K Nearest Neighbours

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

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

In [3]:
```
df=pd.read_csv(r"C:\Users\JANHAVI\Desktop\breast cancer.csv")
```

In [4]:
```
df
```

Out[4]:

	1000025	5	1	1.1	1.2	2	1.3	3	1.4	1.5	2.1
0	1002945	5	4	4	5	7	10	3	2	1	2
1	1015425	3	1	1	1	2	2	3	1	1	2
2	1016277	6	8	8	1	3	4	3	7	1	2
3	1017023	4	1	1	3	2	1	3	1	1	2
4	1017122	8	10	10	8	7	10	9	7	1	4
...	...	...	...	...	...	...	...	...	...	...	...
693	776715	3	1	1	1	3	2	1	1	1	2
694	841769	2	1	1	1	2	1	1	1	1	2
695	888820	5	10	10	3	7	3	8	10	2	4
696	897471	4	8	6	4	3	4	10	6	1	4
697	897471	4	8	8	5	4	5	10	4	1	4

698 rows × 11 columns

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

Out[5]:
```
(698, 11)
```

In [6]:
```
col_names = ['Id', 'Clump_thickness', 'Uniformity_Cell_Size', 'Uniformity_Cell_Shap
             'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Norm

df.columns = col_names

df.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]:
```
df.head()
```

Out[7]:

	Id	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Sing
0	1002945	5	4	4	5	
1	1015425	3	1	1	1	
2	1016277	6	8	8	1	
3	1017023	4	1	1	3	
4	1017122	8	10	10	8	

In [8]:
```
 #drop Id column from dataset
df.drop('Id', axis=1, inplace=True)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 698 entries, 0 to 697
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   Clump_thickness              698 non-null    int64 
 1   Uniformity_Cell_Size         698 non-null    int64 
 2   Uniformity_Cell_Shape        698 non-null    int64 
 3   Marginal_Adhesion            698 non-null    int64 
 4   Single_Epithelial_Cell_Size  698 non-null    int64 
 5   Bare_Nuclei                  698 non-null    object
 6   Bland_Chromatin              698 non-null    int64 
 7   Normal_Nucleoli              698 non-null    int64 
 8   Mitoses                      698 non-null    int64 
 9   Class                        698 non-null    int64 
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

In [10]:
```
for var in df.columns:

    print(df[var].value_counts())
```

```
Clump_thickness
1     145
5     129
3     108
4      80
10     69
2      50
8      46
6      34
7      23
9      14
Name: count, dtype: int64
Uniformity_Cell_Size
1     383
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     352
2      59
10     58
3      56
4      44
5      34
6      30
7      30
8      28
9       7
Name: count, dtype: int64
Marginal_Adhesion
1     406
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     385
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     401
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     164
1     152
7      73
4      40
5      34
8      28
10     20
9      11
6      10
Name: count, dtype: int64
Normal_Nucleoli
1     442
10     61
3      44
2      36
8      24
6      22
5      19
4      18
7      16
9      16
Name: count, dtype: int64
Mitoses
1     578
2      35
3      33
10     14
4      12
7       9
8       8
5       6
6       3
Name: count, dtype: int64
Class
2    457
4    241
Name: count, dtype: int64
```

Convert Data Type of Bare_Nuclei to integer

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

In [13]:
```
df.dtypes
```

Out[13]:
```
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 Variable

In [14]:
```
df.isnull().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
```

In [15]:
```
df.isna().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['Bare_Nuclei'].value_counts()
```

Out[16]:
```
Bare_Nuclei
1.0     401
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 [17]:
```
df['Bare_Nuclei'].unique()
```

Out[17]:
```
array([10.,  2.,  4.,  1.,  3.,  9.,  7., nan,  5.,  8.,  6.])
```

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

Out[18]:
```
16
```

In [19]:
```
df['Class'].value_counts()
```

Out[19]:
```
Class
2    457
4    241
Name: count, dtype: int64
```

Check Percentage of frequency distribution of class

In [21]:
```
df['Class'].value_counts() / float(len(df))
```

Out[21]:
```
Class
2    0.654728
4    0.345272
Name: count, dtype: float64
```

Outlier in Numerical Variables

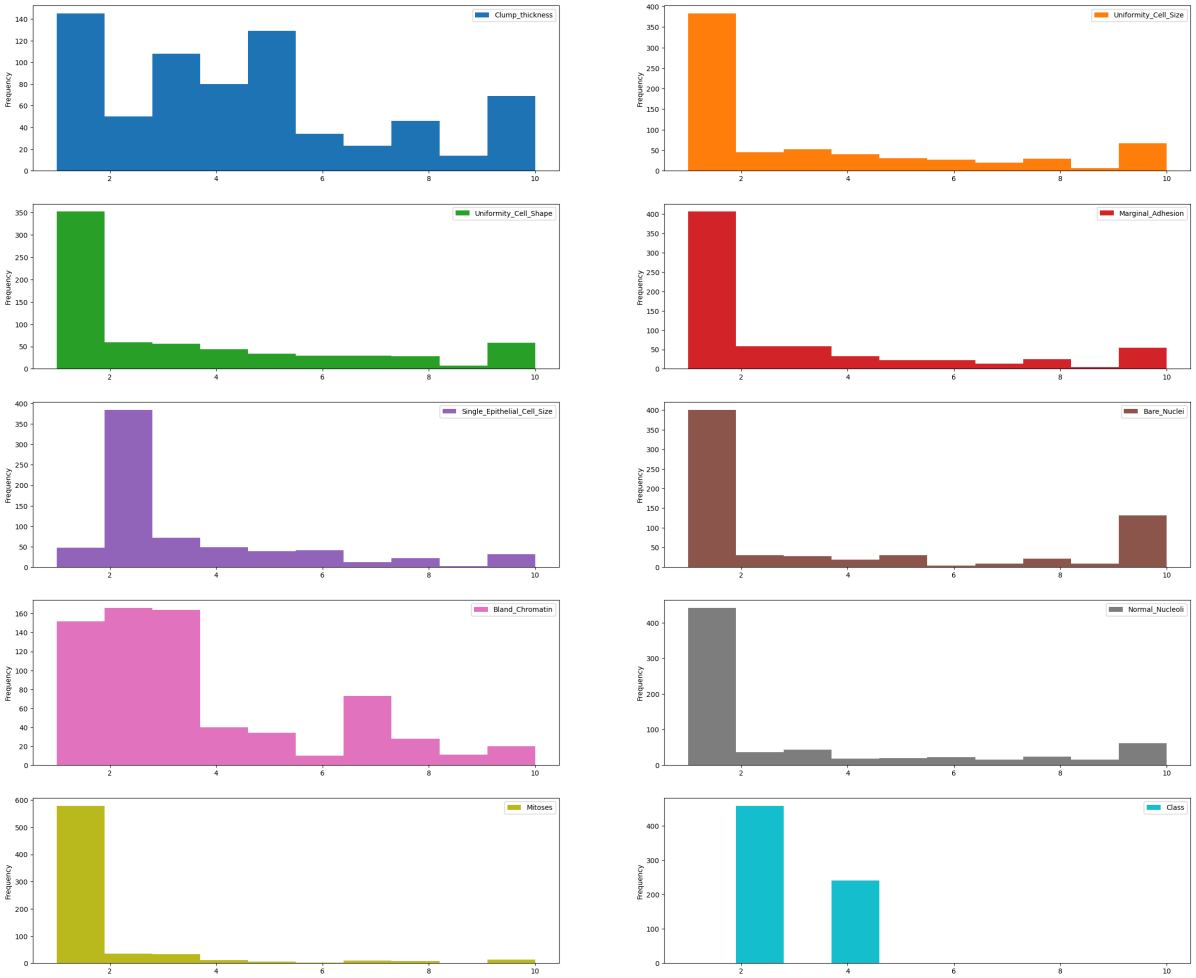
In [22]:
```
print(round(df.describe(),2))
```

```
       Clump_thickness  Uniformity_Cell_Size  Uniformity_Cell_Shape  \
count           698.00                698.00                 698.00
mean              4.42                  3.14                   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             698.00                       698.00       682.00
mean                2.81                         3.22         3.55
std                 2.86                         2.22         3.65
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           698.00           698.00   698.00  698.00
mean              3.44             2.87     1.59    2.69
std               2.44             3.06     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 [23]:
```
plt.rcParams['figure.figsize']=(30,25)
```

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

plt.show()
```



Estimating Correlation Coefficient

In [24]:
```
correlation = df.corr()
```

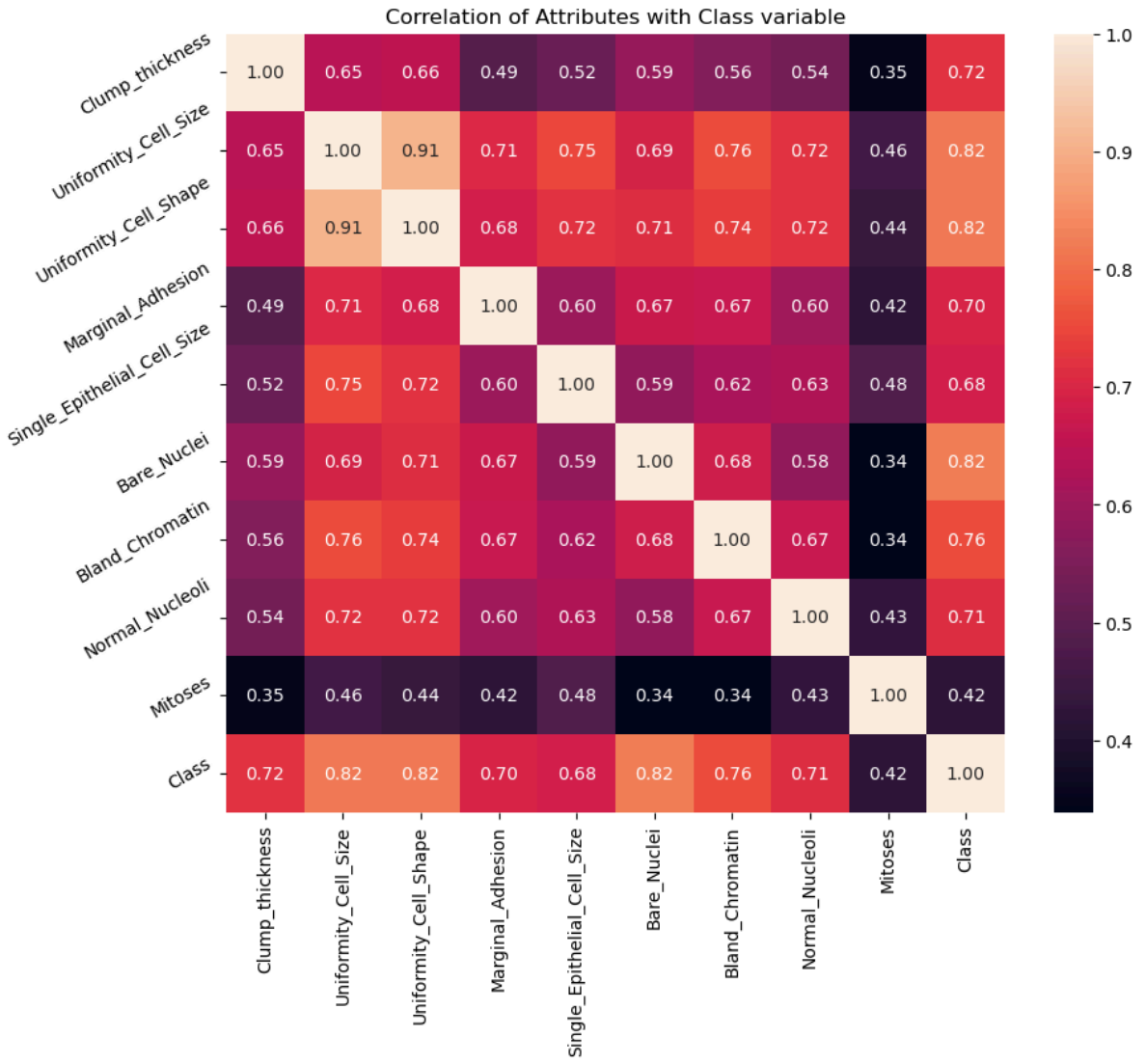
In [26]:
```
correlation['Class'].sort_values(ascending=False)
```

Out[26]:
```
Class                          1.000000
Bare_Nuclei                    0.822563
Uniformity_Cell_Shape          0.818794
Uniformity_Cell_Size           0.817772
Bland_Chromatin                0.756732
Clump_thickness                0.716509
Normal_Nucleoli                0.712067
Marginal_Adhesion              0.696605
Single_Epithelial_Cell_Size    0.682618
Mitoses                        0.423008
Name: Class, dtype: float64
```

Correlation Heat Map

In [27]:
```
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()
```



Declare feature vector and target variable

In [28]:
```
X = df.drop(['Class'], axis=1)

y = df['Class']
```

Split datainto separate training and test set

In [29]:
```
# 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_s
```

In [30]:
```
X_train.shape, X_test.shape
```

Out[30]:
```
((558, 9), (140, 9))
```

Feature Engineering

In [31]:
```
X_train.dtypes
```

Out[31]:
```
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 [32]:
```
X_train.isnull().sum()
```

Out[32]:
```
Clump_thickness                0
Uniformity_Cell_Size           0
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                   15
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
dtype: int64
```

In [33]:
```
X_test.isnull().sum()
```

Out[33]:
```
Clump_thickness                0
Uniformity_Cell_Size           0
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                    1
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
dtype: int64
```

In [34]:
```
for col in X_train.columns:
    if X_train[col].isnull().mean()>0:
        print(col, round(X_train[col].isnull().mean(),4))
```

```
Bare_Nuclei 0.0269
```

In [35]:
```
# impute missing values in X_train and X_test with respective column median in X_tr

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 [36]:
```
X_train.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                   0
Bland_Chromatin               0
Normal_Nucleoli               0
Mitoses                       0
dtype: int64
```

In [37]:
```
X_test.isnull().sum()
```

Out[37]:
```
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 [38]:
```
X_train.head()
```

Out[38]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epith
62	6	3	4	1	
193	3	1	1	1	
263	7	9	4	10	
222	7	5	6	3	
140	2	1	1	1	

In [39]:
```
X_test.head()
```

Out[39]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epith
603	5	3	2	8	
619	3	1	1	1	
452	4	5	5	8	
85	3	3	6	4	
416	1	1	1	1	

Feature Scaling

In [40]:
```
cols = X_train.columns
```

In [41]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [42]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [43]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [44]:
```
X_train.head()
```

Out[44]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epitheli
0	0.574621	-0.040143	0.277515	-0.629622	
1	-0.497748	-0.680143	-0.721540	-0.629622	
2	0.932077	1.879857	0.277515	2.541854	
3	0.932077	0.599857	0.943552	0.075150	
4	-0.855205	-0.680143	-0.721540	-0.629622	

Fit K Neighbours Classifier to the training eet

In [45]:
```
# import KNeighbors ClaSSifier from sklearn
from sklearn.neighbors import KNeighborsClassifier


# instantiate the model
knn = KNeighborsClassifier(n_neighbors=3)


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

Out[45]:

▾ KNeighborsClassifier

▶ Parameters

Predcit test-set result

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

y_pred
```

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

Predict Probability

In [47]:
```
# probability of getting output as 2 - benign cancer

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

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

In [48]:
```
# probability of getting output as 4 - malignant cancer

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

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

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
```

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_pre
```

```
Training-set accuracy score: 0.9803
```

Checking Overfitting and Underfitting

In [52]:
```
# print the scores 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: 0.9803
Test set score: 0.9714
```

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

y_test.value_counts()
```

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

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
```

Rebuild KNN Classification Model using different Values of k = 5

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


# fit the model to the training set
knn_5.fit(X_train, y_train)


# 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_p
```

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

Rebuild KNN Classification Model using K = 6

In [56]:
```
# instantiate the model with k=6
knn_6 = KNeighborsClassifier(n_neighbors=6)


# fit the model to the training set
knn_6.fit(X_train, y_train)


# 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_p
```

```
Model accuracy score with k=6 : 0.9643
```

Rebuild KNN Classification Model using K = 7

In [57]:
```
# instantiate the model with k=7
knn_7 = KNeighborsClassifier(n_neighbors=7)
```

```
# fit the model to the training set
knn_7.fit(X_train, y_train)


# 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_p
```

```
Model accuracy score with k=7 : 0.9571
```

Rebuild KNN Classification Model using K = 8

In [58]:
```
# 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_p
```

```
Model accuracy score with k=8 : 0.9643
```

Rebuild KNN Classification Model using K = 9

In [59]:
```
# 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_p
```

```
Model accuracy score with k=9 : 0.9643
```

Confusion Matrix

In [60]:
```
# 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 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])
```

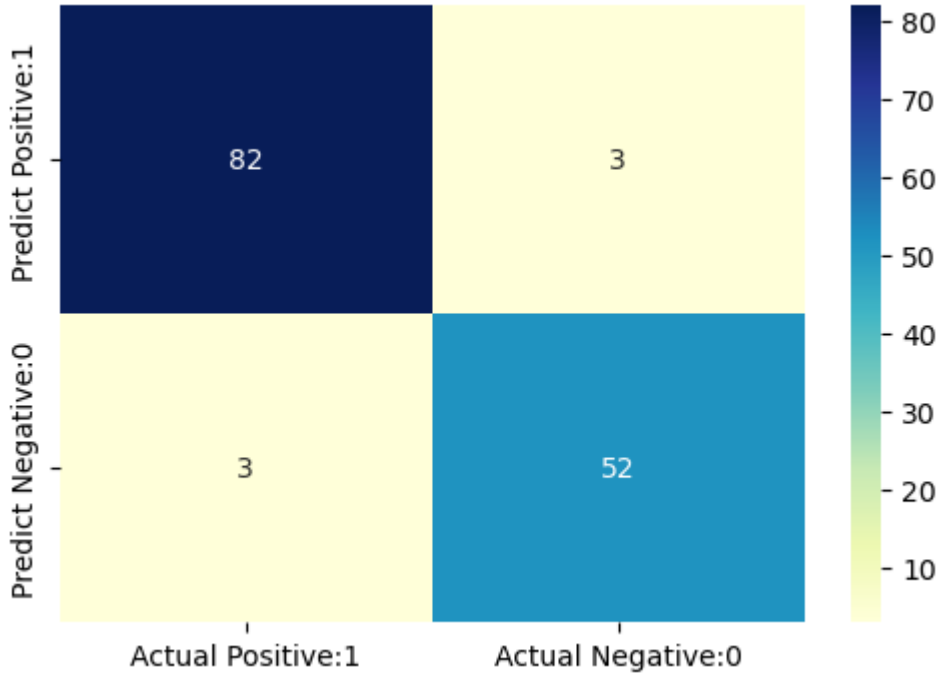
```
Confusion matrix

 [[83  2]
 [ 2 53]]

True Positives(TP) =  83

True Negatives(TN) =  53

False Positives(FP) =  2

False Negatives(FN) =  2
```

In [61]:
```
# 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

 [[82  3]
 [ 3 52]]

True Positives(TP) =  82

True Negatives(TN) =  52

False Positives(FP) =  3

False Negatives(FN) =  3
```

In [62]:
```
# visualize confusion matrix with seaborn heatmap

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

cm_matrix = pd.DataFrame(data=cm_7, columns=['Actual Positive:1', 'Actual Negative:
                                   index=['Predict Positive:1', 'Predict Negative:0']

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[62]:
```
<Axes: >
```



Classification Metrices

In [63]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_7))
```

```
              precision    recall  f1-score   support

           2       0.96      0.96      0.96        85
           4       0.95      0.95      0.95        55

    accuracy                           0.96       140
   macro avg       0.96      0.96      0.96       140
weighted avg       0.96      0.96      0.96       140
```

Classification Accuracy

In [64]:
```
TP = cm_7[0,0]
TN = cm_7[1,1]
FP = cm_7[0,1]
FN = cm_7[1,0]
```

In [65]:
```
# print classification accuracy

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

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

```
Classification accuracy : 0.9571
```

Classification Error

In [66]:
```
# print classification error
```

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

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

```
Classification error : 0.0429
```

Precision

In [67]:
```
# print precision score

precision = TP / float(TP + FP)


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

```
Precision : 0.9647
```

Recall

In [68]:
```
recall = TP / float(TP + FN)

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

```
Recall or Sensitivity : 0.9647
```

True Positive Rate

In [70]:
```
true_positive_rate = TP / float(TP + FN)
print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 0.9647
```

False Positive Rate

In [71]:
```
false_positive_rate = FP / float(FP + TN)
print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.0545
```

Specificity

In [72]:
```
specificity = TN / (TN + FP)

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

```
Specificity : 0.9455
```

In [73]:
```
# 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[73]:
```
array([[0.        , 1.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.33333333, 0.66666667],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ]])
```

In [74]:
```
# store the probabilities in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - benign cancer (

y_pred_prob_df
```

Out[74]:

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

In [75]:
```
# print the first 10 predicted probabilities for class 4 - Probability of malignant

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

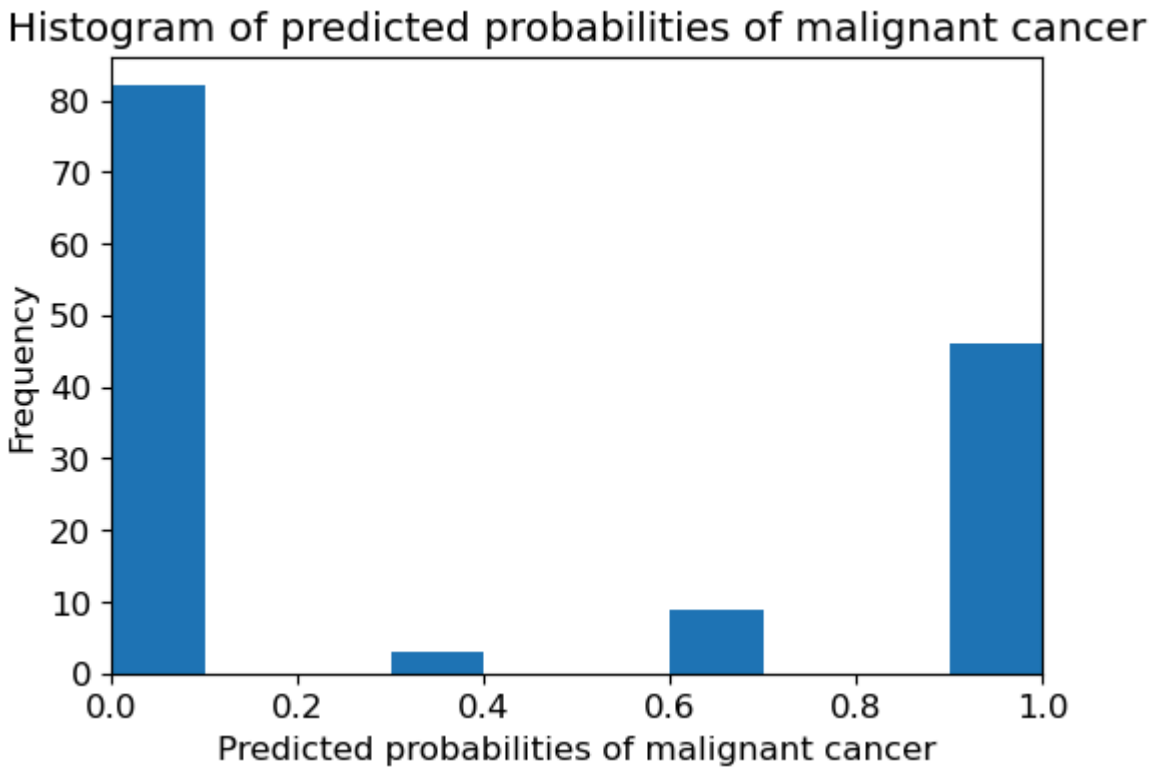
Out[75]:
```
array([1.        , 0.        , 1.        , 0.66666667, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ])
```

In [76]:
```
# store the predicted probabilities for class 4 - Probability of malignant cancer

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

In [77]:
```
# 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)
```

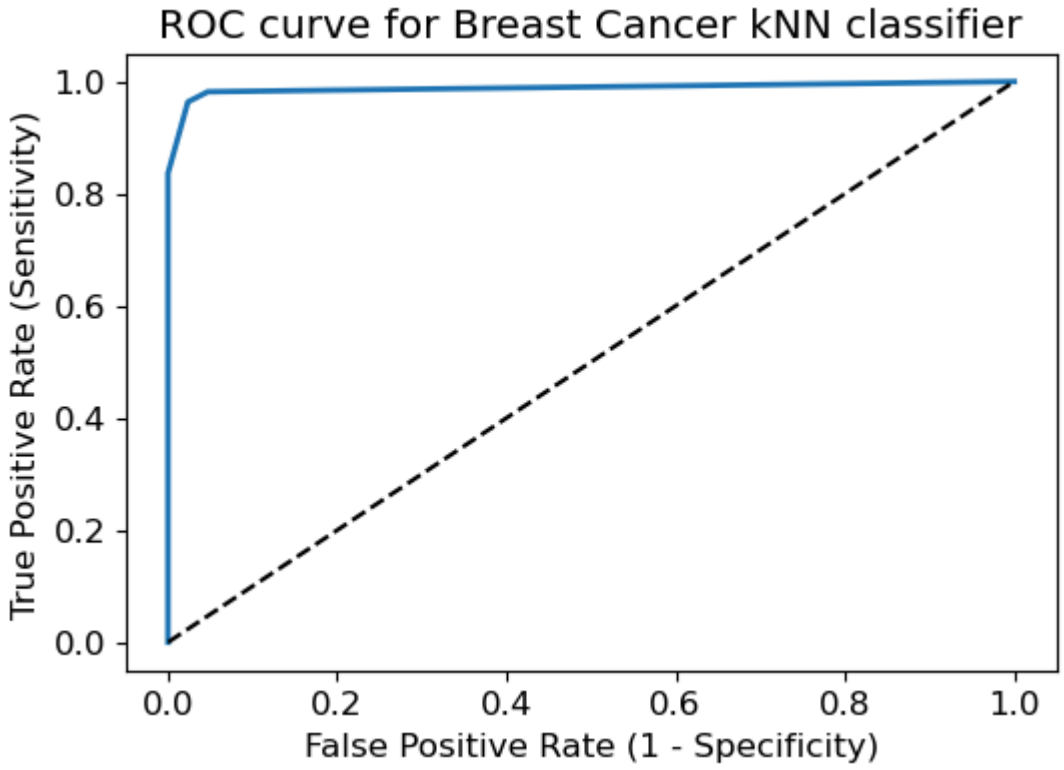
```
# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of malignant cancer')


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


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

Out[77]:
```
Text(0, 0.5, 'Frequency')
```



ROC- AUC

In [78]:
```
# 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 [79]:
```
# 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 : 0.9883
```

Interpretation

In [83]:
```
# 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='r

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

```
Cross validated ROC AUC : 0.9811
```

K- Fold Cross Validation

In [84]:
```
# Applying 10-Fold Cross Validation

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:[0.96428571 0.98214286 0.96428571 0.98214286 0.96428571 0.
94642857
 0.96428571 1.         0.98181818 0.96363636]
```

In [85]:
```
# compute Average cross-validation score

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

```
Average cross-validation score: 0.9713
```