Breast Cancer Classification using (KNN)

Hello friends,

kNN or k-Nearest Neighbours Classifier is a very simple and easy to understand machine learning algorithm. In this kernel, I build a k Nearest Neighbours classifier to classify the patients suffering from Breast Cancer.

SO, let get started.

Import library

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Import Dataset

In [2]:
```
data = pd.read_csv(r'E:\NIT(Data Science)by (prakash senapati sir (kodi))\Data science all
```

In [3]:
```
data.head(5)
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10
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

Exploratory Data Analysis

Now i will explore the data to gain insights about the data

View dimensions of dataset

In [4]:
```
data.shape
```

Out[4]:
```
(699, 11)
```

In [5]:
```
data.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	10
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

Rename column names

In [6]:
```
col_names = ['Id', 'Clump_thickness', 'Uniformity_Cell_Size', 'Uniformity_Cell_Shape', 'Ma
             'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucl

data.columns = col_names

data.columns
```

Out[6]:
```
Index(['Id', 'Clump_thickness', 'Uniformity_Cell_Size',
       'Uniformity_Cell_Shape', 'Marginal_Adhesion',
       'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin',
       'Normal_Nucleoli', 'Mitoses', 'Class'],
      dtype='object')
```

In [7]:
```
data.head()
```

Out[7]:

	Id	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_
0	1000025	5	1	1	1	
1	1002945	5	4	4	5	
2	1015425	3	1	1	1	
3	1016277	6	8	8	1	
4	1017023	4	1	1	3	

Drop Redundant columns

In [8]:
```
data.drop("Id", axis=1, inplace=True)
```

View summary of dataset

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   Clump_thickness              699 non-null    int64 
 1   Uniformity_Cell_Size         699 non-null    int64 
 2   Uniformity_Cell_Shape        699 non-null    int64 
 3   Marginal_Adhesion            699 non-null    int64 
 4   Single_Epithelial_Cell_Size  699 non-null    int64 
 5   Bare_Nuclei                  699 non-null    object
 6   Bland_Chromatin              699 non-null    int64 
 7   Normal_Nucleoli              699 non-null    int64 
 8   Mitoses                      699 non-null    int64 
 9   Class                        699 non-null    int64 
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

Frequency Distribution of value in variables

In [10]:
```
for var in data.columns:
    print(data[var].value_counts())
```

```
1     145
5     130
3     108
4      80
10     69
2      50
8      46
6      34
7      23
9      14
Name: Clump_thickness, dtype: int64
1     384
10     67
3      52
2      45
4      40
5      30
8      29
6      27
7      19
9       6
Name: Uniformity_Cell_Size, dtype: int64
1     353
2      59
10     58
3      56
4      44
5      34
6      30
7      30
8      28
9       7
Name: Uniformity_Cell_Shape, dtype: int64
1     407
3      58
2      58
10     55
4      33
8      25
5      23
6      22
7      13
9       5
Name: Marginal_Adhesion, dtype: int64
2     386
3      72
4      48
1      47
6      41
5      39
10     31
8      21
7      12
9       2
Name: Single_Epithelial_Cell_Size, dtype: int64
1     402
10    132
2      30
5      30
3      28
8      21
4      19
?      16
9       9
7       8
```

```
6       4
Name: Bare_Nuclei, dtype: int64
2     166
3     165
1     152
7      73
4      40
5      34
8      28
10     20
9      11
6      10
Name: Bland_Chromatin, dtype: int64
1     443
10     61
3      44
2      36
8      24
6      22
5      19
4      18
7      16
9      16
Name: Normal_Nucleoli, dtype: int64
1     579
2      35
3      33
10     14
4      12
7       9
8       8
5       6
6       3
Name: Mitoses, dtype: int64
2    458
4    241
Name: Class, dtype: int64
```

Convert Data type of Bare_Nuclei to integer

In [11]:
```
data["Bare_Nuclei"] = pd.to_numeric(data["Bare_Nuclei"], errors="coerce")
```

Check data type of columns of DataFrame

In [12]:
```
data.dtypes
```

Out[12]:
```
Clump_thickness                int64
Uniformity_Cell_Size           int64
Uniformity_Cell_Shape          int64
Marginal_Adhesion              int64
Single_Epithelial_Cell_Size    int64
Bare_Nuclei                  float64
Bland_Chromatin                int64
Normal_Nucleoli                int64
Mitoses                        int64
Class                          int64
dtype: object
```

Missing Values in Variables

Check Missing values in variables

In [13]:
```
data.isnull().sum()
```

Out[13]:
```
Clump_thickness               0
Uniformity_Cell_Size          0
Uniformity_Cell_Shape         0
Marginal_Adhesion             0
Single_Epithelial_Cell_Size   0
Bare_Nuclei                  16
Bland_Chromatin               0
Normal_Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

check 'na' values in the dataFrame

In [14]:
```
data.isna().sum()
```

Out[14]:
```
Clump_thickness               0
Uniformity_Cell_Size          0
Uniformity_Cell_Shape         0
Marginal_Adhesion             0
Single_Epithelial_Cell_Size   0
Bare_Nuclei                  16
Bland_Chromatin               0
Normal_Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

check frequency distribution of Bare_Nuclei column

In [15]:
```
data["Bare_Nuclei"].value_counts()
```

Out[15]:
```
1.0     402
10.0    132
2.0      30
5.0      30
3.0      28
8.0      21
4.0      19
9.0       9
7.0       8
6.0       4
Name: Bare_Nuclei, dtype: int64
```

check unique value in "Bare_Nuclei" columns

In [16]:
```
data["Bare_Nuclei"].unique()
```

Out[16]:
```
array([ 1., 10.,  2.,  4.,  3.,  9.,  7., nan,  5.,  8.,  6.])
```

We can see that there are `nan` values in the `Bare_Nuclei` column.

check for nan values in bare_nuclei column

In [17]:
```
data["Bare_Nuclei"].isna().sum()
```

Out[17]:
```
16
```

We can see that there are 16 `nan` values in the dataset. I will impute missing values after dividing the dataset into training and test set.

Check frequency distribution of target variable class

VALUE FREQUENCY DISTRIBUTION OF VALUES IN CLASS VARIABLE

In [18]:
```
data["Class"].value_counts()
```

Out[18]:
```
2    458
4    241
Name: Class, dtype: int64
```

Check parcentage of frequency distribution of Class

In [19]:
```
data["Class"].value_counts()/np.float(len(data))
```

Out[19]:
```
2    0.655222
4    0.344778
Name: Class, dtype: float64
```

We can see that the `Class` variable contains 2 class labels - `2` and `4`. `2` stands for benign and `4` stands for malignant cancer.

Outliers in numerical Variable

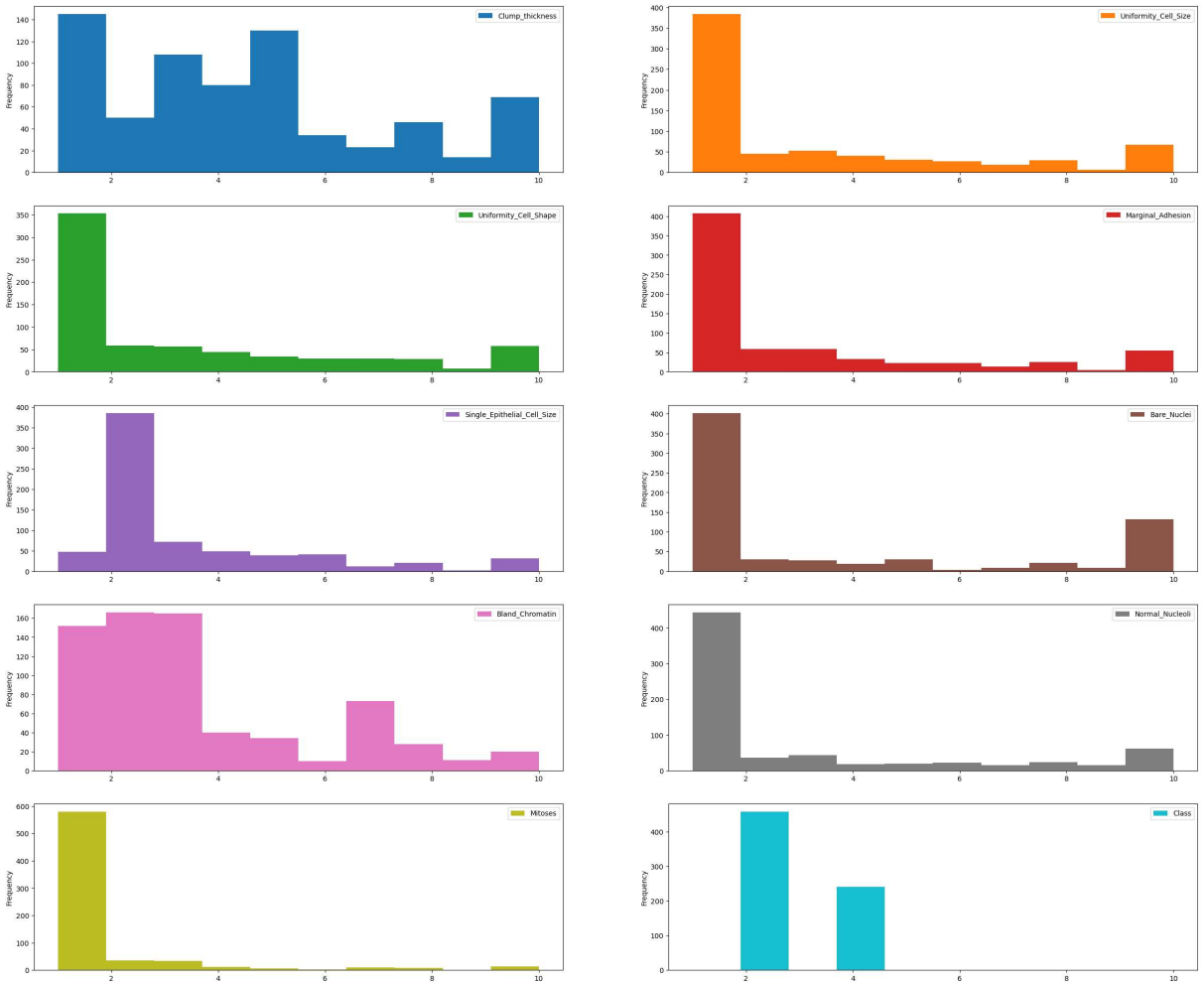
In [20]:
```
# view summary statistics in numerical variables

print(round(data.describe(),2))
```

```
       Clump_thickness  Uniformity_Cell_Size  Uniformity_Cell_Shape  \
count           699.00                699.00                 699.00
mean              4.42                  3.13                   3.21
std               2.82                  3.05                   2.97
min               1.00                  1.00                   1.00
25%               2.00                  1.00                   1.00
50%               4.00                  1.00                   1.00
75%               6.00                  5.00                   5.00
max              10.00                 10.00                  10.00

       Marginal_Adhesion  Single_Epithelial_Cell_Size  Bare_Nuclei  \
count             699.00                       699.00       683.00
mean                2.81                         3.22         3.54
std                 2.86                         2.21         3.64
min                 1.00                         1.00         1.00
25%                 1.00                         2.00         1.00
50%                 1.00                         2.00         1.00
75%                 4.00                         4.00         6.00
max                10.00                        10.00        10.00

       Bland_Chromatin  Normal_Nucleoli  Mitoses   Class
count           699.00           699.00   699.00  699.00
mean              3.44             2.87     1.59    2.69
std               2.44             3.05     1.72    0.95
min               1.00             1.00     1.00    2.00
25%               2.00             1.00     1.00    2.00
50%               3.00             1.00     1.00    2.00
75%               5.00             4.00     1.00    4.00
max              10.00            10.00    10.00    4.00
```

Data Visualization

In [21]:
```
# Check the distribution of variables
```

In [22]:
```
# plot histograms of the variables

plt.rcParams['figure.figsize']=(30,25)

data.plot(kind='hist', bins=10, subplots=True, layout=(5,2), sharex=False, sharey=False)

plt.show()
```



Multivariate plot

Estimating correlation coefficients

In [23]:
```
correlation = data.corr()
```

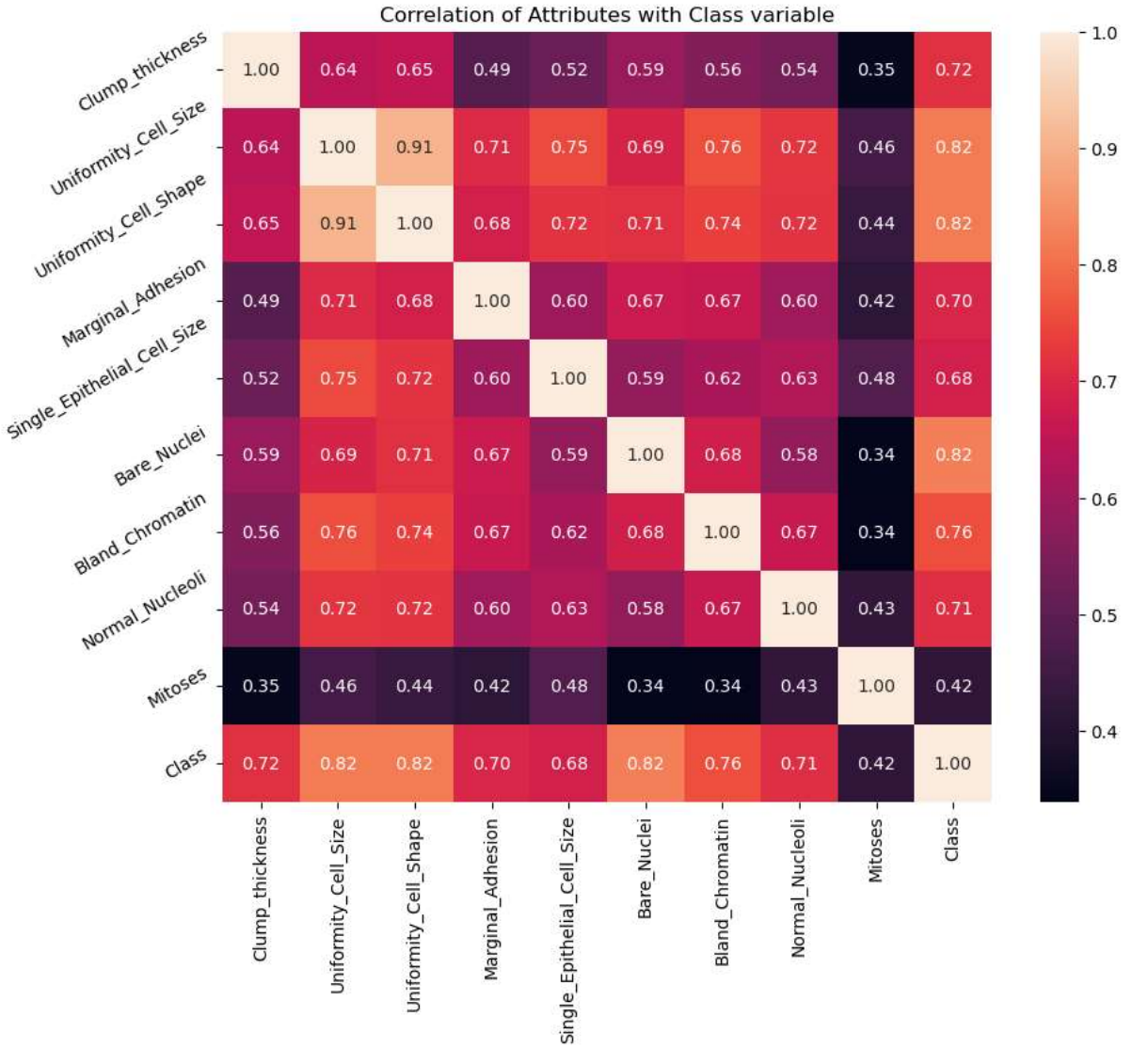
Our target variable is `Class` . So, we should check how each attribute correlates with the `Class` variable. We can do it as follows:-

In [24]:
```
correlation['Class'].sort_values(ascending=False)
```

Out[24]:
```
Class                          1.000000
Bare_Nuclei                    0.822696
Uniformity_Cell_Shape          0.818934
Uniformity_Cell_Size           0.817904
Bland_Chromatin                0.756616
Clump_thickness                0.716001
Normal_Nucleoli                0.712244
Marginal_Adhesion              0.696800
Single_Epithelial_Cell_Size    0.682785
Mitoses                        0.423170
Name: Class, dtype: float64
```

Correlation Heat map

In [25]:
```
plt.figure(figsize=(10,8))
plt.title('Correlation of Attributes with Class variable')
a = sns.heatmap(correlation, square=True, annot=True, fmt='.2f', linecolor='white')
a.set_xticklabels(a.get_xticklabels(), rotation=90)
a.set_yticklabels(a.get_yticklabels(), rotation=30)
plt.show()
```



Interpretation

From the above correlation heat map, we can conclude that :-

1. `Class` is highly positive correlated with `Uniformity_Cell_Size` , `Uniformity_Cell_Shape` and `Bare_Nuclei` . (correlation coefficient = 0.82).
2. `Class` is positively correlated with `Clump_thickness` (correlation coefficient=0.72), `Marginal_Adhesion` (correlation coefficient=0.70), `Single_Epithelial_Cell_Size)` (correlation coefficient = 0.68) and `Normal_Nucleoli` (correlation coefficient=0.71).
3. `Class` is weekly positive correlated with `Mitoses` (correlation coefficient=0.42).
4. The `Mitoses` variable is weekly positive correlated with all the other variables(correlation coefficient < 0.50).

Declare feature vactor and Target

In [26]:
```
x = data.drop(["Class"], axis=1)

y = data["Class"]
```

Split data into separate training and test set

In [27]:
```
# split x and y into training and testing sets

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 =
```

In [28]:
```
# check the shape of x_train and x_test

x_train.shape, x_test.shape
```

Out[28]: ((559, 9), (140, 9))

Feature Engineering

Feature Engineering is the process of transforming raw data into useful features that help us to understand our model better and increase its predictive power. I will carry out feature engineering on different types of variables.

In [29]:
```
# check data type in x_train

x_train.dtypes
```

Out[29]:
```
Clump_thickness                int64
Uniformity_Cell_Size           int64
Uniformity_Cell_Shape          int64
Marginal_Adhesion              int64
Single_Epithelial_Cell_Size    int64
Bare_Nuclei                  float64
Bland_Chromatin                int64
Normal_Nucleoli                int64
Mitoses                        int64
dtype: object
```

Engineering missing values in variables

In [30]:
```
# check missing value in numerical variables in x_train

x_train.isnull().sum()
```

Out[30]:
```
Clump_thickness                0
Uniformity_Cell_Size           0
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                   13
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
dtype: int64
```

In [31]:
```
# check missing value in numerical variables in x_test

x_test.isnull().sum()
```

Out[31]:
```
Clump_thickness                0
Uniformity_Cell_Size           0
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                    3
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
dtype: int64
```

In [32]:
```
# print percentage of missing values in the numerical variablein training set

for col in x_train.columns:
    if x_train[col].isnull().mean()>0:
        print(col, round(x_train[col].isnull().mean(),4))
```

```
Bare_Nuclei 0.0233
```

In [33]:
```
# impute missing values in x_train and x_test with respective columnmedian in x_train
for df1 in [x_train, x_test]:
    for col in x_train.columns:
        col_median=x_train[col].median()
        df1[col].fillna(col_median, inplace=True)
```

In [34]:
```
x_train.isnull().sum()
```

Out[34]:
```
Clump_thickness               0
Uniformity_Cell_Size          0
Uniformity_Cell_Shape         0
Marginal_Adhesion             0
Single_Epithelial_Cell_Size   0
Bare_Nuclei                   0
Bland_Chromatin               0
Normal_Nucleoli               0
Mitoses                       0
dtype: int64
```

In [35]:
```
x_test.isnull().sum()
```

Out[35]:
```
Clump_thickness               0
Uniformity_Cell_Size          0
Uniformity_Cell_Shape         0
Marginal_Adhesion             0
Single_Epithelial_Cell_Size   0
Bare_Nuclei                   0
Bland_Chromatin               0
Normal_Nucleoli               0
Mitoses                       0
dtype: int64
```

we can see that there is no missing values in numerical variable in x_train, x_test

In [36]:
```
x_train.head()
```

Out[36]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Siz
293	10	4	4	6	
62	9	10	10	1	1
485	1	1	1	3	
422	4	3	3	1	
332	5	2	2	2	

In [37]:
```
x_test.head()
```

Out[37]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Siz
476	4	1	2	1	
531	4	2	2	1	
40	6	6	6	9	
432	5	1	1	1	
14	8	7	5	10	

now i have training and testing set ready for model building. Before that, we should map all the feature variables onto the same scale. It is called `feature scaling` . I will do it as follows.

Feature Scaling

In [38]:
```
cols = x_train.columns
```

In [39]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [40]:
```
x_train = pd.DataFrame(x_train, columns=[cols])
```

In [41]:
```
x_test = pd.DataFrame(x_test, columns=[cols])
```

In [42]:
```
x_train.head()
```

Out[42]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size
0	2.028383	0.299506	0.289573	1.119077	-0.546543
1	1.669451	2.257680	2.304569	-0.622471	3.106879
2	-1.202005	-0.679581	-0.717925	0.074148	-1.003220
3	-0.125209	-0.026856	-0.046260	-0.622471	-0.546543
4	0.233723	-0.353219	-0.382092	-0.274161	-0.546543

We now have `X_train` dataset ready to be fed into the Logistic Regression classifier. I will do it as follows.

Fit K Neighbors classifier to the training set

In [43]:
```
# import KNeighbors Classification from sklearn
from sklearn.neighbors import KNeighborsClassifier
```

In [44]:
```
# iinstantiate the model
knn = KNeighborsClassifier(n_neighbors=3)
```

In [45]:
```
# fit the model to the training set
knn.fit(x_train, y_train)
```

Out[45]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Predict test-set results

In [46]:
```
y_pred = knn.predict(x_test)
y_pred
```

Out[46]:
```
array([2, 2, 4, 2, 4, 2, 4, 2, 4, 2, 2, 2, 4, 4, 4, 2, 2, 4, 4, 2, 4, 4,
       2, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2,
       4, 4, 2, 4, 2, 4, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 4,
       4, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2,
       4, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 4, 2, 2, 2, 4, 2, 2, 2, 2, 2,
       4, 4, 4, 2, 2, 2, 2, 2, 4, 4, 4, 4, 2, 4, 2, 2, 4, 4, 4, 4, 4, 2,
       2, 4, 4, 2, 2, 4, 2, 2], dtype=int64)
```

predict_proba method

predict_proba method gives the probabilities for the target variable(2 and 4) in this case, in array form.

`2 is for probability of benign cancer and 4 is for probability of malignant cancer.`

In [47]:
```
# probablity of getting out put as 2 - benign cancer

knn.predict_proba(x_test)[:,0]
```

Out[47]:
```
array([1.        , 1.        , 0.33333333, 1.        , 0.        ,
       1.        , 0.        , 1.        , 0.        , 0.66666667,
       1.        , 1.        , 0.        , 0.33333333, 0.        ,
       1.        , 1.        , 0.        , 0.        , 1.        ,
       0.        , 0.        , 1.        , 1.        , 1.        ,
       0.        , 1.        , 1.        , 0.        , 0.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.66666667, 1.        , 0.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.        ,
       0.        , 1.        , 0.        , 1.        , 0.        ,
       0.        , 1.        , 1.        , 0.        , 1.        ,
       1.        , 1.        , 1.        , 0.66666667, 1.        ,
       0.        , 1.        , 1.        , 0.        , 0.        ,
       0.33333333, 0.        , 1.        , 1.        , 0.        ,
       1.        , 1.        , 0.        , 0.        , 1.        ,
       1.        , 1.        , 1.        , 0.        , 1.        ,
       1.        , 1.        , 0.        , 1.        , 1.        ,
       1.        , 0.        , 1.        , 0.        , 0.        ,
       1.        , 1.        , 0.66666667, 0.        , 1.        ,
       1.        , 1.        , 0.        , 1.        , 0.        ,
       0.        , 1.        , 1.        , 1.        , 0.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.        , 0.33333333, 0.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 0.        , 0.        ,
       0.        , 0.33333333, 1.        , 0.        , 1.        ,
       1.        , 0.33333333, 0.33333333, 0.        , 0.        ,
       0.        , 1.        , 1.        , 0.33333333, 0.        ,
       1.        , 1.        , 0.        , 1.        , 1.        ])
```

In [48]:
```
# # probablity of getting out put as 4 - benign cancer

knn.predict_proba(x_test)[:,1]
```

Out[48]:
```
array([0.        , 0.        , 0.66666667, 0.        , 1.        ,
       0.        , 1.        , 0.        , 1.        , 0.33333333,
       0.        , 0.        , 1.        , 0.66666667, 1.        ,
       0.        , 0.        , 1.        , 1.        , 0.        ,
       1.        , 1.        , 0.        , 0.        , 0.        ,
       1.        , 0.        , 0.        , 1.        , 1.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.33333333, 0.        , 1.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 1.        ,
       1.        , 0.        , 1.        , 0.        , 1.        ,
       1.        , 0.        , 0.        , 1.        , 0.        ,
       0.        , 0.        , 0.        , 0.33333333, 0.        ,
       1.        , 0.        , 0.        , 1.        , 1.        ,
       0.66666667, 1.        , 0.        , 0.        , 1.        ,
       0.        , 0.        , 1.        , 1.        , 0.        ,
       0.        , 0.        , 0.        , 1.        , 0.        ,
       0.        , 0.        , 1.        , 0.        , 0.        ,
       0.        , 1.        , 0.        , 1.        , 1.        ,
       0.        , 0.        , 0.33333333, 1.        , 0.        ,
       0.        , 0.        , 1.        , 0.        , 1.        ,
       1.        , 0.        , 0.        , 0.        , 1.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       1.        , 0.66666667, 1.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 1.        , 1.        ,
       1.        , 0.66666667, 0.        , 1.        , 0.        ,
       0.        , 0.66666667, 0.66666667, 1.        , 1.        ,
       1.        , 0.        , 0.        , 0.66666667, 1.        ,
       0.        , 0.        , 1.        , 0.        , 0.        ])
```

check accuracy score

In [49]:
```
from sklearn.metrics import accuracy_score
print("Model accuracy score: {0:0.4f}". format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.9714
```

Compare the train-set and test-set accuracy

Now, I will compare the train-set and test-set accuracy to check for overfitting.

In [50]:
```
y_pred_train = knn.predict(x_train)
```

In [51]:
```
print("Training-set accuracy score: {0:0.4f}". format(accuracy_score(y_train, y_pred_train
```

```
Training-set accuracy score: 0.9821
```

Check the overfitting and underfitting

In [52]:
```
# print the score on training and test set
print('Training set score: {:.4f}'.format(knn.score(x_train, y_train)))

print('Test set score: {:.4f}'.format(knn.score(x_test, y_test)))
```

```
Training set score: 0.9821
Test set score: 0.9714
```

Compare model accuracy with null accuracy

In [53]:
```
# check class distribution in test set

y_test.value_counts()
```

Out[53]:
```
2    85
4    55
Name: Class, dtype: int64
```

We can see that the occurences of most frequent class is 85. So, we can calculate null accuracy by dividing 85 by total number of occurences.

In [54]:
```
# check null accuracy score

null_accuracy = (85/(85+55))
print("Null accuracy score: {0:0.4f}". format(null_accuracy))
```

```
Null accuracy score: 0.6071
```

We can see that our model accuracy score is 0.9714 but null accuracy score is 0.6071. So, we can conclude that our K Nearest Neighbors model is doing a very good job in predicting the class labels.

Rebuild KNN classification model using different values of k

i have build the kNN classification model using k=3. Now, I will increase the value of k and see its effect on accuracy.

Knn classification model using K=5

In [55]:
```
# instantiate  the model with k=5
knn_5 = KNeighborsClassifier(n_neighbors=5)
```

In [56]:
```
# fit the model to the training set
knn_5.fit(x_train, y_train)
```

Out[56]:
```
KNeighborsClassifier

KNeighborsClassifier()
```

In [57]:
```
# predict on the test-set
y_pred_5 = knn_5.predict(x_test)

print("Model accuracy score with k=5 : {0:0.4f}". format(accuracy_score(y_test, y_pred_5))
```

```
Model accuracy score with k=5 : 0.9714
```

Rebuild KNN classification model using k=6

In [58]:
```
# instantiate  the model with k=6
knn_6 = KNeighborsClassifier(n_neighbors=6)
```

In [59]:
```
# fit the model to the training set
knn_6.fit(x_train, y_train)
```

Out[59]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)
```

In [60]:
```
# predict on the test-set
y_pred_6 = knn_6.predict(x_test)

print("Model accuracy score with k=6 : {0:0.4f}". format(accuracy_score(y_test, y_pred_6))
```

```
Model accuracy score with k=6 : 0.9786
```

Rebuild KNN classification model using k=7

In [61]:
```
# instantiate  the model with k=7
knn_7 = KNeighborsClassifier(n_neighbors=7)
```

In [62]:
```
# fit the model to the training set
knn_7.fit(x_train, y_train)
```

Out[62]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

In [63]:
```
# predict on the test-set
y_pred_7 = knn_7.predict(x_test)

print("Model accuracy score with k=7 : {0:0.4f}". format(accuracy_score(y_test, y_pred_7))
```

```
Model accuracy score with k=7 : 0.9786
```

Rebuild KNN classification model using k=8

In [64]:
```
# instantiate  the model with k=8
knn_8 = KNeighborsClassifier(n_neighbors=8)
```

In [65]:
```
# fit the model to the training set
knn_8.fit(x_train, y_train)
```

Out[65]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=8)
```

In [66]:
```
# predict on the test-set
y_pred_8 = knn_8.predict(x_test)

print("Model accuracy score with k=8 : {0:0.4f}". format(accuracy_score(y_test, y_pred_8))
```

```
Model accuracy score with k=8 : 0.9786
```

Rebuild KNN classification model using k=8

In [67]:
```
# instantiate  the model with k=9
knn_9 = KNeighborsClassifier(n_neighbors=9)
```

In [68]:
```
# fit the model to the training set
knn_9.fit(x_train, y_train)
```

Out[68]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=9)
```

In [69]:
```
# predict on the test-set
y_pred_9 = knn_9.predict(x_test)

print("Model accuracy score with k=9 : {0:0.4f}". format(accuracy_score(y_test, y_pred_9))
```

```
Model accuracy score with k=9 : 0.9714
```

interprepretation

Now, based on the above analysis we can conclude that our classification model accuracy is very good. Our model is doing a very good job in terms of predicting the class labels.

But, it does not give the underlying distribution of values. Also, it does not tell anything about the type of errors our classifer is making.

We have another tool called `Confusion matrix` that comes to our rescue.

Confusion matrix

In [70]:
```
# Print the Confusion Matrix with k =3 and slice it into four pieces
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print("Confusion matrix\n\n", cm)

print("\nTrue Positive(TP) = ", cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[83  2]
 [ 2 53]]

True Positive(TP) =  83

True Negatives(TN) =  53

False Positives(FP) =  2

False Negatives(FN) =  2
```

The confusion matrix shows `83 + 53 = 136 correct predictions` and `2 + 2 = 4 incorrect predictions` .

In this case, we have

- `True Positives` (Actual Positive:1 and Predict Positive:1) - 83
- `True Negatives` (Actual Negative:0 and Predict Negative:0) - 53
- `False Positives` (Actual Negative:0 but Predict Positive:1) - 2 `(Type I error)`
- `False Negatives` (Actual Positive:1 but Predict Negative:0) - 2 `(Type II error)`

In [71]:
```
# # Print the Confusion Matrix with k =7 and slice it into four pieces

cm_7 = confusion_matrix(y_test, y_pred_7)

print('Confusion matrix\n\n', cm_7)

print('\nTrue Positives(TP) = ', cm_7[0,0])

print('\nTrue Negatives(TN) = ', cm_7[1,1])

print('\nFalse Positives(FP) = ', cm_7[0,1])

print('\nFalse Negatives(FN) = ', cm_7[1,0])
```

```
Confusion matrix

 [[83  2]
 [ 1 54]]

True Positives(TP) =  83

True Negatives(TN) =  54

False Positives(FP) =  2

False Negatives(FN) =  1
```

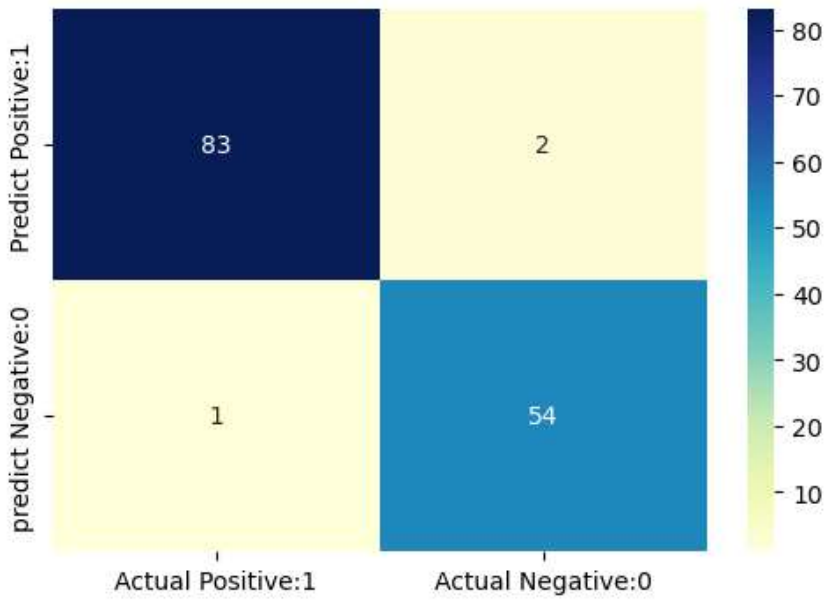
The above confusion matrix shows `83 + 54 = 137 correct predictions` and `2 + 1 = 4 incorrect predictions` .

In this case, we have

- `True Positives` (Actual Positive:1 and Predict Positive:1) - 83
- `True Negatives` (Actual Negative:0 and Predict Negative:0) - 54
- `False Positives` (Actual Negative:0 but Predict Positive:1) - 2 `(Type I error)`
- `False Negatives` (Actual Positive:1 but Predict Negative:0) - 1 `(Type II error)`

In [74]:
```
# Visualize confusion matrix with seaborn heatmap

plt.figure(figsize=(6,4))

cm_matrix = pd.DataFrame(data=cm_7, columns=["Actual Positive:1", "Actual Negative:0"],
                                 index=["Predict Positive:1", "predict Negative:0"])
sns.heatmap(cm_matrix, annot= True, fmt="d", cmap="YlGnBu")
```

Out[74]: <Axes: >



Classification metrics

Classification report is another way to evaluate the classification model performance. It displays the **precision**, **recall**, **f1** and **support** scores for the model. I have described these terms in later.

We can print a classification report as follows:-

In [76]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_7))
```

```
              precision    recall  f1-score   support

           2       0.99      0.98      0.98        85
           4       0.96      0.98      0.97        55

    accuracy                           0.98       140
   macro avg       0.98      0.98      0.98       140
weighted avg       0.98      0.98      0.98       140
```

Classification Accuracy

In [77]:
```
TP = cm_7[0,0]
TN = cm_7[1,1]
FP = cm_7[0,1]
FN = cm_7[1,0]
```

In [78]:
```
# print Classification

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
print("classification accuracy : {0:0.4f}". format(classification_accuracy))
```

```
classification accuracy : 0.9786
```

classification error

In [79]:
```
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print("classification error : {0:0.4f}".format(classification_error))
```

```
classification error : 0.0214
```

precision

In [80]:
```
# print precision score

precision = TP / float(TP + FP)

print("Precision : {0:0.4f}".format(precision))
```

```
Precision : 0.9765
```

Recall

In [81]:
```
recall = TP / float(TP + FN)
print("Recall or Sensitivity : {0:0.4f}".format(recall))
```

```
Recall or Sensitivity : 0.9881
```

True Positive Rate

True Positive Rate is synonymous with Recall

In [84]:
```
true_positive_rate = TP / float(TP + FN)

print("True Positive Rate : {0:0.4f}".format(true_positive_rate))
```

```
True Positive Rate : 0.9881
```

False positive Rate

In [85]:
```
false_positive_rate = FP / float(FP + TN)

print("False Positive Rate : {0:0.4f}".format(false_positive_rate))
```

```
False Positive Rate : 0.0357
```

Specificity

In [86]:
```
specificity = TN / (TN + FP)

print("Specificity : {0:0.4f}". format(specificity))
```

```
Specificity : 0.9643
```

Adjusting the classification threshold level

In [88]:
```
# print the first 10 prediction probablities of two classes 2 and 4

y_pred_prob = knn.predict_proba(x_test)[0:10]
y_pred_prob
```

Out[88]:
```
array([[1.        , 0.        ],
       [1.        , 0.        ],
       [0.33333333, 0.66666667],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.66666667, 0.33333333]])
```

In [90]:
```
# store the probablities in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=["Prob of - benign Cancer (2)", "P

y_pred_prob_df
```

Out[90]:

	Prob of - benign Cancer (2)	Prob of - malignant Cancer (4)
0	1.000000	0.000000
1	1.000000	0.000000
2	0.333333	0.666667
3	1.000000	0.000000
4	0.000000	1.000000
5	1.000000	0.000000
6	0.000000	1.000000
7	1.000000	0.000000
8	0.000000	1.000000
9	0.666667	0.333333

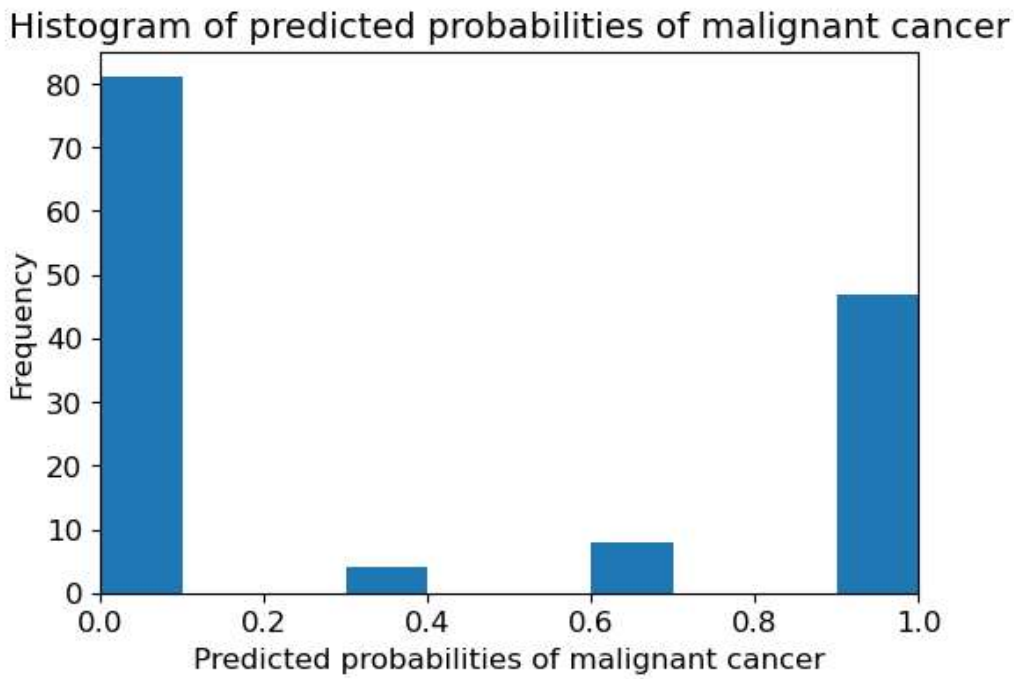
In [91]:
```
# print the first 10 pridiction probablities for class 4 - probablities of maligant cancer
knn.predict_proba(x_test)[0:10, 1]
```

Out[91]:
```
array([0.        , 0.        , 0.66666667, 0.        , 1.        ,
       0.        , 1.        , 0.        , 1.        , 0.33333333])
```

In [92]:
```
## store the predicted probabilities for class 4 - Probability of malignant cancer

y_pred_1 = knn.predict_proba(x_test)[:, 1]
```

In [94]:
```
# plot histogram of predicted probabilities
# adjust figure size

plt.figure(figsize=(6,4))

# adjust the font size
plt.rcParams["font.size"] = 12

# plot histogram with 10 bins
plt.hist(y_pred_1, bins = 10)

# set the title of prediction probablities
plt.title('Histogram of predicted probabilities of malignant cancer')


# set the x-axis limit
plt.xlim(0,1)


# set the title
plt.xlabel('Predicted probabilities of malignant cancer')
plt.ylabel('Frequency')
```

Out[94]:
```
Text(0, 0.5, 'Frequency')
```

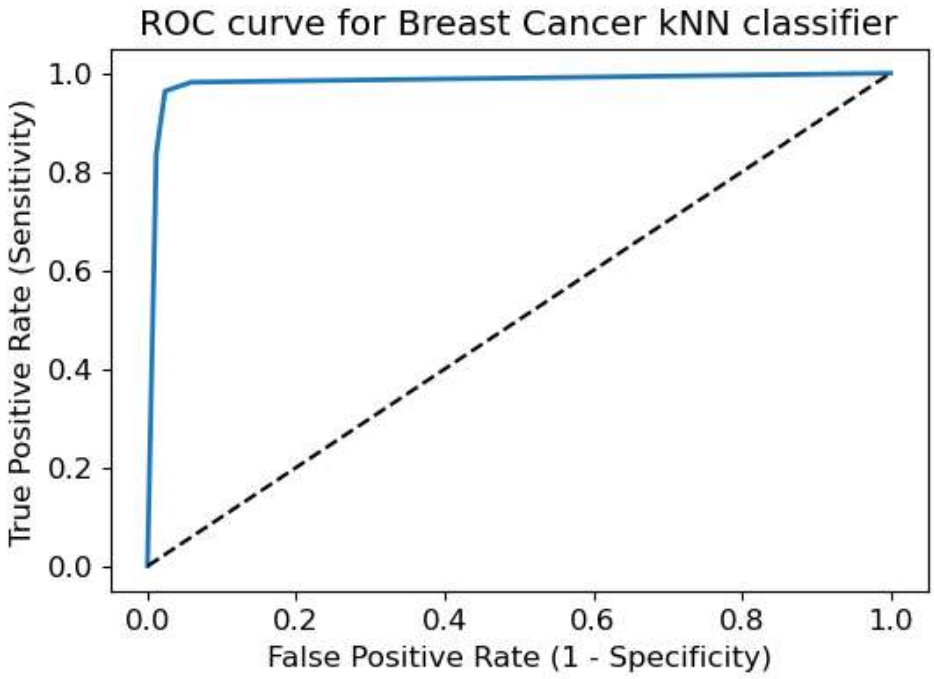


Observations

- We can see that the above histogram is positively skewed.
- The first column tell us that there are approximately 80 observations with 0 probability of malignant cancer.
- There are few observations with probability > 0.5.
- So, these few observations predict that there will be malignant cancer.

Roc-AUC

In [95]:
```
# plot ROC Curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_1, pos_label=4)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Breast Cancer kNN classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



In [96]:
```
# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred_1)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.9825
```

In [98]:
```
# calculate cross-validated ROC AUC

from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(knn_7, x_train, y_train, cv=5, scoring='roc_auc'

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.9910
```

k-fold Cross Validation

In [100]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(knn_7, x_train, y_train, cv = 10, scoring="accuracy")
print("cross-validation scores:{}".format(scores))
```

```
cross-validation scores:[0.875      0.96428571 0.94642857 0.98214286 0.96428571 0.9642857
1
 0.98214286 0.98214286 1.         0.98181818]
```

We can summarize the cross-validation accuracy by calculating its mean.

In [101]:
```
# compute Average cross-validation score

print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Average cross-validation score: 0.9643
```

Interpretation

- Using the mean cross-validation, we can conclude that we expect the model to be around 96.46 % accurate on average.
- If we look at all the 10 scores produced by the 10-fold cross-validation, we can also conclude that there is a relatively high variance in the accuracy between folds, ranging from 100% accuracy to 87.72% accuracy. So, we can conclude that the model is very dependent on the particular folds used for training, but it also be the consequence of the small size of the dataset.
- We can see that 10-fold cross-validation accuracy does not result in performance improvement for this model.

Results and Conclusion

1. In this project, I build a kNN classifier model to classify the patients suffering from breast cancer. The model yields very good performance as indicated by the model accuracy which was found to be 0.9786 with k=7.
2. With k=3, the training-set accuracy score is 0.9821 while the test-set accuracy to be 0.9714. These two values are quite comparable. So, there is no question of overfitting.
3. I have compared the model accuracy score which is 0.9714 with null accuracy score which is 0.6071. So, we can conclude that our K Nearest Neighbors model is doing a very good job in predicting the class labels.
4. Our original model accuracy score with k=3 is 0.9714. Now, we can see that we get same accuracy score of 0.9714 with k=5. But, if we increase the value of k further, this would result in enhanced accuracy. With k=6,7,8 we get accuracy score of 0.9786. So, it results in performance improvement. If we increase k to 9, then accuracy decreases again to 0.9714. So, we can conclude that our optimal value of k is 7.
5. kNN Classification model with k=7 shows more accurate predictions and less number of errors than k=3 model. Hence, we got performance improvement with k=7.

6. ROC AUC of our model approaches towards 1. So, we can conclude that our classifier does a good job in predicting whether it is benign or malignant cancer.
7. Using the mean cross-validation, we can conclude that we expect the model to be around 96.46 % accurate on average.
8. If we look at all the 10 scores produced by the 10-fold cross-validation, we can also conclude that there is a relatively high variance in the accuracy between folds, ranging from 100% accuracy to 87.72% accuracy. So, we can conclude that the model is very dependent on the particular folds used for training, but it also be the consequence of the small size of the dataset.

In [ ]: