Урок 3. Логистическая регрессия. Log Loss

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set(style="whitegrid")
sns.set_context("paper", font_scale=2)
```

```
%matplotlib inline
plt.style.use('seaborn-ticks')
plt.rcParams.update({'font.size': 14})
```

```
### Logistic Regression
```

```
### Домашние задания
```

```
from scipy.stats import mode
```

```
df = pd.read_csv('/content/framingham1000.csv')
X = df[['male', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'glucose']]
y = df[['TenYearCHD']]
values = {'education': float(X.education.mode()),
         'cigsPerDay': float(X.cigsPerDay.mode()),
         'glucose': float(X.glucose.mode())}
X = X.fillna(values)
```

```
y.isna().sum(), X.isna().sum()
```

```
(TenYearCHD    0
 dtype: int64, male             0
 age              0
 education        0
 currentSmoker    0
 cigsPerDay       0
 glucose          0
 dtype: int64)
```

```
def calc_std_feat(x):
    res = (x - x.mean()) / x.std()
    return res
X_st = X.copy()
cols = ['age', 'education', 'cigsPerDay', 'glucose']
for col in cols:
    X_st[col] = calc_std_feat(X_st[col])

X_st.T.values.shape
```

```
(6, 999)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_st, y, test_size = 0.4,random_state=42)
X_train, X_test, y_train, y_test = X_train.T, X_test.T, y_train.T, y_test.T
```

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

$$Logloss = -y\ln(p) - (1-y)\ln(1-p)$$

```
def calc_logloss(y, y_pred):
    y_pred = np.clip(y_pred, 1e-12, 1)
    err = np.mean(- y * np.log(y_pred) - (1.0 - y) * np.log(1.0 - y_pred))
    return err
```

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

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

```
def eval_LR_model(X, y, iterations, alpha=1e-4):
    np.random.seed(42)
    w = np.random.randn(X.shape[0])
    n = X.shape[1]
    errors = []

    for i in range(1, iterations + 1):
        z = np.dot(w, X)
        y_pred = sigmoid(z)
        y_pred = np.clip(y_pred, 1e-6, 0.9999)
        err = calc_logloss(y, y_pred)
        w = w - alpha * (1/n * np.dot((y_pred - y), X.T))

        if i % (iterations / 10) == 0:
            errors.append(err)
            print(i, w, err)

    return w, errors
```

```
delim = '-' * 8
best_alpha = 0
err = np.inf
alphas = [1e-6, 1e-4, 1e-2, 0.1, 0.111, 8e-4, 8e-9]
# alphas = [0.1, 0.1999, 0.111, 0.5, 0.9, 1.1, 1.5, 1.9, 2.6, 10]
for a in alphas:
    print(delim + f' α = {a} ' + delim)
    w, errors = eval_LR_model(X_train.values, y_train.values, iterations=1000, alpha=a)

    if errors[-1] < err:
        err = errors[-1]
        best_alpha = a
print(f'logloss: {err}\tbest_alpha: {best_alpha}')
```

```
900 [[ 0.47591557 -0.12414055  0.63779655  1.49360556 -0.23975537 -0.22279904]] 1.2385505601714886
1000 [[ 0.47361599 -0.12259134  0.63670252  1.49035252 -0.24035989 -0.22155059]] 1.236411377832733
-------- α = 0.01 --------
100 [[ 2.76628982e-01  4.66199945e-04  5.42765966e-01  1.21163478e+00
  -2.79929113e-01 -1.18954206e-01]] 1.0683298388586462
200 [[ 0.08338301  0.1003542   0.45083509  0.93695589 -0.29006221 -0.02884796]] 0.9280836076969348
300 [[-0.08226335  0.16773569  0.37389012  0.69845529 -0.27070203  0.03829022]] 0.8278344953097787
400 [[-0.2225823   0.21123022  0.31178344  0.4919489  -0.2308064   0.08737544]] 0.7553712291650156
500 [[-0.3410746   0.23865299  0.26275526  0.31218332 -0.17846757  0.1232637 ]] 0.7015930056241297
600 [[-0.44128739  0.25586932  0.22447258  0.15429744 -0.11962203  0.14973439]] 0.6605429759812218
700 [[-0.52634612  0.26687992  0.19469258  0.01426905 -0.05820065  0.16950747]] 0.6284252618906847
800 [[-0.59885668  0.27427573  0.1715204  -0.11109487  0.00333689  0.18449709]] 0.6027762852541189
900 [[-0.6609441   0.27968082  0.15344909 -0.224299    0.06351534  0.1960457 ]] 0.5819437625316489
1000 [[-0.714328    0.28408645  0.13931452 -0.32730985  0.1214933   0.20509744]] 0.5647815255619336
-------- α = 0.1 --------
100 [[-0.71792602  0.28466354  0.13837967 -0.33111759  0.12257619  0.20610375]] 0.5655576245122945
200 [[-0.98171684  0.32924131  0.09264757 -1.03051926  0.55725427  0.24282899]] 0.4857824145661484
300 [[-1.03880719  0.3788761   0.09334568 -1.43871816  0.8045256   0.25431802]] 0.46174962181573836
400 [[-1.02880581  0.41855831  0.09913351 -1.71691824  0.95894104  0.25904816]] 0.4512737495806093
500 [[-0.99379655  0.4476581   0.10409379 -1.92146936  1.06326318  0.26055831]] 0.4457260732143472
600 [[-0.95046538  0.46873083  0.10754861 -2.07888283  1.1377573   0.26044513]] 0.44243063910326486
700 [[-0.9062359   0.48414337  0.10977597 -2.2037368   1.19319842  0.25955214]] 0.4403293412435759
800 [[-0.8644747   0.49562665  0.11114766 -2.30488596  1.23582114  0.25833541]] 0.4389279586823716
900 [[-0.82658845  0.50437161  0.1119587  -2.38810166  1.26945381  0.25703879]] 0.437965585067343
1000 [[-0.79299649  0.51118281  0.11241476 -2.45735166  1.29655419  0.25578741]] 0.4372914510867728
-------- α = 0.111 --------
100 [[-0.76854305  0.28892651  0.12635826 -0.43482035  0.18376209  0.21392238]] 0.5498438184971888
200 [[-1.00445403  0.34062357  0.09148255 -1.13691971  0.62355856  0.24633092]] 0.47832882921858083
300 [[-1.0401484   0.39329537  0.09519571 -1.54142096  0.86313167  0.25640389]] 0.4574402328078082
400 [[-1.0152497   0.4325675   0.10150815 -1.81423955  1.00963952  0.26000188]] 0.44844460571039557
500 [[-0.97041443  0.46011     0.10617556 -2.01294153  1.10719866  0.26063757]] 0.44372197444195405
600 [[-0.92110923  0.47944454  0.10913743 -2.16445207  1.1761139   0.25991261]] 0.44094713012781045
700 [[-0.87374883  0.49328425  0.11089463 -2.28350644  1.22698763  0.25863021]] 0.439204104339156
800 [[-0.83088352  0.50345285  0.11188585 -2.37902498  1.26585167  0.25719395]] 0.4380633806867191
900 [[-0.79328035  0.51113434  0.112413   -2.4568105   1.29635018  0.25579904]] 0.437296936819083
```

```
1000 [[-0.76087459  0.5170925   0.11266803 -2.52084952  1.32079036  0.25453119]] 0.43677272788022076
-------- α = 0.0008 --------
100 [[ 0.47821681 -0.12569267  0.63889132  1.49686098 -0.23914824 -0.22404915]] 1.2408449908587253
200 [[ 0.45986575 -0.11337848  0.63015945  1.47090182 -0.24391331 -0.21410675]] 1.2238410247987013
300 [[ 0.44166419 -0.10132266  0.62149441  1.44515604 -0.24844666 -0.20431247]] 1.20719999459169
400 [[ 0.42361525 -0.08952592  0.61289774  1.41962708 -0.25274665 -0.19466888]] 1.1909214597730688
500 [[ 0.40572187 -0.07798864  0.60437101  1.3943182  -0.256812   -0.1851783 ]] 1.1750044616855577
600 [[ 0.38798686 -0.06671096  0.59591582  1.36923241 -0.26064177 -0.17584289]] 1.1594475280391052
700 [[ 0.37041289 -0.05569271  0.58753381  1.34437252 -0.26423535 -0.16666455]] 1.144248681245982
800 [[ 0.35300244 -0.04493345  0.5792266   1.31974108 -0.26759248 -0.15764498]] 1.1294054504133193
900 [[ 0.33575783 -0.03443242  0.57099586  1.29534038 -0.27071326 -0.1487856 ]] 1.1149148868151506
1000 [[ 0.3186812  -0.02418862  0.56284322  1.27117251 -0.27359812 -0.14008761]] 1.1007735826104867
-------- α = 8e-09 --------
100 [[ 0.49671397 -0.13826417  0.64768845  1.52302959 -0.23415343 -0.23413686]] 1.258036177780209
200 [[ 0.49671378 -0.13826405  0.64768836  1.52302933 -0.23415348 -0.23413675]] 1.2580360023445842
300 [[ 0.4967136  -0.13826392  0.64768827  1.52302907 -0.23415353 -0.23413665]] 1.2580358269089955
400 [[ 0.49671341 -0.13826379  0.64768818  1.52302881 -0.23415358 -0.23413655]] 1.2580356514734425
500 [[ 0.49671322 -0.13826367  0.6476881   1.52302854 -0.23415363 -0.23413645]] 1.2580354760379266
600 [[ 0.49671304 -0.13826354  0.64768801  1.52302828 -0.23415368 -0.23413635]] 1.2580353006024463
700 [[ 0.49671285 -0.13826341  0.64768792  1.52302802 -0.23415373 -0.23413625]] 1.2580351251670026
800 [[ 0.49671267 -0.13826329  0.64768783  1.52302775 -0.23415378 -0.23413614]] 1.2580349497315948
900 [[ 0.49671248 -0.13826316  0.64768774  1.52302749 -0.23415383 -0.23413604]] 1.2580347742962232
1000 [[ 0.4967123  -0.13826303  0.64768766  1.52302723 -0.23415389 -0.23413594]] 1.2580345988608876
logloss: 0.43677272788022076    best_alpha: 0.111
```

3. 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).

```
def calc_pred_proba(w, X):
    return sigmoid(np.dot(w, X))
```

```
w, err = eval_LR_model(X_train.values, y_train.values, iterations=20000, alpha=best_alpha)
```

```
2000 [[-0.6154564   0.54048347  0.11242931 -2.79640843  1.42188692  0.2486447 ]] 0.43557376569849593
4000 [[-0.58207771  0.54595793  0.1122301  -2.86136329  1.44552723  0.24736286]] 0.4355276828751495
6000 [[-0.58065702  0.54619809  0.11222159 -2.86417265  1.44655338  0.2473097 ]] 0.43552759786151335
8000 [[-0.58059486  0.54620862  0.11222122 -2.86429566  1.44659832  0.24730738]] 0.43552759769861454
10000 [[-0.58059213  0.54620908  0.11222121 -2.86430105  1.44660029  0.24730728]] 0.43552759769830185
12000 [[-0.58059202  0.5462091   0.1122212  -2.86430129  1.44660038  0.24730727]] 0.4355275976983013
14000 [[-0.58059201  0.5462091   0.1122212  -2.8643013   1.44660038  0.24730727]] 0.4355275976983012
16000 [[-0.58059201  0.5462091   0.1122212  -2.8643013   1.44660038  0.24730727]] 0.4355275976983013
18000 [[-0.58059201  0.5462091   0.1122212  -2.8643013   1.44660038  0.24730727]] 0.4355275976983012
20000 [[-0.58059201  0.5462091   0.1122212  -2.8643013   1.44660038  0.24730727]] 0.4355275976983012
```

4. 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).

```
def calc_pred(w, X):
    y_pred_proba = calc_pred_proba(w, X)
    return (y_pred_proba > 0.5).astype('int')

y_pred = calc_pred(w, X_train.values)
y_pred.shape
```

```
(1, 599)
```

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

```
def calc_accuracy(target, prediction):
    is_equal = (target == prediction).astype('int')
#     print(target.T)
#     print(prediction.T)
#     print(is_equal.T)
    return float(sum(is_equal) / len(is_equal))
```

```
def make_confusion_matrix(prediction, target):
    if len(target) == len(prediction):

        models_ind = ['a(x) = 1', 'a(x) = 0']
        true_cols = ['y = 1', 'y = 0']
        df = pd.DataFrame(np.zeros((2, 2)), index=models_ind, columns=true_cols)
```

```
        true_positive = np.equal(prediction, 1) & np.equal(target, 1)
        true_negative = np.equal(prediction, 0) & np.equal(target, 0)
        false_positive = np.equal(prediction, 1) & np.equal(target, 0)
        false_negative = np.equal(prediction, 0) & np.equal(target, 1)
        df.iloc[[0], 0] = true_positive.sum()
        df.iloc[[1], 1] = true_negative.sum()
        df.iloc[[0], 1] = false_positive.sum()
        df.iloc[[1], 0] = false_negative.sum()
        return df
    else:
        return 'target and prediction arrays must have the same lengths'
```

```
conf_matrix = make_confusion_matrix(y_pred, y_train.values)
conf_matrix
```

	y = 1	y = 0
a(x) = 1	9.0	17.0
a(x) = 0	81.0	492.0

```
def calc_precision(confusion_matrix):
    if isinstance(confusion_matrix, pd.DataFrame):
        df = confusion_matrix.copy()
        TP, FP = float(df.iloc[[0], 0]), float(df.iloc[[0], 1])
        return TP / (FP + TP)
    else:
        return 'unexpected type of matrix'
```

```
def calc_recall(confusion_matrix):
    if isinstance(confusion_matrix, pd.DataFrame):
        df = confusion_matrix.copy()
        TP, FN = float(df.iloc[[0], 0]), float(df.iloc[[1], 0])
        return TP / (TP + FN)
    else:
        return 'unexpected type of matrix'
```

```
def calc_F_score(precision, recall):
    return (2 * precision * recall) / (precision + recall)
```

```
def calc_F_beta_score(precision, recall, beta=10):
    return (1 + beta**2) * (precision * recall) / (beta**2 * (precision + recall))
```

```
accuracy = calc_accuracy(y_train.values.T, y_pred.T)
precision = calc_precision(conf_matrix)
recall = calc_recall(conf_matrix)
fscore = calc_F_score(precision, recall)
f_beta_score = calc_F_beta_score(precision, recall)
print(f'Accuracy:\t{accuracy}\nPrecision:\t{precision}\nRecall: \t{recall}\nF-score:\t{fscore}\nF-beta-score:\t{f_beta
```

```
Accuracy:       0.8363939899833055
Precision:      0.34615384615384615
Recall:         0.1
F-score:        0.15517241379310348
F-beta-score:   0.07836206896551724
```

```
delim = '-' * 8
best_alpha = 0
err = np.inf
alphas = [1e-6, 1e-4, 1e-2, 0.1, 0.111, 0.9, 8e-4, 8e-9]
for a in alphas:
    print(delim + f' α = {a} ' + delim)
    w, errors = eval_LR_model(X_test.values, y_test.values, iterations=1000, alpha=a)

    if errors[-1] < err:
        err = errors[-1]
        best_alpha = a
print(f'logloss: {err}\tbest_alpha: {best_alpha}')
```

```
700 [[-0.50487699  0.33391684  0.04177043  0.15307002 -0.07414147 -0.05898976]] 0.6524667252553048
800 [[-0.57021322  0.34236347  0.02096381  0.04628997 -0.01201268 -0.05385006]] 0.6323429908503079
900 [[-0.6258065   0.34806819  0.00635799 -0.04955538  0.04755228 -0.04957504]] 0.6162100692004775
```

```
1000 [[-0.67339242  0.35218554 -0.00383757 -0.13630478  0.10380071 -0.04583103]] 0.6030891574782821
-------- α = 0.1 --------
100 [[-0.67655123  0.35297736 -0.0050452  -0.13950578  0.10530055 -0.04573701]] 0.6036631383091171
200 [[-0.90868051  0.37881875 -0.02268796 -0.70986962  0.48807292 -0.01867987]] 0.5470514989987432
300 [[-0.95674286  0.40173728 -0.01719317 -1.01170731  0.67049872 -0.00211493]] 0.5337455863679628
400 [[-0.94833638  0.41668063 -0.01353967 -1.19425235  0.76713494  0.0080915 ]] 0.5293124230273424
500 [[-0.92212147  0.4251374  -0.01169171 -1.31305707  0.82252378  0.01487417]] 0.527475575496857
600 [[-0.89251978  0.42962417 -0.01084945 -1.39417064  0.85612501  0.01966606]] 0.5265988067101154
700 [[-0.86507237  0.43189689 -0.01052217 -1.4514248   0.87748175  0.02317378]] 0.5261422761977195
800 [[-0.84162446  0.43299332 -0.01044454 -1.49280961  0.89161676  0.02579072]] 0.5258923095840278
900 [[-0.8224348   0.43348739 -0.01047836 -1.52323912  0.90130478  0.02776307]] 0.5257515510127416
1000 [[-0.80711867  0.43368441 -0.01055405 -1.54588916  0.90814106  0.02925801]] 0.5256710425256473
-------- α = 0.111 --------
100 [[-0.72147209  0.35638586 -0.01254625 -0.22634178  0.16345454 -0.0420359 ]] 0.5918110484681557
200 [[-0.92816321  0.3844372  -0.02153594 -0.79172499  0.53988092 -0.0142664 ]] 0.5425737979677618
300 [[-0.95761445  0.40757636 -0.01572683 -1.08166954  0.70902176  0.00178116]] 0.5317884340915954
400 [[-0.93787967  0.42105918 -0.0125541  -1.25257741  0.79519571  0.01139495]] 0.5283238304491595
500 [[-0.90579061  0.42797749 -0.0111394  -1.3613703   0.84300521  0.01770557]] 0.5269207284308571
600 [[-0.8739916   0.43130371 -0.0105941  -1.43413391  0.87125878  0.02210071]] 0.526267029548433
700 [[-0.84661719  0.4328141  -0.01044822 -1.48446338  0.88886127  0.0252565 ]] 0.5259375882197976
800 [[-0.82450253  0.43345033 -0.01047085 -1.52009447  0.90033416  0.02755703]] 0.5257645188866581
900 [[-0.80723545  0.43368446 -0.01055314 -1.54572876  0.90809559  0.02924718]] 0.525671611432171
1000 [[-0.79401736  0.43374401 -0.01064664 -1.56437574  0.91350149  0.03049417]] 0.5256211595539052
-------- α = 0.9 --------
100 [[-0.82096563  0.4336318  -0.01046119 -1.52639692  0.90249087  0.02795143]] 0.5257480895537607
200 [[-0.76051665  0.43359037 -0.0109693  -1.60918267  0.92584692  0.03353707]] 0.5255613185944493
300 [[-0.75507631  0.43354617 -0.01102906 -1.61630173  0.9277623   0.03402332]] 0.5255599131598944
400 [[-0.75459626  0.43354253 -0.01103434 -1.61693177  0.92793224  0.0340663 ]] 0.525559902180096
500 [[-0.75455377  0.43354221 -0.0110348  -1.61698756  0.92794729  0.03407011]] 0.5255599020940546
600 [[-0.75455001  0.43354219 -0.01103484 -1.61699249  0.92794862  0.03407044]] 0.5255599020933802
700 [[-0.75454968  0.43354218 -0.01103485 -1.61699293  0.92794874  0.03407047]] 0.5255599020933749
800 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933749
900 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933748
1000 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933748
-------- α = 0.0008 --------
100 [[ 0.47748782 -0.12389727  0.63453745  1.49801242 -0.24078367 -0.22909923]] 1.2298048180324337
200 [[ 0.45842995 -0.10980955  0.62148341  1.47321664 -0.24713032 -0.22414758]] 1.212155195416183
300 [[ 0.43954485 -0.09600281  0.6085305   1.44864684 -0.2531908  -0.21928349]] 1.1949129332139166
400 [[ 0.42083668 -0.08247846  0.59568296  1.42430715 -0.25896297 -0.21450828]] 1.178078296214846
500 [[ 0.40230943 -0.0692376   0.58294509  1.40020146 -0.26444511 -0.20982321]] 1.1616509515887836
600 [[ 0.38396689 -0.05628107  0.57032127  1.37633343 -0.26963589 -0.20522938]] 1.145629965458465
700 [[ 0.36581265 -0.04360939  0.55781592  1.35270643 -0.27453443 -0.20072776]] 1.1300138032755727
800 [[ 0.3478501  -0.03122278  0.5454335   1.32932358 -0.27914027 -0.19631921]] 1.114800334052001
900 [[ 0.3300824  -0.01912112  0.53317843  1.30618772 -0.28345337 -0.19200441]] 1.099986838458014
1000 [[ 0.31251247 -0.00730402  0.52105515  1.28330138 -0.28747415 -0.18778394]] 1.085570020757134
-------- α = 8e-09 --------
100 [[ 0.49671396 -0.13826416  0.64768841  1.52302961 -0.23415344 -0.23413691]] 1.2476781958420415
200 [[ 0.49671377 -0.13826401  0.64768827  1.52302935 -0.23415351 -0.23413686]] 1.2476780133032457
300 [[ 0.49671357 -0.13826387  0.64768814  1.5230291  -0.23415358 -0.2341368 ]] 1.2476778307644907
400 [[ 0.49671338 -0.13826372  0.64768801  1.52302885 -0.23415365 -0.23413675]] 1.247677648225776
500 [[ 0.49671319 -0.13826358  0.64768788  1.5230286  -0.23415371 -0.2341367 ]] 1.247677465687102
600 [[ 0.49671299 -0.13826343  0.64768775  1.52302835 -0.23415378 -0.23413665]] 1.2476772831484686
700 [[ 0.4967128  -0.13826329  0.64768761  1.5230281  -0.23415385 -0.2341366 ]] 1.2476771006098752
800 [[ 0.49671261 -0.13826314  0.64768748  1.52302785 -0.23415392 -0.23413655]] 1.2476769180713227

900 [[ 0.49671242 -0.138263    0.64768735  1.5230276  -0.23415398 -0.2341365 ]] 1.2476767355328104
1000 [[ 0.49671222 -0.13826285  0.64768722  1.52302734 -0.23415405 -0.23413645]] 1.2476765529943392
logloss: 0.5255599020933748     best alpha: 0.9
```

```
w, errors = eval_LR_model(X_test.values, y_test.values, iterations=2000, alpha=best_alpha)
```

```
200 [[-0.76051665  0.43359037 -0.0109693  -1.60918267  0.92584692  0.03353707]] 0.5255613185944493
400 [[-0.75459626  0.43354253 -0.01103434 -1.61693177  0.92793224  0.0340663 ]] 0.525559902180096
600 [[-0.75455001  0.43354219 -0.01103484 -1.61699249  0.92794862  0.03407044]] 0.5255599020933802
800 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933749
1000 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933748
1200 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933748
1400 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933749
1600 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933749
1800 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933749
2000 [[-0.75454965  0.43354218 -0.01103485 -1.61699297  0.92794875  0.03407048]] 0.5255599020933749
```

```
y_test_pred = calc_pred(w, X_test.values)
```

```
conf_matrix = make_confusion_matrix(y_test_pred, y_test.values)
conf_matrix
```

	y = 1	y = 0
a(x) = 1	10.0	14.0
a(x) = 0	68.0	308.0

```
accuracy = calc_accuracy(y_test.values.T, y_test_pred.T)
precision = calc_precision(conf_matrix)
recall = calc_recall(conf_matrix)
fscore = calc_F_score(precision, recall)
f_beta_score = calc_F_beta_score(precision, recall)
print(f'Accuracy:\t{accuracy}\nPrecision:\t{precision}\nRecall: \t{recall}\nF-score:\t{fscore}\nF-beta-score:\t{f_beta
```

```
Accuracy:       0.795
Precision:      0.4166666666666667
Recall:         0.1282051282051282
F-score:        0.196078431372549
F-beta-score:   0.09901960784313722
```

6. Могла ли модель переобучиться? Почему?

Модель может переобучиться из-за её сложности и избыточности количества признаков. Чтобы узнать, не переобучилась ли модель, которую я строила для датасета риска сердечной недостаточности, я разделила датасет на train и test. Точность(accuracy) на train получилась 0.836, что немного превышает точность предсказаний на test'е - 0.795. При этом точность(precision) на test'е немного выше, но всё ещё меньше 50%, полнота (recall) так же определила низкое значение. На мой взгляд проблема в несбалансированности выборки конкретно в данной ситуации.

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

L1-regularization

$$\sum_{i=1}^{n} L_i(\vec{x}_i, y_i, \vec{w}) + \lambda \sum_{j=1}^{m} |w_j| \rightarrow \min_w$$

L2-regularization

$$\sum_{i=1}^{n} L_i(\vec{x}_i, y_i, \vec{w}) + \lambda \sum_{j=1}^{m} w_j^2 \rightarrow \min_w$$

```
def eval_LR_model_l1(X, y, iterations, alpha=1e-4, lambda_=1e-8):
    np.random.seed(42)
    w = np.random.randn(X.shape[0])
    n = X.shape[1]
    m = X.shape[0]
    errors = []
    for i in range(1, iterations + 1):
        z = np.dot(w, X)
        y_pred = sigmoid(z)
        y_pred = np.clip(y_pred, 0.00001, 0.99999)
        err = calc_logloss(y, y_pred) + lambda_ / m * np.linalg.norm(w, ord=1)
        w = w - alpha * (1/n * np.dot((y_pred - y), X.T) + lambda_ / m * sum(np.sign(w)))
        if i % (iterations / 10) == 0:
            errors.append(err)
            print(i, w, err)
    return w, errors
```

```
delim = '-' * 8
best_alpha = 0
best_lambda = 0
err = np.inf
alphas = [1e-8, 1e-6, 1e-3, 1e-1, 1, 0.999]
lambdas = [1e-6, 1e-4, 1e-2, 0.1, 0.111, 0.9, 8e-4, 8e-9]
for a in alphas:
    for l in lambdas:
        print(delim + f' α = {a}, λ = {l} ' + delim)
        w, errors = eval_LR_model_l1(X_train.values, y_train.values, 2000, alpha=a, lambda_=l)
        if errors[-1] < err:
            err = errors[-1]
            best_alpha = a
            best_lambda = l
print(f'logloss: {err}\tbest_alpha: {best_alpha} \tbest_lambda: {best_lambda}')
```

```
-------- α = 1e-08, λ = 1e-06 --------
200 [[ 0.49671369 -0.13826398  0.64768832  1.5230292  -0.2341535  -0.2341367 ]] 1.2580361689032797
400 [[ 0.49671322 -0.13826367  0.6476881   1.52302854 -0.23415363 -0.23413645]] 1.2580357303140137
600 [[ 0.49671276 -0.13826335  0.64768788  1.52302789 -0.23415376 -0.2341362 ]] 1.258035291724974
800 [[ 0.4967123  -0.13826303  0.64768766  1.52302723 -0.23415389 -0.23413594]] 1.2580348531361607
1000 [[ 0.49671183 -0.13826271  0.64768743  1.52302657 -0.23415401 -0.23413569]] 1.2580344145475737
1200 [[ 0.49671137 -0.1382624   0.64768721  1.52302592 -0.23415414 -0.23413543]] 1.2580339759592125
1400 [[ 0.4967109  -0.13826208  0.64768699  1.52302526 -0.23415427 -0.23413518]] 1.2580335373710783
1600 [[ 0.49671044 -0.13826176  0.64768677  1.5230246  -0.2341544  -0.23413493]] 1.2580330987831707
1800 [[ 0.49670998 -0.13826144  0.64768655  1.52302395 -0.23415452 -0.23413467]] 1.2580326601954896
2000 [[ 0.49670951 -0.13826113  0.64768633  1.52302329 -0.23415465 -0.23413442]] 1.2580322216080349
-------- α = 1e-08, λ = 0.0001 --------
200 [[ 0.49671369 -0.13826398  0.64768832  1.5230292  -0.2341535  -0.2341367 ]] 1.2580612988559614
400 [[ 0.49671322 -0.13826367  0.6476881   1.52302854 -0.23415363 -0.23413645]] 1.2580608602263974
600 [[ 0.49671276 -0.13826335  0.64768788  1.52302789 -0.23415376 -0.23413619]] 1.2580604215970597
800 [[ 0.4967123  -0.13826303  0.64768766  1.52302723 -0.23415389 -0.23413594]] 1.2580599829679484
1000 [[ 0.49671183 -0.13826271  0.64768743  1.52302657 -0.23415401 -0.23413569]] 1.2580595443390636
1200 [[ 0.49671137 -0.1382624   0.64768721  1.52302592 -0.23415414 -0.23413543]] 1.2580591057104054
1400 [[ 0.4967109  -0.13826208  0.64768699  1.52302526 -0.23415427 -0.23413518]] 1.2580586670819736
1600 [[ 0.49671044 -0.13826176  0.64768677  1.5230246  -0.2341544  -0.23413492]] 1.2580582284537685
1800 [[ 0.49670997 -0.13826144  0.64768655  1.52302395 -0.23415452 -0.23413467]] 1.25805778982579
2000 [[ 0.49670951 -0.13826113  0.64768633  1.52302329 -0.23415465 -0.23413442]] 1.2580573511980377
-------- α = 1e-08, λ = 0.01 --------
200 [[ 0.49671369 -0.13826398  0.64768831  1.5230292  -0.2341535  -0.2341367 ]] 1.2605742941186922
400 [[ 0.49671322 -0.13826366  0.64768809  1.52302854 -0.23415362 -0.23413644]] 1.2605738514538531
600 [[ 0.49671275 -0.13826334  0.64768787  1.52302788 -0.23415375 -0.23413619]] 1.260573408789243
800 [[ 0.49671228 -0.13826302  0.64768764  1.52302722 -0.23415387 -0.23413593]] 1.2605729661248621
1000 [[ 0.49671182 -0.1382627   0.64768742  1.52302656 -0.234154   -0.23413567]] 1.260572523460709
1200 [[ 0.49671135 -0.13826238  0.64768719  1.5230259  -0.23415412 -0.23413541]] 1.2605720807967844
1400 [[ 0.49671088 -0.13826206  0.64768697  1.52302524 -0.23415425 -0.23413516]] 1.2605716381330883
1600 [[ 0.49671041 -0.13826173  0.64768675  1.52302458 -0.23415437 -0.2341349 ]] 1.2605711954696208
1800 [[ 0.49670995 -0.13826141  0.64768652  1.52302392 -0.23415449 -0.23413464]] 1.2605707528063812
2000 [[ 0.49670948 -0.13826109  0.6476863   1.52302326 -0.23415462 -0.23413438]] 1.2605703101433707
-------- α = 1e-08, λ = 0.1 --------
200 [[ 0.49671366 -0.13826395  0.64768828  1.52302917 -0.23415347 -0.23413667]] 1.2834197051035603
400 [[ 0.49671316 -0.1382636   0.64768803  1.52302848 -0.23415356 -0.23413638]] 1.2834192252549084
600 [[ 0.49671266 -0.13826325  0.64768778  1.52302779 -0.23415366 -0.2341361 ]] 1.2834187454065042
800 [[ 0.49671216 -0.1382629   0.64768752  1.5230271  -0.23415375 -0.23413581]] 1.2834182655583477
1000 [[ 0.49671167 -0.13826255  0.64768727  1.52302641 -0.23415385 -0.23413552]] 1.2834177857104392
1200 [[ 0.49671117 -0.1382622   0.64768701  1.52302572 -0.23415394 -0.23413523]] 1.2834173058627778
1400 [[ 0.49671067 -0.13826185  0.64768676  1.52302503 -0.23415404 -0.23413495]] 1.2834168260153644
1600 [[ 0.49671017 -0.13826149  0.64768651  1.52302434 -0.23415413 -0.23413466]] 1.2834163461681989
1800 [[ 0.49670968 -0.13826114  0.64768625  1.52302365 -0.23415422 -0.23413437]] 1.2834158663212807
2000 [[ 0.49670918 -0.13826079  0.647686    1.52302296 -0.23415432 -0.23413408]] 1.28341538647461
-------- α = 1e-08, λ = 0.111 --------
200 [[ 0.49671365 -0.13826395  0.64768828  1.52302916 -0.23415347 -0.23413667]] 1.2862119219406054
400 [[ 0.49671315 -0.13826359  0.64768802  1.52302847 -0.23415356 -0.23413638]] 1.2862114374855431
600 [[ 0.49671265 -0.13826324  0.64768777  1.52302778 -0.23415365 -0.23413608]] 1.2862109530307313
800 [[ 0.49671215 -0.13826288  0.64768751  1.52302708 -0.23415374 -0.23413579]] 1.2862104685761693
1000 [[ 0.49671165 -0.13826253  0.64768725  1.52302639 -0.23415383 -0.2341355 ]] 1.2862099841218573
1200 [[ 0.49671115 -0.13826217  0.64768699  1.52302569 -0.23415392 -0.23413521]] 1.2862094996677957
1400 [[ 0.49671064 -0.13826182  0.64768673  1.523025   -0.23415401 -0.23413492]] 1.2862090152139842
1600 [[ 0.49671014 -0.13826147  0.64768648  1.52302431 -0.2341541  -0.23413463]] 1.2862085307604225
1800 [[ 0.49670964 -0.13826111  0.64768622  1.52302361 -0.23415419 -0.23413434]] 1.2862080463071108
2000 [[ 0.49670914 -0.13826076  0.64768596  1.52302292 -0.23415428 -0.23413405]] 1.2862075618540496
-------- α = 1e-08, λ = 0.9 --------
200 [[ 0.49671339 -0.13826369  0.64768802  1.5230289  -0.2341532  -0.2341364 ]] 1.486489985809078
400 [[ 0.49671263 -0.13826307  0.6476875   1.52302794 -0.23415303 -0.23413585]] 1.4864891358821741
600 [[ 0.49671186 -0.13826245  0.64768698  1.52302699 -0.23415286 -0.2341353 ]] 1.4864882859557444
```

```
w, errors = eval_LR_model_l1(X_train.values, y_train.values, 2000, alpha=best_alpha, lambda_=best_lambda)
y_train_pred = calc_pred(w, X_train.values)
conf_matrix = make_confusion_matrix(y_train_pred, y_train.values)
conf_matrix
```

```
200 [[-0.6273514   0.53859915  0.11249817 -2.77369288  1.41365133  0.24911511]] 0.4356124288576214
400 [[-0.58323803  0.54576223  0.11223703 -2.85907155  1.44469037  0.24740636]] 0.4355278784449196
600 [[-0.58074692  0.54618285  0.11222212 -2.86399465  1.44648834  0.24731305]] 0.43552760246789574
800 [[-0.58060109  0.54620753  0.11222124 -2.86428323  1.44659376  0.2473076 ]] 0.43552760152063996
1000 [[-0.58059253  0.54620898  0.11222119 -2.86430017  1.44659995  0.24730728]] 0.43552760151738074
1200 [[-0.58059203  0.54620907  0.11222119 -2.86430116  1.44660031  0.24730726]] 0.4355276015173697
1400 [[-0.580592    0.54620907  0.11222119 -2.86430122  1.44660033  0.24730726]] 0.43552760151736963
1600 [[-0.580592    0.54620907  0.11222119 -2.86430122  1.44660033  0.24730726]] 0.43552760151736963
1800 [[-0.580592    0.54620907  0.11222119 -2.86430122  1.44660033  0.24730726]] 0.43552760151736963
2000 [[-0.580592    0.54620907  0.11222119 -2.86430122  1.44660033  0.24730726]] 0.43552760151736963
```

	y = 1	y = 0
a(x) = 1	9.0	17.0
a(x) = 0	81.0	492.0

```
accuracy = calc_accuracy(y_train.values.T, y_train_pred.T)
precision = calc_precision(conf_matrix)
recall = calc_recall(conf_matrix)
fscore = calc_F_score(precision, recall)
f_beta_score = calc_F_beta_score(precision, recall)
print(f'Accuracy:\t{accuracy}\nPrecision:\t{precision}\nRecall: \t{recall}\nF-score:\t{fscore}\nF-beta-score:\t{f_beta
```

```
Accuracy:       0.8363939899833055
Precision:      0.34615384615384615
Recall:         0.1
F-score:        0.15517241379310348
F-beta-score:   0.07836206896551724
```

```
def eval_LR_model_l2(X, y, iterations, alpha=1e-4, lambda_=1e-8):
    np.random.seed(42)
    w = np.random.randn(X.shape[0])
    m = X.shape[0]
    errors = []
    n = X.shape[1]
    for i in range(1, iterations + 1):
        z = np.dot(w, X)
        y_pred = sigmoid(z)
        y_pred = np.clip(y_pred, 0.00001, 0.99999)
        err = calc_logloss(y, y_pred) + lambda_ / (2 * m) * np.linalg.norm(w, ord=2)
        w = w - alpha * (1/n * np.dot((y_pred - y), X.T) + lambda_ / m * np.sum(w))
        if i % (iterations / 10) == 0:
            errors.append(err)
            print(i, w, err)
    return w, errors
```

```
delim = '-' * 8
best_alpha = 0
best_lambda = 0
err = np.inf
alphas = [1e-8, 1e-6, 1e-3, 1e-1, 1, 0.999]
lambdas = [1e-6, 1e-4, 1e-2, 0.1, 0.111, 0.9, 8e-4, 8e-9]
for a in alphas:
    for l in lambdas:
        print(delim + f' α = {a}, λ = {l} ' + delim)
        w, errors = eval_LR_model_l2(X_train.values, y_train.values, 2000, alpha=a, lambda_=l)
        if errors[-1] < err:
            err = errors[-1]
            best_alpha = a
            best_lambda = l
print(f'logloss: {err}\tbest_alpha: {best_alpha} \tbest_lambda: {best_lambda}')
```

```
1000 [[-0.56461201  0.56182263  0.12822151 -2.84562114  1.45236594  0.25598073]] 0.43834973760817164
1200 [[-0.56461157  0.56182271  0.12822151 -2.84562202  1.45236627  0.25598072]] 0.43834973823277873
1400 [[-0.56461154  0.56182272  0.12822151 -2.84562207  1.45236628  0.25598071]] 0.43834973826849033
1600 [[-0.56461154  0.56182272  0.12822151 -2.84562207  1.45236629  0.25598071]] 0.43834973827053203
1800 [[-0.56461154  0.56182272  0.12822151 -2.84562207  1.45236629  0.25598071]] 0.43834973827064877
2000 [[-0.56461154  0.56182272  0.12822151 -2.84562207  1.45236629  0.25598071]] 0.43834973827065543
-------- α = 0.999, λ = 0.1 --------
200 [[-0.5272156   0.63360452  0.20585857 -2.68594668  1.45770355  0.30287252]] 0.46457340606416614
400 [[-0.4886802   0.64125819  0.20667245 -2.76107593  1.48647107  0.30192064]] 0.46511121189161675
600 [[-0.48678492  0.64164884  0.20671551 -2.76483151  1.48791693  0.30187745]] 0.46514153668810304
800 [[-0.48668881  0.64166868  0.2067177  -2.76502212  1.48799034  0.30187527]] 0.4651430843969732
1000 [[-0.48668393  0.64166969  0.20671781 -2.7650318   1.48799407  0.30187516]] 0.46514316303147135
1200 [[-0.48668368  0.64166974  0.20671782 -2.76503229  1.48799426  0.30187515]] 0.4651431670257431
1400 [[-0.48668367  0.64166975  0.20671782 -2.76503232  1.48799426  0.30187515]] 0.46514316722863136
1600 [[-0.48668367  0.64166975  0.20671782 -2.76503232  1.48799427  0.30187515]] 0.46514316723893695
1800 [[-0.48668367  0.64166975  0.20671782 -2.76503232  1.48799427  0.30187515]] 0.4651431672394605
2000 [[-0.48668367  0.64166975  0.20671782 -2.76503232  1.48799427  0.30187515]] 0.465143167239487
-------- α = 0.999, λ = 0.111 --------
200 [[-0.52149452  0.63936247  0.21130412 -2.68153743  1.46076879  0.30625491]] 0.46777197911935753
400 [[-0.48324343  0.64703128  0.2121555  -2.7562291   1.48944084  0.30533266]] 0.468375822931013
600 [[-0.48137457  0.64742005  0.21220018 -2.75993745  1.49087214  0.30529122]] 0.46840921707876354
800 [[-0.48128045  0.64743966  0.21220244 -2.76012438  1.49094431  0.30528915]] 0.4684109088118548
1000 [[-0.4812757   0.64744065  0.21220256 -2.76013381  1.49094795  0.30528904]] 0.4684109941733458
1200 [[-0.48127546  0.6474407   0.21220256 -2.76013428  1.49094813  0.30528904]] 0.4684109984796556
1400 [[-0.48127545  0.64744071  0.21220256 -2.76013431  1.49094814  0.30528904]] 0.4684109986968979
1600 [[-0.48127545  0.64744071  0.21220256 -2.76013431  1.49094814  0.30528904]] 0.4684109987078571
1800 [[-0.48127545  0.64744071  0.21220256 -2.76013431  1.49094814  0.30528904]] 0.46841099870841013
2000 [[-0.48127545  0.64744071  0.21220256 -2.76013431  1.49094814  0.30528904]] 0.468410998708438
-------- α = 0.999, λ = 0.9 --------
200 [[-0.43135268  0.73496811  0.29897143 -2.62260238  1.51772982  0.36438398]] 0.6839731826669101
400 [[-0.39701607  0.74269271  0.30012731 -2.69229849  1.54529734  0.36370259]] 0.6891080831183471
```

```
600 [[-0.39547674  0.74305118  0.30018245 -2.69546679  1.54655736  0.36367533]] 0.6893459154955465
800 [[-0.39540576  0.74306773  0.300185   -2.69561296  1.54661551  0.36367408]] 0.6893568978323229
1000 [[-0.39540249  0.7430685   0.30018512 -2.69561971  1.54661819  0.36367402]] 0.6893574048963428
1200 [[-0.39540234  0.74306853  0.30018512 -2.69562002  1.54661832  0.36367402]] 0.6893574283077917
1400 [[-0.39540233  0.74306854  0.30018512 -2.69562004  1.54661832  0.36367402]] 0.6893574293887121
1600 [[-0.39540233  0.74306854  0.30018512 -2.69562004  1.54661832  0.36367402]] 0.6893574294386188
1800 [[-0.39540233  0.74306854  0.30018512 -2.69562004  1.54661832  0.36367402]] 0.689357429440923
2000 [[-0.39540233  0.74306854  0.30018512 -2.69562004  1.54661832  0.36367402]] 0.6893574294410293
-------- α = 0.999, λ = 0.0008 --------
200 [[-0.62600307  0.539907    0.11385662 -2.77201077  1.41408727  0.24984499]] 0.43582922788380046
400 [[-0.58185957  0.54709959  0.11361756 -2.85742654  1.44516589  0.24814583]] 0.4357491876125515
600 [[-0.57936593  0.54752212  0.11360399 -2.86235318  1.44696672  0.24805308]] 0.43574917983278183
800 [[-0.5792199   0.54754693  0.11360319 -2.86264206  1.44707235  0.24804766]] 0.4357491946470403
1000 [[-0.57921133  0.54754838  0.11360315 -2.86265902  1.44707855  0.24804734]] 0.4357491955690074
1200 [[-0.57921083  0.54754847  0.11360314 -2.86266001  1.44707891  0.24804732]] 0.43574919562330316
1400 [[-0.5792108   0.54754847  0.11360314 -2.86266007  1.44707893  0.24804732]] 0.4357491956264907
1600 [[-0.5792108   0.54754848  0.11360314 -2.86266008  1.44707893  0.24804732]] 0.4357491956266778
1800 [[-0.5792108   0.54754848  0.11360314 -2.86266008  1.44707893  0.24804732]] 0.4357491956266888
2000 [[-0.5792108   0.54754848  0.11360314 -2.86266008  1.44707893  0.24804732]] 0.43574919562668957
-------- α = 0.999, λ = 8e-09 --------
200 [[-0.62749049  0.53857727  0.11249898 -2.77342826  1.41355546  0.24912066]] 0.435612926212378
400 [[-0.58325325  0.5457597   0.11223715 -2.85904158  1.44467945  0.24740695]] 0.4355278800193282
600 [[-0.58074826  0.54618266  0.11222215 -2.86399207  1.44648742  0.24731312]] 0.4355276008768039
800 [[-0.58060119  0.54620756  0.11222127 -2.86428308  1.44659374  0.24730762]] 0.43552759991333145
1000 [[-0.58059254  0.54620902  0.11222122 -2.86430021  1.44659999  0.2473073 ]] 0.43552759991000456
1200 [[-0.58059203  0.54620911  0.11222122 -2.86430122  1.44660036  0.24730728]] 0.4355275999099935
1400 [[-0.580592    0.54620911  0.11222122 -2.86430128  1.44660038  0.24730728]] 0.4355275999099935
1600 [[-0.580592    0.54620911  0.11222122 -2.86430128  1.44660039  0.24730728]] 0.4355275999099935
```

```
w, errors = eval_LR_model_l2(X_train.values, y_train.values, 2000, alpha=best_alpha, lambda_=best_lambda)
y_train_pred = calc_pred(w, X_train.values)
conf_matrix = make_confusion_matrix(y_train_pred, y_train.values)
conf_matrix
```

```
200 [[-0.6273514   0.53859919  0.1124982  -2.77369293  1.41365138  0.24911513]] 0.43561242722663895
400 [[-0.58323803  0.54576227  0.11223706 -2.85907161  1.44469042  0.24740638]] 0.43552787683534167
600 [[-0.58074692  0.54618289  0.11222215 -2.86399471  1.44648839  0.24731307]] 0.4355276008603852
800 [[-0.58060109  0.54620757  0.11222127 -2.86428329  1.44659381  0.24730762]] 0.435527599913256
1000 [[-0.58059253  0.54620902  0.11222122 -2.86430022  1.4466      0.2473073 ]] 0.4355275999100042
1200 [[-0.58059203  0.54620911  0.11222122 -2.86430122  1.44660036  0.24730728]] 0.4355275999099935
1400 [[-0.580592    0.54620911  0.11222122 -2.86430128  1.44660038  0.24730728]] 0.4355275999099935
1600 [[-0.580592    0.54620911  0.11222122 -2.86430128  1.44660039  0.24730728]] 0.4355275999099934
1800 [[-0.580592    0.54620911  0.11222122 -2.86430128  1.44660039  0.24730728]] 0.4355275999099935
2000 [[-0.580592    0.54620911  0.11222122 -2.86430128  1.44660039  0.24730728]] 0.4355275999099935
```

	y = 1	**y = 0**
a(x) = 1	9.0	17.0
a(x) = 0	81.0	492.0

```
accuracy = calc_accuracy(y_train.values.T, y_train_pred.T)
precision = calc_precision(conf_matrix)
recall = calc_recall(conf_matrix)
fscore = calc_F_score(precision, recall)
f_beta_score = calc_F_beta_score(precision, recall)
print(f'Accuracy:\t{accuracy}\nPrecision:\t{precision}\nRecall: \t{recall}\nF-score:\t{fscore}\nF-beta-score:\t{f_beta
```

```
Accuracy:       0.8363939899833055
Precision:      0.34615384615384615
Recall:         0.1
F-score:        0.15517241379310348
F-beta-score:   0.07836206896551724
```