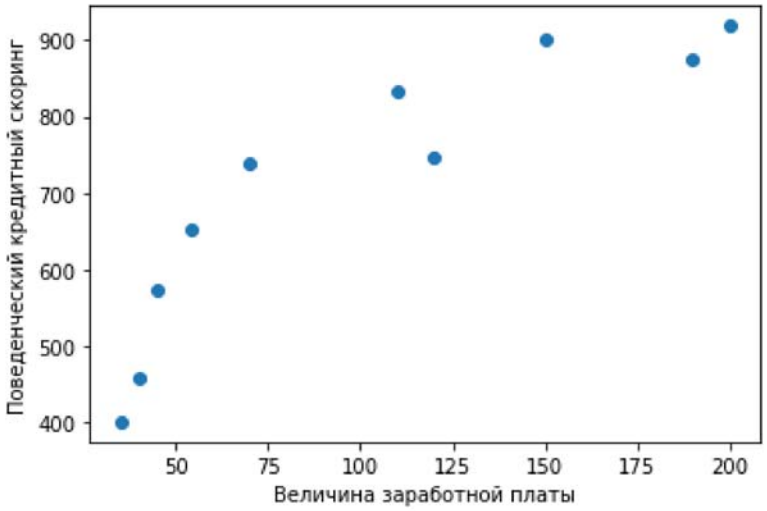
In [1]:
```
# Даны значения величины заработной платы заемщиков банка (zp) и значения их поведенческого кредитного скоринга (ks): zp = [35, 4
# Используя математические операции, посчитать коэффициенты линейной регрессии, приняв за X заработную плату (то есть, zp - призн
# Произвести расчет как с использованием intercept, так и без.
# Посчитать коэффициент линейной регрессии при заработной плате (zp), используя градиентный спуск (без intercept).
# **3. Произвести вычисления как в пункте 2, но с вычислением intercept. Учесть, что изменение коэффициентов должно производиться
# на каждом шаге одновременно (то есть изменение одного коэффициента не должно влиять на изменение другого во время одной итераци
```

In [2]:
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
zp = np.array([35, 45, 190, 200, 40, 70, 54, 150, 120, 110])
ks = np.array([401, 574, 874, 919, 459, 739, 653, 902, 746, 832])
```

In [1]:
```
# Построим график и посмотрим, есть ли зависимость между данными
```

In [3]:
```
plt.scatter(zp,ks)
plt.xlabel('Величина заработной платы')
plt.ylabel('Поведенческий кредитный скоринг', rotation=90)
plt.show()
```



In [4]:
```
# По графикку можно предположить некоторую линейную зависимость между двумя выборками.
# Исходя из условия задачи получаем следующую формулу.
# ks = a + d*zp
```

In [11]:
```
# Произведем расчет коэффициента d и интерсепта a
# В лекции рассматривались два способа, проверим оба
# Первый способ
d=(len(zp)*np.sum(ks*zp)-np.sum(zp)*np.sum(ks))/(len(zp)*np.sum(zp**2)-np.sum(zp)**2)
d    # d = 2.6205388824027653
```

Out[11]:
```
2.6205388824027653
```

In [12]:
```
# Второй способ
d=(np.mean(zp*ks)-np.mean(zp)*np.mean(ks))/(np.mean(zp**2)-np.mean(zp)**2)
d    # b1 = 2.620538882402765
```

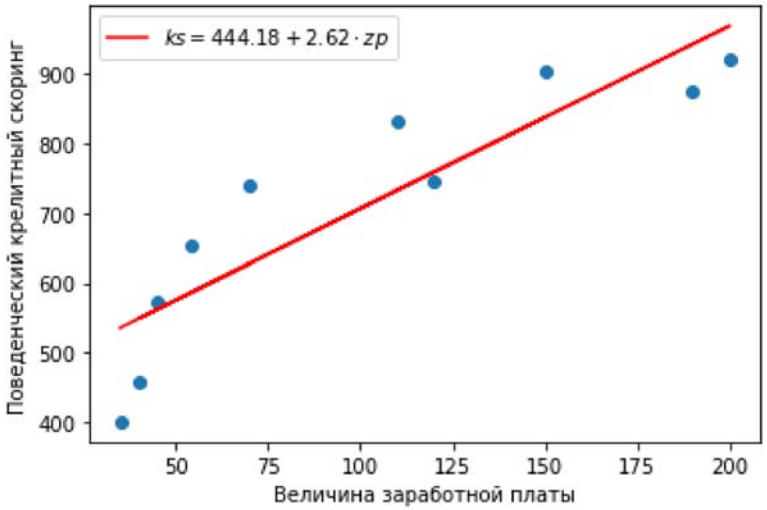
Out[12]:
```
2.620538882402765
```

In [13]:
```
a=np.mean(ks)-b1*np.mean(zp)   # интерсепт
a    # a = 444.17735732435955
```

Out[13]:
```
444.17735732435955
```

In [5]:
```
# Подставим найденные значения в формулу
# ks=444.18 + 2.62*zp
# Отобразим результат на графике (будет красиво)
```

In [6]:
```
plt.scatter(zp,ks)
plt.plot(zp, 444.18 + 2.62*zp, c='r', label=r'$ks=444.18+2.62\cdot zp$')
plt.legend()
plt.xlabel('Величина заработной платы')
plt.ylabel('Поведенческий крелитный скоринг', rotation=90)
plt.show()
```



In [ ]:
```
# Думаю, теперь линейная взаимосвязь между величиной заработной платы и кредитным скорингом очевидна.
```

In [7]:
```
# Градиентный спуск без интерсепта
y=ks
x=zp
# задание функции потерь
def mse_(d, y=y, x=x, n=10):
    return np.sum((d*x-y)**2)/n

alpha=1e-6  # скорость обучения, которое регулирует скорость подбора коэффициента d
d=0.1
n=10
```

In [8]:
```
for i in range (3000):
    d-=alpha*(2/n)*np.sum((d*x-y)*x)
    if i%500==0:
        print('Iteration={i}, d={d}, mse={mse}'.format(i=i, d=d, mse=mse_(d)))
```

```
Iteration=0, d=0.25952808, mse=493237.7212546963
Iteration=500, d=5.889815595583751, mse=56516.858416040064
Iteration=1000, d=5.8898204201285544, mse=56516.85841571941
Iteration=1500, d=5.889820420132673, mse=56516.85841571943
Iteration=2000, d=5.889820420132673, mse=56516.85841571943
Iteration=2500, d=5.889820420132673, mse=56516.85841571943
```

In [24]:
```
# Iteration=0, d=0.25952808, mse=493237.7212546963
# Iteration=500, d=5.889815595583751, mse=56516.858416040064
# Iteration=1000, d=5.8898204201285544, mse=56516.85841571941
# Iteration=1500, d=5.889820420132673, mse=56516.85841571943
# Iteration=2000, d=5.889820420132673, mse=56516.85841571943
# Iteration=2500, d=5.889820420132673, mse=56516.85841571943
```

In [25]:
```
# Проверка коэффициента d с помощью ранее записанной функции потерь mse
# и сравнение со значением из цикла
mse_(5.889820420132673)

# полученный результат 56516.85841571943, что соответствует значению из цикла
```

Out[25]:
```
56516.85841571943
```

In [26]:
```
# Градиентный спуск с интерсептом
```

In [38]:
```
# Функция потерь
def _mse_ad(a,d, x, y):
    return np.sum(((a+d*x)-y)**2)/len(x)
```

In [39]:
```
# Частная производная функции потерь по a:
def _mse_pa(a,d,x,y):
    return 2*np.sum((a+d*x)-y)/len(x)
```

In [40]:
```
# Частная производная функции потерь по b:
def _mse_pd(a,d,x,y):
    return 2*np.sum(((a+d*x)-y)*x)/len(x)
```

In [41]:
```
# Скорость обучения
alpha=5e-05
```

In [45]:
```
d=0.1
a=0.1
msead_min=_mse_ad(a,d,zp,ks)
i_min=1
d_min=d
a_min=a

for i in range(1000000):
```

```
    a-=alpha*_mse_pa(a,d,zp,ks)
    d-=alpha*_mse_pd(a,d,zp,ks)
    if i%50000==0:
        print(f'Iteration ={i}, a={a}, d={d}, mse={_mse_ad(a, d, zp,ks)}')
    if _mse_ad(a, d,zp,ks)>msead_min:
        print(f'Iteration ={i_min}, a={a_min}, d={d_min}, mse={msead_min},\nДостигнут минимум.')
        break
    else:
        msead_min=_mse_ad(a, d,zp,ks)
        i_min=i
        d_min=d
        a_min=a
print(f'a={a_min}\nd={d_min}')
```

```
Iteration =0, a=0.169966, d=8.07468054476, mse=122318.06397097567
Iteration =50000, a=319.27767648420047, d=3.5398324356503275, mse=10427.569111705801
Iteration =100000, a=409.0442373734796, d=2.879127619051743, mse=6783.521961452364
Iteration =150000, a=434.29473705519484, d=2.693277491833349, mse=6495.188684804794
Iteration =200000, a=441.3974680483413, d=2.6409995775222037, mse=6472.374468908443
Iteration =250000, a=443.39540029510493, d=2.62629428586797, mse=6470.569306309746
Iteration =300000, a=443.95740007610897, d=2.622157823932053, mse=6470.426473787141
Iteration =350000, a=444.1154853937451, d=2.6209942756156086, mse=6470.415172240385
Iteration =400000, a=444.159953325044, d=2.6206669802831115, mse=6470.414278011555
Iteration =450000, a=444.1724617410292, d=2.6205749151465225, mse=6470.414207256183
Iteration =500000, a=444.1759802422447, d=2.6205490180788695, mse=6470.414201657699
Iteration =520164, a=444.17653163778414, d=2.62054495966686, mse=6470.414201349592,
Достигнут минимум.
a=444.17653163778414
d=2.62054495966686
```

In [ ]:

```
# Iteration =0, a=0.169966, d=8.07468054476, mse=122318.06397097567
# Iteration =50000, a=319.27767648420047, d=3.5398324356503275, mse=10427.569111705801
# Iteration =100000, a=409.0442373734796, d=2.879127619051743, mse=6783.521961452364
# Iteration =150000, a=434.29473705519484, d=2.693277491833349, mse=6495.188684804794
# Iteration =200000, a=441.3974680483413, d=2.6409995775222037, mse=6472.374468908443
# Iteration =250000, a=443.39540029510493, d=2.62629428586797, mse=6470.569306309746
# Iteration =300000, a=443.95740007610897, d=2.622157823932053, mse=6470.426473787141
# Iteration =350000, a=444.1154853937451, d=2.6209942756156086, mse=6470.415172240385
# Iteration =400000, a=444.159953325044, d=2.6206669802831115, mse=6470.414278011555
# Iteration =450000, a=444.1724617410292, d=2.6205749151465225, mse=6470.414207256183
# Iteration =500000, a=444.1759802422447, d=2.6205490180788695, mse=6470.414201657699
# Iteration =520164, a=444.17653163778414, d=2.62054495966686, mse=6470.414201349592,
# Достигнут минимум.
```

```
# a=444.17653163778414
# d=2.62054495966686
```