Practical No:01
Name:Anuj Shailendra Naikodi
Roll No:41

```
import re

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

import nltk

nltk.download('stopwords')

text = """Natural language processing (NLP) is a subfield of linguistics, computer science,

and artificial intelligence concerned with the interactions between computers and human language."""

print("Original Text:\n", text)

text = text.lower()

tokens = re.findall(r'\b[a-z]+\b', text)

print("\nTokens:\n", tokens)

stop_words = set(stopwords.words('english'))

filtered_tokens = [w for w in tokens if w not in stop_words]

print("\nAfter Stopword Removal:\n", filtered_tokens)

stemmer = PorterStemmer()

stemmed_tokens = [stemmer.stem(w) for w in filtered_tokens]

print("\nAfter Stemming:\n", stemmed_tokens)
```

Output:

```
Original Text:
 Natural language processing (NLP) is a subfield of linguistics, computer science,
and artificial intelligence concerned with the interactions between computers and human language.

Tokens:
 ['natural', 'language', 'processing', 'nlp', 'is', 'a', 'subfield', 'of', 'linguistics', 'computer', 'science', 'and', 'artificial', 'intelligence', 'concerned', 'with', 't

After Stopword Removal:
 ['natural', 'language', 'processing', 'nlp', 'subfield', 'linguistics', 'computer', 'science', 'artificial', 'intelligence', 'concerned', 'interactions', 'computers', 'huma

After Stemming:
 ['natur', 'languag', 'process', 'nlp', 'subfield', 'linguist', 'comput', 'scienc', 'artifici', 'intellig', 'concern', 'interact', 'comput', 'human', 'languag']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Practical No:02
Name:Anuj Shailendra Naikodi
Roll No:41

```
from collections import defaultdict
import re

documents = {
    1: "Natural language processing enables computers to understand human language.",
    2: "Machine learning and artificial intelligence are key components of NLP.",
    3: "Text mining involves analyzing large volumes of text data.",
    4: "Deep learning methods are used in computer vision and NLP."
}

inverted_index = defaultdict(set)
for doc_id, text in documents.items():
    words = re.findall(r'\b\w+\b', text.lower())
    for word in words:
        inverted_index[word].add(doc_id)
print("Inverted Index:")
for word, doc_ids in sorted(inverted_index.items()):
    print(f"{word}: {sorted(doc_ids)}")
def search(query):
    query_word = query.lower()
    if query_word in inverted_index:
        print(f"\nDocuments containing '{query_word}':
{sorted(inverted_index[query_word])}")
    else:
        print(f"\nNo documents found containing '{query_word}'.")

search("NLP")
```

search("learning")

search("python")

Output:

```
Inverted Index:
analyzing: [3]
and: [2, 4]
are: [2, 4]
artificial: [2]
components: [2]
computer: [4]
computers: [1]
data: [3]
deep: [4]
enables: [1]
human: [1]
in: [4]
intelligence: [2]
involves: [3]
key: [2]
language: [1]
large: [3]
learning: [2, 4]
machine: [2]
methods: [4]
mining: [3]
natural: [1]
nlp: [2, 4]
of: [2, 3]
processing: [1]
text: [3]
to: [1]
understand: [1]
used: [4]
vision: [4]
volumes: [3]

Documents containing 'nlp': [2, 4]

Documents containing 'learning': [2, 4]

No documents found containing 'python'.
```

Practical No:03
Name:Anuj Shailendra Naikodi
Roll No:41

```
import pandas as pd


data = pd.DataFrame({
    'Age': [67, 57, 43, 71, 36],
    'Sex': [0, 1, 0, 1, 0],
    'HeartDisease': [1, 1, 0, 0, 1]
})
print("Sample Dataset:\n", data)


p_hd1 = data['HeartDisease'].mean()
p_hd0 = 1 - p_hd1
print(f"\nP(HeartDisease=1) = {p_hd1:.2f}, P(HeartDisease=0) = {p_hd0:.2f}")


age_hd1 = data[data['HeartDisease']==1]['Age'].value_counts(normalize=True)
age_hd0 = data[data['HeartDisease']==0]['Age'].value_counts(normalize=True)
print("\nP(Age|HeartDisease=1):\n", age_hd1)
print("\nP(Age|HeartDisease=0):\n", age_hd0)


def predict_heart_disease(age):
    ph1 = p_hd1 * age_hd1.get(age, 0.01)
    ph0 = p_hd0 * age_hd0.get(age, 0.01)
    prob_hd1 = ph1 / (ph1 + ph0)
    return prob_hd1


age_test = 57
prob = predict_heart_disease(age_test)
print(f"\nPredicted probability for HeartDisease given Age={age_test}: {prob:.2f}")
```

Output:

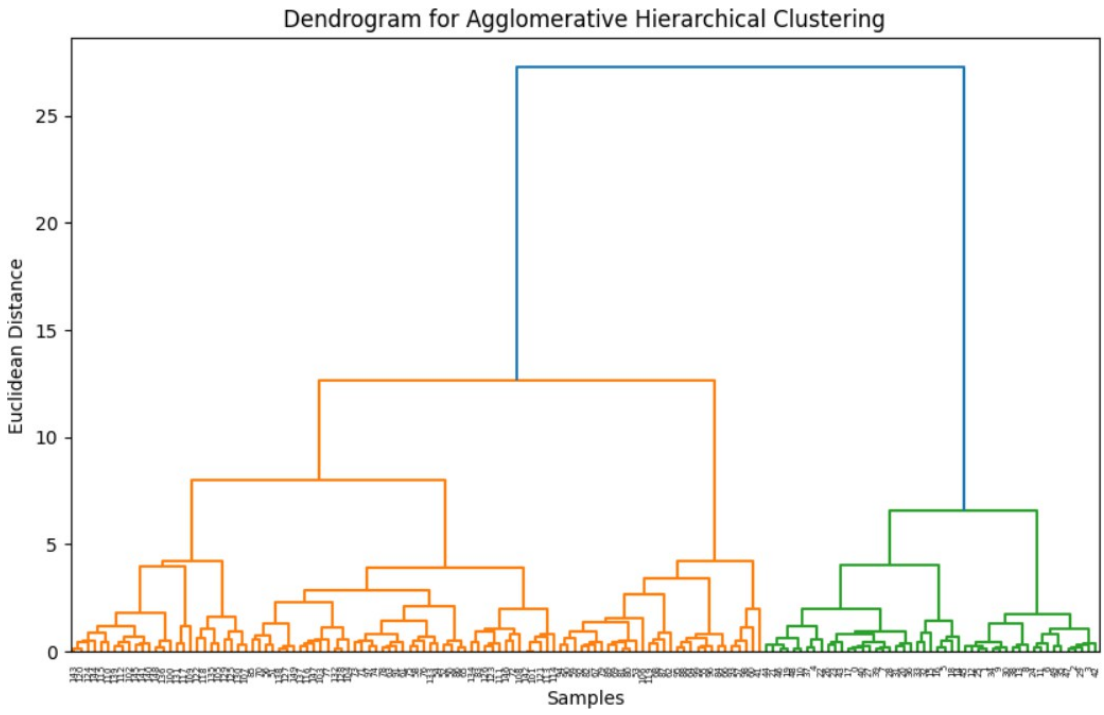
```
Sample Dataset:
    Age  Sex  HeartDisease
0   67    0             1
1   57    1             1
2   43    0             0
3   71    1             0
4   36    0             1

P(HeartDisease=1) = 0.60, P(HeartDisease=0) = 0.40

P(Age|HeartDisease=1):
 Age
67    0.333333
57    0.333333
36    0.333333
Name: proportion, dtype: float64

P(Age|HeartDisease=0):
 Age
43    0.5
71    0.5
Name: proportion, dtype: float64

Predicted probability for HeartDisease given Age=57: 0.98
```

Practical No:04
Name:Anuj Shailendra Naikodi
Roll No:41

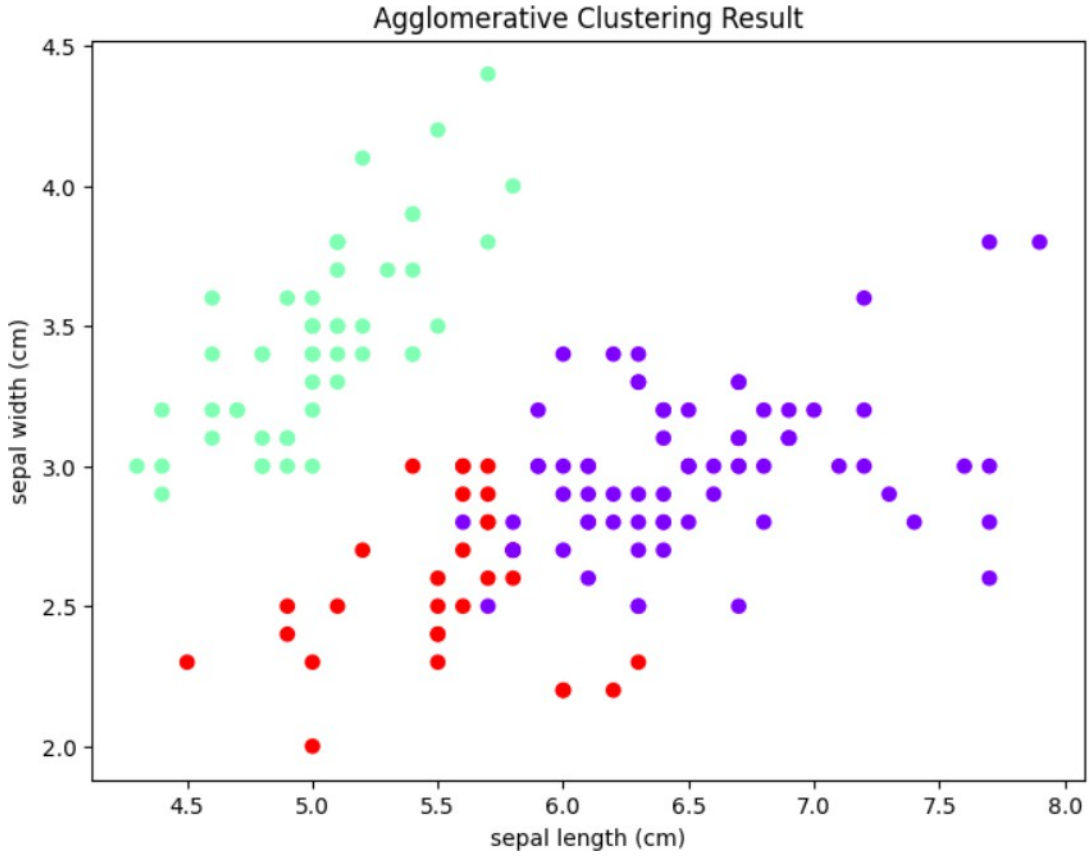
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

iris = load_iris()
X = iris.data
feature_names = iris.feature_names

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

linked = linkage(X_scaled, method='ward')

plt.figure(figsize=(10, 6))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title("Dendrogram for Agglomerative Hierarchical Clustering")
plt.xlabel("Samples")
plt.ylabel("Euclidean Distance")
plt.show()

cluster = AgglomerativeClustering(n_clusters=3, linkage='ward')
labels = cluster.fit_predict(X_scaled)

df = pd.DataFrame(X, columns=feature_names)
```

```
df['Cluster'] = labels
print("\nClustered Data (first 10 rows):\n", df.head(10))


plt.figure(figsize=(8,6))
plt.scatter(df.iloc[:,0], df.iloc[:,1], c=df['Cluster'], cmap='rainbow')
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
plt.title("Agglomerative Clustering Result")
plt.show()
```

Output:



```
Clustered Data (first 10 rows):
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                 5.1               3.5                1.4               0.2
1                 4.9               3.0                1.4               0.2
2                 4.7               3.2                1.3               0.2
3                 4.6               3.1                1.5               0.2
4                 5.0               3.6                1.4               0.2
5                 5.4               3.9                1.7               0.4
6                 4.6               3.4                1.4               0.3
7                 5.0               3.4                1.5               0.2
8                 4.4               2.9                1.4               0.2
9                 4.9               3.1                1.5               0.1

   Cluster
0        1
1        1
2        1
3        1
4        1
5        1
6        1
7        1
8        1
9        1
```



Practical No:05
Name:Anuj Shailendra Naikodi
Roll No:41

```
import numpy as np
import pandas as pd


graph = {
    "A": ["B", "C"],
    "B": ["C"],
    "C": ["A"],
    "D": ["C"]
}


pages = list(graph.keys())
N = len(pages)
damping = 0.85
pr = {page: 1/N for page in pages}


iterations = 100
for i in range(iterations):
    new_pr = {}
    for page in pages:
        rank_sum = 0
        for p in pages:
            if page in graph[p]:
                rank_sum += pr[p] / len(graph[p])
        new_pr[page] = (1 - damping) / N + damping * rank_sum
    pr = new_pr


pr_df = pd.DataFrame(list(pr.items()), columns=["Page", "PageRank"])
```

```
pr_df = pr_df.sort_values(by="PageRank", ascending=False)
print(pr_df)
```

Output:

```
  Page  PageRank
2    C  0.394149
0    A  0.372527
1    B  0.195824
3    D  0.037500
```