Name: Garcia, John Carlos M.

Section: CPE22S3

Date Performed: 04/22/2024

Date Submitted: 04/27/2024

Logistic Regression Analysis

Task: Determine the origin of wines

Setup

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.model_selection import train_test_split
6
7 %matplotlib inline
```

```
1 pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

```
1 from ucimlrepo import fetch_ucirepo
2
3 # fetch dataset
4 wine = fetch_ucirepo(id=109)
5
6 # data (as pandas dataframes)
7 X = wine.data.features
8 y = wine.data.targets
9
10 # metadata
11 print("Metadata:\n",wine.metadata)
12
13 # variable information
14 print("\nVariables:\n",wine.variables)
15
```

```
Metadata:
 {'uci_id': 109, 'name': 'Wine', 'repository_url': 'https://archive.ics.uci.edu/dataset/109/wine', 'data_url': 'https://archive.ics.uci.

Variables:
                            name     role         type demographic  \
0                          class   Target  Categorical        None
1                        Alcohol  Feature   Continuous        None
2                      Malicacid  Feature   Continuous        None
3                            Ash  Feature   Continuous        None
4              Alcalinity_of_ash  Feature   Continuous        None
5                      Magnesium  Feature      Integer        None
6                  Total_phenols  Feature   Continuous        None
7                     Flavanoids  Feature   Continuous        None
8           Nonflavanoid_phenols  Feature   Continuous        None
9                Proanthocyanins  Feature   Continuous        None
10               Color_intensity  Feature   Continuous        None
11                           Hue  Feature   Continuous        None
12  0D280_0D315_of_diluted_wines  Feature   Continuous        None
13                       Proline  Feature      Integer        None

   description units missing_values
0         None  None             no
1         None  None             no
```

```
2         None  None             no
3         None  None             no
4         None  None             no
5         None  None             no
6         None  None             no
7         None  None             no
8         None  None             no
9         None  None             no
10        None  None             no
11        None  None             no
12        None  None             no
13        None  None             no
```

Concatenation

```
1 logistic_df = pd.concat([X, y], axis=1) #Combine both dataframes into one for more efficient manipulation of data
```

Exploration

```
1 print("Head:\n",logistic_df.head(), "\n\n")
2
3 print("DTypes:\n",logistic_df.dtypes, "\n\n")
4
5 print("Description:\n",logistic_df.describe())
```

```
Head:
    Alcohol  Malicacid   Ash  Alcalinity_of_ash  Magnesium  Total_phenols  \
0    14.23       1.71  2.43               15.6        127           2.80
1    13.20       1.78  2.14               11.2        100           2.65
2    13.16       2.36  2.67               18.6        101           2.80
3    14.37       1.95  2.50               16.8        113           3.85
4    13.24       2.59  2.87               21.0        118           2.80

   Flavanoids  Nonflavanoid_phenols  Proanthocyanins  Color_intensity   Hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   0D280_0D315_of_diluted_wines  Proline  class
0                          3.92     1065      1
1                          3.40     1050      1
2                          3.17     1185      1
3                          3.45     1480      1
4                          2.93      735      1


DTypes:
 Alcohol                        float64
Malicacid                      float64
Ash                            float64
Alcalinity_of_ash              float64
Magnesium                        int64
Total_phenols                  float64
Flavanoids                     float64
Nonflavanoid_phenols           float64
Proanthocyanins                float64
Color_intensity                float64
Hue                            float64
0D280_0D315_of_diluted_wines   float64
Proline                          int64
class                            int64
dtype: object


Description:
          Alcohol   Malicacid         Ash  Alcalinity_of_ash   Magnesium  \
count  178.000000  178.000000  178.000000         178.000000  178.000000
mean    13.000618    2.336348    2.366517          19.494944   99.741573
std      0.811827    1.117146    0.274344           3.339564   14.282484
min     11.030000    0.740000    1.360000          10.600000   70.000000
25%     12.362500    1.602500    2.210000          17.200000   88.000000
50%     13.050000    1.865000    2.360000          19.500000   98.000000
75%     13.677500    3.082500    2.557500          21.500000  107.000000
max     14.830000    5.800000    3.230000          30.000000  162.000000
```

```
       Total_phenols  Flavanoids  Nonflavanoid_phenols  Proanthocyanins  \
count     178.000000  178.000000            178.000000       178.000000
mean        2.295112    2.029270              0.361854         1.590899
std         0.625851    0.998859              0.124453         0.572359
min         0.980000    0.340000              0.130000         0.410000
25%         1.742500    1.205000              0.270000         1.250000
```

Identification of Missing Values

```
print("Nulls:\n",logistic_df.isnull().sum())
```

```
Nulls:
 Alcohol                        0
Malicacid                       0
Ash                             0
Alcalinity_of_ash               0
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
Nonflavanoid_phenols            0
Proanthocyanins                 0
Color_intensity                 0
Hue                             0
0D280_0D315_of_diluted_wines    0
Proline                         0
class                           0
dtype: int64
```

Identification of Duplicated Rows

```
duplicates = logistic_df.duplicated()
print("Duplicates:\n\n", duplicates, "\n\n")
print("Duplicate Rows:\n\n",logistic_df[duplicates])
```

```
Duplicates:

 0      False
1       False
2       False
3       False
4       False
        ...
173     False
174     False
175     False
176     False
177     False
Length: 178, dtype: bool


Duplicate Rows:

 Empty DataFrame
Columns: [Alcohol, Malicacid, Ash, Alcalinity_of_ash, Magnesium, Total_phenols, Flavanoids, Nonflavanoid_phenols, Proanthocyanins, Color
Index: []
```

```
print("Columns:\n",logistic_df.columns, "\n\n")
print("\n\nUnique values in 'class' column:")
print(logistic_df['class'].unique())

print("\n\nUnique values in 'Alcohol' column:")
print(logistic_df['Alcohol'].unique())

print("\n\nUnique values in 'Malicacid' column:")
print(logistic_df['Malicacid'].unique())

print("\n\nUnique values in 'Ash' column:")
print(logistic_df['Ash'].unique())

print("\n\nUnique values in '0D280_0D315_of_diluted_wines' column:")
print(logistic_df['0D280_0D315_of_diluted_wines'].unique())

print("\n\nUnique values in 'Proline' column:")
print(logistic_df['Proline'].unique())
```

```
Unique values in 'Alcohol' column:
[14.23 13.2  13.16 14.37 13.24 14.2  14.39 14.06 14.83 13.86 14.1  14.12
 13.75 14.75 14.38 13.63 14.3  13.83 14.19 13.64 12.93 13.71 12.85 13.5
 13.05 13.39 13.3  13.87 14.02 13.73 13.58 13.68 13.76 13.51 13.48 13.28
 13.07 14.22 13.56 13.41 13.88 14.21 13.9  13.94 13.82 13.77 13.74 13.29
 13.72 12.37 12.33 12.64 13.67 12.17 13.11 13.34 12.21 12.29 13.49 12.99
 11.96 11.66 13.03 11.84 12.7  12.   12.72 12.08 12.67 12.16 11.65 11.64
 12.69 11.62 12.47 11.81 12.6  12.34 11.82 12.51 12.42 12.25 12.22 11.61
 11.46 12.52 11.76 11.41 11.03 12.77 11.45 11.56 11.87 12.07 12.43 11.79
 12.04 12.86 12.88 12.81 12.53 12.84 13.36 13.52 13.62 12.87 13.32 13.08
 12.79 13.23 12.58 13.17 13.84 12.45 14.34 12.36 13.69 12.96 13.78 13.45
 12.82 13.4  12.2  14.16 13.27 14.13]


Unique values in 'Malicacid' column:
[1.71 1.78 2.36 1.95 2.59 1.76 1.87 2.15 1.64 1.35 2.16 1.48 1.73 1.81
 1.92 1.57 1.59 3.1  1.63 3.8  1.86 1.6  2.05 1.77 1.72 1.9  1.68 1.5
 1.66 1.83 1.53 1.8  1.65 3.99 3.84 1.89 3.98 4.04 3.59 2.02 1.75 1.67
 1.7  1.97 1.43 0.94 1.1  1.36 1.25 1.13 1.45 1.21 1.01 1.17 1.19 1.61
 1.51 1.09 1.88 0.9  2.89 0.99 3.87 0.92 3.86 0.89 0.98 2.06 1.33 2.83
 1.99 1.52 2.12 1.41 1.07 3.17 2.08 1.34 2.45 2.55 1.29 3.74 2.43 2.68
 0.74 1.39 1.47 3.43 2.4  4.43 5.8  4.31 2.13 4.3  2.99 2.31 3.55 1.24
 2.46 4.72 5.51 2.96 2.81 2.56 4.95 3.88 3.57 5.04 4.61 3.24 3.9  3.12
 2.67 3.3  5.19 4.12 3.03 3.83 3.26 3.27 3.45 2.76 4.36 3.7  3.37 2.58
 4.6  2.39 2.51 5.65 3.91 4.28 4.1 ]


Unique values in 'Ash' column:
[2.43 2.14 2.67 2.5  2.87 2.45 2.61 2.17 2.27 2.3  2.32 2.41 2.39 2.38
 2.7  2.72 2.62 2.48 2.56 2.28 2.65 2.36 2.52 3.22 2.8  2.21 2.84 2.55
 2.1  2.51 2.31 2.12 2.59 2.29 2.44 2.4  2.04 2.6  2.42 2.68 2.25 2.46
 1.36 2.02 1.92 2.16 2.53 1.7  1.75 2.24 1.71 2.23 1.95 2.   2.2  2.58
 2.26 2.22 2.74 1.98 1.9  1.88 1.94 1.82 2.92 1.99 2.19 3.23 2.73 2.13
 2.78 2.54 2.64 2.35 2.15 2.75 2.69 2.86 2.37]


Unique values in '0D280_0D315_of_diluted_wines' column:
[3.92 3.4  3.17 3.45 2.93 2.85 3.58 3.55 2.82 2.9  2.73 3.   2.88 2.65
 2.57 3.36 3.71 3.52 4.   3.63 3.82 3.2  3.22 2.77 3.59 2.71 2.87 3.47
 2.78 2.51 2.69 3.53 3.38 3.56 3.35 3.33 3.44 2.75 3.1  2.91 3.37 3.26
 3.03 3.31 2.84 1.82 1.67 1.59 2.46 2.23 2.3  3.18 3.48 1.93 3.07 3.16
 3.5  3.13 2.14 2.48 2.52 2.31 3.12 3.14 2.72 2.01 3.08 2.26 3.21 2.27
 2.06 3.3  2.96 2.63 2.74 2.83 2.44 3.57 2.42 3.02 2.81 2.5  3.19 2.12
 3.05 3.39 3.69 3.64 3.28 1.29 1.42 1.36 1.51 1.58 1.27 1.69 2.15 2.47
 2.05 2.   1.68 1.33 1.86 1.62 1.3  1.47 1.55 1.48 1.64 1.73 1.96 1.78
 2.11 1.75 1.56 1.8  1.92 1.83 1.63 1.71 1.74 1.6 ]


Unique values in 'Proline' column:
[1065 1050 1185 1480  735 1450 1290 1295 1045 1510 1280 1320 1150 1547
 1310 1130 1680  845  780  770 1035 1015  830 1195 1285  915 1515  990
 1235 1095  920  880 1105 1020  760  795  680  885 1080  985 1060 1260
 1265 1190 1375 1120  970 1270  520  450  630  420  355  678  502  510
  750  718  870  410  472  886  428  392  500  463  278  714  515  495
  562  625  480  290  345  937  660  406  710  438  415  672  315  488
  312  325  607  434  385  407  372  564  465  365  380  378  352  466
  342  580  530  560  600  650  695  720  590  550  855  425  675  640
  725  620  570  615  685  470  740  835  840]
```

```
1 logistic_df
2 categorical = [var for var in logistic_df.columns if logistic_df[var].dtype=='O']
3 print('There are {} categorical variables\n'.format(len(categorical)))
4 print('The categorical variables are :', categorical)
```

```
There are 0 categorical variables

The categorical variables are : []
```

Cleaning

Nothing to clean. There are no object columns, missing values, and duplicated rows.

Logistic Regression

Declare Feature Vector and Target Variable

```
X = logistic_df.drop('class', axis = 1)
y = logistic_df['class']
```

Split data into separate training and test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
X_train.shape, X_test.shape
```

```
((142, 13), (36, 13))
```

Feature Engineering

```
X_train.dtypes
```

```
Alcohol                         float64
Malicacid                       float64
Ash                             float64
Alcalinity_of_ash               float64
Magnesium                         int64
Total_phenols                   float64
Flavanoids                      float64
Nonflavanoid_phenols            float64
Proanthocyanins                 float64
Color_intensity                 float64
Hue                             float64
0D280_0D315_of_diluted_wines    float64
Proline                           int64
dtype: object
```

```
#Display Categorical Variables
categorical = [col for col in X_train.columns if X_train[col].dtypes == '0']
categorical
```

```
[]
```

```
#Display Numerical Variables
numerical = [col for col in X_train.columns if X_train[col].dtypes != '0']
numerical
```

```
['Alcohol',
 'Malicacid',
 'Ash',
 'Alcalinity_of_ash',
 'Magnesium',
 'Total_phenols',
 'Flavanoids',
 'Nonflavanoid_phenols',
 'Proanthocyanins',
 'Color_intensity',
 'Hue',
 '0D280_0D315_of_diluted_wines',
 'Proline']
```

Engineering Missing Values in Numerical Variables

```
#Check Missing Values in numerical variables in X_train
X_train[numerical].isnull().sum()
```

```
Alcohol                         0
Malicacid                       0
Ash                             0
Alcalinity_of_ash               0
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
Nonflavanoid_phenols            0
Proanthocyanins                 0
```

```
Color_intensity                0
Hue                            0
0D280_0D315_of_diluted_wines   0
Proline                        0
dtype: int64
```

```
#Check Missing Values in numerical variables in X_test
X_test[numerical].isnull().sum()
```

```
Alcohol                        0
Malicacid                      0
Ash                            0
Alcalinity_of_ash              0
Magnesium                      0
Total_phenols                  0
Flavanoids                     0
Nonflavanoid_phenols           0
Proanthocyanins                0
Color_intensity                0
Hue                            0
0D280_0D315_of_diluted_wines   0
Proline                        0
dtype: int64
```

```
#Print Percentage of missing values in the numerical variables in training set
for col in numerical:
  if X_train[col].isnull().mean()>0:
    print(col, round(X_train[col].isnull().mean(), 4))
```

No missing data

Feature Scaling

```
X_train.describe()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	F
count	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	
mean	12.984859	2.372606	2.366901	19.554930	100.063380	2.258662	
std	0.807175	1.115360	0.269684	3.442549	14.249158	0.611691	
min	11.030000	0.740000	1.360000	10.600000	70.000000	1.100000	
25%	12.347500	1.602500	2.222500	17.250000	89.000000	1.705000	
50%	13.040000	1.895000	2.360000	19.500000	98.000000	2.210000	
75%	13.637500	3.222500	2.560000	21.500000	106.750000	2.735000	
max	14.750000	5.650000	3.220000	30.000000	162.000000	3.880000	

```
cols = X_train.columns
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train = pd.DataFrame(X_train, columns = [cols])
```

```
X_test = pd.DataFrame(X_test, columns = [cols])
```

```
X_train.describe()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	F
count	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	
mean	0.525500	0.332506	0.541345	0.461594	0.326776	0.416785	
std	0.216983	0.227161	0.144991	0.177451	0.154882	0.220033	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.354167	0.175662	0.463710	0.342784	0.206522	0.217626	
50%	0.540323	0.235234	0.537634	0.458763	0.304348	0.399281	
75%	0.700941	0.505601	0.645161	0.561856	0.399457	0.588129	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Model Training

```
#Train a Logistic regression model on the training set
from sklearn.linear_model import LogisticRegression

#Instantiate the model
logreg = LogisticRegression(solver = 'liblinear', random_state = 0)

#fit the model
logreg.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0, solver='liblinear')
```

Predict Results

```
y_pred_test = logreg.predict(X_test)
y_pred_test
```

```
array([1, 3, 2, 1, 2, 2, 1, 3, 2, 2, 3, 3, 1, 2, 3, 2, 1, 1, 3, 1, 2, 1,
       1, 2, 2, 2, 2, 2, 2, 3, 1, 1, 2, 1, 1, 1])
```

predict_proba method gives the probabilities for the class, or origin of wine, which in this case is 1, 2, and 3, in array form.

```
#Probability of getting output as 1
logreg.predict_proba(X_test)[:,0]
```

```
array([0.83411707, 0.08169456, 0.33794154, 0.79491944, 0.2372458 ,
       0.237406  , 0.87096795, 0.03861458, 0.16257044, 0.05946295,
       0.1482802 , 0.03716879, 0.92972498, 0.47744117, 0.0838979 ,
       0.13261824, 0.78069736, 0.95517064, 0.0825623 , 0.84310019,
       0.47020366, 0.67960296, 0.47193266, 0.24693731, 0.08366256,
       0.16146061, 0.21250699, 0.05442331, 0.07420845, 0.07434986,
       0.83480123, 0.85379414, 0.08617887, 0.8224449 , 0.87731904,
       0.6611851 ])
```

```
#Probability of getting output as 2
logreg.predict_proba(X_test)[:,1]
```

```
array([0.1240818 , 0.06966025, 0.65228402, 0.1607158 , 0.61192177,
       0.75455152, 0.08070117, 0.14839743, 0.77629276, 0.77373406,
       0.13986198, 0.07780088, 0.03752654, 0.51475621, 0.07579536,
       0.85273997, 0.16126423, 0.02246023, 0.44592902, 0.13936754,
       0.52143812, 0.24521823, 0.43573301, 0.71879674, 0.58483752,
       0.76777135, 0.73705714, 0.8411666 , 0.73358901, 0.06019109,
       0.12898216, 0.11348826, 0.61642905, 0.06319566, 0.08703462,
       0.31648131])
```

```
#Probability of getting output as 3
logreg.predict_proba(X_test)[:,2]
```

```
array([0.04180113, 0.84864519, 0.00977444, 0.04436476, 0.15083243,
       0.00804248, 0.04833088, 0.81298799, 0.0611368 , 0.16680299,
       0.71185782, 0.88503033, 0.03274848, 0.00780262, 0.84030674,
```

```
        0.01464178, 0.0580384 , 0.02236913, 0.47150868, 0.01753227,
        0.00835821, 0.0751788 , 0.09233433, 0.03426595, 0.33149992,
        0.07076804, 0.05043587, 0.10441009, 0.19220254, 0.86545905,
        0.03621662, 0.0327176 , 0.29739208, 0.11435944, 0.03564634,
        0.02233359])
```

Check Accuracy Score

```
1 from sklearn.metrics import accuracy_score
2 print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
    Model accuracy score: 0.9722
```

Compare the train-set and test-set accuracy

```
1 y_pred_train = logreg.predict(X_train)
2 y_pred_train
```

```
    array([3, 2, 2, 3, 1, 1, 2, 2, 2, 1, 3, 2, 3, 1, 3, 3, 1, 3, 1, 2, 3, 3,
           2, 3, 3, 1, 3, 3, 2, 3, 3, 2, 1, 2, 2, 2, 1, 1, 2, 2, 3, 3, 2, 2,
           2, 3, 3, 1, 3, 2, 2, 2, 2, 2, 1, 1, 2, 1, 3, 1, 3, 1, 1, 2, 1, 2,
           2, 1, 3, 2, 1, 2, 2, 2, 3, 1, 3, 3, 1, 1, 2, 3, 1, 1, 2, 2, 1, 1,
           1, 3, 2, 1, 2, 3, 1, 2, 3, 3, 1, 1, 3, 1, 3, 2, 1, 1, 2, 1, 3, 2,
           3, 1, 3, 3, 3, 1, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 3, 3, 1, 1, 3, 2,
           2, 2, 1, 1, 1, 2, 2, 2, 1, 3])
```

```
1 print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
    Training-set accuracy score: 0.9789
```

Check for overfitting and underfitting

```
1 #Print the scores on training and test set
2 print('Training set score: {:.4f}'. format(logreg.score(X_train, y_train)))
3 print('Test set score: {:.4f}'. format(logreg.score(X_test, y_test)))
```

```
    Training set score: 0.9789
    Test set score: 0.9722
```

As the training set score and test set score are quite comparable to each other, there is no question of overfitting.

Use C = 100

```
1 #Fit the Logistic Regression model with C = 100
2 #Instantiate the model
3 logreg100 = LogisticRegression(C = 100, solver = 'liblinear', random_state = 0)
4 #Fit the model
5 logreg100.fit(X_train, y_train)
```

```
    LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

```
1 #Print the scores on training and test set
2 print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))
3 print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
```

```
    Training set score: 1.0000
    Test set score: 1.0000
```

We can see that C = 100 results in higher accuracy, meaning that this model performed the best. But let us be careful and use other means, in case that this is a result of overfitting.

```
#Fit the Logistic Regression model with C = 0.01
#Instantiate the model
logreg001 = LogisticRegression(C = 0.01, solver = 'liblinear', random_state = 0)
#Fit the model
logreg001.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=0.01, random_state=0, solver='liblinear')
```

```
#Print the scores on training and test set
print('Training set score: {:.4f}'.format(logreg001.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(logreg001.score(X_test, y_test)))
```

```
Training set score: 0.8944
Test set score: 0.8889
```

Setting a different value for C other than 100 actually prevents the model from overfitting, as we can see from when C is equals to 1 and 0.01.

Compare model accuracy with null accruacy

```
#Check class distribution in test set
y_test.value_counts()
```

```
class
2    16
1    14
3     6
Name: count, dtype: int64
```

We can see that the occurences of the most frequent class is 16. So, we can calculate the null accuracy by dividing 16 by the total number of occurences

```
#Check Null Accuracy Score
null_accuracy = (16/(16+14+6))
print('Null Accuracy Score: {0:0.4f}'.format(null_accuracy))
```

```
Null Accuracy Score: 0.4444
```

As the scores for the training and test when C is equal to one are 0.9789 and 0.9722, respectively, a null accuracy score of 0.4444—which is significantly lesser than both score—tells us that the model is performing exceptionally well and is learning meaningful patterns from the data.

Confusion Matrix

```
#Print the Confusion Matrix and slice it into four pieces
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
print('Confusion matrix\n\n', cm)
print("\nTrue Positives(TP) = ", cm[0,0])
print("\nTrue Negatives(TN) = ", cm[1,1])
print("\nFalse Positivves(FP) = ", cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[14  0  0]
 [ 0 15  1]
 [ 0  0  6]]

True Positives(TP) =  14

True Negatives(TN) =  15

False Positivves(FP) =  0

False Negatives(FN) =  0
```

The confusion matrix shows 19 correct predictions, and 0 incorrect predictions!!

```
#Visualize confusion matrix with seaborn heatmap
cm_matrix = pd.DataFrame(data = cm, columns=['Predicted Class 1', 'Predicted Class 2', 'Predicted Class 3'],
                         index=['Actual Class 1', 'Actual Class 2', 'Actual Class 3'])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```