Importing required libraries and modules.

In [1]:
```
import numpy as np
import pandas as pd

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import roc_curve, RocCurveDisplay, roc_auc_score, log_loss
```

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing dataset from UC Irvine Machine Learning Repository.

The dataset can be found in the following link.

https://archive.ics.uci.edu/dataset/244/fertility

Directly importing the '*Fertility*' dataset using UC Irvine Machine Learning Repository's API.

In [3]:
```
!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
```

In [4]:
```
from ucimlrepo import fetch_ucirepo

# fetch dataset
fertility = fetch_ucirepo(id=244)

# data (as pandas dataframes)
X_df = fertility.data.features
y_df = fertility.data.targets

# metadata
print(fertility.metadata)

# variable information
print(fertility.variables)
```

{'uci_id': 244, 'name': 'Fertility', 'repository_url': 'https://archive.ics.uci.edu/dataset/244/fertility', 'data_url': 'https://archive.ics.uci.edu/static/public/244/data.csv', 'abstract': '100 volunteers provide a semen sample analyzed according to the WHO 2010 criteria. Sperm concentration are related to socio-demographic data, environmental factors, health status, and life habits', 'area': 'Health and Medicine', 'tasks': ['Classification', 'Regression'], 'characteristics': ['Multivariate'], 'num_instances': 100, 'num_features': 9, 'feature_types': ['Real'], 'demographics': ['Age'], 'target_col': ['diagnosis'], 'index_col': None, 'has_missing_values': 'no', 'missing_values_symbol': None, 'year_of_dataset_creation': 2012, 'last_updated': 'Fri Mar 15 2024', 'dataset_doi': '10.24432/C5Z01Z', 'creators': ['David Gil', 'Jose Girela'], 'intro_paper': {'title': 'Predicting seminal quality with artificial intelligence methods', 'authors': 'David Gil, J. L. Girela, Joaquin De Juan, M. Jose Gomez-Torres, Magnus Johnsson', 'published_in': 'Expert systems with applications', 'year': 2012, 'url': 'https://www.semanticscholar.org/paper/Predicting-seminal-quality-with-artificial-methods-Gil-Girela/92759c5ee08b9e6e7b17d1ccd48a7f8c02aba893', 'doi': None}, 'additional_info': {'summary': None, 'purpose': None, 'funded_by': None, 'instances_represent': None, 'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'Season in which the analysis was performed. \t1) winter, 2) spring, 3) Summer, 4) fall. \t(-1, -0.33, 0.33, 1) \r\n\r\nAge at the time of analysis. \t18-36 \t(0, 1) \r\n\r\nChildish diseases (ie , chicken pox, measles, mumps, polio)\t1) yes, 2) no. \t(0, 1) \r\n\r\nAccident or serious trauma \t1) yes, 2) no. \t(0, 1) \r\n\r\nSurgical intervention \t1) yes, 2) no. \t(0, 1) \r\n\r\nHigh fevers in the last year \t1) less than three months ago, 2) more than three months ago, 3) no. \t(-1, 0, 1) \r\n\r\nFrequency of alcohol consumption \t1) several times a day, 2) every day, 3) several times a week, 4) once a week, 5) hardly ever or never \t(0, 1) \r\n\r\nSmoking habit \t1) never, 2) occasional 3) daily. \t(-1, 0, 1) \r\n\r\nNumber of hours spent sitting per day \tene-16\t(0, 1) \r\n\r\nOutput: Diagnosis\tnormal (N), altered (O)\t\r\n', 'citation': None}}

	name	role	type	demographic	description	units	missing_values
0	season	Feature	Continuous	None	None	None	no
1	age	Feature	Integer	Age	None	None	no
2	child_diseases	Feature	Binary	None	None	None	no
3	accident	Feature	Binary	None	None	None	no
4	surgical_intervention	Feature	Binary	None	None	None	no
5	high_fevers	Feature	Categorical	None	None	None	no
6	alcohol	Feature	Categorical	None	None	None	no
7	smoking	Feature	Categorical	None	None	None	no
8	hrs_sitting	Feature	Integer	None	None	None	no
9	diagnosis	Target	Binary	None	None	None	no

Taking a glance at the dataset.

In [5]:
```
X_df.head()
```

Out[5]:

	season	age	child_diseases	accident	surgical_intervention	high_fevers	alcohol	smoki
0	-0.33	0.69	0	1	1	0	0.8	
1	-0.33	0.94	1	0	1	0	0.8	
2	-0.33	0.50	1	0	0	0	1.0	
3	-0.33	0.75	0	1	1	0	1.0	
4	-0.33	0.67	1	1	0	0	0.8	

In [6]:
```
y_df.head()
```

Out[6]:

	diagnosis
0	N
1	O
2	N
3	N
4	O

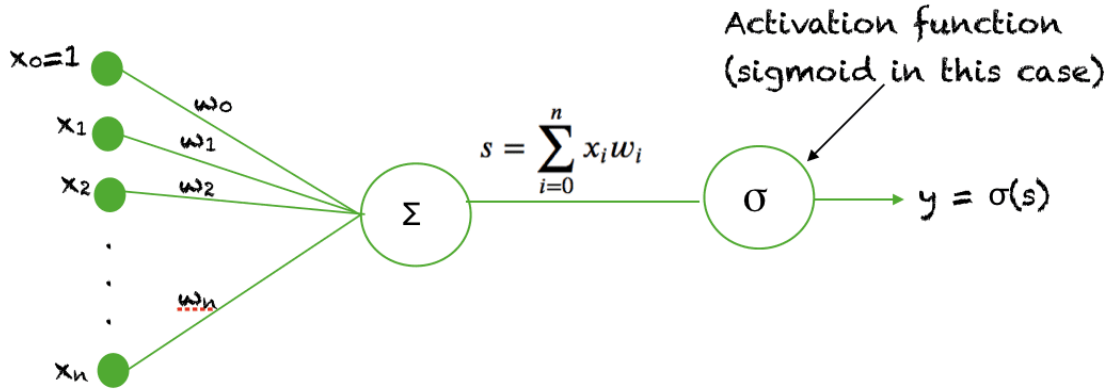
Converting dataframes to numpy arrays for ease of use in our Logistic Regression implementation from scratch.

In [7]:
```
X = X_df.to_numpy()

y = y_df.diagnosis.replace({'N': 0, 'O': 1}).to_numpy()
```

Logistic Model

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = 1 - \sigma(-x).$$



Why The Sigmoid Function Is Important In Neural Networks?

If we use a linear activation function in a neural network, then this model can only learn linearly separable problems. However, with the addition of just one hidden layer and a sigmoid activation function in the hidden layer, the neural network can easily learn a non-linearly separable problem. Using a non-linear function produces non-linear boundaries and hence, the sigmoid function can be used in neural networks for learning complex decision functions.

The only non-linear function that can be used as an activation function in a neural network is one which is monotonically increasing. So for example, sin(x) or cos(x) cannot be used as activation functions. Also, the activation function should be defined everywhere and should be continuous everywhere in the space of real numbers. The function is also required to be differentiable over the entire space of real numbers.

Typically a back propagation algorithm uses gradient descent to learn the weights of a neural network. To derive this algorithm, the derivative of the activation function is required.

The fact that the sigmoid function is monotonic, continuous and differentiable everywhere, coupled with the property that its derivative can be expressed in terms of itself, makes it easy to derive the update equations for learning the weights in a neural network when using back propagation algorithm.

Logistsic Regression implementation from scratch

Sigmoid function

In [8]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

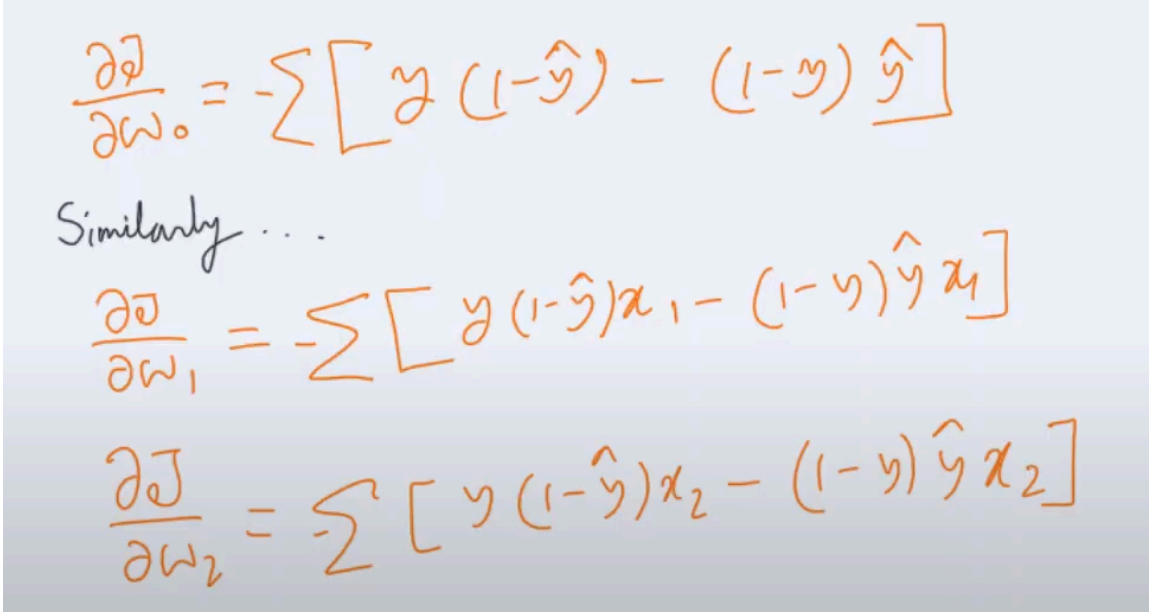
Hypothesis Function

In [9]:
```
def hx(w, X):
  ones = np.ones((X.shape[0], 1))
  X_with_bias = np.hstack([ones, X])
  z = np.dot(X_with_bias, w)
  return sigmoid(z)
```

Cost Function - Binary Cross Entropy

In [10]:
```
def cost(w, X, Y):
    y_pred = hx(w, X)
    return -np.mean(Y * np.log(y_pred) + (1 - Y) * np.log(1 - y_pred))
```

Gradient Descent



Gradient calculation function

In [11]:
```
def grad(w, X, Y):
    y_pred = hx(w, X)
    errors = y_pred - Y
    ones = np.ones((X.shape[0], 1))
    X_with_bias = np.hstack([ones, X])
    gradients = np.dot(X_with_bias.T, errors) / Y.size
    return gradients
```

Gradient descent function

In [12]:
```
def descent(w_init, lr, X, Y, max_iter=1000, tolerance=1e-6):
    w = w_init
    for j in range(max_iter):
        gradients = grad(w, X, Y)
        w_new = w - lr * gradients

        if np.linalg.norm(w_new - w, 2) < tolerance:
            print(f"Converged after {j+1} iterations.")
            return w_new

        w = w_new

        if (j + 1) % 100 == 0:
            print(f"Iteration {j+1}: Cost {cost(w, X, Y)}")
            print(f"weights: {w}")

    print("Max iterations reached without convergence.")
    return w
```

Results function

In [13]:
```
def generate_results(y_test, y_pred, y_proba=None):

  print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
  print(f"Precision: {precision_score(y_test, y_pred)}")
  print(f"Recall: {recall_score(y_test, y_pred)}")
  print(f"F1 Score: {f1_score(y_test, y_pred)}")

  cm = confusion_matrix(y_test, y_pred)
  auc = roc_auc_score(y_test, y_pred)

  print(f"\nAUC: {auc}")
  print(f"\nLog loss: {log_loss(y_test, y_pred)}\n")

  fpr, tpr, thresholds = roc_curve(y_test, y_proba if y_proba is not None else y_pr

  fig, ax = plt.subplots(1, 2, figsize=(12, 6))

  ConfusionMatrixDisplay(confusion_matrix=cm).plot(ax=ax[0])
  ax[0].set_title('Confusion Matrix')

  roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr)
  roc_display.plot(ax=ax[1])
  ax[1].plot([0, 1], [0, 1], color='green', linestyle='--')
  ax[1].set_title('ROC Curve')

  plt.tight_layout()
  plt.show()
```

Initializing Parameters

In [14]:
```
w_init = np.zeros(X.shape[1] + 1)

lr = 0.01
```

Training the Model

In [15]:
```
w_optimal = descent(w_init, lr, X, y, 1000, 1e-6)

print(f'\nOptimal weights after training Logistic Regression model:')
print(w_optimal)
```

```
Iteration 100: Cost 0.4354144569747552
weights: [-0.25213367  0.06214325 -0.16364031 -0.22027804 -0.12934375 -0.11664533
 -0.06819453 -0.21693991  0.09301912 -0.10129138]
Iteration 200: Cost 0.3798059970753205
weights: [-0.36956183  0.10632317 -0.23679483 -0.32026913 -0.19576786 -0.16247225
 -0.10838357 -0.32125459  0.13358173 -0.147873  ]
Iteration 300: Cost 0.36189020717787557
weights: [-0.4342601   0.14340965 -0.27464901 -0.37354977 -0.23816295 -0.18143329
 -0.13765424 -0.38159125  0.15516677 -0.17301437]
Iteration 400: Cost 0.35432965341921785
weights: [-0.47340047  0.17654133 -0.29543329 -0.40425669 -0.2690011  -0.18772299
 -0.16146321 -0.42064174  0.16806627 -0.18774766]
Iteration 500: Cost 0.3503795840111965
weights: [-0.4984162   0.20688017 -0.30680088 -0.42246229 -0.29340023 -0.1872079
 -0.18203867 -0.44793453  0.1763724  -0.19671612]
Iteration 600: Cost 0.3479050028352468
weights: [-0.51500724  0.23495813 -0.31256145 -0.43316069 -0.31386821 -0.18277927
 -0.20047005 -0.46821317  0.18204551 -0.20223478]
Iteration 700: Cost 0.34611759821097265
weights: [-0.5263383   0.26106513 -0.31482795 -0.43911656 -0.33178187 -0.17602981
 -0.21735719 -0.48410883  0.18614235 -0.20558894]
Iteration 800: Cost 0.3446938738939797
weights: [-0.53429279  0.28538639 -0.31485683 -0.44197446 -0.34795561 -0.16790146
 -0.233059   -0.4971863   0.18927683 -0.2075434 ]
Iteration 900: Cost 0.34348930626732527
weights: [-0.54004344  0.30805829 -0.31342988 -0.4427606  -0.36289615 -0.15897698
 -0.24780413 -0.50841954  0.19182272 -0.20857438]
Iteration 1000: Cost 0.3424337401350715
weights: [-0.54434237  0.32919307 -0.31104814 -0.44213641 -0.37693011 -0.14962702
 -0.26174597 -0.51843299  0.19401388 -0.20898728]
Max iterations reached without convergence.

Optimal weights after training Logistic Regression model:
[-0.54434237  0.32919307 -0.31104814 -0.44213641 -0.37693011 -0.14962702
 -0.26174597 -0.51843299  0.19401388 -0.20898728]
```

Testing the trained model / weights with the original data

In [16]:
```
y_prob = hx(w_optimal, X)
y_pred = np.array([1 if prob > 0.5 else 0 for prob in y_prob])
```

Generating results

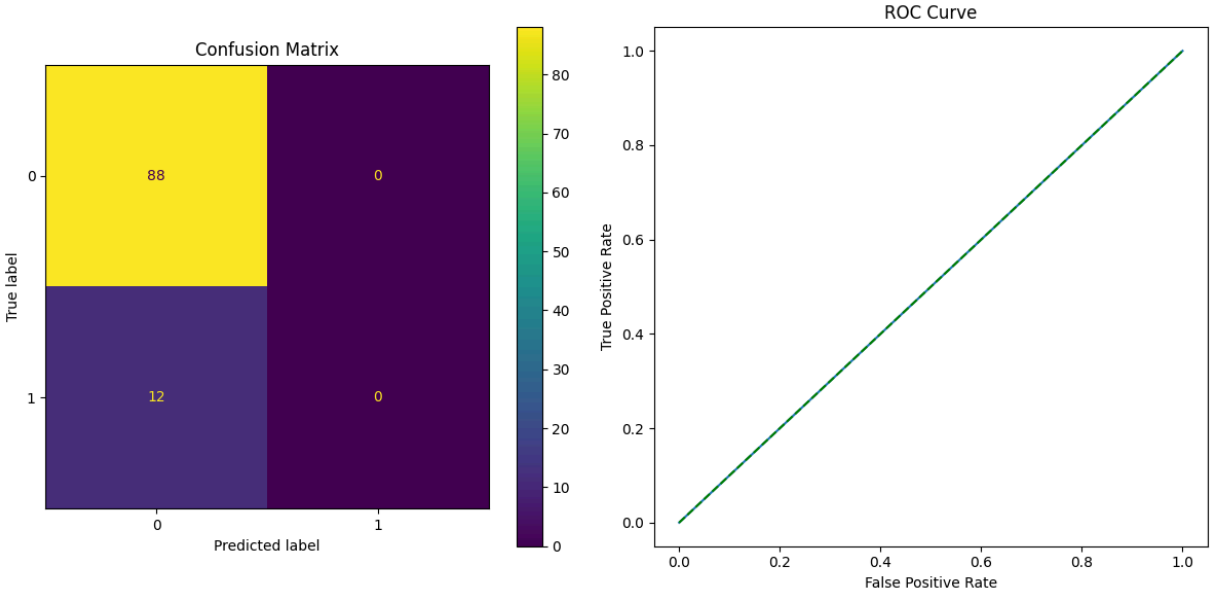
In [17]:
```
# generate_results(y, y_pred, y_prob)
generate_results(y, y_pred)
```

```
Accuracy: 0.88
Precision: 0.0
Recall: 0.0
F1 Score: 0.0

AUC: 0.5

Log loss: 4.325238406694059
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
efinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predict
ed samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



Logistic Regression implementation with Scikit learn library

In [18]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X, y)
clf.score(X, y)
```

Out[18]: 0.88

In [19]:
```
y_prob_clf = clf.predict_proba(X)
y_pred_clf = clf.predict(X)
```

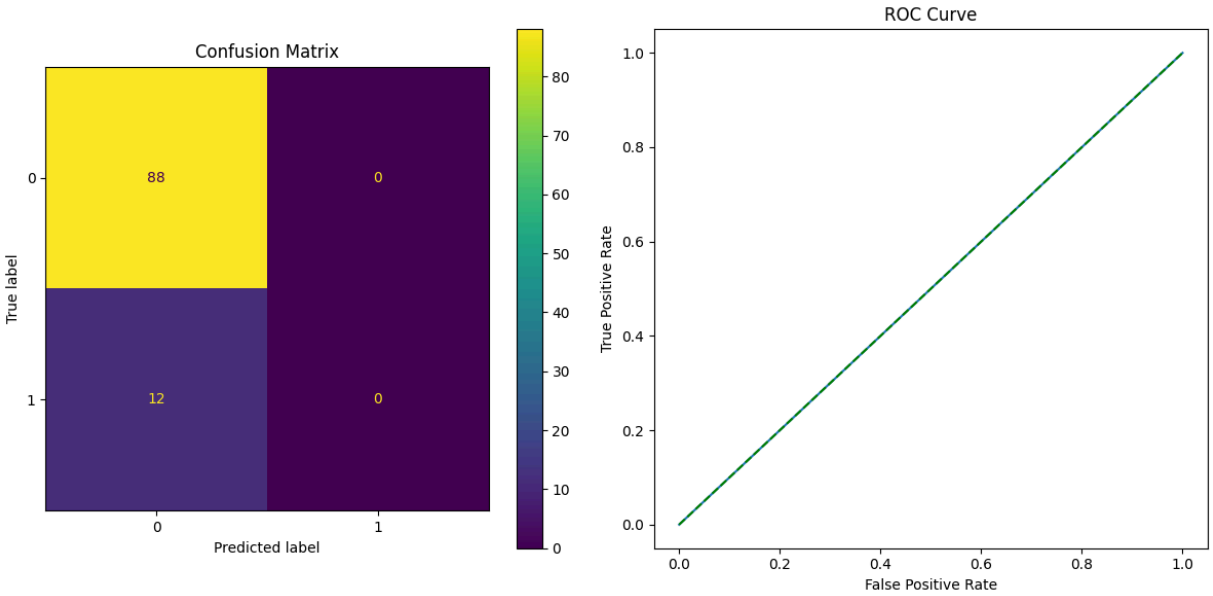
In [20]:
```
# generate_results(y, y_pred_clf, y_prob_clf[:,1])
generate_results(y, y_pred_clf)
```

```
Accuracy: 0.88
Precision: 0.0
Recall: 0.0
F1 Score: 0.0

AUC: 0.5

Log loss: 4.325238406694059
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: Und
efinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predict
ed samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



Conclusion

Both *Logistic Regression Scratch* and *Logistic Regression Scikit-Learn* implementations show exact same results and are consistent.

The consistency shows that the scratch implementation correctly mirrors the behivor of the scikit-learn library methods closely.

The `UndefinedMetricWarning` suggests that the model predicted no positive samples; hence precision is ill-defined.

Both of them fail to accurately predict 'O' or 1 class [among 'N' and 'O' classes]. This is due to the dataset being highly imbalanced with very limited numbers of data with 'O' target class making the models unable to learn their patterns.

In [20]: