**A MINI PROJECT REPORT**

**ON**

**“GLASS CLASSIFICATION USING KNN”**

Submitted to

SAVITRIBAI PHULE PUNE UNIVERSITY

in completion of

**LAB PRACTICE 2**

**(B.E Computer Engineering)**

**BY**

Shruti Houji Exam No :405203

Rhishabh Hattarki Exam No : 405204

Sahil Dixit Exam No : 405205

Sanika Patil Exam No : 405206



Department of Computer Engineering

Sinhgad College of Engineering, Pune-41

**Accredited by NAAC with grade ‘A’**

**YEAR 2019-20**

**CERTIFICATE**



Sinhgad Technical Education Society,

Department of Computer Engineering

Sinhgad College of Engineering, Pune-41

**Accredited by NAAC with grade ‘A’**

**“GLASS CLASSIFICATION USING KNN”**

Submitted to

SAVITRIBAI PHULE PUNE UNIVERSITY

in completion of

**LAB PRACTICE 2**

**(B.E Computer Engineering)**

BY

Shruti Houji Exam No :405203

Rhishabh Hattarki Exam No : 405204

Sahil Dixit Exam No : 405205

Sanika Patil Exam No : 405206

Prof. K.B. Satpute Prof. M. P. Wankhade

Internal Guide Head of Dept.

Department of Computer Engineering Department of Computer Engineering

Dr. S.D. Lokhande

Principal

SCOE, Pune

**CONTENTS**

# TITLE PAGE NO

# 

Certificate I

Acknowledgement II

Abstract III

List of Figures IV

List of Tables V

1. **INTRODUCTION** 
   1. Background And Basics
   2. Problem Statement
      1. Scope Statement
2. **PROJECT PLANNING & MANAGEMENT**
   1. Hardware requirement
   2. Software requirement
   3. Process Modelling

**3. ANALYSIS & DESIGN**

3.1 Use-Case Diagrams

3.2 Sequence Diagrams

3.3 Activity Diagram

3.4 DFD0 & DFD1

**4. IMPLEMENTION & CODING**

* 1. Dataset

4.2 Algorithms and coding

4.2.1 Preprocessing the dataset

4.2.2 Training the algorithm

4.2.2.1 KNN Classifier

4.2.2.2 K-Means Clustering

4.3 GUI Design /screen shots

**5. TESTING**

5.1 Basics/Introduction

5.2 Unit Testing

5.2.1 Test Cases

5.2.2 Code for unit testing

5.3 Functional Testing

5.2.1 Test Cases

5.2.2 Code for functional testing

**6. RESULTS & DISCUSSION**

6.1 Visualization of results

6.1.1 Homepage

6.1.2 Result

6.1.3 Accuracy

6.1.4 K-means Clustering

**CONCLUSION**

**REFERENCES**

**Acknowledgement**

We are profoundly grateful Prof. K.B. Satpute for her expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

We would like to express deepest appreciation towards Dr. S.D. Lokhande, Principal, Sinhgad College Of Engineering, Pune, Prof. M. P. Wankhade, Head of Department of Computer Engineering, whose invaluable guidance supported us in completing this project.

At last, we must express our sincere heartfelt gratitude to all the staff members of Computer Engineering Department who helped us directly or indirectly during this course of work.

Shruti Houji

Rhishabh Hattarki

Sahil Dixit

Sanika Patil

**Abstract**

This report documents glass type classification using machine learning. Glass Identification Data Set is used from UCI. It contains 10 attributes including id. The response is glass type (discrete 7 values)

KNN algorithm involves predicting the type of glass in which the data points are separated into several classes to predict the classification. K-means method is used for clustering analysis, especially in glass data which aims to partition a set of observations into a number of clusters (**k**), resulting in the partitioning of the data into cells.

Finally, testing methods such as unit testing and functional testing have shown capable methods of testing. Unit testing consists a level of software testing where individual units of the software are tested and functional testing is done using Selenium web testing tool which uses simple scripts to run tests directly within a browser.

**List of figures**

Fig 2.2 Proposed process model for executing the project

Fig 3.1 Use Case diagrams

Fig 3.2 Sequence diagram

Fig 3.3 Activity Diagram

Fig 3.4 DFD

Fig 4.1 Sample raw dataset

Fig 6.1 Homepage

Fig 6.2 Result

Fig 6.3 Accuracy

Fig 6.4 K-Means

**List of Tables**

Table 4.1 Example of final dataset

Table 5.1 Unit testing-1

Table 5.2 Unit testing-2

Table 5.3 Unit testing-3

Table 5.4 Functional testing-1

Table 5.5 Functional testing-2

**1.INTRODUCTION**

**1.1 BACKGROUND AND BASICS**

**Glass** is a non-crystalline, amorphous solid that is most often [transparent](https://en.wikipedia.org/wiki/Transparency_and_translucency) and has widespread practical, technological, and decorative uses in, for example, [window](https://en.wikipedia.org/wiki/Window) panes, [tableware](https://en.wikipedia.org/wiki/Tableware), [optics](https://en.wikipedia.org/wiki/Optics), and [optoelectronics](https://en.wikipedia.org/wiki/Optoelectronic). The most familiar, and historically the oldest, types of manufactured glass are "silicate glasses" based on the chemical compound [silica](https://en.wikipedia.org/wiki/Silicon_dioxide) (silicon dioxide, or [quartz](https://en.wikipedia.org/wiki/Quartz)), the primary constituent of [sand](https://en.wikipedia.org/wiki/Sand). The term *glass*, in popular usage, is often used to refer only to this type of material, which is familiar from use as window glass and in glass bottles. Of the many silica-based glasses that exist, ordinary glazing and container glass is formed from a specific type called [soda-lime glass](https://en.wikipedia.org/wiki/Soda-lime_glass), composed of approximately 75% [silicon dioxide](https://en.wikipedia.org/wiki/Silicon_dioxide) (SiO2), [sodium oxide](https://en.wikipedia.org/wiki/Sodium_oxide) (Na2O) from [sodium carbonate](https://en.wikipedia.org/wiki/Sodium_carbonate) (Na2CO3), [calcium oxide](https://en.wikipedia.org/wiki/Calcium_oxide) (CaO), also called [lime](https://en.wikipedia.org/wiki/Lime_(material)), and several minor additives.

Many applications of silicate glasses derive from their optical [transparency](https://en.wikipedia.org/wiki/Transparency_and_translucency), giving rise to their primary use as window panes. Glass will [transmit](https://en.wikipedia.org/wiki/Transmission_coefficient#Optics), [reflect](https://en.wikipedia.org/wiki/Reflection_(physics)) and [refract](https://en.wikipedia.org/wiki/Refraction) light; these qualities can be enhanced by cutting and polishing to make [optical lenses](https://en.wikipedia.org/wiki/Lens_(optics)), [prisms](https://en.wikipedia.org/wiki/Prism), fine glassware, and [optical fibers](https://en.wikipedia.org/wiki/Optical_fiber) for high speed data transmission by light. Glass can be coloured by adding metallic salts, and can also be painted and printed with [vitreous enamels](https://en.wikipedia.org/wiki/Vitreous_enamel). These qualities have led to the extensive use of glass in the manufacture of [art objects](https://en.wikipedia.org/wiki/Art_objects) and in particular, [stained glass windows](https://en.wikipedia.org/wiki/Stained_glass) etc. In this way, based on its properties there are various uses of glass.

Glass manufacturers or companies and factories that use glass as their raw material can have difficulty understanding and classifying the glass materials. What sort of glass is suitable for what purpose scientifically? This project will answer that question.

Our aim to make this whole process easier for all the parties involved. Different properties of the glass such as refractive index and its various constituents are taken into account and finally the glass is classified into various categories tableware, container, window, vehicle glass etc.

**1.2 PROBLEM STATEMENT**

**1.2.1 Scope Statement**

Glass Glass Classification is one of the active research areas. It can be used for the following applications

* Packaging
* Tableware
* Housing and buildings
* Interior design and furnitures
* Appliances and Electronics
* Automotive and transport
* Medical technology
* Biotechnology
* life science engineering
* optical glass
* Radiation protection from X-Rays (radiology) and gamma-rays (nuclear)
* Fibre optic cables (phones, TV, computer: to carry information)
* Renewable energy (solar-energy glass, wind turbines) .

The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

**2.PROJECT PLANNING AND MANAGEMENT**

**2.1 HARDWARE REQUIREMENTS**

**2.1 Basic Requirements**

* Processor: Intel core i5
* Ram: 8 GB
* Hard Disc: 800 GB

**2.2 SOFTWARE REQUIREMENTS**

**2.1.1 Basic requirements**

* Operating system: Windows 64 bit
* Language: Python with various libraries
* Tools: VS Code, Jupyter
* Testing tools: Selenium, PyTest
* Front end: Django
* Back end: Python

**2.3 PROCESS MODELLING**

The Waterfall Model can be considered as a generic process model.

1**.Requirements analysis and definition** :The requirements are established by consultation with system users. After that they are defined in detail and serve as the system specification.

2**.System and Software design**: The overall system architecture is defined. The fundamental software system abstractions and their abstractions are identified.

3.**Implementation and unit testing** : The software design is realized as a set of program units; testing verifies that each unit meets its specification.

4.**Integration and system testing** : Program units are integrated and tested as a complete system

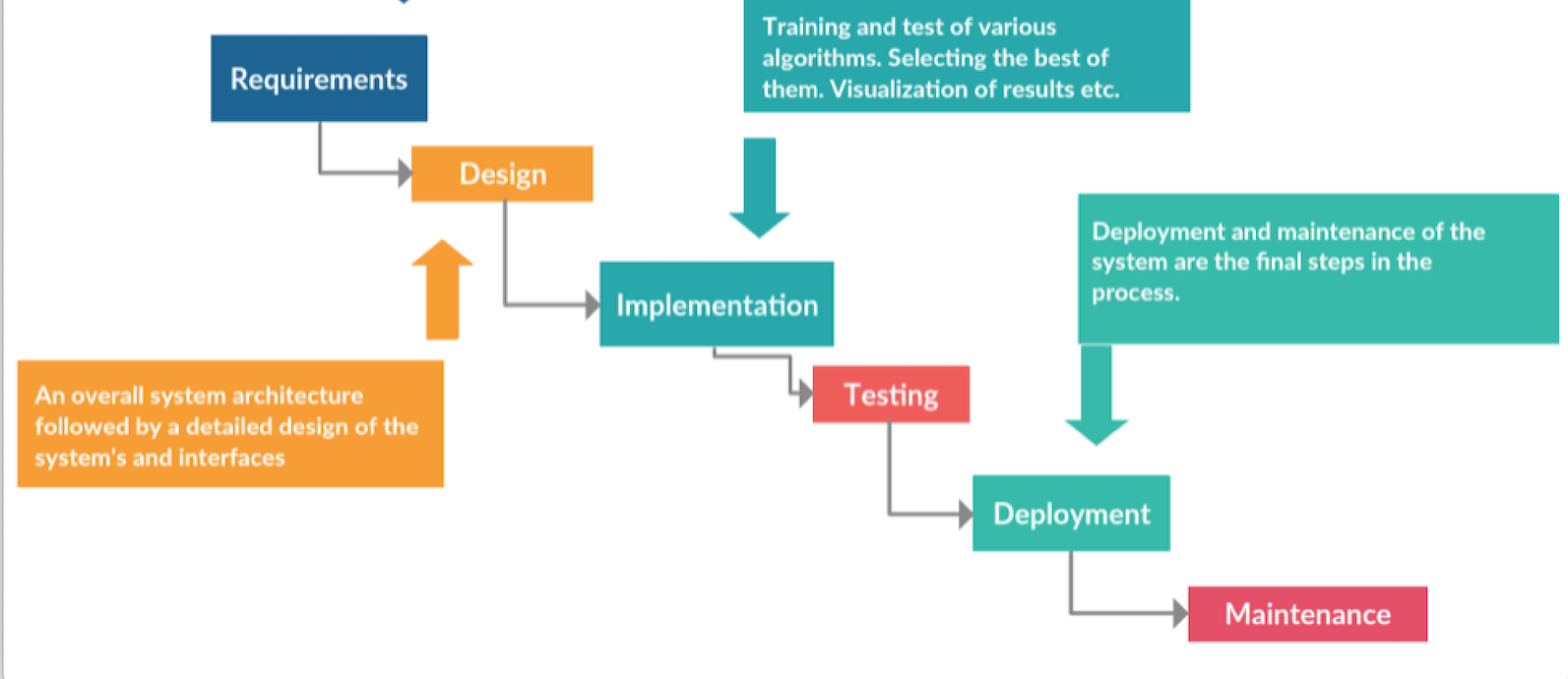
**Key Properties of the Waterfall Model**

• The result of each phase is a set of artifacts that is approved.

• The following phase starts after the previous phase has finished. (In practice there might be some overlapping.)

• In case of errors previous process stages have to be repeated.

• Fits with other (hardware) engineering process models.

****

**Fig 2.2 Proposed process model for executing the project**

**3.ANALYSIS & DESIGN**

**3.1. Use Case Diagram**

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams

model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform.



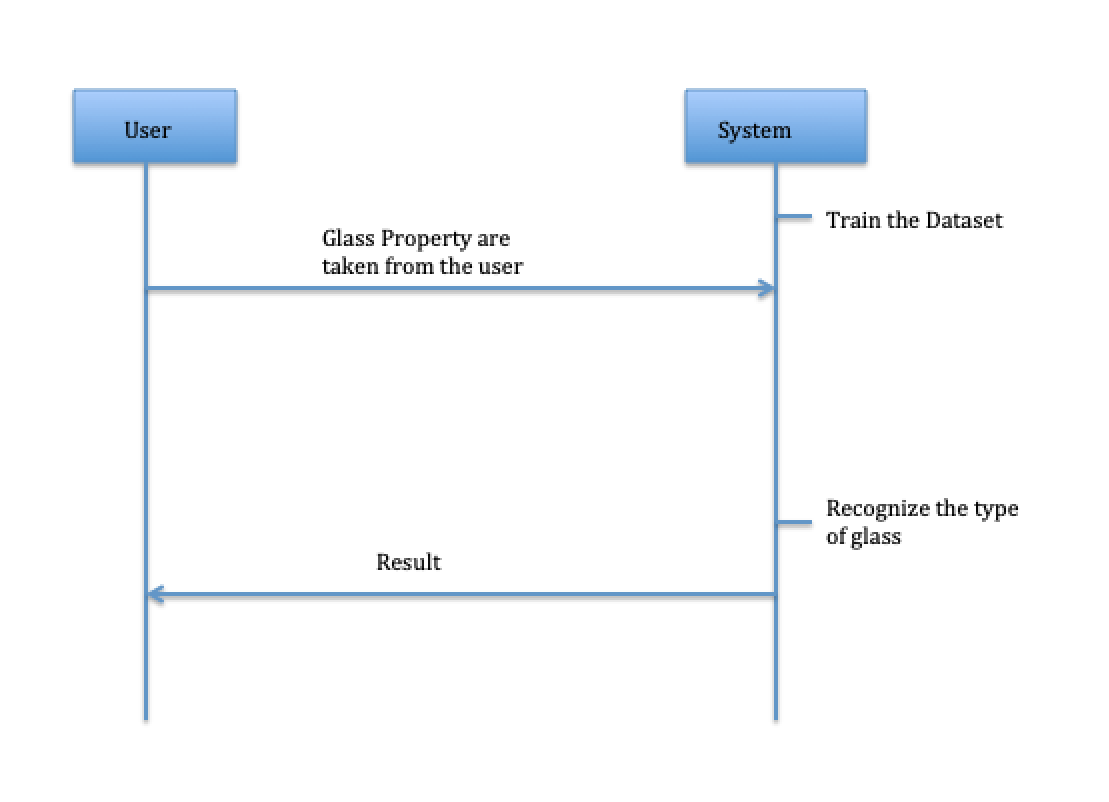
**Fig 3.1 Use Case diagram**

**3.2. Sequence Diagram**

A sequence diagram simply depicts interaction between objects in a sequential order

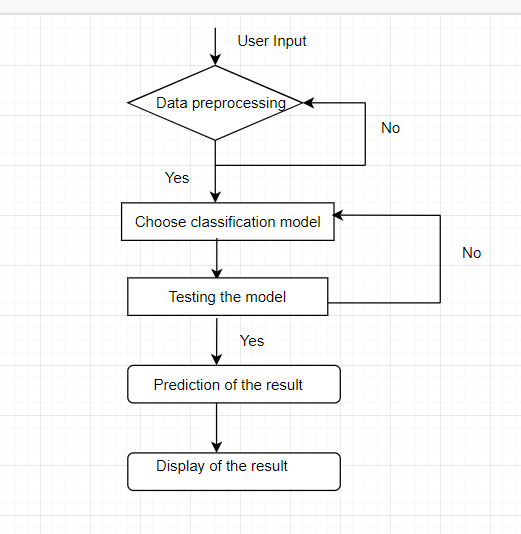
i.e. the order in which these interactions take place.Sequence diagrams describe how and in what

order the objects in a system function.



**Fig 3.2 Sequence diagram**

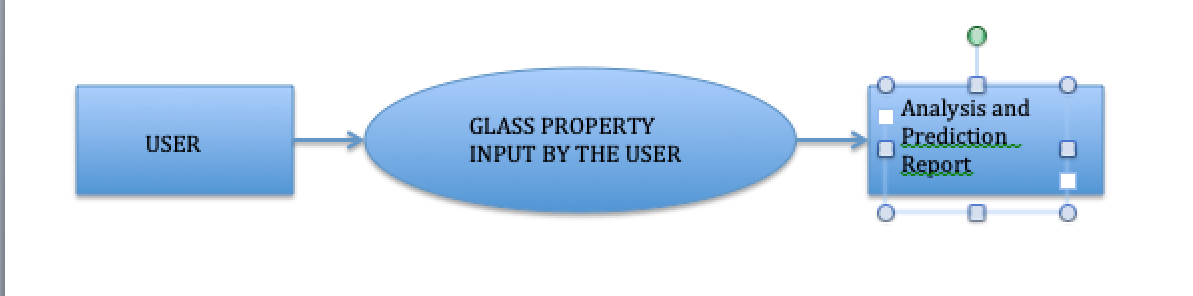
**3.3 Activity Diagram**



**Fig: 3.3 Activity Diagram**

**3.4 Data Flow Diagram**

A data flow diagram (DFD) is a graphical representation of the &quot;flow&quot; of data through an information system, modelling its process aspects. ADFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated.

****

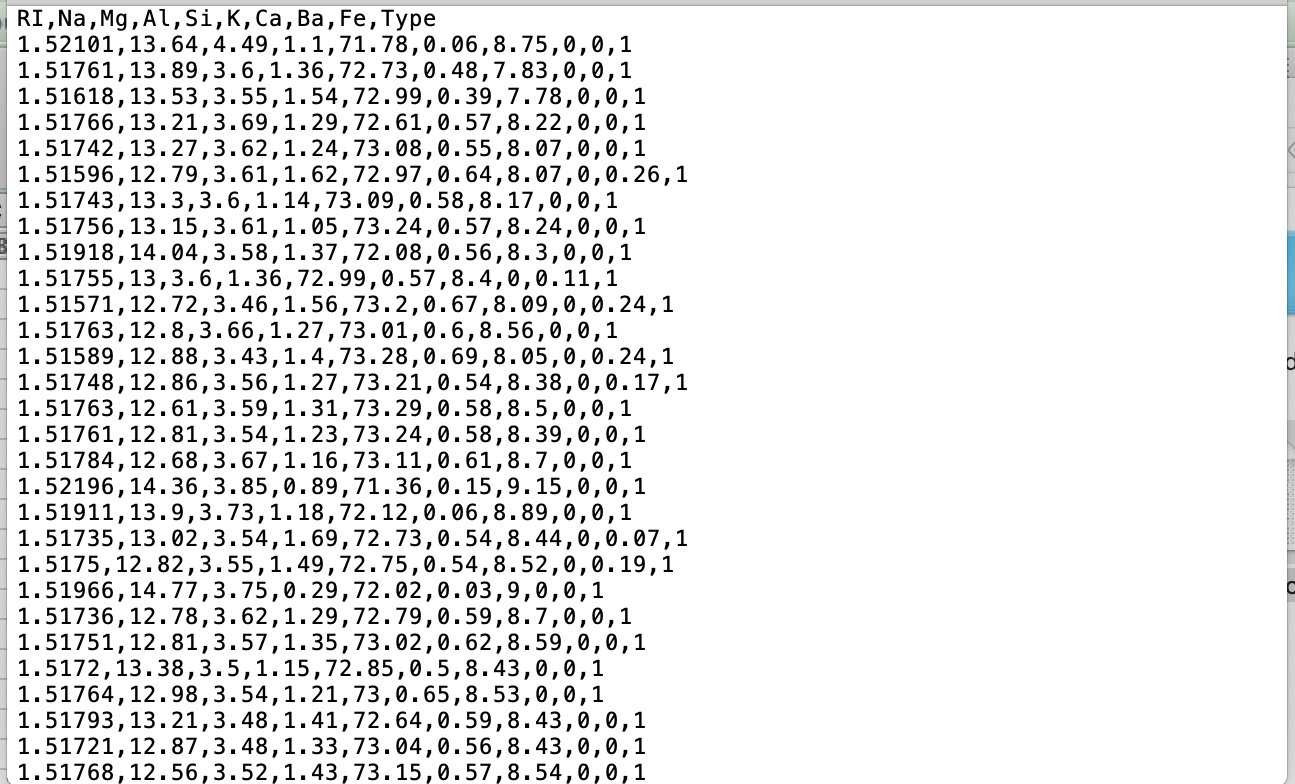
**Fig 3.4 DFD**

**4.IMPLEMENTATION AND CODING**

**4.1 DATASET**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RI** | **Na** | **Mg** | **Al** | **Si** | **K** | **Ca** | **Ba** | **Fe** | **Type** |
| 1.52101 | 13.64 | 4.49 | 1.1 | 71.78 | 0.06 | 8.75 | **0** | **0** | **1** |
| 1.51761 | 13.89 | 3.6 | 1.36 | 72.73 | 0.48 | 7.83 | **0** | **0** | **1** |
| **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** | **…** |

**Table 4.1 Example of final dataset**

****

**Fig 4.1 Sample raw dataset**

**4.2 ALGORITHMS AND CODING**

**4.2.1 Preprocessing the dataset**

df = pd.read\_csv('glass.csv')  
 X = df.iloc[:, [3, 4]].values  
 m = X.shape[0] # number of training examples  
 n = X.shape[1] # number of features.

**4.2.2 Training the algorithm**

The list of algorithms implemented is as follows:

# Model 0: KNN 83%

# Model 1: K-means Clustering 7 clusters

* + - 1. **KNN Classifier**

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

*k*-NN is a type of [instance-based learning](https://en.wikipedia.org/wiki/Instance-based_learning), or [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning), where the function is only approximated locally and all computation is deferred until classification.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor.[[2]](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#cite_note-2)

CODE

class KNN:  
    def \_\_init\_\_(self, k, d\_metric, p=1):  
        self.k = k  
        self.d\_metric = d\_metric  
        self.d\_metric\_to\_fn = {  
            'euclidean': self.euclidean,  
            'manhattan': self.manhattan,  
            'minkowski': self.minkowski  
        }  
        self.p = p  
  
    def fit(self, X, y):  
        self.X = np.copy(X)  
        self.y = np.copy(y)  
  
    def manhattan(self, x\_test):  
        return np.sum(np.abs(self.X - x\_test), axis=-1)  
  
   
  
         
  
    def distance(self, x\_test):  
        return self.d\_metric\_to\_fn[self.d\_metric](x\_test)  
  
    def predict(self, x\_test):  
       
         
        distances = self.distance(x\_test)  
        sorted\_labels = self.y[np.argsort(distances)]  
        k\_sorted\_labels = sorted\_labels[:self.k]  
        unique\_labels, counts = np.unique(k\_sorted\_labels, return\_counts=True)  
        pred1 = unique\_labels[np.argmax(counts)]  
  
        return pred1  
     
     
    def predict2(self, x\_test):  
        preds = []  
        for index in range(x\_test.shape[0]):  
            distances = self.distance(x\_test[index])  
            sorted\_labels = self.y[np.argsort(distances)]  
            k\_sorted\_labels = sorted\_labels[:self.k]  
            unique\_labels, counts = np.unique(k\_sorted\_labels, return\_counts=True)  
            pred = unique\_labels[np.argmax(counts)]  
            preds.append(pred)  
        return np.array(preds)  
     
    def accuracy(self, data, labels):  
        pred = self.predict2(data)  
        count = 0  
        for i in range(len(pred)):  
            if pred[i] == labels[i]:  
                count += 1  
        return float(count)/len(pred)

* + - 1. **K-means Clustering**

K-means is  one of  the simplest unsupervised  learning  algorithms  that  solve  the well  known clustering problem. The procedure follows a simple and  easy  way  to classify a given data set  through a certain number of  clusters (assume k clusters) fixed apriori. The  main  idea  is to define k centers, one for each cluster. These centers  should  be placed in a cunning  way  because of  different  location  causes different  result. So, the better  choice  is  to place them  as  much as possible  far away from each other.

CODE:

K = 7  
n\_iter = 100  
Centroids = np.array([]).reshape(n, 0)  
  
for i in range(K):  
 rand = rd.randint(0, m - 1)  
 Centroids = np.c\_[Centroids, X[rand]]  
  
Output = {}  
  
EuclidianDistance = np.array([]).reshape(m, 0)  
for k in range(K):  
 tempDist = np.sum((X - Centroids[:, k]) \*\* 2, axis=1)  
 EuclidianDistance = np.c\_[EuclidianDistance, tempDist]  
C = np.argmin(EuclidianDistance, axis=1) + 1  
  
for i in range(n\_iter):  
 # step 2.a  
 EuclidianDistance = np.array([]).reshape(m, 0)  
 for k in range(K):  
 tempDist = np.sum((X - Centroids[:, k]) \*\* 2, axis=1)  
 EuclidianDistance = np.c\_[EuclidianDistance, tempDist]  
 C = np.argmin(EuclidianDistance, axis=1) + 1  
 # step 2.b  
 Y = {}  
 for k in range(K):  
 Y[k + 1] = np.array([]).reshape(2, 0)  
 for i in range(m):  
 Y[C[i]] = np.c\_[Y[C[i]], X[i]]  
  
 for k in range(K):  
 Y[k + 1] = Y[k + 1].T  
  
 for k in range(K):  
 Centroids[:, k] = np.mean(Y[k + 1], axis=0)  
 Output = Y  
  
color = ['red', 'blue', 'green', 'cyan', 'magenta', 'yellow', 'black']  
labels = ['bw\_fp', 'bw\_nfp', 'v', 'v\_nfp', 'containers', 'tableware', 'headlamps']  
for k in range(K):  
 plt.scatter(Output[k + 1][:, 0], Output[k + 1][:, 1], c=color[k], label=labels[k])  
plt.scatter(Centroids[0, :], Centroids[1, :], s=100, c='grey', label='Centroids')  
plt.xlabel('RI')  
plt.ylabel('Na')  
plt.legend()  
plt.show()  
from mpl\_toolkits.mplot3d import Axes3D  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
for k in range(K):  
ax.scatter(Output[k + 1][:, 0], Output[k + 1][:, 1], c=color[k], label=labels[k])  
ax.scatter(Centroids[0, :], Centroids[1, :], s=100, c='grey', label='Centroids')  
  
ax.set\_xlabel('RI')  
ax.set\_ylabel('Na')  
ax.set\_zlabel('Mg')  
plt.legend()  
plt.show()

**5. TESTING**

**5.1 BASICS/INTRODUCTION**

#### Software Testing

Software testing is the process of evaluation a software item to detect differences between given input and expected output. Also to assess the feature of A software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words software testing is a verification and validation process.

#### Types of testing

There are many types of testing like

* Unit Testing
* Integration Testing
* Functional Testing
* System Testing
* Stress Testing
* Performance Testing
* Usability Testing
* Acceptance Testing
* Regression Testing
* Beta Testing

In our project we will be doing unit testing using PyTest and functional testing using Selenium

**5.2 UNIT TESTING**

Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing. It is often done by the programmer to test that the unit he/she has implemented is producing expected output against given input.

We are using PyTest for conduction unit testing.

**5.2.1. Test cases for unit testing**

Table 5.1: Unit Test - 1

|  |  |
| --- | --- |
|  |  |
| Title | Glass classification check |
|  |  |
| Test Item | predict() |
|  |  |
| Input Specification | Particular row containing numeric values(Numpy array)  [[1.51514,14.85,0,2.42,73.72,0,8.39,0.56,0]] |
|  |  |
| Description | Numeric values for all columns for a particular row is entered to get the classification of glass |
|  |
|  |
|  |  |
| Expected Results | Category(type) of glass  7 |
|  |  |
| Result | Pass |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  | Table 5.1: Unit Test - 2 |
| Title | Accuracy function check |
|  |  |
| Test Item | accur() |
|  |  |
| Input Specification | Particular row containing numeric values (Euclidean function) |
|  |  |
| Description | Numeric values for all columns for a particular row is entered to get the accuracy for particular row |
|  |
|  |
|  |  |
| Expected Results | Accuracy expected for particular row  0.83644 |
|  |  |
| Result | Pass |
|  |  |

|  |  |
| --- | --- |
|  | Table 5.1: Unit Test - 3 |
|  |  |
| Title | Test result function check |
|  |  |
| Test Item | Result() |
|  |  |
| Input Specification | Particular row containing numeric values |
| (Euclidean function) |
|  |
|  |  |
| Description | Numeric values for all columns for a particular row is entered to get the category of glass |
|  |  |
|  |  |
|  |  |
| Expected Results | Category (type) of glass  2 |
|  |  |
| Result | Pass |
|  | c |

**5.2.2. Code for unit testing**

|  |
| --- |
| import unittest |
|  |

|  |
| --- |
| from cal import KNN |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Testcal(unittest.TestCase): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def test\_predict(self): |
|  |

|  |
| --- |
| cal = KNN(3, 'euclidean') |
|  |

|  |
| --- |
| result=cal.predict([[1.51514,14.85,0,2.42,73.72,0,8.39,0.56,0]]) |
|  |

|  |
| --- |
| self.assertEqual(result,7) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def test\_accur(self): |
|  |

|  |
| --- |
| cal = KNN(3, 'euclidean') |
|  |

|  |
| --- |
| result=cal.accur('euclidean') |
|  |

|  |
| --- |
| self.assertEqual(result,0.83644) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def test\_result\_knn(self): |
|  |

|  |
| --- |
| cal = KNN(3, 'euclidean') |
|  |

|  |
| --- |
| result=cal.result\_knn('euclidean') |
|  |

|  |
| --- |
| self.assertEqual(result,2) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if \_\_name\_\_== '\_\_main\_\_': |
|  |

unittest.main()

------------------------------------------------------------------------------------

cal.py

|  |
| --- |
| import pandas as pd |
|  |

|  |
| --- |
| import numpy as np |
|  |

|  |
| --- |
| import os |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for dirname, \_, filenames in os.walk('glass.csv'): |
|  |

|  |
| --- |
| for filename in filenames: |
|  |

|  |
| --- |
| print(os.path.join(dirname, filename)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # modulePath = os.path.dirname(\_\_file\_\_) |
|  |

|  |
| --- |
| # data = pd.read\_csv(os.path.join(modulePath, 'glass.csv')) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class KNN: |
|  |

|  |
| --- |
| def \_\_init\_\_(self, k, d\_metric, p=1): |
|  |

|  |
| --- |
| self.k = k |
|  |

|  |
| --- |
| self.d\_metric = d\_metric |
|  |

|  |
| --- |
| self.d\_metric\_to\_fn = { |
|  |

|  |
| --- |
| 'euclidean': self.euclidean, |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| self.p = p |
|  |

|  |
| --- |
| self.data = pd.read\_csv("glass.csv") |
|  |

|  |
| --- |
| self.labels = self.data.pop("Type").values |
|  |

|  |
| --- |
| self.data = self.data.values |
|  |

|  |
| --- |
| self.labels = self.labels.reshape(-1, 1) |
|  |

|  |
| --- |
| self.fit(self.data, self.labels) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def fit(self, X, y): |
|  |

|  |
| --- |
| self.X = np.copy(X) |
|  |

|  |
| --- |
| self.y = np.copy(y) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def euclidean(self, x\_test): |
|  |

|  |
| --- |
| sq\_diff = (self.X - x\_test) \*\* 2 |
|  |

|  |
| --- |
| return np.sqrt(np.sum(sq\_diff, axis=-1)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def distance(self, x\_test): |
|  |

|  |
| --- |
| return self.d\_metric\_to\_fn[self.d\_metric](x\_test) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def predict(self, x\_test): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| distances = self.distance(x\_test) |
|  |

|  |
| --- |
| sorted\_labels = self.y[np.argsort(distances)] |
|  |

|  |
| --- |
| k\_sorted\_labels = sorted\_labels[:self.k] |
|  |

|  |
| --- |
| unique\_labels, counts = np.unique(k\_sorted\_labels, return\_counts=True) |
|  |

|  |
| --- |
| pred1 = unique\_labels[np.argmax(counts)] |
|  |

|  |
| --- |
| return pred1 |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def predict2(self, x\_test): |
|  |

|  |
| --- |
| preds = [] |
|  |

|  |
| --- |
| for index in range(x\_test.shape[0]): |
|  |

|  |
| --- |
| distances = self.distance(x\_test[index]) |
|  |

|  |
| --- |
| sorted\_labels = self.y[np.argsort(distances)] |
|  |

k\_sorted\_labels = sorted\_labels[:self.k]

|  |
| --- |
| unique\_labels, counts = np.unique(k\_sorted\_labels, return\_counts=True) |
|  |

|  |
| --- |
| pred = unique\_labels[np.argmax(counts)] |
|  |

|  |
| --- |
| preds.append(pred) |
|  |

|  |
| --- |
| return np.array(preds) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def accuracy(self, data, labels): |
|  |

|  |
| --- |
| pred = self.predict2(data) |
|  |

|  |
| --- |
| count = 0 |
|  |

|  |
| --- |
| for i in range(len(pred)): |
|  |

|  |
| --- |
| if pred[i] == labels[i]: |
|  |

|  |
| --- |
| count += 1 |
|  |

|  |
| --- |
| return float(count)/len(pred) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def result\_knn(self, metric): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # knn = KNN(k=3, d\_metric=metric) |
|  |

|  |
| --- |
| # knn.fit(data, labels) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # prediction = knn.predict(data[100,:]) |
|  |

|  |
| --- |
| prediction = self.predict(self.data[100,:]) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return prediction |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def accur(self, metric): |
|  |

|  |
| --- |
| accuraci = {} |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for k in range(1, 21): |
|  |

|  |
| --- |
| knn = KNN(k=k, d\_metric=metric) |
|  |

|  |
| --- |
| knn.fit(self.data, self.labels) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| accuraci[k] = knn.accuracy(self.data, self.labels) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return accuraci[3] |
|  |

**5.3 FUNCTIONAL TESTING**

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

We are using Selenium for conduction functional testing.

**5.3.1. Test cases for functional testing**

|  |  |
| --- | --- |
|  | Table 5.4:Functional test - 1 |
| Title | Graph page linking |
|  |  |
| Test Item | Page linking |
|  |  |
| Test Steps | 1. Open website  2. Locate graph button, click  3. Locate graph button, click  4. Locate glass heading button, click |
|  |  |
| Description | Navigate to graphs page, refresh, navigate back to home, page |
|  |
|  |
|  |  |
| Expected Results | User should navigate to and fro the home page and graphs page |
|  |  |
| Actual Result | As Expected |
| Pass/ Fail | Pass |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  | Table 5.5:Functional test - 2 |
| Title | Random value form check |
|  |  |
| Test Item | Home page form |
|  |  |
| Test Steps | 1. Locate input box 2. Add random float value 3. Repeat for all boxes 4. Submit form |
|  |  |
| Description | Fill home page form, submit it, get classification report |
|  |
|  |
|  |  |
| Expected Results | Classification report shows after submitting form |
|  |  |
| Actual Result | As Expected |
| Pass/ Fail | Pass |
|  |  |
|  |  |

**5.3.2. Code for functional testing**

# from django.test import TestCase

# Create your tests here.

from selenium import webdriver

from django.contrib.staticfiles.testing import StaticLiveServerTestCase

from django.urls import reverse

import time

import random

from selenium.webdriver.common.keys import Keys

class SelTest(StaticLiveServerTestCase):

def setUp(self):

self.browser = webdriver.Firefox()

super(SelTest, self).setUp()

def tearDown(self):

return self.browser.quit()

super(SelTest, self).tearDown()

# def test\_foo(self):

# self.assertEquals(0, 1)

def graph\_page\_linking(self):

self.browser.get('http://localhost:8000') # open website

time.sleep(0.5)

self.browser.find\_element\_by\_id('graph\_button').click() # click on graph button

time.sleep(0.5)

self.browser.find\_element\_by\_id('graph\_button').click() # click on graph button

time.sleep(0.5)

self.browser.find\_element\_by\_id('glass\_heading').click() # click on home button

time.sleep(0.5)

def random\_value\_form\_check(self):

# self.browser.get('http://localhost:8000') # open website

# time.sleep(0.5)

elements = ['ri', 'na', 'mg', 'al', 'si', 'k', 'ca', 'ba', 'fe']

for i in elements:

ri = self.browser.find\_element\_by\_id(i) # send key to ri

ri.click() # focus on input box

ri.send\_keys(str(random.uniform(0, 100))) # send random float value in range 0, 100 both inclusive, depending on rounding

time.sleep(0.5)

self.browser.find\_element\_by\_id('submit\_button').click() # click submit button

time.sleep(2)

def test\_all\_together(self):

self.graph\_page\_linking()

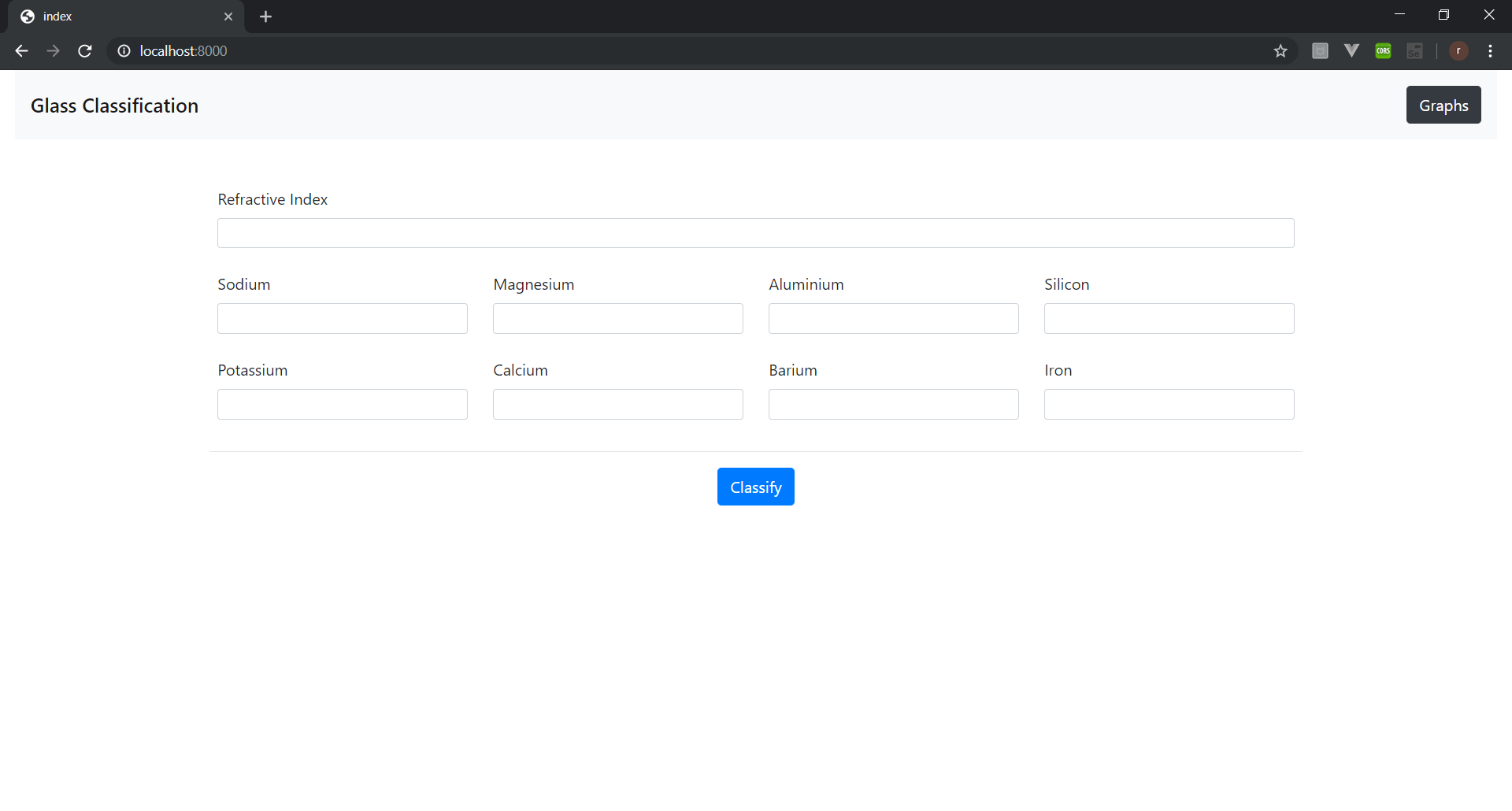
self.random\_value\_form\_check()

**6.RESULTS AND DISCUSSION**

**6.1 VISUALIZATION OF RESULTS**

**6.1.1 Homepage**

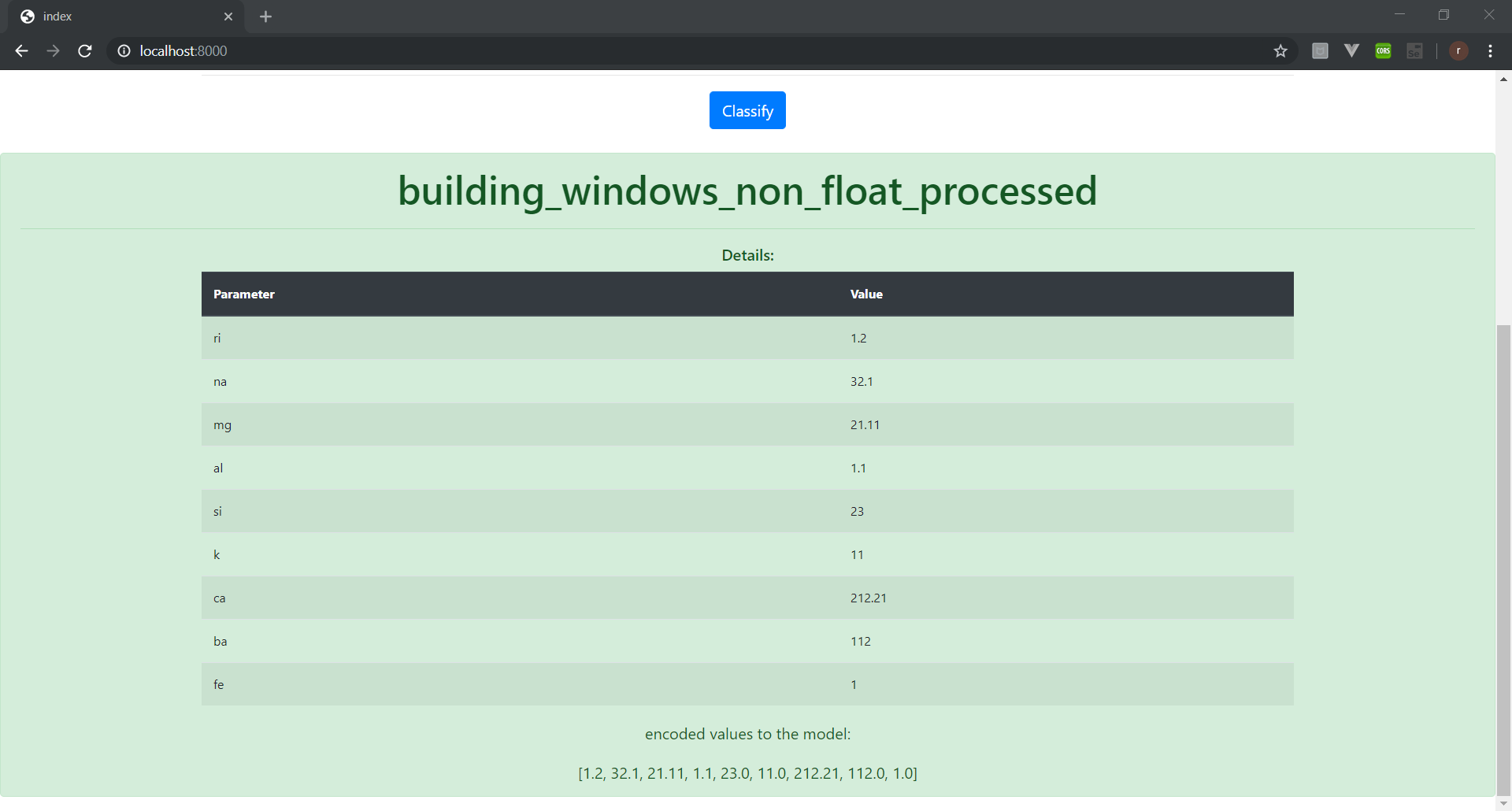
This is the first page of the WebApp created for the project. This page takes the attributes as input.



**Fig 6.1 Homepage**

**6.1.2. Result**

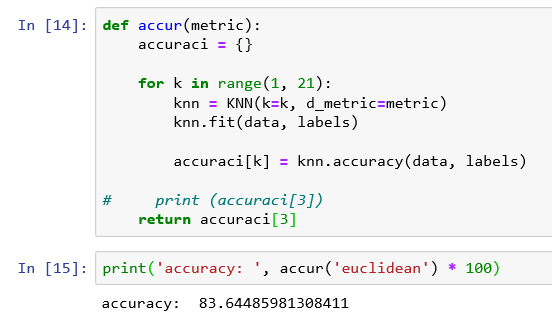
This is the output of our project. Taking into account the input values from the previous page, the data is processed using KNN and output is displayed. There are total 7 categories of glass. The output can be any one of the 7.



**Fig 6.2 Result Page**

**6.1.3 Accuracy of KNN classification**

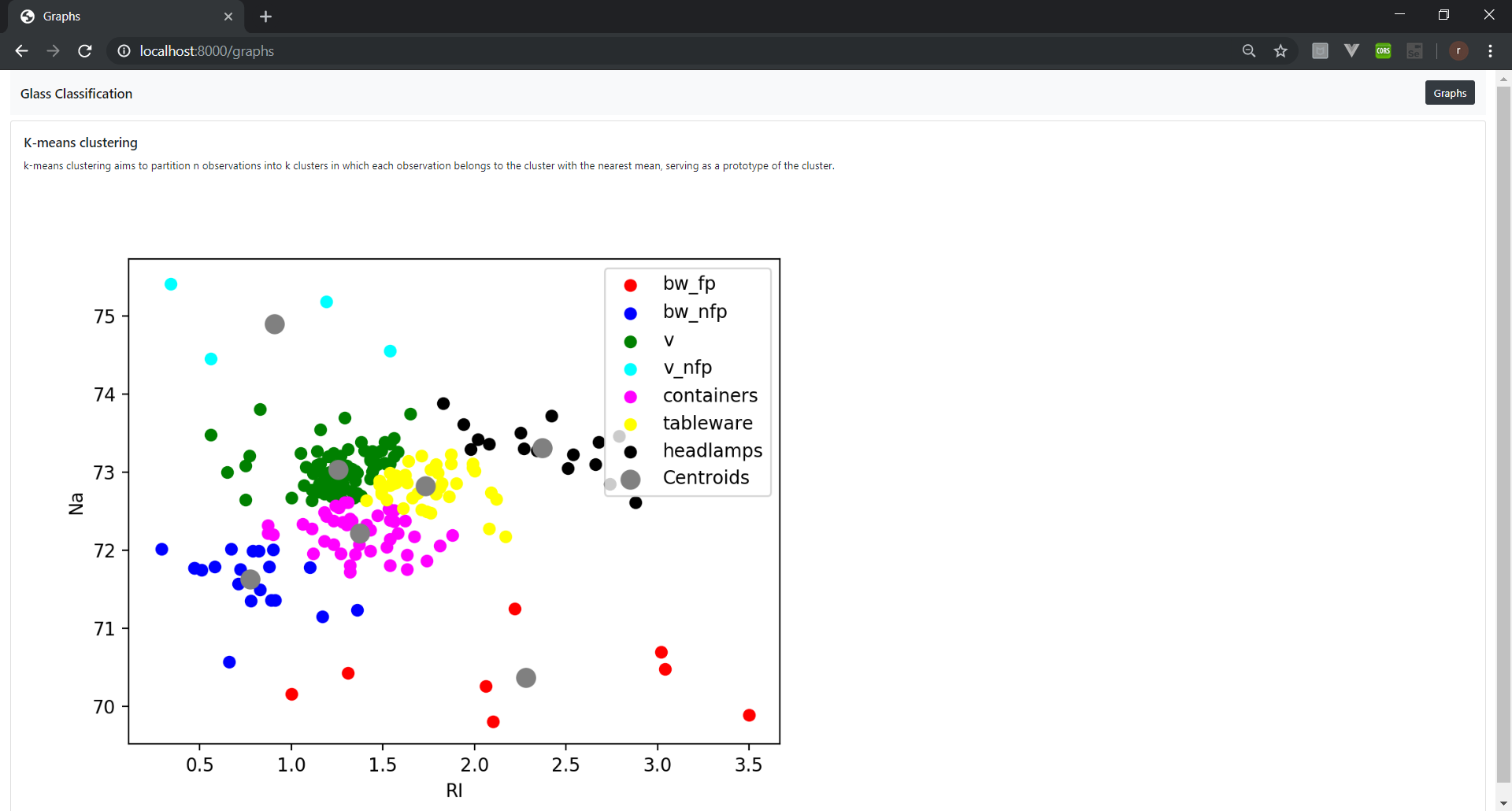
KNN classification gives 83% accuracy when implemented in this project.



**Fig 6.3 Accuracy**

**6.1.4. K-means clustering**

K-means clustering is also performed. The formed clusters as as follows.



**Fig 6.4 K-Means**

**CONCLUSION**

* We have created a system that can take input from the user about various attributes which are basically properties of the glass. These readings are then successfully classified according to the various categories of the glass.
* Testing is performed which makes sure that the expected and actual results are a match for every single module. Different types such as unit testing, functional testing etc. are performed to make the project defect-free.
* As a concluding point, given the values of the attributes, the algorithm has been trained to correctly predict the type of glass..

**REFERENCES**

**-World wide web**

1) <https://archive.ics.uci.edu/ml/datasets/Glass+Identification>

2) <https://www.kaggle.com/yundoou/knn-precision-rate-grows-from-72-to-92/data>

3) <https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm>

4) <https://www.glassallianceeurope.eu/en/applications>

# 5) https://www.schaeferglas.com/news/classification-glass/