***Измените функцию calc_logloss так, чтобы нули по возможности не попадали в np.log (как вариант - np.clip).**

Подберите аргументы функции eval_LR_model для логистической регрессии таким образом, чтобы log loss был минимальным.

Создайте функцию calc_pred_proba, возвращающую предсказанную вероятность класса 1 (на вход подаются веса, которые уже посчитаны функцией eval_LR_model и X, на выходе - массив y_pred_proba).

Создайте функцию calc_pred, возвращающую предсказанный класс (на вход подаются веса, которые уже посчитаны функцией eval_LR_model и X, на выходе - массив y_pred).

Посчитайте accuracy, матрицу ошибок, precision и recall, а также F1-score.

Могла ли модель переобучиться? Почему?

***Создайте функции eval_LR_model_l1 и eval_LR_model_l2 с применением L1 и L2 регуляризации соответственно.**

In [92]:

```
import numpy as np

X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
              [1, 1, 2, 1, 3, 0, 5, 10, 1, 2],   # стаж
              [500, 700, 750, 600, 1450,         # средняя стоимость заняти
я
               800, 1500, 2000, 450, 1000],
              [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]], dtype = np.float64) # квали
фикация репетитора

y = np.array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1]) # подходит или нет репетитор
```

In [60]:

```
def calc_std_feat(x):
    res = (x - x.mean()) / x.std()
    return res

def calc_mse(y, y_pred):
    err = np.mean((y - y_pred)**2)
    return err

def calc_logloss(y, y_pred):
    eps = 1e-15
    y_pred = np.clip(y_pred, eps, 1 - eps)
    err = np.mean(- y * np.log(y_pred) - (1.0 - y) * np.log(1.0 - y_pred))
```

```
    return err

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

def eval_LR_model(X, y, iterations, alpha=1e-4, lambda_=1e-3, penalty=None
):
    np.random.seed(42)
    w = np.random.randn(X.shape[0])
    n = X.shape[1]
    min_err_logloss = float('inf')
    for i in range(1, iterations + 1):

        z = np.dot(w, X)
        y_pred = sigmoid(z)
        err_logloss = calc_logloss(y, y_pred)
        if min(err_logloss, min_err_logloss) < min_err_logloss:
            min_err_logloss = err_logloss
        else:
            break

        y_pred = np.dot(w, X)
        err_mse = calc_mse(y, y_pred)

        if penalty == 'l2':
            w -= alpha * (1/n * np.dot((y_pred - y), X.T) + 2 * lambda_ *
w)
        elif penalty == 'l1':
            w -= alpha * (1/n * np.dot((y_pred - y), X.T) + lambda_ * abs(
w))
        else:
            w -= alpha * (1/n * np.dot((y_pred - y), X.T))
    return w, min_err_logloss
```

In [63]:

```
X_st = X.copy()
X_st[2, :] = calc_std_feat(X[2, :])

best_params = dict()
min_err_logloss = float('inf')

header = ('\t').join(['n_iter','alpha','penalty','err_logloss'])
print(f'{header}\n')

for n_iter in [500, 800, 1000, 1200, 1400]:
    for alpha_ in [1e-1, 1e-2, 1e-3, 1e-4, 1e-5]:
        for penalty in ['l1', 'l2', None]:
            w, err_logloss = eval_LR_model(X_st, y, iterations=n_iter, alp
ha=alpha_, lambda_=1e-3, penalty=penalty)
            if min(err_logloss, min_err_logloss) < min_err_logloss:
                min_err_logloss = err_logloss
                best_params = {
                    'err_logloss': err_logloss,
                    'weights': w,
                    'params': {
                        'n_iter': n_iter,
                        'alpha': alpha_,
                        'penalty': penalty,
```

```
                    }
                }
            print(f'{n_iter}\t{alpha_}\t{penalty}\t{err_logloss}')
print(f'\nBest params: {best_params}')
```

```
n_iter  alpha   penalty err_logloss

500     0.1     l1      0.7812040218425931
500     0.1     l2      0.7812499548693961
500     0.1     None    0.7811211780116794
500     0.01    l1      0.5924901479443765
500     0.01    l2      0.5925957172495678
500     0.01    None    0.5924438857850406
500     0.001   l1      0.60115125781248
500     0.001   l2      0.6011783423449634
500     0.001   None    0.6011526187544622
500     0.0001  l1      0.7830961063600291
500     0.0001  l2      0.7831007049991416
500     0.0001  None    0.7831301378886715
500     1e-05   l1      1.1512387324105906
500     1e-05   l2      1.1512363784057977
500     1e-05   None    1.1512446696677174
800     0.1     l1      0.7812040218425931
800     0.1     l2      0.7812499548693961
800     0.1     None    0.7811211780116794
800     0.01    l1      0.5924901479443765
800     0.01    l2      0.5925957172495678
800     0.01    None    0.5924438857850406
800     0.001   l1      0.5948631899624397
800     0.001   l2      0.5949206496380038
800     0.001   None    0.5948453040243635
800     0.0001  l1      0.6931530304542741
800     0.0001  l2      0.6931633850641272
800     0.0001  None    0.6931865473756946
800     1e-05   l1      1.1132117788888256
800     1e-05   l2      1.1132085415086708
800     1e-05   None    1.113221009182572
1000    0.1     l1      0.7812040218425931
1000    0.1     l2      0.7812499548693961
1000    0.1     None    0.7811211780116794
1000    0.01    l1      0.5924901479443765
1000    0.01    l2      0.5925957172495678
1000    0.01    None    0.5924438857850406
1000    0.001   l1      0.5931618723297586
1000    0.001   l2      0.5932408933027126
1000    0.001   None    0.593131173956033
1000    0.0001  l1      0.6639738186304763
1000    0.0001  l2      0.6639849915614102
1000    0.0001  None    0.6640043180338439
1000    1e-05   l1      1.089180735951696
1000    1e-05   l2      1.0891771047238443
1000    1e-05   None    1.0891920468526797
1200    0.1     l1      0.7812040218425931
1200    0.1     l2      0.7812499548693961
1200    0.1     None    0.7811211780116794
1200    0.01    l1      0.5924901479443765
1200    0.01    l2      0.5925957172495678
1200    0.01    None    0.5924438857850406
1200    0.001   l1      0.5925753576032513
1200    0.001   l2      0.5926755281012894
```

```
1200    0.001   None    0.5925323068691
1200    0.0001  l1      0.6467657543179413
1200    0.0001  l2      0.6467766612433915
1200    0.0001  None    0.6467931287621036
1200    1e-05   l1      1.066164924745117
1200    1e-05   l2      1.0661610414977671
1200    1e-05   None    1.066178224578054
1400    0.1     l1      0.7812040218425931
1400    0.1     l2      0.7812499548693961
1400    0.1     None    0.7811211780116794
1400    0.01    l1      0.5924901479443765
1400    0.01    l2      0.5925957172495678
1400    0.01    None    0.5924438857850406
1400    0.001   l1      0.5925490452231488
1400    0.001   l2      0.5926548581303688
1400    0.001   None    0.5925023889219063
1400    0.0001  l1      0.6360656837785937
1400    0.0001  l2      0.6360760297761928
1400    0.0001  None    0.6360903278440416
1400    1e-05   l1      1.0441308612203168
1400    1e-05   l2      1.0441268560300292
1400    1e-05   None    1.0441460583136046

Best params: {'err_logloss': 0.5924438857850406, 'weights': a
rray([ 0.1544927 , -0.47062209,  0.62442963,  0.96986182]),
'params': {'n_iter': 500, 'alpha': 0.01, 'penalty': None}}
```

In [104]:

```
def calc_pred_proba(w, X):
    w = w.reshape(X.shape[0], 1)
    return sigmoid(np.dot(w.T, X))

def calc_pred(w, X):
    w = w.reshape(X.shape[0], 1)
    y_pred = np.zeros((1, X.shape[1]))
    y_ = sigmoid(np.dot(w.T, X))

    for i in range(y_.shape[1]):
        if (y_[:,i] > 0.5):
            y_pred[:, i] = 1
        elif (y_[:,i] <= 0.5):
            y_pred[:, i] = 0

    return y_pred[0]

def calc_metrics(y, y_pred):
    l = 0
    tp = 0
    fp = 0
    fn = 0
    tn = 0
    for idx, i in enumerate(y):
        if y[idx] == y_pred[idx]:
            l += 1
        if y[idx] == 1 and y_pred[idx] == 1:
            tp += 1
        if y[idx] == 0 and y_pred[idx] == 0:
            tn += 1
        if y[idx] == 1 and y_pred[idx] == 0:
```

```
            fn += 1
        if y[idx] == 0 and y_pred[idx] == 1:
            fp += 1

    accuracy = (l / len(y))
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    f1 = 2 * precision * recall / (precision + recall)
    matrix = np.array([[tp, fp], [fn, tn]])

    return accuracy, precision, recall, f1, matrix
```

In [105]:

```
y_pred = calc_pred(best_params["weights"], X_st)
y_pred_proba = calc_pred_proba(best_params["weights"], X_st)

accuracy, precision, recall, f1, matrix = calc_metrics(y.tolist(), y_pred.
tolist())

print(f'Predicted classes: \n{y_pred}\n')
print(f'Prediction probability: \n{y_pred_proba}\n')
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-score: {f1}')
print(f'Matrix: \n{matrix}')
```

```
Predicted classes:
[1. 1. 1. 1. 1. 1. 1. 0. 0. 1.]

Prediction probability:
[[0.51051035 0.57434451 0.70333437 0.54260229 0.78482936 0.71
074333
  0.80011934 0.42014182 0.49441516 0.76587541]]

Accuracy: 0.5
Precision: 0.5
Recall: 0.8
F1-score: 0.6153846153846154
Matrix:
[[4 4]
 [1 1]]
```

Могла ли модель переобучиться? Почему?

Могла, так как имеем высокую долю false positive результатов.