Logistic Regression

In [58]:
```
import pandas as pd
fish = pd.read_csv('dataset_Fish.csv')
fish.head()
```

Out[58]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

In [59]:
```
fish['Species'].unique()
```

Out[59]:
```
array(['Bream', 'Roach', 'Whitefish', 'Parkki', 'Perch', 'Pike', 'Smelt'],
      dtype=object)
```

In [60]:
```
fish.isnull().sum()
```

Out[60]:
```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

In [61]:
```
X = fish.iloc[:, 1:]
y = fish.loc[:, 'Species']
```

In [62]:
```
X
```

Out[62]:

	Weight	Length1	Length2	Length3	Height	Width
0	242.0	23.2	25.4	30.0	11.5200	4.0200
1	290.0	24.0	26.3	31.2	12.4800	4.3056
2	340.0	23.9	26.5	31.1	12.3778	4.6961
3	363.0	26.3	29.0	33.5	12.7300	4.4555
4	430.0	26.5	29.0	34.0	12.4440	5.1340
...	...	...	...	...	...	...
154	12.2	11.5	12.2	13.4	2.0904	1.3936
155	13.4	11.7	12.4	13.5	2.4300	1.2690
156	12.2	12.1	13.0	13.8	2.2770	1.2558
157	19.7	13.2	14.3	15.2	2.8728	2.0672
158	19.9	13.8	15.0	16.2	2.9322	1.8792

159 rows × 6 columns

Scaling the input features using MinMaxScaler

In [63]:
```
y
```

Out[63]:
```
0      Bream
1      Bream
2      Bream
3      Bream
4      Bream
       ...
154    Smelt
155    Smelt
156    Smelt
157    Smelt
158    Smelt
Name: Species, Length: 159, dtype: object
```

In [65]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

Label Encoding the target variable using LabelEncoder

In [76]:
```python
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
y
```

Out[76]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5], dtype=int64)
```

Splitting into train and test datasets using train_test_split

In [77]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Model Building and training

In [78]:
```python
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression()
# training the model
logReg.fit(X_train, y_train)
```

Out[78]:
```
LogisticRegression()
```

Predicting the output

In [79]:
```python
y_pred = logReg.predict(X_test)
```

Computing the accuracy

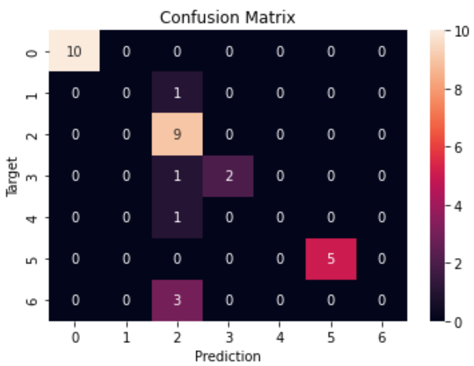
In [80]:
```python
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Accuracy: 81.25%
```

Confusion Matrix

In [50]:
```python
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cf = confusion_matrix(y_test, y_pred)
plt.figure()
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

Out[50]:
```
Text(0.5, 1.0, 'Confusion Matrix')
```



Hyperparameter tuning

In [82]:
```
from sklearn.datasets import make_blobs
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")

# define models and parameters
solvers = ['newton-cg', 'lbfgs', 'liblinear','saga','sag']
penalty = ['l1','l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
```

In [83]:
```
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=logReg, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train, y_train)
```

In [84]:
```
# summarize results
print("Best: %f using %s\n" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
print('Mean (Std Dev) with Parameters')
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.942949 using {'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}

Mean (Std Dev) with Parameters
0.000000 (0.000000) with: {'C': 100, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 100, 'penalty': 'l1', 'solver': 'lbfgs'}
0.942949 (0.063379) with: {'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}
0.821795 (0.081394) with: {'C': 100, 'penalty': 'l1', 'solver': 'saga'}
0.000000 (0.000000) with: {'C': 100, 'penalty': 'l1', 'solver': 'sag'}
0.853419 (0.076829) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.853419 (0.076829) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.816667 (0.077165) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.808974 (0.069641) with: {'C': 100, 'penalty': 'l2', 'solver': 'saga'}
0.829701 (0.071663) with: {'C': 100, 'penalty': 'l2', 'solver': 'sag'}
0.000000 (0.000000) with: {'C': 10, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 10, 'penalty': 'l1', 'solver': 'lbfgs'}
0.845513 (0.088132) with: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
0.800855 (0.079823) with: {'C': 10, 'penalty': 'l1', 'solver': 'saga'}
0.000000 (0.000000) with: {'C': 10, 'penalty': 'l1', 'solver': 'sag'}
0.789957 (0.080672) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.789957 (0.080672) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.753632 (0.054284) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.789957 (0.080672) with: {'C': 10, 'penalty': 'l2', 'solver': 'saga'}
0.789957 (0.080672) with: {'C': 10, 'penalty': 'l2', 'solver': 'sag'}
0.000000 (0.000000) with: {'C': 1.0, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 1.0, 'penalty': 'l1', 'solver': 'lbfgs'}
0.700427 (0.052390) with: {'C': 1.0, 'penalty': 'l1', 'solver': 'liblinear'}
0.687393 (0.070797) with: {'C': 1.0, 'penalty': 'l1', 'solver': 'saga'}
0.000000 (0.000000) with: {'C': 1.0, 'penalty': 'l1', 'solver': 'sag'}
0.631838 (0.067975) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.631838 (0.067975) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.610684 (0.061014) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.631838 (0.067975) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'saga'}
0.631838 (0.067975) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'sag'}
0.000000 (0.000000) with: {'C': 0.1, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 0.1, 'penalty': 'l1', 'solver': 'lbfgs'}
0.369231 (0.023500) with: {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
0.369231 (0.023500) with: {'C': 0.1, 'penalty': 'l1', 'solver': 'saga'}
0.000000 (0.000000) with: {'C': 0.1, 'penalty': 'l1', 'solver': 'sag'}
0.369231 (0.023500) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.369231 (0.023500) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.369231 (0.023500) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.369231 (0.023500) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'saga'}
0.369231 (0.023500) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'sag'}
0.000000 (0.000000) with: {'C': 0.01, 'penalty': 'l1', 'solver': 'newton-cg'}
0.000000 (0.000000) with: {'C': 0.01, 'penalty': 'l1', 'solver': 'lbfgs'}
0.196154 (0.034899) with: {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
0.358547 (0.048929) with: {'C': 0.01, 'penalty': 'l1', 'solver': 'saga'}
0.000000 (0.000000) with: {'C': 0.01, 'penalty': 'l1', 'solver': 'sag'}
0.369231 (0.023500) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.369231 (0.023500) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.369231 (0.023500) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
0.369231 (0.023500) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'saga'}
0.369231 (0.023500) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'sag'}
```

SVC_Apple_orange

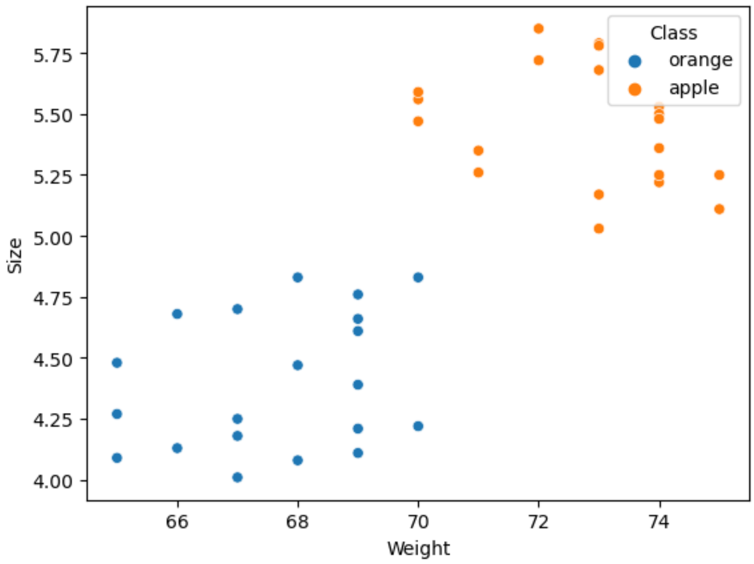
In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

In [4]:
```
data = pd.read_csv('apples_and_oranges.csv')
print(data)
```

```
    Weight  Size   Class
0       69  4.39  orange
1       69  4.21  orange
2       65  4.09  orange
3       72  5.85   apple
4       67  4.70  orange
5       73  5.68   apple
6       70  5.56   apple
7       75  5.11   apple
8       74  5.36   apple
9       65  4.27  orange
10      73  5.79   apple
11      70  5.47   apple
12      74  5.53   apple
13      68  4.47  orange
14      74  5.22   apple
15      65  4.48  orange
16      69  4.66  orange
17      75  5.25   apple
18      67  4.18  orange
19      74  5.50   apple
20      66  4.13  orange
21      70  4.83  orange
22      69  4.61  orange
23      68  4.08  orange
24      67  4.25  orange
25      71  5.35   apple
26      67  4.01  orange
27      70  4.22  orange
28      74  5.25   apple
29      71  5.26   apple
30      73  5.78   apple
31      66  4.68  orange
32      72  5.72   apple
33      73  5.17   apple
34      68  4.83  orange
35      69  4.11  orange
36      69  4.76  orange
37      74  5.48   apple
38      70  5.59   apple
39      73  5.03   apple
```

In [5]:
```
import seaborn as sns
sns.scatterplot(x="Weight",
                y="Size",
                hue="Class",
                data=data)
```

Out[5]: `<AxesSubplot:xlabel='Weight', ylabel='Size'>`



In [6]:
```
# splitting data into training and test set
training_set,test_set = train_test_split(data,test_size=0.2,random_state=1)
print("train:",training_set)
print("test:",test_set)
```

```
train:     Weight  Size   Class
19      74  5.50   apple
26      67  4.01  orange
32      72  5.72   apple
17      75  5.25   apple
30      73  5.78   apple
36      69  4.76  orange
33      73  5.17   apple
28      74  5.25   apple
4       67  4.70  orange
14      74  5.22   apple
10      73  5.79   apple
35      69  4.11  orange
23      68  4.08  orange
24      67  4.25  orange
34      68  4.83  orange
20      66  4.13  orange
18      67  4.18  orange
25      71  5.35   apple
6       70  5.56   apple
13      68  4.47  orange
7       75  5.11   apple
38      70  5.59   apple
1       69  4.21  orange
16      69  4.66  orange
0       69  4.39  orange
15      65  4.48  orange
5       73  5.68   apple
11      70  5.47   apple
9       65  4.27  orange
8       74  5.36   apple
12      74  5.53   apple
37      74  5.48   apple
test:     Weight  Size   Class
2       65  4.09  orange
31      66  4.68  orange
3       72  5.85   apple
21      70  4.83  orange
27      70  4.22  orange
29      71  5.26   apple
22      69  4.61  orange
39      73  5.03   apple
```

In [7]:
```python
x_train = training_set.iloc[:,0:2].values  # data
y_train = training_set.iloc[:,2].values  # target
x_test = test_set.iloc[:,0:2].values  # data
y_test = test_set.iloc[:,2].values  # target
print(x_train,y_train)
print(x_test,y_test)
```

```
[[74.    5.5 ]
 [67.    4.01]
 [72.    5.72]
 [75.    5.25]
 [73.    5.78]
 [69.    4.76]
 [73.    5.17]
 [74.    5.25]
 [67.    4.7 ]
 [74.    5.22]
 [73.    5.79]
 [69.    4.11]
 [68.    4.08]
 [67.    4.25]
 [68.    4.83]
 [66.    4.13]
 [67.    4.18]
 [71.    5.35]
 [70.    5.56]
 [68.    4.47]
 [75.    5.11]
 [70.    5.59]
 [69.    4.21]
 [69.    4.66]
 [69.    4.39]
 [65.    4.48]
 [73.    5.68]
 [70.    5.47]
 [65.    4.27]
 [74.    5.36]
 [74.    5.53]
 [74.    5.48]] ['apple' 'orange' 'apple' 'apple' 'apple' 'orange' 'apple' 'apple'
 'orange' 'apple' 'apple' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'apple' 'apple' 'orange' 'apple' 'apple' 'orange' 'orange'
 'orange' 'orange' 'apple' 'apple' 'orange' 'apple' 'apple' 'apple']
[[65.    4.09]
 [66.    4.68]
 [72.    5.85]
 [70.    4.83]
 [70.    4.22]
 [71.    5.26]
 [69.    4.61]
 [73.    5.03]] ['orange' 'orange' 'apple' 'orange' 'orange' 'apple' 'orange' 'apple']
```

In [8]:
```python
# fitting the data (train a model)
classifier = SVC(kernel='rbf',random_state=1,C=1,gamma='auto')
classifier.fit(x_train,y_train)
```

Out[8]:
```
SVC

SVC(C=1, gamma='auto', random_state=1)
```

In [9]:
```python
y_pred = classifier.predict(x_test)
print(y_pred)
```

```
['orange' 'orange' 'apple' 'apple' 'orange' 'apple' 'orange' 'apple']
```

In [ ]:
```python
# creating confusion matrix and accuracy calculation
cm = confusion_matrix(y_test,y_pred)
print(cm)
accuracy = float(cm.diagonal().sum())/len(y_test)
print('model accuracy is:',accuracy*100,'%')


#x1_test = [[73,6]] # for new data testing
```