9.1

In [476]:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
from pandas import read_csv
from numpy.linalg import inv
import math
import random

%matplotlib inline
```

In [477]:

```
def shuffle(data):
    data = read_csv('forestfires.csv')
    data = data.iloc[np.random.permutation(len(data))]

    data.replace(['aug', 'jun', 'jul', 'jan', 'feb', 'mar', 'apr', 'may', 'sep
             [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'month')
    area = data.area.copy()
    data.drop('day',axis=1,inplace=True)
    data.drop('area',axis=1,inplace=True)
    data['lenear'] = 1
    return data, area
data, area = shuffle(data)
data
```

Out[477]:

	X	Y	month	FFMC	DMC	DC	ISI	temp	RH	wind	rain	lenear
128	3	5	0	91.4	37.9	673.8	5.2	15.9	46	3.6	0	1
392	1	3	0	91.0	276.3	825.1	7.1	21.9	43	4.0	0	1
60	2	2	0	89.3	51.3	102.2	9.6	11.5	39	5.8	0	1
0	7	5	0	86.2	26.2	94.3	5.1	8.2	51	6.7	0	1
129	2	5	0	92.6	46.5	691.8	8.8	15.4	35	0.9	0	1
275	4	6	0	84.6	26.4	352.0	2.0	5.1	61	4.9	0	1
224	7	4	0	90.1	82.9	735.7	6.2	15.4	57	4.5	0	1
1	7	4	0	90.6	35.4	669.1	6.7	18.0	33	0.9	0	1
13	6	5	0	90.9	126.5	686.5	7.0	21.3	42	2.2	0	1
296	6	4	1	90.4	89.5	290.8	6.4	14.3	46	1.8	0	1
189	7	4	0	90.7	44.0	92.4	5.5	11.5	60	4.0	0	1
397	5	6	1	91.6	181.3	613.0	7.6	24.3	33	3.6	0	1
414	5	4	1	93.6	235.1	723.1	10.1	24.1	50	4.0	0	1
65	2	2	1	91.7	114.3	661.3	6.3	18.6	44	4.5	0	1
510	6	5	1	91.0	166.9	752.6	7.1	18.2	62	5.4	0	1
285	2	5	1	93.9	169.7	411.8	12.3	23.4	40	6.3	0	1
197	4	5	0	92.9	137.0	706.4	9.2	21.5	15	0.9	0	1
371	3	4	1	91.9	133.6	520.5	8.0	14.2	58	4.0	0	1
68	2	2	0	92.4	117.9	668.0	12.2	19.6	33	6.3	0	1
52	4	3	1	92.1	111.2	654.1	9.6	20.4	42	4.9	0	1
394	6	5	0	84.1	4.6	46.7	2.2	5.3	68	1.8	0	1
32	6	3	0	88.6	69.7	706.8	5.8	20.6	37	1.8	0	1

322	6	5	0	92.8	119.0	783.5	7.5	16.8	28	4.0	0	1
324	6	5	0	88.1	53.3	726.9	5.4	13.7	56	1.8	0	1
46	5	6	0	90.9	126.5	686.5	7.0	14.7	70	3.6	0	1
101	3	4	1	88.8	147.3	614.5	9.0	14.4	66	5.4	0	1
49	4	4	0	87.6	52.2	103.8	5.0	11.0	46	5.8	0	1
312	2	4	0	50.4	46.2	706.6	0.4	12.2	78	6.3	0	1
298	8	6	1	91.2	147.8	377.2	12.7	19.6	43	4.9	0	1
373	5	4	1	94.8	222.4	698.6	13.9	20.3	42	2.7	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...
11	7	5	0	92.8	73.2	713.0	22.6	19.3	38	4.0	0	1
278	4	4	0	85.4	25.4	349.7	2.6	4.6	21	8.5	0	1
43	4	4	0	92.5	88.0	698.6	7.1	19.6	48	2.7	0	1
30	6	3	0	94.3	85.1	692.3	15.9	25.4	24	3.6	0	1
143	1	2	1	90.0	51.3	296.3	8.7	16.6	53	5.4	0	1
258	3	4	1	91.8	170.9	692.3	13.7	20.6	59	0.9	0	1
281	6	5	0	85.4	25.4	349.7	2.6	5.1	24	8.5	0	1
44	4	4	0	90.1	82.9	735.7	6.2	12.9	74	4.9	0	1
290	2	5	1	91.6	104.2	474.9	9.0	18.7	53	1.8	0	1
63	2	2	1	90.2	99.6	631.2	6.3	20.8	33	2.7	0	1
240	6	3	0	88.0	17.2	43.5	3.8	15.2	51	2.7	0	1
288	7	4	1	91.6	104.2	474.9	9.0	24.2	32	1.8	0	1
199	2	4	0	63.5	70.8	665.3	0.8	22.6	38	3.6	0	1
513	2	4	1	81.6	56.7	665.6	1.9	21.9	71	5.8	0	1
170	5	4	0	92.9	133.3	699.6	9.2	21.9	35	1.8	0	1
376	8	6	1	92.1	207.0	672.6	8.2	21.1	54	2.2	0	1
172	7	4	1	91.4	142.4	601.4	10.6	20.1	39	5.4	0	1
225	4	4	0	93.5	149.3	728.6	8.1	22.9	39	4.9	0	1
506	1	2	1	91.0	166.9	752.6	7.1	18.5	73	8.5	0	1
426	8	6	1	91.6	248.4	753.8	6.3	20.4	56	2.2	0	1
385	2	4	1	91.6	181.3	613.0	7.6	20.9	50	2.2	0	1
469	6	3	0	91.0	14.6	25.6	12.3	13.7	33	9.4	0	1
153	5	4	0	94.3	85.1	692.3	15.9	20.1	47	4.9	0	1
255	2	5	1	87.5	77.0	694.8	5.0	22.3	46	4.0	0	1
116	3	4	0	91.7	35.8	80.8	7.8	11.6	30	6.3	0	1

444	2	5	0	90.3	290.0	855.3	7.4	16.2	58	3.6	0	1
168	6	5	0	91.2	48.3	97.8	12.5	14.6	26	9.4	0	1
66	2	2	0	92.4	117.9	668.0	12.2	23.0	37	4.5	0	1
93	8	6	1	91.4	142.4	601.4	10.6	18.2	43	4.9	0	1
100	3	4	1	91.4	142.4	601.4	10.6	19.8	39	5.4	0	1

517 rows × 12 columns

Разделим данные в отношении 7:3.

In [478]:

```
first = data[:362]
area_1 = area[:362]
second = data[363:]
area_2 = area[363:]
```

Построим регрессионную модель по первой части выборки.

$X = Z\theta + \varepsilon$. Найдем оценку коэффициентов линейной регрессии методом наименьших квадратов:

$\hat{\theta} = (Z^T Z)^{-1} Z^T X$, где X - вектор данных area, Z - матрица остальных данных.

In [479]:

```
def est(z, x):
    Z = np.matrix(z)
    X = np.matrix(x)
    Z_T = Z.transpose()
    theta = inv(Z_T*Z) * Z_T * X.transpose()
    return theta
```

In [480]:

```
theta_1 = est(first.as_matrix(), area_1.as_matrix())
for j in range(len(theta_1)):
    print(j + 1, ": ", theta[j])
```

```
1 :  [[ 0.04141153]]
2 :  [[ 0.00174335]]
3 :  [[-0.41973402]]
4 :  [[ 0.00509793]]
5 :  [[ 0.00382227]]
6 :  [[-0.00025025]]
7 :  [[-0.01490258]]
8 :  [[ 0.01151988]]
9 :  [[-0.00817923]]
10 :  [[ 0.06621594]]
11 :  [[ 0.16797497]]
12 :  [[ 0.31239513]]
```

- вектор $\hat{\theta}$. Посчитаем среднеквадратичную ошибку $\hat{\sigma} = \frac{\|X - Z\hat{\theta}\|^2}{n-k}$ по второй части выборки.

In [481]:

```
def sigma(z, x, t):
    Z = np.matrix(z)
    X = np.matrix(x)

    alpha = (np.array(X.transpose()-Z*t))**2
    return (alpha.sum()/(len(x) - 12))**0.5
```

In [482]:

```
print("Полученная среднеквадратичная ошибка: ", sigma(second.as_matrix(), area
```

```
Полученная среднеквадратичная ошибка:  32.7987720055
```

Сделаем для area преобразование $f(x) = ln(c + x)$ и построим для нее регрессионную модель. Посчитаем среднеквадратичную ошибку для преобразованных значений по данному правилу и для исходных, применив в последнем случае к оценкам обратное к f преобразование.

In [483]:

```
def error(first, second, area_1, area_2, c):
    new_area_1 = [math.log(c + x) for x in area_1]
    new_area_2 = [math.log(c + x) for x in area_2]
    theta = est(first.as_matrix(), new_area_1)
    new_theta = np.matrix([math.exp(x) - c for x in theta]).transpose()
    one = sigma(second.as_matrix(), new_area_2, theta)
    two = sigma(second.as_matrix(), area_2.as_matrix(), new_theta)
    return one, two
```

Сделаем сначала расчет для произвольного c. Например, возьмем c = 3:

In [484]:

```
err = error(first, second, area_1, area_2, 3)
print("Ошибка для преобразованных данных по правилу f=ln(x+c): ", err[0])
print("Ошибка для исходных данных: ", err[1])
```

```
Ошибка для преобразованных данных по правилу f=ln(x+c):  1.064810
68753
Ошибка для исходных данных:  1895.86278438
```

Как видим, преобразованные данные дают лучшую оценку.

Рассчитаем среднеквадратичные ошибки при различных с и найдем, при каком с предсказания получаются лучше.

In [485]:

```
L = np.array([])
x = np.arange(0.1,100,0.1)
for i in x:
    L = np.append(L, error(first, second, area_1, area_2, i)[1])
print("Предсказания получаются лучше всего при с = ", x[np.where(L == L.min())
```

```
Предсказания получаются лучше всего при с =  1.0
```

Тогда наше пробразование$f(x) = ln(x + c)$хорошо тем, что оно $area = 0$(которых в нашей выборке 247 штук) переводит в $f(area) = 0$. Таким образом, логарифмирование улучшает нашу выборку и сопоставляет старым величинам величины, более близкие к нулю. Таким образом, наша выборка становится более симметричной.

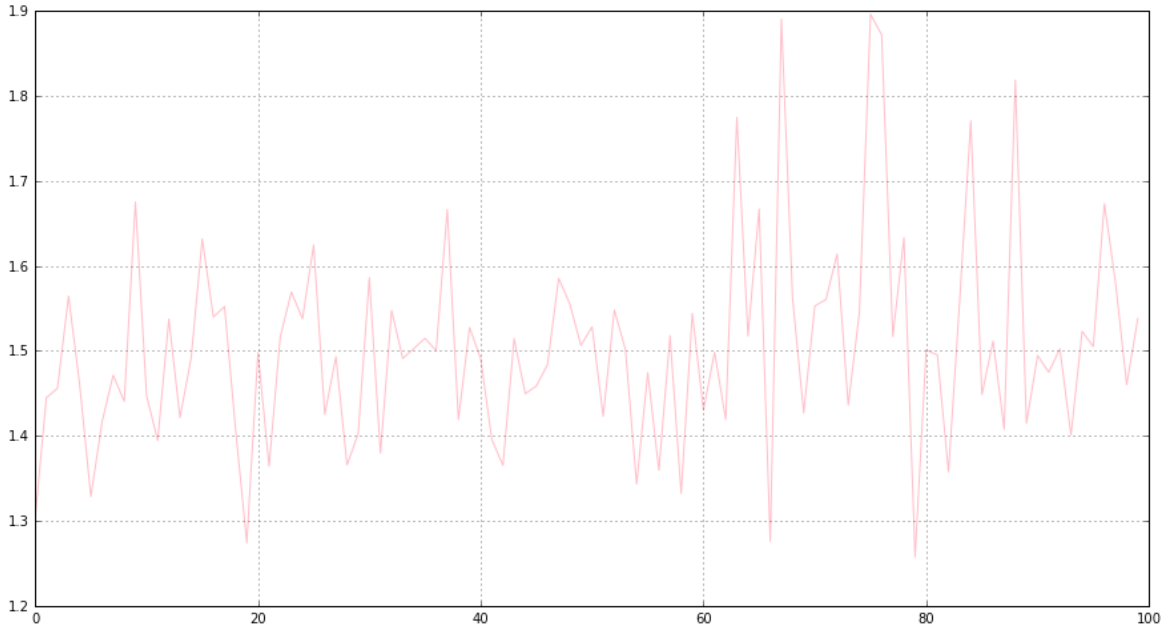
Разобьем выборку несколькими способами и пронаблюдаем зависимость среднеквадратичной ошибки от разбиения выборки. Произведем 100 разбиений и построим график значения среднеквадратичной ошибки от номера разбиения.

In [488]:

```
R = np.array([])
n = np.arange(100)
for i in n:
    data, area = shuffle(data)
    first = data[:362]
    area_1 = area[:362]
    second = data[363:]
    area_2 = area[363:]
    R = np.append(R, error(first, second, area_1, area_2, 1)[0])
```

In [491]:

```
plt.figure(figsize=(15, 8))
plt.plot(n, R, color='pink')
plt.grid()
plt.show()
```



Как видно из графика, разброс среднеквадратичной ошибки в зависимости от разбиения колеблется в допустимых пределах. Следовательно, никакого ухудшения качества не замечено.