Breast Cancer Prediction

Breast Cancer Prediction is a classification task aimed at predicting the diagnosis of a breast mass as either malignant or benign. The dataset used for this prediction consists of features computed from a digitized image of a fine needle aspirate (FNA) of the breast mass. These features describe various characteristics of the cell nuclei present in the image.

The dataset contains the following information for each instance:

1. ID number: A unique identifier for each sample.
2. Diagnosis: The target variable indicating the diagnosis, where 'M' represents malignant and 'B' represents benign.

For each cell nucleus, ten real-valued features are computed, which are:

1. Radius: The mean distance from the center to points on the perimeter of the nucleus.
2. Texture: The standard deviation of gray-scale values in the nucleus.
3. Perimeter: The perimeter of the nucleus.
4. Area: The area of the nucleus.
5. Smoothness: A measure of local variation in radius lengths.
6. Compactness: Computed as the square of the perimeter divided by the area minus 1.0.
7. Concavity: Describes the severity of concave portions of the nucleus contour.
8. Concave points: Represents the number of concave portions of the nucleus contour.
9. Symmetry: Measures the symmetry of the nucleus.
10. Fractal dimension: This feature approximates the "coastline" of the nucleus, using the concept of fractal geometry.

These features provide quantitative measurements that can be used to assess the characteristics of cell nuclei and aid in distinguishing between malignant and benign breast masses. By training a machine learning model on this dataset, it is possible to develop a predictive model that can assist in the early detection and diagnosis of breast cancer.

In [1]:
```
# importing the libraries
import numpy as np # This line imports the NumPy library and renames it as np. NumPy is a fundamental library for numer
import pandas as pd # Pandas is a popular data manipulation library that provides data structures (e.g., DataFrames) an
import matplotlib.pyplot as plt # Matplotlib is a widely-used library for creating static, animated, and interactive vi
import seaborn as sns # Seaborn is a data visualization library built on top of Matplotlib that provides a high-level i
```

In [2]:
```
#importing the dataset
df = pd.read_csv('C:/Users/zizhe/Desktop/Leo Zhao/Breast Cancer Prediction/data.csv')
df.head() #This line displays the first five rows (by default) of the DataFrame 'df' using the head() method. This is a
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poi
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 33 columns

Data Preprocessing Part 1

In [3]:
```
# dropping unnecessary columns
df.drop(['Unnamed: 32','id'],axis=1,inplace=True)
# axis=1 specifies that you want to drop columns (as opposed to rows). Columns are along the horizontal axis (axis=1),
#inplace=True is an optional argument that, when set to True, means that the DataFrame will be modified in place (i.e.,
```

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

Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430

5 rows × 31 columns

In [5]:
```
#checking for the missing values
df.isnull().sum()
# df.isnull(): This part of the code generates a Boolean DataFrame of the same shape as 'df', where each element is eit
#.sum(): After generating the Boolean DataFrame, the sum() method is applied. When applied to a Boolean DataFrame, it c
```

Out[5]:
```
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
dtype: int64
```

In [ ]:
```
# This information is useful for data cleaning and preprocessing because you can decide how to handle or impute missing
```

In [6]:
```
#checking the data types of the columns
df.dtypes
# Knowing the data types of columns is important for data analysis and modeling, as it helps you understand how the dat
```

Out[6]:

```
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

In [19]:

```
# checking the data description
df.describe()
# generate summary statistics for the numerical columns in the DataFrame 'df'.
# count: The number of non-null values in the column.
# mean: The average (arithmetic mean) of the values in the column.
# std: The standard deviation, which measures the spread or dispersion of the values.
# min: The minimum value in the column.
# 25%: The 25th percentile value, also known as the first quartile.
# 50%: The median (middle) value, also known as the second quartile.
# 75%: The 75th percentile value, also known as the third quartile.
# max: The maximum value in the column.
```

Out[19]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symm
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	

8 rows × 30 columns

In [11]:
```
# Remove any leading or trailing spaces from column names
df.columns = df.columns.str.strip()
# After running this code, any leading or trailing spaces in the column names of 'df' will be removed. This can be usef
```

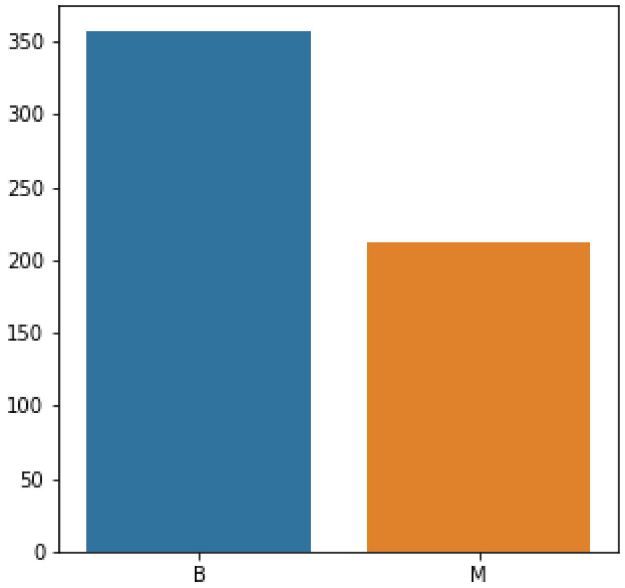
In [12]:
```
# Check the corrected column names
print(df.columns)
#  It will display the names of the columns in 'df' without any extra spaces, allowing you to verify that the column na
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

Exploratory Data Analysis

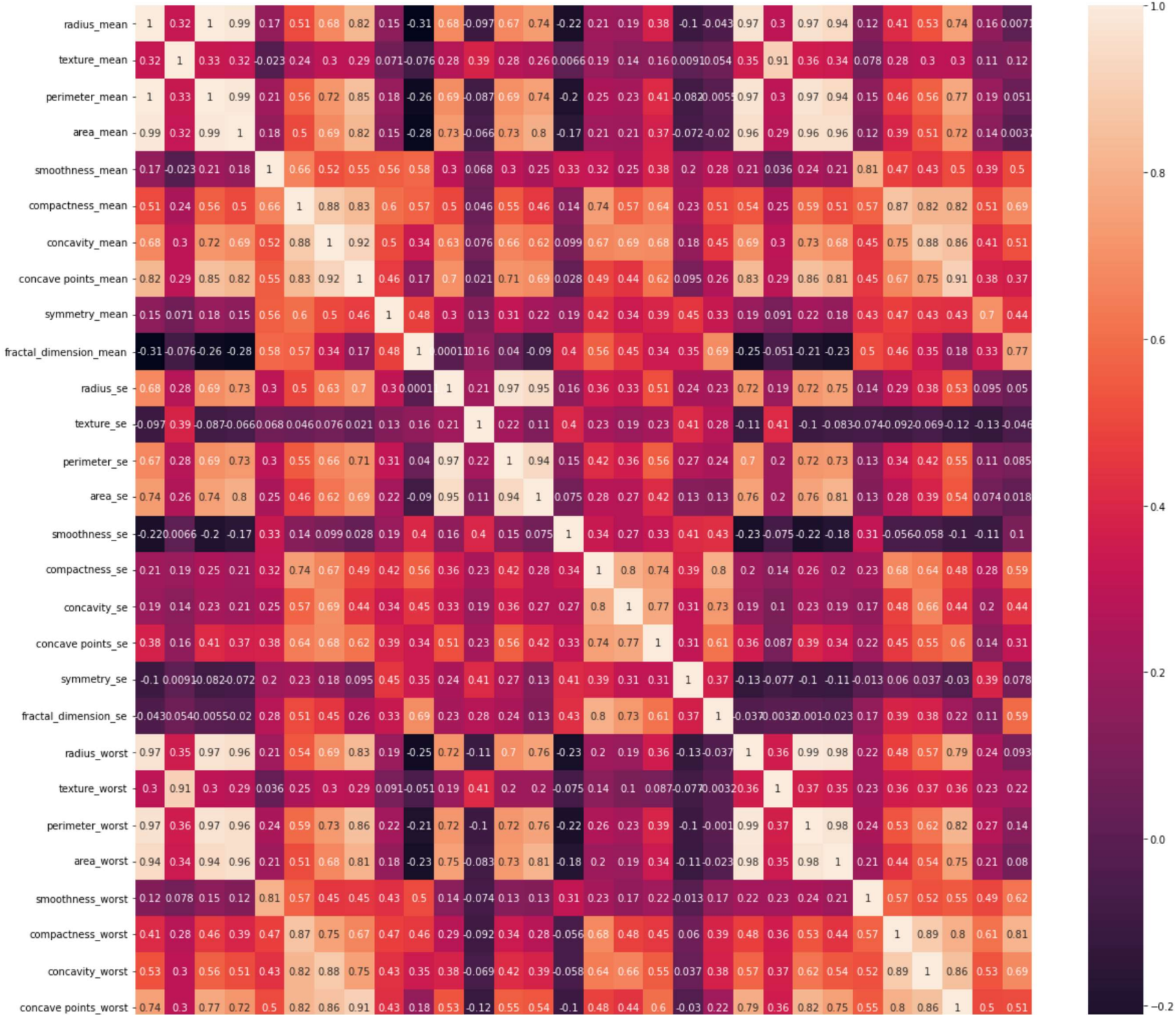
In [17]:
```
# bar plot for the number of diagnosis
plt.figure(figsize=(5,5)) #plt.figure(figsize=(5, 5)): This line sets the figure size for the plot using Matplotlib. Th
sns.barplot(x=df['diagnosis'].value_counts().index,y=df['diagnosis'].value_counts().values)

#this code creates a bar plot that visualizes the number of occurrences of each unique diagnosis value ('M' for maligna
```

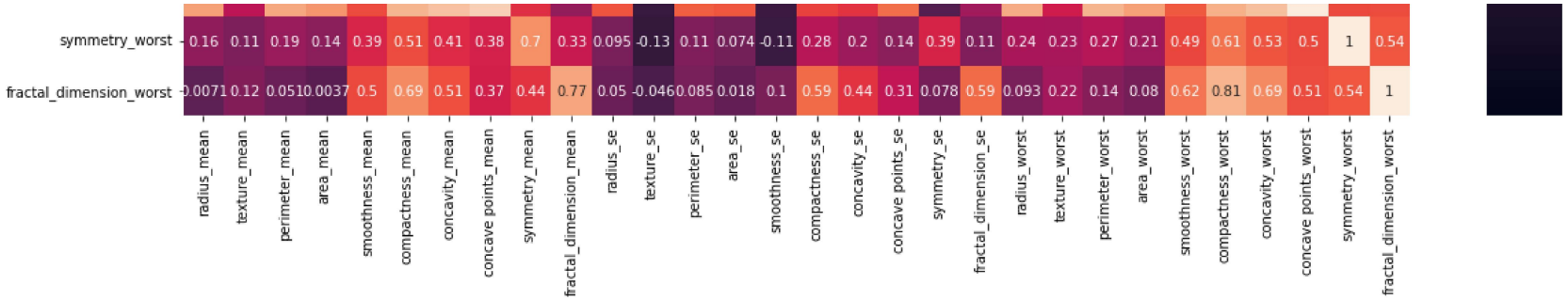
Out[17]:
```
<AxesSubplot:>
```



In [21]:
```
# create a heatmap to check the correlation.
plt.figure(figsize=(20,20)) #The plt.figure() function is used to create a new figure, and figsize specifies the width
sns.heatmap(df.corr(),annot=True) # df.corr(): This calculates the correlation matrix for your DataFrame 'df'. It compu
# sns.heatmap(): This function from Seaborn is used to create a heatmap. It takes the correlation matrix as its data an
#annot=True: This parameter is set to True to display the actual correlation coefficients within each square on the hea
#Positive correlations are often shown in one color, while negative correlations are shown in another color, with a col
```

Out[21]:
```
<AxesSubplot:>
```


radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
radius_se
texture_se
perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave points_se
symmetry_se
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
smoothness_worst
compactness_worst
concavity_worst
concave points_worst
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



Train Test Split

In [22]:
```
#split a DataFrame 'df' into training and testing datasets for machine learning purposes.

from sklearn.model_selection import train_test_split

# df.drop(['diagnosis'],axis=1 This part of the code selects all columns in the DataFrame 'df' except for the 'diagnosi
# It effectively creates a DataFrame containing the features (independent variables) to use for training and testing ma
# test_size=0.3 This parameter specifies that 30% of the data should be used for testing, while the remaining 70% will
#random_state=42: This parameter sets the random seed for the randomization process during the split. Setting a random
# X_train will contain the training features, X_test will contain the testing features, y_train will contain the traini

X_train,X_test,y_train,y_test = train_test_split(df.drop(['diagnosis'],axis=1),df['diagnosis'],test_size=0.3,random_sta
```

Using Decision Tree Classifier

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)

#After running this code, dtree will be a trained decision tree classifier model. You can use this model to make predic
```

Out[23]:
```
DecisionTreeClassifier()
```

In [24]:
```
#predicting the diagnosis
y_pred = dtree.predict(X_test)

#dtree: This is the decision tree classifier model that you previously trained on your training data using dtree.fit(X_

# X_test: This is your test dataset containing the features (independent variables) for which you want to make predicti
```

```
# y_pred: This variable stores the predicted labels for the test data. After running this line of code, y_pred will be
```

Model Evaluation

In [25]:
```
# printing samples from predicted and actual values
print('Predicted values: ',y_pred[:10])
print('Actual values: ',y_test[:10])

#printing the first 10 samples of both the predicted values (y_pred) and the actual values (y_test) to compare them. Th
#This allows you to quickly assess how well your model is performing on a small subset of the test data. If the predict
```

```
Predicted values:  ['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B']
Actual values:  204    B
70     M
131    M
431    B
540    B
567    M
369    M
29     M
81     B
477    B
Name: diagnosis, dtype: object
```

In [26]:
```
# model evaluation
print(dtree.score(X_test,y_test))
#The score method calculates and returns the accuracy of the model on the provided test data. The score method computes
```

```
0.9415204678362573
```

In [ ]:
```
# 0.94 means that the model is making correct predictions for approximately 94% of the samples in the test data.
```

Using logistic regression

In [28]:
```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
C:\Users\zizhe\New folder\lib\site-packages\sklearn\linear_model\_logistic.py:814: 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[28]:
```
LogisticRegression()
```

In [30]:
```
yhat = logmodel.predict(X_test)

#yhat: This variable stores the predicted labels for the test data. After running this line of code, yhat will be an ar
```

Model Evaluation

In [31]:
```
# printing samples from predicted and actual values
print('Predicted values: ',yhat[:10])
print('Actual values: ',y_test[:10])
# If the predicted values closely match the actual values, it's a positive sign that your model is making accurate pred
```

```
Predicted values:  ['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B']
Actual values:  204    B
70     M
131    M
431    B
540    B
567    M
369    M
29     M
81     B
477    B
Name: diagnosis, dtype: object
```

In [ ]:
```
#The numbers "204" and "477" you see in the output are actually indices or row numbers from your test dataset. In the c
```

In [32]:
```
# model evaluation
print(logmodel.score(X_test,y_test))

#logmodel.score(X_test, y_test), calculates and prints the accuracy of the logistic regression model (logmodel) on the
```

```
0.9707602339181286
```

In [ ]:
```
# Conclusion
```

In [ ]:
```
# From both the models we can see that the accuracy is 94% and 97% respectively. But we can see that the recall value f
```