МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторные работы по дисциплине

«Введение в машинное обучение»

Выполнил:

Студент гр.09-305

Шарафеев Марат

Преподаватель:

Шустова Е.П.

Казань-2024

# **Задание №11. Реализация алгоритмов нейронных сетей.**

**Выполните следующее задачи:**

1. Подберите свои данные для решения задачи классификации с помощью дерева решений. Для этого надо воспользоваться известными репозиториями данных (см., например, http://archive.ics.uci.edu/ml/index.php). К указанному ниже сроку закрепите эти данные в Teams через ответить в начатой мною беседе. Закрепите это означает, что надо привести краткое описание выбранных Вами данных и интернет-ссылку на эти данные. Перед поиском своих данных посмотрите в указанной выше беседе какие данные уже заняты другими студентами.
2. Установите необходимые внешние библиотеки.
3. Приведите скриншоты путей, в которые Вы установили Python, PyCharm и внешние библиотеки.
4. Импортируйте необходимые пакеты и классы:
5. Осуществите описание Вашего набора данных. Загрузите Ваши данные и распечатайте первые 10 наблюдений.
6. Подготовьте данные для работы. А именно: определите, есть ли в наборе данных пропущенные значения; если в данных присутствуют пропущенные значения, то либо удалите соответствующие наблюдения, либо рассчитайте их значение известным Вам методом; удалите выбросы, если они есть.
7. Рассчитайте и выведите матрицу корреляции для Ваших данных. Укажите какой признак Вы берете за переменную отклика, а какие за факторы. Обоснуйте свое решение.
8. Сформируйте обучающую и тестовую выборки с помощью метода train\_test\_split из scikitlearn (например так: train\_test\_split(X, y, test\_size = 0.2, random\_state=9).
9. Если число факторов m <= 2, то визуализируйте данные (постройте облако точек).
10. Спроектируйте и реализуйте на Python простейшую нейронную сеть - персептрон. Опишите алгоритм работы Вашей нейронной сети, объясните полученные результаты работы Вашей сети. Сделайте выводы из полученного результата.
11. Сравните теоретические и эмпирические значения переменной отклика на каких-нибудь двух значениях наблюдений из тестовой выборки. Сделайте вывод.
12. Приведите полный код для решения задачи. Код сопровождайте комментариями.
13. Оформите отчёт о результатах исследования и прикрепите его в ЭОР «Введение в машинное обучение на Python» по ссылке <https://repository.kpfu.ru/?p_id=236426> в указанный ниже период до или во время своей пары.
14. К этому же заданию прикрепите файл с данными и py-файл с программой.

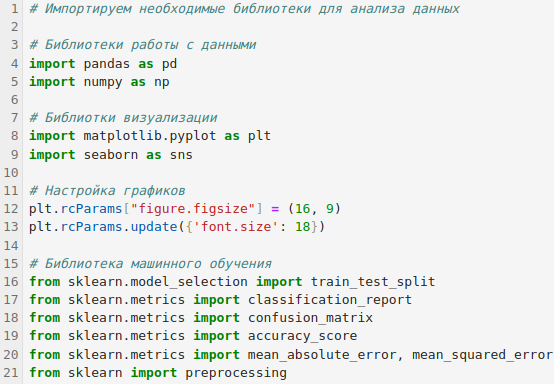
**Ход работы:**

1. Для задачи классификации были выбраны данные о предсказании возраста морского ушка (моллюск) по физическим измерениям. Автор набора данных утверждает, что процесс нахождения возраста моллюска весьма трудоемкий и по этой причине был выпущен датасет, который был призван для разработки модели, которая бы по внешним физическим показателем мог бы с достаточно большой точностью определить возраст ушка. Набор данных содержит 4177 экземпляров. С оригинал датасета можно получить по [ссылке](http://archive.ics.uci.edu/ml/datasets/Abalone).

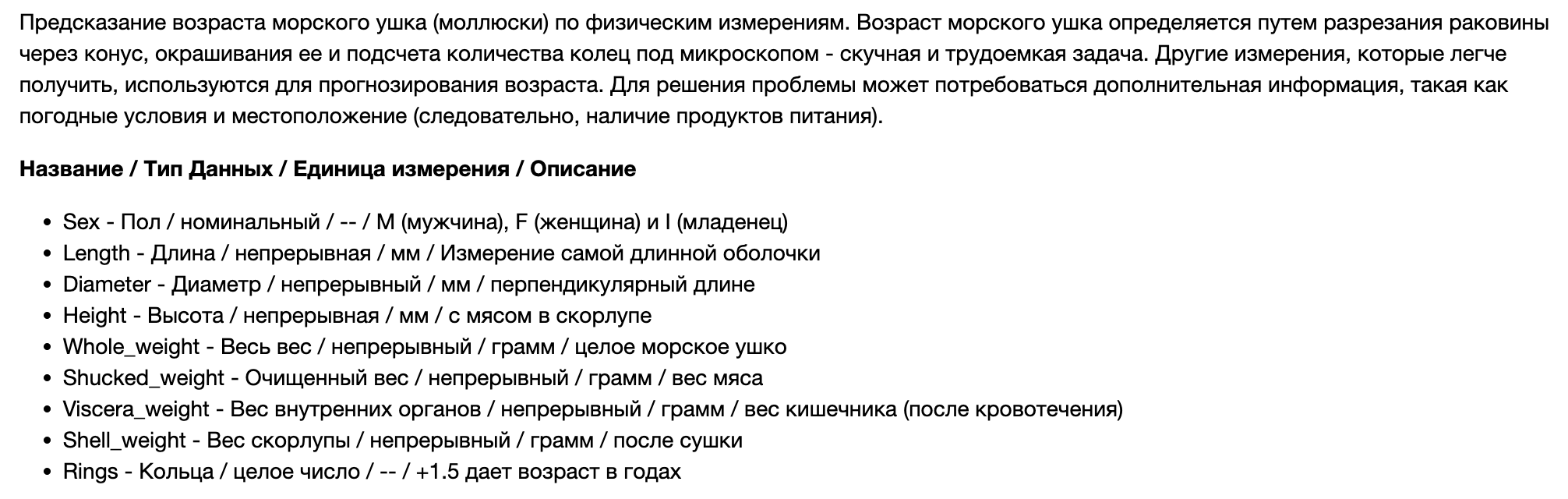
Данные содержат следующие характеристики:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Тип данных** | **Ед. изм.** | **Описание** |
| Sex - Пол | номинативная | – | M (мужчина), F (женщина) и I (младенец) |
| Length - Длина | непрерывная | мм | Измерение самой длинной оболочки |
| Diameter - Диаметр | непрерывный | мм | перпендикулярный длине |
| Height - Высота | непрерывная | мм | с мясом в скорлупе |
| Whole\_weight - Весь вес | непрерывный | грамм | целое морское ушко |
| Shucked\_weight - Очищенный вес | непрерывный | грамм | вес мяса |
| Viscera\_weight - Вес внутренних органов | непрерывный | грамм | вес кишечника (после кровотечения) |
| Shell\_weight - Вес скорлупы | непрерывный | грамм | после сушки |
| Rings - Кольца | целое число | – | +1.5 дает возраст в годах |

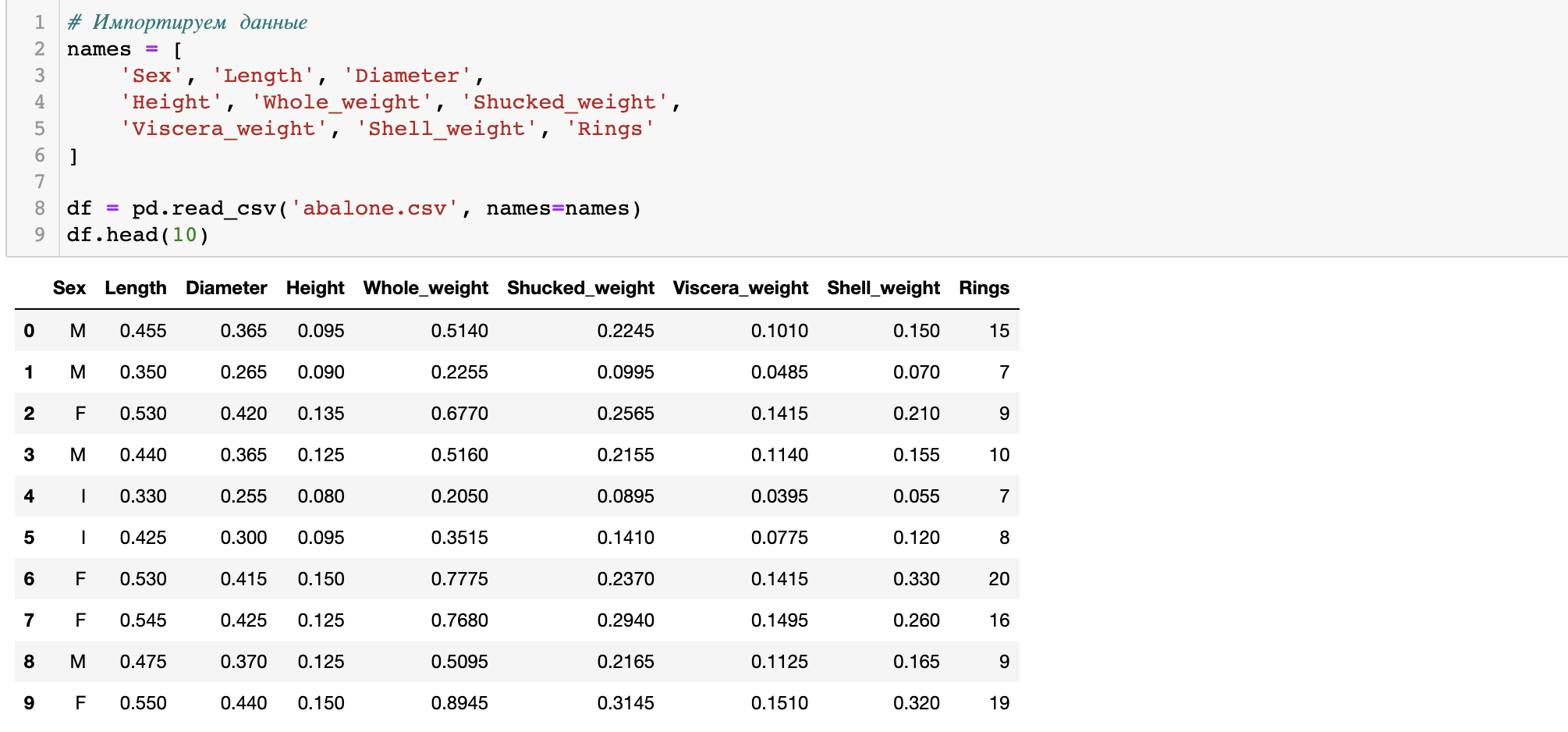
1. Импортируем все необходимые библиотеки:



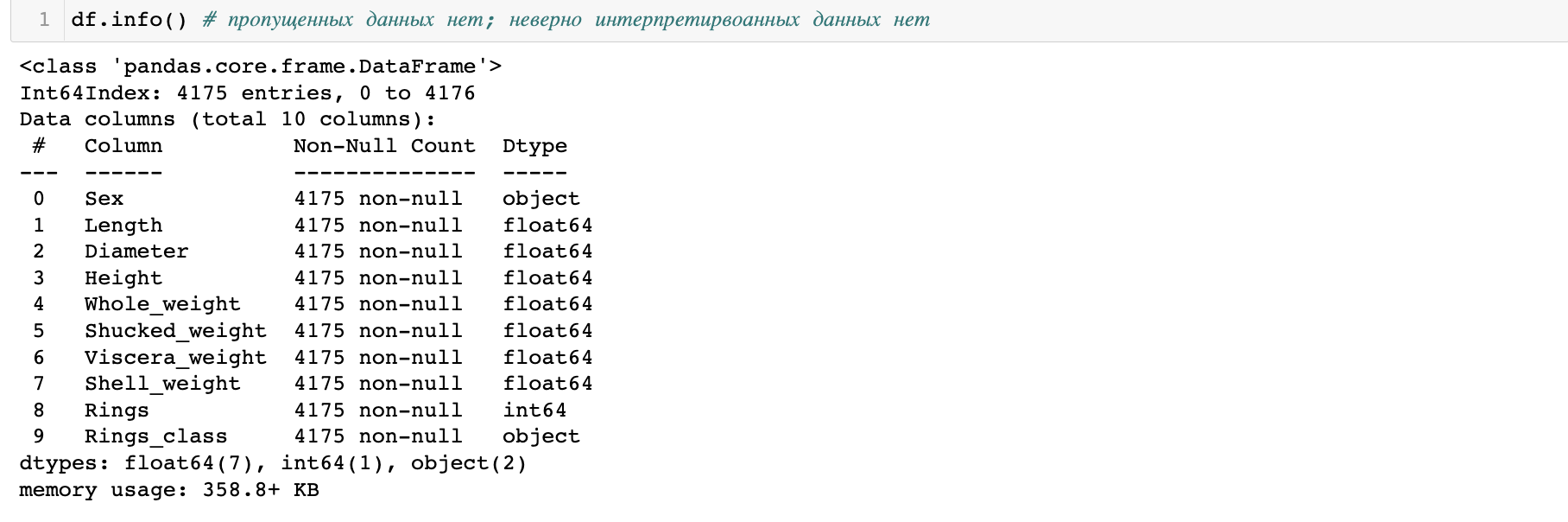
1. Осуществим описание набора данных.



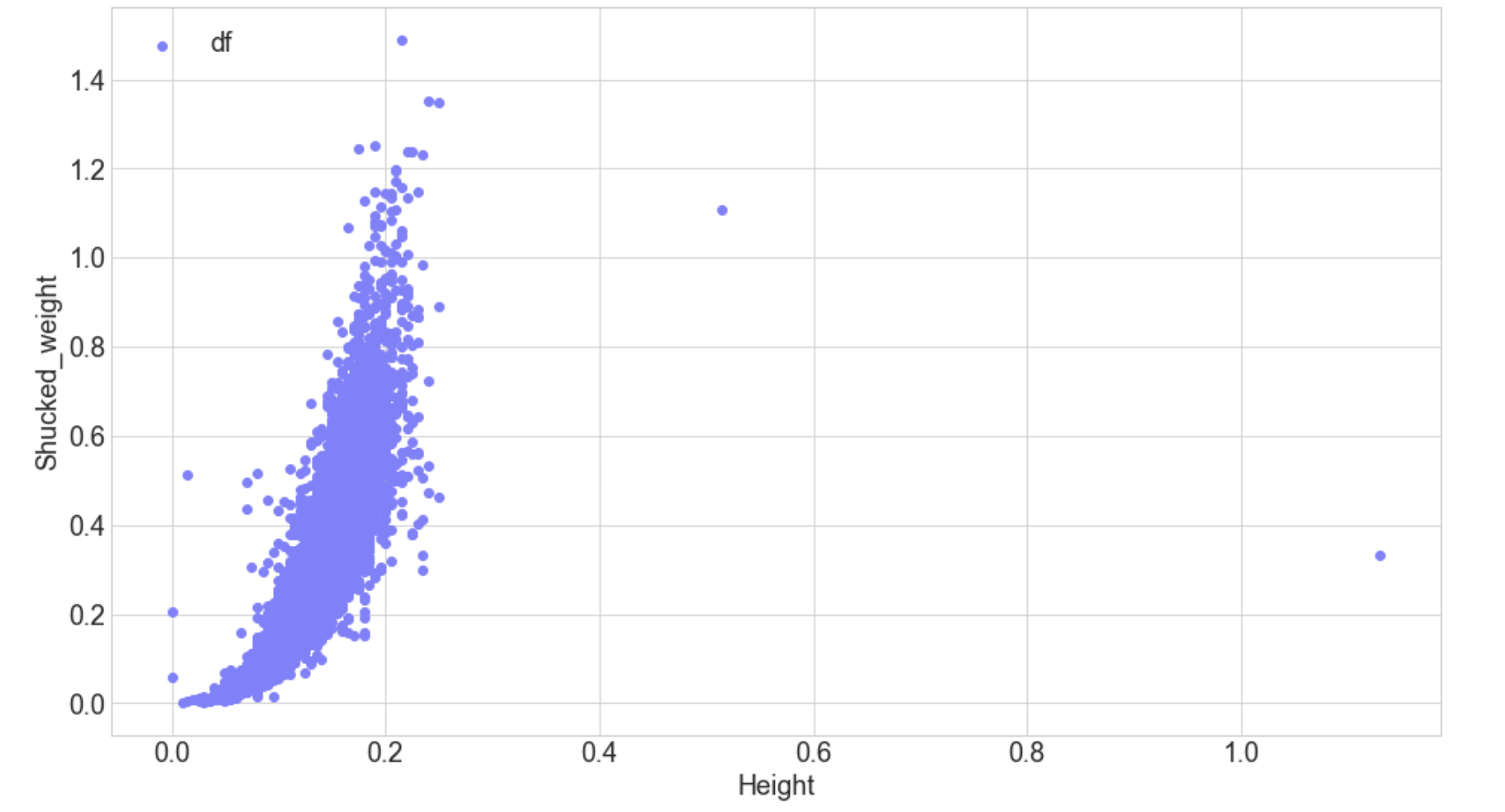
Загрузим наши данные и распечатаем первые 10 наблюдений.



1. Подготовим данные для работы. Определим имеются ли в наших данных пропуски:



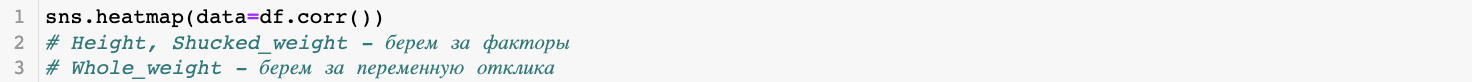
Видно, что пропусков нет и все данные корректно интерпретированы в свои типы. Теперь проверим визуально на выбросы.

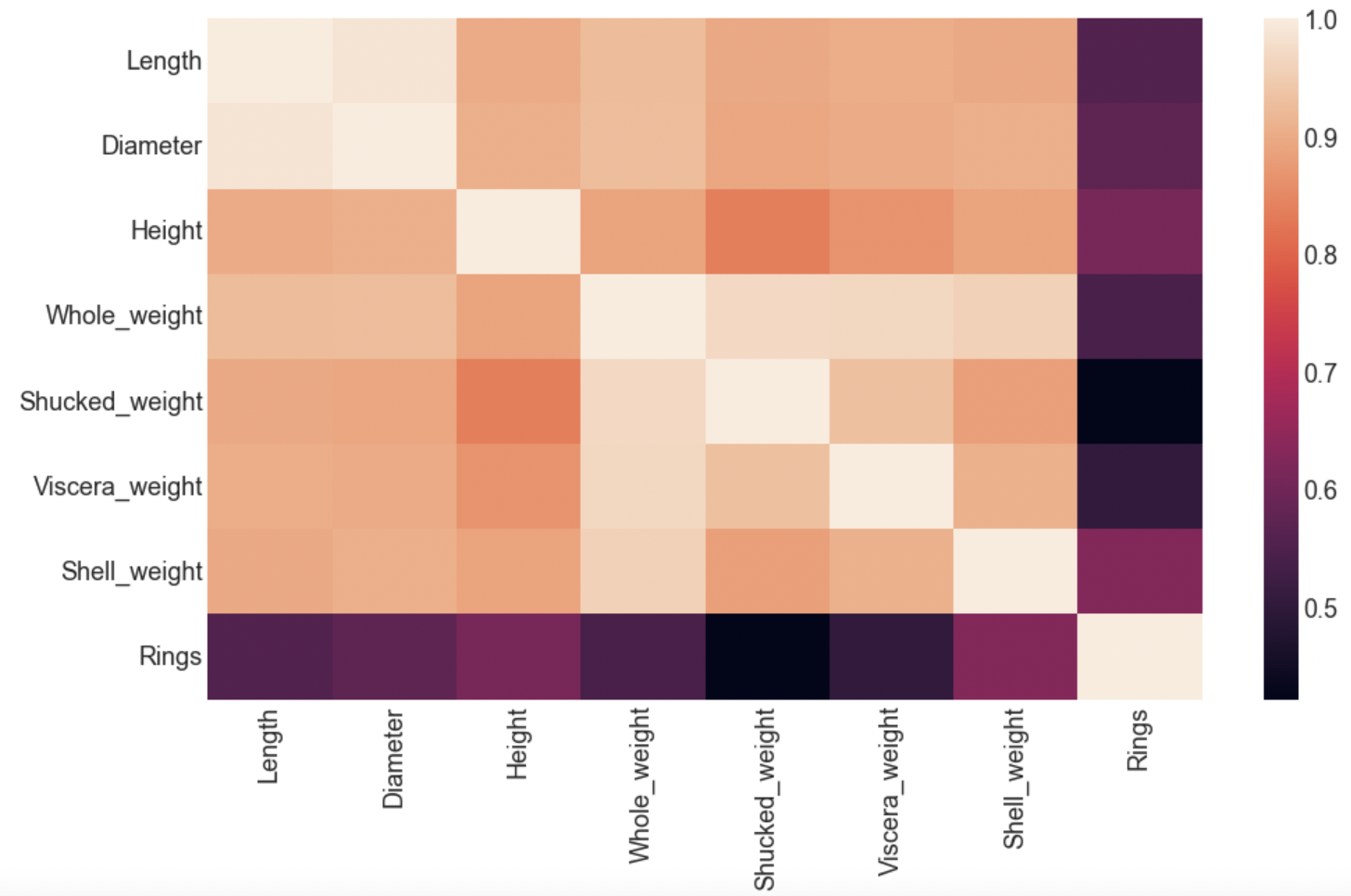
 

Видно, что имеется два выброса. Удалим их.

Рисунок 64

1. Рассчитаем матрицу корреляции.





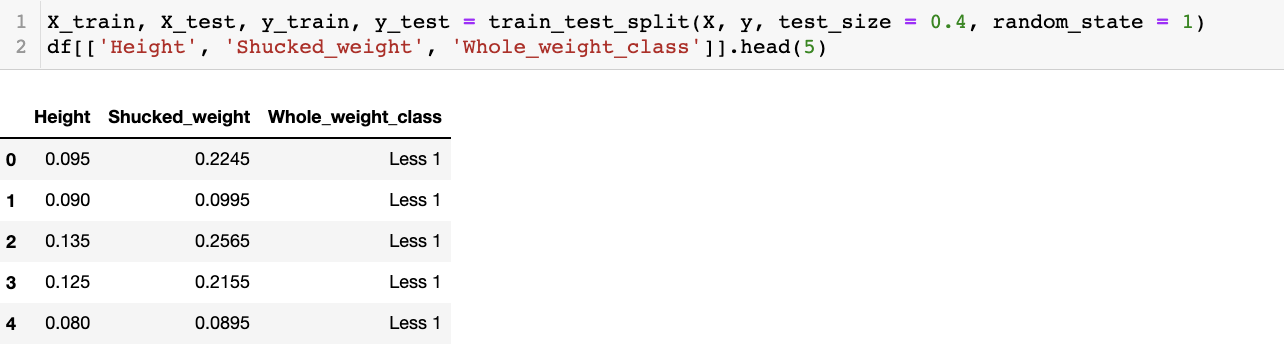
Переменную отклика Whole\_weight разделим на две группы «Over 1» и «Less 1»

Рисунок 12

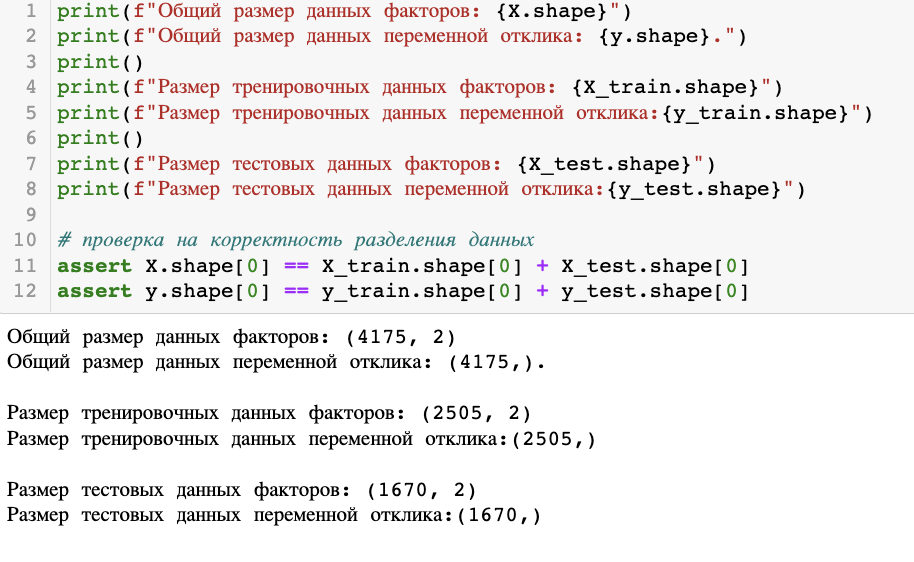
Переменные Height (Высота), Shucked\_weight (Очищенный вес) - менее всего коррелируют, по этой причине мы берем их за факторы, а Whole\_weight\_class - берем за переменную отклика

Рисунок 68

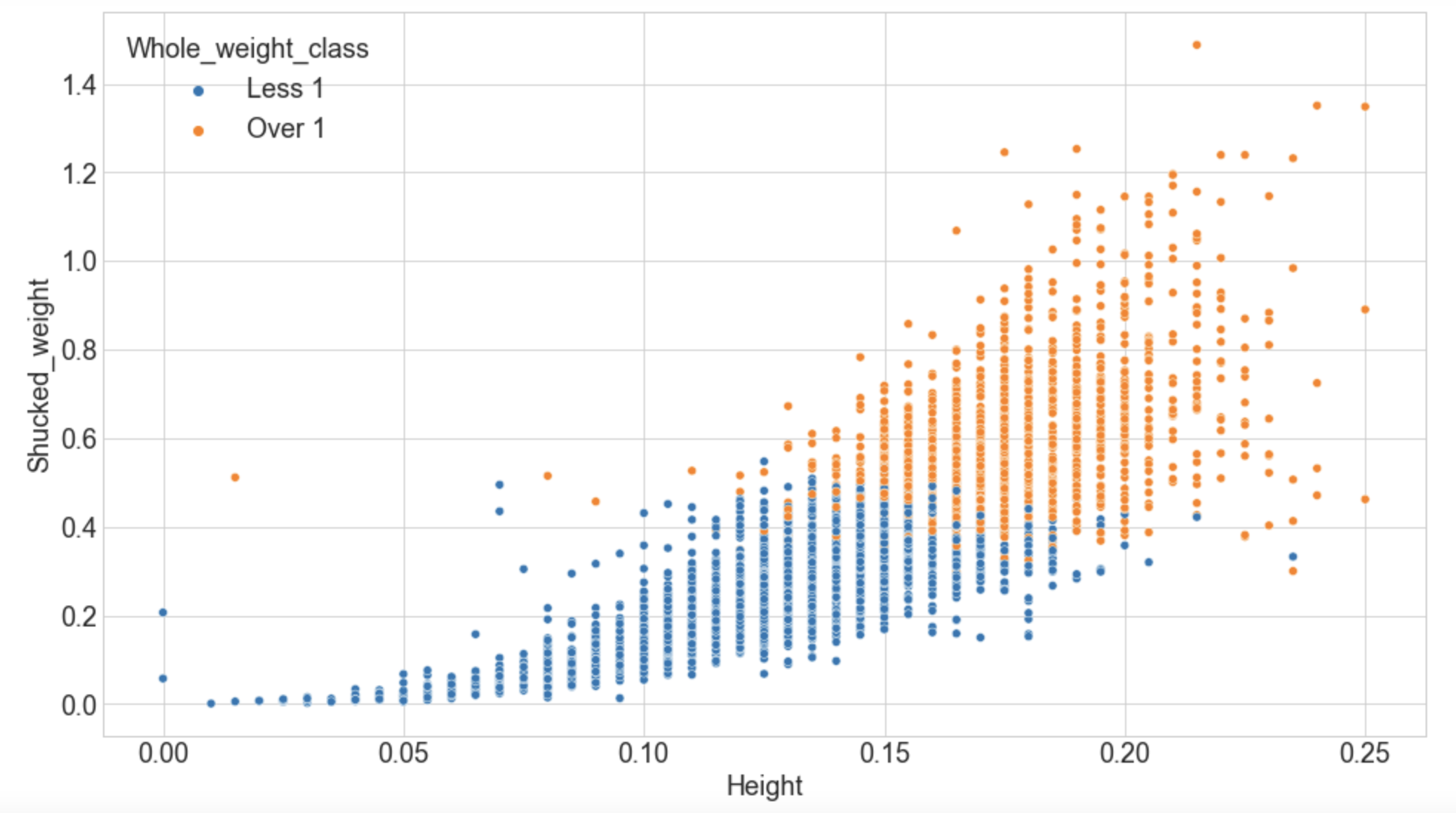
1. Сформируем обучающую и тестовую выборки.



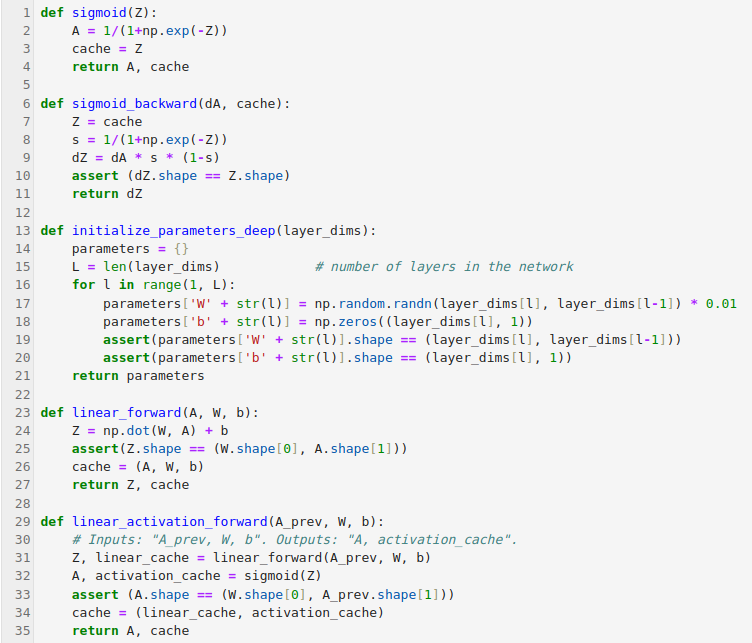
Здесь мы разделили наши данные на обучающую и тестовую и сразу же разделили факторы и отклики.

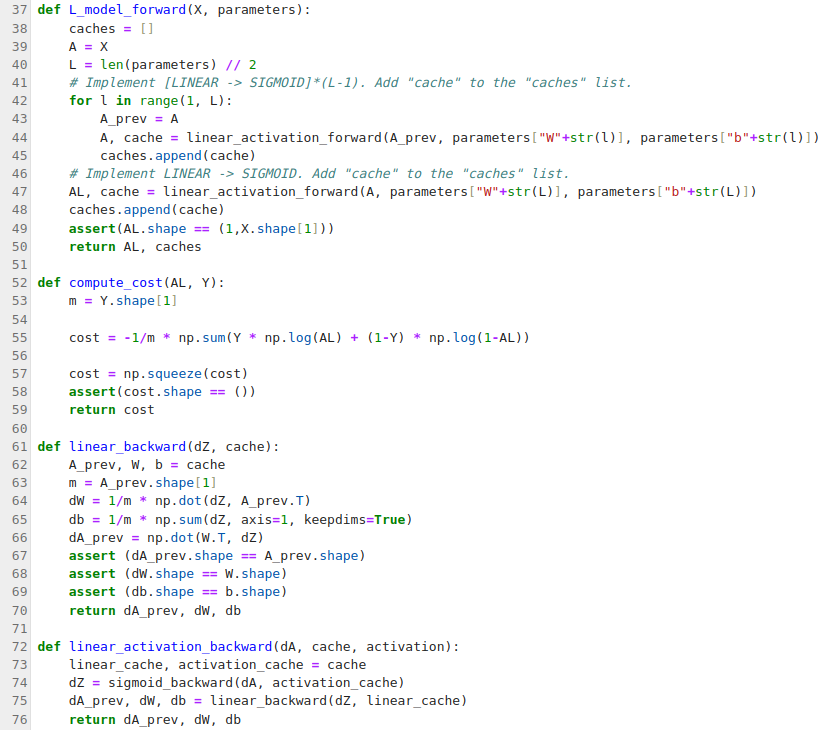


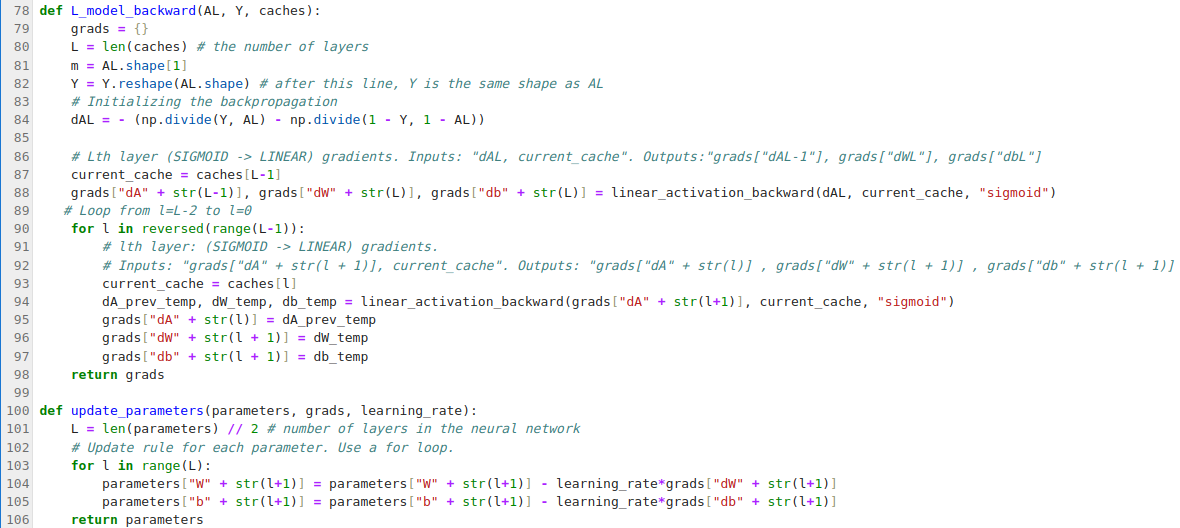
1. Визуализируем наши данные: посмотрим как распределены разные классы.

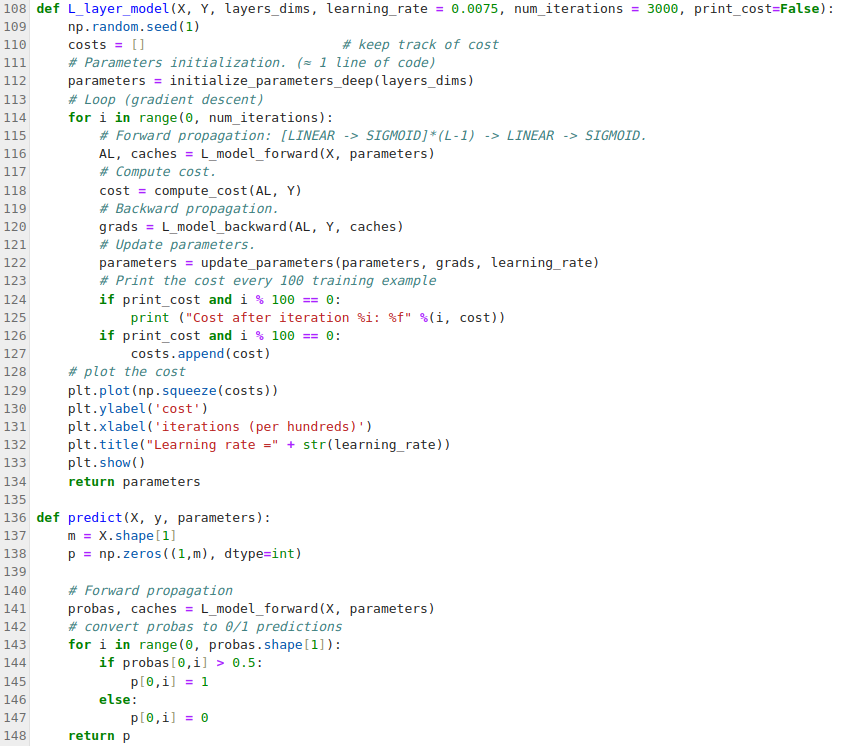
Рисунок 5

1. Проектируем на Python простейшую нейронную сеть - персептрон.

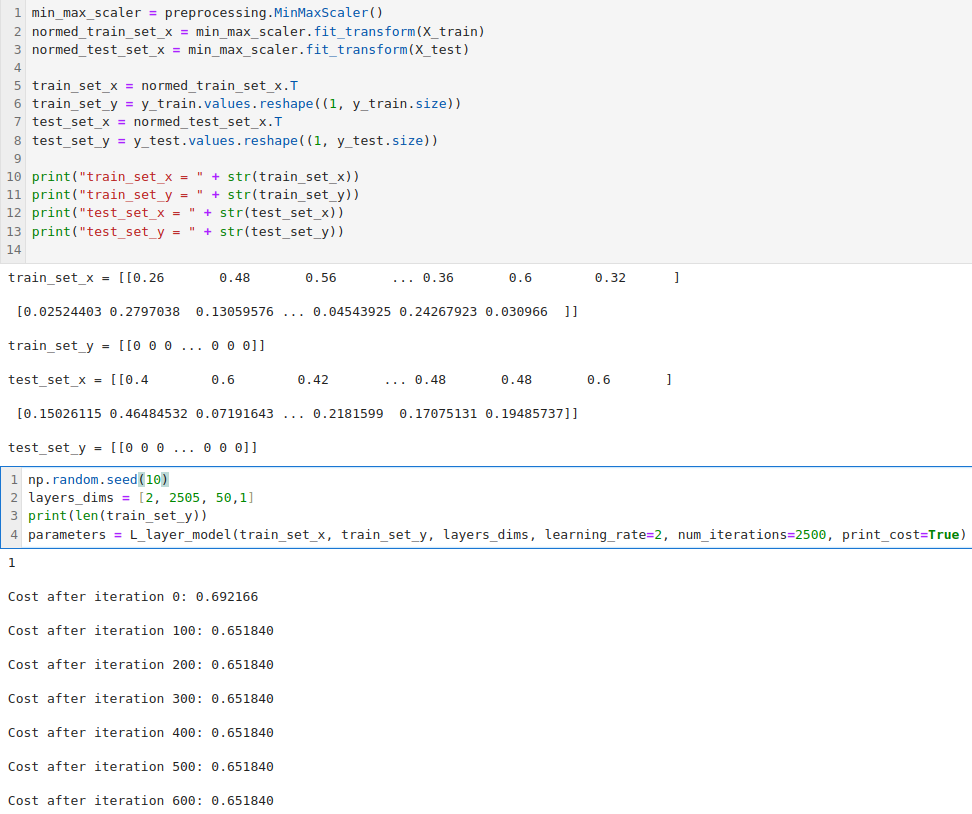




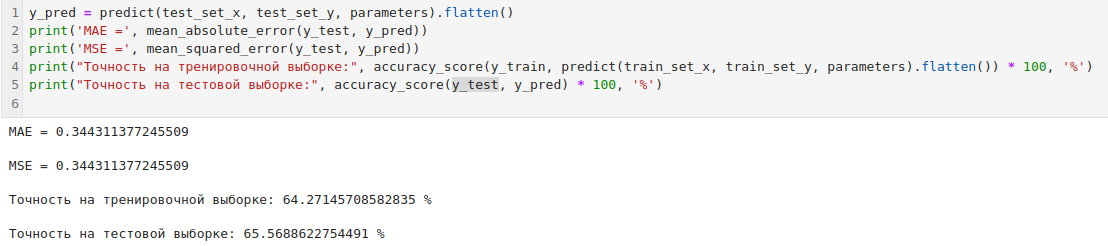


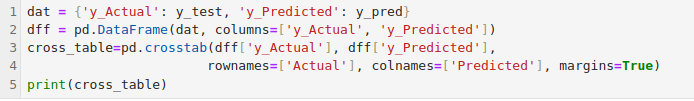


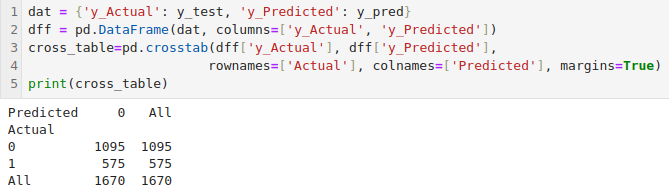
Создадим и обучим персептрон



1. Модель ошиблась в 575 случаях из 1670. Таким образом, точность нашей модели составила 66%. Что ниже, чем все предыдущие методы. Вычислим абсолютные и квадратичные ошибки.

Так же для оценки используем кросс-валидационную таблицу





1. Вывод: Таким образом, мы допускаем, что при наличие информации о очищенном весе и высоте моллюсков персептрон не дает достаточно хорошую оценку о том, легкий моллюск или старый.

**Приложение:**

|  |
| --- |
| # Импортируем необходимые библиотеки для анализа данных  # Библиотеки работы с данными  import pandas as pd  import numpy as np  # Библиотки визуализации  import matplotlib.pyplot as plt  import seaborn as sns  # Настройка графиков  plt.rcParams["figure.figsize"] = (16, 9)  plt.rcParams.update({'font.size': 18})  # Библиотека машинного обучения  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  from sklearn.metrics import accuracy\_score  from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  from sklearn import preprocessing |
| # Импортируем данные  names = [  'Sex', 'Length', 'Diameter',  'Height', 'Whole\_weight', 'Shucked\_weight',  'Viscera\_weight', 'Shell\_weight', 'Rings'  ]  df = pd.read\_csv('abalone.csv', names=names)  df.head(10) |
| df['Whole\_weight\_class'] = np.where(df.Whole\_weight > 1, 1, 0) |
| df.head(5) |
| df.info() # пропущенных данных нет; неверно интерпретирвоанных данных нет |
| fig, ax = plt.subplots()  ax.scatter(df.Height, df.Shucked\_weight, label='df', c='#8080ff')  ax.set\_xlabel('Height')  ax.set\_ylabel('Shucked\_weight')  plt.legend(loc='upper left') |
| df = df.drop(index=list(df.query("Height>0.4").index)) |
| sns.heatmap(data=df.corr())  # Height, Shucked\_weight - менее всего коррелируют  # Height, Shucked\_weight - берем за факторы |
| X, y = df[['Height', ‘Shucked\_weight']].values, df['Whole\_weight\_class'] |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.4, random\_state = 1)  df[['Height', 'Shucked\_weight', 'Whole\_weight\_class']].head(5) |
| print(f"Общий размер данных факторов: {X.shape}")  print(f"Общий размер данных переменной отклика: {y.shape}.")  print()  print(f"Размер тренировочных данных факторов: {X\_train.shape}")  print(f"Размер тренировочных данных переменной отклика:{y\_train.shape}")  print()  print(f"Размер тестовых данных факторов: {X\_test.shape}")  print(f"Размер тестовых данных переменной отклика:{y\_test.shape}")  # проверка на корректность разделения данных  assert X.shape[0] == X\_train.shape[0] + X\_test.shape[0]  assert y.shape[0] == y\_train.shape[0] + y\_test.shape[0] |
| sns.scatterplot(data=df, x='Height', y='Shucked\_weight', hue='Whole\_weight\_class')def sigmoid(Z):  A = 1/(1+np.exp(-Z))  cache = Z  return A, cache  def sigmoid\_backward(dA, cache):  Z = cache  s = 1/(1+np.exp(-Z))  dZ = dA \* s \* (1-s)  assert (dZ.shape == Z.shape)  return dZ  def initialize\_parameters\_deep(layer\_dims):  parameters = {}  L = len(layer\_dims) # number of layers in the network  for l in range(1, L):  parameters['W' + str(l)] = np.random.randn(layer\_dims[l], layer\_dims[l-1]) \* 0.01  parameters['b' + str(l)] = np.zeros((layer\_dims[l], 1))  assert(parameters['W' + str(l)].shape == (layer\_dims[l], layer\_dims[l-1]))  assert(parameters['b' + str(l)].shape == (layer\_dims[l], 1))  return parameters  def linear\_forward(A, W, b):  Z = np.dot(W, A) + b  assert(Z.shape == (W.shape[0], A.shape[1]))  cache = (A, W, b)  return Z, cache  def linear\_activation\_forward(A\_prev, W, b):  # Inputs: "A\_prev, W, b". Outputs: "A, activation\_cache".  Z, linear\_cache = linear\_forward(A\_prev, W, b)  A, activation\_cache = sigmoid(Z)  assert (A.shape == (W.shape[0], A\_prev.shape[1]))  cache = (linear\_cache, activation\_cache)  return A, cache  def L\_model\_forward(X, parameters):  caches = []  A = X  L = len(parameters) // 2  # Implement [LINEAR -> SIGMOID]\*(L-1). Add "cache" to the "caches" list.  for l in range(1, L):  A\_prev = A  A, cache = linear\_activation\_forward(A\_prev, parameters["W"+str(l)], parameters["b"+str(l)])  caches.append(cache)  # Implement LINEAR -> SIGMOID. Add "cache" to the "caches" list.  AL, cache = linear\_activation\_forward(A, parameters["W"+str(L)], parameters["b"+str(L)])  caches.append(cache)  assert(AL.shape == (1,X.shape[1]))  return AL, caches  def compute\_cost(AL, Y):  m = Y.shape[1]  cost = -1/m \* np.sum(Y \* np.log(AL) + (1-Y) \* np.log(1-AL))  cost = np.squeeze(cost)  assert(cost.shape == ())  return cost  def linear\_backward(dZ, cache):  A\_prev, W, b = cache  m = A\_prev.shape[1]  dW = 1/m \* np.dot(dZ, A\_prev.T)  db = 1/m \* np.sum(dZ, axis=1, keepdims=True)  dA\_prev = np.dot(W.T, dZ)  assert (dA\_prev.shape == A\_prev.shape)  assert (dW.shape == W.shape)  assert (db.shape == b.shape)  return dA\_prev, dW, db  def linear\_activation\_backward(dA, cache, activation):  linear\_cache, activation\_cache = cache  dZ = sigmoid\_backward(dA, activation\_cache)  dA\_prev, dW, db = linear\_backward(dZ, linear\_cache)  return dA\_prev, dW, db  def L\_model\_backward(AL, Y, caches):  grads = {}  L = len(caches) # the number of layers  m = AL.shape[1]  Y = Y.reshape(AL.shape) # after this line, Y is the same shape as AL  # Initializing the backpropagation  dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))    # Lth layer (SIGMOID -> LINEAR) gradients. Inputs: "dAL, current\_cache". Outputs:"grads["dAL-1"], grads["dWL"], grads["dbL"]  current\_cache = caches[L-1]  grads["dA" + str(L-1)], grads["dW" + str(L)], grads["db" + str(L)] = linear\_activation\_backward(dAL, current\_cache, "sigmoid")  # Loop from l=L-2 to l=0  for l in reversed(range(L-1)):  # lth layer: (SIGMOID -> LINEAR) gradients.  # Inputs: "grads["dA" + str(l + 1)], current\_cache". Outputs: "grads["dA" + str(l)] , grads["dW" + str(l + 1)] , grads["db" + str(l + 1)]  current\_cache = caches[l]  dA\_prev\_temp, dW\_temp, db\_temp = linear\_activation\_backward(grads["dA" + str(l+1)], current\_cache, "sigmoid")  grads["dA" + str(l)] = dA\_prev\_temp  grads["dW" + str(l + 1)] = dW\_temp  grads["db" + str(l + 1)] = db\_temp  return grads  def update\_parameters(parameters, grads, learning\_rate):  L = len(parameters) // 2 # number of layers in the neural network  # Update rule for each parameter. Use a for loop.  for l in range(L):  parameters["W" + str(l+1)] = parameters["W" + str(l+1)] - learning\_rate\*grads["dW" + str(l+1)]  parameters["b" + str(l+1)] = parameters["b" + str(l+1)] - learning\_rate\*grads["db" + str(l+1)]  return parameters  def L\_layer\_model(X, Y, layers\_dims, learning\_rate = 0.0075, num\_iterations = 3000, print\_cost=False):  np.random.seed(1)  costs = [] # keep track of cost  # Parameters initialization. (≈ 1 line of code)  parameters = initialize\_parameters\_deep(layers\_dims)  # Loop (gradient descent)  for i in range(0, num\_iterations):  # Forward propagation: [LINEAR -> SIGMOID]\*(L-1) -> LINEAR -> SIGMOID.  AL, caches = L\_model\_forward(X, parameters)  # Compute cost.  cost = compute\_cost(AL, Y)  # Backward propagation.  grads = L\_model\_backward(AL, Y, caches)  # Update parameters.  parameters = update\_parameters(parameters, grads, learning\_rate)  # Print the cost every 100 training example  if print\_cost and i % 100 == 0:  print ("Cost after iteration %i: %f" %(i, cost))  if print\_cost and i % 100 == 0:  costs.append(cost)  # plot the cost  plt.plot(np.squeeze(costs))  plt.ylabel('cost')  plt.xlabel('iterations (per hundreds)')  plt.title("Learning rate =" + str(learning\_rate))  plt.show()  return parameters  def predict(X, y, parameters):  m = X.shape[1]  p = np.zeros((1,m), dtype=int)    # Forward propagation  probas, caches = L\_model\_forward(X, parameters)  # convert probas to 0/1 predictions  for i in range(0, probas.shape[1]):  if probas[0,i] > 0.5:  p[0,i] = 1  else:  p[0,i] = 0  return p |
| min\_max\_scaler = preprocessing.MinMaxScaler()  normed\_train\_set\_x = min\_max\_scaler.fit\_transform(X\_train)  normed\_test\_set\_x = min\_max\_scaler.fit\_transform(X\_test)  train\_set\_x = normed\_train\_set\_x.T  train\_set\_y = y\_train.values.reshape((1, y\_train.size))  test\_set\_x = normed\_test\_set\_x.T  test\_set\_y = y\_test.values.reshape((1, y\_test.size))  print("train\_set\_x = " + str(train\_set\_x))  print("train\_set\_y = " + str(train\_set\_y))  print("test\_set\_x = " + str(test\_set\_x))  print("test\_set\_y = " + str(test\_set\_y)) |
| np.random.seed(10)  layers\_dims = [2, 2505, 50,1]  parameters = L\_layer\_model(train\_set\_x, train\_set\_y, layers\_dims, learning\_rate=2, num\_iterations=2500, print\_cost=True) |
| y\_pred = predict(test\_set\_x, test\_set\_y, parameters).flatten()  print('MAE =', mean\_absolute\_error(y\_test, y\_pred))  print('MSE =', mean\_squared\_error(y\_test, y\_pred))  print("Точность на тренировочной выборке:", accuracy\_score(y\_train, predict(train\_set\_x, train\_set\_y, parameters).flatten()) \* 100, '%')  print("Точность на тестовой выборке:", accuracy\_score(y\_test, y\_pred) \* 100, '%') |
| dat = {'y\_Actual': y\_test, 'y\_Predicted': y\_pred}  dff = pd.DataFrame(dat, columns=['y\_Actual', 'y\_Predicted'])  cross\_table=pd.crosstab(dff['y\_Actual'], dff['y\_Predicted'],  rownames=['Actual'], colnames=['Predicted'], margins=True)  print(cross\_table) |