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

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df = pd.read_csv(r"C:\Users\santo\OneDrive\Desktop\Data science\Jan 2026\3rd - KNN\projects\KNN\breast-cancer-wisconsin.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	id	clumpthickness	uniformcellsize	uniformcellshape	margadhesion	epithelial	barenuclei	blandchromatin	normalnucleoli	mit
0	1000025	5	1	1	1	2	1	3	1	
1	1002945	5	4	4	5	7	10	3	2	
2	1015425	3	1	1	1	2	2	3	1	
3	1016277	6	8	8	1	3	4	3	7	
4	1017023	4	1	1	3	2	1	3	1	

In [5]:
```
df.shape
```

Out[5]:
```
(699, 11)
```

In [6]:
```
df.head()
```

Out[6]:

	id	clumpthickness	uniformcellsize	uniformcellshape	margadhesion	epithelial	barenuclei	blandchromatin	normalnucleoli	mit
0	1000025	5	1	1	1	2	1	3	1	
1	1002945	5	4	4	5	7	10	3	2	
2	1015425	3	1	1	1	2	2	3	1	
3	1016277	6	8	8	1	3	4	3	7	
4	1017023	4	1	1	3	2	1	3	1	

In [7]:
```
col_names = ['Id', 'Clump_thickness', 'Uniformity_Cell_Size', 'Uniformity_Cell_Shape', 'Marginal_Adhesion',
             'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucleoli', 'Mitoses', 'Class']

df.columns = col_names

df.columns
```

Out[7]:
```
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 [8]:
```
df.head()
```

Out[8]:

	Id	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Blan
0	1000025	5	1	1	1	2	1	
1	1002945	5	4	4	5	7	10	
2	1015425	3	1	1	1	2	2	
3	1016277	6	8	8	1	3	4	
4	1017023	4	1	1	3	2	1	

In [9]:
```
df.drop('Id', axis=1, inplace=True)
```

In [10]:
```
df.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
```

In [12]:
```
for var in df.columns:
    print(df[var].value_counts())
```

```
Clump_thickness
1     145
5     130
3     108
4      80
10     69
2      50
8      46
6      34
7      23
9      14
Name: count, dtype: int64
Uniformity_Cell_Size
1     384
10     67
3      52
2      45
4      40
5      30
8      29
6      27
7      19
9       6
Name: count, dtype: int64
Uniformity_Cell_Shape
1     353
2      59
10     58
3      56
4      44
5      34
7      30
6      30
8      28
9       7
Name: count, dtype: int64
Marginal_Adhesion
1     407
3      58
2      58
10     55
```

```
4      33
8      25
5      23
6      22
7      13
9       5
Name: count, dtype: int64
Single_Epithelial_Cell_Size
2     386
3      72
4      48
1      47
6      41
5      39
10     31
8      21
7      12
9       2
Name: count, dtype: int64
Bare_Nuclei
1     402
10    132
2      30
5      30
3      28
8      21
4      19
?      16
9       9
7       8
6       4
Name: count, dtype: int64
Bland_Chromatin
2     166
3     165
1     152
7      73
4      40
5      34
8      28
10     20
```

```
9      11
6      10
Name: count, dtype: int64
Normal_Nucleoli
1     443
10     61
3      44
2      36
8      24
6      22
5      19
4      18
7      16
9      16
Name: count, dtype: int64
Mitoses
1     579
2      35
3      33
10     14
4      12
7       9
8       8
5       6
6       3
Name: count, dtype: int64
Class
2    458
4    241
Name: count, dtype: int64
```

In [13]:
```
df['Bare_Nuclei'] = pd.to_numeric(df['Bare_Nuclei'], errors='coerce')
```

In [14]:
```
df.dtypes
```

Out[14]:
```
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
```

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
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
```

In [16]:
```
df.isna().sum()
```

Out[16]:
```
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
```

In [17]:
```
df['Bare_Nuclei'].value_counts()
```

Out[17]:
```
Bare_Nuclei
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: count, dtype: int64
```

In [18]:
```
df['Bare_Nuclei'].unique()
```

Out[18]:
```
array([ 1., 10.,  2.,  4.,  3.,  9.,  7., nan,  5.,  8.,  6.])
```

In [19]:
```
df['Bare_Nuclei'].isna().sum()
```

Out[19]:
```
np.int64(16)
```

In [20]:
```
df['Class'].value_counts()
```

Out[20]:
```
Class
2    458
4    241
Name: count, dtype: int64
```

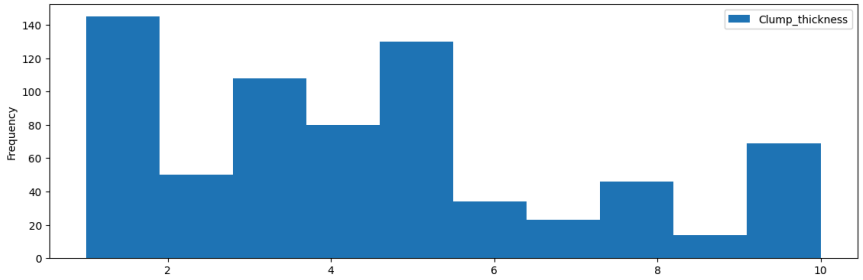
In [23]:
```
df['Class'].value_counts()/np.float64(len(df))
```

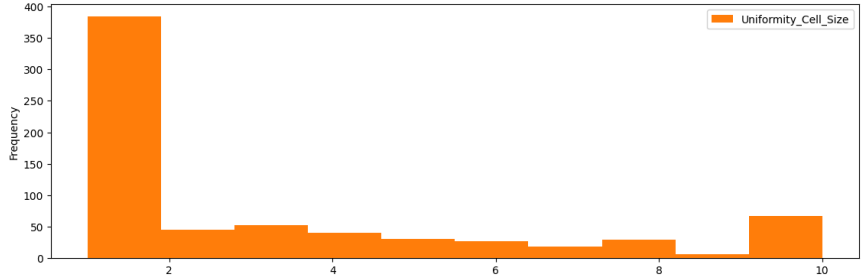
Out[23]:
```
Class
2    0.655222
4    0.344778
Name: count, dtype: float64
```

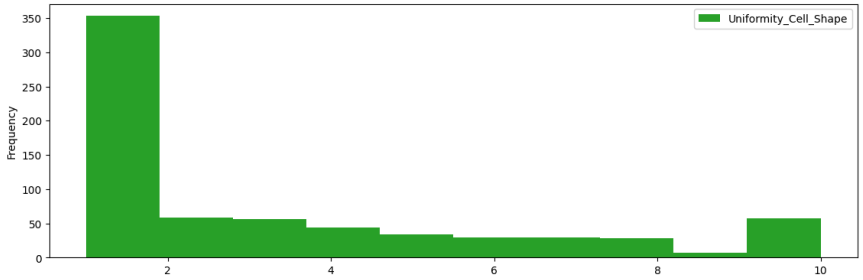
In [24]:
```
print(round(df.describe(),2))
```

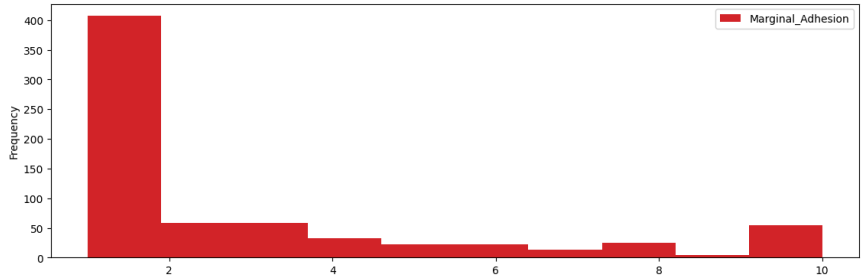
```
       Clump_thickness  Uniformity_Cell_Size  Uniformity_Cell_Shape  \
count           699.00                699.00                 699.00
mean              4.42                  3.13                   3.21
std               2.82                  3.05                   2.97
min               1.00                  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
```

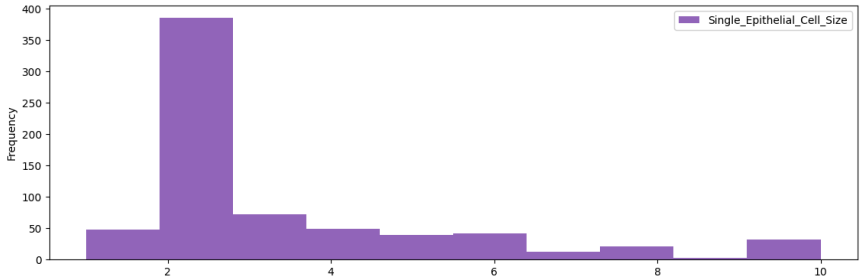
In [26]:
```
plt.rcParams['figure.figsize']=(30,25)

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

plt.show()
```

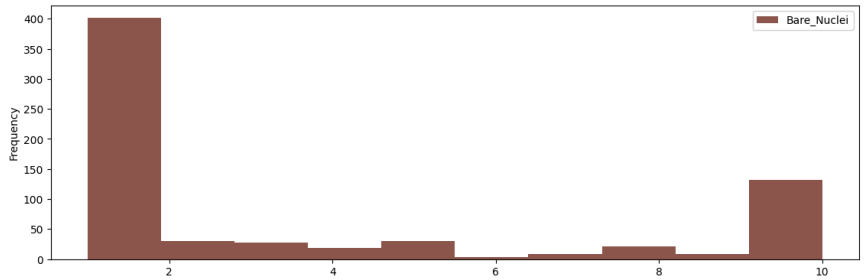

Clump_thickness
Frequency

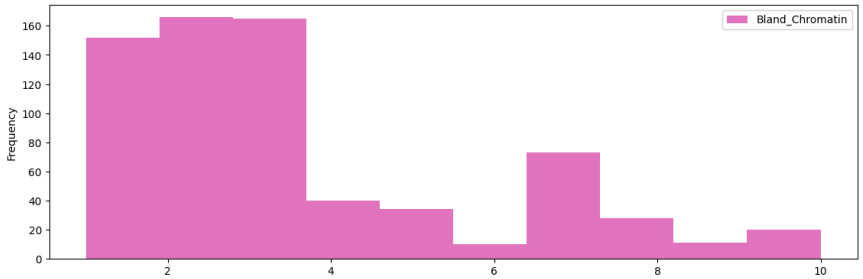

Uniformity_Cell_Size
Frequency

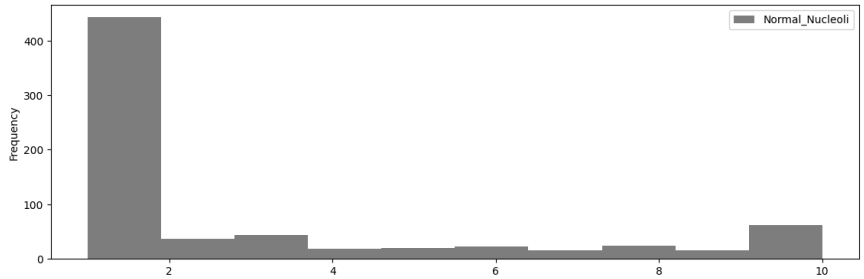

Uniformity_Cell_Shape
Frequency


Marginal_Adhesion
Frequency


Single_Epithelial_Cell_Size
Frequency


Bare_Nuclei
Frequency


Bland_Chromatin
Frequency


Normal_Nucleoli
Frequency


Mitoses
Frequency

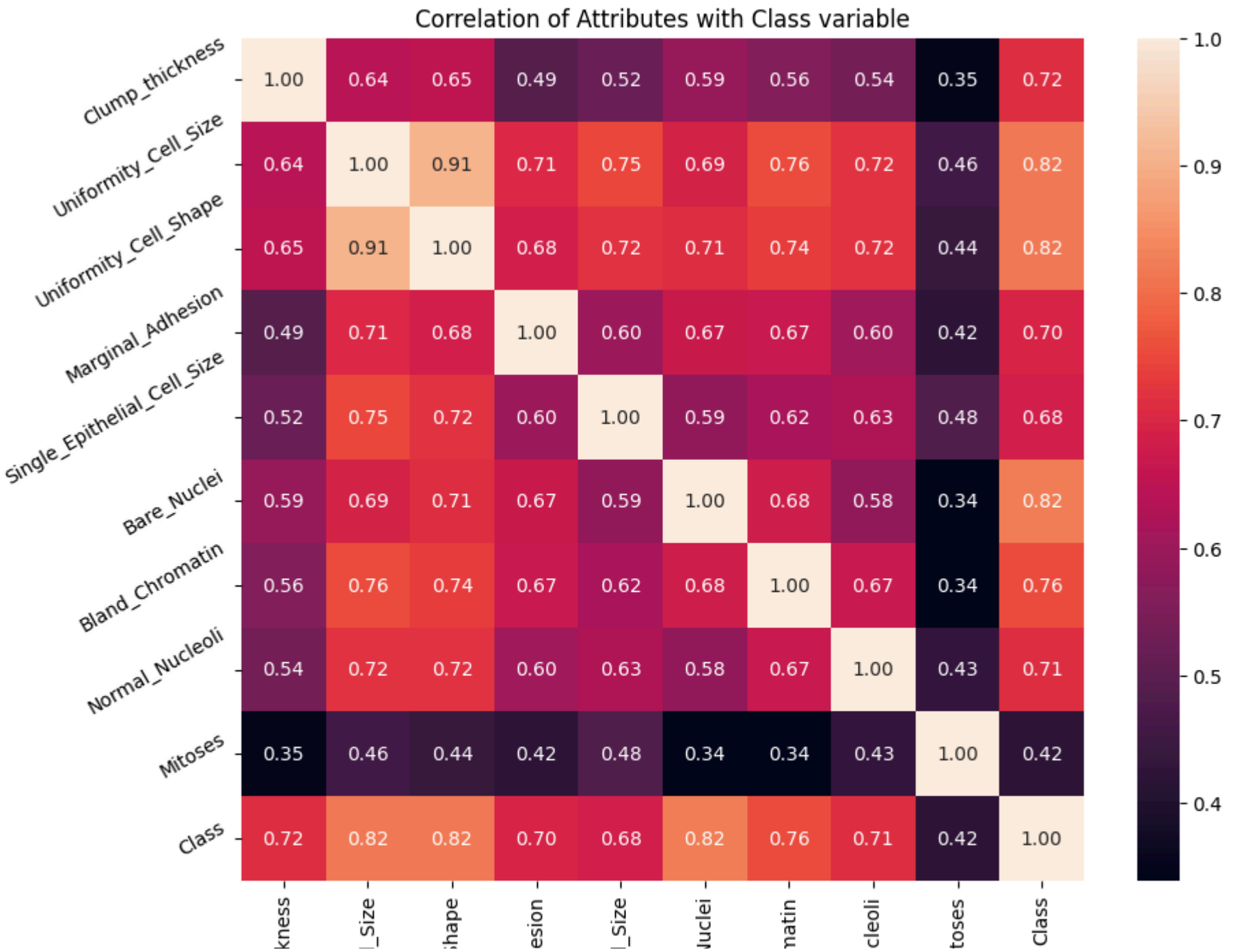

Class
Frequency

In [27]:
```
correlation = df.corr()
```

In [28]:
```
correlation['Class'].sort_values(ascending=False)
```

Out[28]:
```
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
```

In [29]:
```
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()
```


Correlation of Attributes with Class variable
Clump_thickness
Uniformity_Cell_Size
Uniformity_Cell_Shape
Marginal_Adhesion
Single_Epithelial_Cell_Size
Bare_Nuclei
Bland_Chromatin
Normal_Nucleoli
Mitoses
Class
1.00 0.64 0.65 0.49 0.52 0.59 0.56 0.54 0.35 0.72
0.64 1.00 0.91 0.71 0.75 0.69 0.76 0.72 0.46 0.82
0.65 0.91 1.00 0.68 0.72 0.71 0.74 0.72 0.44 0.82
0.49 0.71 0.68 1.00 0.60 0.67 0.67 0.60 0.42 0.70
0.52 0.75 0.72 0.60 1.00 0.59 0.62 0.63 0.48 0.68
0.59 0.69 0.71 0.67 0.59 1.00 0.68 0.58 0.34 0.82
0.56 0.76 0.74 0.67 0.62 0.68 1.00 0.67 0.34 0.76
0.54 0.72 0.72 0.60 0.63 0.58 0.67 1.00 0.43 0.71
0.35 0.46 0.44 0.42 0.48 0.34 0.34 0.43 1.00 0.42
0.72 0.82 0.82 0.70 0.68 0.82 0.76 0.71 0.42 1.00
kness
_Size
hape
esion
_Size
Nuclei
matin
cleoli
toses
Class
1.0
0.9
0.8
0.7
0.6
0.5
0.4

In [30]:
```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

In [31]:
```
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)
```

In [32]:
```
X_train.shape, X_test.shape
```

Out[32]:
```
((559, 9), (140, 9))
```

In [34]:
```
X_train.dtypes
```

Out[34]:
```
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
```

In [35]:
```
X_train.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                  13
Bland_Chromatin               0
Normal_Nucleoli               0
Mitoses                       0
dtype: int64
```

In [36]:
```
X_test.isnull().sum()
```

Out[36]:
```
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 [37]:
```
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 [38]:
```
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 [39]:
```
X_train.isnull().sum()
```

Out[39]:
```
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 [40]:
```
X_test.isnull().sum()
```

Out[40]:
```
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 [41]:
```
X_train.head()
```

Out[41]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chron
293	10	4	4	6	2	10.0	
62	9	10	10	1	10	8.0	
485	1	1	1	3	1	3.0	
422	4	3	3	1	2	1.0	
332	5	2	2	2	2	1.0	

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

Out[42]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chron
476	4	1	2	1	2	1.0	
531	4	2	2	1	2	1.0	
40	6	6	6	9	6	1.0	
432	5	1	1	1	2	1.0	
14	8	7	5	10	7	9.0	

12 Features Scalling

In [43]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [47]:
```
cols = X.columns   # feature names

X_train = pd.DataFrame(X_train, columns=cols)
X_test  = pd.DataFrame(X_test, columns=cols)
```

In [48]:
```
X_train.head()
```

Out[48]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromat
0	2.028383	0.299506	0.289573	1.119077	-0.546543	1.858357	-0.5777
1	1.669451	2.257680	2.304569	-0.622471	3.106879	1.297589	-0.1599
2	-1.202005	-0.679581	-0.717925	0.074148	-1.003220	-0.104329	-0.9955
3	-0.125209	-0.026856	-0.046260	-0.622471	-0.546543	-0.665096	-0.1599
4	0.233723	-0.353219	-0.382092	-0.274161	-0.546543	-0.665096	-0.5777

13. Fit K Neighbours Classifier to the training eet

In [50]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)
```

Out[50]:

KNeighborsClassifier

KNeighborsClassifier(n_neighbors=3)

14. Predict test-set results

In [51]:
```
y_pred = knn.predict(X_test)
y_pred
```

Out[51]:
```
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])
```

In [52]:
```
knn.predict_proba(X_test)[:,0]
```

Out[52]:
```
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 [53]:
```
knn.predict_proba(X_test)[:,1]
```

Out[53]:
```
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.        ])
```

15. Check accuracy score

In [54]:
```
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
```

In [55]:
```
y_pred_train = knn.predict(X_train)
```

In [56]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

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

Check overfitting and underfitting

In [57]:
```
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
```

In [58]:
```
y_test.value_counts()
```

Out[58]:
```
Class
2    85
4    55
Name: count, dtype: int64
```

In [59]:
```
null_accuracy = (85/(85+55))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.6071
```

16. Rebuild kNN Classification model using different values of k

In [60]:
```
knn_5 = KNeighborsClassifier(n_neighbors=5)


knn_5.fit(X_train, y_train)



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
```

In [61]:
```
knn_6 = KNeighborsClassifier(n_neighbors=6)

knn_6.fit(X_train, y_train)

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
```

In [62]:
```
knn_7 = KNeighborsClassifier(n_neighbors=7)
knn_7.fit(X_train, y_train)
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
```

In [63]:
```
# instantiate the model with k=8
knn_8 = KNeighborsClassifier(n_neighbors=8)


# fit the model to the training set
knn_8.fit(X_train, y_train)


# 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
```

In [64]:
```
# instantiate the model with k=9
knn_9 = KNeighborsClassifier(n_neighbors=9)


# fit the model to the training set
```

```
knn_9.fit(X_train, y_train)


# 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
```

17 Confusion matrix

In [65]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

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

print('\nTrue Positives(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 Positives(TP) =  83

True Negatives(TN) =  53

False Positives(FP) =  2

False Negatives(FN) =  2
```

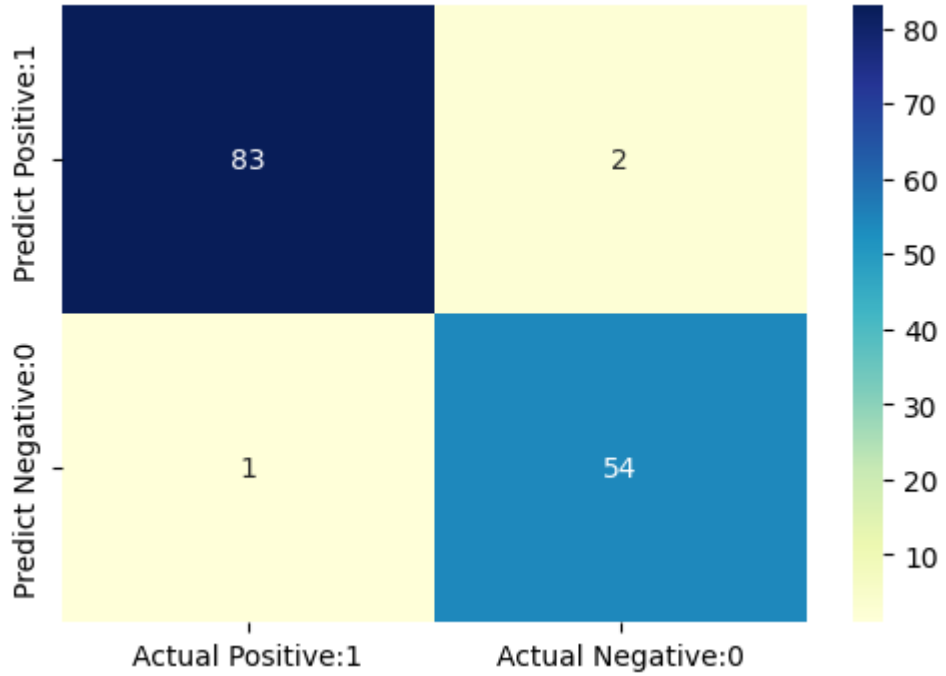
In [66]:
```
cm_7 = confusion_matrix(y_test, y_pred_7)

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
```

In [67]:
```
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[67]: <Axes: >

In [68]:
```
plt.show()
```



In [70]:
```
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
```

In [73]:
```
TP = cm_7[0,0]
TN = cm_7[1,1]
FP = cm_7[0,1]
FN = cm_7[1,0]
```

In [74]:
```
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.9786
```

In [75]:
```
classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.0214
```

In [76]:
```
precision = TP / float(TP + FP)


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

```
Precision : 0.9765
```

In [77]:
```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9881
```

True Positive rate

In [78]:
```
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 [79]:
```
false_positive_rate = FP / float(FP + TN)
```

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

```
False Positive Rate : 0.0357
```

In [80]:
```
specificity = TN / (TN + FP)

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

```
Specificity : 0.9643
```

In [81]:
```
# print the first 10 predicted probabilities of two classes- 2 and 4

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

y_pred_prob
```

Out[81]:
```
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 [82]:
```
# store the probabilities in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - benign cancer (2)', 'Prob of - malignant cancer (4)'])

y_pred_prob_df
```

Out[82]:

	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 [83]:
```
knn.predict_proba(X_test)[0:10, 1]
```

Out[83]:
```
array([0.        , 0.        , 0.66666667, 0.        , 1.        ,
       0.        , 1.        , 0.        , 1.        , 0.33333333])
```

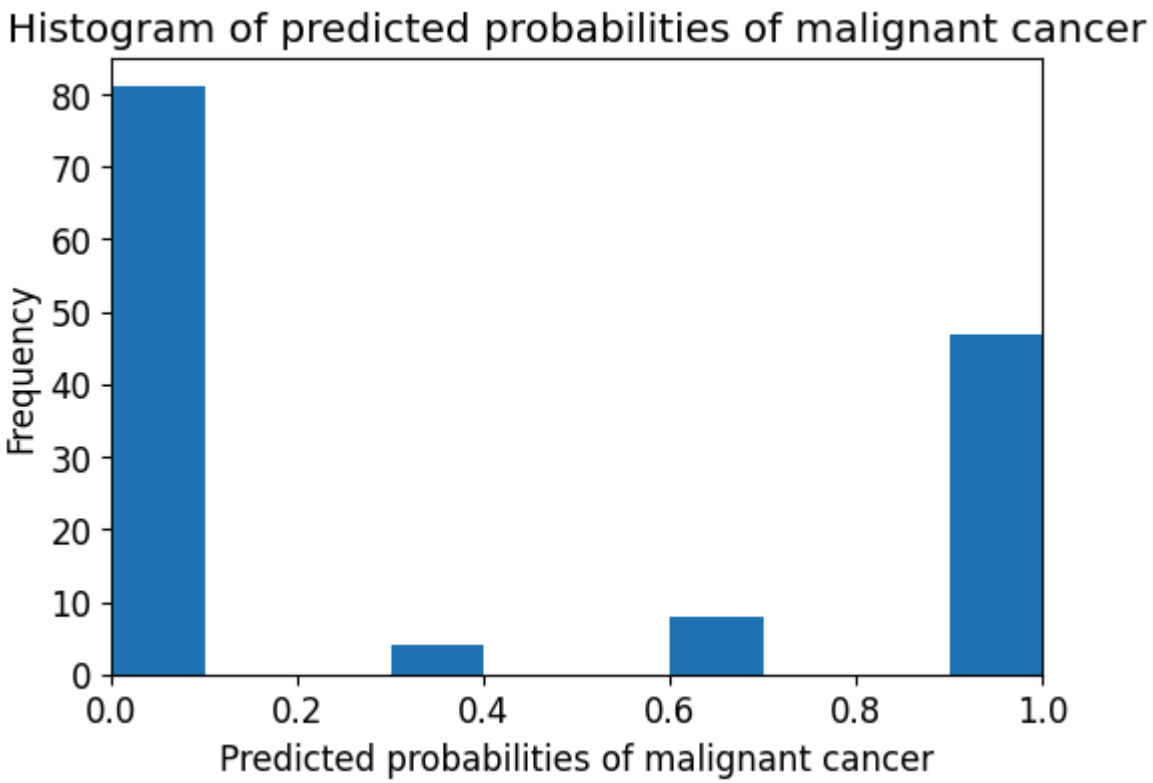
In [84]:
```
y_pred_1 = knn.predict_proba(X_test)[:, 1]
```

In [85]:
```
# 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 predicted probabilities
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')
```

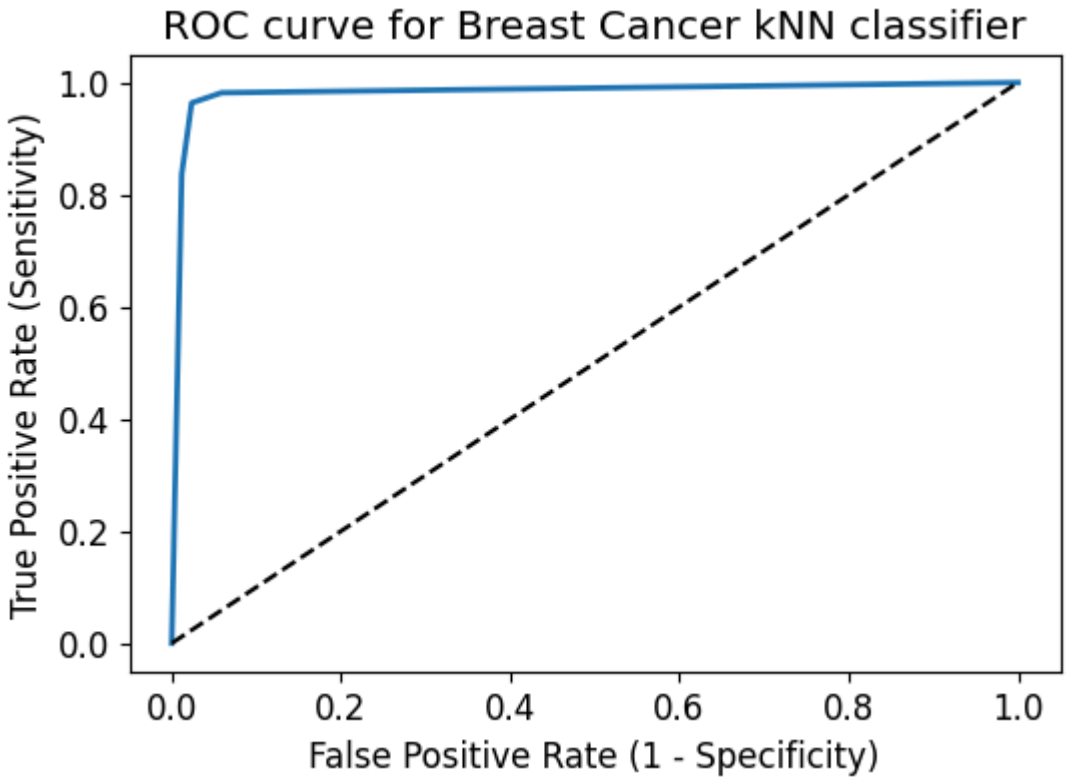
Out[85]:
```
Text(0, 0.5, 'Frequency')
```

In [86]:
```
plt.show()
```



In [87]:
```
# 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 [88]:
```
# 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 [ ]: