|  |
| --- |
| МИНОБРНАУКИ РОССИИ  Федеральное государственное  бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий

Кафедра корпоративных информационных систем

|  |  |  |
| --- | --- | --- |
| **ОТЧЕТ**  **по лабораторной работе №4** | | |
| **по дисциплине** | | |
| **«Структуры и алгоритмы обработки данных»**  **Тема лабораторной работы: «**Корреляционный и регрессионный анализ**»** | | |
| Студент группы | ИКБО-08-18 | Валяев Д.А. |
| Принял | ассистент кафедры КИС | Исаева.И.А. |
|  |  |  |
| Выполнено | «17» мая 2020 г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись студента)* |
| Зачтено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись преподавателя)* |

Москва 2020

1. **Часть №1**
   1. **Теоретическое часть**

Основная работа направлена на взаимодействие с модулем Pandas. Данный пакет является мощным инструментом для анализа данных, он позволяет строить сводные таблицы, выполнять группировки, предоставляет удобный доступ к табличным данным.

Основные структуры для хранения данных в Pandas – Series и DataFrame.

Series – это проиндексированный одномерный массив значений. Он похож на простой словарь типа dict, где имя элемента будет соответствовать индексу, а значение – значению записи.

DataFrame – это проиндексированный многомерный массив значений, соответственно каждый столбец DataFrame, является структурой Series.

Seaborn – библиотека, используемая для визуализации данных, построенная на основе Matplotlib.

Seaborn решает две основные проблемы Matplotlib, а именно проблему стандартных параметров Matplotlib и работу с фреймами данных.

Seaborn построен поверх базовой библиотеки визуализации Python Matplotlib. Он должен служить дополнением, а не заменой. Тем не менее, Seaborn имеет некоторые очень важные функции.

* Встроенные темы для оформления графики matplotlib
* Визуализация одномерных и двумерных данных
* Подгонка и визуализация моделей линейной регрессии
* Построение статистических данных временных рядов
* Seaborn хорошо работает со структурами данных NumPy и Pandas
* Он поставляется со встроенными темами для оформления графики Matplotlib
  1. **Практическая часть**

# # ЧАСТЬ 1

# In[1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# ## Получение всех данных из csv, вывод (данные согласно варианту)

# In[2]:

print("Задание №1 Все данные из csv")

print("----------------------------------------------------------------------")

csv\_path = "F:/breast-cancer-wisconsin.csv"

lst = pd.read\_csv(csv\_path, sep=',')

print(lst)

# ## Получение 10 первых строк, 10 последних строк

# In[3]:

print("Задание №2 Первые и последние 10 строк")

print("----------------------------------------------------------------------")

print(lst.head(10))

print(lst.tail(10))

# ## Создание таблицы для всех числовых значений со столбцами: среднее значение (математическое ожидание), медиана, мода, минимальное значение, макcимальное значение, размах, дисперсия, среднеквадратичное отклонение)

# In[5]:

print("Задание №3 Создание таблицы для всех числовых значений со столбцами: математическое ожидание, медиана, мода, минимальное значение, макcимальное значение, размах, дисперсия, среднеквадратичное отклонение)")

print("----------------------------------------------------------------------")

newlist = {}

def created\_full\_columns\_list(columns\_list, lst):

columns = ['название', 'Мат. ожидание', 'Медиана', 'Мода', 'Мин. значение', 'Макс. значение',

'Размах', 'Дисперсия', 'Ср. кв. отклонение']

x = lst.columns.tolist()

for col in columns:

columns\_list[col] = []

for col in x:

columns\_list['название'].append(col)

def Average(dict, lst):

x = lst.columns.tolist()

for col in x:

dict['Мат. ожидание'].append(lst[col].mean())

def Mediana(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_mass = lst[col].tolist()

new\_mass.sort()

dict['Медиана'].append(new\_mass[len(new\_mass)//2])

def moda(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_list = lst[col].tolist()

new\_mass = {}

for per in new\_list:

if per in new\_mass:

new\_mass[per] += 1

else:

new\_mass[per] = 0

max = -1

maxn = None

for l in new\_mass.keys():

if new\_mass[l] > max:

max = new\_mass[l]

maxn = l

dict['Мода'].append(maxn)

def Min(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_mass = lst[col].tolist()

new\_mass.sort()

dict['Мин. значение'].append(new\_mass[0])

def Max(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_mass = lst[col].tolist()

new\_mass.sort()

dict['Макс. значение'].append(new\_mass[len(new\_mass)-1])

def Scope(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_mass = lst[col].tolist()

new\_mass.sort()

dict['Размах'].append(new\_mass[len(new\_mass)-1]-new\_mass[0])

def Disper(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_mass = lst[col].tolist()

s = 0

for i in new\_mass:

s += (i-lst[col].mean())\*(i - lst[col].mean())

s /= lst[col].count()

dict['Дисперсия'].append(s)

def sko(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_mass = lst[col].tolist()

s = 0

for i in new\_mass:

s += (i-lst[col].mean())\*(i - lst[col].mean())

s /= lst[col].count()

import math

math.sqrt(s)

dict['Ср. кв. отклонение'].append(s)

def n(dict, lst):

x = lst.columns.tolist()

for col in x:

new\_mass = lst[col].tolist()

s = 0

for i in new\_mass:

s += (i-lst[col].mean())\*(i - lst[col].mean())

s /= lst[col].count()

import math

math.sqrt(s)

dict['Мода'].append(s)

created\_full\_columns\_list(newlist, lst)

Average(newlist, lst)

Mediana(newlist, lst)

moda(newlist, lst)

Min(newlist, lst)

Max(newlist, lst)

Scope(newlist, lst)

Disper(newlist, lst)

sko(newlist, lst)

new\_df = pd.DataFrame(newlist)

print(new\_df)

# In[6]:

print("гистограмма 1")

x = np.arange(0, lst['perimeter'].count())

y = lst['perimeter'].tolist()

fig, ax = plt.subplots()

ax.bar(x, y)

ax.set\_facecolor('seashell')

fig.set\_facecolor('floralwhite')

fig.set\_figwidth(12)

fig.set\_figheight(6)

plt.show()

# In[7]:

print("гистограмма 2")

x = np.arange(0, lst['area'].count())

y = lst['area'].tolist()

fig, ax = plt.subplots()

ax.bar(x, y)

ax.set\_facecolor('seashell')

fig.set\_facecolor('floralwhite')

fig.set\_figwidth(12)

fig.set\_figheight(6)

plt.show()

# ## Вставить две гистограммы рядом для двух признаков

# In[8]:

print("Задание №4 Две гистограммы рядом для двух признаков")

print("----------------------------------------------------------------------")

x = np.arange(0, lst['perimeter'].count())

y1 = lst['perimeter'].tolist()

y2 = lst['area'].tolist()

gridsize = (1, 2)

fig = plt.figure(figsize=(20, 8))

ax2 = plt.subplot2grid(gridsize, (0, 0))

ax3 = plt.subplot2grid(gridsize, (0, 1))

ax2.bar(x,y1)

ax3.bar(x,y2)

plt.show()

# ## Вставить изображение точек на плоскости для этих двух признаков для различных категорий

# In[9]:

print("Задание №5 Вставить изображение точек на плоскости для этих двух признаков")

print("----------------------------------------------------------------------")

x = np.arange(0, lst['perimeter'].count())

y1 = lst['perimeter'].tolist()

y2 = lst['area'].tolist()

fig, ax = plt.subplots()

ax.scatter(x, y1, c='orange')

ax.scatter(x + 1, y2, c='blue')

ax.set\_facecolor('white')

fig.set\_figwidth(8)

fig.set\_figheight(8)

plt.show()

# ## Создать таблицу из двух столбцов с названиями "Первый столбец", "Второй столбец" с N данными, распределенными согласно нормальному закону с указанными математическим ожиданием M и среднеквадатичным отклонением отклонением S (N, S, M - из варианта)

# In[10]:

print("Задание №6 Создать таблицу из двух столбцов с названиями Первый столбец, Второй столбец с N данными, распределенными согласно нормальному закону")

print("-------------------------------------------------------------------")

size = 120

mean = -1

sigma = 1

n\_lst = pd.DataFrame(columns = ['Первый столбец', 'Второй столбец'] )

n\_lst['Первый столбец'] = np.random.normal(mean,sigma,size)

n\_lst['Второй столбец'] = np.random.normal(mean,sigma,size)

print(n\_lst)

# ## Вставить две гистограммы рядом для двух столбцов

# In[11]:

print("Задание №7 Две гистограммы рядом для двух признаков")

print("----------------------------------------------------------------------")

x = np.arange(0, n\_lst['Первый столбец'].count())

y1 = n\_lst['Первый столбец'].tolist()

y2 = n\_lst['Второй столбец'].tolist()

gridsize = (1, 2)

fig = plt.figure(figsize=(20, 8))

ax2 = plt.subplot2grid(gridsize, (0, 0))

ax3 = plt.subplot2grid(gridsize, (0, 1))

ax2.bar(x,y1)

ax3.bar(x,y2)

plt.show()

# ## Вставить изображение точек на плоскости для этих двух столбцов

# In[12]:

print("Задание №8 Вставить изображение точек на плоскости для этих двух столбцов")

print("-------------------------------------------------------------------")

x = np.arange(0, size)

y1 = n\_lst['Первый столбец'].tolist()

y2 = n\_lst['Второй столбец'].tolist()

fig, ax = plt.subplots()

ax.scatter(x, y1, c='r')

ax.scatter(x + 1, y2, c='blue')

ax.set\_facecolor('white')

fig.set\_figwidth(8)

fig.set\_figheight(8)

plt.show()

1. **Часть №2**
   1. **Теоретическое часть**

Pandas предоставляет широкий выбор источников данных, например:

В данной работе мы работаем с текстовым

* SQL
* Текстовые файлы
* Excel файлы
* HTML

В данной работе мы взаимодействуем с текстовым файлом. Для этого используется функция read\_csv(). Она позволяет так же задать и разделитель.

Функция insert() добавляет столбец в набор данных.

Для получения списка столбцов можно привести набор данных к списку.

Для добавления записи, нужно выполнить функцию append().

Для соединения двух наборов данных существует функция merge().

Функция pivot\_table() строить сводную таблицу.

В модуле Seaborn для построения гистограммы можно воспользоваться функцией distplot().

Объединить две гистограммы можно при помощи subplots().

Изображение точек на плоскости можно реализовать при помощи lmplot(), передав туда названия столбцов X и Y, набор данных, столбец, данные которого будут отображаться и False параметру fit\_reg, чтобы не строить прямую на графике.

* 1. **Практическая часть**

# # ЧАСТЬ 2

# In[24]:

import numpy as np

from scipy import stats

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# ## Получение всех данных из csv, вывод

# In[25]:

print("Задание №1 Все данные из csv")

print("----------------------------------------------------------------------")

csv\_path = "F:/breast-cancer-wisconsin.csv"

lst = pd.read\_csv(csv\_path, sep=',')

print(lst)

# ## Создание нового датафрейма только с нужными для исследования признаками

# In[26]:

print("Задание №2 Создание нового датафрейма только с нужными для исследования признаками")

print("----------------------------------------------------------------------")

new\_lst = lst[['perimeter', 'area']]

print(new\_lst)

# In[27]:

sns.lmplot(x = "perimeter", y = "area", data = new\_lst)

plt.show()

# ## Определение коэффициента корреляции Пирсона для этих двух признаков вручную и с применением существующего метода из numpy/scipy

# In[28]:

print("Задание №3 Определение коэффициента корреляции Пирсона")

print("----------------------------------------------------------------------")

def pirson(lst):

aver1 = lst['perimeter'].mean()

aver2 = lst['area'].mean()

mas1 = lst['perimeter'].tolist()

mas2 = lst['area'].tolist()

s1, s2, s3 = 0, 0, 0

for x, y in zip(mas1, mas2):

s1 += (x-aver1)\*(y-aver2)

s2 += (x - aver1) \* (x - aver1)

s3 += (y - aver2) \* (y - aver2)

import math

s2 = math.sqrt(s2)

s3 = math.sqrt(s3)

s = s1/(s2\*s3)

return s

print("коэффициента корреляции Пирсона в ручную: ", pirson(new\_lst))

aver1 = new\_lst['perimeter'].tolist()

aver2 = new\_lst['area'].tolist()

x = np.corrcoef(np.vstack((aver1, aver2)))

otv = x[0][1]

print("коэффициента корреляции Пирсона с помощью np.corrcoef(): ",otv)

# In[15]:

np.corrcoef(x, y)[0, 1]

# ## Вычисление p-value вручную и с применением существующего метода из numpy/scipy

# In[29]:

print("Задание №4 Вычисление p-value")

print("----------------------------------------------------------------------")

print("по формуле: ", stats.ttest\_ind(aver1,aver2).pvalue)

pval = stats.ttest\_ind(aver1,aver2).pvalue

import math

def d\_ttest(probability, X, Y):

mean1, mean2 = np.mean(X), np.mean(Y)

n = len(X)

q1 = sum([(X[i]-Y[i])\*\*2 for i in range(n)])

q2 = sum([X[i]-Y[i] for i in range(n)])

sd = math.sqrt((q1 - (q2\*\*2 / n)) / (n - 1))

sed = sd / math.sqrt(n)

ts = (mean1 - mean2) / sed

lst = n - 1

cv = stats.t.ppf(1.0 - probability, lst)

p = (1.0 - stats.t.cdf(abs(ts), lst)) \* 2.0

return p

print("Подсчет вручную: ", d\_ttest(0.05,new\_lst['perimeter'], new\_lst['area']))

# ## Описание корреляции признаков в выборке и оценка возможности отклонить нулевую гипотезу о том, что в генеральной совокупности коэффициент корреляции равен нулю

# In[30]:

print("Описание корреляции признаков в выборке и оценка возможности отклонить нулевую гипотезу о том, что в генеральной совокупности коэффициент корреляции равен нулю")

print("----------------------------------------------------------------------")

if pval < 0.05 :

print('Гипотезу можно отвергнуть.')

else:

print('Гипотезу нельзя отвергнуть.')

# ## Моделирование независимых случайных векторов X, Y имеющих нормальное распределение с заданным математическим ожиданием и корреляционной матрицей. Данные добавлены в таблицу из двух столбцов с названиями "X", "Y"

# In[31]:

print("Задание №5 Моделирование независимых случайных векторов X, Y имеющих нормальное распределение с заданным математическим ожиданием и корреляционной матрицей")

print("----------------------------------------------------------------------")

mean =[-1,1]

cov = [[1,0.1],[0.1,2]]

n = 1000

x\_n, y\_n = np.random.multivariate\_normal(mean, cov, n).T

table\_n = pd.DataFrame({'X': x\_n, 'Y': y\_n})

print(table\_n)

# ## Изображение точек на плоскости для этих двух признаков с регрессионной прямой

# In[32]:

print("Задание №6 Изображение точек на плоскости для этих двух признаков с регрессионной прямой")

print("----------------------------------------------------------------------")

sns.lmplot(x = "X", y = "Y", data = table\_n)

plt.show()

# ## Определение коэффициента корреляции Пирсона для этих двух признаков вручную и с применением существующего метода из numpy/scipy

# In[33]:

print("Задание №7 Определение коэффициента корреляции Пирсона")

print("----------------------------------------------------------------------")

def pirson1(lst):

aver1 = lst['X'].mean()

aver2 = lst['Y'].mean()

mas1 = lst['X'].tolist()

mas2 = lst['Y'].tolist()

s1, s2, s3 = 0, 0, 0

for x, y in zip(mas1, mas2):

s1 += (x-aver1)\*(y-aver2)

s2 += (x - aver1) \* (x - aver1)

s3 += (y - aver2) \* (y - aver2)

import math

s2 = math.sqrt(s2)

s3 = math.sqrt(s3)

s = s1/(s2\*s3)

return s

print("коэффициента корреляции Пирсона в ручную: ", pirson1(table\_n))

aver1 = table\_n['X'].tolist()

aver2 = table\_n['Y'].tolist()

x = np.corrcoef(np.vstack((aver1, aver2)))

otv = x[0][1]

print("коэффициента корреляции Пирсона с помощью np.corrcoef(): ",otv)

# ## Описание корреляции признаков в выборке и оценка возможности отклонить нулевую гипотезу о том, что в генеральной совокупности коэффициент корреляции равен нулю

# In[34]:

print("Описание корреляции признаков в выборке и оценка возможности отклонить нулевую гипотезу о том, что в генеральной совокупности коэффициент корреляции равен нулю")

print("----------------------------------------------------------------------")

if pval < 0.05 :

print('Гипотезу можно отвергнуть.')

else:

print('Гипотезу нельзя отвергнуть.')

1. **Часть №3**
   1. **Теоретическое часть**

Для корректного анализа данных необходимо проконтролировать, чтобы в таблице корректно были обработаны None значения. Так же во многих операциях None значения следует исключить, проконтролировав, чтобы структура данных не была нарушена (если мы исключили None значения из записи в одном столбце, но не исключим всю запись, мы нарушим структуру данных).

Данные Pandas можно преобразовать в любой удобный для работы формат, будь то Numpy массив, список и т.п. Но если это возможно, удобнее работать непосредственно с структурами данных Pandas. Они поддерживают многие типичные операция, а, например, Seaborn позволяет визуализировать данные, передав их в структуре данных модуля Pandas.

Мы можем работать и с Matplotlib Pyplot, если это необходимо.

Для изображения точек на плоскости для двух признаков можно использовать функцию scatter, передав ей в качестве параметров столбцы X и Y отдельно.

Для отображения прямых на данном графике в figure необходимо передать figzise, dpi, по желанию facecolor, edgecolor и т.п., использовать Pyplot plot для каждой прямой, передав данные по двум точкам и цвет и аналогично scatter для точек на графике, передав данные столбцов X и Y отдельно.

* 1. **Практическая часть**

# # ЧАСТЬ 3

# In[42]:

import scipy

# ## Моделирование данных по функции . В качестве нужно использовать гауссовский белый шум с нулевым математическим ожиданием и заданной дисперсией . Значения выбираются через равные промежутки на отрезке (количество точек , параметры a, b, σ^2 согласно варианту)

# In[43]:

print("Задание №1 Моделирование данных по функции . В качестве нужно использовать гауссовский белый шум с нулевым математическим ожиданием и заданной дисперсией . Значения выбираются через равные промежутки на отрезке (количество точек , параметры согласно варианту)")

print("----------------------------------------------------------------------")

n = 120

a = 0.5

b = 1

s = 0.1

x\_i = []

y\_i = []

y\_i\_2 = []

step = 1 / n

for i in range(n):

x = step

# e = sp.stats.norm.rvs(0, s, n)

x\_i.append(x)

y\_i.append(a \* x + b + scipy.stats.norm.rvs(loc=0, scale=math.sqrt(s)))

y\_i\_2.append(a \* x + b)

step += 1 / n

n5\_data = pd.DataFrame(data={

'X': x\_i,

'Y': y\_i,

'Y\_1': y\_i\_2

})

print(n5\_data)

# ## Изображение точек на плоскости для этих двух признаков

# In[44]:

print("Задание №2 Изображение точек на плоскости для этих двух признаков")

print("----------------------------------------------------------------------")

x = n5\_data['X'].tolist()

y = n5\_data['Y'].tolist()

fig, ax = plt.subplots()

ax.scatter(x, y, c='blue')

ax.set\_facecolor('white')

fig.set\_figwidth(8)

fig.set\_figheight(8)

plt.show()

# ## Построение модели линейной регрессии, вычисление оптимальных параметров из условия минимизации суммы квадратов отклонений для заданных значений признаков, используя общее матричное решение

# In[45]:

print("Задание №3 Построение модели линейной регрессии, вычисление оптимальных параметров из условия минимизации суммы квадратов отклонений для заданных значений признаков, используя общее матричное решение")

print("----------------------------------------------------------------------")

X = n5\_data['X']

Y = n5\_data['Y']

XY = X \* Y

a = np.array([[X.apply(lambda x: x\*x).sum(), X.sum()], [X.sum(), n]])

b = np.array([XY.sum(), Y.sum()])

x = np.linalg.solve(a, b)

print('a = ' + str(x[0]))

print('b = ' + str(x[1]))

y\_reg = []

for i in range(n):

y\_reg.append( x[0] \* X[i] + x[1])

n5\_data.insert(2, column='Y\_REG', value=y\_reg)

print(n5\_data)

# ## Визуализация полученных данных (изображение точек и 2 прямых)

# In[46]:

print("Задание №4 Визуализация полученных данных (изображение точек и 2 прямых) ")

print("----------------------------------------------------------------------")

plt.figure(num=None, figsize=(8, 6), dpi=120, facecolor='w', edgecolor='k')

plt.plot(n5\_data['X'], n5\_data['Y\_REG'], 'r')

plt.plot(n5\_data['X'], n5\_data['Y\_1'], 'b--')

plt.plot(n5\_data['X'], n5\_data['Y'], 'go')

plt.show()

# ## Получение всех данных из csv, вывод (данные согласно варианту)

# In[47]:

print("Задание №5 Вычислить коэффициент детерминации")

print("----------------------------------------------------------------------")

rss = 0

tss = 0

y\_ = n5\_data['Y'].sum()/n5\_data['Y'].size

for i in range(n5\_data['Y'].size):

rss += (n5\_data['Y'][i] - n5\_data['Y\_REG'][i])\*\*2

for i in range(n5\_data['Y'].size):

tss += (n5\_data['Y'][i] - y\_)\*\*2

r = 1 - rss/tss

print('R^2 = ' + str(r))

# In[48]:

print("Задание №6 Все данные из csv")

print("----------------------------------------------------------------------")

csv\_path = "F:/breast-cancer-wisconsin.csv"

lst = pd.read\_csv(csv\_path, sep=',')

print(lst)

# ## Создание нового датафрейма только с нужными признаками: предиктор и зависимая переменная (указаны в варианте)

# In[49]:

print("Задание №7 Создание нового датафрейма только с нужными для исследования признаками")

print("----------------------------------------------------------------------")

new\_lst = lst[['perimeter', 'area']]

print(new\_lst)

# ## Изображение точек на плоскости для этих двух признаков

# In[50]:

print("Задание №8 Вставить изображение точек на плоскости для этих двух столбцов")

print("----------------------------------------------------------------------")

x = new\_lst['perimeter'].tolist()

y = new\_lst['area'].tolist()

fig, ax = plt.subplots()

ax.scatter(x, y, c='blue')

ax.set\_facecolor('white')

fig.set\_figwidth(8)

fig.set\_figheight(8)

plt.show()

# ## Построение модели линейной регрессии, вычисление оптимальных параметров из условия минимизации суммы квадратов отклонений для заданных значений признаков, используя общее матричное решение

# In[51]:

print("Задание №9 Построение модели линейной регрессии, вычисление оптимальных параметров из условия минимизации суммы квадратов отклонений для заданных значений признаков, используя общее матричное решение")

print("----------------------------------------------------------------------")

X = new\_lst['perimeter']

Y = new\_lst['area']

XY = X \* Y

a = np.array([[X.apply(lambda x: x\*x).sum(), X.sum()], [X.sum(), n]])

b = np.array([XY.sum(), Y.sum()])

x = np.linalg.solve(a, b)

print('a = ' + str(x[0]))

print('b = ' + str(x[1]))

y\_reg = []

for i in range(349):

y\_reg.append( x[0] \* X[i] + x[1])

new\_lst.insert(2, column='Y\_REG', value=y\_reg)

print(new\_lst)

# ## Визуализация полученных данных (изображение точек и прямая)

# In[52]:

print("Задание №10 Визуализация полученных данных (изображение точек и 2 прямых) ")

print("----------------------------------------------------------------------")

plt.figure(num=None, figsize=(8, 6), dpi=120, facecolor='w', edgecolor='k')

plt.plot(new\_lst['perimeter'], new\_lst['Y\_REG'], 'r')

plt.plot(new\_lst['perimeter'], new\_lst['area'], 'go')

plt.show()

# ## Вычислить коэффициент детерминации

# In[53]:

print("Задание №11 Вычислить коэффициент детерминации")

print("----------------------------------------------------------------------")

rss = 0

tss = 0

y\_ = new\_lst['area'].sum()/new\_lst['area'].size

for i in range(new\_lst['area'].size):

rss += (new\_lst['area'][i] - new\_lst['Y\_REG'][i])\*\*2

for i in range(new\_lst['area'].size):

tss += (new\_lst['area'][i] - y\_)\*\*2

r = 1 - rss/tss

print('R^2 = ' + str(r))