Breast Cancer Wisconsin (Diagnostic)

The dataset used for this analysis is the Breast Cancer Wisconsin (Diagnostic) dataset. It contains data on 569 instances of breast cancer, categorized as malignant or benign, with 30 numeric features derived from digitized images of fine needle aspirates of breast masses. These features describe characteristics such as the radius, texture, smoothness, and compactness of cell nuclei. The target variable indicates whether the tumor is malignant (1) or benign (0). This dataset is widely used in medical research for predictive modeling and classification tasks.

In [39]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
```

In [3]:
```
df = pd.read_csv("data.csv")
```

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

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.
2	84300903	M	19.69	21.25	130.00	1203.0	0.
3	84348301	M	11.42	20.38	77.58	386.1	0.
4	84358402	M	20.29	14.34	135.10	1297.0	0.

5 rows × 33 columns

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

Out[5]:
```
Index(['id', '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', 'Unnamed: 32'],
      dtype='object')
```

In [6]:
```
df.shape
```

Out[6]:
```
(569, 33)
```

In [7]:
```
df.info()
```

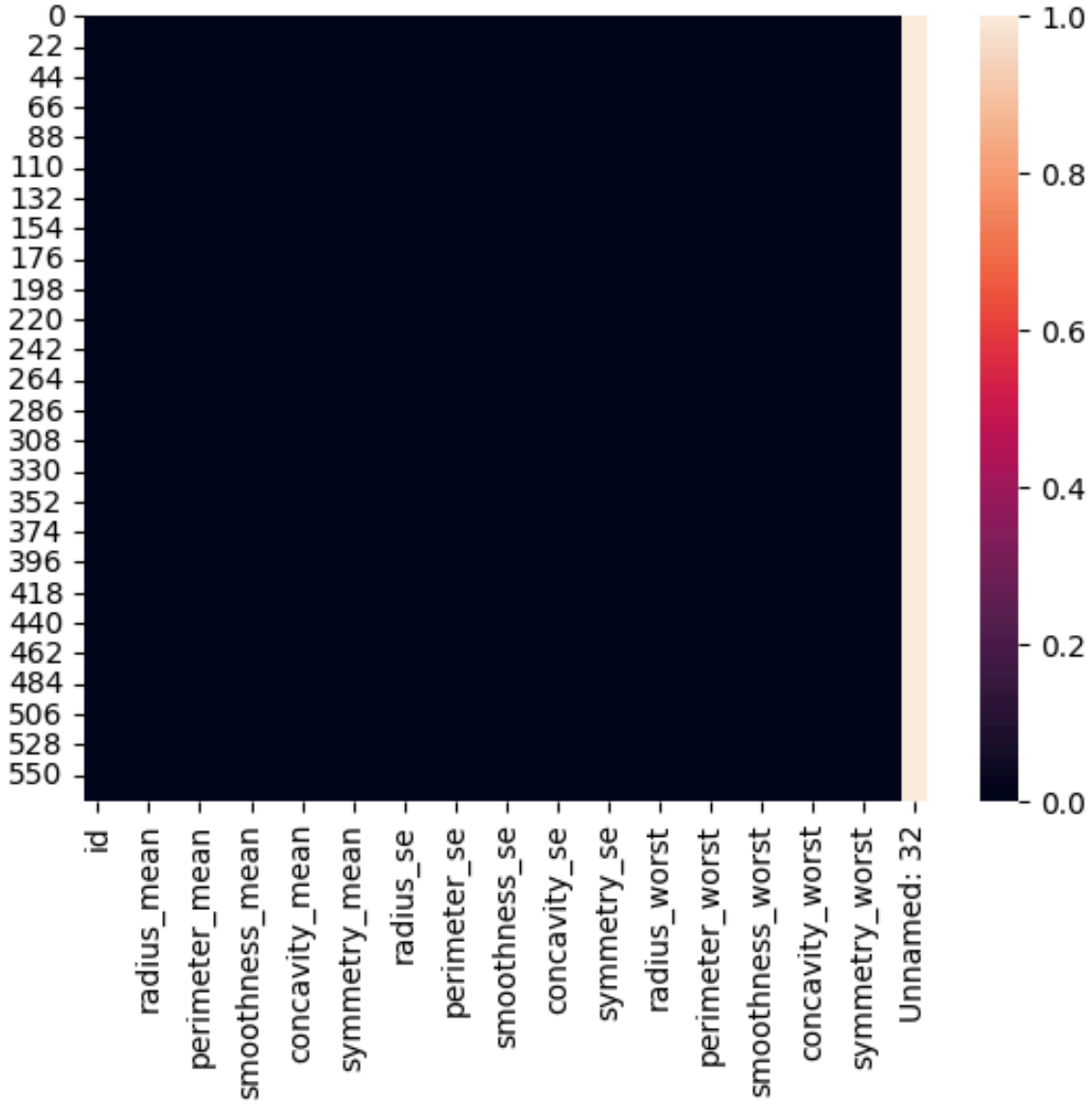
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
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 [9]:
```
sns.heatmap(df.isnull())
```

Out[9]:
```
<Axes: >
```



In [10]:
```
# Drop the column ("Unnamed : 32")
df.drop(columns = "Unnamed: 32",inplace=True)
```

In [11]:
```
df.drop(columns=  "id",inplace=True)
```

In [12]:
```
# for column in df.columns:
#     print(f"---------- Column {column} --------------")
#     print(df[column].value_counts())
```

In [13]:
```
df.describe()
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactr
count	569.000000	569.000000	569.000000	569.000000	569.000000	5
mean	14.127292	19.289649	91.969033	654.889104	0.096360	
std	3.524049	4.301036	24.298981	351.914129	0.014064	
min	6.981000	9.710000	43.790000	143.500000	0.052630	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	

8 rows × 30 columns

In [1]:
```
# for column in df.columns:
#     nunique_value = df[column].apply(type).nunique()
#     if nunique_value > 1:
#         print(f"Column {column} has multiple datatypes")
#     else:
#         print(f"Column {column} has consistent datatype")
```

In [15]:
```
# Encoding Target Variable (Diagnosis) using LabelEncoder
```

Splitting tha Data

In [16]:
```
target_col = df.iloc[:,0]
input_cols = df.iloc[:,1:]
```

In [17]:
```
X_train,X_test,y_train,y_test = train_test_split(input_cols,target_col,test_size=0.3,r
```

In [18]:
```
le = LabelEncoder()
```

In [19]:
```
y_train_transformed = le.fit_transform(y_train)
```

In [20]:
```
y_test_transformed = le.transform(y_test)
```

In [21]:
```
le.classes_
```

Out[21]:
```
array(['B', 'M'], dtype=object)
```

In [22]:
```
# B is 0 and M is 1
```

Normalize the Data

In [23]:
```
Scaler = StandardScaler()
```

In [24]:
```
X_train_scaled = Scaler.fit_transform(X_train)
```

In [25]:
```
X_test_scaled = Scaler.transform(X_test)
```

Train the model

In [26]:
```
log = LogisticRegression()
```

In [27]:
```
log.fit(X_train_scaled,y_train_transformed)
```

Out[27]:
```
▾ LogisticRegression
LogisticRegression()
```

In [28]:
```
y_predict = log.predict(X_test_scaled)
```

Evaluation of the model

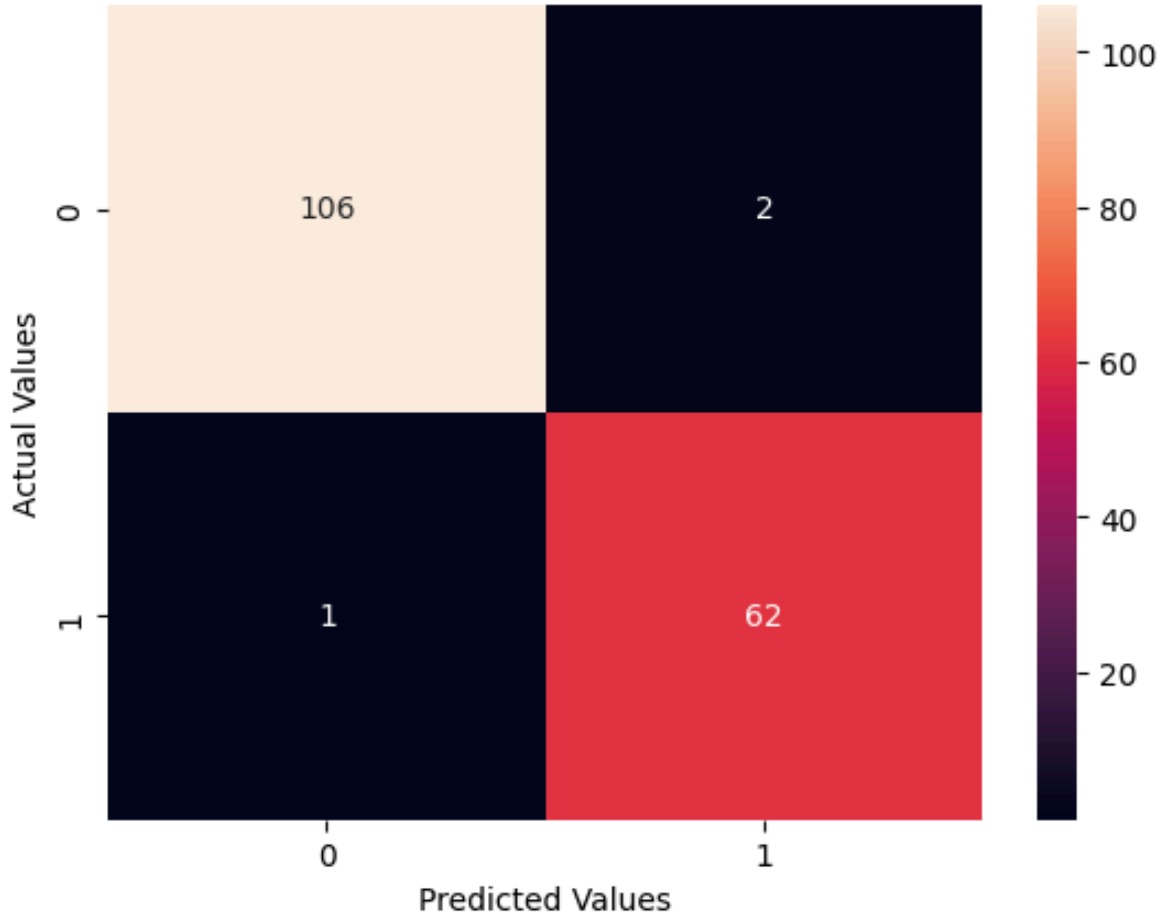
In [29]:
```
Accuracy_score = accuracy_score(y_test_transformed,y_predict)
print(f"Accuracy_Score {Accuracy_score:.2f}")
```

```
Accuracy_Score 0.98
```

In [32]:
```
conf_matrix = confusion_matrix(y_test_transformed,y_predict)
conf_matrix
```

Out[32]:
```
array([[106,   2],
       [  1,  62]])
```

In [36]:
```
sns.heatmap(conf_matrix,annot=True,fmt="d")
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.show()
```



In [31]:
```
Classification_report = classification_report(y_test_transformed,y_predict)
print(Classification_report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       108
           1       0.97      0.98      0.98        63

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

In [ ]:

In [ ]: