Prediction Of Breast Cancer

PRAGATI KUNDU, ASANSOL ENGINEERING COLLEGE, REG NO-181080110288

MOITREYEE BOSE, ASANSOL ENGINEERING COLLEGE, REG NO-181080110273

PAYAL JAISWAL ASANSOL ENGINEERING COLLEGE, REG NO-181080110286

SATAKSHI PANDEY, ASANSOL ENGINEERING COLLEGE, REG NO-181080110306

MOUSUMI ROY, ASANSOL ENGINEERING COLLEGE, REG NO-181080110276

Table Of Contents

Acknowledgement

I would like to express special thanks to my teacher, Mr. Arnab Chakraborty, who gave me this golden opportunity to do the project on ***"Prediction Of Breast Cancer" using different Machine Learning Models*** and also provided support to complete this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I'm about to embark.

I'm obliged to my project team members for the value information provided by them in their respective fields. I'm grateful for their cooperation during the period of my assignment.

(TEAM MEMBERS)

SATAKSHI PANDEY

PAYAL JAISWAL

MOUSUMI ROY

PRAGATI KUNDU

MOITREYEE BOSE

Project Objective

Breast cancer became the major source of mortality between women. The accessibility of healthcare datasets and data analysis promote the researchers to apply study in extracting unknown pattern from healthcare datasets. Most common type of medical hazard found in middle aged women is, breast cancer.

Mortality rate of women due to breast cancer can be reduced if can be detected at a relatively early stage. With the help of latest, efficient and advanced screening methods, the majority of such cancers are diagnosed when the disease is still at a localized stage. The utility of machine learning techniques in healthcare analysis is growing progressively.

Certainly, analysis of patient's clinical data and physician's judgment are the most considerable features in diagnosis. Most of the possible medical flaws can be avoided by the using classification systems, and also offer healthcare data to be analyse in lesser time and in more exhaustive manner.

Accurate and timely prediction of breast cancer allows physicians and healthcare providers to make most favourable decision about the patient treatment.

The Objective of this study is to design <u>a prediction system</u> that can predict the incidence of the breast cancer at early stage by analysing smallest set of attributes that has been selected from the clinical dataset. *Wisconsin breast cancer dataset (WBCD)* have been used to conduct the proposed experiment.

We have made four different machine learning models (**Logistic Regression, Decision Tree, K-NN, Naïve Bayes**) which determines whether the tumour type i.e., *Benign or Malignant* using the given the clinical data set and have calculated the accuracy of each model.

Project Scope

The broad scope of the Prediction of Breast Cancer project includes: -

- The project shall be available on an online platform for 24x7 access for the medical staff for fast diagnostics.
- The project can change the way in which we see the medical time line. We can work a lot faster and save many lives due to the fast prediction.
- If any patient wants to know or wants to confirm the test results, they can easily provide the required data and know the predicted result.
- These models can also reduce the man power.
- We can modify the models as per the requirement and use them to in different situation such as predicting different diseases and many more.

Data Description

Source: The dataset was obtained from UCI repository and is a benchmark dataset. Breast Cancer Wisconsin (Original) Dataset contains 683 instances.

Data Description: There are in total 11 features. These 11 features provide precise information pertaining to the occurrence of breast cancer.

Attribute name	**Type**	**Description**	**Target Attribute**
Sample Code Number	Non-categorical	A code number for each observation.	No
Clump Thickness	Non-categorical	If cells are mono-layered or multi-layered. Benign cells tend to be grouped in mono-layers, while cancerous cells are often grouped in multi-layer.	No
Uniformity of Cell Size	Non-categorical	It is used to evaluate the consistency in the size of cells in the sample. Cancer cells tend to vary in size.	No
Uniformity of Cell Shape	Non-categorical	It is used to estimate the equality of cell shapes and identifies marginal variances, because cancer cells tend to vary in shape.	No
Marginal Adhesion	Non-categorical	Normal cells tend to stick together. Cancer cells tend to lose this ability. So, loss of adhesion is a sign of malignancy.	No
Single Epithelial Cell Size	Non-categorical	It is related to the uniformity. Epithelial cells that are significantly enlarged may be a malignant cell.	No

Bare Nuclei	Non-categorical	This is a term used for nuclei that is not surrounded by cytoplasm. Those are typically seen in benign tumours.	No
Bland Chromatin	Non-categorical	Describes a uniform “texture” of the nucleus seen in benign cells. In cancer cells, the chromatin tends to be coarser.	No
Normal Nucleoli	Non-categorical	Nucleoli are small structures seen in the nucleus. In normal cells the nucleolus is usually very small if visible at all. In cancer cells the nucleoli become much more prominent, and sometimes there are more of them.	No
Mitoses	Non-categorical	It is an estimate of the number of mitosis that has taken place. Larger the value, greater is the chance of malignancy.	No
Class	Non-categorical	For identifying patients at risk of breast cancer (where Class = 2 refers to BENIGN, and Class = 4 refers to MALIGNANT)	Yes

Table 1: Data description table

Data statistics:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Normal Nucleoli	Mitoses
mean	4.442167	3.150805	3.215227	2.830161	3.234261	2.869693	1.603221
std	2.820761	3.065145	2.988581	2.864562	2.223085	3.052666	1.732674
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	4.000000	1.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

Table 2: Statistics of the given dataset

Now we will pre-process the data. The methodology followed is given below:

- **Checking for null values**.

    - If null values are present, we will fill them or drop the row containing the null value based on the dataset.
- **Checking for categorical data**.
    - If some columns are categorical, we convert them into non-categorical data accordingly.
- **Checking for columns that are not necessary.**
    - If there are some columns such as Id, sample number etc. which are irrelevant for training the model, then we will drop that column.

We have searched for null values and formed the following table accordingly.

Attribute name	**Count of Null Values**
Sample Code Number	0
Clump Thickness	0
Uniformity of Cell Size	0
Uniformity of Cell Shape	0
Marginal Adhesion	0
Single Epithelial Cell Size	0
Bare Nuclei	0
Bland Chromatin	0
Normal Nucleoli	0
Mitoses	0
Class	0

Table 3: Count of null values

To visualise the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:

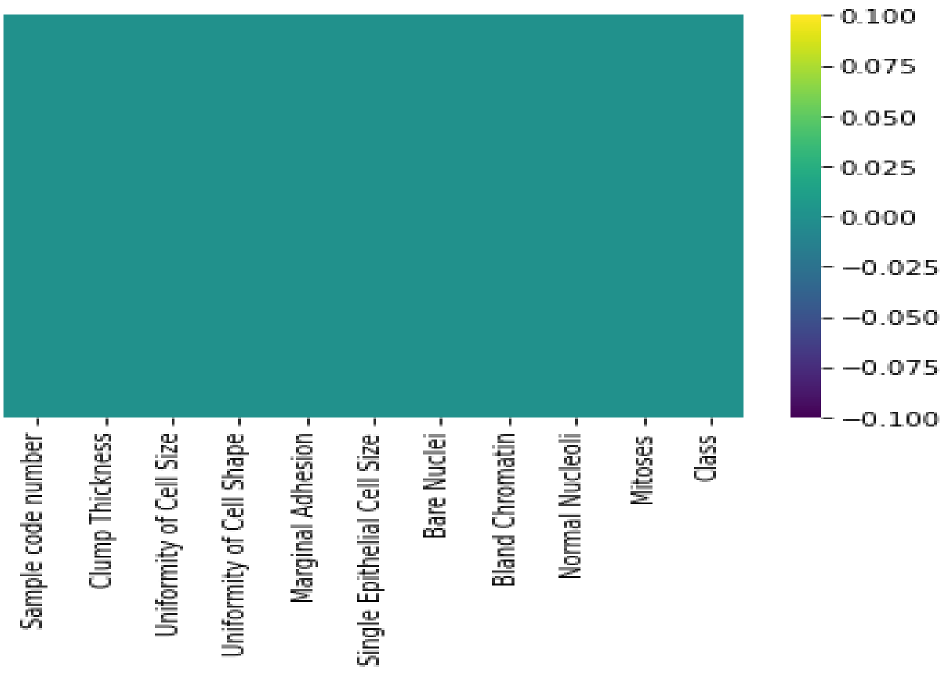


Fig 1: Heatmap of the dataset

The heatmap shows that the dataset has no null values

Since we have zero null values and zero categorical data so, we didn't perform any operations. The data set is intact as it was at the beginning.

So, we moving to find the correlations of different attributes to our target i.e., "Class" column in the dataset.

The following table gives the correlation value of each attribute with our target attribute i.e., "Class" column:

Attribute name	**Correlation value**
Clump Thickness	0.71
Uniformity of Cell Size	0.82
Uniformity of Cell Shape	0.82
Marginal Adhesion	0.71
Single Epithelial Cell Size	0.69
Bare Nuclei	0.82
Bland Chromatin	0.76
	0.72

Normal Nucleoli	
Mitoses	0.42

Table 4: Correlation value with target attribute

To visualise the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:

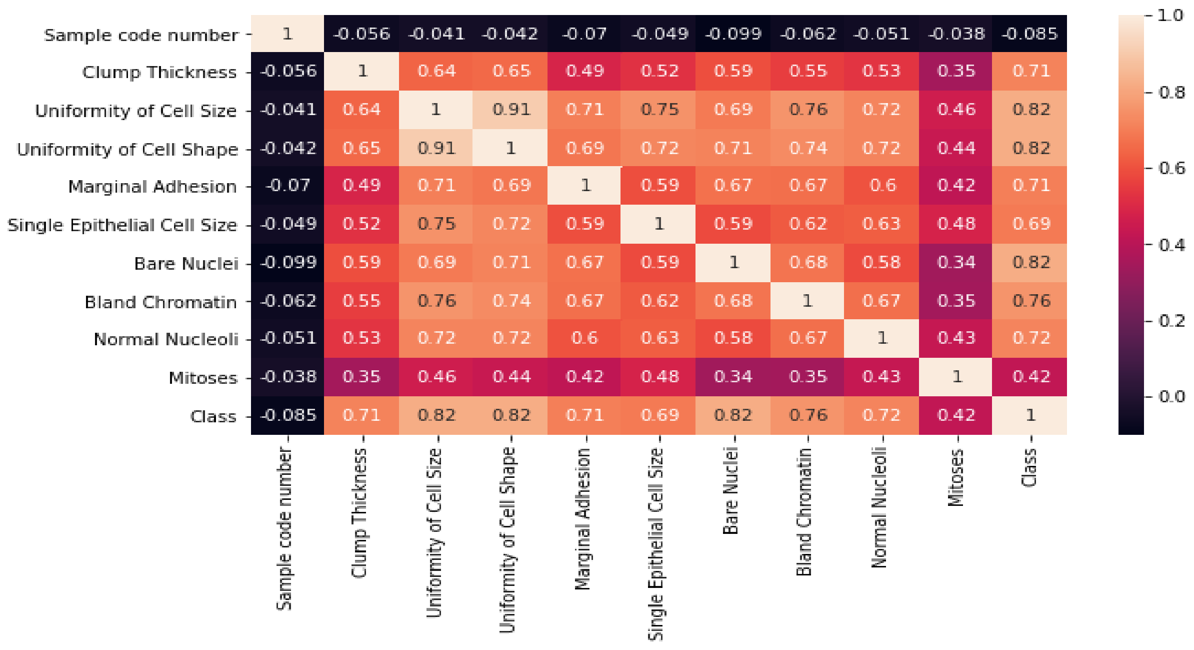


Fig 2: Correlation with the target attribute

We will now visualise other attributes with respect to our target column i.e., ‘Class’. We made a bar graphs using seaborn library function countplot. The bar graphs are given below:

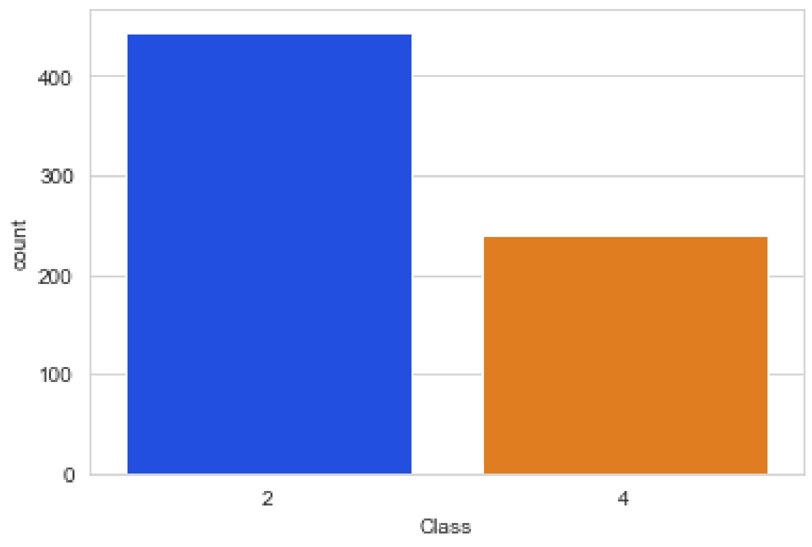


Fig 3: Count of class=2(benign) and class =4(malignant)

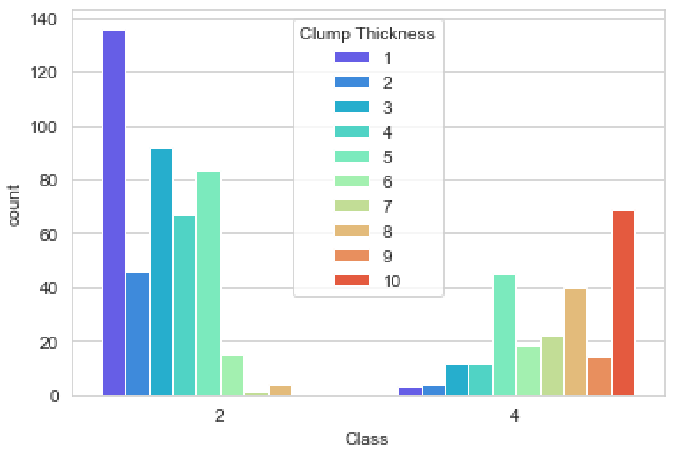


Fig 4: Plot between Clump thickness & Class

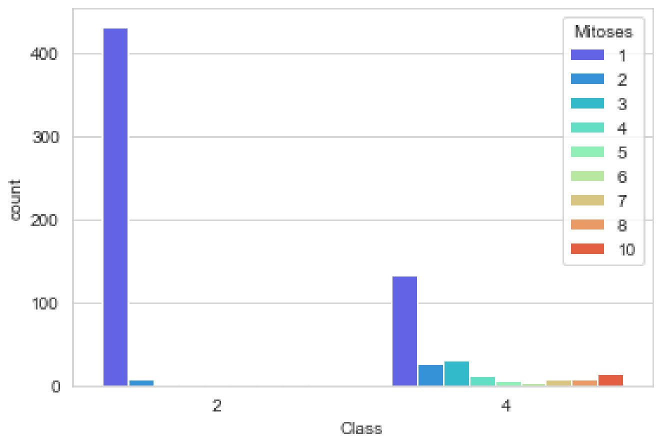


Fig 5: Plot between Mitoses & Class

Now we will delete the Sample Code Number column since it is not relevant for our model. After removing this is the information of our final dataset:

#	Column	Count	Non-Null	Dtype
0	Clump Thickness	683	non-null	int64
1	Uniformity of Cell Size	683	non-null	int64
2	Uniformity of Cell Shape	683	non-null	int64
3	Marginal Adhesion	683	non-null	int64
4	Single Epithelial Cell Size	683	non-null	int64
5	Bare Nuclei	683	non-null	int64
6	Bland Chromatin	683	non-null	int64
7	Normal Nucleoli	683	non-null	int64
8	Mitoses	683	non-null	int64
9	Class	683	non-null	int64

Table 5: Information of dataset

Model Building

Splitting data for training and testing purpose.

We split the given dataset into two parts for training and testing purpose. The split ratio we used is 0.70 which indicates we used 70% data for training purpose and 30% data for testing purpose. We will be the same split ratio for all the models trained.

Now we will be training our required models. Our project goal requires us to train specific 4 classifier models viz.

1. **Logistic Regression Classifier**
2. **Decision Tree Classifier**
3. **KNN Classifier**
4. **Naïve Bayes Classifier**

We will be using the final dataset obtained after pre-processing the given train dataset to train our required models.

LOGISTIC REGRESSION

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist.
In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labelled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labelled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labelled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labelling; the function that converts log-odds to probability is the logistic function, hence the name.

The object description of the Logistic Regression used is given below:

Logistic Regression classifier

Parameters	***Values***
C	1.0
class_weight	None
dual	False
fit_intercept	True
intercept_scaling	1
l1_ratio	None
max_iter	100
multi_class	auto
n_jobs	None
penalty	l2
random_state	0
solver	lbfgs
tol	0.0001
verbose	0

warm_start	False

Table 6: Object Parameter table for Logistic Regression classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

	Predicted Class 2	***Predicted class 4***
Class 2	84	3
Class 4	3	47

Table 7: Confusion Matrix of Logistic Regression classifier

Now the classification report of our decision tree:

	Precision	***Recall***	***F1-score***	***Support***
2	0.97	0.97	0.97	87
4	0.94	0.94	0.94	50
weighted avg	0.96	0.96	0.96	137

Table 8: Classification Report of Decision tree classifier

Accuracy of our model: 96.70 %

Standard Deviation of our model: 1.97 %

Advantages

- Logistic regression is easier to implement, interpret, and very efficient to train.
- It makes no assumptions about distributions of classes in feature space.
- It can easily extend to multiple classes(multinomial regression) and a natural probabilistic view of class predictions.
- It is very fast at classifying unknown records.

- Good accuracy for many simple data sets and it performs well when the dataset is linearly separable.

Disadvantages

- It is tough to obtain complex relationships using logistic regression. More powerful and compact algorithms such as Neural Networks can easily outperform this algorithm.
- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.

DECISION TREE CLASSIFIER

Decision tree learning is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity. There are two main types of Decision Trees:

1. **Classification trees (Yes/No types)**

   The outcome is a variable like 'Yes' or 'No'. Here the decision variable is Categorical.

2. **Regression trees (Continuous data types)**

   Here the decision or the outcome variable is Continuous, e.g., a number like 123. Decision variable is Non-categorical.

The term Classification And Regression Tree (CART) analysis is an umbrella term used to refer to both of the above procedures.

Notable decision tree algorithms include:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)
- Chi-square automatic interaction detection (CHAID). Performs multi-level splits when computing classification trees.[14]
- MARS: extends decision trees to handle numerical data better.

We have developed our model using the CART algorithm.

The object description of the decision tree Classifier used is given below:

DecisionTreeClassifier	
Parameters	***Values***
criterion	gini
Splitter	best
max_depth	5
min_samples_split	2
min_samples_leaf	5
min_weight_fraction_leaf	0.0
max_features	None
random_state	100
max_leaf_nodes	None

min_impurity_decrease	0.0
min_impurity_split	0
class_weight	None
ccp_alpha	0.0

Table 6: Object Parameter table for Decision tree classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

	Predicted Class 2	Predicted Class 4
Actual class 2	126	6
Actual class 4	10	63

Table 7: Confusion Matrix of Decision tree classifier

Now the classification report of our decision tree:

	Precision	Recall	F1-score	Support
2	0.93	0.95	0.94	132
4	0.91	0.86	0.89	73
Weighted avg.	0.92	0.92	0.92	205

Table 8: Classification Report of Decision tree classifier

Accuracy of our model: 92.19512195121952%

The decision tree prepared by the model:

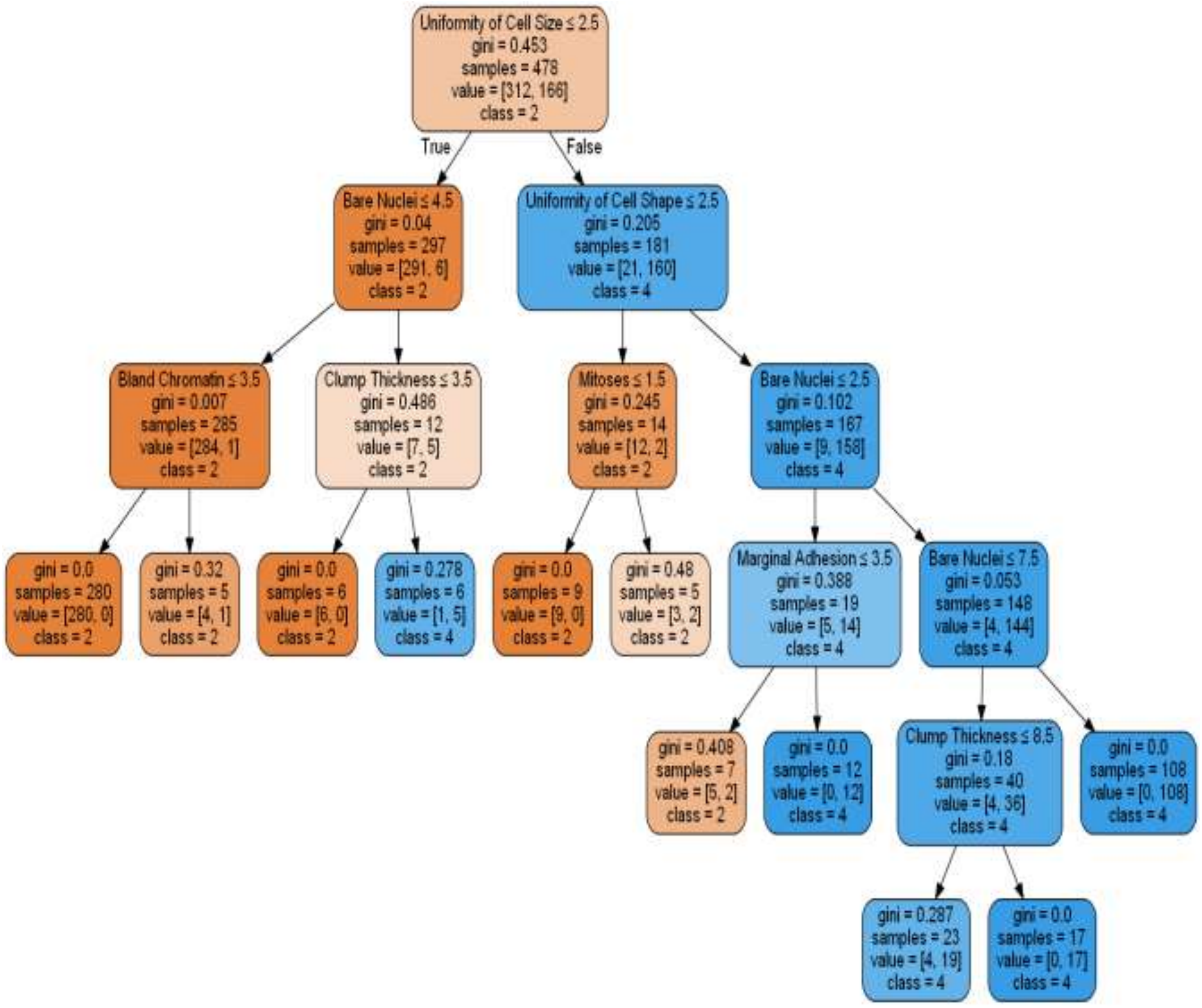


Fig 6: Decision Tree Visualization

Advantages

- Simple to understand and interpret.
- Able to handle both numerical and categorical data
- Requires little data preparation
- Performs well with large datasets.
- Mirrors human decision making more closely than other approaches

Limitations

- Trees can be very non-robust. A small change in the training data can result in a large change in the tree and consequently the final predictions.
- If we allow them to grow limitlessly, they can completely "memorize" the
- training data, just from creating more and more and more branches.

KNN CLASSIFIER

k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

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.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

K-Nearest Neighbor (KNN) Algorithm for Machine Learning

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

K-Nearest Neighbor (KNN) Algorithm for Machine Learning

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.
- Disadvantages of KNN Algorithm:
- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

NAÏVE BAYES CLASSIFIER

Introduction

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of weather conditions and corresponding target variable "Play". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

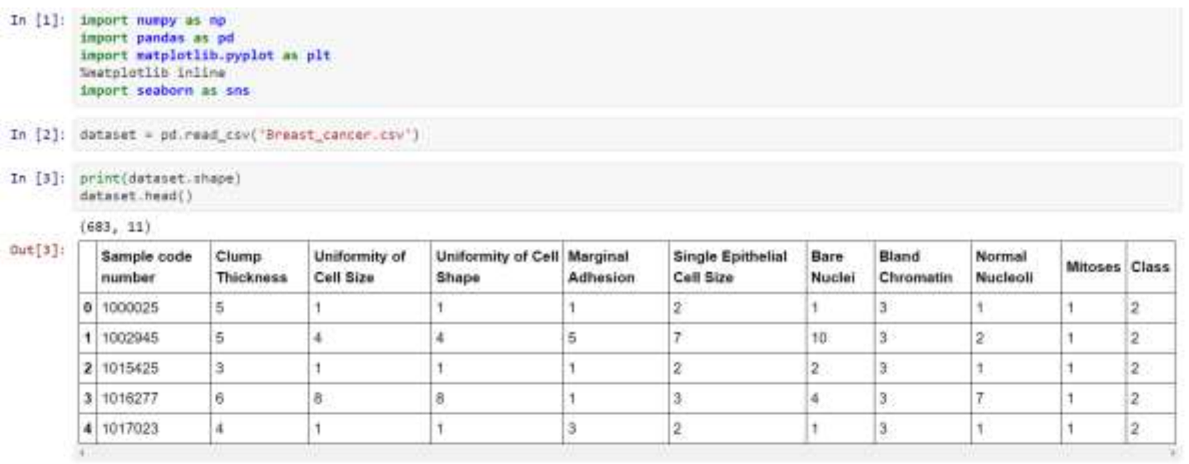
Cons

The followings are some cons of using Naïve Bayes classifiers –

- One of the most important cons of Naïve Bayes classification is its strong feature independence because in real life it is almost impossible to have a set of features which are completely independent of each other.
- Another issue with Naïve Bayes classification is its ‘zero frequency’ which means that if a categorial variable has a category but not being observed in training data set, then Naïve Bayes model will assign a zero probability to it and it will be unable to make a prediction.

Codes

DATASET CLEANING: -



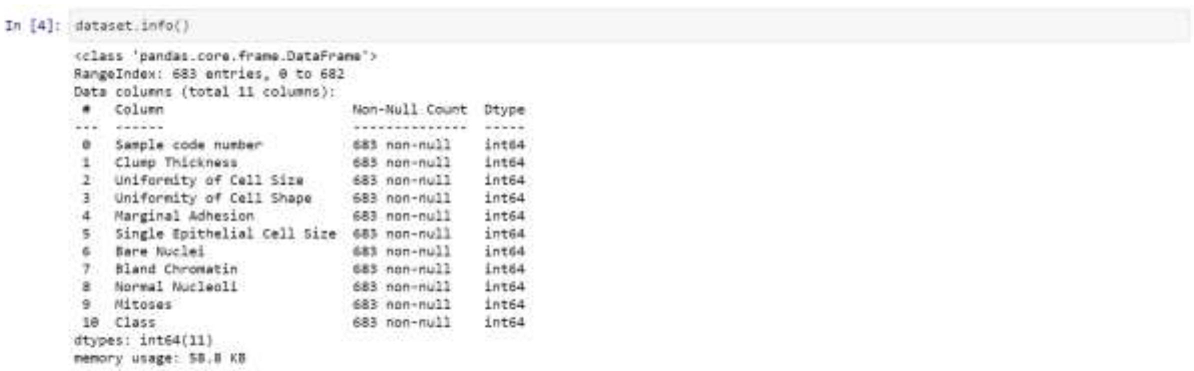
```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
```

```
In [2]: dataset = pd.read_csv('Breast_cancer.csv')
```

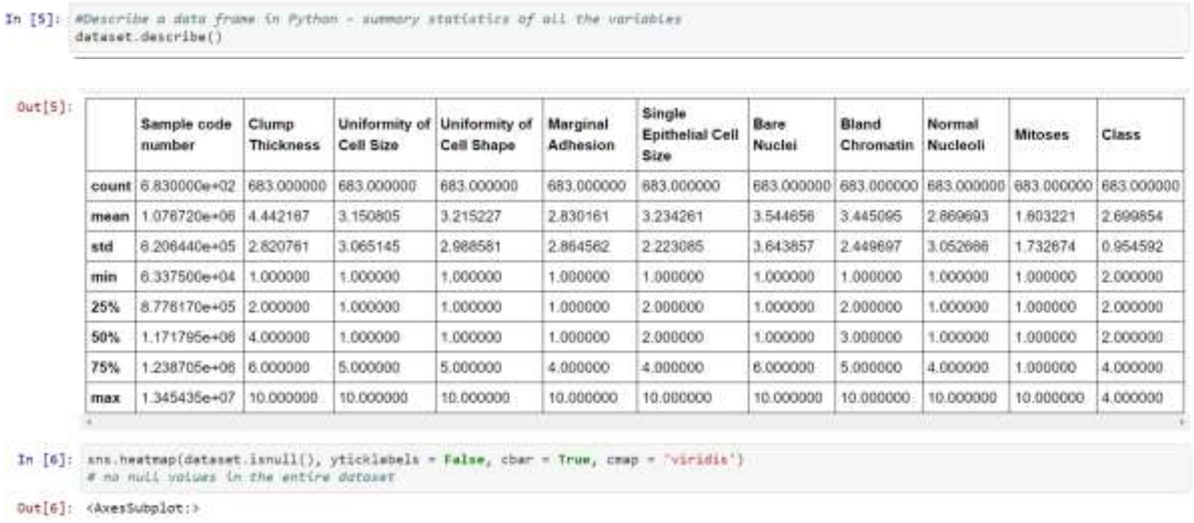
```
In [3]: print(dataset.shape)
        dataset.head()
```

```
        (683, 11)
Out[3]:
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2



```
In [4]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 683 entries, 0 to 682
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Sample code number           683 non-null    int64
 1   Clump Thickness              683 non-null    int64
 2   Uniformity of Cell Size      683 non-null    int64
 3   Uniformity of Cell Shape     683 non-null    int64
 4   Marginal Adhesion            683 non-null    int64
 5   Single Epithelial Cell Size  683 non-null    int64
 6   Bare Nuclei                  683 non-null    int64
 7   Bland Chromatin              683 non-null    int64
 8   Normal Nucleoli              683 non-null    int64
 9   Mitoses                      683 non-null    int64
 10  Class                        683 non-null    int64
dtypes: int64(11)
memory usage: 58.8 KB
```



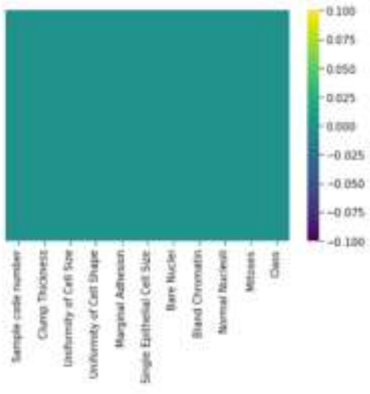
```
In [5]: #Describe a data frame in Python - summary statistics of all the variables
        dataset.describe()
```

```
Out[5]:
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
count	6.830000e+02	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000
mean	1.076720e+06	4.442167	3.150805	3.215227	2.830161	3.234261	3.544656	3.445095	2.869693	1.603221	2.699854
std	6.206440e+05	2.820761	3.065145	2.988581	2.864562	2.223085	3.643857	2.449697	3.052666	1.732674	0.954592
min	6.337500e+04	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000
25%	8.776170e+05	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000	2.000000
50%	1.171795e+06	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.000000	2.000000
75%	1.238705e+06	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.000000	4.000000
max	1.345435e+07	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4.000000

```
In [6]: sns.heatmap(dataset.isnull(), yticklabels = False, cbar = True, cmap = 'viridis')
        # no null values in the entire dataset

Out[6]: <AxesSubplot:>
```

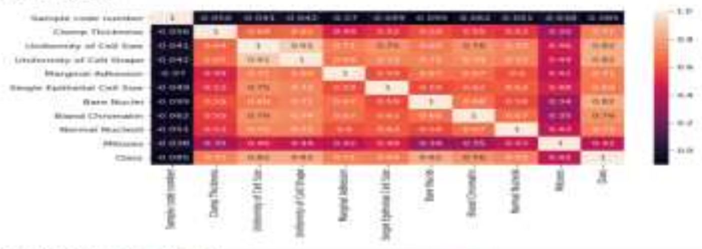
```
Out[6]: <AxesSubplot:>
```



```
In [7]: plt.figure(figsize = (10, 5))
        sns.heatmap(dataset.corr(), annot = True, annot_kws={"size":10})
```

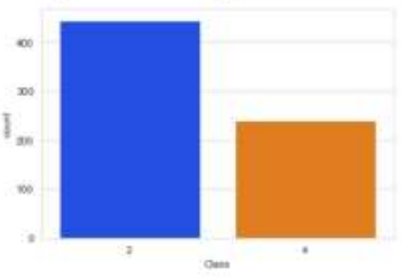
```
In [7]: plt.figure(figsize = (10, 5))
        sns.heatmap(dataset.corr(), annot = True, annot_kws={"size":10})
```

```
Out[7]: <AxesSubplot:>
```



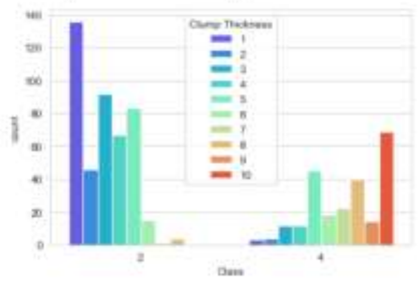
```
In [8]: sns.set_style('whitegrid')
        sns.countplot(x ='Class', data = dataset, palette ='bright')
        # 2 - BENIGN, 4 - MALIGNANT
```

```
Out[8]: <AxesSubplot:xlabel='Class', ylabel='count'>
```



```
In [9]: sns.countplot(x = 'Class', hue = 'Clump Thickness', data = dataset, palette = 'rainbow')
        # 2(BENIGN) - grouped in mono-layers, 4(MALIGNANT) - grouped in multi-layer.
```

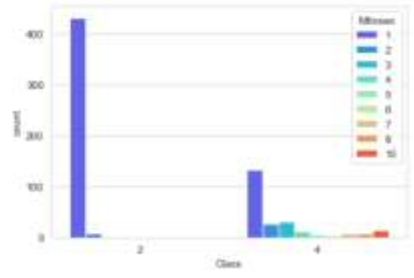
```
Out[9]: <AxesSubplot:xlabel='Class', ylabel='count'>
```



```
In [10]: sns.set_style('whitegrid')
         sns.countplot(x = 'Class', hue = 'Mitoses', data = dataset, palette = 'rainbow')
         # 2(BENIGN) - small value(1,2), 4(MALIGNANT) - large value(10,8,...)
```

```
Out[10]: <AxesSubplot:xlabel='Class', ylabel='count'>
```

```
In [10]: sns.set_style('whitegrid')
         sns.countplot(x = 'Class', hue = 'Mitoses', data = dataset, palette = 'rainbow')
         # 2(BENIGN) - small value(1,2), 4(MALIGNANT) - large value(10,8,...)

Out[10]: <AxesSubplot:xlabel='Class', ylabel='count'>
```

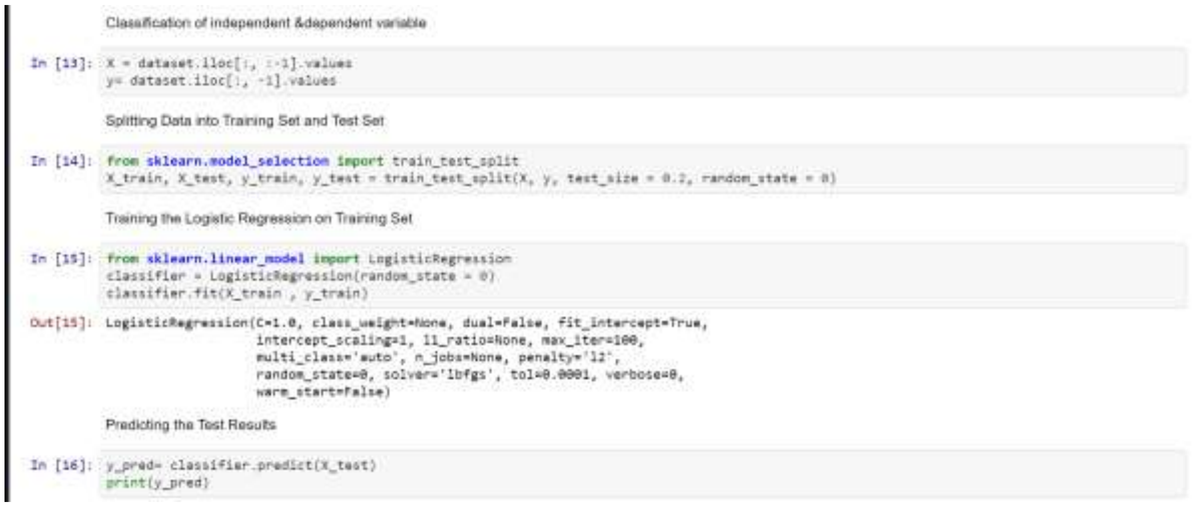


```
In [11]: # Drop a column
         dataset.drop('Sample code number', inplace = True, axis = 1)

         print(dataset.shape)
         dataset.head()

         (683, 10)
Out[11]:
```

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2

```
In [12]: dataset.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 683 entries, 0 to 682
         Data columns (total 10 columns):
          #   Column                       Non-Null Count  Dtype
         ---  ------                       --------------  -----
          0   Clump Thickness              683 non-null    int64
          1   Uniformity of Cell Size      683 non-null    int64
          2   Uniformity of Cell Shape     683 non-null    int64
          3   Marginal Adhesion            683 non-null    int64
          4   Single Epithelial Cell Size  683 non-null    int64
          5   Bare Nuclei                  683 non-null    int64
          6   Bland Chromatin              683 non-null    int64
          7   Normal Nucleoli              683 non-null    int64
          8   Mitoses                      683 non-null    int64
          9   Class                        683 non-null    int64
         dtypes: int64(10)
         memory usage: 53.5 KB

In [ ]:
```

LOGISTIC REGRESSION: -



Classification of independent &dependent variable

```
In [13]: X = dataset.iloc[:, :-1].values
         y= dataset.iloc[:, -1].values
```

Splitting Data into Training Set and Test Set

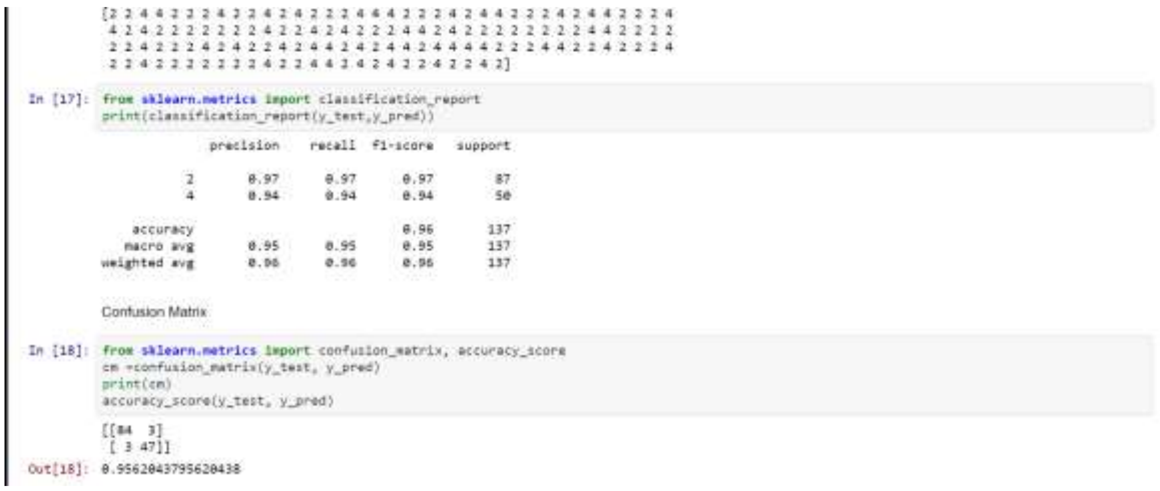
```
In [14]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Training the Logistic Regression on Training Set

```
In [15]: from sklearn.linear_model import LogisticRegression
         classifier = LogisticRegression(random_state = 0)
         classifier.fit(X_train , y_train)

Out[15]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                            warm_start=False)
```

Predicting the Test Results

```
In [16]: y_pred= classifier.predict(X_test)
         print(y_pred)
```

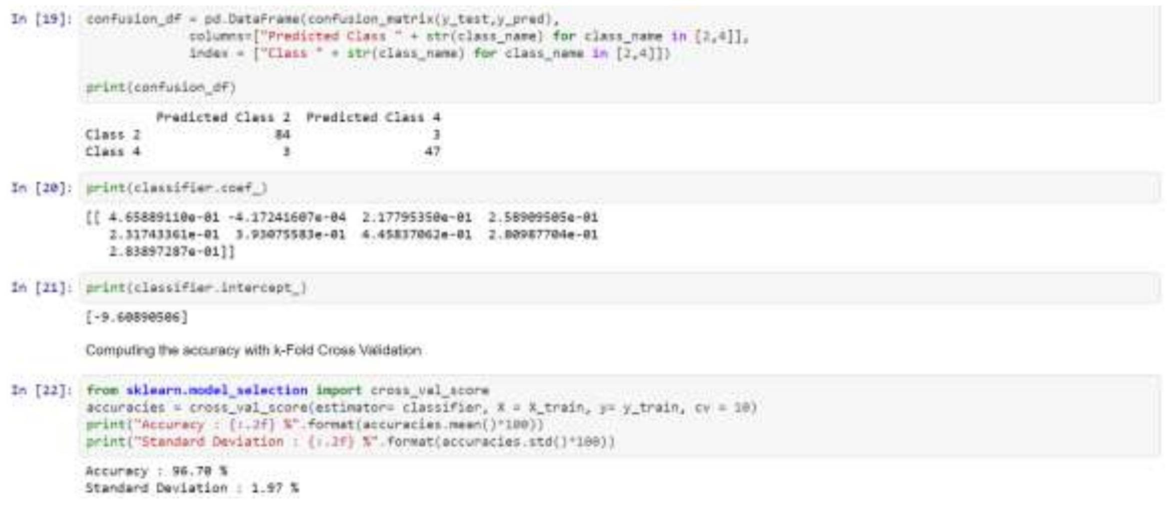


```
[2 2 4 4 2 2 2 4 2 2 4 2 4 2 2 2 4 4 4 2 2 2 4 2 4 4 2 2 2 4 2 4 4 2 2 2 4
 4 2 4 2 2 2 2 2 2 2 4 2 2 4 2 4 2 2 2 4 4 2 4 2 2 2 2 2 2 2 2 4 4 2 2 2 2
 2 2 4 2 2 2 4 2 4 2 2 4 2 4 4 2 4 2 4 4 2 4 4 4 4 2 2 2 4 4 2 2 4 2 2 2 4
 2 2 4 2 2 2 2 2 2 2 4 2 2 4 4 2 4 2 4 2 2 4 2 2 4 2]
```

```
In [17]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           2       0.97      0.97      0.97        87
           4       0.94      0.94      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.95      0.95       137
weighted avg       0.96      0.96      0.96       137
```

Confusion Matrix

```
In [18]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm =confusion_matrix(y_test, y_pred)
         print(cm)
         accuracy_score(y_test, y_pred)

         [[84  3]
          [ 3 47]]

Out[18]: 0.9562043795620438
```



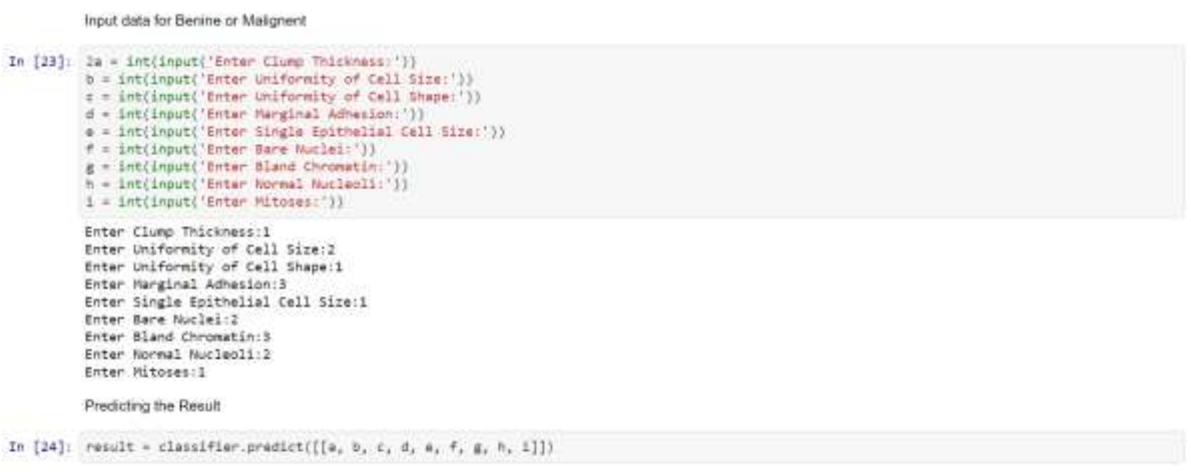
```
In [19]: confusion_df = pd.DataFrame(confusion_matrix(y_test,y_pred),
                      columns=["Predicted Class " + str(class_name) for class_name in [2,4]],
                      index = ["Class " + str(class_name) for class_name in [2,4]])

         print(confusion_df)

                  Predicted Class 2  Predicted Class 4
         Class 2                 84                  3
         Class 4                  3                 47
```

```
In [20]: print(classifier.coef_)

         [[ 4.65889110e-01 -4.17241607e-04  2.17795350e-01  2.58909505e-01
            2.31743361e-01  3.93075583e-01  4.45837062e-01  2.80987704e-01
            2.83897287e-01]]
```

```
In [21]: print(classifier.intercept_)

         [-9.60890506]
```

Computing the accuracy with k-Fold Cross Validation

```
In [22]: from sklearn.model_selection import cross_val_score
         accuracies = cross_val_score(estimator= classifier, X = X_train, y= y_train, cv = 10)
         print("Accuracy : {:.2f} %".format(accuracies.mean()*100))
         print("Standard Deviation : {:.2f} %".format(accuracies.std()*100))

         Accuracy : 96.70 %
         Standard Deviation : 1.97 %
```



Input data for Benine or Malignent

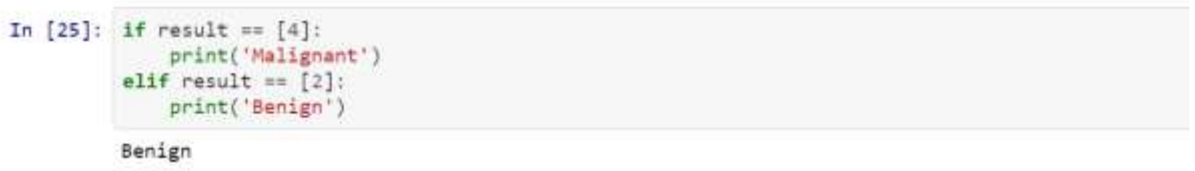
```
In [23]: 2a = int(input('Enter Clump Thickness:'))
         b = int(input('Enter Uniformity of Cell Size:'))
         c = int(input('Enter Uniformity of Cell Shape:'))
         d = int(input('Enter Marginal Adhesion:'))
         e = int(input('Enter Single Epithelial Cell Size:'))
         f = int(input('Enter Bare Nuclei:'))
         g = int(input('Enter Bland Chromatin:'))
         h = int(input('Enter Normal Nucleoli:'))
         i = int(input('Enter Mitoses:'))
```

```
Enter Clump Thickness:1
Enter Uniformity of Cell Size:2
Enter Uniformity of Cell Shape:1
Enter Marginal Adhesion:3
Enter Single Epithelial Cell Size:1
Enter Bare Nuclei:2
Enter Bland Chromatin:3
Enter Normal Nucleoli:2
Enter Mitoses:1
```

Predicting the Result

```
In [24]: result = classifier.predict([[a, b, c, d, e, f, g, h, i]])
```

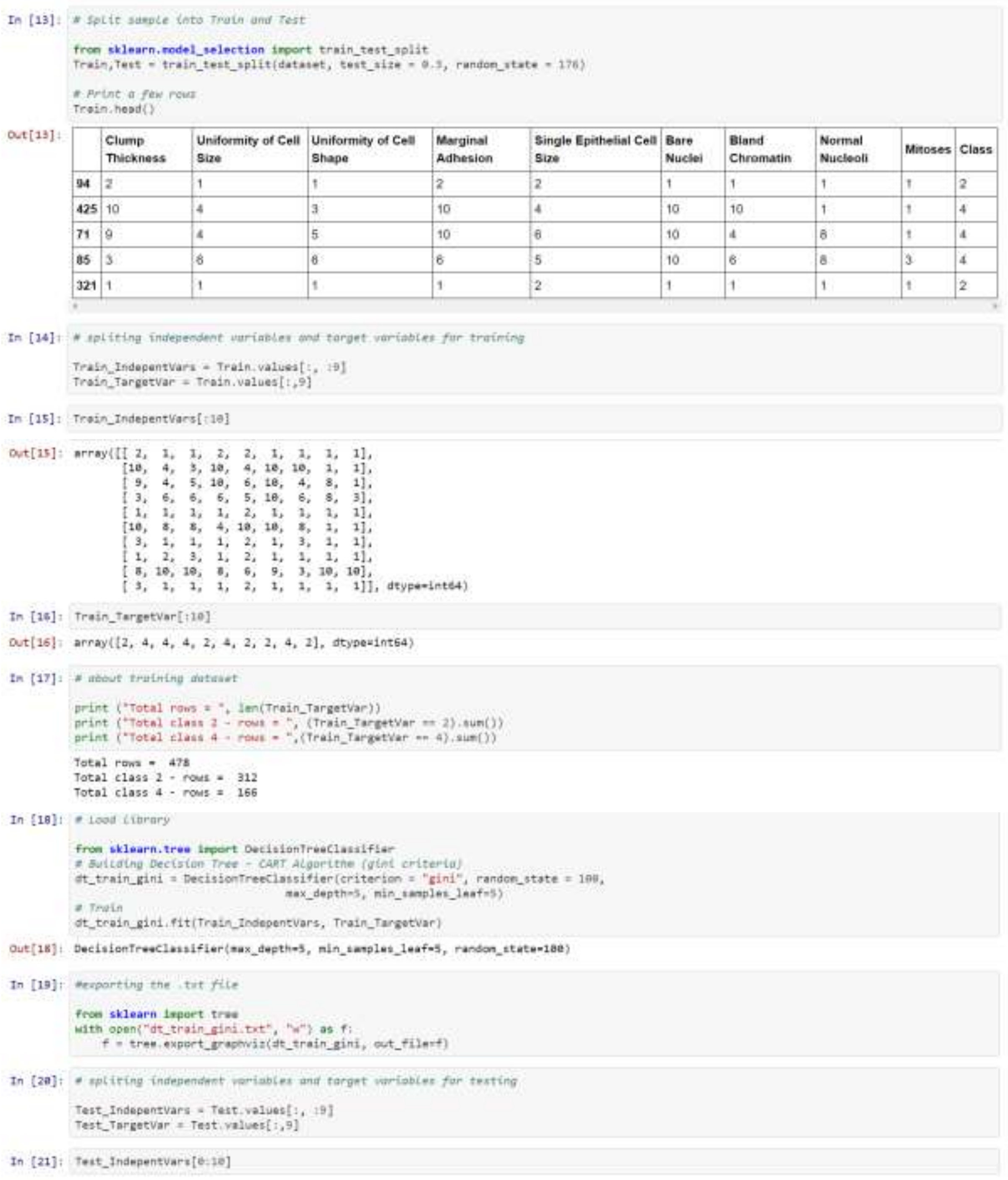
THE OUTPUT



```
In [25]: if result == [4]:
             print('Malignant')
         elif result == [2]:
             print('Benign')
```

```
Benign
```

DECISION TREE:

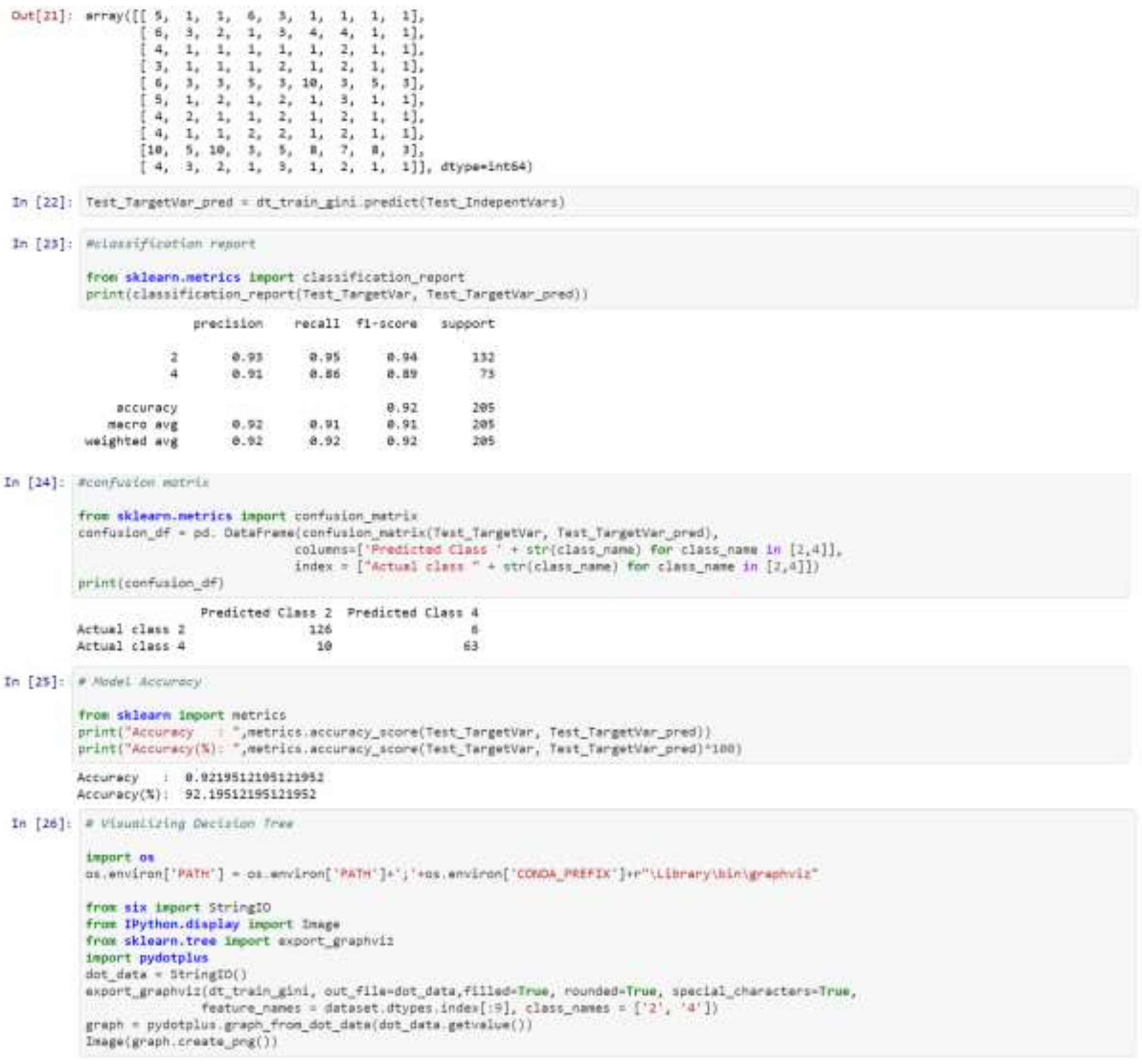


```
In [13]: # Split sample into Train and Test

         from sklearn.model_selection import train_test_split
         Train,Test = train_test_split(dataset, test_size = 0.3, random_state = 176)

         # Print a few rows
         Train.head()
```

Out[13]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
94	2	1	1	2	2	1	1	1	1	2
425	10	4	3	10	4	10	10	1	1	4
71	9	4	5	10	6	10	4	8	1	4
85	3	6	6	6	5	10	6	8	3	4
321	1	1	1	1	2	1	1	1	1	2

```
In [14]: # spliting independent variables and target variables for training

         Train_IndepentVars = Train.values[:, :9]
         Train_TargetVar = Train.values[:,9]

In [15]: Train_IndepentVars[:10]

Out[15]: array([[ 2,  1,  1,  2,  2,  1,  1,  1,  1],
                [10,  4,  3, 10,  4, 10, 10,  1,  1],
                [ 9,  4,  5, 10,  6, 10,  4,  8,  1],
                [ 3,  6,  6,  6,  5, 10,  6,  8,  3],
                [ 1,  1,  1,  1,  2,  1,  1,  1,  1],
                [10,  8,  8,  4, 10, 10,  8,  1,  1],
                [ 3,  1,  1,  1,  2,  1,  3,  1,  1],
                [ 1,  2,  3,  1,  2,  1,  1,  1,  1],
                [ 8, 10, 10,  8,  6,  9,  3, 10, 10],
                [ 3,  1,  1,  1,  2,  1,  1,  1,  1]], dtype=int64)

In [16]: Train_TargetVar[:10]

Out[16]: array([2, 4, 4, 4, 2, 4, 2, 2, 4, 2], dtype=int64)

In [17]: # about training dataset

         print ("Total rows = ", len(Train_TargetVar))
         print ("Total class 2 - rows = ", (Train_TargetVar == 2).sum())
         print ("Total class 4 - rows = ",(Train_TargetVar == 4).sum())

         Total rows =  478
         Total class 2 - rows =  312
         Total class 4 - rows =  166

In [18]: # Load library

         from sklearn.tree import DecisionTreeClassifier
         # Building Decision Tree - CART Algorithm (gini criteria)
         dt_train_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,
                                        max_depth=5, min_samples_leaf=5)
         # Train
         dt_train_gini.fit(Train_IndepentVars, Train_TargetVar)

Out[18]: DecisionTreeClassifier(max_depth=5, min_samples_leaf=5, random_state=100)

In [19]: #exporting the .txt file

         from sklearn import tree
         with open("dt_train_gini.txt", "w") as f:
             f = tree.export_graphviz(dt_train_gini, out_file=f)

In [20]: # spliting independent variables and target variables for testing

         Test_IndepentVars = Test.values[:, :9]
         Test_TargetVar = Test.values[:,9]

In [21]: Test_IndepentVars[0:10]
```



```
Out[21]: array([[ 5,  1,  1,  6,  3,  1,  1,  1,  1],
                [ 6,  3,  2,  1,  3,  4,  4,  1,  1],
                [ 4,  1,  1,  1,  1,  1,  2,  1,  1],
                [ 3,  1,  1,  1,  2,  1,  2,  1,  1],
                [ 6,  3,  3,  5,  3, 10,  3,  5,  3],
                [ 5,  1,  2,  1,  2,  1,  3,  1,  1],
                [ 4,  2,  1,  1,  2,  1,  2,  1,  1],
                [ 4,  1,  1,  2,  2,  1,  2,  1,  1],
                [10,  5, 10,  3,  5,  8,  7,  8,  3],
                [ 4,  3,  2,  1,  3,  1,  2,  1,  1]], dtype=int64)
```

```
In [22]: Test_TargetVar_pred = dt_train_gini.predict(Test_IndepentVars)
```

```
In [23]: #classification report

         from sklearn.metrics import classification_report
         print(classification_report(Test_TargetVar, Test_TargetVar_pred))
```

```
              precision    recall  f1-score   support

           2       0.93      0.95      0.94       132
           4       0.91      0.86      0.89        73

    accuracy                           0.92       205
   macro avg       0.92      0.91      0.91       205
weighted avg       0.92      0.92      0.92       205
```

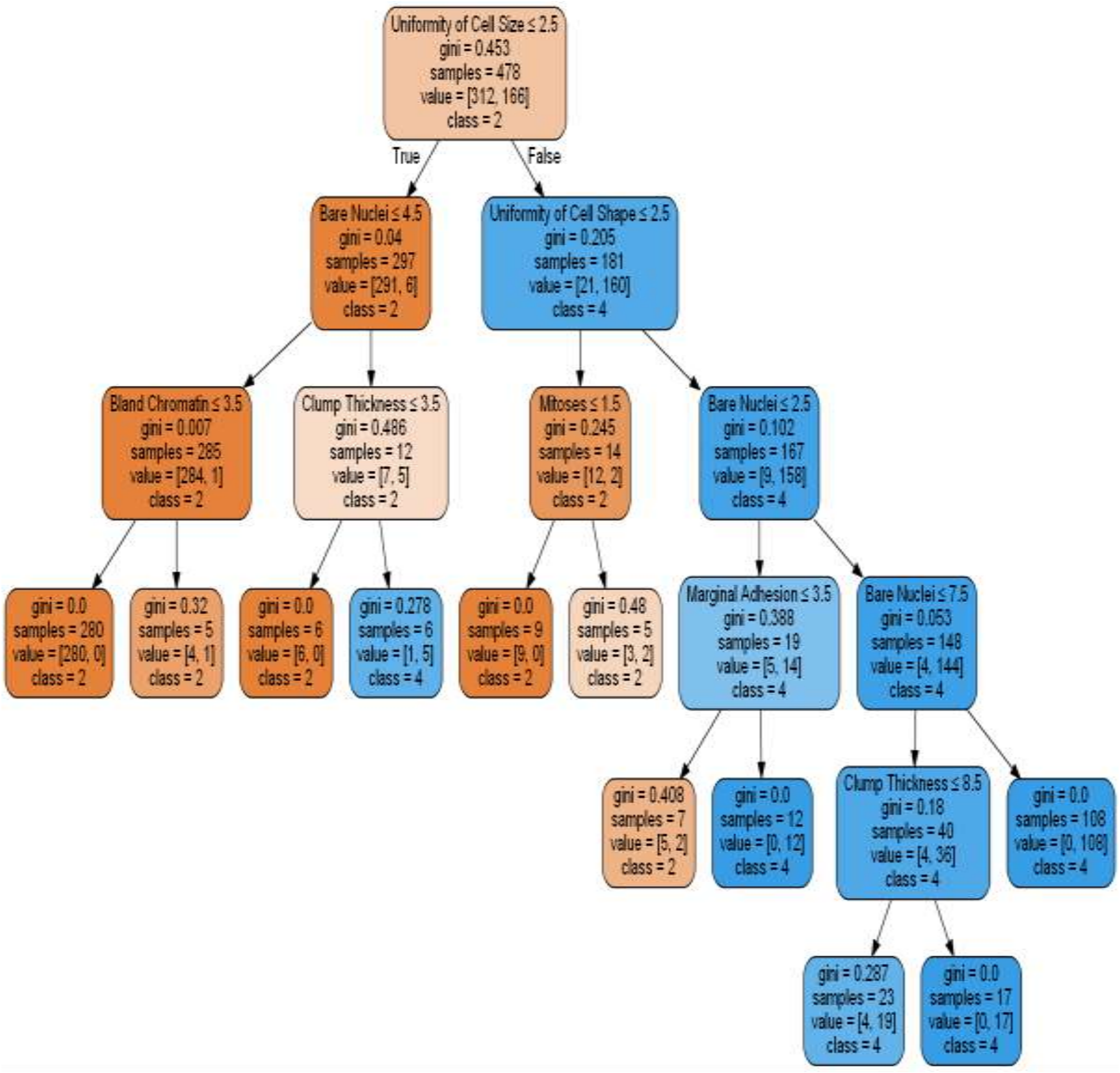
```
In [24]: #confusion matrix

         from sklearn.metrics import confusion_matrix
         confusion_df = pd. DataFrame(confusion_matrix(Test_TargetVar, Test_TargetVar_pred),
                                      columns=['Predicted Class ' + str(class_name) for class_name in [2,4]],
                                      index = ["Actual class " + str(class_name) for class_name in [2,4]])
         print(confusion_df)
```

```
                Predicted Class 2  Predicted Class 4
Actual class 2                126                  6
Actual class 4                 10                 63
```

```
In [25]: # Model Accuracy

         from sklearn import metrics
         print("Accuracy    : ",metrics.accuracy_score(Test_TargetVar, Test_TargetVar_pred))
         print("Accuracy(%): ",metrics.accuracy_score(Test_TargetVar, Test_TargetVar_pred)*100)
```

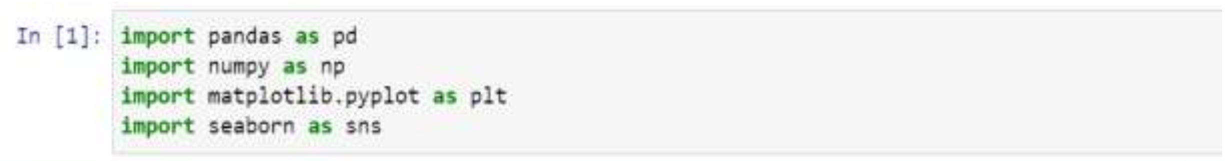
```
Accuracy    :  0.9219512195121952
Accuracy(%):  92.19512195121952
```

```
In [26]: # Visualizing Decision Tree

         import os
         os.environ['PATH'] = os.environ['PATH']+';'+os.environ['CONDA_PREFIX']+r"\Library\bin\graphviz"

         from six import StringIO
         from IPython.display import Image
         from sklearn.tree import export_graphviz
         import pydotplus
         dot_data = StringIO()
         export_graphviz(dt_train_gini, out_file=dot_data,filled=True, rounded=True, special_characters=True,
                         feature_names = dataset.dtypes.index[:9], class_names = ['2', '4'])
         graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
         Image(graph.create_png())
```

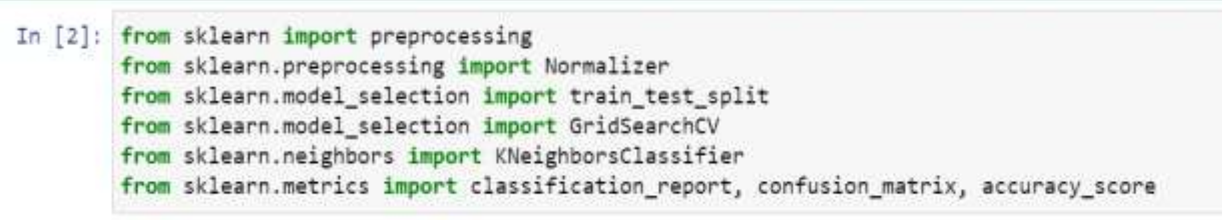


Uniformity of Cell Size ≤ 2.5 gini = 0.453 samples = 478 value = [312, 166] class = 2
True
False
Bare Nuclei ≤ 4.5 gini = 0.04 samples = 297 value = [291, 6] class = 2
Uniformity of Cell Shape ≤ 2.5 gini = 0.205 samples = 181 value = [21, 160] class = 4
Bland Chromatin ≤ 3.5 gini = 0.007 samples = 285 value = [284, 1] class = 2
Clump Thickness ≤ 3.5 gini = 0.486 samples = 12 value = [7, 5] class = 2
Mitoses ≤ 1.5 gini = 0.245 samples = 14 value = [12, 2] class = 2
Bare Nuclei ≤ 2.5 gini = 0.102 samples = 167 value = [9, 158] class = 4
gini = 0.0 samples = 280 value = [280, 0] class = 2
gini = 0.32 samples = 5 value = [4, 1] class = 2
gini = 0.0 samples = 6 value = [6, 0] class = 2
gini = 0.278 samples = 6 value = [1, 5] class = 4
gini = 0.0 samples = 9 value = [9, 0] class = 2
gini = 0.48 samples = 5 value = [3, 2] class = 2
Marginal Adhesion ≤ 3.5 gini = 0.388 samples = 19 value = [5, 14] class = 4
Bare Nuclei ≤ 7.5 gini = 0.053 samples = 148 value = [4, 144] class = 4
gini = 0.408 samples = 7 value = [5, 2] class = 2
gini = 0.0 samples = 12 value = [0, 12] class = 4
Clump Thickness ≤ 8.5 gini = 0.18 samples = 40 value = [4, 36] class = 4
gini = 0.0 samples = 108 value = [0, 108] class = 4
gini = 0.287 samples = 23 value = [4, 19] class = 4
gini = 0.0 samples = 17 value = [0, 17] class = 4

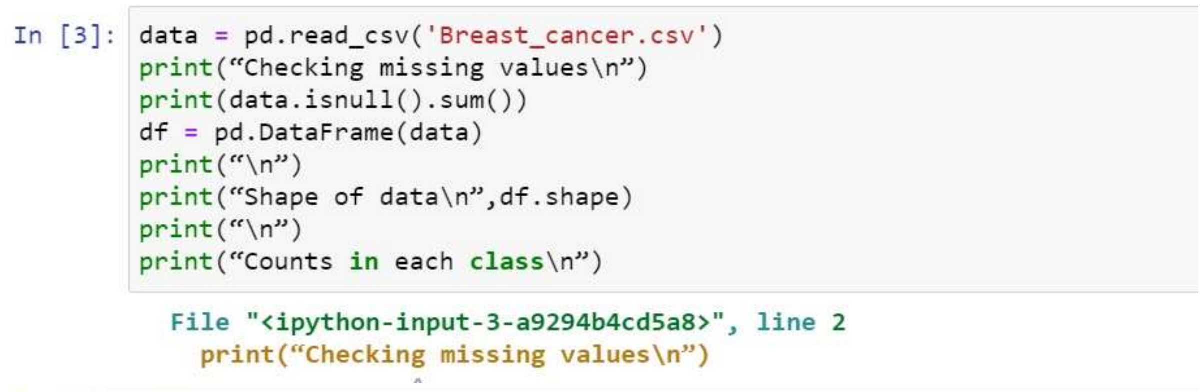
KNN CLASSIFIER:



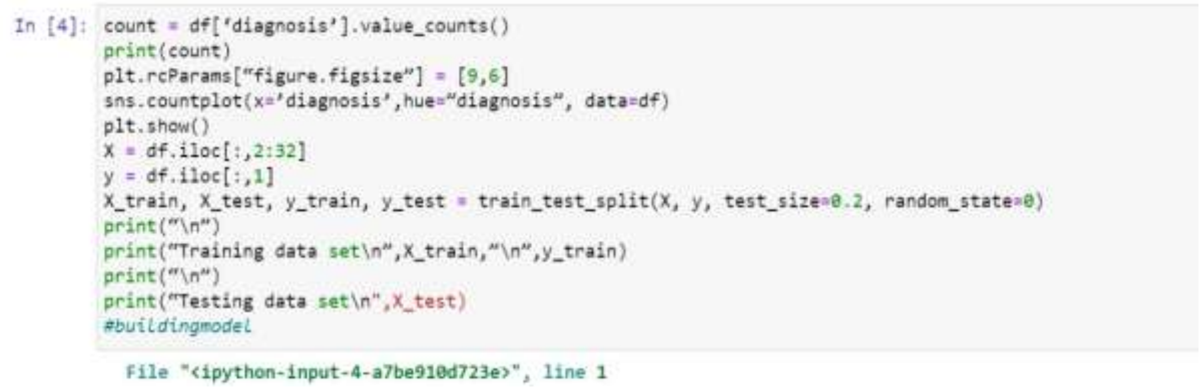
```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```



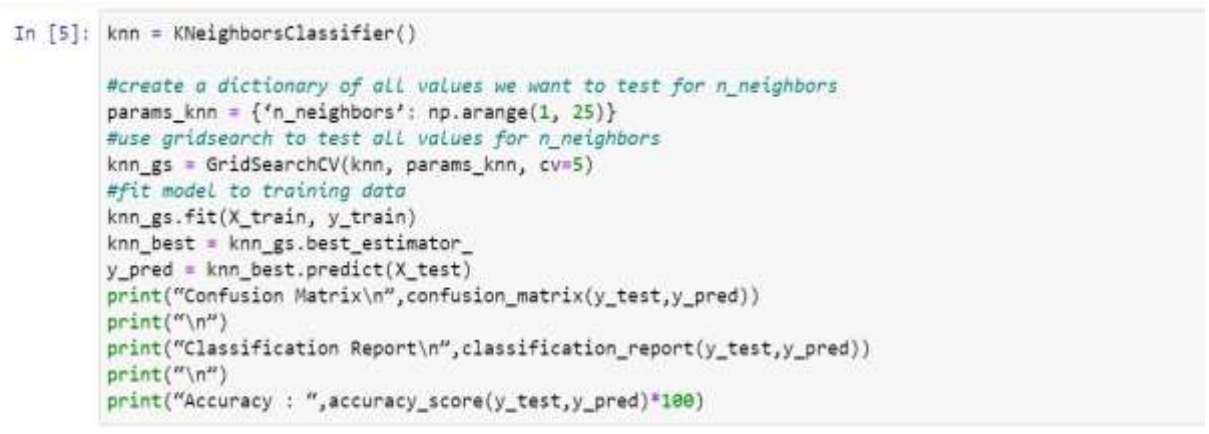
```
In [2]: from sklearn import preprocessing
        from sklearn.preprocessing import Normalizer
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import GridSearchCV
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```



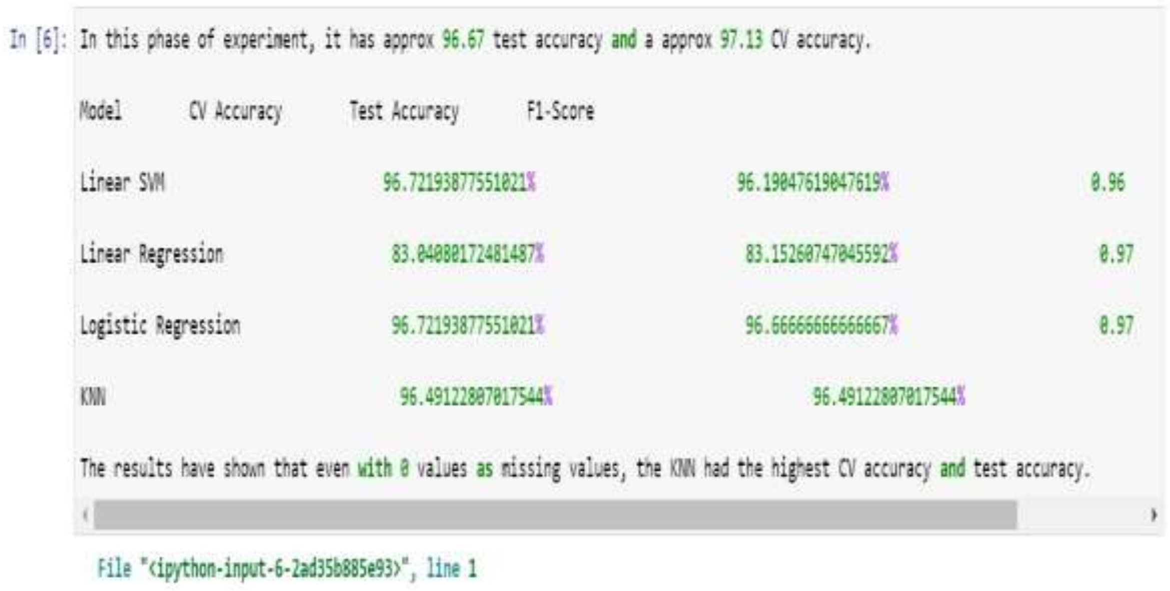
```
In [3]: data = pd.read_csv('Breast_cancer.csv')
        print(“Checking missing values\n”)
        print(data.isnull().sum())
        df = pd.DataFrame(data)
        print(“\n”)
        print(“Shape of data\n”,df.shape)
        print(“\n”)
        print(“Counts in each class\n”)

          File "<ipython-input-3-a9294b4cd5a8>", line 2
            print(“Checking missing values\n”)
                        ^
```



```
In [4]: count = df[‘diagnosis’].value_counts()
        print(count)
        plt.rcParams[“figure.figsize”] = [9,6]
        sns.countplot(x=‘diagnosis’,hue=“diagnosis”, data=df)
        plt.show()
        X = df.iloc[:,2:32]
        y = df.iloc[:,1]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
        print(“\n”)
        print(“Training data set\n”,X_train,“\n”,y_train)
        print(“\n”)
        print(“Testing data set\n”,X_test)
        #buildingmodel

          File "<ipython-input-4-a7be910d723e>", line 1
```

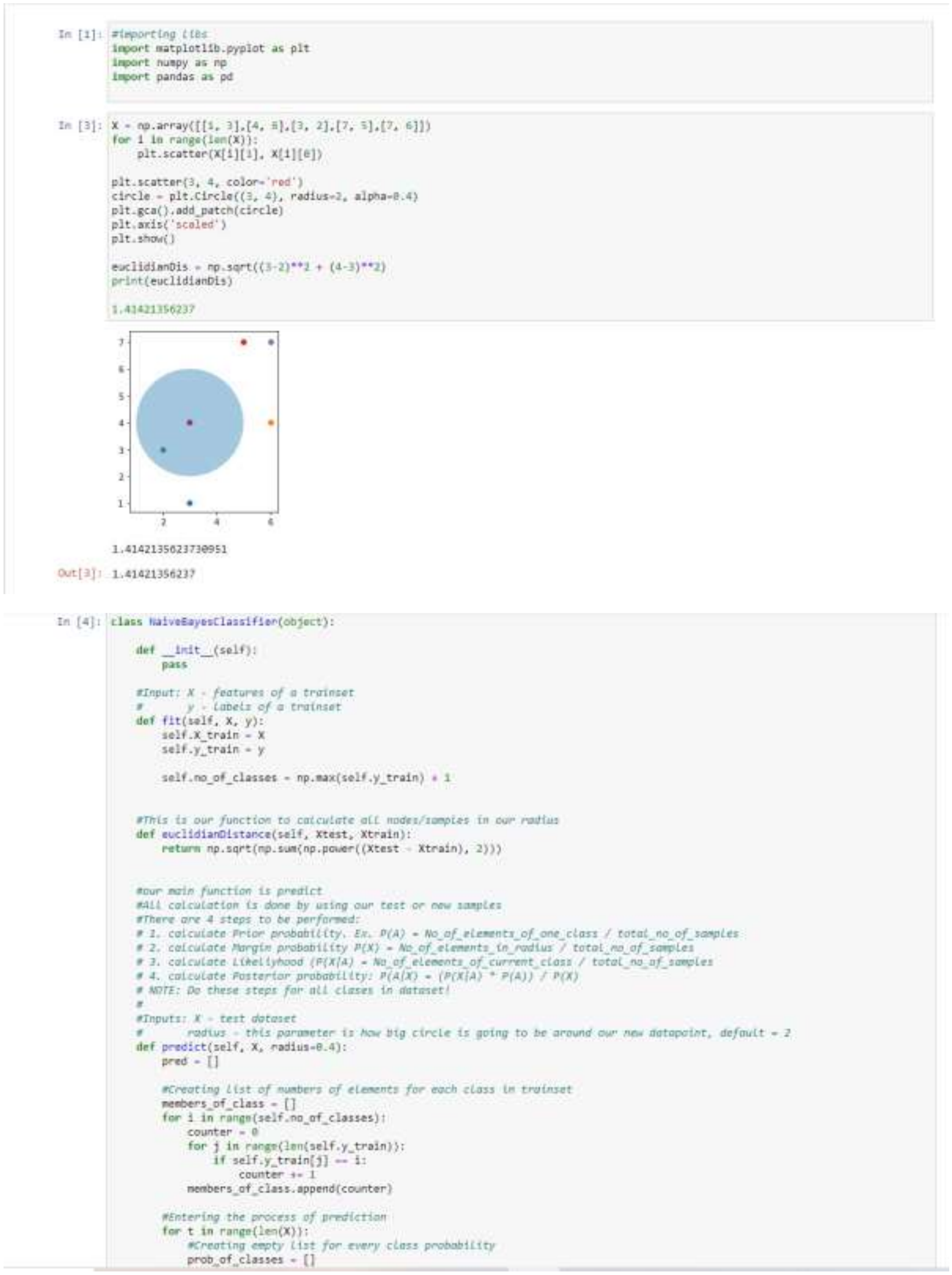


```
In [5]: knn = KNeighborsClassifier()

        #create a dictionary of all values we want to test for n_neighbors
        params_knn = {‘n_neighbors’: np.arange(1, 25)}
        #use gridsearch to test all values for n_neighbors
        knn_gs = GridSearchCV(knn, params_knn, cv=5)
        #fit model to training data
        knn_gs.fit(X_train, y_train)
        knn_best = knn_gs.best_estimator_
        y_pred = knn_best.predict(X_test)
        print(“Confusion Matrix\n”,confusion_matrix(y_test,y_pred))
        print(“\n”)
        print(“Classification Report\n”,classification_report(y_test,y_pred))
        print(“\n”)
        print(“Accuracy : ”,accuracy_score(y_test,y_pred)*100)
```



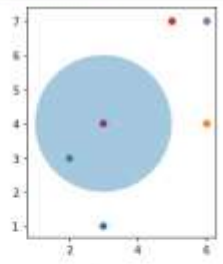
```
In [6]: In this phase of experiment, it has approx 96.67 test accuracy and a approx 97.13 CV accuracy.

        Model        CV Accuracy        Test Accuracy        F1-Score

        Linear SVM                    96.72193877551021%                    96.19047619047619%                    0.96

        Linear Regression              83.04080172481487%                    83.15260747045592%                    0.97

        Logistic Regression            96.72193877551021%                    96.66666666666667%                    0.97

        KNN                             96.49122807017544%                        96.49122807017544%

        The results have shown that even with 0 values as missing values, the KNN had the highest CV accuracy and test accuracy.

          File "<ipython-input-6-2ad35b885e93>", line 1
```

NAÏVE BAYES CLASSIFIER:



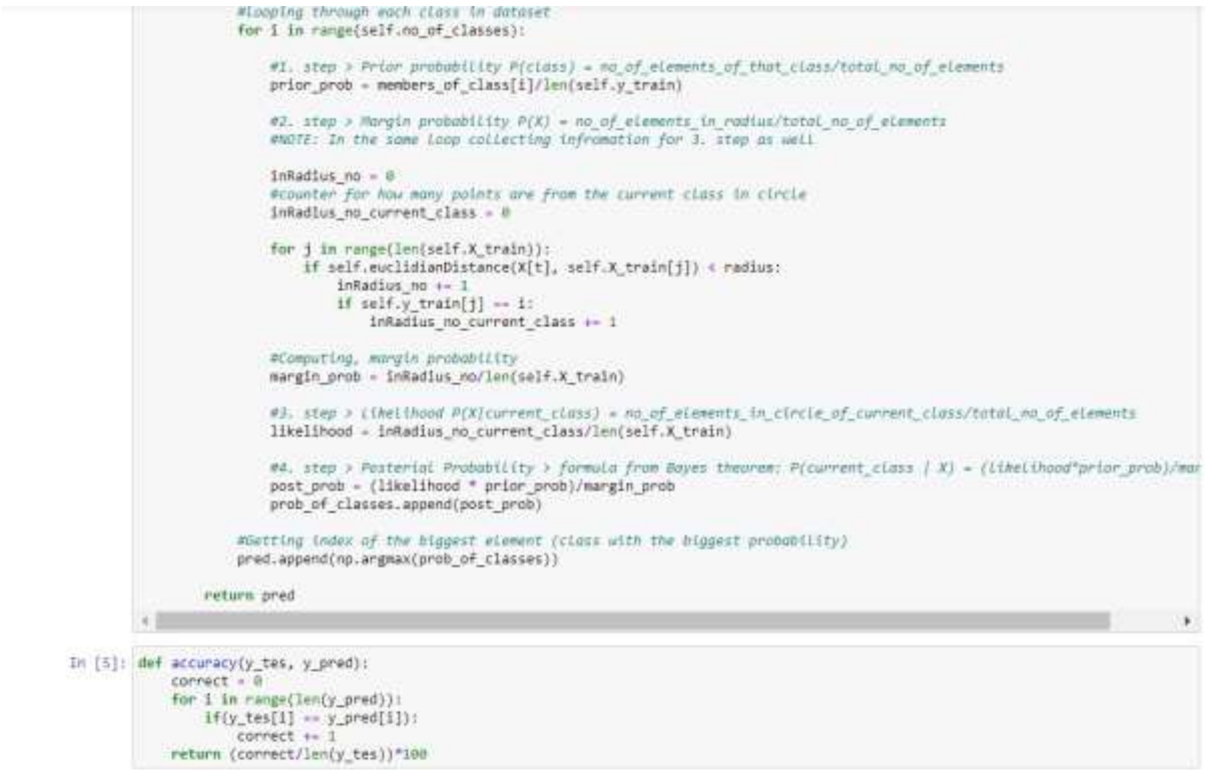
```
In [1]: #importing libs
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
```

```
In [3]: X = np.array([[1, 3],[4, 6],[3, 2],[7, 5],[7, 6]])
        for i in range(len(X)):
            plt.scatter(X[i][1], X[i][0])

        plt.scatter(3, 4, color='red')
        circle = plt.Circle((3, 4), radius=2, alpha=0.4)
        plt.gca().add_patch(circle)
        plt.axis('scaled')
        plt.show()

        euclidianDis = np.sqrt((3-2)**2 + (4-3)**2)
        print(euclidianDis)

        1.41421356237
```



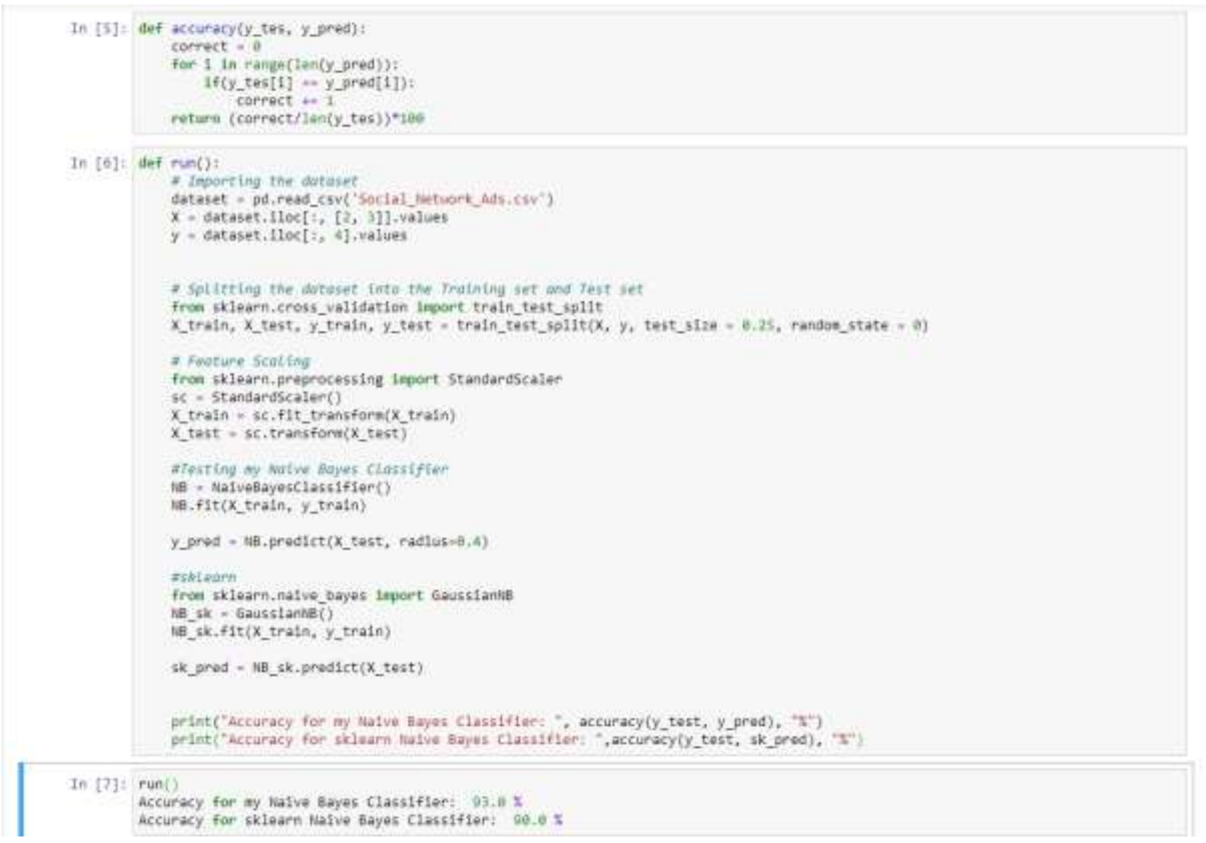
```
        1.4142135623730951

Out[3]: 1.41421356237
```

```
In [4]: class NaiveBayesClassifier(object):

            def __init__(self):
                pass

            #Input: X - features of a trainset
            #       y - labels of a trainset
            def fit(self, X, y):
                self.X_train = X
                self.y_train = y

                self.no_of_classes = np.max(self.y_train) + 1


            #This is our function to calculate all nodes/samples in our radius
            def euclidianDistance(self, Xtest, Xtrain):
                return np.sqrt(np.sum(np.power((Xtest - Xtrain), 2)))


            #our main function is predict
            #All calculation is done by using our test or new samples
            #There are 4 steps to be performed:
            # 1. calculate Prior probability. Ex. P(A) = No_of_elements_of_one_class / total_no_of_samples
            # 2. calculate Margin probability P(X) = No_of_elements_in_radius / total_no_of_samples
            # 3. calculate Likelihood (P(X|A) = No_of_elements_of_current_class / total_no_of_samples
            # 4. calculate Posterior probability: P(A|X) = (P(X|A) * P(A)) / P(X)
            # NOTE: Do these steps for all clases in dataset!
            #
            #Inputs: X - test dataset
            #        radius - this parameter is how big circle is going to be around our new datapoint, default = 2
            def predict(self, X, radius=0.4):
                pred = []

                #Creating list of numbers of elements for each class in trainset
                members_of_class = []
                for i in range(self.no_of_classes):
                    counter = 0
                    for j in range(len(self.y_train)):
                        if self.y_train[j] == i:
                            counter += 1
                    members_of_class.append(counter)

                #Entering the process of prediction
                for t in range(len(X)):
                    #Creating empty list for every class probability
                    prob_of_classes = []
```



```
        #Looping through each class in dataset
        for i in range(self.no_of_classes):

            #1. step > Prior probability P(class) = no_of_elements_of_that_class/total_no_of_elements
            prior_prob = members_of_class[i]/len(self.y_train)

            #2. step > Margin probability P(X) = no_of_elements_in_radius/total_no_of_elements
            #NOTE: In the same loop collecting infromation for 3. step as well

            inRadius_no = 0
            #counter for how many points are from the current class in circle
            inRadius_no_current_class = 0

            for j in range(len(self.X_train)):
                if self.euclidianDistance(X[t], self.X_train[j]) < radius:
                    inRadius_no += 1
                    if self.y_train[j] == i:
                        inRadius_no_current_class += 1

            #Computing, margin probability
            margin_prob = inRadius_no/len(self.X_train)

            #3. step > likelihood P(X|current_class) = no_of_elements_in_circle_of_current_class/total_no_of_elements
            likelihood = inRadius_no_current_class/len(self.X_train)

            #4. step > Posterial Probability > formula from Bayes theorem: P(current_class | X) = (likelihood*prior_prob)/mar
            post_prob = (likelihood * prior_prob)/margin_prob
            prob_of_classes.append(post_prob)

        #Getting index of the biggest element (class with the biggest probability)
        pred.append(np.argmax(prob_of_classes))

    return pred
```

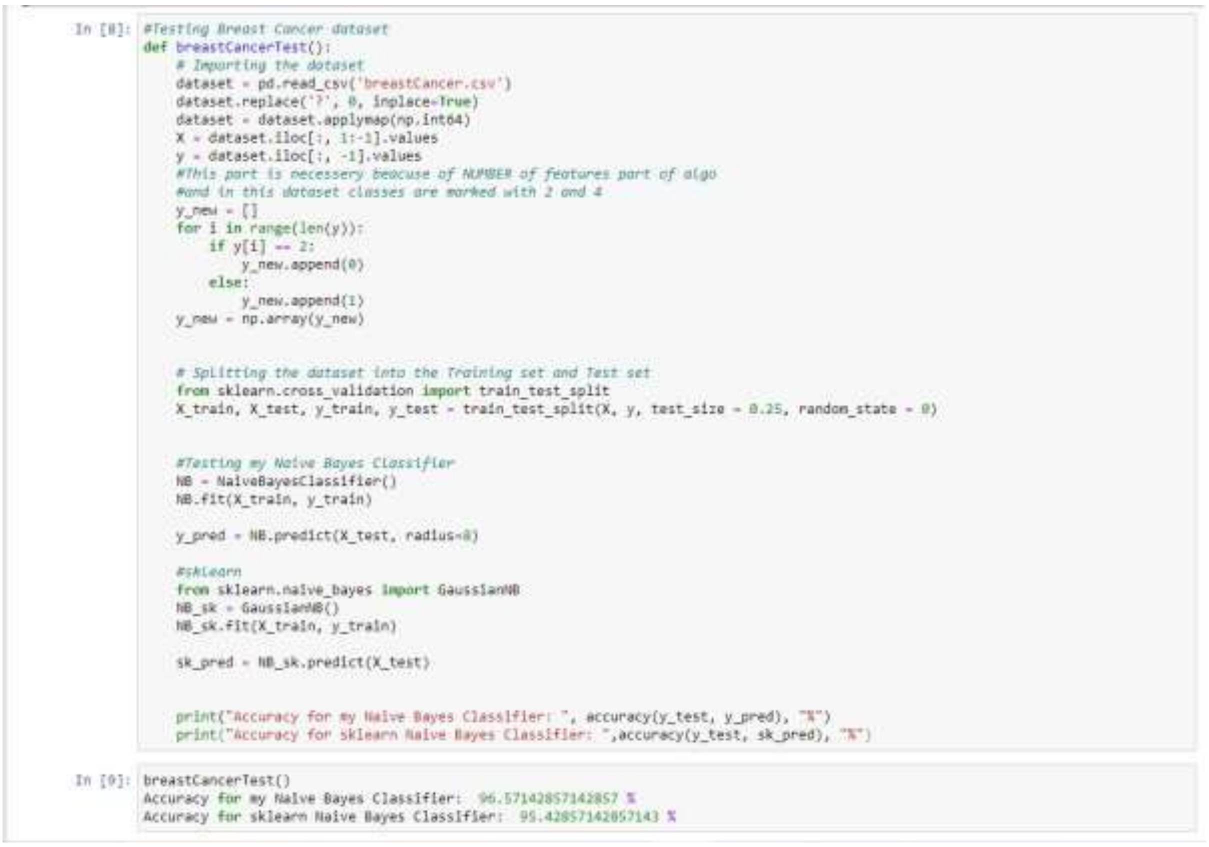
```
In [5]: def accuracy(y_tes, y_pred):
            correct = 0
            for i in range(len(y_pred)):
                if(y_tes[i] == y_pred[i]):
                    correct += 1
            return (correct/len(y_tes))*100
```



```
In [5]: def accuracy(y_tes, y_pred):
            correct = 0
            for i in range(len(y_pred)):
                if(y_tes[i] == y_pred[i]):
                    correct += 1
            return (correct/len(y_tes))*100
```

```
In [6]: def run():
            # Importing the dataset
            dataset = pd.read_csv('Social_Network_Ads.csv')
            X = dataset.iloc[:, [2, 3]].values
            y = dataset.iloc[:, 4].values


            # Splitting the dataset into the Training set and Test set
            from sklearn.cross_validation import train_test_split
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

            # Feature Scaling
            from sklearn.preprocessing import StandardScaler
            sc = StandardScaler()
            X_train = sc.fit_transform(X_train)
            X_test = sc.transform(X_test)

            #Testing my Naive Bayes Classifier
            NB = NaiveBayesClassifier()
            NB.fit(X_train, y_train)

            y_pred = NB.predict(X_test, radius=0.4)

            #sklearn
            from sklearn.naive_bayes import GaussianNB
            NB_sk = GaussianNB()
            NB_sk.fit(X_train, y_train)

            sk_pred = NB_sk.predict(X_test)


            print("Accuracy for my Naive Bayes Classifier: ", accuracy(y_test, y_pred), "%")
            print("Accuracy for sklearn Naive Bayes Classifier: ",accuracy(y_test, sk_pred), "%")
```

```
In [7]: run()
        Accuracy for my Naive Bayes Classifier:  93.0 %
        Accuracy for sklearn Naive Bayes Classifier:  90.0 %
```



```
In [8]: #Testing Breast Cancer dataset
        def breastCancerTest():
            # Importing the dataset
            dataset = pd.read_csv('breastCancer.csv')
            dataset.replace('?', 0, inplace=True)
            dataset = dataset.applymap(np.int64)
            X = dataset.iloc[:, 1:-1].values
            y = dataset.iloc[:, -1].values
            #This part is necessery beacuse of NUMBER of features part of algo
            #and in this dataset classes are marked with 2 and 4
            y_new = []
            for i in range(len(y)):
                if y[i] == 2:
                    y_new.append(0)
                else:
                    y_new.append(1)
            y_new = np.array(y_new)


            # Splitting the dataset into the Training set and Test set
            from sklearn.cross_validation import train_test_split
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)


            #Testing my Naive Bayes Classifier
            NB = NaiveBayesClassifier()
            NB.fit(X_train, y_train)

            y_pred = NB.predict(X_test, radius=8)

            #sklearn
            from sklearn.naive_bayes import GaussianNB
            NB_sk = GaussianNB()
            NB_sk.fit(X_train, y_train)

            sk_pred = NB_sk.predict(X_test)


            print("Accuracy for my Naive Bayes Classifier: ", accuracy(y_test, y_pred), "%")
            print("Accuracy for sklearn Naive Bayes Classifier: ",accuracy(y_test, sk_pred), "%")

In [9]: breastCancerTest()
        Accuracy for my Naive Bayes Classifier:  96.57142857142857 %
        Accuracy for sklearn Naive Bayes Classifier:  95.42857142857143 %
```

Comparing the accuracy of all the models:

Models	Accuracy
Logistic Regression	96.70%
Decision Tree	92.19%
K-NN	96.49%
Naïve bayes	95.42%

Table 9: Accuracy of all the models

From the table we can see that Logistic Regression is having the highest accuracy for the given dataset i.e., Breast cancer dataset.

So, for the breast cancer prediction we can go with the Logistic regression model for best predicted results.

Future Scope of Improvements

- Many hospitals, clinics can use this model and reshape it according to their need to detect the cancer. This will reduce the man power and time efficiently.
- You perform clinical tests, either at a clinic or at home. Data is inputted into the ML system. A few minutes later, you will know about your cancer.
- By adding more data, we can get a lot more detailed report about the cancer. We can also use this model to predict many more diseases by modifying the code according to the requirement.
- By further developing we can give the trained models a shape like making application or web application so that people can use it anywhere , easily without any need to understand what's going behind the scenes and have a smooth experience.

Certificate

*This is to certify that **Ms. Pragati Kundu** of **Asansol Engineering College**, Registration number: **181080110288**, has successfully completed a project on "**Prediction of Breast Cancer**" using **Machine Learning With Python** under the guidance of **Prof. Arnab Chakraborty**.*

[Name of your faculty]

Asansol Engineering College

Certificate

This is to certify that ***Ms. Moitreyee Bose*** *of* ***Asansol Engineering College****,* *Registration number:* ***181080110273****, has successfully completed a project on “****Prediction of Breast Cancer****” using* ***Machine Learning With Python*** *under the guidance of* ***Prof. Arnab Chakraborty.***

[Name of your faculty]

Asansol Engineering College

Certificate

This is to certify that ***Ms. Payal Jaiswal*** *of* ***Asansol Engineering College,*** *Registration number:* ***181080110286****, has successfully completed a project on "****Prediction of Breast Cancer****" using* ***Machine Learning With Python*** *under the guidance of* ***Prof. Arnab Chakraborty.***

[Name of your faculty]

Asansol Engineering College

Certificate

This is to certify that ***Ms. Satakshi Pandey*** of *Asansol Engineering College*, Registration number: 1**81080110306**, has successfully completed a project on ***"Prediction of Breast Cancer***" using **Machine Learning With Python** under the guidance of **Prof. Arnab Chakraborty**.

[Name of your faculty]

Asansol Engineering College

Certificate

This is to certify that **Ms. Mousumi Roy** of **Asansol Engineering College**, Registration number: 1**81080110276**, has successfully completed a project on "**Prediction of Breast Cancer**" using **Machine Learning With Python** under the guidance of **Prof. Arnab Chakraborty**.

[Name of your faculty]

Asansol Engineering College