Import the necessary libraries

In [1]:
```
import numpy as np

import matplotlib.pyplot as plt
import pandas as pd

import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error,accuracy_score, mean_squared_error,confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler


import joblib
```

Define the Objective for Building the Model

Objective

To build a model to classify tumours as Malignant or Benign

Read the cancer data as a csv

In [2]:
```
cancer = pd.read_csv("C:\Users\user\Desktop\Machine Learning\cancer_data.csv")
```

In [3]:
```
cancer.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_wo
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0	0.1622	0.6656	0.71
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0	0.1238	0.1866	0.24
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0	0.1444	0.4245	0.45
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7	0.2098	0.8663	0.68
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0	0.1374	0.2050	0.40

5 rows × 33 columns

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

Out[4]:
```
(569, 33)
```

In [5]:
```
cancer["diagnosis"].unique()
```

Out[5]:
```
array(['M', 'B'], dtype=object)
```

M = Malignant ---- severe tumour

B = Benign ---- mild tumour

In [6]:
```
cancer.isnull().sum()
```

Out[6]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

In [7]:
```
cancer.drop(["id","Unnamed: 32"], axis = 1, inplace = True)
```

In [8]:
```
cancer.head()
```

Out[8]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	184.60	2019.0	0.1622	0
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	158.80	1956.0	0.1238	0
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	152.50	1709.0	0.1444	0
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	98.87	567.7	0.2098	0
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	152.20	1575.0	0.1374	0

5 rows × 31 columns

Encoding

In classification, categories are normally changed to numerical values.

This is because not all algorithms can deal with non-numeric categories

In [9]:
```
#Create an instance of the encoder
le = LabelEncoder()
```

In [10]:
```
#Convert the diagnosis column to numerical categories
cancer["diagnosis"] = le.fit_transform(cancer["diagnosis"])
```

In [11]:
```
cancer.head()
```

Out[11]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	184.60	2019.0	0.1622	0
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	158.80	1956.0	0.1238	0
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	152.50	1709.0	0.1444	0
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	98.87	567.7	0.2098	0
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	152.20	1575.0	0.1374	0

5 rows × 31 columns

In [12]:
```
cancer["diagnosis"].unique()
```

Out[12]:
```
array([1, 0])
```

After Encoding,

Benign --- 0

Malignant --- 1

Choose the target and predictors variables

Target --- diagnosis

Predictors --- all the other columns

In [13]:
```
y = cancer["diagnosis"]
X = cancer.drop("diagnosis", axis = 1 )
```

In [14]:
```
y
```

Out[14]:
```
0    1
1    1
2    1
3    1
4    1
    ..
564  1
565  1
566  1
567  1
568  0
Name: diagnosis, Length: 569, dtype: int32
```

In [15]:
```
X
```

In [15]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_wor
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.1374
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.1410
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.1166
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.1139
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.1650
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.0899

569 rows × 30 columns

Split the data to train and test

In [16]:
```
#Splitting your data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

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

Out[17]:
```
(569, 31)
```

In [18]:
```
X_train.shape
```

Out[18]:
```
(455, 30)
```

In [19]:
```
X_test.shape
```

Out[19]:
```
(114, 30)
```

In [20]:
```
y_train.shape
```

Out[20]:
```
(455,)
```

In [21]:
```
y_test.shape
```

Out[21]:
```
(114,)
```

Building the Classification Model - Logistic Regression

In [22]:
```
#Build the model
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[22]:
```
* LogisticRegression
LogisticRegression()
```

Test the Logistic Regression Model

In [23]:
```
#Testing the model
#Using the variable y_pred, and testing data X_test, predict using the model to make comparison with the original y_test to measure the accuracy of the model
y_pred = model.predict(X_test)
```

In [24]:
```
results = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
```

In [25]:
```
results
```

Out[25]:

	Actual	Predicted
204	0	0
70	1	1
131	1	1
431	0	0
540	0	0
...	**...**	**...**
486	0	0
75	1	1
249	0	0
238	0	1
265	1	1

114 rows × 2 columns

Evaluate the Logistic Regression Model

In [26]:
```
#Find the mean squared error and then the square root of the mse to obtain the root mean squared error (rmse)
mse = mean_squared_error(y_pred,y_test)
rmse = np.sqrt(mse)
```

In [27]:
```
print(f"Root mean squared error: {rmse: 2f}")
```

```
Root mean squared error:  0.209427
```

Accuracy

In [28]:
```
#Find the accuracy of the model
acc = accuracy_score(y_pred,y_test)
```

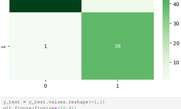
In [29]:
```
acc
```

Out[29]:
```
0.9561403508771929
```

Confusion Matrix

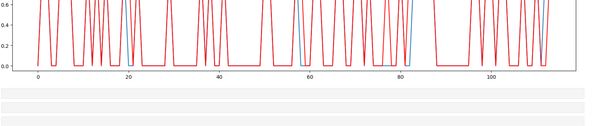
In [30]:
```
#Confusion Matrix
cnf_matrix = confusion_matrix(y_pred,y_test)
sns.heatmap(cnf_matrix, annot = True, cmap="Greens")
```

Out[30]:
```
<Axes: >
```



In [31]:
```
y_test = y_test.values.reshape(-1,1)
plt.figure(figsize=(5,4))
plt.plot(y_pred)(200])
plt.plot(y_test)(200], "r")
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0x1d17053f40>]
```



In [ ]:

In [ ]:

In [ ]: