Task#01:

Use HeartDisease dataset and apply LinearRegression and then pass the result to sigmoid function (write from scratch) and then compare the accuracy of both models.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
```

```
dataframe = pd.read_csv('heart_disease_dataset_UCI.csv')
```

```
dataframe
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

```
X = dataframe.drop('target', axis=1)
```

```
X
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 13 columns

```
y = dataframe['target']
```

```
y
```

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
```

```
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
linearRegressionModel = LinearRegression()
linearRegressionModel.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
linearRegressionPrediction = linearRegressionModel.predict(X_test)
```

```
def sigmoidFunction(x):
    return 1 / (1 + np.exp(-x))
```

```
logisticRegressionPrediction = sigmoidFunction(linearRegressionPrediction)
```

```
logisticRegressionBinaryPrediction = np.round(logisticRegressionPrediction)
```

```
linearRegressionAccuracy = accuracy_score(y_test, np.round(linearRegressionPrediction))
```

```
logisticRegressionAccuracy = accuracy_score(y_test, logisticRegressionBinaryPrediction)
```

```
print(f"Linear Regression Accuracy: {linearRegressionAccuracy*100}")
print(f"Logistic Regression Accuracy: {logisticRegressionAccuracy*100}")
```

```
Linear Regression Accuracy: 86.88524590163934
Logistic Regression Accuracy: 72.1311475409836
```

Task#02:

Calculate binary cross entropy loss on the above experiment. Write from scratch.

```
def binaryCrossEntropyLoss(y_true, y_pred):
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return loss
```

```
linearRegressionLoss = binaryCrossEntropyLoss(y_test, linearRegressionPrediction)
```

```
logisticRegressionLoss = binaryCrossEntropyLoss(y_test, logisticRegressionPrediction)
```

```
print(f"Binary Cross Entropy Loss (Linear Regression): {linearRegressionLoss}")
print(f"Binary Cross Entropy Loss (Logistic Regression): {logisticRegressionLoss}")
```

```
Binary Cross Entropy Loss (Linear Regression): 0.35139237326122025
Binary Cross Entropy Loss (Logistic Regression): 0.5733253697803196
```

Task#03:

Download a new dataset from UCI Repository https://archive.ics.uci.edu/ml/datasets.php and evaluate its accuracy on 5 cross fold.

```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.3-py3-none-any.whl (7.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.3
```

```
from ucimlrepo import fetch_ucirepo
```

```
breastCancer = fetch_ucirepo(id=17)


breastCancer
```

```
{'data': {'ids':            ID
  0      842302
  1      842517
  2    84300903
  3    84348301
  4    84358402
  ..        ...
  564    926424
  565    926682
  566    926954
  567    927241
  568     92751

  [569 rows x 1 columns],
  'features':      radius1  texture1  perimeter1   area1  smoothness1  compactness1  \
  0      17.99     10.38      122.80  1001.0      0.11840       0.27760
  1      20.57     17.77      132.90  1326.0      0.08474       0.07864
  2      19.69     21.25      130.00  1203.0      0.10960       0.15990
  3      11.42     20.38       77.58   386.1      0.14250       0.28390
  4      20.29     14.34      135.10  1297.0      0.10030       0.13280
  ..       ...       ...         ...     ...          ...           ...
  564    21.56     22.39      142.00  1479.0      0.11100       0.11590
  565    20.13     28.25      131.20  1261.0      0.09780       0.10340
  566    16.60     28.08      108.30   858.1      0.08455       0.10230
  567    20.60     29.33      140.10  1265.0      0.11780       0.27700
  568     7.76     24.54       47.92   181.0      0.05263       0.04362

       concavity1  concave_points1  symmetry1  fractal_dimension1  ...  radius3  \
  0       0.30010          0.14710     0.2419             0.07871  ...   25.380
  1       0.08690          0.07017     0.1812             0.05667  ...   24.990
  2       0.19740          0.12790     0.2069             0.05999  ...   23.570
  3       0.24140          0.10520     0.2597             0.09744  ...   14.910
  4       0.19800          0.10430     0.1809             0.05883  ...   22.540
  ..          ...              ...        ...                 ...  ...      ...
  564     0.24390          0.13890     0.1726             0.05623  ...   25.450
  565     0.14400          0.09791     0.1752             0.05533  ...   23.690
  566     0.09251          0.05302     0.1590             0.05648  ...   18.980
  567     0.35140          0.15200     0.2397             0.07016  ...   25.740
  568     0.00000          0.00000     0.1587             0.05884  ...    9.456

       texture3  perimeter3   area3  smoothness3  compactness3  concavity3  \
  0       17.33      184.60  2019.0      0.16220       0.66560      0.7119
  1       23.41      158.80  1956.0      0.12380       0.18660      0.2416
  2       25.53      152.50  1709.0      0.14440       0.42450      0.4504
  3       26.50       98.87   567.7      0.20980       0.86630      0.6869
  4       16.67      152.20  1575.0      0.13740       0.20500      0.4000
  ..        ...         ...     ...          ...           ...         ...
  564     26.40      166.10  2027.0      0.14100       0.21130      0.4107
  565     38.25      155.00  1731.0      0.11660       0.19220      0.3215
  566     34.12      126.70  1124.0      0.11390       0.30940      0.3403
  567     39.42      184.60  1821.0      0.16500       0.86810      0.9387
  568     30.37       59.16   268.6      0.08996       0.06444      0.0000

       concave_points3  symmetry3  fractal_dimension3
  0             0.2654     0.4601             0.11890
  1             0.1860     0.2750             0.08902
  2             0.2430     0.3613             0.08758
  3             0.2575     0.6638             0.17300
```

```
X = breastCancer.data.features
y = breastCancer.data.targets


X
```

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	cor
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	
...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	

569 rows × 30 columns

```
y
```

	Diagnosis
0	M
1	M
2	M
3	M
4	M
...	...
564	M
565	M
566	M
567	M
568	B

569 rows × 1 columns

```
import numpy as np
import pandas as pd
from sklearn.model_selection import cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


classifier = RandomForestClassifier(random_state=42)


k_fold = KFold(n_splits=5, shuffle=True, random_state=42)


cvAccuracy = cross_val_score(classifier, X, y, cv=k_fold, scoring='accuracy')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was pas
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was pas
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was pas
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was pas
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was pas
  estimator.fit(X_train, y_train, **fit_params)
```

```
for i, accuracy in enumerate(cvAccuracy, start=1):
    print(f"Fold {i} Accuracy: {accuracy*100}")
```

```
Fold 1 Accuracy: 96.49122807017544
Fold 2 Accuracy: 96.49122807017544
Fold 3 Accuracy: 93.85964912280701
```

```
Fold 4 Accuracy: 96.49122807017544
Fold 5 Accuracy: 96.46017699115043
```

```
meanAccuracy = np.mean(cvAccuracy)
```

```
print(f"Mean Accuracy: {meanAccuracy*100}")
```

```
Mean Accuracy: 95.95870206489676
```

Task#05:

Consider a logistic regression model with w1=0.5 and w2=0.31 and b=0.09. X1 = 5, X2 = 3 and actual y is 1. Calculate this numerical by hand and verify your answers by coding the functions below. Write below functions from scratch: ● ForwardPropagation() ● LossCalculation() ● BackwardPropagation() # This function must update the old weights ● MainLoop which must iterate 5 times and call all the above functions.

```
w1 = 0.5
w2 = 0.31
b = 0.09

X = [5, 3]
y_true = 1

learning_rate = 0.01
iterations = 5


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


def ForwardPropagation(X, w1, w2, b):
    linearCombination = X[0] * w1 + X[1] * w2 + b
    output = sigmoid(linearCombination)
    return output


def LossCalculation(y_true, y_pred):
    loss = - (y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return loss


def BackwardPropogation(X, y_true, y_pred, w1, w2, b, learning_rate):
    dw1 = X[0] * (y_pred - y_true)
    dw2 = X[1] * (y_pred - y_true)
    db = y_pred - y_true

    w1 -= learning_rate * dw1
    w2 -= learning_rate * dw2
    b -= learning_rate * db

    return w1, w2, b


def main_loop(X, y_true, w1, w2, b, learning_rate, iterations):
    for i in range(iterations):
        y_pred = ForwardPropagation(X, w1, w2, b)

        loss = LossCalculation(y_true, y_pred)
        print(f"\nIteration {i + 1}, Y Predicited {round(y_pred, 4)}, Loss: {round(loss, 4)}")

        w1, w2, b = BackwardPropogation(X, y_true, y_pred, w1, w2, b, learning_rate)
        print(f"w1 {round(w1, 4)}, w2 {round(w2, 4)}, b {round(b, 4)}")

    return w1, w2, b


final_w1, final_w2, final_b = main_loop(X, y_true, w1, w2, b, learning_rate, iterations)
```

```
Iteration 1, Y Predicited 0.9713, Loss: 0.0292
w1 0.5014, w2 0.3109, b 0.0903

Iteration 2, Y Predicited 0.9715, Loss: 0.0289
w1 0.5029, w2 0.3117, b 0.0906

Iteration 3, Y Predicited 0.9718, Loss: 0.0286
w1 0.5043, w2 0.3126, b 0.0909

Iteration 4, Y Predicited 0.9721, Loss: 0.0283
w1 0.5057, w2 0.3134, b 0.0911
```

```
Iteration 5, Y Predicited 0.9723, Loss: 0.0281
w1 0.507, w2 0.3142, b 0.0914
```

```
print("Final Weights and Bias:")
print(f"w1: {final_w1} \nw2: {final_w2} \nb: {final_b}")
```

```
Final Weights and Bias:
w1: 0.5070499368603588
w2: 0.3142299621162153
b: 0.09140998737207177
```