1. Single variate binary classification problem

Step 1. Import packages, functions and classes

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

Step 2: Get Data

In [14]:
```
# The array x is required to be two•dimensional.To make x two•dimensional,
# you apply .reshape() with the arguments -1 to get as many rows as needed
# and 1 to get one column.
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1]) # 0s & 1s-binary classification
x, y
```

Out[14]:
```
(array([[0],
        [1],
        [2],
        [3],
        [4],
        [5],
        [6],
        [7],
        [8],
        [9]]),
 array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1]))
```

Step 3: Create a Model and Train It

In [19]:
```
# solver is a string ('liblinear' by default) that decides what
# solver to use for fitting the model.
model = LogisticRegression(solver='liblinear', random_state=0)

# Model fitting is the process of determining the coeficients b₀, b₁, …, bᵣ
# that correspond to the best value of the cost function.
model.fit(x, y)

# get the attributes of your model.
model.classes_
```

Out[19]:
```
array([0, 1])
```

In [20]:
```
# get the value of the slope b₁ and the intercept b₀ of the
# linear function f:
print(model.intercept_)
print(model.coef_)
```

```
[-1.04608067]
[[0.51491375]]
```

Step 4: Evaluate the Model

In [21]:
```
# You can check its performance with .predict_proba(), which returns the
# matrix of probabilities that the predicted output equal to zero or one:
model.predict_proba(x)
```

Out[21]:
```
array([[0.74002157, 0.25997843],
       [0.62975524, 0.37024476],
       [0.5040632 , 0.4959368 ],
       [0.37785549, 0.62214451],
       [0.26628093, 0.73371907],
       [0.17821501, 0.82178499],
       [0.11472079, 0.88527921],
       [0.07186982, 0.92813018],
       [0.04422513, 0.95577487],
       [0.02690569, 0.97309431]])
```

In [22]:
```
# get the actual predictions, based on the probability matrix
# and the values of p(x), with .predict():
model.predict(x)
```

Out[22]:
```
array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

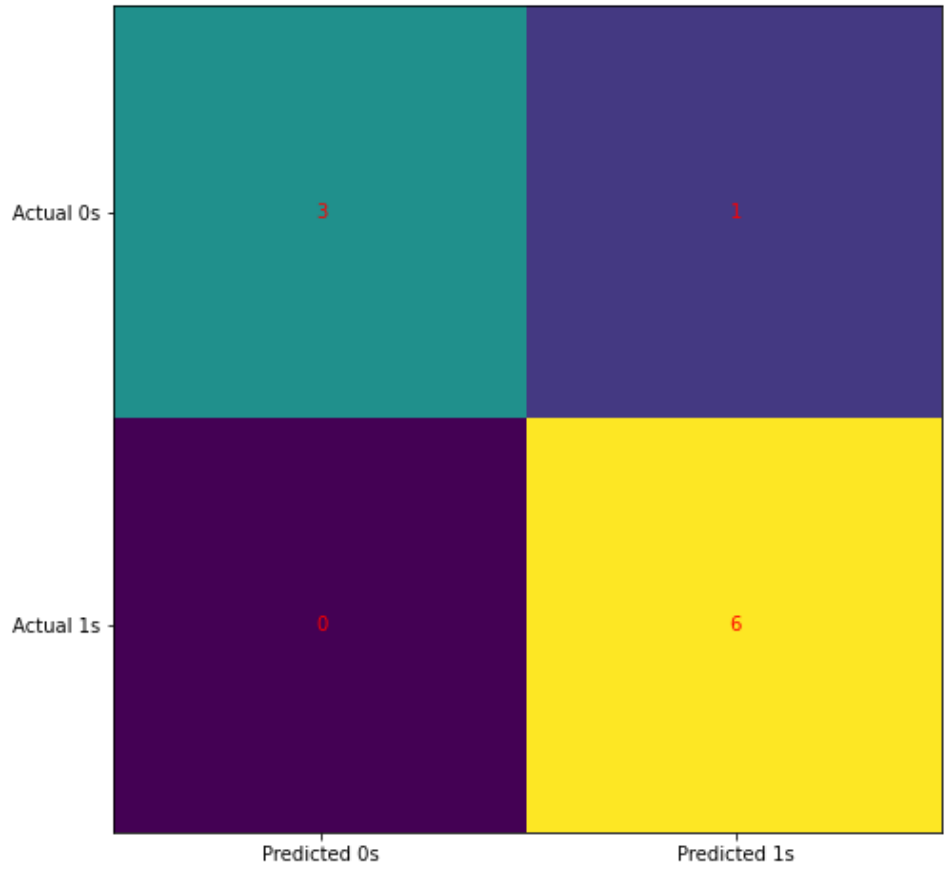
In [24]:
```
# Accuracy of the model
model.score(x, y)
```

Out[24]:
```
0.9
```

In [25]:
```
# Confusion matrix
confusion_matrix(y, model.predict(x))
```

Out[25]:
```
array([[3, 1],
       [0, 6]], dtype=int64)
```

In [39]:
```
# To visualize the confusion matrix
cm = confusion_matrix(y, model.predict(x))
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='r')
plt.show()
```



In [40]:
```
# get a more comprehensive report on the classification
print(classification_report(y, model.predict(x)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.75      0.86         4
           1       0.86      1.00      0.92         6

    accuracy                           0.90        10
   macro avg       0.93      0.88      0.89        10
weighted avg       0.91      0.90      0.90        10
```

5. Improve the Model

In [42]:
```
# improve your model by setting di呉erent parameters.
model = LogisticRegression(solver='liblinear', C=10.0, random_state=0)
model.fit(x, y)
```

Out[42]:
```
LogisticRegression(C=10.0, random_state=0, solver='liblinear')
```

In [43]:
```
model.intercept_,  model.coef_, model.predict_proba(x), model.predict(x)
```

Out[43]:
```
(array([-3.51335372]),
 array([[1.12066084]]),
 array([[0.97106534, 0.02893466],
        [0.9162684 , 0.0837316 ],
        [0.7810904 , 0.2189096 ],
        [0.53777071, 0.46222929],
        [0.27502212, 0.72497788],
        [0.11007743, 0.88992257],
        [0.03876835, 0.96123165],
        [0.01298011, 0.98701989],
        [0.0042697 , 0.9957303 ],
        [0.00139621, 0.99860379]]),
 array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1]))
```

In [47]:
```
model.score(x, y), confusion_matrix(y, model.predict(x))
```

Out[47]:
```
(1.0,
 array([[4, 0],
        [0, 6]], dtype=int64))
```

In [48]:
```
print(classification_report(y, model.predict(x)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         6

    accuracy                           1.00        10
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

In [ ]:
```

```