In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
```

In [2]:

```
df=pd.read_csv(r"C:\Users\chila\Downloads\ionosphere.csv")
df
```

Out[2]:

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	...	-0.5117
0	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.2656
1	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.4022
2	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.9069
3	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.6515
4	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	...	-0.0153
...	...	...	...	...	...	...	...	...	...	...	...	
345	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.0420
346	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.0136
347	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.0319
348	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.0209
349	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.1511

350 rows × 35 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 350 entries, 0 to 349
Data columns (total 35 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   1          350 non-null    int64  
 1   0          350 non-null    int64  
 2   0.99539    350 non-null    float64
 3   -0.05889   350 non-null    float64
 4   0.85243    350 non-null    float64
 5   0.02306    350 non-null    float64
 6   0.83398    350 non-null    float64
 7   -0.37708   350 non-null    float64
 8   1.1        350 non-null    float64
 9   0.03760    350 non-null    float64
 10  0.85243.1  350 non-null    float64
 11  -0.17755   350 non-null    float64
 12  0.59755    350 non-null    float64
 13  -0.44945   350 non-null    float64
 14  0.60536    350 non-null    float64
 15  -0.38223   350 non-null    float64
 16  0.84356    350 non-null    float64
 17  -0.38542   350 non-null    float64
 18  0.58212    350 non-null    float64
 19  -0.32192   350 non-null    float64
 20  0.56971    350 non-null    float64
 21  -0.29674   350 non-null    float64
 22  0.36946    350 non-null    float64
 23  -0.47357   350 non-null    float64
 24  0.56811    350 non-null    float64
 25  -0.51171   350 non-null    float64
 26  0.41078    350 non-null    float64
 27  -0.46168   350 non-null    float64
 28  0.21266    350 non-null    float64
 29  -0.34090   350 non-null    float64
 30  0.42267    350 non-null    float64
 31  -0.54487   350 non-null    float64
 32  0.18641    350 non-null    float64
 33  -0.45300   350 non-null    float64
 34  g          350 non-null    object 
dtypes: float64(32), int64(2), object(1)
memory usage: 95.8+ KB
```

In [4]:

```
df.describe()
```

Out[4]:

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708
count	350.000000	350.0	350.000000	350.000000	350.000000	350.000000	350.000000	350.000000
mean	0.891429	0.0	0.640330	0.044667	0.600350	0.116154	0.549284	0.120779
std	0.311546	0.0	0.498059	0.442032	0.520431	0.461443	0.493124	0.520816
min	0.000000	0.0	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	1.000000	0.0	0.471517	-0.065388	0.412555	-0.024868	0.209105	-0.053483
50%	1.000000	0.0	0.870795	0.016700	0.808620	0.021170	0.728000	0.015085
75%	1.000000	0.0	1.000000	0.194727	1.000000	0.335317	0.970445	0.451572
max	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 34 columns

In [20]:

```
pd.set_option('display.max_rows',10000000000)
pd.set_option('display.max_columns',10000000000)
pd.set_option('display.width',95)
```

In [6]:

```
df.tail(20)
```

Out[6]:

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	0.85243.1
330	1	0	0.74468	0.10638	0.88706	0.00982	0.88542	0.01471	0.87234	-0.01418	0.73050
331	1	0	0.87578	0.03727	0.89951	0.00343	0.89210	0.00510	0.86335	0.00000	0.95031
332	1	0	0.97513	0.00710	0.98579	0.01954	1.00000	0.01954	0.99290	0.01599	0.95737
333	1	0	1.00000	0.01105	1.00000	0.01105	1.00000	0.02320	0.99448	-0.01436	0.99448
334	1	0	1.00000	-0.01342	1.00000	0.01566	1.00000	-0.00224	1.00000	0.06264	0.97763
335	1	0	0.88420	0.36724	0.67123	0.67382	0.39613	0.86399	0.02424	0.93182	-0.35148
336	1	0	0.90147	0.41786	0.64131	0.75725	0.30440	0.95148	-0.20449	0.96534	-0.55483
337	1	0	0.32789	0.11042	0.15970	0.29308	0.14020	0.74485	-0.25131	0.91993	-0.16503
338	1	0	0.65845	0.43617	0.44681	0.74804	0.05319	0.85106	-0.32027	0.82139	-0.68253
339	1	0	0.19466	0.05725	0.04198	0.25191	-0.10557	0.48866	-0.18321	-0.18321	-0.41985
340	1	0	0.98002	0.00075	1.00000	0.00000	0.98982	-0.00075	0.94721	0.02394	0.97700
341	1	0	0.82254	-0.07572	0.80462	0.00231	0.87514	-0.01214	0.86821	-0.07514	0.72832
342	1	0	0.35346	-0.13768	0.69387	-0.02423	0.68195	-0.03574	0.55717	-0.06119	0.61836
343	1	0	0.76046	0.01092	0.86335	0.00258	0.85821	0.00384	0.79988	0.02304	0.81504
344	1	0	0.66667	-0.01366	0.97404	0.06831	0.49590	0.50137	0.75683	-0.00273	0.65164
345	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	0.89391
346	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	0.96510
347	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	0.94124
348	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	0.89724
349	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	0.78735

In [21]:

```
print('This DataFrame has %d Rows and %d columns'%(df.shape))
```

```
This DataFrame has 350 Rows and 35 columns
```

In [8]:

```
df.head()
```

Out[8]:

0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	0.85243.1	-0.17755
0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	0.50874	-0.67743
0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	0.73082	0.05346
0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	0.00000	0.00000
0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	0.52798	-0.20275
0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	0.03786	-0.06302

In [22]:

```
features_matrix = df.iloc[:,0:34]
```

In [23]:

```
target_vector = df.iloc[:,-1]
```

In [24]:

```
print('The Features Matrix Has %d Rows And %d columns(s)'%(features_matrix.shape))
```

```
The Features Matrix Has 350 Rows And 34 columns(s)
```

In [11]:

```
feature_matrix_standardized = StandardScaler().fit_transform(feature_matrix)
```

In [27]:

```
algorithm = LogisticRegression(penalty=None,dual=False, tol=1e-4,C=1.0, fit_intercept=True,i
 class_weight=None,random_state=None,solver='lbfgs',max_iter=10000,
 multi_class='auto',verbose=0, warm_start=False, n_jobs=None,l1_ratio=None)
```

In [26]:

```
features_matrix_standardized = StandardScaler().fit_transform(features_matrix)
```

In [13]:

```
Algorithm = LogisticRegression(penalty=None,dual=False, tol=1e-4,C=1.0, fit_intercept=True,i
                               class_weight=None,random_state=None,solver='lbfgs',max_iter=1
                               multi_class='auto',verbose=0, warm_start=False, n_jobs=None,l
```

In [28]:

```
Logistic_Regression_Model = algorithm.fit(features_matrix_standardized,target_vector)
```

In [29]:

```
observation = [[1, 0, 0.99539, -0.05889, 0.8524299999999999, 0.02306, 0.8339799999999999, -0
 0.8524299999999999, -0.17755, 0.59755, -0.44945, 0.60536, -0.38223, 0.8435600000000001, -0.
 0.58212, -0.32192, 0.56971, -0.29674, 0.36946, -0.47357, 0.56811, -0.51171, 0.41078000000000
 -0.4616800000000003, 0.21266, -0.3409,0.112267,-0.54487,0.18641,-0.453]]
```

In [30]:

```
predictions = Logistic_Regression_Model.predict(observation)
print('The Model predicted The observation To Belong To Class %s'%(predictions))
```

```
The Model predicted The observation To Belong To Class ['g']
```

In [31]:

```
print('The Algorithm Was Trained To predict The One Of The Classes: %s'%(algorithm.classes_)
```

```
The Algorithm Was Trained To predict The One Of The Classes: ['b' 'g']
```

In [32]:

```
print("""The Model Says The Probability Of The observation We Passed belonging To The Class
 %(algorithm.predict_proba(observation)[0][0]))
print()
```

```
The Model Says The Probability Of The observation We Passed belonging To The C
lass ['b'] is 4.3227218160968306e-05
```

In [34]:

```
print("""The Model Says The Probability Of The observation We Passed belonging To The Class
 %(algorithm.predict_proba(observation)[0][1]))
```

```
The Model Says The Probability Of The observation We Passed belonging To The C
lass ['g'] is 0.999956772781839
```