```
In [1]: import os
        print("Current_Working_Directory:", os.getcwd())
```

```
Current_Working_Directory: C:\Users\viraj\Code_a\Machine_learning\Knn
```

```
In [2]: directory_path = "C:/Users/viraj/Code_a/Machine_learning/Knn"
        contents = os.listdir(directory_path)
        print("Contents")
        for i in contents:
            print(i)
```

```
Contents
.ipynb_checkpoints
breast_cancer_wisconsin.csv
Knn.ipynb
KNN_Tut.ipynb
Untitled.ipynb
```

```
In [3]: import pandas as pd
        import numpy as np
        import requests
```

```
In [4]: # URL of the dataset
        url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wiscons
```

```
In [5]: # Fetch the data
        response = requests.get(url)
        data = response.text
```

```
In [6]: # Save the data to a temporary csv file
        with open('breast_cancer_wisconsin.csv', 'w') as file:
            file.write(data)

        # Load the data into a DataFrame
        column_names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size',
                        'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Ce
                        'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class
        df = pd.read_csv('breast_cancer_wisconsin.csv', names=column_names, header=None, na_va
```

```
In [7]: df
```

Out[7]:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli
0	1000025	5	1	1	1	2	1.0	3	1
1	1002945	5	4	4	5	7	10.0	3	2
2	1015425	3	1	1	1	2	2.0	3	1
3	1016277	6	8	8	1	3	4.0	3	7
4	1017023	4	1	1	3	2	1.0	3	1
...	...	...	...	...	...	...	...	...	...
694	776715	3	1	1	1	3	2.0	1	1
695	841769	2	1	1	1	2	1.0	1	1
696	888820	5	10	10	3	7	3.0	8	10
697	897471	4	8	6	4	3	4.0	10	6
698	897471	4	8	8	5	4	5.0	10	4

699 rows × 11 columns

What is Knn?

Imagine you have a bunch of points scattered on a graph, each representing something like houses with their sizes and prices. Now, k Nearest Neighbors (kNN) is like having a smart friend who helps you predict things about new houses. Here's how it works:

First off, it's super flexible. You don't have to assume anything about the data beforehand. It looks at the points you already have and figures out patterns from them. When you want to predict something about a new house, like whether it's expensive or affordable, kNN kicks in.

It does this by looking at the k closest houses you already know about. If you're trying to decide if a new house is expensive or not, it checks out the prices of those closest houses. If most of them are pricey, it guesses that the new one is probably expensive too. If it's trying to guess the price of a new house, it looks at the prices of those nearest houses and takes the average. That usually gives a pretty good estimate.

One cool thing is that kNN doesn't do much work upfront. It just remembers where all the houses are on the graph. When you need to make a prediction, it looks at those remembered houses and makes a decision based on them.

Sometimes, though, some houses should have more say in the prediction than others. Maybe the ones that are really close to the new house should count more. kNN can handle that too. It can give more weight to the closer houses when making its guess.

Intitution Behind KNN!

The intuition behind k Nearest Neighbors (kNN) stems from the notion that similar data points tend to share similar characteristics or outcomes. At its essence, kNN leverages the principle of proximity: points that are close to each other in a feature space are likely to exhibit similar behaviors or belong to the same category.

Imagine plotting data points on a graph, where each point represents an observation characterized by various features. The intuition behind kNN is akin to saying, "Show me your friends, and I'll tell you who you are." In other words, if you want to understand something about a new data point, look at its nearest neighbors, because they are the most similar ones.

For instance, in a classification task where you're trying to determine whether an email is spam or not, kNN would look at the characteristics (like keywords, sender, etc.) of the closest emails to the one you're classifying. If most of the closest emails are spam, then it's likely that the new one is spam too.

Similarly, in a regression task where you're predicting the price of a house, kNN would examine the prices of the nearest houses to the one you're interested in. By averaging those prices, it can make a reasonable estimate for the new house's price.

This intuition underscores kNN's adaptability and simplicity. It doesn't rely on complex mathematical models or assumptions about the data distribution. Instead, it leverages the inherent structure of the data to make predictions. However, it's crucial to choose an appropriate value for 'k' (the number of nearest neighbors to consider) to ensure reliable predictions and avoid overfitting or underfitting the model.

Exploratory data analysis

In [8]:
```
df.shape
```
Out[8]:
```
(699, 11)
```

In [9]:
```
type(df)
```
Out[9]:
```
pandas.core.frame.DataFrame
```

In [10]:
```
df.describe()
```

Out[10]:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bla Chroma
count	6.990000e+02	699.000000	699.000000	699.000000	699.000000	699.000000	683.000000	699.000
mean	1.071704e+06	4.417740	3.134478	3.207439	2.806867	3.216023	3.544656	3.437
std	6.170957e+05	2.815741	3.051459	2.971913	2.855379	2.214300	3.643857	2.438
min	6.163400e+04	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000
25%	8.706885e+05	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000
50%	1.171710e+06	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000
75%	1.238298e+06	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000
max	1.345435e+07	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000

In [11]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Sample code number           699 non-null    int64  
 1   Clump Thickness              699 non-null    int64  
 2   Uniformity of Cell Size      699 non-null    int64  
 3   Uniformity of Cell Shape     699 non-null    int64  
 4   Marginal Adhesion            699 non-null    int64  
 5   Single Epithelial Cell Size  699 non-null    int64  
 6   Bare Nuclei                  683 non-null    float64
 7   Bland Chromatin              699 non-null    int64  
 8   Normal Nucleoli              699 non-null    int64  
 9   Mitoses                      699 non-null    int64  
 10  Class                        699 non-null    int64  
dtypes: float64(1), int64(10)
memory usage: 60.2 KB
```

In [12]:

```
df.isnull().sum()
```

Out[12]:

```
Sample code number              0
Clump Thickness                 0
Uniformity of Cell Size         0
Uniformity of Cell Shape        0
Marginal Adhesion               0
Single Epithelial Cell Size     0
Bare Nuclei                    16
Bland Chromatin                 0
Normal Nucleoli                 0
Mitoses                         0
Class                           0
dtype: int64
```

In [13]:

```
#checking what percent of data is been missing
df.isnull().sum()/len(data)*100
```

Out[13]:
```
Sample code number             0.000000
Clump Thickness                0.000000
Uniformity of Cell Size        0.000000
Uniformity of Cell Shape       0.000000
Marginal Adhesion              0.000000
Single Epithelial Cell Size    0.000000
Bare Nuclei                    0.080446
Bland Chromatin                0.000000
Normal Nucleoli                0.000000
Mitoses                        0.000000
Class                          0.000000
dtype: float64
```

In [14]:
```
#So we could remove the rows that have th null values as it is less than 1% of the whc
```

In [15]:
```
# Drop rows with missing Bare Nuclei
df.dropna(subset=['Bare Nuclei'], inplace=True)
```

In [16]:
```
df.isnull().sum()/len(data)*100
```

Out[16]:
```
Sample code number             0.0
Clump Thickness                0.0
Uniformity of Cell Size        0.0
Uniformity of Cell Shape       0.0
Marginal Adhesion              0.0
Single Epithelial Cell Size    0.0
Bare Nuclei                    0.0
Bland Chromatin                0.0
Normal Nucleoli                0.0
Mitoses                        0.0
Class                          0.0
dtype: float64
```

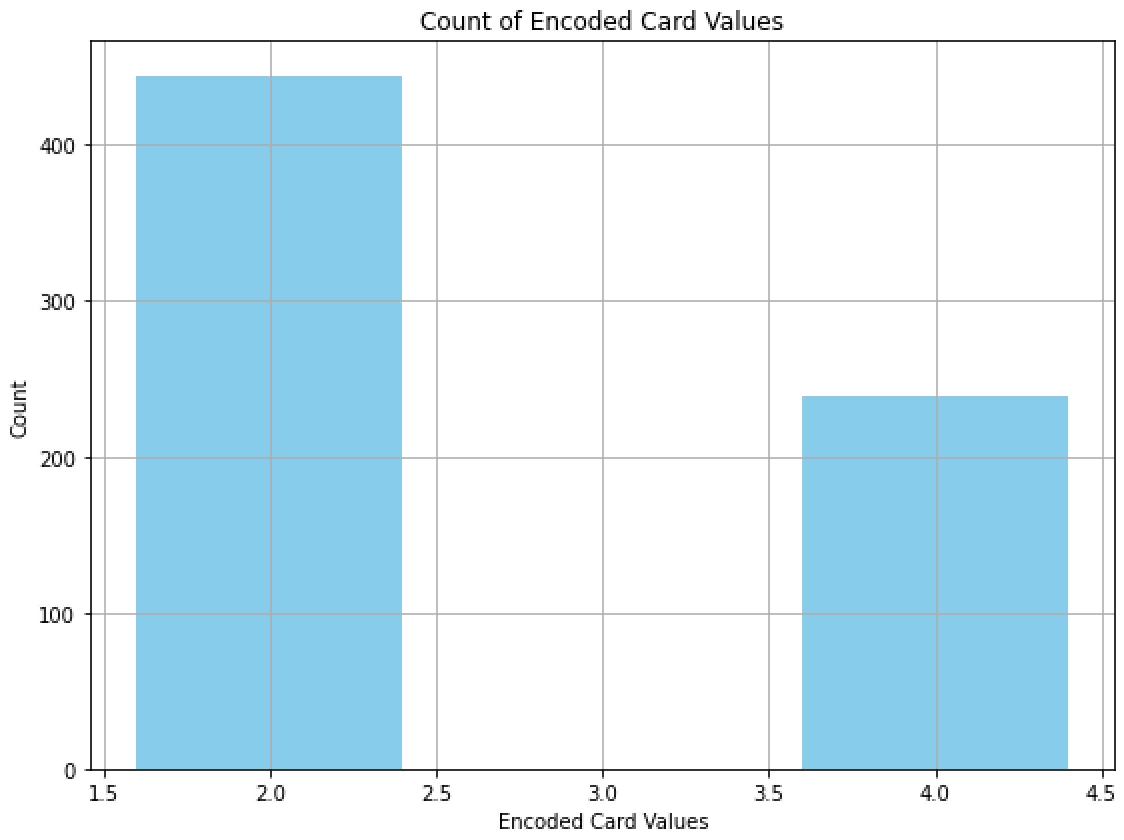
In [17]:
```
df.shape
```

Out[17]:
```
(683, 11)
```

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

Out[18]:
```
array([2, 4], dtype=int64)
```

In [19]:
```
import matplotlib.pyplot as plt
import seaborn as sns
value_counts = df['Class'].value_counts()

# Create a bar plot
plt.figure(figsize=(8, 6))
plt.bar(value_counts.index, value_counts.values, color='skyblue')

# Add labels and title
plt.xlabel('Encoded Card Values')
plt.ylabel('Count')
plt.title('Count of Encoded Card Values')

# Show plot
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [20]:
```
# Compute the imbalance ratio
imbalance_ratio = value_counts.max() / value_counts.min()
print("Imbalance Ratio:", imbalance_ratio)
```

```
Imbalance Ratio: 1.8577405857740585
```

An imbalance ratio of approximately 1.86, it indicates that the number of instances in the majority class is about 1.86 times the number of instances in the minority class. This suggests some level of imbalance, though it's not extremely severe. Typically, an imbalance ratio closer to 1 indicates a more balanced dataset, whereas larger values indicate greater imbalance.

In [21]:
```
for var in df.columns:
    print(df[var].value_counts())
```

```
Sample code number
1182404    6
1276091    5
1198641    3
897471     2
1168736    2
          ..
1232225    1
1236043    1
1241559    1
1241679    1
832226     1
Name: count, Length: 630, dtype: int64
Clump Thickness
1     139
5     128
3     104
4      79
10     69
2      50
8      44
6      33
7      23
9      14
Name: count, dtype: int64
Uniformity of Cell Size
1     373
10     67
3      52
2      45
4      38
5      30
8      28
6      25
7      19
9       6
Name: count, dtype: int64
Uniformity of Cell Shape
1     346
10     58
2      58
3      53
4      43
5      32
7      30
6      29
8      27
9       7
Name: count, dtype: int64
Marginal Adhesion
1     393
3      58
2      58
10     55
4      33
8      25
5      23
6      21
7      13
9       4
```

```
Name: count, dtype: int64
Single Epithelial Cell Size
2     376
3      71
4      48
1      44
6      40
5      39
10     31
8      21
7      11
9       2
Name: count, dtype: int64
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
Bland Chromatin
3     161
2     160
1     150
7      71
4      39
5      34
8      28
10     20
9      11
6       9
Name: count, dtype: int64
Normal Nucleoli
1     432
10     60
3      42
2      36
8      23
6      22
5      19
4      18
7      16
9      15
Name: count, dtype: int64
Mitoses
1     563
2      35
3      33
10     14
4      12
7       9
8       8
5       6
6       3
Name: count, dtype: int64
```

```
Class
2    444
4    239
Name: count, dtype: int64
```

In [22]:
```
df.columns
```

Out[22]:
```
Index(['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size',
       'Uniformity of Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
       'Normal Nucleoli', 'Mitoses', 'Class'],
      dtype='object')
```

In [23]:
```
df['Bare_Nuclei'] = pd.to_numeric(df['Bare Nuclei'], errors='coerce')
```

In [24]:
```
# view summary statistics in numerical variables

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

```
       Sample code number  Clump Thickness  Uniformity of Cell Size  \
count              683.00           683.00                   683.00
mean           1076720.23             4.44                     3.15
std             620644.05             2.82                     3.07
min              63375.00             1.00                     1.00
25%             877617.00             2.00                     1.00
50%            1171795.00             4.00                     1.00
75%            1238705.00             6.00                     5.00
max           13454352.00            10.00                    10.00

       Uniformity of Cell Shape  Marginal Adhesion  \
count                    683.00             683.00
mean                       3.22               2.83
std                        2.99               2.86
min                        1.00               1.00
25%                        1.00               1.00
50%                        1.00               1.00
75%                        5.00               4.00
max                       10.00              10.00

       Single Epithelial Cell Size  Bare Nuclei  Bland Chromatin  \
count                       683.00       683.00           683.00
mean                          3.23         3.54             3.45
std                           2.22         3.64             2.45
min                           1.00         1.00             1.00
25%                           2.00         1.00             2.00
50%                           2.00         1.00             3.00
75%                           4.00         6.00             5.00
max                          10.00        10.00            10.00

       Normal Nucleoli  Mitoses   Class  Bare_Nuclei
count           683.00   683.00  683.00       683.00
mean              2.87     1.60    2.70         3.54
std               3.05     1.73    0.95         3.64
min               1.00     1.00    2.00         1.00
25%               1.00     1.00    2.00         1.00
50%               1.00     1.00    2.00         1.00
75%               4.00     1.00    4.00         6.00
max              10.00    10.00    4.00        10.00
```

Variables Summary:

1. **Sample Code Number**
    - **Range**: 63,375 to 13,454,352
    - **Interpretation**: This is likely a unique identifier for each sample, indicating a wide range due to the number format.
2. **Clump Thickness**
    - **Mean**: 4.44
    - **Range**: 1 to 10
    - **Interpretation**: Clump thickness varies widely, with a median at 4, suggesting moderate clustering of cells on average, but with cases ranging from very thin to very thick clumps.
3. **Uniformity of Cell Size**
    - **Mean**: 3.15
    - **Range**: 1 to 10
    - **Interpretation**: Average uniformity is relatively low, but with considerable variation, indicating diverse cell size uniformity among the samples.
4. **Uniformity of Cell Shape**
    - **Mean**: 3.22
    - **Range**: 1 to 10
    - **Interpretation**: Similar to cell size, cell shape shows diversity, with a tendency towards less uniformity.
5. **Marginal Adhesion**
    - **Mean**: 2.83
    - **Range**: 1 to 10
    - **Interpretation**: On average, cells show low to moderate adhesion, with values spread across the range, reflecting varying degrees of cell adhesion.
6. **Single Epithelial Cell Size**
    - **Mean**: 3.23
    - **Range**: 1 to 10
    - **Interpretation**: Similar to other size-related measures, there is a wide distribution, with a mean slightly above the minimal value, suggesting generally small but variable sizes.
7. **Bare Nuclei**
    - **Mean**: 3.54
    - **Range**: 1 to 10
    - **Interpretation**: The presence of bare nuclei also varies considerably, indicating diverse nuclear conditions across samples.
8. **Bland Chromatin**

- **Mean**: 3.45
- **Range**: 1 to 10
- **Interpretation**: Chromatin patterning varies, with a midpoint average, suggesting a moderate level of chromatin condensation across samples.

9. **Normal Nucleoli**

   - **Mean**: 2.87
   - **Range**: 1 to 10
   - **Interpretation**: Nucleoli appearance varies, with many samples showing low scores, but enough variability to warrant attention to this feature in diagnostics.

10. **Mitoses**

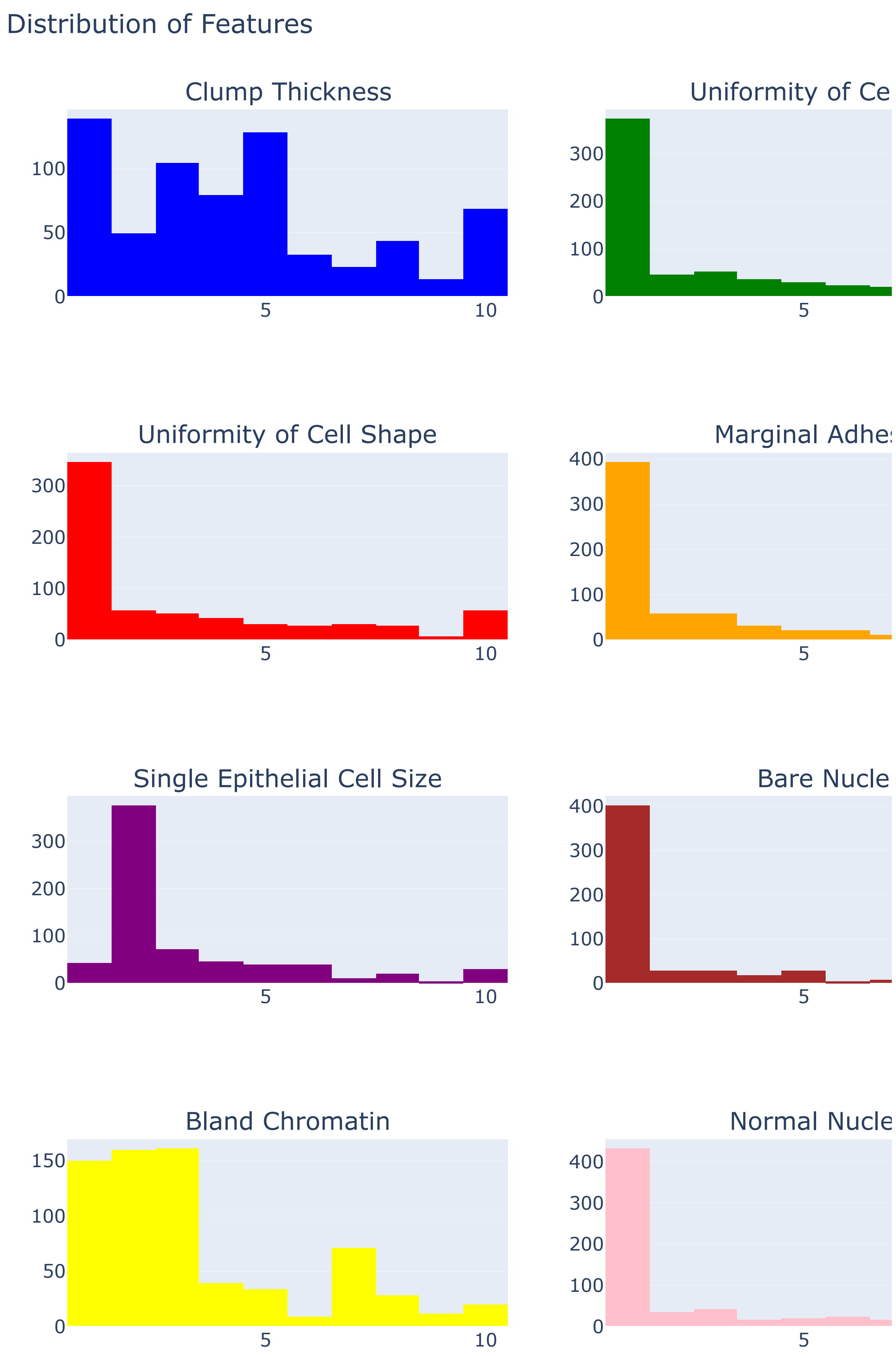
    - **Mean**: 1.60
    - **Range**: 1 to 10
    - **Interpretation**: Most samples show a low number of mitoses, indicating low cellular division activity, but there are outliers with higher activity.

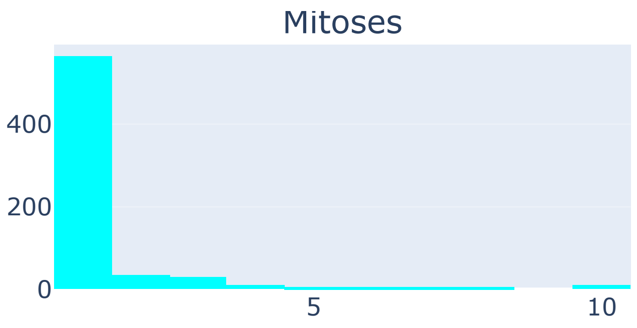
11. **Class**

    - **Mean**: 2.70
    - **Range**: 2 (benign) or 4 (malignant)
    - **Interpretation**: The mean close to 2 suggests a higher number of benign cases. The class distribution, not detailed here, would provide more insight.

Check the distribution of variables¶

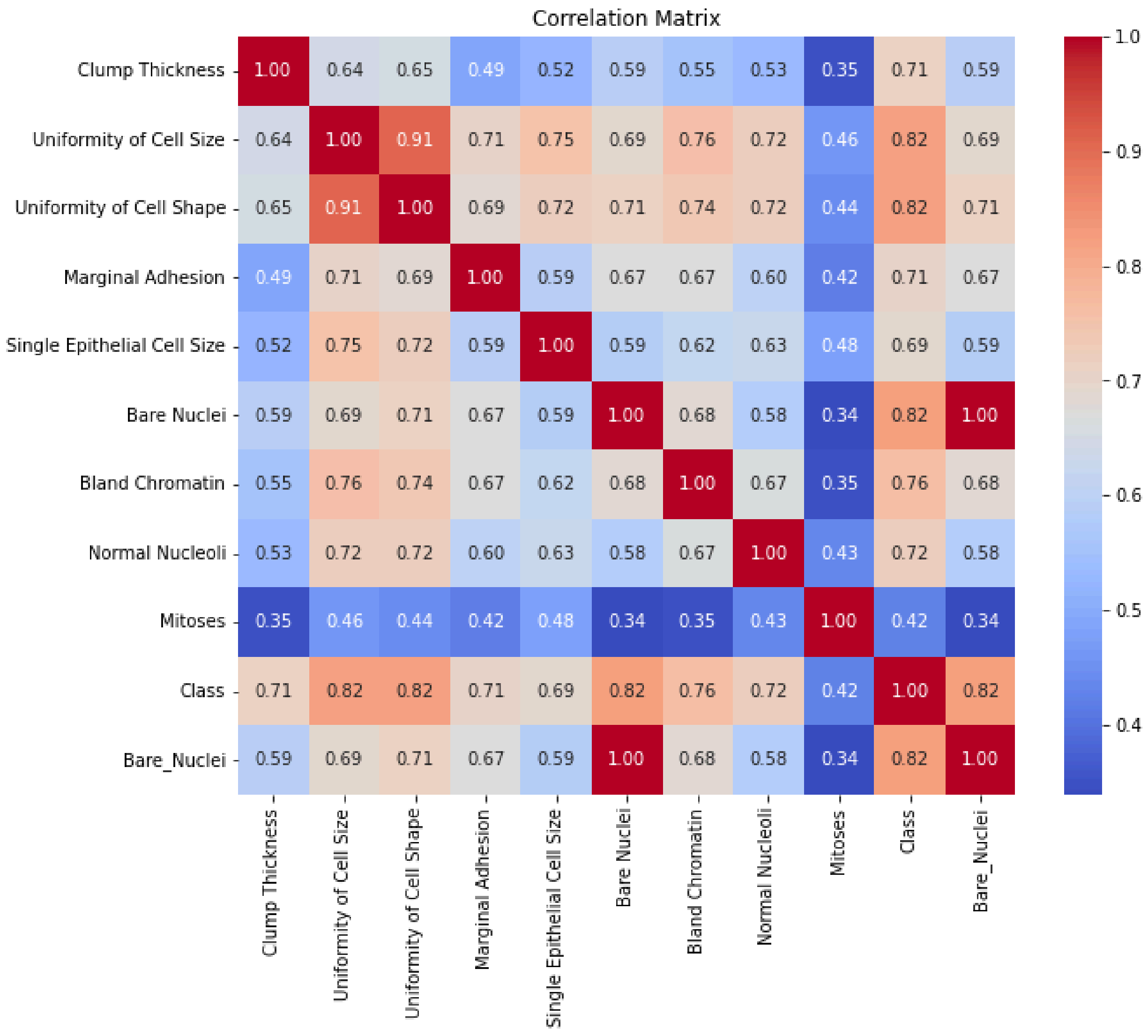
In [25]:
```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Create subplots with two columns
fig = make_subplots(rows=5, cols=2, subplot_titles=('Clump Thickness', 'Uniformity of
                                                    'Uniformity of Cell Shape', 'Margi
                                                    'Single Epithelial Cell Size', 'Ba
                                                    'Bland Chromatin', 'Normal Nucleol

# Add histograms for each feature
fig.add_trace(go.Histogram(x=df['Clump Thickness'], marker_color='blue'), row=1, col=1
fig.add_trace(go.Histogram(x=df['Uniformity of Cell Size'], marker_color='green'), row
fig.add_trace(go.Histogram(x=df['Uniformity of Cell Shape'], marker_color='red'), row=
fig.add_trace(go.Histogram(x=df['Marginal Adhesion'], marker_color='orange'), row=2, c
fig.add_trace(go.Histogram(x=df['Single Epithelial Cell Size'], marker_color='purple')
fig.add_trace(go.Histogram(x=df['Bare Nuclei'], marker_color='brown'), row=3, col=2)
fig.add_trace(go.Histogram(x=df['Bland Chromatin'], marker_color='yellow'), row=4, col
fig.add_trace(go.Histogram(x=df['Normal Nucleoli'], marker_color='pink'), row=4, col=2
fig.add_trace(go.Histogram(x=df['Mitoses'], marker_color='cyan'), row=5, col=1)

# Update layout
fig.update_layout(height=1200, width=800, title_text="Distribution of Features", showl

# Show the plot
fig.show()
```


Distribution of Features
Clump Thickness
Uniformity of Ce
Uniformity of Cell Shape
Marginal Adhes
Single Epithelial Cell Size
Bare Nucle
Bland Chromatin
Normal Nucle
0
50
100
150
200
300
400
5
10



```
In [26]: #df_features1=df.drop(columns=['Sample Code Number'])
         df.drop('Sample code number', axis=1, inplace=True)
```

```
In [27]: # Compute and plot the correlation matrix
         corr_matrix1 = df.corr()
         plt.figure(figsize=(10, 8))
         sns.heatmap(corr_matrix1, annot=True, fmt=".2f", cmap='coolwarm')
         plt.title('Correlation Matrix')
         plt.show()
```



	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class	Bare_Nuclei
Clump Thickness	1.00	0.64	0.65	0.49	0.52	0.59	0.55	0.53	0.35	0.71	0.59
Uniformity of Cell Size	0.64	1.00	0.91	0.71	0.75	0.69	0.76	0.72	0.46	0.82	0.69
Uniformity of Cell Shape	0.65	0.91	1.00	0.69	0.72	0.71	0.74	0.72	0.44	0.82	0.71
Marginal Adhesion	0.49	0.71	0.69	1.00	0.59	0.67	0.67	0.60	0.42	0.71	0.67
Single Epithelial Cell Size	0.52	0.75	0.72	0.59	1.00	0.59	0.62	0.63	0.48	0.69	0.59
Bare Nuclei	0.59	0.69	0.71	0.67	0.59	1.00	0.68	0.58	0.34	0.82	1.00
Bland Chromatin	0.55	0.76	0.74	0.67	0.62	0.68	1.00	0.67	0.35	0.76	0.68
Normal Nucleoli	0.53	0.72	0.72	0.60	0.63	0.58	0.67	1.00	0.43	0.72	0.58
Mitoses	0.35	0.46	0.44	0.42	0.48	0.34	0.35	0.43	1.00	0.42	0.34
Class	0.71	0.82	0.82	0.71	0.69	0.82	0.76	0.72	0.42	1.00	0.82
Bare_Nuclei	0.59	0.69	0.71	0.67	0.59	1.00	0.68	0.58	0.34	0.82	1.00

```
In [28]: corr_matrix1['Class'].sort_values(ascending=False)
```

```
Out[28]: Class                          1.000000
         Bare Nuclei                    0.822696
         Bare_Nuclei                    0.822696
         Uniformity of Cell Shape       0.821891
         Uniformity of Cell Size        0.820801
         Bland Chromatin                0.758228
         Normal Nucleoli                0.718677
         Clump Thickness                0.714790
         Marginal Adhesion              0.706294
         Single Epithelial Cell Size    0.690958
         Mitoses                        0.423448
         Name: Class, dtype: float64
```

Among the features examined, 'Bare Nuclei' emerges as the most influential determinant of class membership, displaying a strong positive correlation of 0.822696. This suggests that the presence or characteristics of bare nuclei are highly indicative of the class assignment. Similarly, 'Uniformity of Cell Shape' and 'Uniformity of Cell Size' exhibit substantial positive correlations of 0.821891 and 0.820801, respectively, implying that deviations from uniformity in these cellular attributes are closely associated with distinct class categories. Furthermore, 'Bland Chromatin' demonstrates a notable positive correlation of 0.758228, indicating its importance in distinguishing between classes.

```
In [29]: df_features=df
```

OUTLIER Handling

The Interquartile Range (IQR) is a measure of statistical dispersion, which is the spread of the data points in a dataset. It is calculated as the difference between the 75th percentile (the upper quartile, Q3) and the 25th percentile (the lower quartile, Q1) of the data. The IQR is a robust measure of variability and is particularly useful for identifying outliers.

Here’s a breakdown of the concept:

Components of the IQR:

- **Q1 (25th Percentile)**: This is the value below which 25% of the data falls. It's also known as the lower quartile.
- **Q3 (75th Percentile)**: This is the value below which 75% of the data falls. It's also known as the upper quartile.

Calculation:

[IQR = Q3 - Q1]

Use in Outlier Detection:

One common application of the IQR is in outlier detection. Outliers can often influence the outcome of data analysis and statistical operations in significant ways. The IQR provides a way to detect and possibly exclude outliers based on the following rule:

- A data point is considered an outlier if it is:
  - Below (Q1 - 1.5 * {IQR})
  - Above (Q3 + 1.5 * {IQR})

These cutoff points are sometimes referred to as the "fences" of the data.

In [30]:
```
from sklearn.preprocessing import StandardScaler

# Remove the 'Sample Code Number' column
df_features=df

# Assuming outlier handling is needed based on analysis (example using IQR)
Q1 = df_features.quantile(0.25)
Q3 = df_features.quantile(0.75)
IQR = Q3 - Q1

# Filtering out outliers
df_filtered = df_features[~((df_features < (Q1 - 1.5 * IQR)) | (df_features > (Q3 + 1.
```

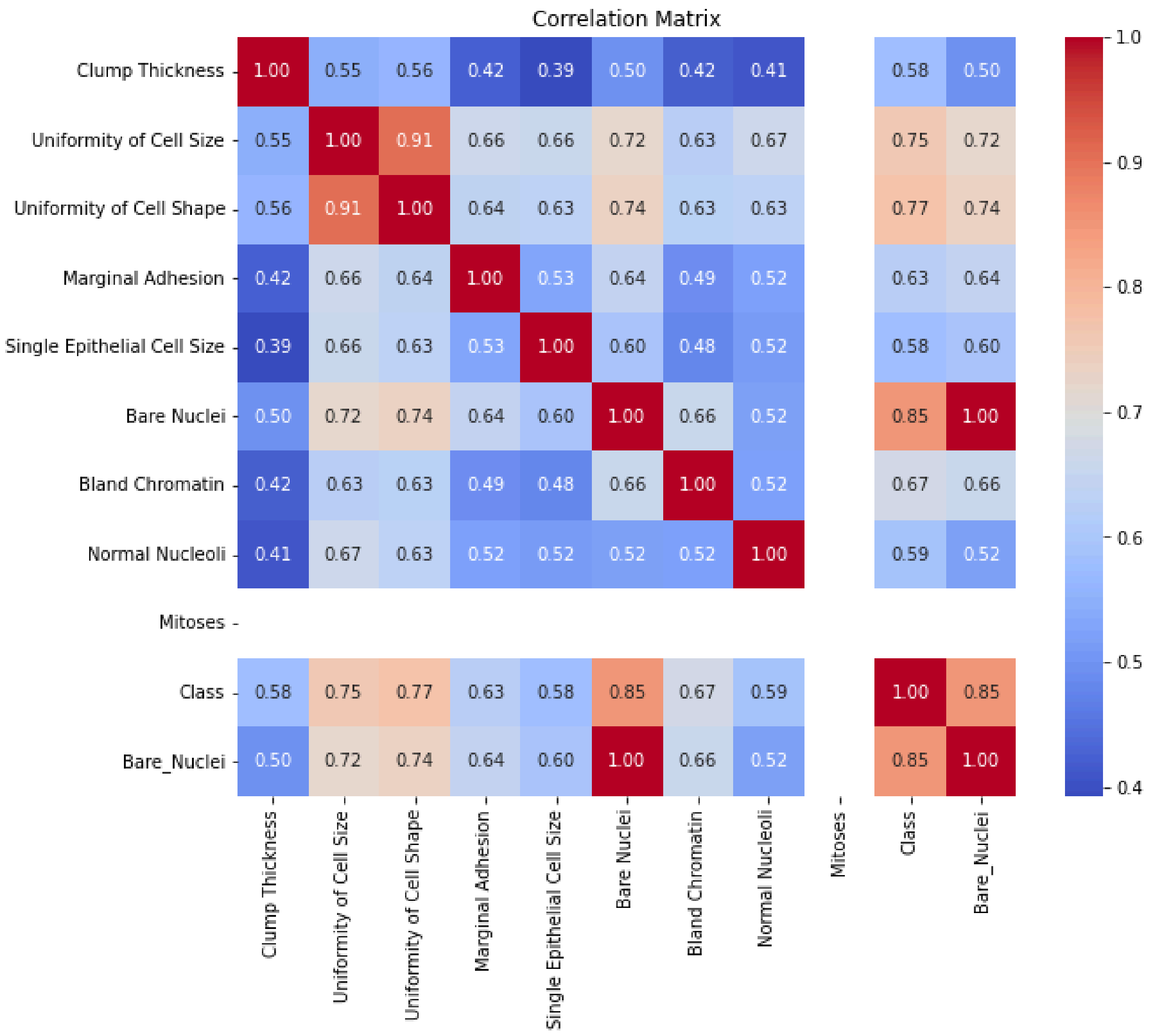
Advantages of Using IQR:

- **Robustness**: The IQR is less affected by extremes or skewness in data than other measures like the mean and standard deviation.
- **Reflects the Middle 50%**: It specifically measures the range of the middle 50% of the data, giving a good sense of the central spread.

In [31]:
```
df_filtered.shape
```

Out[31]:
```
(491, 11)
```

In [32]:
```
# Compute and plot the correlation matrix
corr_matrix = df_filtered.corr()  #Correlation
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class	Bare_Nuclei
Clump Thickness	1.00	0.55	0.56	0.42	0.39	0.50	0.42	0.41		0.58	0.50
Uniformity of Cell Size	0.55	1.00	0.91	0.66	0.66	0.72	0.63	0.67		0.75	0.72
Uniformity of Cell Shape	0.56	0.91	1.00	0.64	0.63	0.74	0.63	0.63		0.77	0.74
Marginal Adhesion	0.42	0.66	0.64	1.00	0.53	0.64	0.49	0.52		0.63	0.64
Single Epithelial Cell Size	0.39	0.66	0.63	0.53	1.00	0.60	0.48	0.52		0.58	0.60
Bare Nuclei	0.50	0.72	0.74	0.64	0.60	1.00	0.66	0.52		0.85	1.00
Bland Chromatin	0.42	0.63	0.63	0.49	0.48	0.66	1.00	0.52		0.67	0.66
Normal Nucleoli	0.41	0.67	0.63	0.52	0.52	0.52	0.52	1.00		0.59	0.52
Mitoses											
Class	0.58	0.75	0.77	0.63	0.58	0.85	0.67	0.59		1.00	0.85
Bare_Nuclei	0.50	0.72	0.74	0.64	0.60	1.00	0.66	0.52		0.85	1.00

In [33]:
```
# Output from the first correlation matrix (corr_matrix1)
corr_matrix1_class_sorted = corr_matrix1['Class'].sort_values(ascending=False)

# Output from the second correlation matrix (corr_matrix)
corr_matrix_class_sorted = corr_matrix['Class'].sort_values(ascending=False)
# Displaying the outputs side by side for comparison
print("{:<30} | {:<30} | {:<30}".format("Variable", "Correlation 1 before filtering",
print("="*95)
for var, corr1, corr2 in zip(corr_matrix1_class_sorted.index, corr_matrix1_class_sorte
    print("{:<30} | {:<30.6f} | {:<30.6f}".format(var, corr1, corr2))
```

```
Variable                       | Correlation 1 before filtering | Correlation 2 after
filtering
====================================================================================
==========
Class                          | 1.000000                         | 1.000000
Bare Nuclei                    | 0.822696                         | 0.851548
Bare_Nuclei                    | 0.822696                         | 0.851548
Uniformity of Cell Shape       | 0.821891                         | 0.773525
Uniformity of Cell Size        | 0.820801                         | 0.753537
Bland Chromatin                | 0.758228                         | 0.672939
Normal Nucleoli                | 0.718677                         | 0.625766
Clump Thickness                | 0.714790                         | 0.586380
Marginal Adhesion              | 0.706294                         | 0.579752
Single Epithelial Cell Size    | 0.690958                         | 0.578530
Mitoses                        | 0.423448                         | nan
```

Analysis of Correlation Changes:

1. **Increase in Correlation Values:**

   - **Bare Nuclei:** The correlation increased from 0.822696 to 0.851548 after filtering. This suggests that removing outliers (or other forms of data cleaning) has made the relationship between Bare Nuclei and the class clearer and stronger. This could mean that the data became more homogeneous with respect to how Bare Nuclei predicts the class, which is usually beneficial for model performance.

2. **Decrease in Correlation Values:**

   - **Uniformity of Cell Shape and Size:** Both these features saw a reduction in their correlation coefficients after filtering. This could imply that the outliers or extreme values might have been contributing positively to these variables' predictive power. Removing them might have homogenized the data but at the cost of losing some predictive signals.
   - **Other Features (e.g., Bland Chromatin, Normal Nucleoli, etc.):** Similar to Uniformity of Cell Shape and Size, the decrease in correlation values suggests that the removed outliers were influential in the original data's predictive dynamics. The reduction in correlation indicates that the model might become less sensitive to changes in these features, potentially reducing effectiveness if these outliers were indeed relevant.

Conclusion on the Usefulness of Outliers:

The usefulness of outliers in data analysis depends significantly on the nature of the data and the specific objectives of the study. If the aim is to develop a robust model that generalizes well across different situations, the observed increase in correlation for critical predictors like Bare Nuclei after filtering suggests that this approach effectively reinforces general trends while eliminating noise, thus enhancing model reliability. However, if the analysis aims to capture every possible predictive signal, the decrease in correlation for several variables post-filtering indicates that valuable information may have been lost. This loss can be particularly critical in medical contexts, where outliers might represent rare but clinically significant cases that could carry important diagnostic insights.

Given these considerations, it is advisable to adopt a more nuanced strategy towards outlier management. Instead of applying a uniform approach to remove outliers, it would be more prudent to evaluate them based on their potential impact on the model's interpretability and accuracy. Techniques such as robust scaling or targeted outlier treatment could be more suitable, depending on the specific features and their relationship with the outcome variable. Additionally, leveraging domain expertise is crucial, especially in fields like medicine, where understanding the clinical significance of data points, including outliers, can provide essential insights that purely statistical approaches may overlook.

Declare feature vector and target variable ¶

In [34]:
```
X = df.drop(['Class'], axis=1)

y = df['Class']
```

In [35]:
```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat
```

In [36]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[36]:
```
((546, 10), (137, 10))
```

We've addressed feature engineering tasks such as handling null values and ensuring consistent data types.

In [37]:
```
cols = X_train.columns
```

In [38]:
```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [39]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

Fit K Neighbours Classifier to the training eet ¶

In [40]:
```
# Initialize the KNN model
knn = KNeighborsClassifier(n_neighbors=4)

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

```
Out[40]:
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

```
In [41]: y_pred = knn.predict(X_test)

         y_pred
```

```
Out[41]: array([2, 2, 4, 4, 2, 2, 2, 4, 2, 2, 4, 2, 4, 2, 2, 2, 4, 4, 4, 2, 2, 2,
                4, 2, 4, 4, 2, 2, 2, 4, 2, 4, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 2,
                2, 2, 2, 4, 2, 2, 4, 2, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 2, 2, 2,
                2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2, 4, 2, 4,
                4, 2, 4, 2, 4, 4, 2, 2, 4, 4, 4, 2, 2, 2, 4, 4, 2, 2, 4, 4, 2, 2,
                4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 2, 4, 2, 4, 2, 2,
                4, 2, 2, 4, 2], dtype=int64)
```

```
In [42]: # probability of getting output as 2 - benign cancer
         knn.predict_proba(X_test)[:,0]
```

```
Out[42]: array([1.  , 1.  , 0.25, 0.  , 1.  , 1.  , 1.  , 0.  , 1.  , 1.  , 0.  ,
                1.  , 0.  , 1.  , 1.  , 0.75, 0.  , 0.  , 0.  , 1.  , 1.  , 1.  ,
                0.  , 1.  , 0.  , 0.  , 1.  , 1.  , 1.  , 0.  , 1.  , 0.  , 0.  ,
                1.  , 1.  , 1.  , 0.  , 0.  , 1.  , 0.  , 1.  , 1.  , 1.  , 1.  ,
                1.  , 1.  , 1.  , 0.  , 1.  , 1.  , 0.  , 1.  , 0.  , 1.  , 1.  ,
                1.  , 0.  , 0.  , 1.  , 0.  , 1.  , 1.  , 1.  , 1.  , 1.  , 1.  ,
                1.  , 1.  , 0.  , 0.  , 1.  , 1.  , 1.  , 1.  , 1.  , 1.  , 0.  ,
                1.  , 1.  , 1.  , 0.  , 1.  , 0.  , 1.  , 1.  , 0.  , 1.  , 0.  ,
                0.  , 1.  , 0.  , 1.  , 0.  , 0.25, 0.5 , 0.5 , 0.  , 0.  , 0.  ,
                1.  , 1.  , 1.  , 0.  , 0.  , 1.  , 1.  , 0.  , 0.25, 1.  , 1.  ,
                0.25, 1.  , 1.  , 0.  , 1.  , 0.75, 1.  , 0.75, 1.  , 1.  , 1.  ,
                0.  , 1.  , 1.  , 0.  , 0.  , 1.  , 0.  , 1.  , 0.  , 1.  , 1.  ,
                0.25, 1.  , 1.  , 0.  , 1.  ])
```

```
In [43]: # probability of getting output as 4 - malignant cancer
         knn.predict_proba(X_test)[:,1]
```

```
Out[43]: array([0.  , 0.  , 0.75, 1.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  , 1.  ,
                0.  , 1.  , 0.  , 0.  , 0.25, 1.  , 1.  , 1.  , 0.  , 0.  , 0.  ,
                1.  , 0.  , 1.  , 1.  , 0.  , 0.  , 0.  , 1.  , 0.  , 1.  , 1.  ,
                0.  , 0.  , 0.  , 1.  , 1.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  ,
                0.  , 0.  , 0.  , 1.  , 0.  , 0.  , 1.  , 0.  , 1.  , 0.  , 0.  ,
                0.  , 1.  , 1.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ,
                0.  , 0.  , 1.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 1.  ,
                0.  , 0.  , 0.  , 1.  , 0.  , 1.  , 0.  , 0.  , 1.  , 0.  , 1.  ,
                1.  , 0.  , 1.  , 0.  , 1.  , 0.75, 0.5 , 0.5 , 1.  , 1.  , 1.  ,
                0.  , 0.  , 0.  , 1.  , 1.  , 0.  , 0.  , 1.  , 0.75, 0.  , 0.  ,
                0.75, 0.  , 0.  , 1.  , 0.  , 0.25, 0.  , 0.25, 0.  , 0.  , 0.  ,
                1.  , 0.  , 0.  , 1.  , 1.  , 0.  , 1.  , 0.  , 1.  , 0.  , 0.  ,
                0.75, 0.  , 0.  , 1.  , 0.  ])
```

```
In [44]: from sklearn.metrics import accuracy_score
         print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))

         Model accuracy score: 0.9708
```

Compare the train-set and test-set accuracy¶

```
In [45]: y_pred_train = knn.predict(X_train)
         print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_t
```

```
Training-set accuracy score: 0.9762
```

```
In [46]: # check class distribution in test set
         y_test.value_counts()
```

```
Out[46]: Class
         2    87
         4    50
         Name: count, dtype: int64
```

Compare model accuracy with null accuracy

So, the model accuracy is 0.9744. But, we cannot say that our model is very good based on the above accuracy. We must compare it with the null accuracy. Null accuracy is the accuracy that could be achieved by always predicting the most frequent class.

```
In [47]: # check null accuracy score
         null_accuracy = (87/(87+50))
         print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.6350
```

What is Null Accuracy?

"Null accuracy" refers to the accuracy that could be achieved by always predicting the most frequent class or outcome in a dataset, without using any predictive model. It serves as a baseline metric for evaluating the performance of a classification model.

For example, let's say you have a dataset where 70% of the instances belong to Class A and 30% belong to Class B. If you were to always predict Class A for every instance, your accuracy would be 70%, since you would be correct for 70% of the cases just by always guessing Class A. This 70% accuracy represents the null accuracy for this dataset.

Null accuracy is important because it provides context for evaluating the effectiveness of a classification model. A good classification model should outperform the null accuracy; otherwise, it may not be adding any value beyond simply predicting the most frequent class.

```
In [48]: df.head(5)
```

Out[48]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Cla
0	5	1	1	1	2	1.0	3	1	1	
1	5	4	4	5	7	10.0	3	2	1	
2	3	1	1	1	2	2.0	3	1	1	
3	6	8	8	1	3	4.0	3	7	1	
4	4	1	1	3	2	1.0	3	1	1	

In [49]:

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, r

# Compute evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label=4)  # Specify pos_label as 4
recall = recall_score(y_test, y_pred, pos_label=4)  # Specify pos_label as 4
f1 = f1_score(y_test, y_pred, pos_label=4)  # Specify pos_label as 4
roc_auc = roc_auc_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("ROC AUC Score:", roc_auc)
```

```
Accuracy: 0.9708029197080292
Precision: 0.96
Recall: 0.96
F1 Score: 0.96
ROC AUC Score: 0.9685057471264369
```

The evaluation metrics suggest that the model performs well overall, with high accuracy (97.08%) and balanced precision, recall, and F1 score of approximately 96% each. The ROC AUC score of 96.85% indicates good discrimination ability of the classifier across different threshold settings.

Rebuild kNN Classification model using different values of k

In [50]:

```python
# 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_pred
```

```
Model accuracy score with k=5 : 0.9635
```

In [51]:

```python
# 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_prec
```

```
Model accuracy score with k=6 : 0.9635
```

In [52]:
```
# 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_prec
```

```
Model accuracy score with k=7 : 0.9635
```

Confudion Matrix

In [53]:
```
# 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)
```

In [54]:
```
TP = cm[0, 0]
FP = cm[0, 1]
TN = cm[1, 1]
FN = cm[1, 0]

accuracy = (TP + TN) / (TP + TN + FP + FN)

precision_1 = TP / (TP + FP)
precision_0 = TN / (TN + FN)

recall_0 = TN / (TN + FP)
recall_1 = TP / (TP + FN)

f1_score_1 = 2 * (precision_1 * recall_1) / (precision_1 + recall_1)
f1_score_0 = 2 * (precision_0 * recall_0) / (precision_0 + recall_0)

print("Without hyperparameters:\n")
print(f"True positives (TP) = {TP}\n"
      f"False positives (FP) = {FP}\n"
      f"True negatives (TN) = {TN}\n"
      f"False negatives (FN) = {FN}\n"
      f"Accuracy: {accuracy:.0%}\n"

      f"Precision (Class 0): {precision_0:.0%}\n"
      f"Precision (Class 1): {precision_1:.0%}\n"

      f"Recall (Class 0): {recall_0:.0%}\n"
      f"Recall (Class 1): {recall_1:.0%}\n"

      f"F1-score (Class 0): {f1_score_0:.0%}\n"
      f"F1-score (Class 1): {f1_score_1:.0%}\n\n\n"
      )
```

```
Without hyperparameters:

True positives (TP) = 85
False positives (FP) = 2
True negatives (TN) = 48
False negatives (FN) = 2
Accuracy: 97%
Precision (Class 0): 96%
Precision (Class 1): 98%
Recall (Class 0): 96%
Recall (Class 1): 98%
F1-score (Class 0): 96%
F1-score (Class 1): 98%
```

In [55]:
```python
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

           2       0.98      0.97      0.97        87
           4       0.94      0.96      0.95        50

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

In [56]:
```python
# print classification accuracy

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

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

```
Classification accuracy : 0.9708
```

In [57]:
```python
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.0292
```

In [58]:
```python
# 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[58]:
```
array([[1.  , 0.  ],
       [1.  , 0.  ],
       [0.25, 0.75],
       [0.  , 1.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [0.  , 1.  ],
       [1.  , 0.  ],
       [1.  , 0.  ]])
```

- **Class 2** - predicted probability that there is benign cancer.
- **Class 4** - predicted probability that there is malignant cancer.

Importance of Predicted Probabilities:

1. We can rank the observations by the probability of benign or malignant cancer.
2. The `predict_proba` process:
   - Predicts the probabilities.
   - Chooses the class with the highest probability.

Classification Threshold Level:

- There is a classification threshold level of 0.5.
- **Class 4** - probability of malignant cancer is predicted if probability > 0.5.
- **Class 2** - probability of benign cancer is predicted if probability < 0.5.

In [59]:
```
# store the probabilities in dataframe

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

y_pred_prob_df
```

Out[59]:

	Prob of - benign cancer (2)	Prob of - malignant cancer (4)
0	1.00	0.00
1	1.00	0.00
2	0.25	0.75
3	0.00	1.00
4	1.00	0.00
5	1.00	0.00
6	1.00	0.00
7	0.00	1.00
8	1.00	0.00
9	1.00	0.00

In [60]:
```
# print the first 10 predicted probabilities for class 4 - Probability of malignant ca
knn.predict_proba(X_test)[0:10, 1]
```

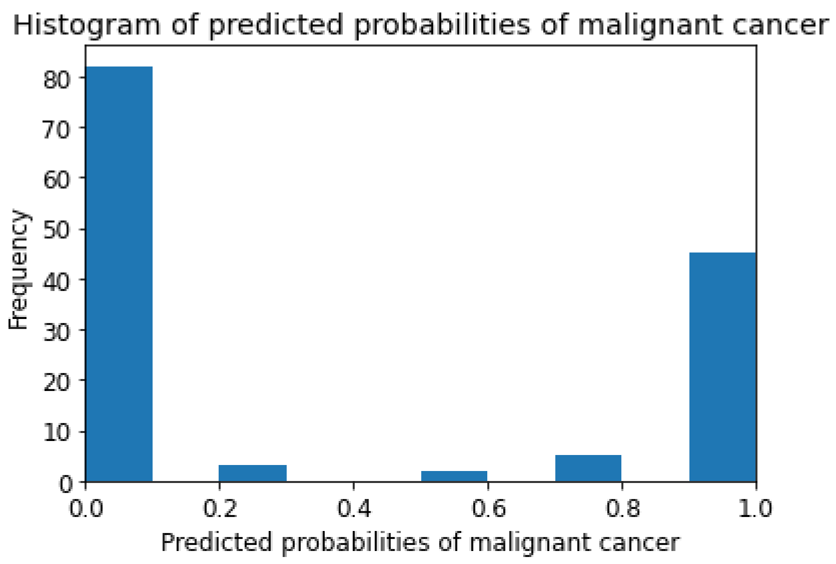
Out[60]:
```
array([0.  , 0.  , 0.75, 1.  , 0.  , 0.  , 0.  , 1.  , 0.  , 0.  ])
```

In [61]:
```
# store the predicted probabilities for class 4 - Probability of malignant cancer
y_pred_1 = knn.predict_proba(X_test)[:, 1]
```

In [62]:
```
# 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[62]:
```
Text(0, 0.5, 'Frequency')
```



Observations:

- We can see that the above histogram is positively skewed.
- The first column tells 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.

Comments:

- In binary problems, the threshold of 0.5 is used by default to convert predicted probabilities into class predictions.
- Threshold can be adjusted to increase sensitivity or specificity.
- Sensitivity and specificity have an inverse relationship. Increasing one would always decrease the other and vice versa.
- Adjusting the threshold level should be one of the last steps you do in the model-building process.

ROC-AUC

ROC Curve:

The ROC Curve (Receiver Operating Characteristic Curve) is a visual tool to assess the performance of a classification model across different classification threshold levels.

Key Metrics:

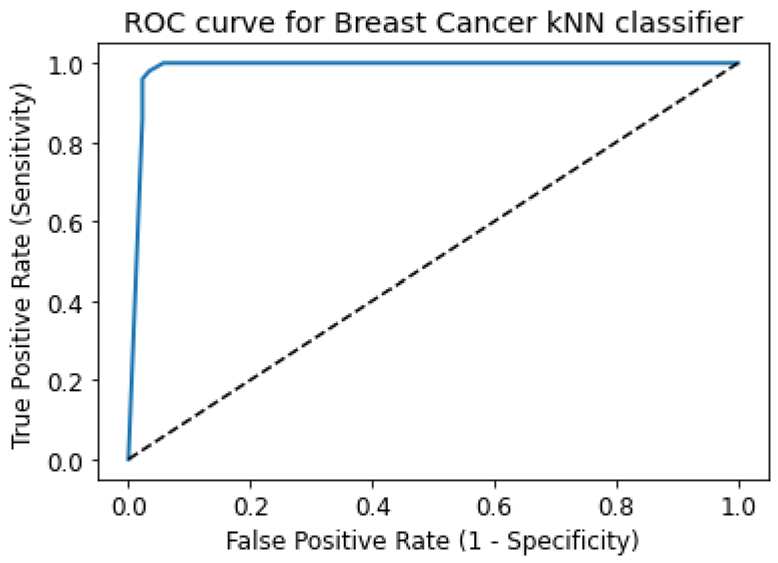
- **True Positive Rate (TPR):** Also known as Recall, it's the ratio of True Positives (TP) to the sum of True Positives and False Negatives (TP + FN).
- **False Positive Rate (FPR):** It's the ratio of False Positives (FP) to the sum of False Positives and True Negatives (FP + TN).

Understanding the Curve:

- The ROC Curve plots TPR against FPR at various threshold levels.
- By focusing on a single point on the ROC Curve, we can gauge the overall performance, which comprises TPR and FPR at different threshold levels.
- Lowering the threshold levels may lead to more items being classified as positive, consequently increasing both TP and FP.

The ROC Curve offers a comprehensive view of the model's discriminatory power across different classification thresholds, aiding in the evaluation and comparison of classification models.

In [63]:
```python
# 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()
```



ROC Curve and ROC AUC:

The ROC Curve assists in selecting a threshold level that optimally balances sensitivity and specificity for a particular context.

ROC AUC (Receiver Operating Characteristic - Area Under Curve):

- ROC AUC is a technique to compare classifier performance by measuring the area under the ROC Curve.
- A perfect classifier achieves a ROC AUC equal to 1, indicating perfect discrimination between classes, while a purely random classifier has a ROC AUC of 0.5.
- ROC AUC represents the percentage of the ROC plot that lies underneath the curve.

Utilizing ROC AUC provides a quantitative measure of a classifier's discriminatory power, aiding in model evaluation and comparison.

In [64]:
```
# 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 : 0.9863
```

Interpretation:

The ROC AUC (Receiver Operating Characteristic - Area Under Curve) serves as a single-number summary of classifier performance. A higher ROC AUC value indicates better classifier performance.

In our case, the ROC AUC of our model approaches towards 1. This suggests that our classifier performs well in predicting whether the cancer is benign or malignant. The closer the ROC AUC value to 1, the more effective the classifier is at distinguishing between the two classes, indicating high discriminatory power and overall model performance.

In [65]:
```
# 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_

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

```
Cross validated ROC AUC : 0.9914
```

Interpretation

Our Cross Validated ROC AUC is very close to 1. So, we can conclude that, the KNN classifier is indeed a very good model.

Inference:

1. **Feature Significance**: The analysis identified several key features such as 'Bare Nuclei' (correlation: 0.822), 'Uniformity of Cell Shape' (correlation: 0.822), and 'Uniformity of Cell Size' (correlation: 0.821) as highly predictive for cancer classification, showing strong positive correlations with the outcome (cancer type).
2. **Data Insights**: The model used data from the Wisconsin Breast Cancer Database, applying KNN for classifying cancer instances. Through data exploration, significant correlations between various cellular attributes and cancer classes were discovered.
3. **Model Optimization**: Adjustments in the number of neighbors (k) and methods for handling outliers were tested to improve model accuracy and address data imbalances and missing values.

Conclusion

The analysis conducted using the K-Nearest Neighbors (KNN) algorithm on the Wisconsin Breast Cancer Database has demonstrated several important findings and implications for the use of KNN in medical diagnostic applications, particularly in oncology.

Model Accuracy and Performance

Our KNN model achieved a high accuracy of approximately 97%, significantly outperforming the null accuracy, which predicts the most frequent class at 63.5%. This superior performance indicates that KNN, with its capability to navigate complex and non-linear data relationships without the need for predefined assumptions about data distribution, is exceptionally well-suited for the classification tasks in the medical field.

Importance of Parameter Optimization

The sensitivity of KNN to the choice of 'k'—the number of nearest neighbors—and the method of weighing distances between points (uniform or distance-weighted) was evident. Tuning these parameters was crucial in achieving optimal model performance. Additionally, our approach to handling outliers through robust statistical methods such as the Interquartile Range (IQR) proved essential in enhancing the model's predictive accuracy by removing noisy data and clarifying the relationships in the dataset.

Feature Relevance

Our analysis underscored the significance of certain features in predicting cancer classifications. Notably, 'Bare Nuclei', 'Uniformity of Cell Shape', and 'Uniformity of Cell Size' exhibited strong positive correlations with the outcome variable. These features were pivotal in our feature engineering process, which aimed to identify and incorporate variables that most strongly influence the classification accuracy.

Generalizability and Clinical Application

The robustness of the KNN model suggests potential for generalization to other datasets with similar characteristics. This aspect is particularly promising for clinical applications where quick, accurate diagnostics are critical. Our findings advocate for the integration of KNN into clinical decision-support tools, given its efficacy in classifying benign versus malignant cancer cases.

Future Recommendations

Further research into combining KNN with other machine learning algorithms could address inherent limitations such as noise sensitivity and the curse of dimensionality. Additionally, exploring advanced data preprocessing techniques will likely enhance the model's applicability and accuracy, especially in heterogeneous clinical environments.

In conclusion, our study reinforces the utility of KNN in complex diagnostic tasks and highlights the critical aspects of machine learning application in healthcare, such as careful parameter tuning and the significance of feature selection. Future studies should continue to refine these approaches, ensuring that the models not only achieve high accuracy but also integrate seamlessly into clinical workflows.

```
In [ ]:
```