Machine Learning Course

Assignment 1: Linear and Logistic Regression

Submitted to: Dr-Abeer Korany

Name	ID
Rana Muhammad Ali	20200182
Hady Abdallah Hafez	20200617
Norhan Hassan Ali	20201202
Mohammed Essam Eldin Mohammed	20200464

assign1

November 16, 2023

[20]:
```python
import pandas
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plot
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score
import warnings

# Suppress SettingWithCopyWarning
warnings.filterwarnings("ignore", category=pandas.core.common.
 ↪SettingWithCopyWarning)
```

[21]:
```python
#Load the "loan_old.csv" dataset.
data = pandas.read_csv('loan_old.csv')

#check whether there are missing values
missing_values = data.isnull().sum()
print('Missing values:\n',missing_values)

print('----------------------')
```

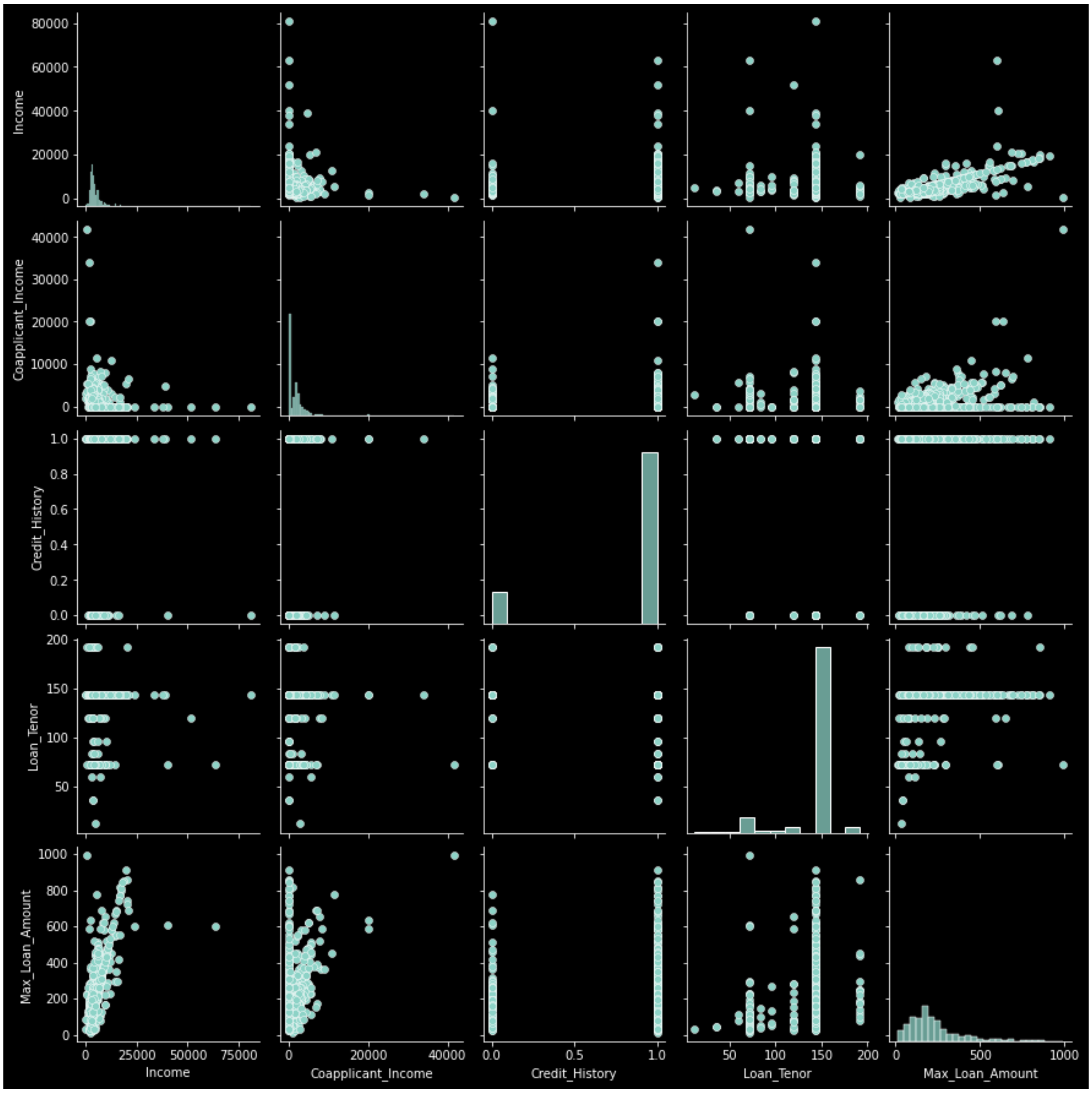
```
Missing values:
 Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor           15
Credit_History       50
Property_Area         0
Max_Loan_Amount      25
Loan_Status           0
dtype: int64
----------------------
```

[22]:

```
#check the type of each feature (categorical or numerical)
data_types = data.dtypes

print('Data types:\n',data_types)

print('----------------------')


#visualize a pairplot between numercial columns
sns.
↪pairplot(data[['Income','Coapplicant_Income','Credit_History','Loan_Tenor','Max_Loan_Amount
#plot.show()
```

```
Data types:
 Loan_ID                 object
Gender                 object
Married                object
Dependents             object
Education              object
Income                  int64
Coapplicant_Income    float64
Loan_Tenor            float64
Credit_History        float64
Property_Area          object
Max_Loan_Amount       float64
Loan_Status            object
dtype: object
----------------------
```

[22]: <seaborn.axisgrid.PairGrid at 0x210b9519288>



[23]:
```
#records containing missing values are removed
data.drop(columns=['Loan_ID'], inplace=True)
if data.isnull().values.any():
    data_cleaned_rows = data.dropna()


data_types = data_cleaned_rows.dtypes

print('-----------------------')

#check whether numerical features have the same scale
print('Numerical features describe: \n')
```

```
numerical_column_name=['Income','Coapplicant_Income','Loan_Tenor']
describe_data=data_cleaned_rows[numerical_column_name].describe()
print(describe_data)


print('----------------------')
```

```
----------------------
Numerical features describe:

             Income  Coapplicant_Income  Loan_Tenor
count    513.000000          513.000000  513.000000
mean    5030.730994         1486.627524  137.660819
std     4469.976643         2102.196620   23.139902
min      150.000000            0.000000   36.000000
25%     2889.000000            0.000000  144.000000
50%     3800.000000         1126.000000  144.000000
75%     5703.000000         2250.000000  144.000000
max    63337.000000        20000.000000  192.000000
----------------------
```

[24]:
```
label_encoder = LabelEncoder()


#categorical features and targets are encoded
for column_name in data_cleaned_rows.columns:
    if data_cleaned_rows[column_name].dtype == 'object':
        data_cleaned_rows.loc[:, column_name] = label_encoder.
 ↪fit_transform(data_cleaned_rows[column_name])
```

[25]:
```
#the features and targets are separated
x=data_cleaned_rows.drop(columns=['Max_Loan_Amount','Loan_Status'])
y=data_cleaned_rows[['Max_Loan_Amount','Loan_Status']]


#the data is shuffled and split into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=100)
y_train_max_loan = y_train['Max_Loan_Amount']
y_test_max_loan = y_test['Max_Loan_Amount']
y_train_loan_status = y_train['Loan_Status']
y_test_loan_status = y_test['Loan_Status']
```

[26]:
```
#numerical features are standardized

x_train_mean=x_train[numerical_column_name].mean()
x_train_std=x_train[numerical_column_name].std()
```

```
x_test[numerical_column_name] = (x_test[numerical_column_name] - x_train_mean) /
 ↪ x_train_std
x_train[numerical_column_name] = (x_train[numerical_column_name] -␣
 ↪x_train_mean) / x_train_std


#Convert data to Numpy array
x_train = x_train.to_numpy().reshape((-1,9))
x_test = x_test.to_numpy().reshape((-1,9))
y_train_max_loan = y_train_max_loan.to_numpy()
y_test_max_loan = y_test_max_loan.to_numpy()
y_train_loan_status = y_train_loan_status.to_numpy()
y_test_loan_status = y_test_loan_status.to_numpy()
```

[27]:
```
#Fit a linear regression model
print("linear regression model: ")
model = linear_model.LinearRegression()
model.fit(x_train,y_train_max_loan)

print('Coefficients: \n', model.coef_, " ", model.intercept_)


#predict the loan amount
y_pred = model.predict(x_test)

r2 = r2_score(y_test_max_loan, y_pred)
print("R-squared score:", r2)

print('----------------------')
```

```
linear regression model:
Coefficients:
 [ 10.38137211   4.13996897   6.55078486 -17.41010274 119.21011348
  66.7485454   50.18745967   8.13355175 -11.4683694 ]   216.24700694689832
R-squared score: 0.7780935494194898
----------------------
```

[28]:
```
print('logistic regression model:')
#logistic regression model
'''
Logistic regression Algorithm:  (z)
1. Define the Sigmoid Function
2. Initialize Parameters (  and B)
3. Compute the Linear Combination: z =  1x1+ 2x2+…+ nx n+b
4. Apply the Sigmoid Function: y =  (z)
5. Define the Cost Function: J( )=- 1/m [y(i)log(y)+(1-y)log(1- y)]
6. Gradient Descent
```

```
     j= j- j/ j
    b = b + - j/ b
'''
def sigmoid(z):
    z = np.array(z,dtype=float)
    return 1 / (1 + np.exp(-z))
def initialize_parameters(dim):
    # Initialize weights and bias to zero
    theta = np.zeros((1, dim))
    b = 0
    return theta, b
def linear_combination(X, w, b):
    return np.dot(X, w.T) + b
def compute_cost(y, y_hat):
    m = len(y)
    return -1/m * np.sum(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
def predict(X, w, b):
    z = linear_combination(X, w, b)
    return sigmoid(z)
def gradient_descent(X, y, w, b, learning_rate, num_iterations):
    m = len(y)
    for i in range(num_iterations):
        # Compute linear combination
        z = linear_combination(X, w, b)
        # Apply sigmoid function and reshape
        y_hat = sigmoid(z).reshape(-1)
        # Compute cost
        cost = compute_cost(y, y_hat)
        # Compute gradients
        dw = 1/m * np.dot(X.T, (y_hat - y))
        db = 1/m * np.sum(y_hat - y)
        # Update parameters
        w -= (learning_rate * dw.T).astype(float) # Transpose dw before␣
↪updating weights
        b -= learning_rate * db
        # Print cost every 100 iterations
        if i % 200 == 0:
            print(f"Cost after iteration {i}: {cost}")
    return w, b
```

logistic regression model:

[29]:
```
w, b = initialize_parameters(x_train.shape[1])
# Set hyperparameters
learning_rate = 0.01
num_iterations = 2000
# Train the logistic regression modely_train_loan_status
```

```
w, b = gradient_descent(x_train,y_train_loan_status , w, b, learning_rate,␣
 ↪num_iterations)
# Print the trained parameters
print("Trained weights:", w)
print("Trained bias:", b)
```

```
Cost after iteration 0: 0.6931471805599453
Cost after iteration 200: 0.5999269197993258
Cost after iteration 400: 0.5881416530028638
Cost after iteration 600: 0.5788959205066386
Cost after iteration 800: 0.5708836171960125
Cost after iteration 1000: 0.5638389399585394
Cost after iteration 1200: 0.5575884650133229
Cost after iteration 1400: 0.5520044793455577
Cost after iteration 1600: 0.5469881804764495
Cost after iteration 1800: 0.5424606059968827
Trained weights: [[-0.0847963   0.23688659  0.03649964 -0.22506948 -0.01431119
-0.00553734
   0.00190672  1.1787928  -0.00819126]]
Trained bias: -0.19801784014892673
```

[30]:
```
def Accuracy(X, y, w, b):
    predictions = predict(X, w, b)
    predictions_as_binary = ((predictions >= 0.5).astype(int)).reshape(-1)
    correct_predictions = (predictions_as_binary == y).sum()
    accuracy = (correct_predictions / len(y))*100
    return accuracy


accuracy = Accuracy(x_test, y_test_loan_status, w, b)
print("Accuracy: ",format(accuracy, ".2f"),'%')
```

```
Accuracy:  80.58 %
```

[31]:
```
#load_new analysis and preprocessing part
# Load the "loan_new.csv" dataset.
print("-----------------------Loan_new.csv Part----------------------")
data = pandas.read_csv('loan_new.csv')

# check whether there are missing values
missing_values = data.isnull().sum()
print('Missing values:\n', missing_values)
```

```
-----------------------Loan_new.csv Part----------------------
Missing values:
 Loan_ID               0
Gender               11
Married               0
```

```
Dependents           10
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor            7
Credit_History       29
Property_Area         0
dtype: int64
```

[32]:
```
# records containing missing values are removed and drop Loan_ID column
data.drop(columns=['Loan_ID'], inplace=True)
if data.isnull().values.any():
    newdata_cleaned_rows = data.dropna()



label_encoder = LabelEncoder()

for column_name in newdata_cleaned_rows.columns:
    if newdata_cleaned_rows[column_name].dtype == 'object':
        newdata_cleaned_rows.loc[:, column_name] = label_encoder.
 ↪fit_transform(newdata_cleaned_rows[column_name])
```

[33]:
```
# numerical values are standardized
newdata_cleaned_rows.loc[:,numerical_column_name] =␣
 ↪(newdata_cleaned_rows[numerical_column_name]-x_train_mean)/x_train_std
x_new = newdata_cleaned_rows.to_numpy()


# use models to predict loan_Amount and status
loan_amount_prediction = model.predict(x_new)
loan_amount_prediction =[0 if i <0 else i for i in loan_amount_prediction]
status_prediction = predict(x_new, w, b)
status_prediction_YorN = ['Y' if prob >= 0.5 else 'N' for prob in␣
 ↪status_prediction]
```

[34]:
```
print("-------------------------------------------")
print("prediction of loan Amount:\n",loan_amount_prediction)
print("----------------------------------------")
print("prediction of loan status:\n",status_prediction_YorN)
```

```
------------------------------------------
prediction of loan Amount:
 [202.274846871364, 187.8160057358955, 251.93609454316305, 119.17516490024413,
199.31157001265473, 100.36978388189598, 166.45618941328115, 311.708073953031,
181.50522463462966, 114.51613139053241, 177.33161468764953, 381.2233598487386,
172.677976167094, 203.35091261713922, 275.3435034464072, 194.80782724205443,
531.0936736774287, 49.30081487373744, 144.13349485088582, 0, 121.10875163093128,
```

327.3001676669592, 798.1142216249362, 361.04424754853875, 46.126160100462414,
100.16535818474699, 253.63922288672055, 198.98930940069965, 210.59081710388404,
174.0491739121638, 136.74690571681734, 224.42191755894504, 200.14819073250965,
200.98066593709723, 205.47259192886344, 224.8376419240589, 135.47159461909257,
157.85103757768002, 308.2464871722648, 136.02564184727453, 165.26564013136235,
281.00103765283296, 177.5597360456494, 257.87876141184273, 50.94226948995478,
174.11119110848907, 115.20495034741424, 160.08767608820767, 126.88939616365552,
245.5926958631068, 53.78802039235052, 165.3918652432836, 243.78554470887343,
193.38510328478466, 156.99270105677178, 173.54082184882012, 252.02029120093283,
162.5119175654929, 148.31614433736726, 262.75121200300663, 290.04090560623456,
177.947003402772, 41.37115111486048, 243.00195056727927, 280.7723961512929,
248.68756477486968, 210.45438616556524, 237.0886183937945, 285.3430461945098,
248.05371652317604, 199.68489556976598, 1904.4427136042827, 275.82715748926364,
293.93862593575886, 19.353547335735527, 299.64111325229186, 206.40162758908426,
142.76500441321917, 203.93736555918153, 194.0184598171582, 441.24849867083157,
262.10584011838756, 235.42209761416075, 255.06814523987984, 236.613539784267,
276.32648376913363, 256.58449819831867, 322.6378771448608, 189.13834919113987,
211.09119874828292, 163.0159995430089, 0, 184.46804783548953,
214.46689458999901, 183.01877907464285, 195.7386376468596, 174.5919575008134,
129.75483458285174, 249.1845037271348, 330.5488875230321, 91.23624509251935,
160.94694632697608, 211.01017367665492, 132.7824889408846, 255.4658190882771,
209.6410529258884, 326.731997442748, 350.36723617777824, 183.08453224591355,
209.64481878751664, 260.20387413691367, 0, 180.74906068504455,
162.4181624138202, 217.9933999825785, 141.41796431739982, 26.13213696600579,
165.45589016218275, 207.58871818974944, 227.30531218579983, 187.93038779395388,
166.77904150687374, 239.4741508608565, 85.40434720103124, 318.1323879447666,
130.64353430270296, 270.65172956221784, 217.44978296632897, 202.6679915865604,
262.3494436146782, 172.54940035946015, 202.9391619658535, 172.96526637209791,
216.5290257354125, 112.10679456911501, 307.2679164306655, 251.31326937012025,
295.5904269897159, 262.5766066283791, 310.84930110260495, 132.06268824218824,
194.68467470775704, 134.8361422613607, 97.12454591654331, 131.66099090664233,
231.0599578558914, 204.07886707610794, 167.0843700568738, 166.3160984080615,
189.7504880429924, 212.74552304218815, 53.94943763088119, 172.13769363344088,
358.15867855580734, 218.14554585580225, 223.40424815785195, 262.4426916162374,
268.32064689862324, 181.07585636656657, 297.08073641891684, 215.21437881649697,
313.65391305661274, 393.2310166196461, 413.0947993873217, 57.76451780075473,
156.9087273317161, 252.51824568198316, 202.81451661261607, 409.8157248442619,
200.98066593709723, 188.89046579704535, 162.73236250689456, 148.4270191138757,
172.82331488801776, 403.35996194157605, 148.8334480551637, 210.99855613376636,
138.95390083539343, 208.93601768394797, 280.1539350257818, 186.56366313310048,
149.8012706570708, 103.18575159503624, 276.11471413340905, 224.9337586305613,
220.7694091273448, 127.44303375364731, 0, 344.67915480932265,
271.68071935854266, 224.13015330382055, 217.76671620149205, 229.79585393627892,
149.19545202212376, 190.69128795747443, 107.79441717491324, 176.23705809640666,
289.97682435744855, 221.4724083083566, 182.78420589233883, 881.8572739003761,
4.7094336213171175, 253.11851409095397, 156.4905262385591, 186.4211259490995,
262.6381045007967, 657.0018185206989, 162.39652291811785, 266.1537755831118,
133.47965697330497, 205.54077088937981, 183.24503484829526, 188.74306048036692,

```
90.82210720040618, 263.0568281920921, 151.58120787251104, 0, 267.61487608645996,
163.16139140614177, 203.45596769595963, 196.21060710800634, 140.86155736340686,
167.30807176331996, 272.4495365114593, 270.0698190255933, 221.82387189959275,
191.88552348812217, 562.175990248377, 203.71751133557117, 270.91621370705616,
261.46230061495805, 255.8137802999856, 165.3646863091514, 165.5189922947127,
269.5618934746184, 696.588530888707, 230.74567208963836, 135.17992132399135,
204.6640499265424, 221.9183395069053, 46.032303937859496, 182.96807706404832,
182.60038985602904, 207.07913093578202, 286.2649188921341, 667.2482032464842,
285.1330454272906, 258.7636166657425, 179.57429575734068, 396.16940472264224,
222.80957263142696, 237.13591015069, 141.98598388057502, 156.91394539344952,
170.42920388613595, 229.39907232435678, 155.50977304119766, 148.91203697894144,
198.01881152115243, 253.65308928058982, 215.2987796302969, 156.43840844818857,
283.14061294183136, 174.03569393455933, 259.59065176628906, 305.88031424251517,
187.91018814498705, 297.9292088527792, 220.18070064842976, 87.09469379290019,
129.5147510534013, 178.64085597613223, 163.60675609372313, 214.91757315415785,
201.7286526526045, 125.44660391521496, 112.0258743210312, 626.9677823430493,
215.4653912532631, 143.08756007169976, 219.54566712165482, 318.91424394583885,
219.89776907871857, 322.1341978465136, 210.72775036739023, 177.90085562846903,
167.15002270402704, 127.62568408844591, 163.2832944380163, 146.13524983673437,
238.51361438499254, 128.11249865390351, 306.94386119247775, 17.675897080649406,
183.13798751198004, 249.73096878230956, 287.75808573431635, 218.16054234957403,
108.90010721396098, 181.92346167081803, 101.63850750862862, 312.5303000003912,
232.36760299983817, 324.59527173038373, 48.93591388760581, 306.2050279156571,
219.07019057239046, 117.04251086753422, 247.32233759601684, 199.22738906708162,
215.42062350539172, 184.48011705047585, 279.78928290815134, 154.28822908568486]
----------------------------------------
prediction of loan status:
 ['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N',
'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y',
'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y']
```