Breast Cancer Prediction

Importing the necessary libraries

In [1]:

```
# Data Analysis Libraries

import pandas as pd

# Visualisation libraries
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objs as go
import plotly.io as pio

# Library for Splitting Dataset
from sklearn.model_selection import train_test_split

#Library for Implementing models
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# Library for metrics generation
from sklearn.metrics import accuracy_score
```

Importing the Dataset and Visualising it

In [2]:
```
cancer = pd.read_csv(r'C:\Users\Tanmayee\OneDrive\Documents\Personal\O
cancer
```

Out[2]:

	patient_id	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhes
0	1000025	5.0	1.0	1	
1	1002945	5.0	4.0	4	
2	1015425	3.0	1.0	1	
3	1016277	6.0	8.0	8	
4	1017023	4.0	1.0	1	
...	...	...	...	...	
694	776715	3.0	1.0	1	
695	841769	2.0	1.0	1	
696	888820	5.0	10.0	10	
697	897471	4.0	8.0	6	
698	897471	4.0	8.0	8	

699 rows × 12 columns

Deleting the columns from the dataset that are no more required

In [3]:
```
cancer = cancer.drop(columns=['patient_id', 'doctor_name'], axis=1)
cancer
```

Out[3]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_
0	5.0	1.0	1	1	
1	5.0	4.0	4	5	
2	3.0	1.0	1	1	
3	6.0	8.0	8	1	
4	4.0	1.0	1	3	
...	...	...	...	...	
694	3.0	1.0	1	1	
695	2.0	1.0	1	1	
696	5.0	10.0	10	3	
697	4.0	8.0	6	4	
698	4.0	8.0	8	5	

699 rows × 10 columns

The value counts() method is used to first calculate the counts of each distinct value in the 'cell size uniformity' column. Using the idxmax() function, it then returns the index (i.e., the value) corresponding to the highest count. This index corresponds to the first value in the array returned by the mode() function, which is the mode of the 'cell size uniformity' column.

In [4]:
```
r1 = cancer['cell_size_uniformity'].value_counts().idxmax()
r1
```

Out[4]: 1.0

The 'cell size uniformity' column's mode is first calculated using the mode() function, and then the iloc method is used to choose the first value in the resultant array. The fillna() function, with the inplace argument set to True, utilises this mode value to replace missing values in the 'cell size uniformity' column. This is comparable to the supplied code snippet, except the mode value is computed inline in the fillna() function call.

In [5]:
```
cancer['cell_size_uniformity'].fillna(cancer['cell_size_uniformity'].m
```

In [6]:
```
r2 = cancer.isna().sum()
r2
```

Out[6]:
```
clump_thickness          1
cell_size_uniformity     0
cell_shape_uniformity    0
marginal_adhesion        0
single_ep_cell_size      0
bare_nuclei              2
bland_chromatin          4
normal_nucleoli          1
mitoses                  0
class                    0
dtype: int64
```

Fills missing values in the 'clump_thickness', 'bare_nuclei', 'bland_chromatin', and 'normal_nucleoli' columns of the 'cancer' dataframe with the mode of each respective column.

In [7]:
```
cols = ['clump_thickness', 'bare_nuclei', 'bland_chromatin', 'normal_n
cancer[cols] = cancer[cols].apply(lambda x: x.fillna(x.mode()[0]))
```

In [8]:
```
r3 = cancer.isna().sum()
r3
```

Out[8]:
```
clump_thickness          0
cell_size_uniformity     0
cell_shape_uniformity    0
marginal_adhesion        0
single_ep_cell_size      0
bare_nuclei              0
bland_chromatin          0
normal_nucleoli          0
mitoses                  0
class                    0
dtype: int64
```

Counting total number of patients registered in a hospital

In [9]:
```
total_patients = len(cancer)
total_patients
```

Out[9]:
```
699
```

Seperating the normal patients and Patients suffering from Cancer

In [10]:
```
normal_patients = cancer["class"].value_counts()[0]
normal_patients
```

Out[10]:
```
458
```

In [11]:
```
breast_cancer_patients = cancer["class"].value_counts()[1]
breast_cancer_patients
```

Out[11]: 241

Using the formula to find total number of people suffering from cancer

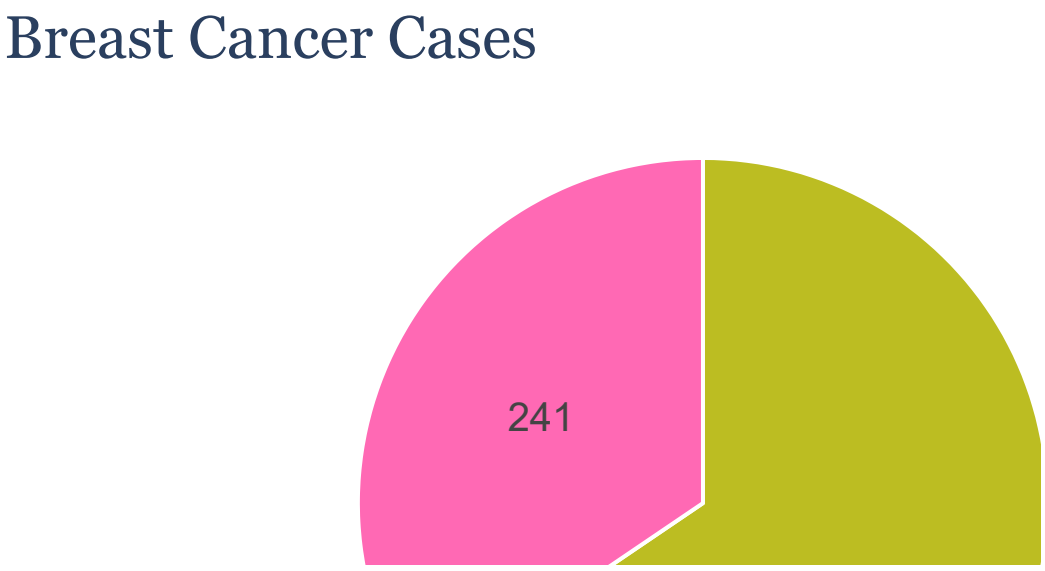
In [12]:
```
percentage_breast_cancer = round((breast_cancer_patients / total_patie
print("Percentage of people with breast cancer: ", percentage_breast_c
```

```
Percentage of people with breast cancer:  34.48 %
```

Plotting the Pie Chart to show the number of people suffering from breast cancer and the normal people

In [13]:

```
labels = ['Without breast cancer', 'With breast cancer']
sizes = [normal_patients, breast_cancer_patients]

fig1 = go.Figure(data=[go.Pie(labels=labels, values=sizes)])

fig1.update_traces(hoverinfo='label+percent', textinfo='value', textfo
                   marker=dict(colors=['#bcbd22', '#FF69B4'], line=dict

fig1.update_layout(title='Breast Cancer Cases', title_font_size=30, ti
                   legend=dict(title='Cases', font=dict(size=20)),
                   font=dict(family='Arial', size=20))

fig1.show()
```



The values "benign" and "malignant" are mapped to 0 and 1, respectively, in a dictionary class mapping that we construct. The values in the "class" column of the "cancer" dataframe are then replaced with the equivalent values from the class mapping dictionary using the map() function.

In [14]:

```
class_mapping = {'benign': 0, 'malignant': 1}
cancer['class'] = cancer['class'].map(class_mapping)
```

Code replaces insufficient data in the 'bare nuclei' column of the 'cancer' dataframe with the mode of the column using the fillna() method.

In [15]:

```
mode = cancer['bare_nuclei'].mode()[0]
cancer['bare_nuclei'] = cancer['bare_nuclei'].replace("?", mode)
print(cancer)
```

```
     clump_thickness  cell_size_uniformity  cell_shape_uniformity  \
0                5.0                   1.0                      1
1                5.0                   4.0                      4
2                3.0                   1.0                      1
3                6.0                   8.0                      8
4                4.0                   1.0                      1
..               ...                   ...                    ...
694              3.0                   1.0                      1
695              2.0                   1.0                      1
696              5.0                  10.0                     10
697              4.0                   8.0                      6
698              4.0                   8.0                      8

     marginal_adhesion  single_ep_cell_size bare_nuclei  bland_chromatin
\
0                    1                    2           1              3.0
1                    5                    7          10              3.0
2                    1                    2           2              3.0
3                    1                    3           4              3.0
4                    3                    2           1              3.0
..                 ...                  ...         ...              ...
694                  1                    3           2              1.0
695                  1                    2           1              1.0
696                  3                    7           3              8.0
697                  4                    3           4             10.0
698                  5                    4           5             10.0

     normal_nucleoli  mitoses  class
0                1.0        1      0
1                2.0        1      0
2                1.0        1      0
3                7.0        1      0
4                1.0        1      0
..               ...      ...    ...
694              1.0        1      0
695              1.0        1      0
696             10.0        2      1
697              6.0        1      1
698              4.0        1      1

[699 rows x 10 columns]
```

This line of code converts the 'bare_nuclei' column of the 'cancer' dataframe from a string data type to an integer data type using the astype() method.

Creating a new dataframe that contains first 9 columns to divide it into train and test

In [16]:
```
cr_x = cancer.loc[:, cancer.columns[:9]]
cr_x
```

Out[16]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_
0	5.0	1.0	1	1	
1	5.0	4.0	4	5	
2	3.0	1.0	1	1	
3	6.0	8.0	8	1	
4	4.0	1.0	1	3	
...	...	...	...	...	
694	3.0	1.0	1	1	
695	2.0	1.0	1	1	
696	5.0	10.0	10	3	
697	4.0	8.0	6	4	
698	4.0	8.0	8	5	

699 rows × 9 columns

Creating a new dataframe that contains last column of dataset to divide it into train and test

In [17]:
```
cr_y = cancer['class']
cr_y
```

Out[17]:
```
0      0
1      0
2      0
3      0
4      0
      ..
694    0
695    0
696    1
697    1
698    1
Name: class, Length: 699, dtype: int64
```

Dividing the dataset into 80:20 ratio for Train and Test respectively

In [18]:
```
split = train_test_split(cr_x, cr_y, test_size=0.2, random_state=1)
crx_train, crx_test, cry_train, cry_test = split
```

In [19]:
```
1 crx_train
```

Out[19]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_
617	1.0	1.0	1	1	
107	1.0	6.0	8	10	
17	4.0	1.0	1	1	
441	5.0	2.0	2	4	
365	2.0	1.0	1	1	
...	...	...	...	...	
144	2.0	1.0	1	1	
645	3.0	1.0	1	1	
72	1.0	3.0	3	2	
235	3.0	1.0	4	1	
37	6.0	2.0	1	1	

559 rows × 9 columns

Logistic Regression

In [20]:
```
1 logi_reg = LogisticRegression()
2 logi_reg.fit(crx_train, cry_train)
3 LR_p = logi_reg.predict(crx_test)
```

In [21]:
```
1 LRaccuracy = accuracy_score(cry_test, LR_p)
2 print("Accuracy on Logistic Regression:", LRaccuracy)
```

```
Accuracy on Logistic Regression: 0.9642857142857143
```

Naive Bayes

In [22]:
```
1 nbx = GaussianNB()
2 nbx.fit(crx_train, cry_train)
3 NB_predx = nbx.predict(crx_test)
```

In [23]:
```
1 NBaccuracy = accuracy_score(cry_test, NB_predx)
2 print("Accuracy of Naive Bayes:", NBaccuracy)
```

```
Accuracy of Naive Bayes: 0.9642857142857143
```

Random Forest

In [24]:
```
rfx = RandomForestClassifier()
rfx.fit(crx_train, cry_train)
RF_predx = rfx.predict(crx_test)
```

In [25]:
```
RFaccuracy = accuracy_score(cry_test, RF_predx)
print("Accuracy of Random Forest", RFaccuracy)
```

```
Accuracy of Random Forest 0.9642857142857143
```

Decision Tree

In [26]:
```
dtx = DecisionTreeClassifier()
dtx.fit(crx_train, cry_train)
DT_predx = dtx.predict(crx_test)
```

In [27]:
```
DTaccuracy = accuracy_score(cry_test, DT_predx)
print("Accuracy of Decision Tree", DTaccuracy)
```

```
Accuracy of Decision Tree 0.8928571428571429
```

Comparison of Accuracy of Implemented Algorithms

In [28]:
```
print("Logistic Regression Accuracy:", LRaccuracy)
print("Naive Bayes Accuracy:", NBaccuracy)
print("Random Forest Accuracy:", RFaccuracy)
print("Decision Tree Accuracy:", DTaccuracy)
```

```
Logistic Regression Accuracy: 0.9642857142857143
Naive Bayes Accuracy: 0.9642857142857143
Random Forest Accuracy: 0.9642857142857143
Decision Tree Accuracy: 0.8928571428571429
```

Code to convert the accuracy into dataframe

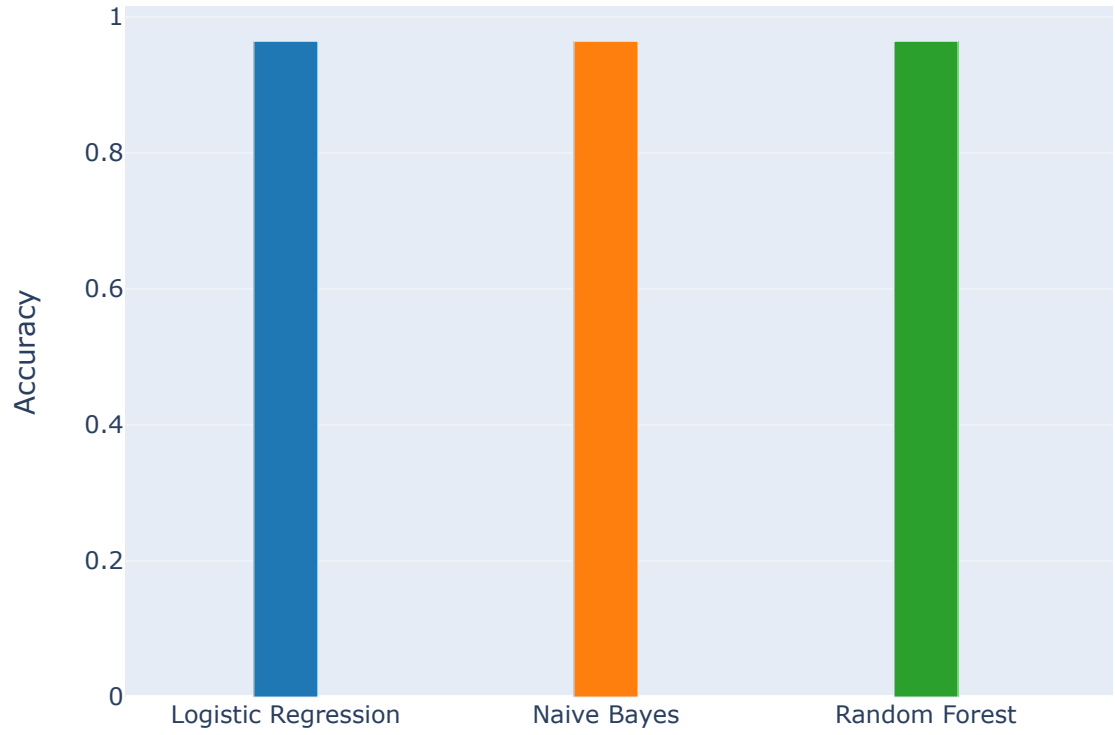
In [29]:
```
data = {
    'Model': ['Logistic Regression', 'Naive Bayes', 'Random Forest', '
    'Accuracy': [LRaccuracy, NBaccuracy, RFaccuracy, DTaccuracy]
}

# create a pandas DataFrame from the dictionary
df = pd.DataFrame(data)

# print the DataFrame
print(df)
```

```
                Model  Accuracy
0  Logistic Regression  0.964286
1          Naive Bayes  0.964286
2        Random Forest  0.964286
3        Decision Tree  0.892857
```

Plot of Accuracy of Implemented Algorithms

In [30]:

```
# set the width and color of the bars
bar_width = 0.2
# define a list of colors for the bars
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

# create the trace
trace = go.Bar(
    x=df['Model'],
    y=df['Accuracy'],
    width=bar_width,
    marker=dict(
        color=colors
    )
)

# create the layout
layout = go.Layout(
    title='Model Comparison',
    yaxis=dict(title='Accuracy')
)

# create the figure
fig = go.Figure(data=[trace], layout=layout)

# adjust the plot width
fig.update_layout(width=800)

fig.show()
```


Model Comparison
Accuracy
1
0.8
0.6
0.4
0.2
0
Logistic Regression
Naive Bayes
Random Forest

In [ ]:
```
1
```