Assigment

The following notebook contains the base architecture for the assignment.

The task is to complete the missing parts, explore the datasets and build two simple binary classifi connected layers and one which also incorporates convolutional and max-pooling layers.

Only numpy is allowed to implement the classes! (Matplotlib and other modules can be used for vis

Due date: 2019 december 3

```
import numpy as np
#For Data Visualization
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from tabulate import tabulate
```

Optimizers

```
class Optimizer:
  def update(self, param, grad):
    pass

  def __call__(self, param, grad):
    self.update(param, grad)


class SGD(Optimizer):
  def __init__(self, learning_rate):
    self.learning_rate = learning_rate

  def update(self, param, grad):
    '''Gradient Descent Update
    This function updates the given 'param' using the 'grad' (gradients).
    Note #1: Use the learning_rate.
    Note #2: There are no return values.

    :param param: Parameters of the layer.
    :param grad: Corresponding gradients.

    '''
    param = param + (self.learning_rate * -1.0 ) * grad  # DONE //
```

Weight Initializers

```
class WeigthInitializer:
  def initialize(self, size):
    return np.ones(size, dtype=np.float)

  def __call__(self, size):
    return self.initialize(size)


class RandomInitializer(WeigthInitializer):
  def __init__(self, shift=-0.5, scale=0.2):
    self.shift = shift
    self.scale = scale

  def initialize(self, size):
    '''Random number initializer
    Note #1: 'self.scale' specifies the range of the values and with 'self.shift' they can
    Note #2: By default (with scale=0.2 and shift=-0.5) it should return a matrix which con
    Note #3: Use the np.random modul!

    :param size: Dimensions of the matrix.
    :returns: A matrix of random numbers with dimensions specified by 'size'.
    '''
    mat = np.random.random(size) + self.shift
    result = mat * self.scale
    return result  # DONE //
```

Function class

```
class Function:
  def forward(self, input):
    return None

  def __call__(self, input):
    return self.forward(input)

  def backward(self, grads):
    return None
```

Activation functions

```
class Activation(Function):
  def __init__(self):
    pass


class Linear(Activation):
  def forward(self, z):
    return z.astype(np.float)
```

```
  def backward(self, z):
    return np.ones_like(z, dtype=np.float)


class Relu(Activation):
  def forward(self, z):
    '''Forward pass of the Rectified Linear Unit activation function.

    :param z: Input tensor.
    :returns: ReLU(z), see the lecture notes for the definition.
    '''
    return np.maximum(0, z)  # DONE //

  def backward(self, z):
    '''Backward pass of the Rectified Linear Unit activation function.

    :param z: Input tensor.
    :returns: ReLU'(z), see the lecture notes for the definition.
    '''
    return np.where(z > 0, 1, 0)  # DONE //

class Sigmoid(Activation):
  def forward(self, z):
    '''Forward pass of the Sigmoid activation function.

    :param z: Input tensor.
    :returns: sigmoid(z), see the lecture notes for the definition.
    '''
    return 1.0 / (1.0 + np.exp(-1.0 * z))  # TODO

  def backward(self, z):
    '''Backward pass of the Sigmoid activation function.

    :param z: Input tensor.
    :returns: sigmoid'(z), see the lecture notes for the definition.
    '''
    sig = 1.0 / (1.0 + np.exp(-1.0 * z))
    return sig * (1.0 - sig)  # TODO
```

Loss functions

```
class Loss(Function):
  def forward(self, y_true, y_pred):
    return None

  def __call__(self, y_true, y_pred):
    return self.forward(y_true, y_pred)

  def backward(self, y_true, y_pred):
    return None


class BinaryCrossentropy(Loss):
```

```
class BinaryCrossentropy(Loss):

  def forward(self, y_true, y_pred):
    '''Forward pass of the Binary Crossentropy loss.
    Note: Both 'y_true' and 'y_pred' contains a batch of labels => y_true.shape == y_pred.s

    :param y_true: Ground truth labels.
    :param y_pred: Predicted labels.
    :returns: Binary crossentropy loss, see the lecture notes for the definition.
    '''
    epsilon = 0.0000000001
    yshape = y_pred.shape[0]
    result = (1./yshape) * (-np.dot(y_true,np.log(y_pred + epsilon)) - np.dot(1-y_true, np.

    return np.sum(result)       # DONE  //

  def backward(self, y_true, y_pred):

    '''Backward pass of the Binary Crossentropy loss.
    Note #1: The gradient should have the same shape as y_pred (<batch size> x 1)
    Note #2: Keep in mind that the derivative of the loss in the lecture notes is for a log
    Note #3: Here, you do not need to derive respect to the weights!

    :param y_true: Ground truth labels.
    :param y_pred: Predicted labels.
    :returns: Derivative of the binary crossentropy loss, see the lecture notes for the "de
    '''
    epsilon = 0.0000000001
    yshape = y_pred.shape[0]
    y_true = y_true.reshape(y_pred.shape)
    result = -y_true/(y_pred + epsilon) + (1-y_true)/((1-y_pred) + epsilon)
    return result / yshape   # DONE  //
```

Layers

```
class Layer(Function):
  def __init__(self, activation, optimizer=None, weight_init=RandomInitializer(), *args, **
    self.activation = activation
    self.optimizer = optimizer
    self.weight_init = weight_init

  def _forward(self, x):
    return None

  def forward(self, X):
    self.X = X
    self.Z = self._forward(X)
    self.h = self.activation(self.Z)
    return self.h

  def _backward(self, dZ):
    return None, None
```

```
  def backward(self, dh):
    dZ = dh * self.activation.backward(self.Z)
    self.dX, self.grads = self._backward(dZ)
    self._update_weights()
    return self.dX

  def _update_weights(self):
    assert len(self.params) == len(self.grads)
    for idx in range(len(self.params)):
      self.optimizer(self.params[idx], self.grads[idx])
```

Fully-connected (dense) layer

```
class Dense(Layer):
  def __init__(self, size, *args, **kwargs):
    super(Dense, self).__init__(*args, **kwargs)
    self.W = self.weight_init(size)
    self.b = self.weight_init((1, size[1]))
    self.params = [self.W, self.b]

  def _forward(self, X):
    '''Forward pass of the dense layer.
    Note #1: Use self.W and self.b
    Note #2: Input times weight add a bias ==> activate is already taken care of! (see self

    :param X: Input matrix
    :returns: Linear combination, see the lecture notes for the definition.
    '''
    return np.dot(X,self.W) + self.b   # DONE  //

  def _backward(self, dZ):
    '''Backward pass of the dense layer.
    Note: Use self.X

    :param dZ: Gradient of the subsequent layer.
    :returns: A pair (dX and [dW, db]) which contains the partial derivatives respect to th
    '''
    dW = np.dot(self.X.T, dZ)   # DONE  //
    db = np.sum(dZ, axis = 0)   # DONE  //
    dX = np.dot(dZ, self.W.T)   # DONE  //
    return dX, [dW, db]
```

Flatten

```
class Flatten(Layer):
  def __init__(self, *args, **kwargs):
    super(Flatten, self).__init__(activation=Linear(), *args, **kwargs)

  def _forward(self, X):
    return X.reshape((len(X), -1))

  def _backward(self, dZ):
```

```
      return dZ.reshape(self.X.shape), []

    def _update_weights(self):
      pass
```

Max pooling

```
  class Maxpool2d(Layer):
    def __init__(self, *args, **kwargs):
      super(Maxpool2d, self).__init__(activation=Linear(), *args, **kwargs)

    def _forward(self, X):

      '''Forward pass of the max pooling layer.
      :param X: Input matrix
      :returns: Matrix (<batch_size> x <height>//2 x <width>//2 x <n_channels>) after max poo
      '''
      shaping = X.reshape(X.shape[0],X.shape[1]//2, 2,X.shape[2]//2, 2,X.shape[3])
      result = shaping.max(axis=2).max(axis=3)
      mask = np.isclose(X,np.repeat(np.repeat(result,2,axis=1),2,axis=2)).astype(int)
      self.mask = mask  # TODO save the mask for later (_backward) use.
      return result  # TODO

    def _backward(self, dZ):
      '''Backward pass of the max pooling layer.
      Note: Use self.mask too.

      :param dZ: Gradient of the subsequent layer.
      :returns: A pair (dX and []) which contains the partial derivative respect to the input
      '''
      dX = self.mask * (np.repeat(np.repeat(dZ, 2, axis=1), 2, axis=2))  # TODO
      return dX, []

    def _update_weights(self):
      pass
```

Convolutional layer

```
  class Conv2d(Layer):
    def __init__(self, kernel_size, n_channels, n_kernels, pad, use_fast=False, *args, **kwar
      super(Conv2d, self).__init__(*args, **kwargs)
      self.W = self.weight_init((kernel_size, kernel_size, n_channels, n_kernels))
      self.b = self.weight_init((1, 1, 1, n_kernels))
      self.params = [self.W, self.b]
      self.pad = pad
      self.use_fast = use_fast


    def _convolution_fast(self, Y):
      '''Optimized version of the convolution operation (Optional).
      Note #1: Use self.X, self.X_padded, self.W
```

```
    Note #2: There are no return values.
    Note #3: It's an optional task.

    :param Y: Destination (output) matrix (image), see the lecture notes for the "definitio
    '''
    pass  # TODO optional

  #defined function for slicing
  def slicing(self, input, W, b):
    return np.sum(np.multiply(input, W)) + float(b)

  def _convolution_slow(self, Y):
    y_height = Y.shape[1] - self.W.shape[0] + 2*self.pad + 1
    y_width = Y.shape[2] - self.W.shape[1] + 2*self.pad + 1
    Y = np.zeros((self.X.shape[0], y_height, y_width, self.W.shape[3]), dtype=np.float16)
    X_pad = self.X_padded
    for i in range(self.X.shape[0]):
        x = X_pad[i]
        for h in range(y_height):
            for w in range(y_width):
                vstart = h
                vend = vstart + self.kernel_size
                hstart = w
                hend = hstart + self.kernel_size
                for c in range(self.kernel_size):
                    seperate = x[vstart: vend, hstart: hend, :]
                    Y[i, h, w, c] = self.slicing(seperate, self.params[0][:, :, :, c],self.

    '''Naive version (with a bunch of for loops) of the convolution operation.
    Note #1: Use self.X, self.X_padded, self.W
    Note #2: There are no return values.
    Note #3: Both convolution and cross-correlation is acceptable.

    :param Y: Destination (output) matrix (image), see the lecture notes for the "definitio
    '''
    #pass  # TODO

  def _forward(self, X):
    y_height = X.shape[1] - self.W.shape[0] + 2*self.pad + 1
    y_width = X.shape[2] - self.W.shape[1] + 2*self.pad + 1
    Y = np.zeros((X.shape[0], y_height, y_width, self.W.shape[3]), dtype=np.float16)

    if 0 < self.pad:
      X_padded = np.pad(X, ((0, 0), (self.pad, self.pad), (self.pad, self.pad), (0, 0)), 'c
    else:
      X_padded = X
    self.X_padded = X_padded

    if self.use_fast:
      self._convolution_fast(Y)
    else:
      self._convolution_slow(Y)

    Y += self.b
    return Y
```

```
  def _backward_fast(self, dZ):

    '''Optimized version of the backward pass (Optional).
    Note #1: Use self.X, self.X_padded, self.W
    Note #2: It's an optional task.

    :param dZ: Gradient of the subsequent layer.
    :returns: A pair (dX, dW and db) which contains the partial derivatives respect to the
    '''
    db = None  # None optional
    dW = None  # None optional
    dX = None  # None optional
    return dX, dW, db

  def _backward_slow(self, dZ):
    '''Naive version (with a bunch of for loops) of the backward pass.
    Note: Use self.X, self.X_padded, self.W

    :param dZ: Gradient of the subsequent layer.
    :returns: A pair (dX, dW and db) which contains the partial derivatives respect to the
    '''
    db = None  # None
    dW = None  # None
    dX = None  # None
    return dX, dW, db

  def _backward(self, dZ):
    if self.use_fast:
      dX, dW, db = self._backward_fast(dZ)
    else:
      dX, dW, db = self._backward_slow(dZ)

    if 0 < self.pad:
      dX = dX[:, self.pad:-self.pad, self.pad:-self.pad, :]

    return dX, [dW, db]
```

Model class

```
class Model:
  def __init__(self, layers=None, loss=None, optimizer=None):
    self.layers = []
    if layers is not None:
      self.layers = layers
    self.loss = loss
    self.optimizer = optimizer

  def add(self, layer):
    assert isinstance(layer, Layer)
    layer.optimizer = self.optimizer
    self.layers.append(layer)
```

```
  def train(self, x_train, y_train, n_epochs, batch_size, randomize=True, display=True):
    self.losses = []
    for epoch in range(n_epochs):
      idx_list = list(range(len(x_train)))
      if randomize:
        np.random.shuffle(idx_list)
      n_batches = (len(idx_list) + batch_size - 1) // batch_size
      loss = 0.
      for batch_idx in range(n_batches):
        data = x_train[idx_list[batch_idx * batch_size:(batch_idx + 1) * batch_size]]
        # TODO forward pass

        for layer in self.layers:
          data = layer.forward(data)
        # loss
        y_pred = data
        y_true = y_train[idx_list[batch_idx * batch_size:(batch_idx + 1) * batch_size]]
        batch_loss = self.loss(y_true, y_pred)
        loss += batch_loss

        # TODO backward pass
        dx = self.loss.backward(y_true, y_pred)
        for layer in reversed(self.layers):
          dx = layer.backward(dx)

        # display
        #print('Epoch {}/{}: batch {}/{}: batch_loss: {}, avg_loss: {}'.format(epoch+1, n_e
      print('Epoch {}/{}: loss: {}'.format(epoch+1, n_epochs, loss/n_batches))
      self.losses.append(loss / n_batches)
    if display:
      # pass # TODO Plot the learning curve after training.
      plt.plot(self.losses, label='losses')
      plt.title('model Losses')
      plt.xlabel('Epoch')
      plt.ylabel('Loss')
      plt.legend(loc='lower right')

  def predict(self, x_test,y_test, display=True):
    for layer in self.layers:
      data = layer.forward(x_test)
    y_pred = x_test
    y_true = y_test
    acc = np.equal(y_true, np.round(y_pred)).mean()
    print('Accuracy: {}'.format(acc*100)) #Done  //
```

Breast Cancer Wisconsin (Diagnostic) Dataset

For more information, see: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Dia

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdb
```

```
--2019-12-15 21:05:25--  https://archive.ics.uci.edu/ml/machine-learning-databases/bre
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connect
HTTP request sent, awaiting response... 200 OK
Length: 124103 (121K) [application/x-httpd-php]
Saving to: ‘wdbc.data.6’

wdbc.data.6         100%[===================>] 121.19K   646KB/s    in 0.2s

2019-12-15 21:05:25 (646 KB/s) - ‘wdbc.data.6’ saved [124103/124103]
```

```
filename = 'wdbc.data'

with open(filename, 'r') as file:
  lines = file.readlines()

words = [line.split(',') for line in lines]
data = words[:-1]

features = [[float(item) for item in rec[2:]] for rec in data]
features = np.array(features, dtype=np.float32)

label_str_to_num = lambda label: 1. if label == 'M' else 0.
labels = [label_str_to_num(rec[1]) for rec in data]
labels = np.array(labels, dtype=np.float32)

# removing records with missing features (if any feature == 0.)
missing_features = np.any(features == 0., axis=1)
features = features[~missing_features,:]
labels = labels[~missing_features]

print(features.shape)
print(labels.shape)
```

```
(556, 30)
(556,)
```

Data exploration and Pre-processing

Tasks:

- Print the distribution of the labels.
- Print the scales of each features. (min, max, avg, std)
- Randomly split the dataset to training and test sets. (Ratio should be 80-20.)
    - After splitting make sure that the distribution of the labels are similar. (Print the distribut
- Normalize the data by each feature. (Use Z-score standardization.)

The Data has explored in tabular form where first column contains ID, 2nd column contains diagnos like Clump Thickness,Uniformity of Cell Size,Uniformity of Cell Shape,Marginal Adhesion,Single Epit Normal Nucleoli,Mitoses.

```
header = ["id","diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","smoot
print(tabulate(data, headers=header))
```

84348301	M	11.42	20.38	77.58	386.1
84358402	M	20.29	14.34	135.1	1297
843786	M	12.45	15.7	82.57	477.1
844359	M	18.25	19.98	119.6	1040
84458202	M	13.71	20.83	90.2	577.9
844981	M	13	21.82	87.5	519.8
84501001	M	12.46	24.04	83.97	475.9
845636	M	16.02	23.24	102.7	797.8
84610002	M	15.78	17.89	103.6	781
846226	M	19.17	24.8	132.4	1123
846381	M	15.85	23.95	103.7	782.7
84667401	M	13.73	22.61	93.6	578.3
84799002	M	14.54	27.54	96.73	658.8
848406	M	14.68	20.13	94.74	684.5
84862001	M	16.13	20.68	108.1	798.8
849014	M	19.81	22.15	130	1260
8510426	B	13.54	14.36	87.46	566.3
8510653	B	13.08	15.71	85.63	520
8510824	B	9.504	12.44	60.34	273.9
8511133	M	15.34	14.26	102.5	704.4
851509	M	21.16	23.04	137.2	1404
852552	M	16.65	21.38	110	904.6
852631	M	17.14	16.4	116	912.7
852763	M	14.58	21.53	97.41	644.8
852781	M	18.61	20.25	122.1	1094
852973	M	15.3	25.27	102.4	732.4
853201	M	17.57	15.05	115	955.1
853401	M	18.63	25.11	124.8	1088
853612	M	11.84	18.7	77.93	440.6
85382601	M	17.02	23.98	112.8	899.3
854002	M	19.27	26.47	127.9	1162
854039	M	16.13	17.88	107	807.2
854253	M	16.74	21.59	110.1	869.5
854268	M	14.25	21.72	93.63	633
854941	B	13.03	18.42	82.61	523.8
855133	M	14.99	25.2	95.54	698.8
855138	M	13.48	20.82	88.4	559.2
855167	M	13.44	21.58	86.18	563
855563	M	10.95	21.35	71.9	371.1
855625	M	19.07	24.81	128.3	1104
856106	M	13.28	20.28	87.32	545.2
85638502	M	13.17	21.81	85.42	531.5
857010	M	18.65	17.6	123.7	1076
85713702	B	8.196	16.84	51.71	201.9
85715	M	13.17	18.66	85.98	534.6
857155	B	12.05	14.63	78.04	449.3
857156	B	13.49	22.3	86.91	561
857343	B	11.76	21.6	74.72	427.9
857373	B	13.64	16.34	87.21	571.8
857374	B	11.94	18.24	75.71	437.6
857392	M	18.22	18.7	120.3	1033
857438	M	15.1	22.02	97.26	712.8
85759902	B	11.52	18.75	73.34	409
857637	M	19.21	18.57	125.5	1152
857793	M	14.71	21.59	95.55	656.9
857810	B	13.05	19.31	82.61	527.2
858477	B	8.618	11.79	54.34	224.5
858970	B	10.17	14.88	64.55	311.9
858981	B	8.598	20.98	54.66	221.8
858986	M	14.25	22.15	96.42	645.7
859196	B	9.173	13.86	59.2	260.9
85922302	M	12.68	23.84	82.69	499

859283	M	14.78	23.94	97.4	668.3
859464	B	9.465	21.01	60.11	269.4
859465	B	11.31	19.04	71.8	394.1
859471	B	9.029	17.33	58.79	250.5
859487	B	12.78	16.49	81.37	502.5
859575	M	18.94	21.31	123.6	1130
859711	B	8.888	14.64	58.79	244
859717	M	17.2	24.52	114.2	929.4
859983	M	13.8	15.79	90.43	584.1
8610175	B	12.31	16.52	79.19	470.9
8610404	M	16.07	19.65	104.1	817.7
8610629	B	13.53	10.94	87.91	559.2
8610637	M	18.05	16.15	120.2	1006
8610862	M	20.18	23.97	143.7	1245
8610908	B	12.86	18	83.19	506.3
861103	B	11.45	20.97	73.81	401.5
8611161	B	13.34	15.86	86.49	520
8611555	M	25.22	24.91	171.5	1878
8611792	M	19.1	26.29	129.1	1132
8612080	B	12	15.65	76.95	443.3
8612399	M	18.46	18.52	121.1	1075
86135501	M	14.48	21.46	94.25	648.2
86135502	M	19.02	24.59	122	1076
861597	B	12.36	21.8	79.78	466.1
861598	B	14.64	15.24	95.77	651.9
861648	B	14.62	24.02	94.57	662.7
861799	M	15.37	22.76	100.2	728.2
861853	B	13.27	14.76	84.74	551.7
862009	B	13.45	18.3	86.6	555.1
862028	M	15.06	19.83	100.3	705.6
86208	M	20.26	23.03	132.4	1264
86211	B	12.18	17.84	77.79	451.1
862261	B	9.787	19.94	62.11	294.5
862485	B	11.6	12.84	74.34	412.6
862548	M	14.42	19.77	94.48	642.5
862717	M	13.61	24.98	88.05	582.7
862722	B	6.981	13.43	43.79	143.5
862965	B	12.18	20.52	77.22	458.7
862980	B	9.876	19.4	63.95	298.3
862989	B	10.49	19.29	67.41	336.1
863030	M	13.11	15.56	87.21	530.2
863031	B	11.64	18.33	75.17	412.5
863270	B	12.36	18.54	79.01	466.7
86355	M	22.27	19.67	152.8	1509
864018	B	11.34	21.26	72.48	396.5
864033	B	9.777	16.99	62.5	290.2
86408	B	12.63	20.76	82.15	480.4
86409	B	14.26	19.65	97.83	629.9
864292	B	10.51	20.19	68.64	334.2
864496	B	8.726	15.83	55.84	230.9
864685	B	11.93	21.53	76.53	438.6
864726	B	8.95	15.76	58.74	245.2
864729	M	14.87	16.67	98.64	682.5
864877	M	15.78	22.91	105.7	782.6
865128	M	17.95	20.01	114.2	982
865137	B	11.41	10.82	73.34	403.3
86517	M	18.66	17.12	121.4	1077
865423	M	24.25	20.2	166.2	1761
865432	B	14.5	10.89	94.28	640.7
865468	B	13.37	16.39	86.1	553.5
86561	B	13.85	17.21	88.44	588.7
866083	M	13.61	24.69	87.76	572.6
[illegible]	M	[illegible]	[illegible]	[illegible]	[illegible]

866203	M	19	18.91	123.4	1138
866458	B	15.1	16.39	99.58	674.5
866674	M	19.79	25.12	130.4	1192
866714	B	12.19	13.29	79.08	455.8
8670	M	15.46	19.48	101.7	748.9
86730502	M	16.16	21.54	106.2	809.8
867387	B	15.71	13.93	102	761.7
867739	M	18.45	21.91	120.2	1075
868202	M	12.77	22.47	81.72	506.3
868223	B	11.71	16.67	74.72	423.6
868682	B	11.43	15.39	73.06	399.8
868826	M	14.95	17.57	96.85	678.1
868871	B	11.28	13.39	73	384.8
868999	B	9.738	11.97	61.24	288.5
869104	M	16.11	18.05	105.1	813
869218	B	11.43	17.31	73.66	398
869224	B	12.9	15.92	83.74	512.2
869254	B	10.75	14.97	68.26	355.3
869476	B	11.9	14.65	78.11	432.8
869691	M	11.8	16.58	78.99	432
86973701	B	14.95	18.77	97.84	689.5
86973702	B	14.44	15.18	93.97	640.1
869931	B	13.74	17.91	88.12	585
871001501	B	13	20.78	83.51	519.4
871001502	B	8.219	20.7	53.27	203.9
8710441	B	9.731	15.34	63.78	300.2
87106	B	11.15	13.08	70.87	381.9
8711002	B	13.15	15.34	85.31	538.9
8711003	B	12.25	17.94	78.27	460.3
8711202	M	17.68	20.74	117.4	963.7
8711216	B	16.84	19.46	108.4	880.2
871122	B	12.06	12.74	76.84	448.6
871149	B	10.9	12.96	68.69	366.8
8711561	B	11.75	20.18	76.1	419.8
8711803	M	19.19	15.94	126.3	1157
871201	M	19.59	18.15	130.7	1214
8712064	B	12.34	22.22	79.85	464.5
8712289	M	23.27	22.04	152.1	1686
8712291	B	14.97	19.76	95.5	690.2
87127	B	10.8	9.71	68.77	357.6
8712729	M	16.78	18.8	109.3	886.3
8712766	M	17.47	24.68	116.1	984.6
8712853	B	14.97	16.95	96.22	685.9
87139402	B	12.32	12.39	78.85	464.1
87163	M	13.43	19.63	85.84	565.4
87164	M	15.46	11.89	102.5	736.9
871641	B	11.08	14.71	70.21	372.7
871642	B	10.66	15.15	67.49	349.6
872113	B	8.671	14.45	54.42	227.2
872608	B	9.904	18.06	64.6	302.4
87281702	M	16.46	20.11	109.3	832.9
873357	B	13.01	22.22	82.01	526.4
873586	B	12.81	13.06	81.29	508.8
873592	M	27.22	21.87	182.1	2250
873593	M	21.09	26.57	142.7	1311
873701	M	15.7	20.31	101.2	766.6
873843	B	11.41	14.92	73.53	402
873885	M	15.28	22.41	98.92	710.6
874158	B	10.08	15.11	63.76	317.5
874217	M	18.31	18.58	118.6	1041
874373	B	11.71	17.19	74.68	420.3
874662	B	11.81	17.39	75.27	428.9
874839	B	12.3	15.9	78.83	463.7

874839	B	12.5	15.9	78.83	463.7
874858	M	14.22	23.12	94.37	609.9
875093	B	12.77	21.41	82.02	507.4
875099	B	9.72	18.22	60.73	288.1
875263	M	12.34	26.86	81.15	477.4
87556202	M	14.86	23.21	100.4	671.4
875878	B	12.91	16.33	82.53	516.4
875938	M	13.77	22.29	90.63	588.9
877159	M	18.08	21.84	117.4	1024
877486	M	19.18	22.49	127.5	1148
877500	M	14.45	20.22	94.49	642.7
877501	B	12.23	19.56	78.54	461
877989	M	17.54	19.32	115.1	951.6
878796	M	23.29	26.67	158.9	1685
87880	M	13.81	23.75	91.56	597.8
87930	B	12.47	18.6	81.09	481.9
879523	M	15.12	16.68	98.78	716.6
879804	B	9.876	17.27	62.92	295.4
879830	M	17.01	20.26	109.7	904.3
8810158	B	13.11	22.54	87.02	529.4
8810436	B	15.27	12.91	98.17	725.5
881046502	M	20.58	22.14	134.7	1290
8810528	B	11.84	18.94	75.51	428
8810703	M	28.11	18.47	188.5	2499
881094802	M	17.42	25.56	114.5	948
8810955	M	14.19	23.81	92.87	610.7
8810987	M	13.86	16.93	90.96	578.9
8811523	B	11.89	18.35	77.32	432.2
8811779	B	10.2	17.48	65.05	321.2
8811842	M	19.8	21.56	129.7	1230
88119002	M	19.53	32.47	128	1223
8812816	B	13.65	13.16	87.88	568.9
8812818	B	13.56	13.9	88.59	561.3
8812844	B	10.18	17.53	65.12	313.1
8812877	M	15.75	20.25	102.6	761.3
8813129	B	13.27	17.02	84.55	546.4
88143502	B	14.34	13.47	92.51	641.2
88147101	B	10.44	15.46	66.62	329.6
88147102	B	15	15.51	97.45	684.5
88147202	B	12.62	23.97	81.35	496.4
881861	M	12.83	22.33	85.26	503.2
881972	M	17.05	19.08	113.4	895
88199202	B	11.32	27.08	71.76	395.7
88203002	B	11.22	33.81	70.79	386.8
88206102	M	20.51	27.81	134.4	1319
882488	B	9.567	15.91	60.21	279.6
88249602	B	14.03	21.25	89.79	603.4
88299702	M	23.21	26.97	153.5	1670
883263	M	20.48	21.46	132.5	1306
883270	B	14.22	27.85	92.55	623.9
88330202	M	17.46	39.28	113.4	920.6
88350402	B	13.64	15.6	87.38	575.3
883539	B	12.42	15.04	78.61	476.5
883852	B	11.3	18.19	73.93	389.4
88411702	B	13.75	23.77	88.54	590
884180	M	19.4	23.5	129.1	1155
884437	B	10.48	19.86	66.72	337.7
884448	B	13.2	17.43	84.13	541.6
884626	B	12.89	14.11	84.95	512.2
88466802	B	10.65	25.22	68.01	347
884689	B	11.52	14.93	73.87	406.3
884948	M	20.94	23.56	138.9	1364
88518501	B	11.5	18.45	73.28	407.4

88518501	B	11.5	18.45	73.28	407.4
885429	M	19.73	19.82	130.7	1206
8860702	M	17.3	17.08	113	928.2
886226	M	19.45	19.33	126.5	1169
886452	M	13.96	17.05	91.43	602.4
88649001	M	19.55	28.77	133.6	1207
886776	M	15.32	17.27	103.2	713.3
887181	M	15.66	23.2	110.2	773.5
88725602	M	15.53	33.56	103.7	744.9
887549	M	20.31	27.06	132.9	1288
888264	M	17.35	23.06	111	933.1
888570	M	17.29	22.13	114.4	947.8
889403	M	15.61	19.38	100	758.6
889719	M	17.19	22.07	111.6	928.3
88995002	M	20.73	31.12	135.7	1419
8910251	B	10.6	18.95	69.28	346.4
8910499	B	13.59	21.84	87.16	561
8910506	B	12.87	16.21	82.38	512.2
8910720	B	10.71	20.39	69.5	344.9
8910721	B	14.29	16.82	90.3	632.6
8910748	B	11.29	13.04	72.23	388
8910988	M	21.75	20.99	147.3	1491
8910996	B	9.742	15.67	61.5	289.9
8911163	M	17.93	24.48	115.2	998.9
8911164	B	11.89	17.36	76.2	435.6
8911230	B	11.33	14.16	71.79	396.6
8911670	M	18.81	19.98	120.9	1102
8911800	B	13.59	17.84	86.24	572.3
8911834	B	13.85	15.18	88.99	587.4
8912049	M	19.16	26.6	126.2	1138
8912055	B	11.74	14.02	74.24	427.3
89122	M	19.4	18.18	127.2	1145
8912280	M	16.24	18.77	108.8	805.1
8912284	B	12.89	15.7	84.08	516.6
8912521	B	12.58	18.4	79.83	489
8912909	B	11.94	20.76	77.87	441
8913	B	12.89	13.12	81.89	515.9
8913049	B	11.26	19.96	73.72	394.1
89143601	B	11.37	18.89	72.17	396
89143602	B	14.41	19.73	96.03	651
8915	B	14.96	19.1	97.03	687.3
891670	B	12.95	16.02	83.14	513.7
891703	B	11.85	17.46	75.54	432.7
891716	B	12.72	13.78	81.78	492.1
891923	B	13.77	13.27	88.06	582.7
891936	B	10.91	12.35	69.14	363.7
892189	M	11.76	18.14	75	431.1
892214	B	14.26	18.17	91.22	633.1
892399	B	10.51	23.09	66.85	334.2
892438	M	19.53	18.9	129.5	1217
892604	B	12.46	19.89	80.43	471.3
89263202	M	20.09	23.86	134.7	1247
892657	B	10.49	18.61	66.86	334.3
89296	B	11.46	18.16	73.59	403.1
893061	B	11.6	24.49	74.23	417.2
89344	B	13.2	15.82	84.07	537.3
89346	B	9	14.4	56.36	246.3
893526	B	13.5	12.71	85.69	566.2
893548	B	13.05	13.84	82.71	530.6
893783	B	11.7	19.11	74.33	418.7
89382601	B	14.61	15.69	92.68	664.9
89382602	B	12.76	13.37	82.29	504.1
893988	B	11.54	10.72	73.73	409.1

894047	B	8.597	18.6	54.09	221.2
894089	B	12.49	16.85	79.19	481.6
894090	B	12.18	14.08	77.25	461.4
894326	M	18.22	18.87	118.7	1027
894329	B	9.042	18.9	60.07	244.5
894335	B	12.43	17	78.6	477.3
894604	B	10.25	16.18	66.52	324.2
894618	M	20.16	19.66	131.1	1274
894855	B	12.86	13.32	82.82	504.8
895100	M	20.34	21.51	135.9	1264
89511501	B	12.2	15.21	78.01	457.9
89511502	B	12.67	17.3	81.25	489.9
89524	B	14.11	12.88	90.03	616.5
895299	B	12.03	17.93	76.09	446
8953902	M	16.27	20.71	106.9	813.7
895633	M	16.26	21.88	107.5	826.8
896839	M	16.03	15.51	105.8	793.2
896864	B	12.98	19.35	84.52	514
897132	B	11.22	19.86	71.94	387.3
897137	B	11.25	14.78	71.38	390
897374	B	12.3	19.02	77.88	464.4
89742801	M	17.06	21	111.8	918.6
897604	B	12.99	14.23	84.08	514.3
897630	M	18.77	21.43	122.9	1092
897880	B	10.05	17.53	64.41	310.8
89812	M	23.51	24.27	155.1	1747
89813	B	14.42	16.54	94.15	641.2
898143	B	9.606	16.84	61.64	280.5
89827	B	11.06	14.96	71.49	373.9
898431	M	19.68	21.68	129.9	1194
89864002	B	11.71	15.45	75.03	420.3
898677	B	10.26	14.71	66.2	321.6
898678	B	12.06	18.9	76.66	445.3
89869	B	14.76	14.74	94.87	668.7
898690	B	11.47	16.03	73.02	402.7
899147	B	11.95	14.96	77.23	426.7
899187	B	11.66	17.07	73.7	421
899667	M	15.75	19.22	107.1	758.6
899987	M	25.73	17.46	174.2	2010
9010018	M	15.08	25.74	98	716.6
901011	B	11.14	14.07	71.24	384.6
9010258	B	12.56	19.07	81.92	485.8
9010259	B	13.05	18.59	85.09	512
901028	B	13.87	16.21	88.52	593.7
9010333	B	8.878	15.49	56.74	241
901034301	B	9.436	18.32	59.82	278.6
901034302	B	12.54	18.07	79.42	491.9
901041	B	13.3	21.57	85.24	546.1
9010598	B	12.76	18.84	81.87	496.6
9010872	B	16.5	18.29	106.6	838.1
9010877	B	13.4	16.95	85.48	552.4
901088	M	20.44	21.78	133.8	1293
9011494	M	20.2	26.83	133.7	1234
9011495	B	12.21	18.02	78.31	458.4
9011971	M	21.71	17.25	140.9	1546
9012000	M	22.01	21.9	147.2	1482
9012315	M	16.35	23.29	109	840.4
9012568	B	15.19	13.21	97.65	711.8
9012795	M	21.37	15.1	141.3	1386
901288	M	20.64	17.35	134.8	1335
9013005	B	13.69	16.07	87.84	579.1
901303	B	16.17	16.07	106.3	788.5

901315	B	10.57	20.22	70.15	338.3
9013579	B	13.46	28.21	85.89	562.1
9013594	B	13.66	15.15	88.27	580.6
9013838	M	11.08	18.83	73.3	361.6
901549	B	11.27	12.96	73.16	386.3
901836	B	11.04	14.93	70.67	372.7
90250	B	12.05	22.72	78.75	447.8
90251	B	12.39	17.48	80.64	462.9
902727	B	13.28	13.72	85.79	541.8
90291	M	14.6	23.29	93.97	664.7
902975	B	12.21	14.09	78.78	462
902976	B	13.88	16.16	88.37	596.6
903011	B	11.27	15.5	73.38	392
90312	M	19.55	23.21	128.9	1174
90317302	B	10.26	12.22	65.75	321.6
903483	B	8.734	16.84	55.27	234.3
903507	M	15.49	19.97	102.4	744.7
903516	M	21.61	22.28	144.4	1407
903554	B	12.1	17.72	78.07	446.2
903811	B	14.06	17.18	89.75	609.1
90401601	B	13.51	18.89	88.1	558.1
90401602	B	12.8	17.46	83.05	508.3
904302	B	11.06	14.83	70.31	378.2
904357	B	11.8	17.26	75.26	431.9
90439701	M	17.91	21.02	124.4	994
904647	B	11.93	10.91	76.14	442.7
904689	B	12.96	18.29	84.18	525.2
9047	B	12.94	16.17	83.18	507.6
904969	B	12.34	14.95	78.29	469.1
904971	B	10.94	18.59	70.39	370
905189	B	16.14	14.86	104.3	800
905190	B	12.85	21.37	82.63	514.5
90524101	M	17.99	20.66	117.8	991.7
905501	B	12.27	17.92	78.41	466.1
905502	B	11.36	17.57	72.49	399.8
905520	B	11.04	16.83	70.92	373.2
905539	B	9.397	21.68	59.75	268.8
905557	B	14.99	22.11	97.53	693.7
905680	M	15.13	29.81	96.71	719.5
905686	B	11.89	21.17	76.39	433.8
905978	B	9.405	21.7	59.6	271.2
90602302	M	15.5	21.08	102.9	803.1
906024	B	12.7	12.17	80.88	495
906290	B	11.16	21.41	70.95	380.3
906539	B	11.57	19.04	74.2	409.7
906564	B	14.69	13.98	98.22	656.1
906616	B	11.61	16.02	75.46	408.2
906878	B	13.66	19.13	89.46	575.3
907145	B	9.742	19.12	61.93	289.7
907367	B	10.03	21.28	63.19	307.3
907409	B	10.48	14.98	67.49	333.6
90745	B	10.8	21.98	68.79	359.9
90769601	B	11.13	16.62	70.47	381.1
90769602	B	12.72	17.67	80.98	501.3
907914	M	14.9	22.53	102.1	685
907915	B	12.4	17.68	81.47	467.8
908194	M	20.18	19.54	133.8	1250
908445	M	18.82	21.97	123.7	1110
908469	B	14.86	16.94	94.89	673.7
908489	M	13.98	19.62	91.12	599.5
908916	B	12.87	19.54	82.67	509.2
909220	B	14.04	15.98	89.78	611.2

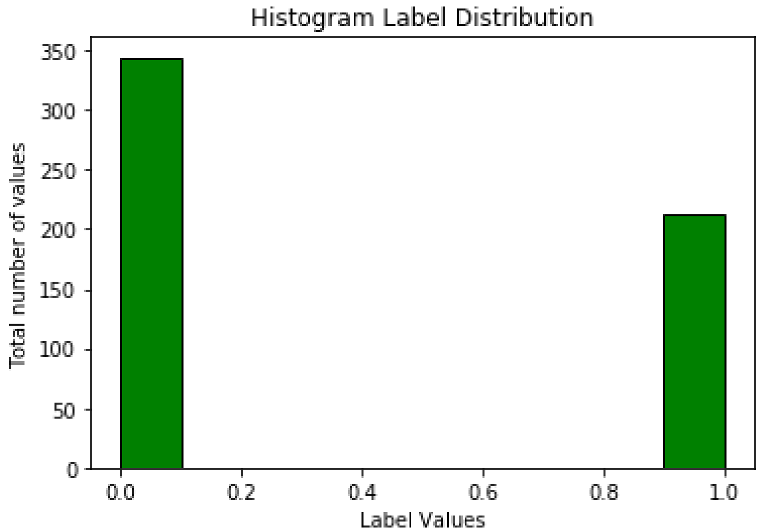
909231	B	13.85	19.6	88.68	592.6
909410	B	14.02	15.66	89.59	606.5
909411	B	10.97	17.2	71.73	371.5
909445	M	17.27	25.42	112.4	928.8
90944601	B	13.78	15.79	88.37	585.9
909777	B	10.57	18.32	66.82	340.9
9110127	M	18.03	16.85	117.5	990
9110720	B	11.99	24.89	77.61	441.3
9110732	M	17.75	28.03	117.3	981.6
9110944	B	14.8	17.66	95.88	674.8
911150	B	14.53	19.34	94.25	659.7
911157302	M	21.1	20.52	138.1	1384
9111596	B	11.87	21.54	76.83	432
9111805	M	19.59	25	127.7	1191
9111843	B	12	28.23	76.77	442.5
911201	B	14.53	13.98	93.86	644.2
911202	B	12.62	17.15	80.62	492.9
9112085	B	13.38	30.72	86.34	557.2
9112366	B	11.63	29.29	74.87	415.1
9112367	B	13.21	25.25	84.1	537.9
9112594	B	13	25.13	82.61	520.2
9112712	B	9.755	28.2	61.68	290.9
911296201	M	17.08	27.15	111.2	930.9
911296202	M	27.42	26.27	186.9	2501
9113156	B	14.4	26.99	92.25	646.1
911320501	B	11.6	18.36	73.88	412.7
911320502	B	13.17	18.22	84.28	537.3
9113239	B	13.24	20.13	86.87	542.9
9113455	B	13.14	20.74	85.98	536.9
9113514	B	9.668	18.1	61.06	286.3
9113538	M	17.6	23.33	119	980.5
911366	B	11.62	18.18	76.38	408.8
9113778	B	9.667	18.49	61.49	289.1
9113816	B	12.04	28.14	76.85	449.9
911384	B	14.92	14.93	96.45	686.9
9113846	B	12.27	29.97	77.42	465.4
911391	B	10.88	15.62	70.41	358.9
911408	B	12.83	15.73	82.89	506.9
911654	B	14.2	20.53	92.41	618.4
911673	B	13.9	16.62	88.97	599.4
911685	B	11.49	14.59	73.99	404.9
911916	M	16.25	19.51	109.8	815.8
912193	B	12.16	18.03	78.29	455.3
91227	B	13.9	19.24	88.73	602.9
912519	B	13.47	14.06	87.32	546.3
912558	B	13.7	17.64	87.76	571.1
912600	B	15.73	11.28	102.8	747.2
913063	B	12.45	16.41	82.85	476.7
913102	B	14.64	16.85	94.21	666
913505	M	19.44	18.82	128.1	1167
913512	B	11.68	16.17	75.49	420.5
913535	M	16.69	20.2	107.1	857.6
91376701	B	12.25	22.44	78.18	466.5
91376702	B	17.85	13.23	114.6	992.1
914062	M	18.01	20.56	118.4	1007
914101	B	12.46	12.83	78.83	477.3
914102	B	13.16	20.54	84.06	538.7
914333	B	14.87	20.21	96.12	680.9
914366	B	12.65	18.17	82.69	485.6
914580	B	12.47	17.31	80.45	480.1
914769	M	18.49	17.52	121.3	1068
91485	M	20.59	21.24	137.8	1320

914862	B	15.04	16.74	98.73	689.4
91504	M	13.82	24.49	92.33	595.9
91505	B	12.54	16.32	81.25	476.3
915143	M	23.09	19.83	152.1	1682
915186	B	9.268	12.87	61.49	248.7
915276	B	9.676	13.14	64.12	272.5
91544001	B	12.22	20.04	79.47	453.1
91544002	B	11.06	17.12	71.25	366.5
915452	B	16.3	15.7	104.7	819.8
915460	M	15.46	23.95	103.8	731.3
91550	B	11.74	14.69	76.31	426
915664	B	14.81	14.7	94.66	680.7
915691	M	13.4	20.52	88.64	556.7
915940	B	14.58	13.66	94.29	658.8
91594602	M	15.05	19.07	97.26	701.9
916221	B	11.34	18.61	72.76	391.2
916799	M	18.31	20.58	120.8	1052
916838	M	19.89	20.26	130.5	1214
917062	B	12.88	18.22	84.45	493.1
917080	B	12.75	16.7	82.51	493.8
917092	B	9.295	13.9	59.96	257.8
91762702	M	24.63	21.6	165.5	1841
91789	B	11.26	19.83	71.3	388.1
917896	B	13.71	18.68	88.73	571
917897	B	9.847	15.68	63	293.2
91805	B	8.571	13.1	54.53	221.3
91813701	B	13.46	18.75	87.44	551.1
91813702	B	12.34	12.27	78.94	468.5
918192	B	13.94	13.17	90.31	594.2
918465	B	12.07	13.44	77.83	445.2
91858	B	11.75	17.56	75.89	422.9
91903901	B	11.67	20.02	75.21	416.2
91903902	B	13.68	16.33	87.76	575.5
91930402	M	20.47	20.67	134.7	1299
919537	B	10.96	17.62	70.79	365.6
919555	M	20.55	20.86	137.8	1308
91979701	M	14.27	22.55	93.77	629.8
919812	B	11.69	24.44	76.37	406.4
921092	B	7.729	25.49	47.98	178.8
921362	B	7.691	25.44	48.34	170.4
921385	B	11.54	14.44	74.65	402.9
921386	B	14.47	24.99	95.81	656.4
921644	B	14.74	25.42	94.7	668.6
922296	B	13.21	28.06	84.88	538.4
922297	B	13.87	20.7	89.77	584.8
922576	B	13.62	23.23	87.19	573.2
922577	B	10.32	16.35	65.31	324.9
922840	B	10.26	16.58	65.85	320.8
923169	B	9.683	19.34	61.05	285.7
923465	B	10.82	24.21	68.89	361.6
923748	B	10.86	21.48	68.51	360.5
923780	B	11.13	22.44	71.49	378.4
924084	B	12.77	29.43	81.35	507.9
924342	B	9.333	21.94	59.01	264
924632	B	12.88	28.92	82.5	514.3
924934	B	10.29	27.61	65.67	321.4
924964	B	10.16	19.59	64.73	311.7
925236	B	9.423	27.88	59.26	271.3
925277	B	14.59	22.68	96.39	657.1
925291	B	11.51	23.93	74.52	403.5
925292	B	14.05	27.15	91.38	600.4
925311	B	11.2	29.37	70.67	386

925622	M	15.22	30.62	103.4	716.9
926125	M	20.92	25.09	143	1347
926424	M	21.56	22.39	142	1479
926682	M	20.13	28.25	131.2	1261
926954	M	16.6	28.08	108.3	858.1
927241	M	20.6	29.33	140.1	1265

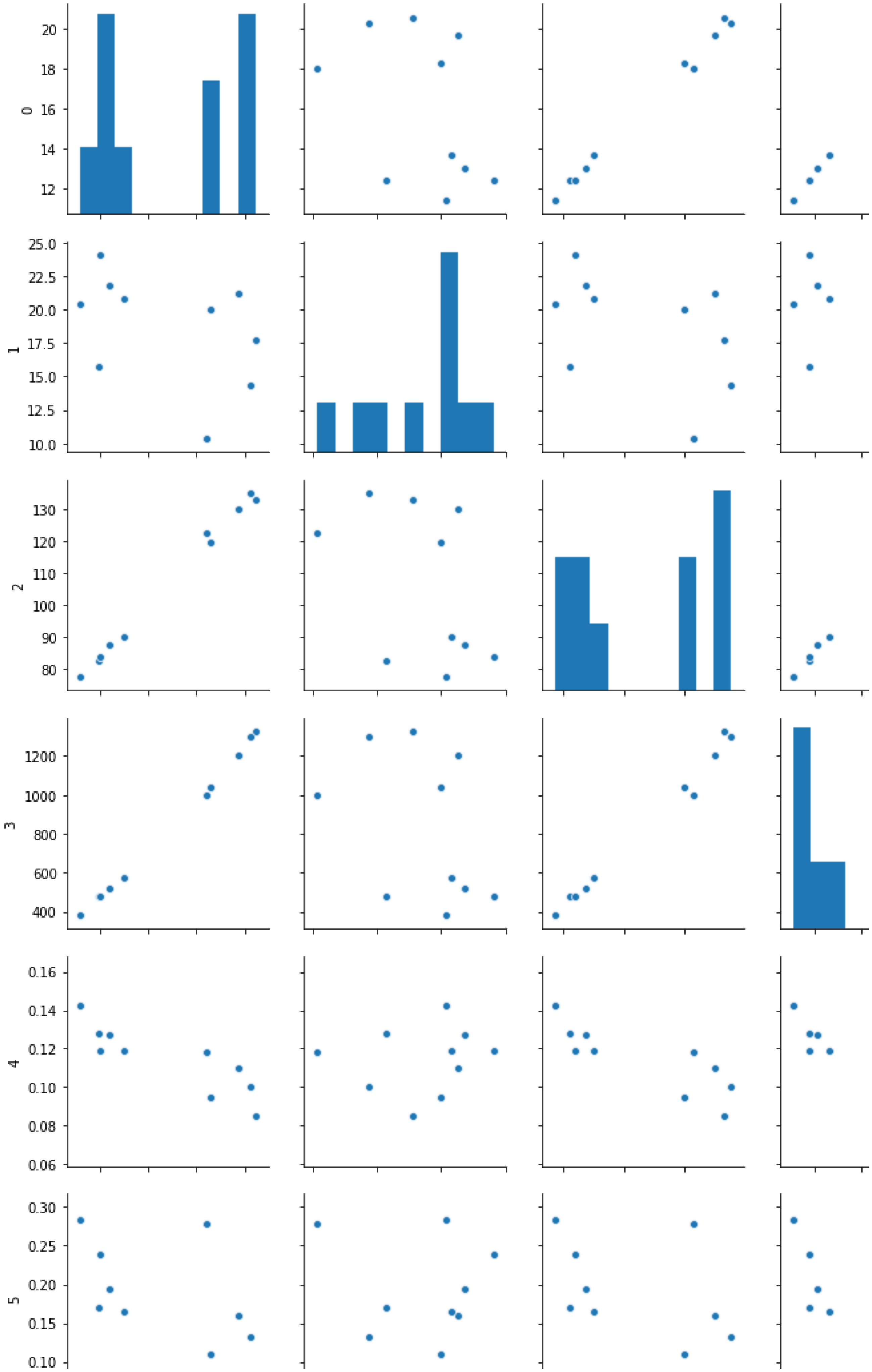
```
plt.hist(labels, color = 'green', edgecolor = 'black')
plt.title('Histogram Label Distribution')
plt.xlabel('Label Values')
plt.ylabel('Total number of values')
```

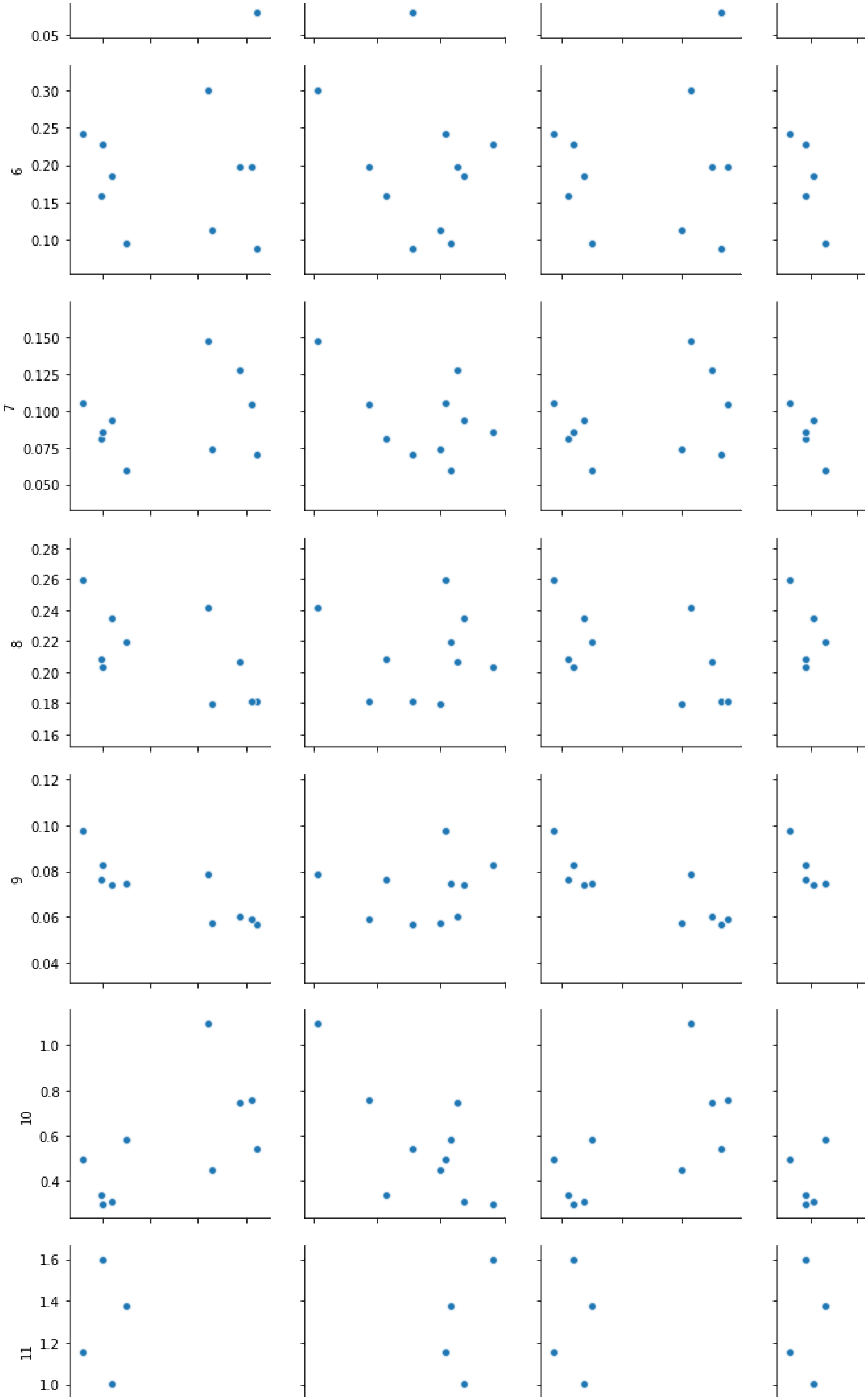
```
Text(0, 0.5, 'Total number of values')
```

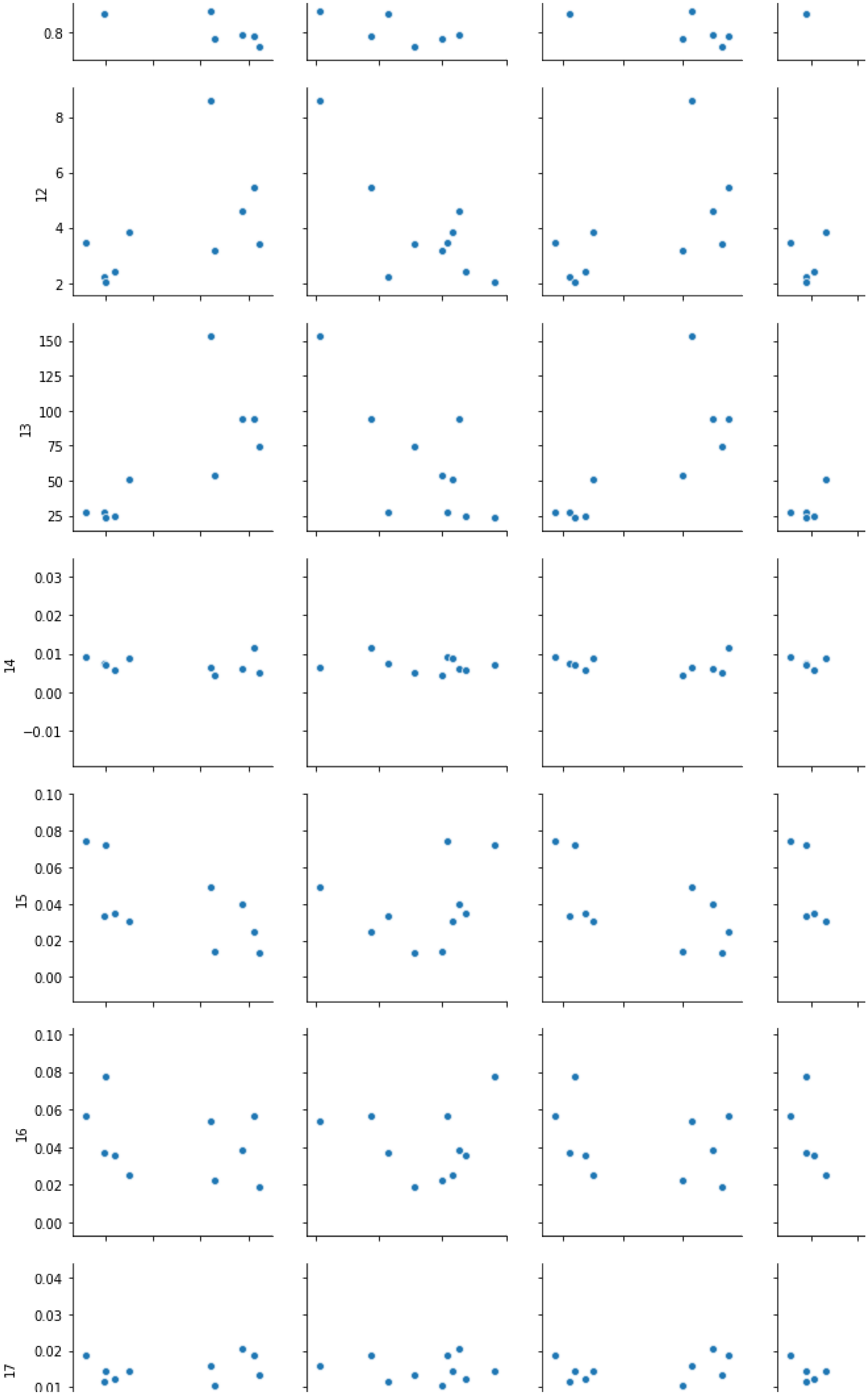


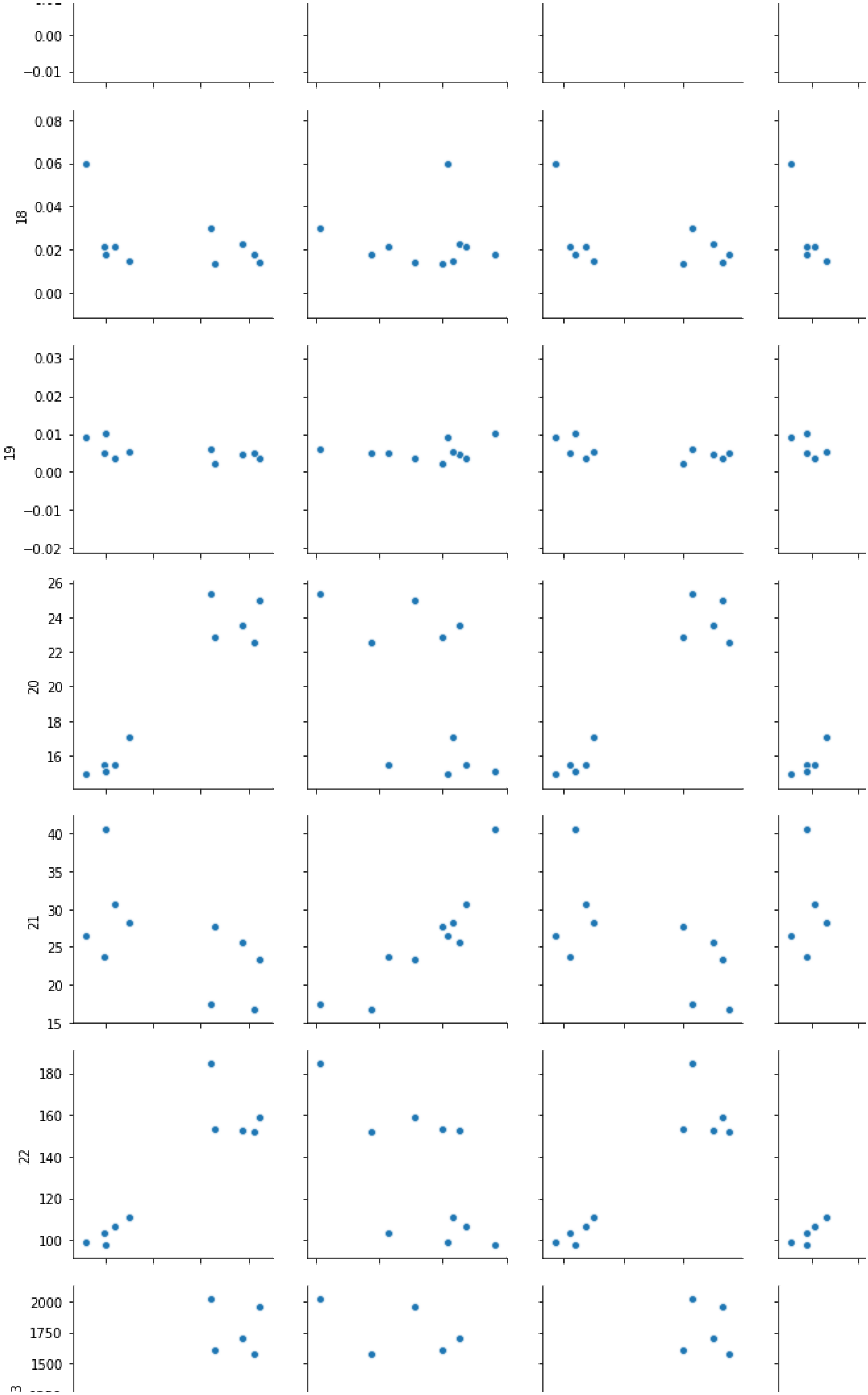

```
import pandas as pd
df = pd.DataFrame(features)
#It takes long time
#sns.pairplot(df[:10])
```

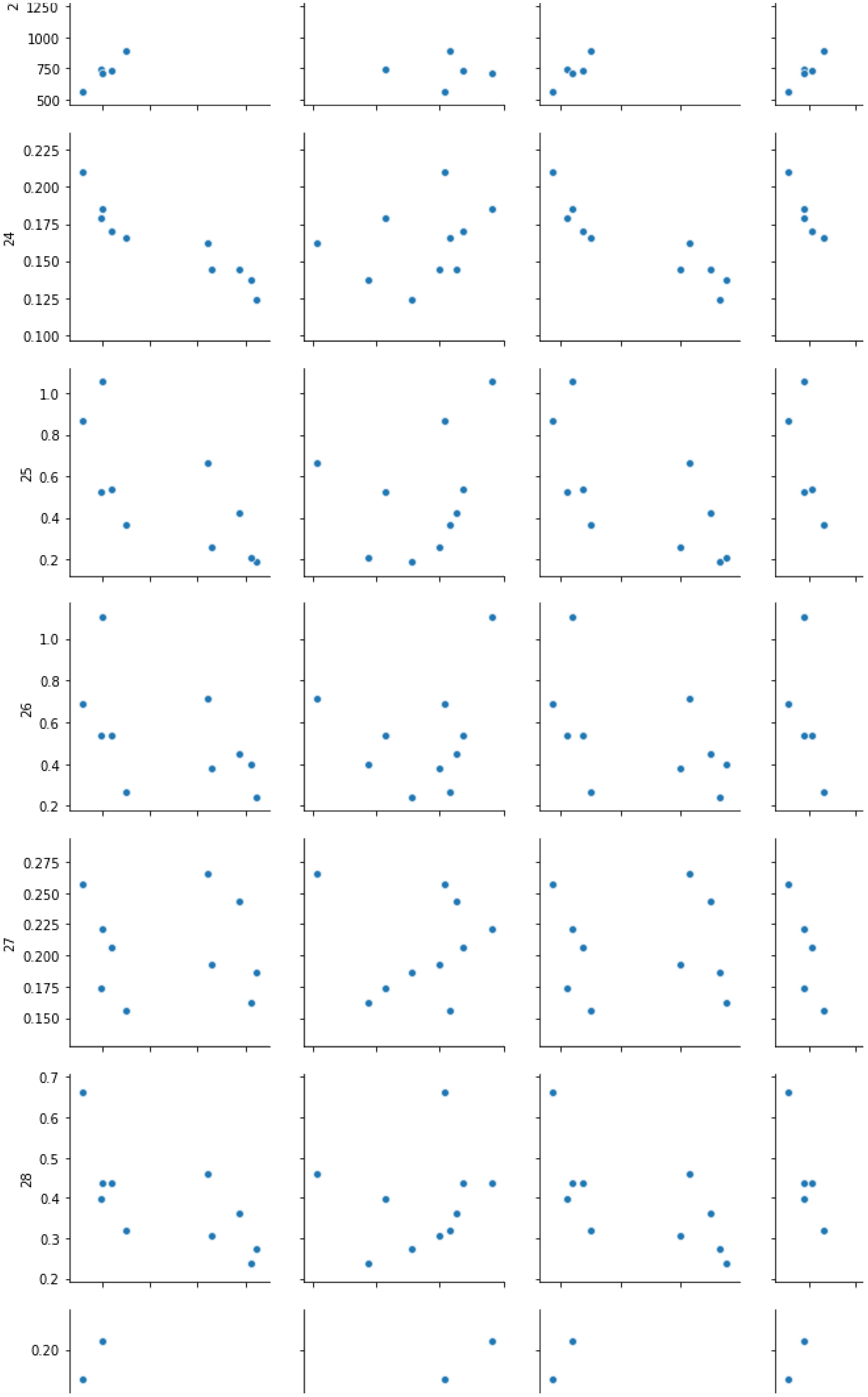
```
<seaborn.axisgrid.PairGrid at 0x7fe48fe3d588>
```

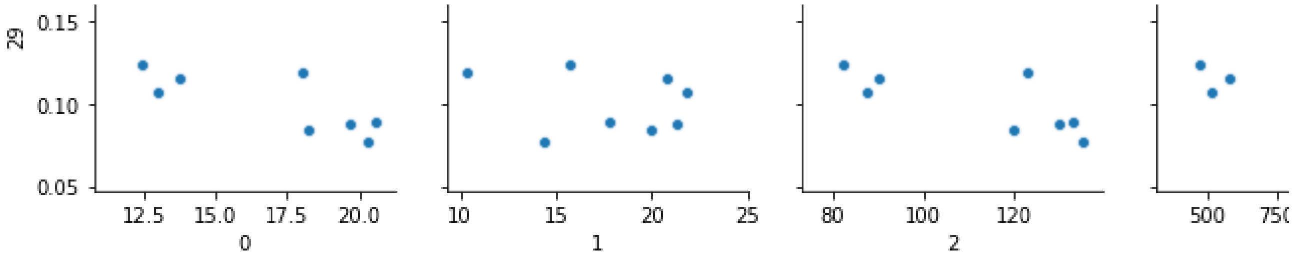



0.05
6
0.30
0.25
0.20
0.15
0.10
7
0.150
0.125
0.100
0.075
0.050
8
0.28
0.26
0.24
0.22
0.20
0.18
0.16
9
0.12
0.10
0.08
0.06
0.04
10
1.0
0.8
0.6
0.4
11
1.6
1.4
1.2
1.0


0.8
12
2
4
6
8
13
25
50
75
100
125
150
14
−0.01
0.00
0.01
0.02
0.03
15
0.00
0.02
0.04
0.06
0.08
0.10
16
0.00
0.02
0.04
0.06
0.08
0.10
17
0.01
0.02
0.03
0.04


0.00
−0.01
18
0.08
0.06
0.04
0.02
0.00
19
0.03
0.02
0.01
0.00
−0.01
−0.02
20
26
24
22
20
18
16
21
40
35
30
25
20
15
22
180
160
140
120
100
2000
1750
1500


2
1250
1000
750
500
24
0.225
0.200
0.175
0.150
0.125
0.100
25
1.0
0.8
0.6
0.4
0.2
26
1.0
0.8
0.6
0.4
0.2
27
0.275
0.250
0.225
0.200
0.175
0.150
28
0.7
0.6
0.5
0.4
0.3
0.2
0.20




```
count=0
mi=[];mx=[];ag=[];std=[]
head=["radius_mean","texture_mean","perimeter_mean","area_mean","smoothness_mean","compactn

for i in np.arange(features.shape[1]):
  mi.append(np.min(features[i]))
  mx.append(np.max(features[i]))
  ag.append(np.average(features[i]))
  std.append(np.std(features[i]))
  count+=1

table = [head,mi,mx,ag,std]
df = pd.DataFrame(table)
df = df.transpose()
df.columns = ["Feature_name","Minimum","Maximum","Average","Std"]

df
```

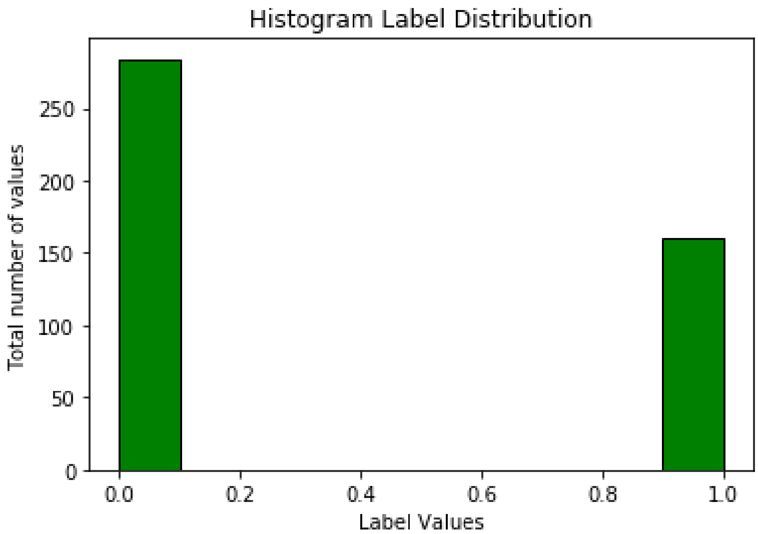
	Feature_name	Minimum	Maximum	Average	Std
0	radius_mean	0.006193	2019	118.873	397.013
1	texture_mean	0.003532	1956	124.697	415.051
2	perimeter_mean	0.004571	1709	112.913	366.81
3	area_mean	0.00911	567.7	41.3334	120.848
4	smoothness_mean	0.005115	1575	111.223	357.933
5	compactness_mean	0.005082	741.6	50.1848	155.081
6	concavity_mean	0.002179	1606	102.265	336.145
7	concave_points_mean	0.005412	897	60.4813	187.397
8	symmetry_mean	0.003749	739.3	52.1632	158.796
9	fractal_dim_mean	0.007149	711.4	49.7811	150.334
10	radius_se	0.003042	1150	77.0848	245.666
11	texture_se	0.004144	1299	82.1065	266.406
12	perimeter_se	0.003139	1332	98.949	305.403
13	area_se	0.003002	876.5	66.7292	206.059
14	smoothness_se	0.006429	697.7	52.9291	159.175
15	compactness_se	0.005466	943.2	65.3177	202.001
16	concavity_se	0.002085	1138	72.5819	233.363
17	concave_points_se	0.004142	1315	83.6794	270.352
18	symmetry_se	0.001997	2398	139.911	476.718
19	fractual_dim_se	0.0023	711.2	51.8429	159.688
20	radius_worst	0.002425	630.5	47.1566	143.649
21	texture_worst	0.002968	314.9	26.0746	73.6016
22	perimeter_worst	0.004394	980.9	67.7467	212.13
23	area_worst	0.001987	2615	151.841	522.299
24	smoothness_worst	0.002801	2215	120.498	422.112
25	compactness_worst	0.007444	1461	94.7638	302.699
26	concavity_worst	0.003711	896.9	62.5318	194.07
27	concave_points_worst	0.004217	1403	98.3413	312.001
28	symmetry_worst	0.002967	1269	80.0702	257.666

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features,labels, test_size = 0.2, rando

print("x train: ", X_train.shape)
print("x test: ", X_test.shape)
print("y train: ", y_train.shape)
print("y test: ", y_test.shape)


plt.hist(y_train, color = 'green', edgecolor = 'black')
plt.title('Histogram Label Distribution')
plt.xlabel('Label Values')
plt.ylabel('Total number of values')
```

```
x train:  (444, 30)
x test:  (112, 30)
y train:  (444,)
y test:  (112,)
Text(0, 0.5, 'Total number of values')
```




```
#Normalization
from scipy.stats import zscore
X_train_normed = zscore(X_train)
X_test_normed = zscore(X_test)

print(X_train_normed.shape)
print(X_test_normed.shape)

print(tabulate(X_train_normed, headers=head))
```