaling policies, Serverless, Batches, Metastore Services, Metastore, Federation, Utilities, Component exchange, and Release Notes. The main content area is titled 'Clusters' and includes buttons for 'CREATE CLUSTER', 'REFRESH', 'START', 'STOP', 'DELETE', 'REGIONS', and '+ 5 RECOMMENDED ALERTS'. Below these buttons is a filter bar with the text 'Filter Search clusters, press Enter'. The main table lists the following cluster:

<input type="checkbox"/>	Name ↑	Status	Region	Zone	Total worker nodes	Scheduled deletion	Cloud Storage stage
<input type="checkbox"/>	cluster-63e4	Running	us-central1	us-central1-a	3	Off	rs-bucket1-datapr

On the right side of the table, there is a 'PERMISSIONS' link and a message box that says 'No clusters selected' and 'Please'.

1-Creating a cluster.

Google Cloud My First Project dataproc Search

Dataprocc

Jobs on Clusters

- Clusters
- Jobs
- Workflows
- Autoscaling policies

Serverless

- Batches

Metastore Services

- Metastore
- Federation

Utilities

- Component exchange
- Release Notes

Create a Dataprocc cluster on Compute Engine

- Set up cluster**
Begin by providing basic information.
- Configure nodes (optional)**
Change node compute and storage capabilities.
- Customize cluster (optional)**
Add cluster properties, features, and actions.
- Manage security (optional)**
Change access, encryption, and security settings.

CREATE **CANCEL**

EQUIVALENT COMMAND LINE

Name

Cluster Name * cluster-1b64

Location

Region * us-central1 Zone * us-central1-a

Cluster type

☒ **Standard (1 master, N workers)**

☐ **Single Node (1 master, 0 workers)**
Provides one node that acts as both master and worker. Good for proof-of-concept or small-scale processing

☐ **High Availability (3 masters, N workers)**
Hadoop High Availability mode provides uninterrupted YARN and HDFS operations despite single-node failures or reboots

Autoscaling

Automates cluster resource management based on an autoscaling policy.

Policy None

Our project has 3 worker node Hadoop cluster.

Google Cloud My First Project dataproc Search

Dataprocc

Jobs on Clusters

- Clusters
- Jobs
- Workflows
- Autoscaling policies

Serverless

- Batches

Metastore Services

- Metastore
- Federation

Utilities

- Component exchange
- Release Notes

Cluster details **SUBMIT JOB** **REFRESH** **START** **STOP** **DELETE** **VIEW LOGS**

Consider using Auto Zone rather than selecting a zone manually. See <https://cloud.google.com/dataprocc/docs/concepts/configuring-clusters/auto-zone>

Name cluster-63e4

Cluster UUID 3d0f0b8d-5604-4e3b-9d6d-f6d5de48e925

Type Dataprocc Cluster

Status Running

MONITORING **JOB** **VM INSTANCES** **CONFIGURATION** **WEB INTERFACES**

Filter Filter instances

	Name	Role	
✓	cluster-63e4-m	Master	SSH
✓	cluster-63e4-w-0	Worker	
✓	cluster-63e4-w-1	Worker	
✓	cluster-63e4-w-2	Worker	

EQUIVALENT REST

Creating google storage bucket.

► We create bucket to save pyspark code and dataset to process the data.

Google Cloud My First Project buc Search

Cloud Storage Bucket details REFRESH HELP ASSISTANT LEARN

Buckets

Monitoring NEW

Settings

rs-bucket1-dataproc

Location: us (multiple regions in United States) Storage class: Standard Public access: Not public Protection: None

OBJECTS CONFIGURATION PERMISSIONS PROTECTION LIFECYCLE OBSERVABILITY NEW INVENTORY REPORTS NEW

Buckets > rs-bucket1-dataproc > notebooks > jupyter

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA MANAGE HOLDS DOWNLOAD DELETE

Filter by name prefix only Filter Filter objects and folders Show deleted data

Name	Size	Type	Created	Storage class	Last modified	Public access	Version
Prediction.ipynb	1.3 MB	application/octet-stream	Apr 26, 2023, 7:07:22 PM	Standard	Apr 26, 2023, 7:07:22 PM	Not public	—
Untitled.ipynb	135.6 KB	application/octet-stream	Apr 26, 2023, 7:59:09 PM	Standard	Apr 26, 2023, 7:59:09 PM	Not public	—
pyspark.ipynb	986 KB	application/octet-stream	May 2, 2023, 7:10:32 PM	Standard	May 2, 2023, 7:10:32 PM	Not public	—
pysparkcode.ipynb	986 KB	application/octet-stream	May 1, 2023, 1:19:11 PM	Standard	May 1, 2023, 1:19:11 PM	Not public	—

Marketplace

Release Notes

We started setting up the Hadoop cluster and then we connected our Hadoop cluster through jupyter notebook interface.

psychic-order-384800 > cluster-63e4 Sign out

File Edit View Run Kernel Git Tabs Settings Help

Launcher

Filter files by name

/ GCS /

Name	Last Modified
Prediction.ipynb	6 days ago
pyspark.ipynb	5 hours ago
pysparkcode.ipynb	a day ago
Untitled.ipynb	6 days ago

Python 3 PySpark R spylon-kernel

Console

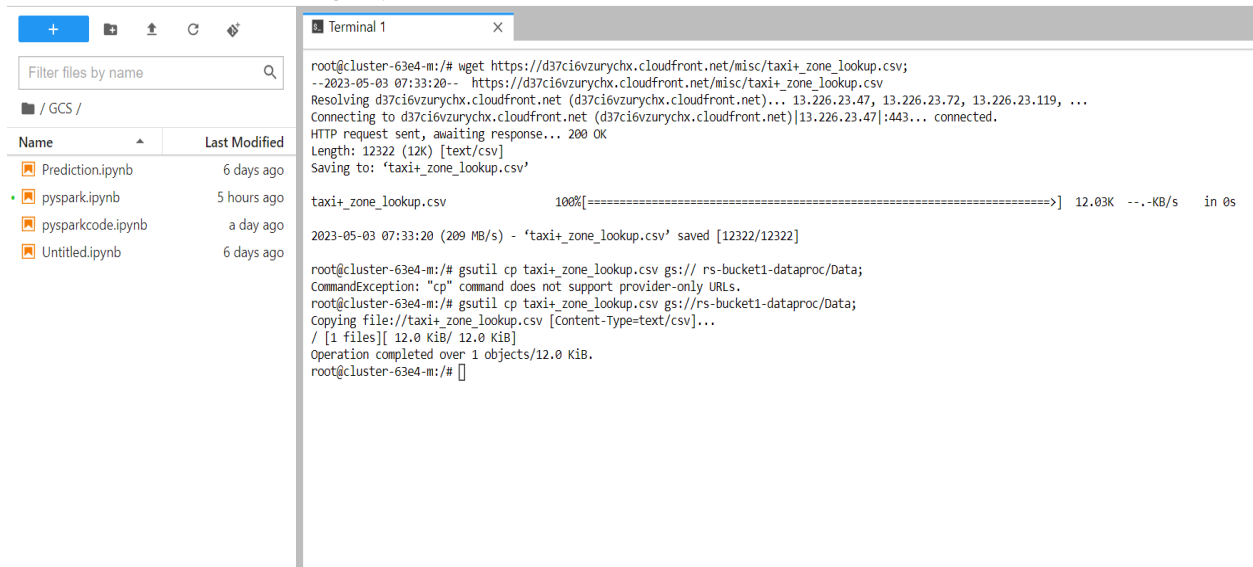
Python 3 PySpark R spylon-kernel

Other

Terminal Text File Markdown File Show Contextual Help

Simple 0 2

Launched



OBJECTS									
CONFIGURATION									
PERMISSIONS									
PROTECTION									
LIFECYCLE									
OBSERVABILITY NEW									
INVENTORY REPORTS NEW									
Buckets > rs-bucket1-dataproc > Data									
UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA MANAGE HOLDS DOWNLOAD DELETE									
Filter by name prefix only Filter Filter objects and folders Show deleted data									
Name	Size	Type	Created	Storage class	Last modified	Public access	Version history		
Testing.csv	13.5 KB	text/csv	Apr 25, 2023, 6:09:30 PM	Standard	Apr 25, 2023, 6:09:30 PM	Not public	—		
Training.csv	1.3 MB	text/csv	Apr 25, 2023, 6:09:31 PM	Standard	Apr 25, 2023, 6:09:31 PM	Not public	—		
raw.csv	60.6 KB	text/csv	Apr 25, 2023, 6:13:40 PM	Standard	Apr 25, 2023, 6:13:40 PM	Not public	—		
taxi+_zone_lookup.csv	12 KB	text/csv	May 3, 2023, 12:36:02 AM	Standard	May 3, 2023, 12:36:02 AM	Not public	—		

Analyzing data using pyspark through jupyter notebook interface.

- PySpark code to read a CSV file stored in Google Cloud Storage and load it into a DataFrame.

Importing the required module

Creating a SparkSession

Reading the CSV file

Configuring Spark SQL

Displaying the first few rows of the DataFrame

```
from pyspark.sql import SparkSession
```

```
# Creating a SparkSession
```

```
spark = SparkSession.builder.appName("ReadCSV").getOrCreate()
```

```
# Reading the CSV file
```

```
df = spark.read.format("csv").option("header", "true").load("gs://rs-bucket1-dataproc/Data/Training.csv")
```

```
spark = SparkSession.builder.config("spark.sql.debug.maxToStringFields", 100).getOrCreate()
```

```
# Displaying the first few rows of the DataFrame
```

```
df.show()
```

- ❖ Reading the training CSV file from GCS bucket:

```
train_df = spark.read.format("csv").option("header", "true").load("gs://rs-bucket1-dataproc/Data/Training.csv")
```

- ❖ Reading the testing CSV file from GCS bucket:

```
test_df = spark.read.format("csv").option("header", "true").load("gs://rs-bucket1-dataproc/Data/Testing.csv")
```

Importing the required modules:

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

Visualizing the value counts using a countplot:

```
sns.set_theme(style="darkgrid")
```

```
plt.figure(figsize=(12, 30))
```

```
plt.xticks(rotation=90)
```

```
sns.countplot(y="prognosis", data=train_df.toPandas())
```

This sets the plot style, adjusts the figure size, and rotates the x-axis labels. Then, it creates a countplot using seaborn's countplot() function, with the "prognosis"

column as the y-axis and the "train_df" DataFrame converted to a Pandas DataFrame using toPandas().The code visualizes the distribution of values in the "prognosis" column using a horizontal bar plot (countplot) for the training DataFrame.

In PySpark, a VectorAssembler is a feature transformer that combines a given list of columns into a single vector column. The resulting vector column can be used as input for machine learning algorithms.

```
from pyspark.ml.feature import VectorAssembler
```

Modeling using the K-Nearest Neighbors (KNN) algorithm for classification.

- ❖ importing the necessary library
- ❖ Creating a KNN classifier object
- ❖ Splitting the data into training and testing sets
- ❖ Fitting the KNN classifier to the training data
- ❖ Making predictions
- ❖ Printing the prediction results
- ❖ Computing the accuracy of the model

Modelling

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors = 3) # k = 5
```

```
x_train, y_train = train_df.loc[:,train_df.columns != "prognosis"],  
train_df.loc[:, "prognosis"]
```

```
x_test, y_test = test_df.loc[:,train_df.columns != "prognosis"],  
test_df.loc[:, "prognosis"]
```

```
knn.fit(x_train, y_train)
```

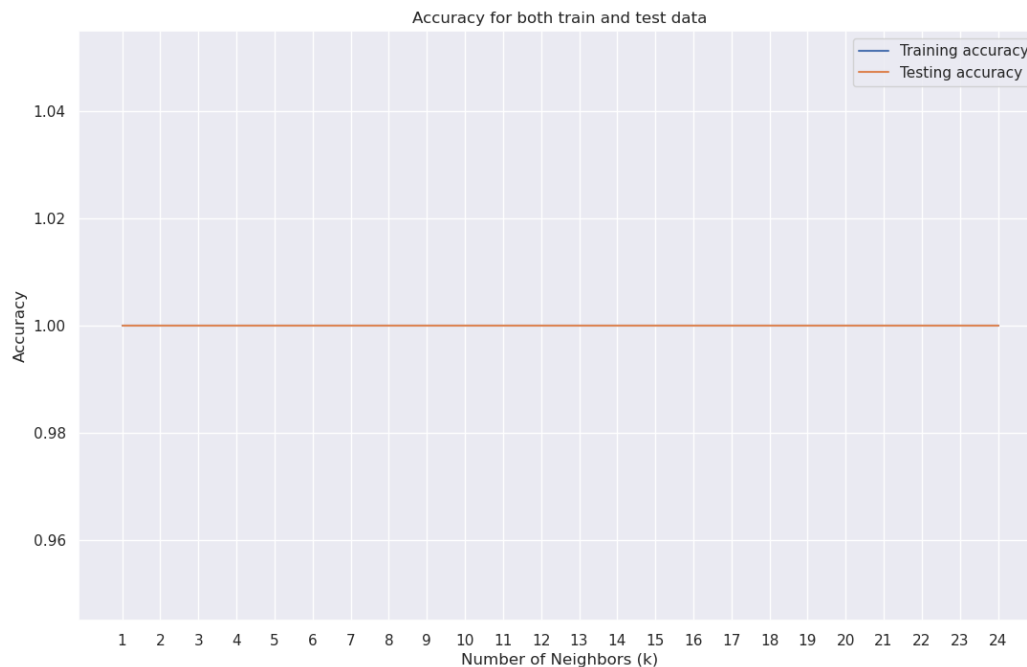
```
prediction = knn.predict(x_test)
```

```
print("Prediction list: {}".format(prediction[0:50]))
```

```
print("With KNN (K=5) accuracy is: ",knn.score(x_test, y_test))
```

Output:

```
Prediction list: ['Fungal infection' 'Allergy' 'GERD' 'Chronic cholestasis'
' 'Drug Reaction'
'Peptic ulcer disease' 'AIDS' 'Diabetes ' 'Gastroenteritis'
'Bronchial Asthma' 'Hypertension ' 'Migraine' 'Cervical spondylosis'
'Paralysis (brain hemorrhage)' 'Jaundice' 'Malaria' 'Chicken pox'
'Dengue' 'Typhoid' 'hepatitis A' 'Hepatitis B' 'Hepatitis C'
'Hepatitis D' 'Hepatitis E' 'Alcoholic hepatitis' 'Tuberculosis'
'Common Cold' 'Pneumonia' 'Dimorphic hemmorhoids(piles)' 'Heart attack'
'Varicose veins' 'Hypothyroidism' 'Hyperthyroidism' 'Hypoglycemia'
'Osteoarthritis' 'Arthritis' '(vertigo) Paroymsal Positional Vertigo'
'Acne' 'Urinary tract infection' 'Psoriasis' 'Impetigo'
'Fungal infection']
With KNN (K=5) accuracy is: 1.0
```



A Decision Tree Classifier is another type of machine learning algorithm used for classification tasks. The accuracy of the decision tree classifier on the test data is 0.9761904761904762, which means that it correctly predicted the diagnosis of 97.6% of the patients in the test set.

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state = 42)

dt.fit(x_train, y_train)
```

```
dt.predict(x_test)
```

```
dt.score(x_test, y_test)
```

Output:

```
0.9761904761904762
```

8) Steps to Run the Application

To run the application, select the cluster and run the cluster and then go to web interface and select the jupyter select the notebook files and run all run cells on jupyter notebook and see the results.

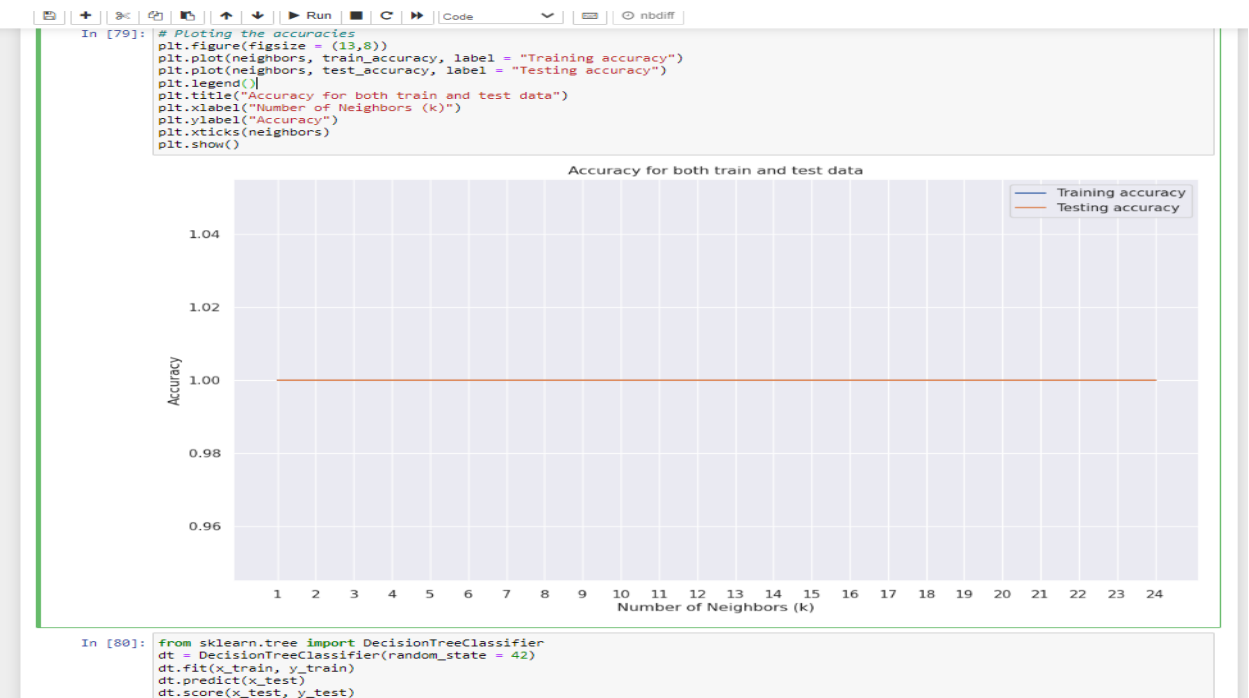
9) Test Results

- ▶ we're using the k-nearest neighbors (KNN) algorithm to classify the diseases based on the symptoms provided.
- ▶ First, we import the KNeighborsClassifier class from the sklearn.neighbors module. Then, we create an instance of this class with a parameter n_neighbors set to 5. This means that the KNN algorithm will classify each data point based on the 5 closest neighbors to it.
- ▶ Next, we split the training and testing data into separate data frames, with the x_train and x_test data frames containing all columns except the "prognosis" column, which is used as the target variable in our classification. The y_train and y_test data frames contain only the "prognosis" column.
- ▶ We then fit the KNN algorithm to the training data using the fit() method. Finally, we use the predict() method to make predictions on the testing data and print out the first 20 predictions. We also calculate the accuracy of the model using the score() method, which compares the predicted values to the actual values in the testing data set. In this case, the accuracy is 1.0,

indicating that the model is predicting the correct diagnosis for all the cases in the testing data set.

```
In [38]: # Modelling
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3) # k = 5
x_train, y_train = train_df.loc[:,train_df.columns != "prognosis"], train_df.loc[:, "prognosis"]
x_test, y_test = test_df.loc[:,train_df.columns != "prognosis"], test_df.loc[:, "prognosis"]
knn.fit(x_train, y_train)
prediction = knn.predict(x_test)
print("Prediction list: {}".format(prediction[0:50]))
print("With KNN (K=5) accuracy is: ",knn.score(x_test, y_test))

Prediction list: ['Fungal infection' 'Allergy' 'GERD' 'Chronic cholestasis' 'Drug Reaction'
'Peptic ulcer disease' 'AIDS' 'Diabetes' 'Gastroenteritis'
'Bronchial Asthma' 'Hypertension' 'Migraine' 'Cervical spondylosis'
'Paralysis (brain hemorrhage)' 'Jaundice' 'Malaria' 'Chicken pox'
'Dengue' 'Typhoid' 'hepatitis A' 'Hepatitis B' 'Hepatitis C'
'Hepatitis D' 'Hepatitis E' 'Alcoholic hepatitis' 'Tuberculosis'
'Common Cold' 'Pneumonia' 'Dimorphic hemorrhoids(piles)' 'Heart attack'
'Varicose veins' 'Hypothyroidism' 'Hyperthyroidism' 'Hypoglycemia'
'Osteoarthritis' 'Arthritis' '(vertigo) Parosymal Positional Vertigo'
'Acne' 'Urinary tract infection' 'Psoriasis' 'Impetigo'
'Fungal infection']
With KNN (K=5) accuracy is: 1.0
```



Decision Tree Classifier is another type of machine learning algorithm used for classification tasks. The accuracy of the decision tree classifier on the test data is 0.9761904761904762, which means that it correctly predicted the diagnosis of 97.6% of the patients in the test set.

10) Conclusion

The method of forecasting disease based on symptoms weighted KNN model has the highest accuracy of 100% for disease. We could simply manage the medical resources required for treatment once the sickness was predicted. This concept would help to reduce the expense of treating the sickness while also improving the recovery process. The results show that the proposed system provides an accuracy of 100% which is higher than that of the other algorithm. It is highly believed that the proposed system can reduce the risk of chronic diseases by diagnosing them earlier and also reduces the cost of diagnosis, treatment, and doctor consultation.