Pramesh Baral (110013536)

```
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


import matplotlib.pyplot as plt
import seaborn as sn
```

Here we are importing the Fertility dataset using UC Irvine Machine Learnin Repository API.

```
!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
```

```
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
                  name     role         type demographic description units  \
0               season  Feature   Continuous        None        None  None
1                  age  Feature      Integer         Age        None  None
2       child_diseases  Feature       Binary        None        None  None
3             accident  Feature       Binary        None        None  None
4  surgical_intervention  Feature       Binary        None        None  None
5           high_fevers  Feature  Categorical        None        None  None
6              alcohol  Feature  Categorical        None        None  None
7              smoking  Feature  Categorical        None        None  None
8          hrs_sitting  Feature      Integer        None        None  None
9            diagnosis   Target       Binary        None        None  None

  missing_values
0             no
1             no
2             no
3             no
4             no
5             no
6             no
7             no
8             no
9             no
```

```
X_df.head()
```

	season	age	child_diseases	accident	surgical_intervention	high_fevers	alcohol
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

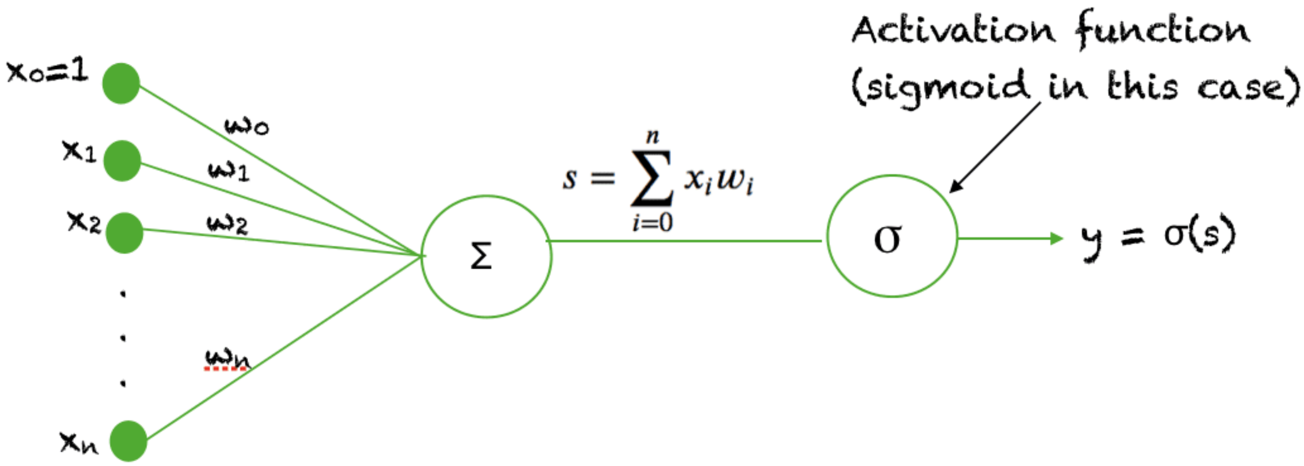
Next steps: Generate code with X_df View recommended plots

```
y_df.head()
```

	diagnosis
0	N
1	O
2	N
3	N
4	O

Next steps: Generate code with y_df View recommended plots

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




```
def sigmoid(z):
 return 1 / (1 + np.exp(-z))
```

```
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

```
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))
```

$$\frac{\partial J}{\partial \omega_0} = -\sum \left[y(1-\hat{y}) - (1-y)\hat{y} \right]$$

Similarly . . .

$$\frac{\partial J}{\partial \omega_1} = -\sum \left[y(1-\hat{y})x_1 - (1-y)\hat{y}x_1 \right]$$

$$\frac{\partial J}{\partial \omega_2} = -\sum \left[y(1-\hat{y})x_2 - (1-y)\hat{y}x_2 \right]$$

```
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


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
```

Visualizing the RESULTS

```python
def generate_results(y_test, y_pred, y_proba=None):
    # Print classification metrics
    print("Classification Metrics:")
    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)}")

    # Compute and print AUC and Log Loss
    auc = roc_auc_score(y_test, y_pred)
    print(f"\nAUC: {auc}")
    print(f"Log Loss: {log_loss(y_test, y_pred)}")

    # Compute and plot Confusion Matrix and ROC Curve
    cm = confusion_matrix(y_test, y_pred)
    fpr, tpr, thresholds = roc_curve(y_test, y_proba if y_proba is not None else y_pred)

    # Plot Confusion Matrix and ROC Curve
    fig, ax = plt.subplots(1, 2, figsize=(12, 6))

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

    # Plot ROC Curve
    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')

    # Show plots
    plt.tight_layout()
    plt.show()
```

```python
w_init = np.zeros(X.shape[1] + 1)
lr = 0.01
```

```python
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]
```

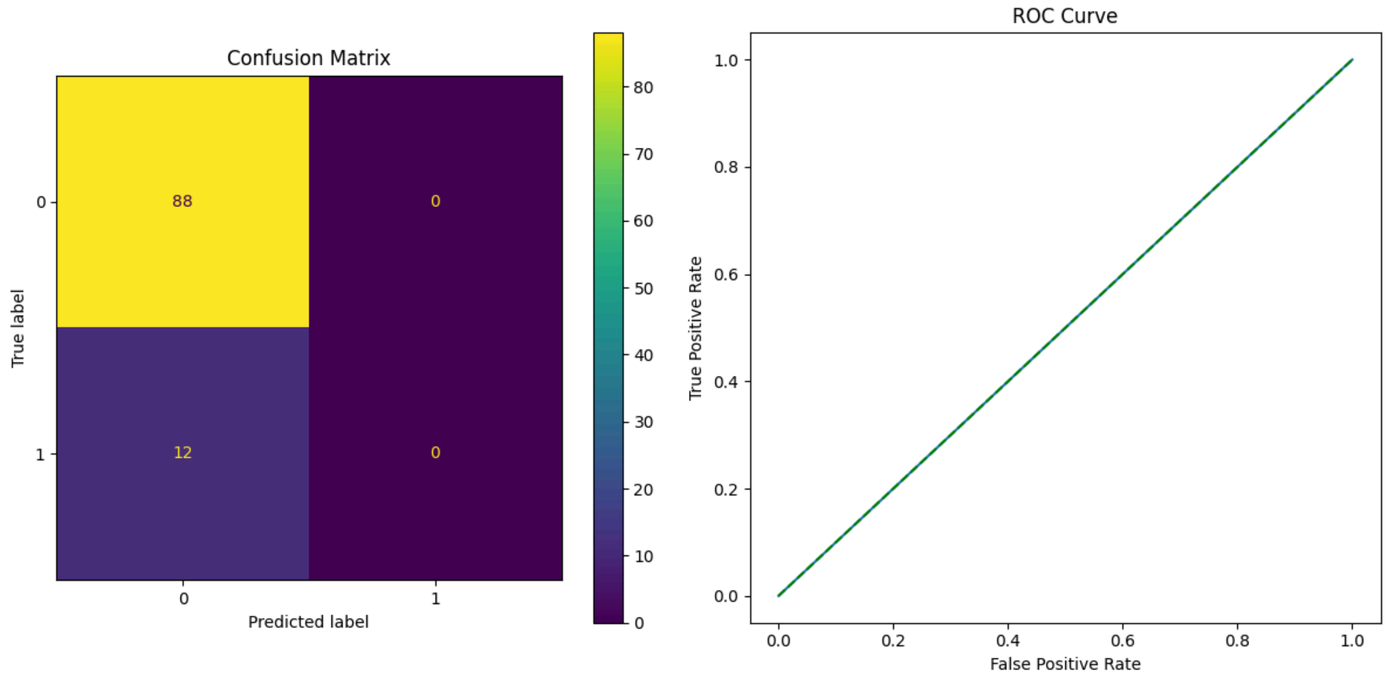
```python
y_prob = hx(w_optimal, X)
```

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

```
#generate_results(y, y_pred, y_prob)
generate_results(y, y_pred)
```

```
Classification Metrics:
  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: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```



Logistic Regression implementation with Scikit learn library.

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X, y)
clf.score(X, y)
```

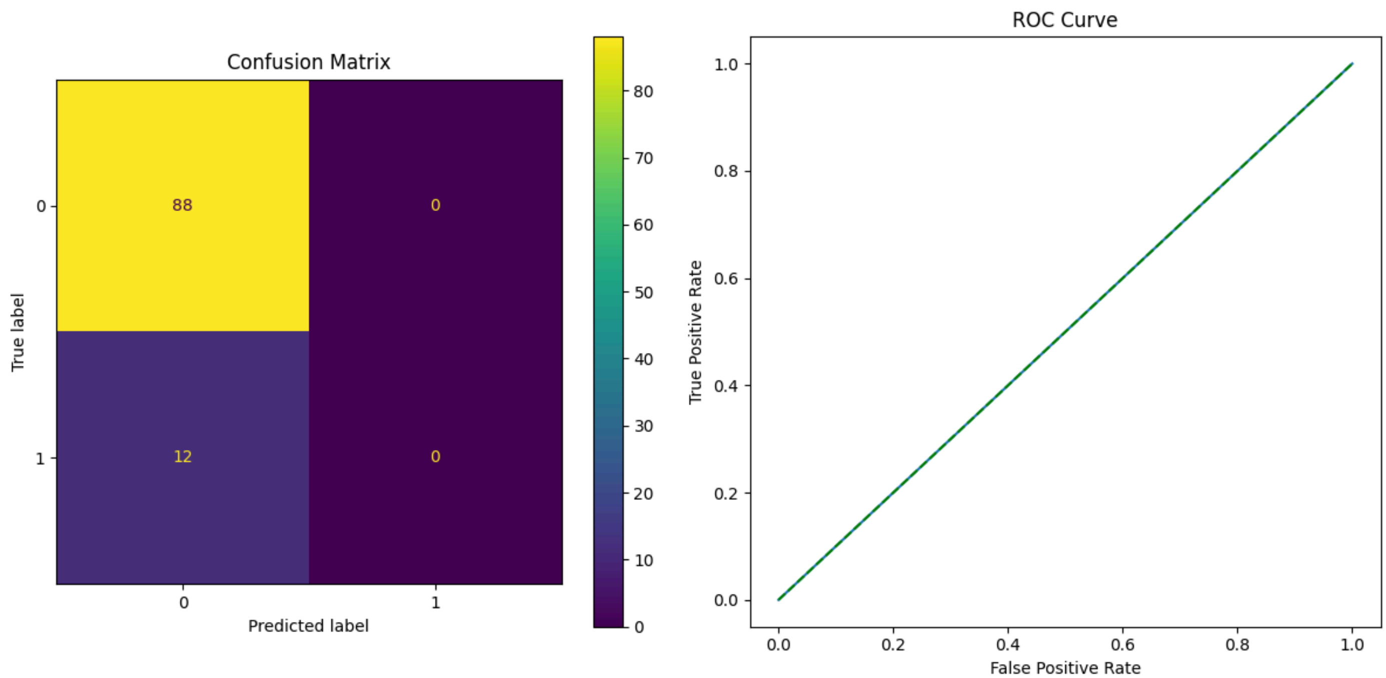
```
0.88
```

```
y_prob_clf = clf.predict_proba(X)
y_pred_clf = clf.predict(X)
```

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

```
Classification Metrics:
  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: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```



Conclusion: The conclusions drawn from both the Logistic Regression Scratch and Logistic Regression Scikit-Learn implementations highlight their identical results and consistent performance.

This consistency indicates that the scratch implementation faithfully replicates the behavior of the scikit-learn library methods.

The occurrence of the UndefinedMetricWarning suggests that the model made no positive predictions, leading to an undefined precision metric.

However, both implementations struggled to accurately predict the 'O' or 1 class among the 'N' and 'O' classes due to the dataset's highly imbalanced nature, with limited data points for the 'O' target class, which hindered the models' ability to learn their patterns effectively.

How can I measure the performance of my model?

We can measure the performance of our machine learning model using various evaluation metrics. Some common metrics include:

Accuracy: Measures the proportion of correct predictions out of the total predictions made by the model.

Confusion Matrix: A table that summarizes the performance of a classification model by showing the counts of true positives, true negatives, false positives, and false negatives.

Precision: Also known as positive predictive value, it measures the accuracy of positive predictions made by the model.

Recall (Sensitivity): Measures the ability of the model to correctly identify true positives from all actual positives.

F1 Score: The harmonic mean of precision and recall, providing a balance between precision and recall.

ROC (Receiver Operating Characteristic) Curve: A graphical representation of the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings.

AUC (Area Under the ROC Curve): Measures the entire two-dimensional area underneath the ROC curve, indicating the model's ability to distinguish between classes.

Log Loss (Logarithmic Loss): Measures the performance of a classification model where the predicted output is a probability value between 0 and 1. Lower log loss values indicate better performance.

3 What are: Accuracy, Confusion Matrix, Precision, Recall & F1 Score, ROC & AUC