# CLASSIFICATION OF ANIMALS INTO CATEGORIES BY APPLYING DIFFERENT ML CLASSIFICATION ALGORITHMS

Group Name: Ctrl Alt Elite (Group no. 14)

*Nirnay Korde - 20ucs133*

*Pranav Chatur - 20ucs138*

*Raghav Khanna - 20ucs153*

*Sourav Jain - 20ucs198*

INTRODUCTION TO DATA SCIENCE

Course Instructor

Dr. Sakthi Balan Muthiah

Department of Computer Science and Engineering The LNM Institute of Information Technology, Jaipur

## INDEX

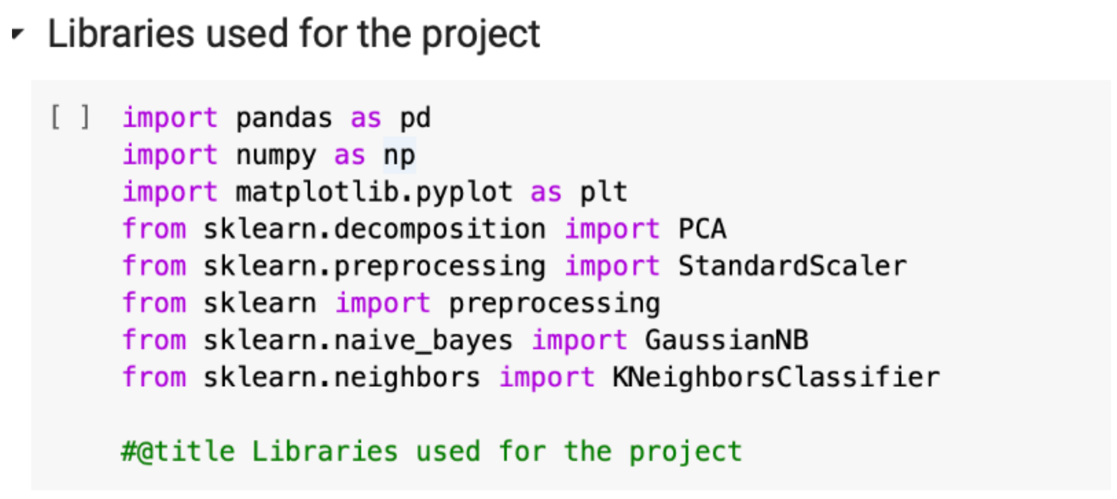
## 1. Objective

The objective of this project was to collect a dataset from UCI Machine Learning Repository: Data Sets, gather inferences by pre-processing it and perform preliminary analysis on it. Finally, classify the data using appropriate Machine Learning Algorithms and measure the performances of each of the algorithms used.

## 2. Introduction to the Dataset

This dataset is from the UCI Machine Learning Repository , It is a simple database containing 17 boolean-valued attributes and 1 output attribute . The classification will tell us about the type in which an animal will lie based on the set of attributes.
Some of the attributes are : Animal Name, Hair, eggs, milk, airborne, aquatic, predator etc.

## 3. Imported Libraries



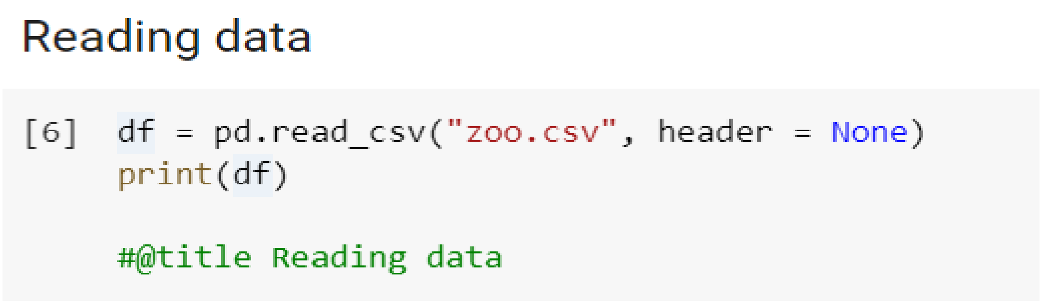
Libraries used for the project

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.decomposition import PCA
    from sklearn.preprocessing import StandardScaler
    from sklearn import preprocessing
    from sklearn.naive_bayes import GaussianNB
    from sklearn.neighbors import KNeighborsClassifier

    #@title Libraries used for the project
```

# 4. Understanding the dataset

Here the 0th attribute represents the animal name , 1st attribute represents hair (0 if the animal has no hair , 1 otherwise ), 2nd attribute represents feathers and so on .



Reading data

```
[6]  df = pd.read_csv("zoo.csv", header = None)
     print(df)

     #@title Reading data
```

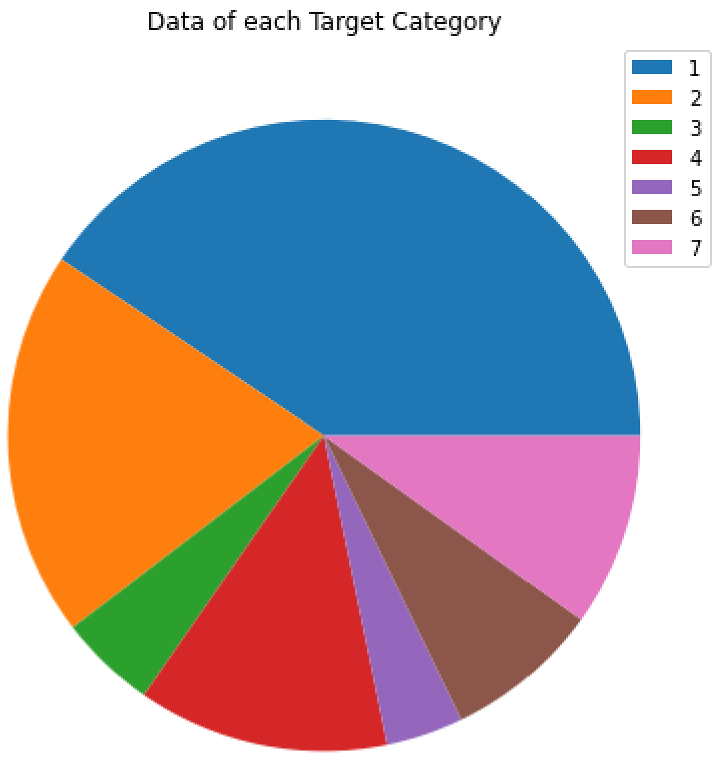
```
        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
0   aardvark  1  0  0  1  0  0  1  1  1  1  0  0  4  0  0  1
1   antelope  1  0  0  1  0  0  0  1  1  1  0  0  4  1  0  1
2       bass  0  0  1  0  0  1  1  1  1  0  0  1  0  1  0  0
3       bear  1  0  0  1  0  0  1  1  1  1  0  0  4  0  0  1
4       boar  1  0  0  1  0  0  1  1  1  1  0  0  4  1  0  1
..       ... .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
96   wallaby  1  0  0  1  0  0  0  1  1  1  0  0  2  1  0  1
97      wasp  1  0  1  0  1  0  0  0  0  1  1  0  6  0  0  0
98      wolf  1  0  0  1  0  0  1  1  1  1  0  0  4  1  0  1
99      worm  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0
100     wren  0  1  1  0  1  0  0  0  1  1  0  0  2  1  0  0
```

**Value of Target Attribute for each Instance**

| Instance No. | Target Attribute |
|---|---|
| 0 | 1 |
| 1 | 1 |
| 2 | 4 |
| 3 | 1 |
| 4 | 1 |
| .. | .. |
| 96 | 1 |
| 97 | 6 |
| 98 | 1 |
| 99 | 7 |
| 100 | 2 |

## 5. Pie chart for given target categories

We have a total of 7 categories in which we have classified the existing instances (animals) present in our dataset. The pie chart gives us a measure of how many instances are present in each category.



## 6. Pre-processing dataset

For preliminary analysis of the dataset, our goal is to perform Principal Component Analysis (PCA). Now, PCA is affected by scale so we need to scale the features in our data before applying PCA, i.e. we need to pre-process the data so as to suit our needs.

Hence we are using StandardScaler to standardize the dataset's features onto unit scale (mean = 0 and variance = 1) which is also a requirement for the optimal performance of many machine learning algorithms .

Standardising and Separating the Target

```
[ ]
    # Seperate out the properties
    properties = np.array(df.columns[1:len(df.columns)-1])

    # Seperate out the target
    target = df.columns[len(df.columns)-1]

    # Extract property values
    prop_vals = df.loc[:,properties].values

    # Extract target values
    tar_vals = df.loc[:,[target]].values

    # Standardise property values (Mean: 0 and Variance: 1)
    stand_prop_vals = StandardScaler().fit_transform(prop_vals)

    #@title Standardising and Separating the Target
```

---

# 7. Preliminary Analysis using PCA

As our original dataset contains 16 total attributes, we need to reduce the no. of dimensions to enhance the efficiency of machine learning models by reducing the computations. Therefore the minimum number of principal components are choosen in such a way the such that 90% of the variance is still retained.

```
[ ] # pca with 90% variance retention
    pca = PCA(n_components=0.90, svd_solver='full')

    # join table
    principalComponents = pca.fit_transform(stand_prop_vals)

    print(len(principalComponents[0]))
    print(principalComponents)

    #@title Preliminary Analysis using PCA
```

Inferences from PCA (from code):

A. Number of principal components needed for 90% variance retention

   **9**

B. Updated dataset
    a. Earlier dataset (17 attributes) is now converted into a final dataset (9 attributes) on which models will be trained and tested .

```
     principal component 1  principal component 2  principal component 3  \


0                -2.535256              -0.260278               1.241842
1                -2.877354              -0.401547              -0.100018
2                 1.761003               3.705592              -0.180105
3                -2.535256              -0.260278               1.241842
4                -2.818975               0.107081               0.136852
..                     ...                    ...                    ...
96               -2.764120              -0.065219              -0.409317
97                1.667523              -3.219581               2.723445
98               -2.818975               0.107081               0.136852
99                2.018815              -1.257793               1.409204
100               2.049971              -1.907380              -2.497322


     principal component 4  principal component 5  principal component 6  \
0                -1.065734              -0.296433               0.261767
1                -0.014988              -0.308969              -0.627116
2                 0.578327              -0.327066              -0.379035
3                -1.065734              -0.296433               0.261767
4                -1.178338               0.147460               0.069023
..                     ...                    ...                    ...
96                0.151818              -0.235706              -0.786741
97                0.743239               2.290063              -0.896837
98               -1.178338               0.147460               0.069023
99                0.715264              -1.014103              -0.963336
100               0.044589              -0.031691              -0.777324


     principal component 7  principal component 8  principal component 9
0                -0.041032              -0.301902               1.287573
1                 0.796777               0.345511              -0.552024
2                -0.560420              -0.200556              -0.158270
3                -0.041032              -0.301902               1.287573
4                -0.144118               0.065897              -0.119328
..                     ...                    ...                    ...
96                0.910330               0.476138               0.099124
97                1.661100              -1.217305              -0.182554
98               -0.144118               0.065897              -0.119328
99                0.479793               1.411127               1.790623
100               0.176177               0.008120              -0.030686
```

## 8. Contribution of each principle component

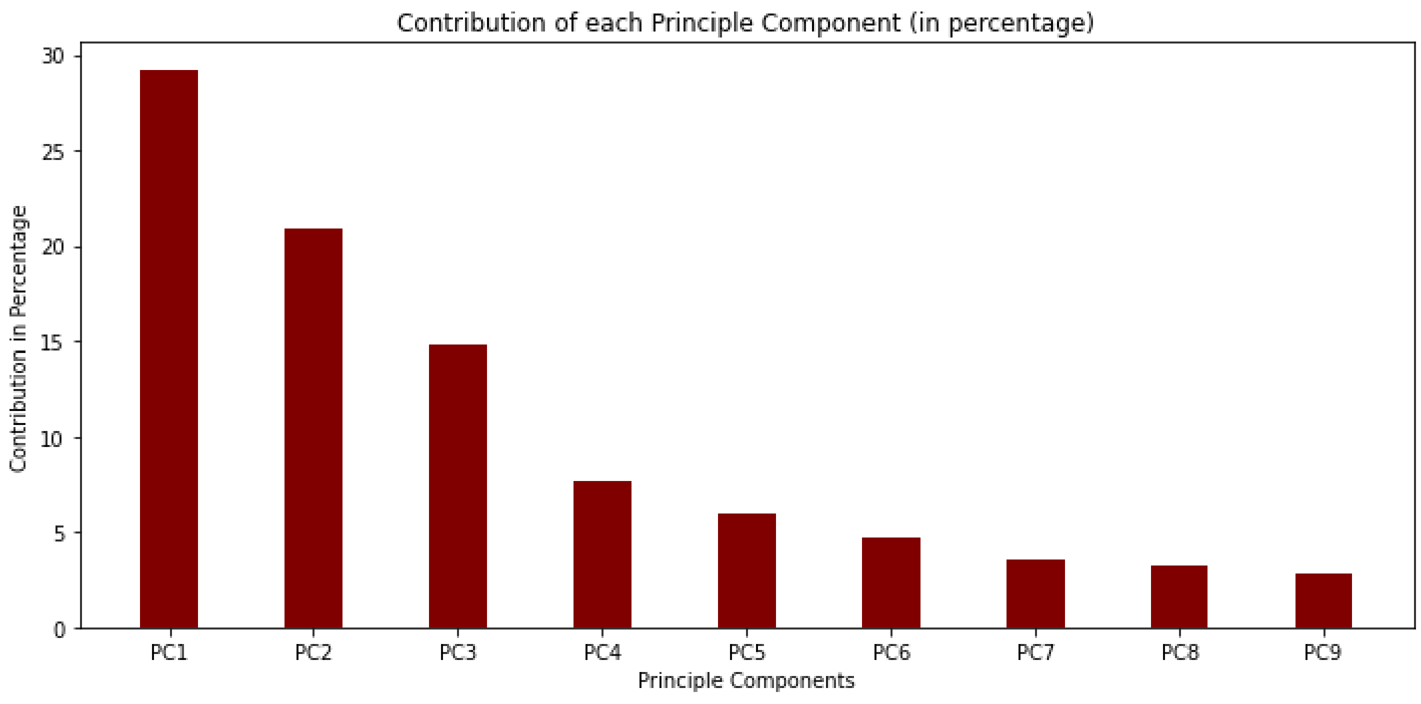
```
[ ]
    # how much each principal component is contributing
    print(pca.explained_variance_ratio_)

    # total sum of contributions, greater than or equal to min variance)
    print(np.sum(np.array(pca.explained_variance_ratio_)))

    # @title Inferences from PCA
```

```
[0.29191091 0.20882904 0.14778223 0.07693359 0.05962114 0.04662709 0.03517581
0.03202669 0.02798703]

0.926893528467885
```



# 9. Splitting dataset into Training and Test set

## A. Initialising variables

```
[ ]
     PC1 = []
     PC2 = []
     PC3 = []
     PC4 = []
     PC5 = []
     PC6 = []
     PC7 = []
     PC8 = []
     PC9 = []
     training_tar_vals = []
     test_df = []
     test_tar_vals = []
```

## B. Randomly deciding test set from dataset

```
[ ] test_set = [2, 4, 13, 32, 33, 55, 56, 65, 89, 92, 96, 97, 98]

    #@title Deciding the test set
```

## C. Initialing Training Set

```
# Creation of training set
i=0
for point in principalComponents:
  if i not in test_set :
    PC1.append(point[0])
    PC2.append(point[1])
    PC3.append(point[2])
    PC4.append(point[3])
    PC5.append(point[4])
    PC6.append(point[5])
    PC7.append(point[6])
    PC8.append(point[7])
    PC9.append(point[8])
  i += 1

i = 0
for val in tar_vals:
  if i not in test_set :
    training_tar_vals.append(val)
  i += 1
```

D. Initialising Test set

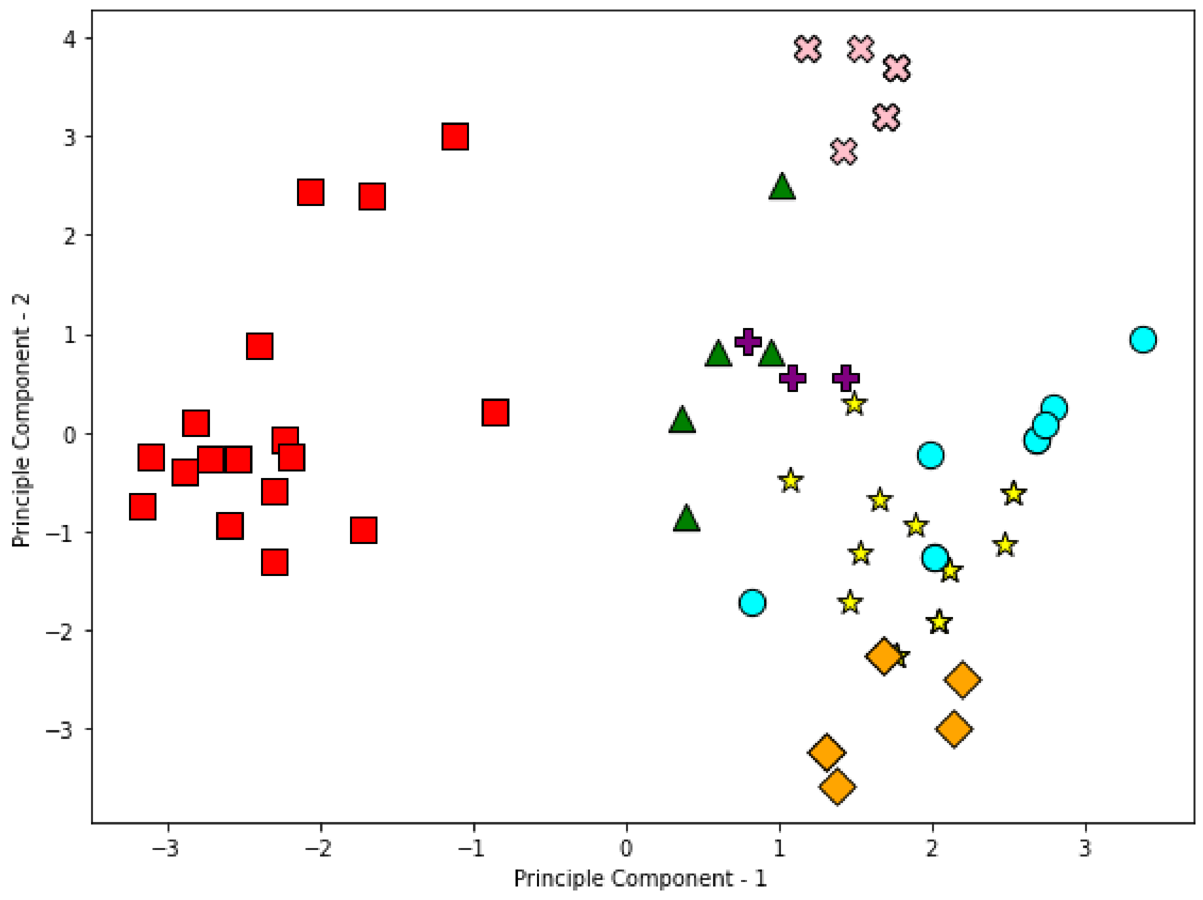
```
# Creation of test set
for i in test_set:
  test_df.append(principalDf.iloc[i])
  test_tar_vals.append(tar_vals[i])

combineC = list(zip(PC1, PC2, PC3, PC4, PC5, PC6, PC7, PC8, PC9))
```

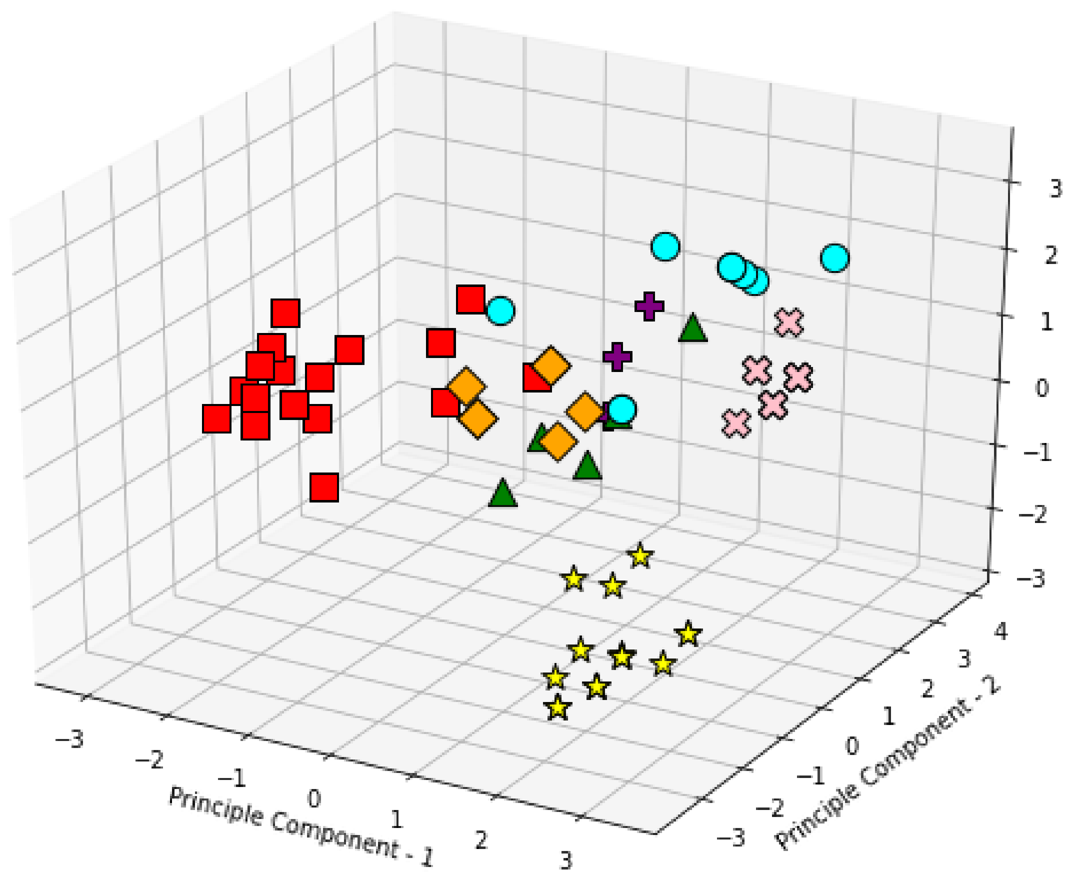
---

## 10. Visualising training set using principle components

A. 2D visualisation using PC-1 & PC-2 (most contributing)



## B. 3D visualisation using PC-1, PC-2 and PC-3 (3 most contributing)



As evident from the above 2 graphs, when more dimensions are plotted, the target categories are segregated more distinctly from one another. This helps us understand the importance of each principle component in classifying the dataset.

# 11. Classification Models

## A. Function for calculating accuracy

```
[ ]  def check_accuracy(predicted, actual):
       wrongs = 0
       if np.array_equal(predicted, actual) == False:
         for i in range(0, len(predicted)):
           if predicted[i] != actual[i]:
             wrongs += 1
       return 1-(wrongs/len(predicted))

     #@title Function for calculating accuracy of classification model
```

This code will help us determine the accuracy of each classification model on the basis of which we will decide which classification model is the best for our given test data.

## B. Naive Bayes Classifier

➢ The Naive Bayes classifier separates data into different classes according to the Bayes' Theorem, along with the assumption that all the predictors are independent of one another. It assumes that a particular feature in a class is not related to the presence of other features.

➢ This algorithm works very fast and can easily predict the class of a test dataset.

```
[ ]  bayes_predicted = []

     model = GaussianNB()
     #training Naive Bayes over avaliable data
     model.fit(combineC, np.ravel(training_tar_vals,order='C'))

     #testing Naive Bayes Model over test data
     for vals in test_df:
       bayes_predicted.append(model.predict([vals]))

     print(check_accuracy(np.array(bayes_predicted), np.array(test_tar_vals)))
```

➢ Accuracy of Bayes Classifier - 0.9230769230769231

➢ Predicted Categories of the instances present in our test data:

```
[[4] [1] [1] [6] [1] [1] [6] [7] [1] [1] [7] [2] [4]]
```

➢ Actual Categories of the instances present in our test data:

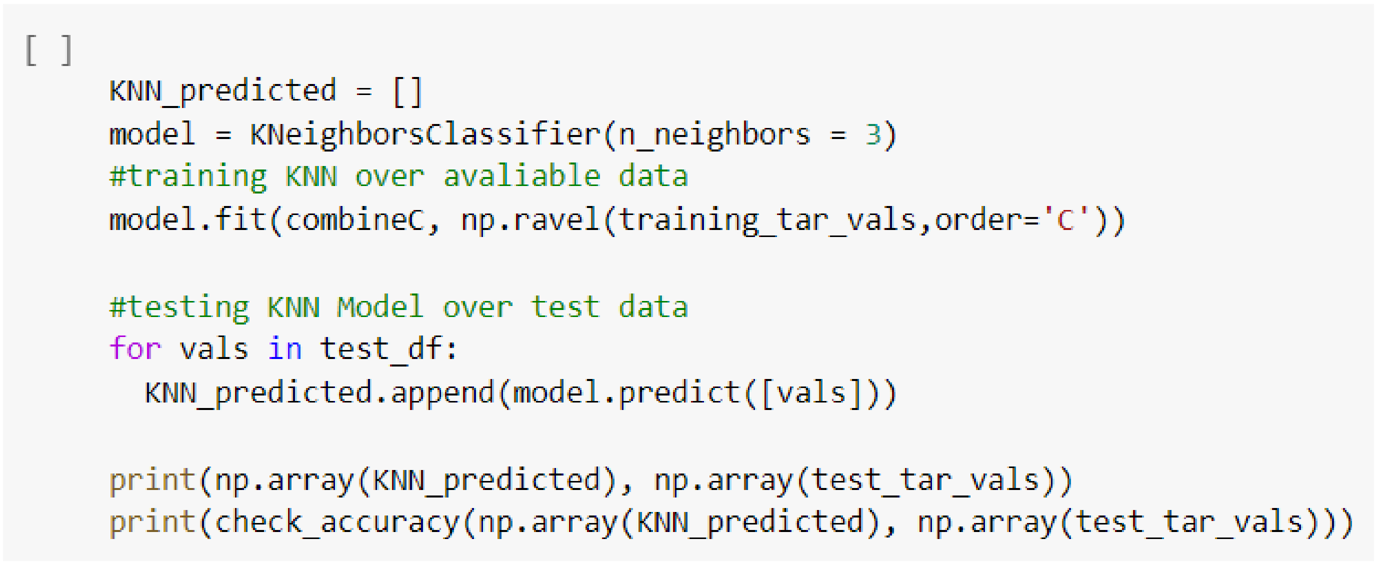
```
[[4] [1] [1] [6] [1] [1] [5] [7] [1] [1] [2] [2] [4]]
```

## C. KNN Classifier

➢ The KNN algorithm can compete with the most accurate models because it makes highly accurate predictions.

➢ The KNN algorithm uses a majority voting mechanism. It collects data from a training data set, and uses this data later to make predictions for new records.

➢ KNN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories.



```
KNN_predicted = []
model = KNeighborsClassifier(n_neighbors = 3)
#training KNN over avaliable data
model.fit(combineC, np.ravel(training_tar_vals,order='C'))

#testing KNN Model over test data
for vals in test_df:
  KNN_predicted.append(model.predict([vals]))

print(np.array(KNN_predicted), np.array(test_tar_vals))
print(check_accuracy(np.array(KNN_predicted), np.array(test_tar_vals)))
```

➢ Accuracy of KNN Classifier - 1.0

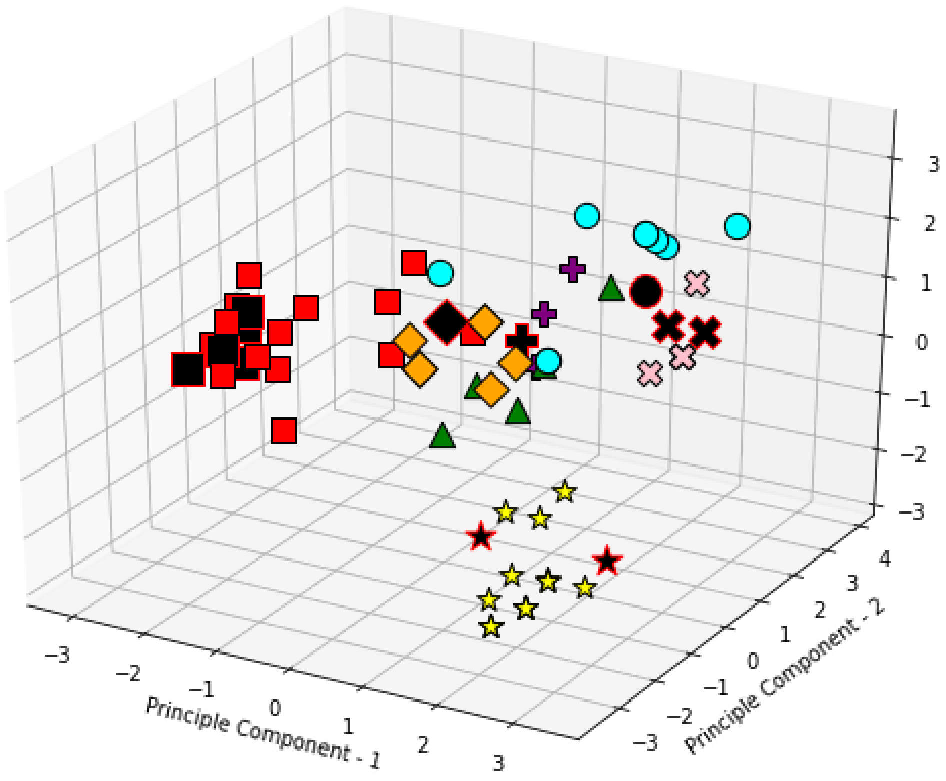
➢ Predicted Categories of the instances present in our test data:

```
[[4] [1] [1] [6] [1] [1] [5] [7] [1] [1] [2] [2] [4]]
```

➢ Actual Categories of the instances present in our test data:

```
[[4] [1] [1] [6] [1] [1] [5] [7] [1] [1] [2] [2] [4]]
```

## 12. Visualising KNN classified test dataset



Here, the black figures represent the test data and their shape indicates the category they were classified into.

## 13. Conclusion

In this project, we had to classify the chosen dataset using machine learning algorithms. To improve the efficiency of training and testing the models, we had to perform some preliminary analysis to reduce the dimensions of the dataset.

We chose Principle Component Analysis (PCA) for this purpose. Also, we pre-processed (standardised and separated) the data to further improve the efficiency of performing PCA. Due to the target classes being arbitrary and loosely-defined, further analysis of data was hampered.

Through PCA, we observed that the 17 available attributes could be reduced down to 9 principle components while still retaining 90% of the variance of the dataset. Hence the dataset was updated accordingly.

Next, we divided the dataset into training and test sets and then we trained 2 classification models, Bayes and KNN and found that KNN performed better in classifying the chosen dataset hence proved to be a better option among the two.

Link to the Google colab jupyter notebook that was created for this project:
https://colab.research.google.com/drive/1WZg4UctDYp8VGyi2cDJyREJ4FQUbOqDe?usp=sharing#scrollTo=EFZpYaPhsykX