Exp2：基于回归分析的大学综合得分预测

目录

一、案例简介

大学排名是一个非常重要同时也极富挑战性与争议性的问题，一所大学的综合实力涉及科研、师资、学生等方方面面。目前全球有上百家评估机构会评估大学的综合得分进行排序，而这些机构的打分也往往并不一致。在这些评分机构中，世界大学排名中心（Center for World University Rankings，缩写CWUR）以评估教育质量、校友就业、研究成果和引用，而非依赖于调查和大学所提交的数据著称，是非常有影响力的一个。

本任务中我们将根据 CWUR 所提供的世界各地知名大学各方面的排名（师资、科研等），一方面通过数据可视化的方式观察不同大学的特点，另一方面希望构建机器学习模型（线性回归）预测一所大学的综合得分。

二、作业说明

使用来自 Kaggle 的数据 (https://www.kaggle.com/mylesoneill/world-university-rankings?select=cwurData.csv)，构建「线性回归」模型，根据大学各项指标的排名预测综合得分。

基本要求：

- 按照 8:2 随机划分训练集测试集，用 RMSE 作为评价指标，得到测试集上线性回归模型的 RMSE 值；
- 对线性回归模型的系数进行分析。

扩展要求：

- 对数据进行观察与可视化，展示数据特点；
- 尝试其他的回归模型，对比效果；
- 尝试将离散的国家特征融入线性回归模型，并对结果进行对比。

注意事项：

- 基本输入特征有 8 个： `quality_of_education`，`alumni_employment`，`quality_of_faculty`，`publications`，`influence`，`citations`，`broad_impact`，`patents`；
- 预测目标为 `score`；
- 可以使用 sklearn 等第三方库，不要求自己实现线性回归；
- 需要保留所有数据集生成、模型训练测试的代码；

备注： 要不要做数据处理、标准化 可以把清华北大排名找出来 不同地区的大学的排名 画图包的使用，数据的可视化 其他的回归模型 国家的特征，进行onehot离散，融入模型 可以用pdf形式进行提交，交实验报告和代码（充分注释） rmse,mse开根号

三、数据概览

假设数据文件位于当前文件夹，我们用 pandas 读入标准 csv 格式文件的函数 `read_csv()` 将数据转换为 `DataFrame` 的形式。观察前几条数据记录：

In [1]:

```
import pandas as pd
import numpy as np

data_df = pd.read_csv('./cwurData.csv')  # 读入 csv 文件为 pandas 的 DataFrame
data_df.head(3).T  # 观察前几列并转置方便观察
```

Out[1]:

	0	1	2
world_rank	1	2	3
institution	Harvard University	Massachusetts Institute of Technology	Stanford University
country	USA	USA	USA
national_rank	1	2	3
quality_of_education	7	9	17
alumni_employment	9	17	11
quality_of_faculty	1	3	5
publications	1	12	4
influence	1	4	2
citations	1	4	2
broad_impact	NaN	NaN	NaN
patents	5	1	15
score	100	91.67	89.5
year	2012	2012	2012

去除其中包含 NaN 的数据，保留 2000 条有效记录。

In [2]:

```
data_df = data_df.dropna()  # 舍去包含 NaN 的 row
len(data_df)
```

Out[2]:

2000

取出对应自变量以及因变量的列，之后就可以基于此切分训练集和测试集，并进行模型构建与分析。

In [3]:

```
feature_cols = ['quality_of_faculty', 'publications', 'citations', 'alumni_employment',
                'influence', 'quality_of_education', 'broad_impact', 'patents']
X = data_df[feature_cols]
Y = data_df['score']
Y
```

Out[3]:

```
200     100.00
201      99.09
202      98.69
203      97.64
204      97.51
         ...
2195     44.03
2196     44.03
2197     44.03
2198     44.02
2199     44.02
Name: score, Length: 2000, dtype: float64
```

四、模型构建

4.1 简单的模型

In [4]:

```
#划分数据集
from sklearn.model_selection import train_test_split, cross_validate
feature_names = list(X[:0])
all_x = np.array(X)
all_y = np.array(Y)
train_x, test_x, train_y, test_y = train_test_split(all_x, all_y, test_size = 0.2, random_state
= 2020)
```

In [5]:

```
#定义模型
class LinearRegression:

    def fit(self, X, y): #求回归系数
        constant = np.ones(len(X))
        x = np.c_[constant, X]
        # B = (X'X)^(-1)X'Y 其中求近似的inverse用到了pseudo inverse
        self.B = np.matmul(np.matmul(np.linalg.pinv(np.matmul(x.T, x)),x.T),y)
        return self

    def predict(self, x): #预测得分
        constant = np.ones(len(x))
        x = np.c_[constant, x]
        b = self.B
        y = np.matmul(x, b)
        return y
```

In [6]:

```
#定义一些函数
def rmse(y, yp): #求rmse
    e = y - yp
    E = np.matmul(e.T, e)
    rmse = np.sqrt(E/len(y))
    return rmse

def mse(y, yp): #求mse
    e = y - yp
    E = np.matmul(e.T, e)
    mse = E/len(y)
    return mse

def mean(y): #求mean
    sum = 0
    for i in range(len(y)):
        sum += y[i]
    mean = sum / len(y)
    return mean

def var(y): #求variance
    ym = np.ones(len(y)) * mean(y)
    v = y - ym
    V = np.matmul(v.T, v)
    var1 = V / len(y)
    return var1

def std(y): #求标准差
    var1 = var(y)
    std1 = np.sqrt(var1)
    return std1

def cod(y, yp): #求线性回归的决定系数（coefficient of determination）
    cod = 1 - mse(y, yp)/ var(y)
    return cod

def rcc(x, y): #求线性回归的相关系数r
    xm = mean(x)
    ym = mean(y)
    xstd = np.sqrt(var(x))
    ystd = np.sqrt(var(y))
    sum = 0
    for i in range(len(x)):
        sum += ((x[i] - xm) / xstd) * ((y[i] - ym) / ystd)
    r = sum / (len(x) - 1)
    return r
```

In [7]:

```
#生成模型，打印回归系数和RMSE
LR = LinearRegression()
LR.fit(train_x, train_y)
p = LR.predict(test_x)
RMSE = rmse(test_y, p)
print(LR.B)
print("RMSE:", RMSE)
```

```
[ 6.55542426e+01 -6.17588386e-02  2.82375384e-04  1.31975766e-05
 -7.20591259e-03  6.57612913e-04 -6.55744975e-03 -2.65494903e-03
 -2.42756500e-03]
RMSE: 3.9990456867468662
```

RMSE值为3.999。

In [8]:

```
#按顺序打印各项回归系数
from copy import deepcopy
b = deepcopy(LR.B)
features = ['y-intercept']
for f in feature_names:
    features.append(f)

for i in range(len(b)):
    for j in range(len(b) - i - 1):
        if b[j]>b[j+1]:
            b[j], b[j+1] = b[j+1], b[j]
            features[j], features[j+1] = features[j+1], features[j]

for i in range(len(b)):
        print("{}:{:.5f}".format(features[i], b[i]))
```

```
quality_of_faculty:-0.06176
alumni_employment:-0.00721
quality_of_education:-0.00656
broad_impact:-0.00265
patents:-0.00243
citations:0.00001
publications:0.00028
influence:0.00066
y-intercept:65.55424
```

常数项（截距）约为65.55，是基础分。 系数（斜率）为负，表示，排名越靠后（值越大），扣分越多。

其中，quality_of_faculty影响最大，每落后一名，约扣0.06176分；citations影响最小，甚至加分。其他特征按对得分的影响从大到小的排序是：alumni_employment,quality_of_education, broad_impact, patents, influence, publications。

很明显这个模型不靠谱，因为最大得分也只有65.55左右，且有些特征排名越靠后反而带来更高的分数。

让我们来看看这个模型的拟合优度:

In [9]:

```
#求决定系数（拟合优度）
VAR = var(test_y)
MSE = mse(test_y, p)
COD = cod(test_y, p)
print("var:{:.4f} mse:{:.4f}".format(VAR, MSE))
print("codfficient of determination:{:.4f}".format(COD))
```

```
var:38.4352 mse:15.9924
codfficient of determination:0.5839
```

由此可见，决定系数并不接近1，回归分析中，自变量对因变量的解释并不是很好。

考虑：可以调整特征值

4.2 优化模型

4.2.1 对特征值取log

考虑：排名为1的时候应该不扣分，排名越靠后，每下降一名对分数的影响越小。

可以考虑使用ln函数。 由于后面用的是线性回归模型，这里log取的底数的取值对最终结果没有影响。

In [10]:

```
#对每个排名求log值
import math
all_x2 = deepcopy(all_x)
print(all_x2[1])
for i, c in enumerate(all_x2):
    for j, num in enumerate(c):
        all_x2[i][j] = math.log(num)
print(all_x2[1])
```

```
[ 4.  5.  3.  2.  3. 11.  4.  6.]
[1.38629436 1.60943791 1.09861229 0.69314718 1.09861229 2.39789527
 1.38629436 1.79175947]
```

In [11]:

```
#重新划分数据集
all_y2 = deepcopy(all_y)
train_x, test_x, train_y, test_y = train_test_split(all_x2, all_y2, test_size = 0.2, random_stat
e = 2020)
```

In [12]:

```
#训练模型
LR = LinearRegression()
LR.fit(train_x, train_y)
p = LR.predict(test_x)
RMSE = rmse(test_y, p)
print(LR.B)
print("RMSE:", RMSE)
```

```
[ 1.01012724e+02 -5.01790321e+00 -1.55445425e-01  2.46085313e-02
 -1.86006918e+00 -7.44085184e-02 -1.30460466e+00 -8.23799023e-01
 -7.03405722e-01]
RMSE: 2.141249284382892
```

优化后的RMSE值为2.141，优化明显。

In [13]:

```
from copy import deepcopy
b = deepcopy(LR.B)
features = ['y-intercept']
for f in feature_names:
    features.append(f)

print(features)
print(b)

for i in range(len(b)):
    for j in range(len(b) - i - 1):
        if b[j]>b[j+1]:
            b[j], b[j+1] = b[j+1], b[j]
            features[j], features[j+1] = features[j+1], features[j]

for i in range(len(b)):
        print("{}:{:.5f}".format(features[i], b[i]))
```

```
['y-intercept', 'quality_of_faculty', 'publications', 'citations', 'alumni_employm
ent', 'influence', 'quality_of_education', 'broad_impact', 'patents']
[ 1.01012724e+02 -5.01790321e+00 -1.55445425e-01  2.46085313e-02
 -1.86006918e+00 -7.44085184e-02 -1.30460466e+00 -8.23799023e-01
 -7.03405722e-01]
quality_of_faculty:-5.01790
alumni_employment:-1.86007
quality_of_education:-1.30460
broad_impact:-0.82380
patents:-0.70341
publications:-0.15545
influence:-0.07441
citations:0.02461
y-intercept:101.01272
```

常数项（截距）约为101.01，是基础分，比之前的65.55更靠近实际值100。

系数（斜率）为负，表示，排名越靠后，log(排名）值越大，扣分越多。 其中，quality_of_faculty影响最大，citations影响最小，甚至加分。

让我们来看一下决定系数：

In [14]:

```
VAR = var(test_y)
MSE = mse(test_y, p)
COD = cod(test_y, p)
print("var:{:.4f} mse:{:.4f}".format(VAR, MSE))
print("codfficient of determination:{:.4f}".format(COD))
```

```
var:38.4352 mse:4.5849
codfficient of determination:0.8807
```

mse和cod明显下降。

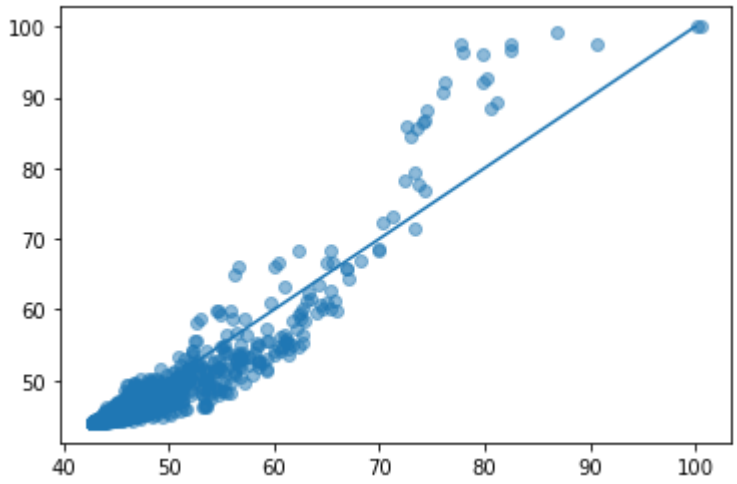
线性回归的决定系数（拟合优度）接近1，模型对数据的解释度较好。

In [15]:

```
pt = LR.predict(train_x)
import matplotlib.pyplot as plt
plt.scatter(pt, train_y, alpha = 0.5)
xx = [50, 100]
plt.plot(xx, xx)
```

Out[15]:

```
[<matplotlib.lines.Line2D at 0x1fe09c14e08>]
```



从上图可以发现，大部分预测值都小于真实值，这是由于少部分过高于真实值的outlier造成的。 思考：怎么去掉这些outlier。

4.2.2 去除outlier

In [16]:

```
error = pt - train_y
sum(abs(error) > 6)
```

Out[16]:

```
54
```

In [17]:

```
train_x2 = []
train_y2 = []
for i, x in enumerate(train_x):
    if (abs(error) > 10)[i] == False:
        train_x2.append(x)
        train_y2.append(train_y[i])

train_x2 = np.array(train_x2)
train_y2 = np.array(train_y2)
```

In [18]:

```
#训练模型
LR = LinearRegression()
LR.fit(train_x2, train_y2)
p = LR.predict(test_x)
RMSE = rmse(test_y, p)
print(LR.B)
print("RMSE:", RMSE)
```

```
[91.91561703 -3.79655741 -0.15965602  0.16635611 -1.63541726  0.19872046
 -0.85145234 -1.44272591 -0.69370197]
RMSE: 2.303957736912779
```

In [19]:

```
VAR = var(test_y)
MSE = mse(test_y, p)
COD = cod(test_y, p)
print("var:{:.4f} mse:{:.4f}".format(VAR, MSE))
print("codfficient of determination:{:.4f}".format(COD))
```

```
var:38.4352 mse:5.3082
codfficient of determination:0.8619
```

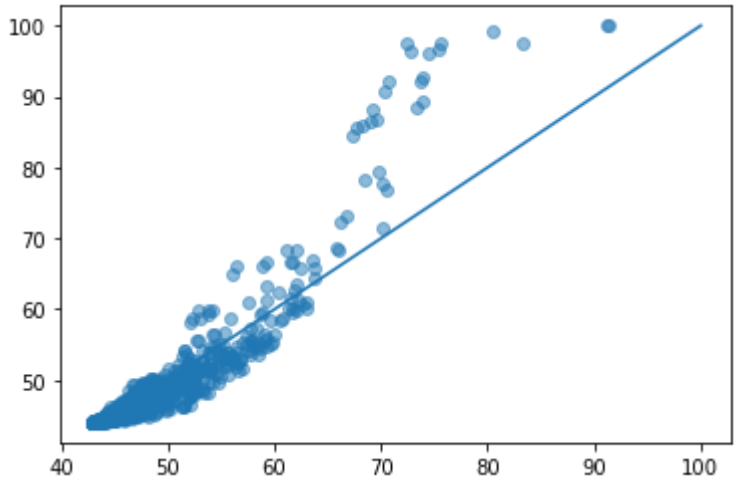
模型反而变差了>.< 难过。

In [20]:

```
pt = LR.predict(train_x)
import matplotlib.pyplot as plt
plt.scatter(pt, train_y, alpha = 0.5)
xx = [50, 100]
plt.plot(xx, xx)
```

Out[20]:

```
[<matplotlib.lines.Line2D at 0x1fe09cbf708>]
```



4.2.3 归一化/标准化，取两次log

将特征值再处理一下试试： 在后面可视化画图的时候发现，log后的图，依旧带有一点反过来的log图的样子，所以这里多取了一次log。

In [21]:

```
X2 = deepcopy(X)

for c in X2.columns:
    X2[c] = np.log(X2[c]) #取log值
    X2[c] = np.log(X2[c]+1) #为避免log0，所以+1
#     X2[c] = np.log(X2[c]+1)
#     X2[c] = X2[c] / (max(X2[c]) - min(X2[c]) ) #归一化
#     X2[c] = (X2[c] - mean(list(X2[c]))) / std(list(X2[c])) #标准化
max(list(X2['quality_of_faculty']))
```

Out[21]:

```
1.8538724045702297
```

备注（经过试验发现：归一化和标准化并不会影响线性回归模型最终得到RMSE和COD）

In [23]:

```
feature_names = list(X2[:0])
all_x2 = np.array(X2)
all_y = np.array(Y)
train_x, test_x, train_y, test_y = train_test_split(all_x2, all_y, test_size = 0.2, random_state
= 2020)
LR = LinearRegression()
LR.fit(train_x, train_y)
p = LR.predict(test_x)
RMSE = rmse(test_y, p)
print(LR.B)
print("RMSE:", RMSE)
VAR = var(test_y)
MSE = mse(test_y, p)
COD = cod(test_y, p)
print("var:{:.4f} mse:{:.4f}".format(VAR, MSE))
print("codfficient of determination:{:.4f}".format(COD))
```

```
[132.2717777  -15.88560002  -1.81866369  -1.33482863  -8.70003909
  -6.84721445  -7.78327306   1.17190408  -4.43475377]
RMSE: 1.5882915474157788
var:38.4352 mse:2.5227
codfficient of determination:0.9344
```

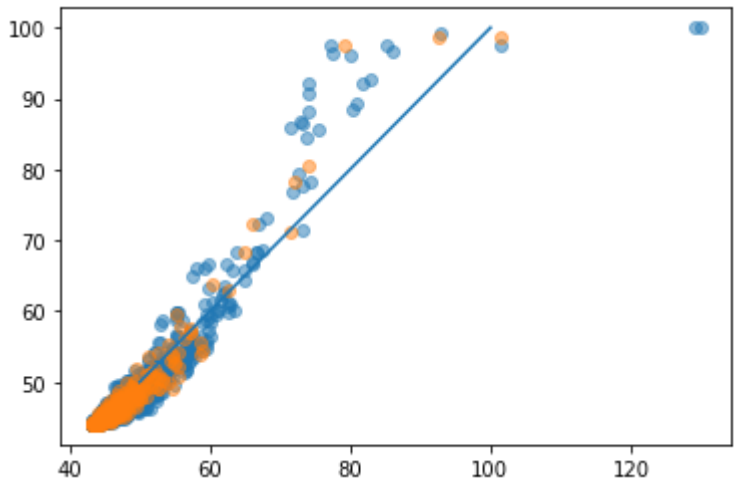
对比发现，将特征数据取两次log值后，RMSE更小了，COD更接近1了，模型效果有提升。

In [24]:

```
pt = LR.predict(train_x)
import matplotlib.pyplot as plt
plt.scatter(pt, train_y, alpha = 0.5)
plt.scatter(p, test_y, alpha = 0.5)
xx = [50, 100]
plt.plot(xx, xx)
error = pt - train_y
sum(abs(error) > 6)
```

Out[24]:

25



对比前面的数据可以发现，train集中error大于6的数量减少了。

五、数据观察与可视化

5.1 世界地图

绘制一张各国拥有世界Top2000的大学数量的世界地图。

In [79]:

```
country_names = np.array(list(set(data_df['country']))) #所有的国家名称
print(country_names)
len(country_names)
```

```
['Iran' 'Iceland' 'Slovak Republic' 'Colombia' 'Canada' 'Argentina'
 'Norway' 'Romania' 'Bulgaria' 'France' 'Lithuania' 'Ireland' 'Mexico'
 'Hungary' 'Switzerland' 'Egypt' 'Australia' 'Netherlands' 'New Zealand'
 'India' 'Japan' 'Malaysia' 'Serbia' 'Slovenia' 'Austria' 'Estonia'
 'Croatia' 'Denmark' 'Singapore' 'Spain' 'Portugal' 'South Korea'
 'Czech Republic' 'Thailand' 'Hong Kong' 'Lebanon' 'Puerto Rico' 'Italy'
 'Belgium' 'Greece' 'South Africa' 'Poland' 'United Arab Emirates' 'China'
 'Israel' 'Uganda' 'Russia' 'Finland' 'Taiwan' 'Uruguay' 'Chile' 'Cyprus'
 'Turkey' 'USA' 'Sweden' 'Germany' 'United Kingdom' 'Brazil'
 'Saudi Arabia']
```

Out[79]:

59

In [80]:

```
top = 1000 #可调参数：要统计的进入排名前多少的大学
NoU = np.zeros(len(country_names))
df2 = data_df[200:top - 200]
for i, c in enumerate(country_names):
    v = df2['country'] == c
    NoU[i] = sum(v)
print(NoU)
```

```
[  1.   1.   1.   1.  18.   2.   4.   0.   0.  34.   0.   4.   2.   4.
   3.   0.  15.   6.   5.  12.  51.   3.   0.   1.  10.   1.   1.   2.
   0.  27.   6.  20.   3.   2.   5.   1.   0.  39.   6.   6.   4.   4.
   0.  39.   3.   0.   1.   8.  11.   0.   3.   0.   5. 131.   5.  37.
  44.   6.   2.]
```

In [81]:

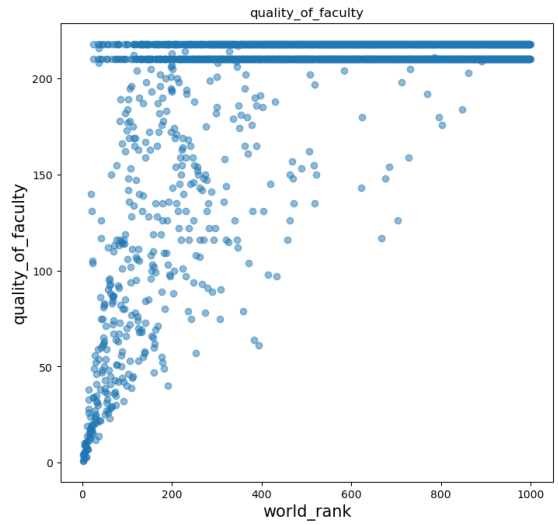
```
from pyecharts import Map
for i, c in enumerate(country_names):
    if c == 'USA':
        country_names[i] = 'United States'
        print(country_names[i])
map0 = Map("World Map", width=1200, height=600)
mapname = 'World Top {:d} Universities'.format(top)
map0.add(mapname, country_names, NoU, maptype="world",  is_visualmap=True, visual_text_color='#
000', visual_range = [0, top/10])
map0.render(path="WorldMap.html")
```

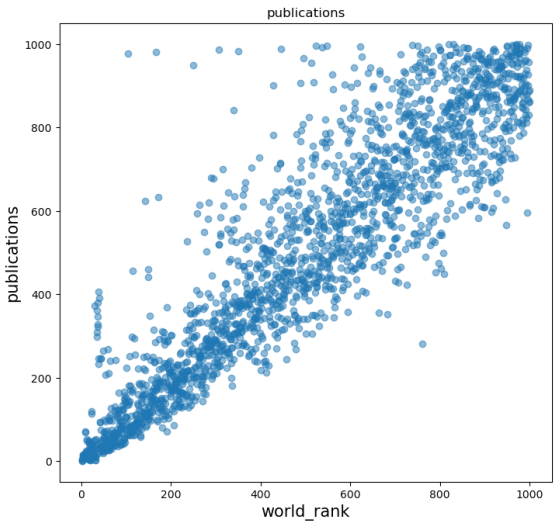
```
United States
```

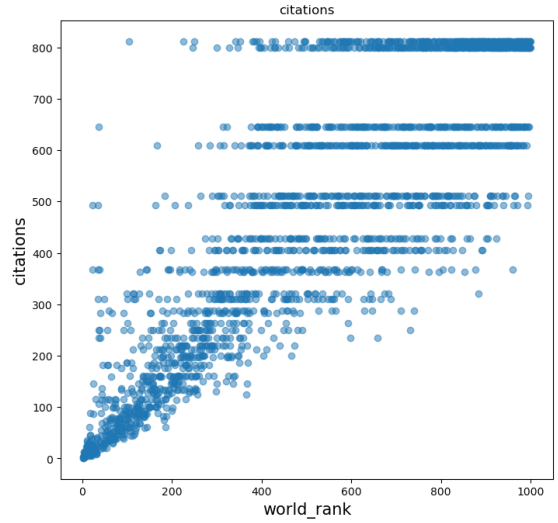
5.2 特征值可视化

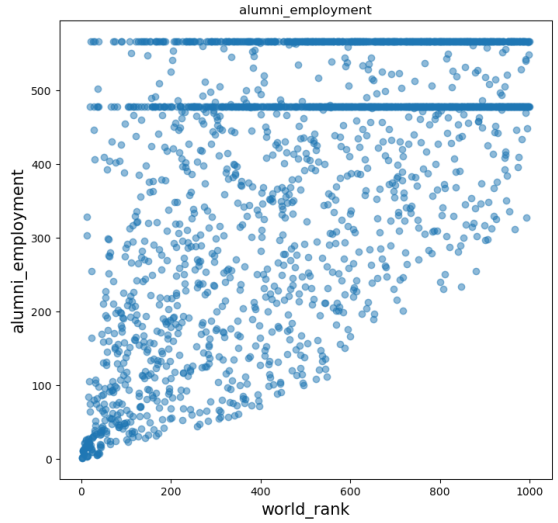
In [28]:

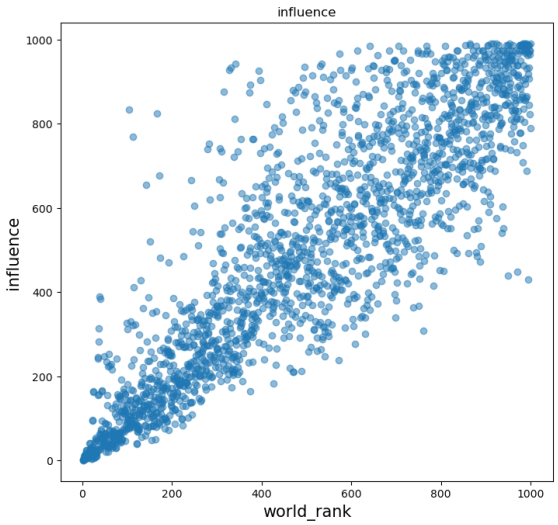
```
import matplotlib.pyplot as plt

#特征against排名
figure, axes = plt.subplots(4, 2, figsize = (18, 36), dpi = 100)
ax = axes.flatten()
for i, c in enumerate(feature_cols):
    ax[i].scatter(data_df['world_rank'], data_df[c], alpha = 0.5)
    ax[i].set_xlabel('world_rank', fontsize=15)
    ax[i].set_ylabel(c, fontsize=15)
    ax[i].set_title(c)

#plt.savefig("features.png") #保存作的图
plt.show()
```

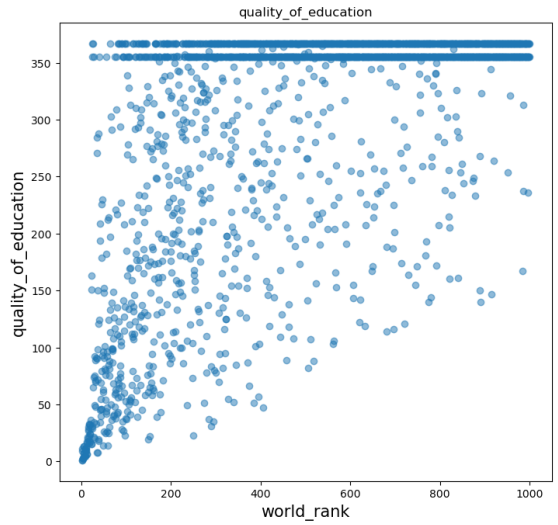

quality_of_faculty
quality_of_faculty
world_rank

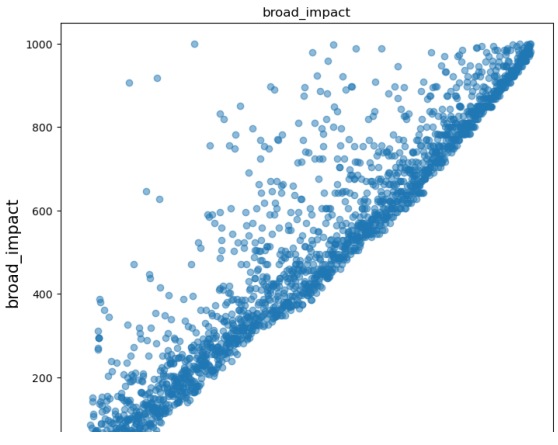

publications
publications
world_rank

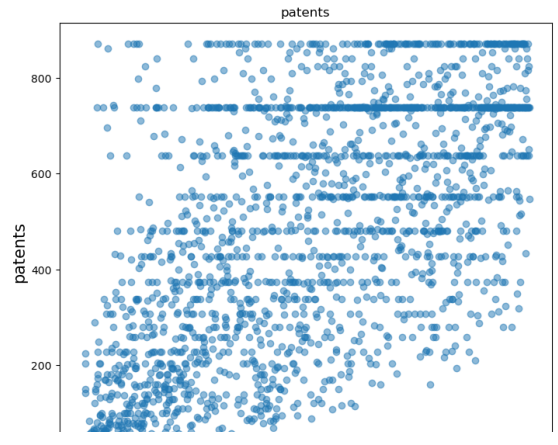

citations
citations
world_rank


alumni_employment
alumni_employment
world_rank


influence
influence
world_rank


quality_of_education
quality_of_education
world_rank


broad_impact
broad_impact


patents
patents

从图中可以发现，大多数feature和排名都是正相关的。

In [29]:

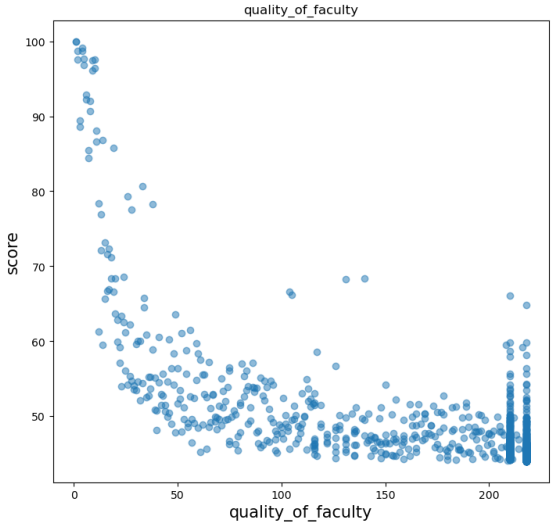
```
for i, c in enumerate(feature_cols):
    r = rcc(np.array(data_df['world_rank']), np.array(data_df[c]))
    print(c, "r = {:.4f}".format(r))
```

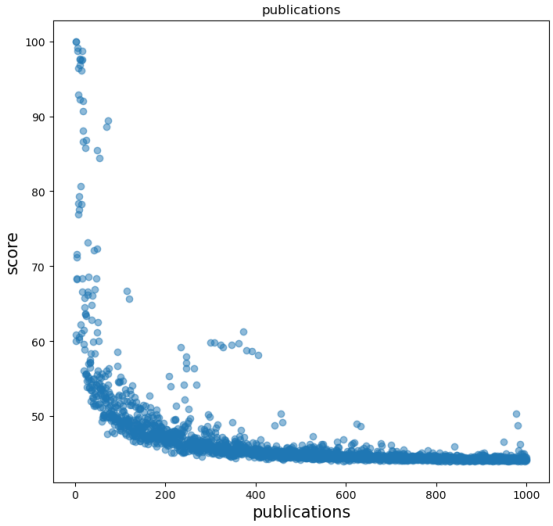
```
quality_of_faculty r = 0.5719
publications r = 0.9070
citations r = 0.8252
alumni_employment r = 0.5863
influence r = 0.8738
quality_of_education r = 0.5879
broad_impact r = 0.9435
patents r = 0.6318
```

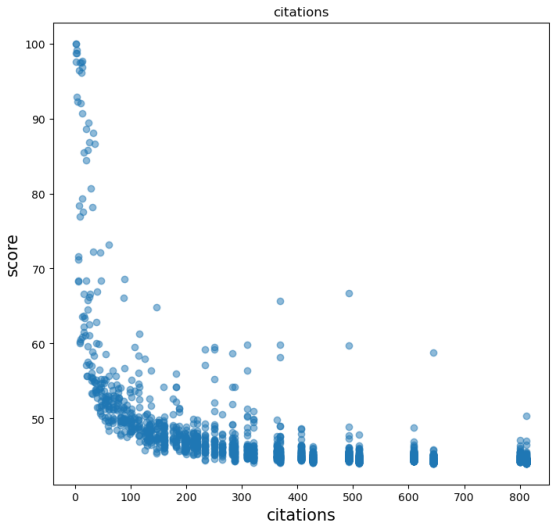
相关系数也佐证了这一点。

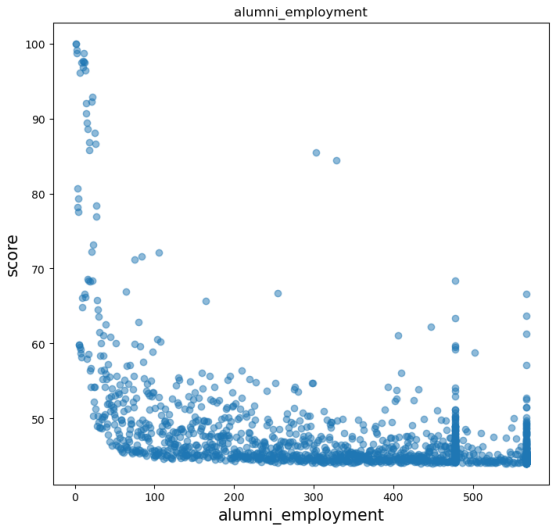
In [30]:

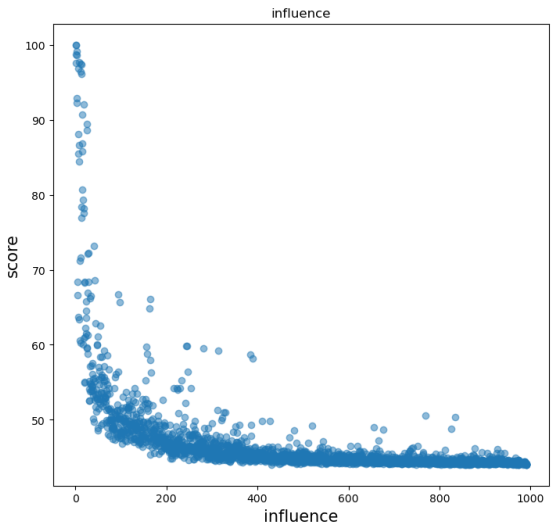
```
import matplotlib.pyplot as plt

#得分against特征值
figure, axes = plt.subplots(4, 2, figsize = (18, 36), dpi = 100)
ax = axes.flatten()
for i, c in enumerate(feature_cols):
    ax[i].scatter(data_df[c],data_df['score'], alpha = 0.5)
    ax[i].set_xlabel(c, fontsize=15)
    ax[i].set_ylabel('score', fontsize=15)
    ax[i].set_title(c)

#plt.savefig("score_against_feature.png") #保存作的图
plt.show()
```

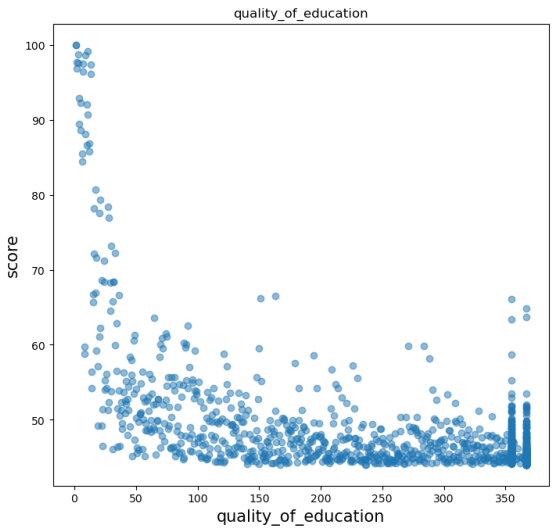

quality_of_faculty
score
quality_of_faculty

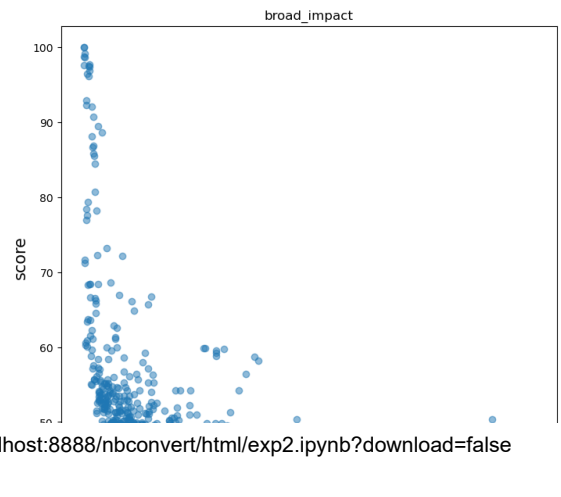

publications
score
publications

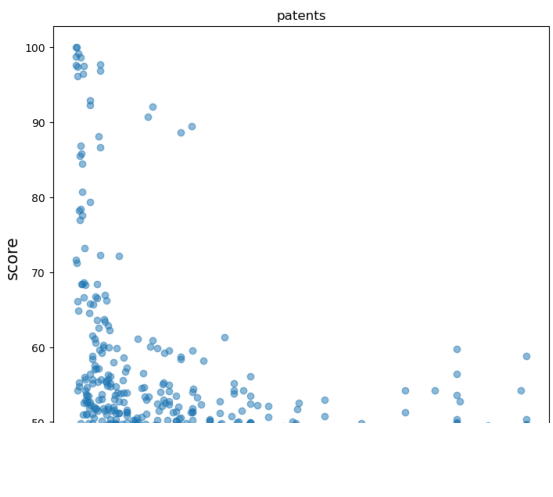

citations
score
citations

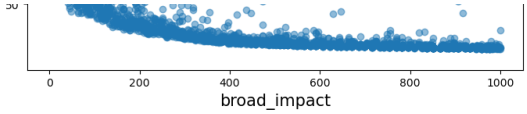

alumni_employment
score
alumni_employment

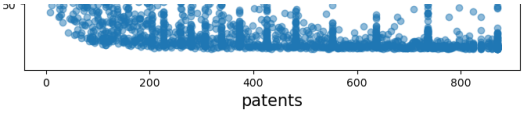

influence
score
influence


quality_of_education
score
quality_of_education


broad_impact
score


patents
score





In [31]:

```
for i, c in enumerate(feature_cols):
    r = rcc(np.array(data_df['score']), np.array(data_df[c]))
    print(c, "r = {:.4f}".format(r))
```

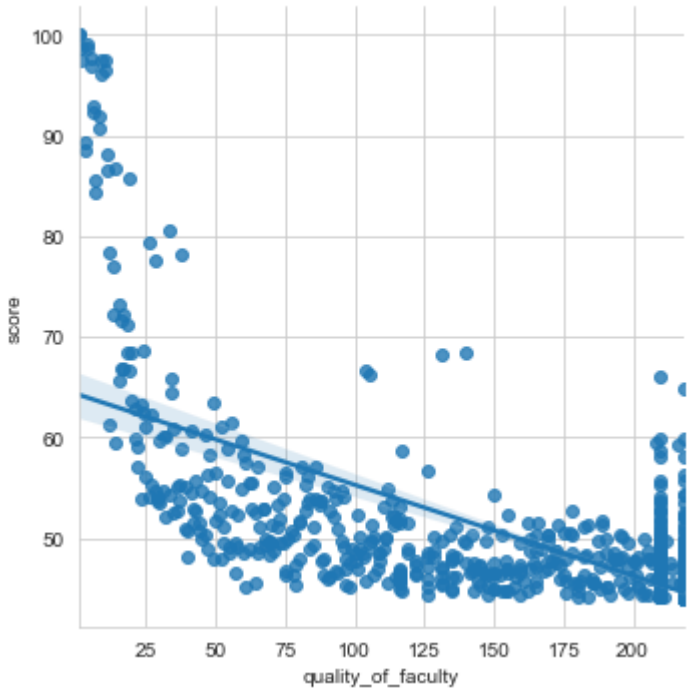
```
quality_of_faculty r = -0.7184
publications r = -0.5271
citations r = -0.5238
alumni_employment r = -0.4938
influence r = -0.5265
quality_of_education r = -0.6047
broad_impact r = -0.5319
patents r = -0.4604
```

从图中观察可知，各个feature和score都是负相关的，但是线性不强。feature的值越大，约接近线性。有点像反过来的log图。

给第一张图(quality_of_faculty)加条拟合线看看:

In [32]:

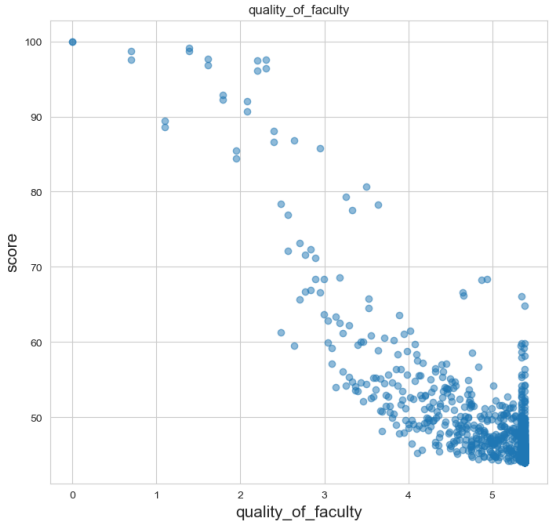
```
import seaborn as sns
sns.set_style("whitegrid")

sns.lmplot(x="quality_of_faculty", y="score", data=data_df)

plt.show()
```

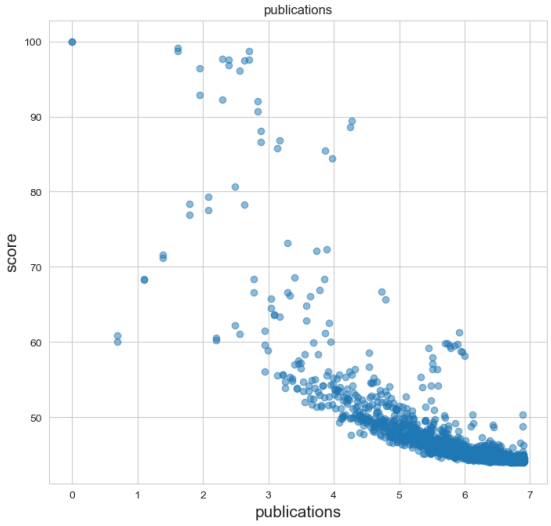


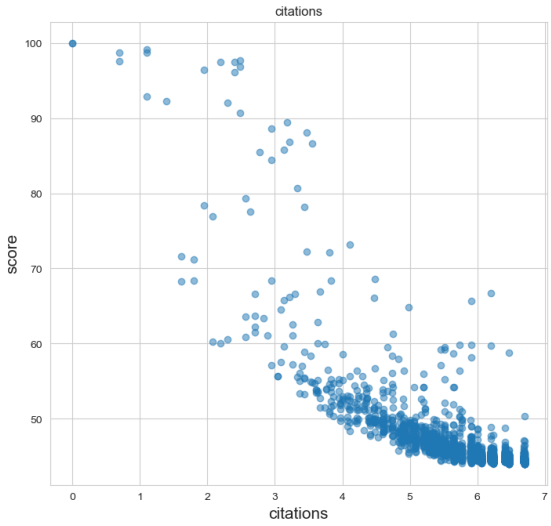
那接下来我们来看看log处理后的feature值吧。

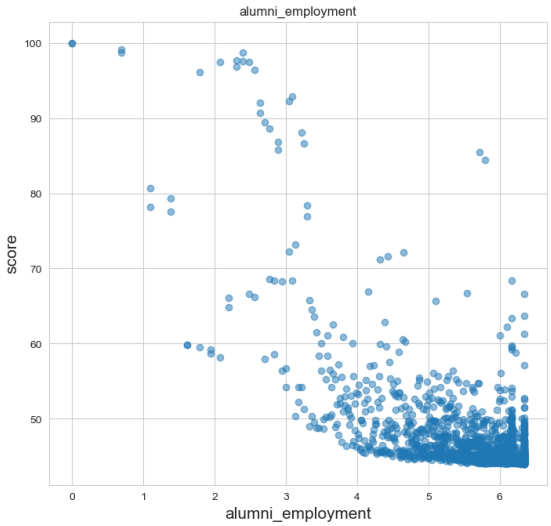
In [42]:

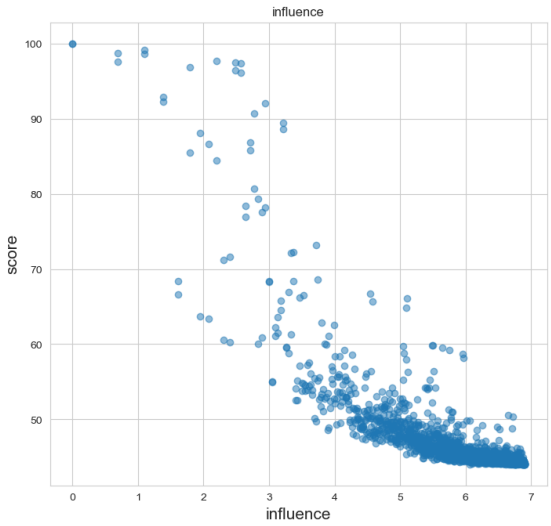
```
import matplotlib.pyplot as plt

figure, axes = plt.subplots(4, 2, figsize = (18, 36), dpi = 100)
ax = axes.flatten()
for i, c in enumerate(feature_cols):
    ax[i].scatter(np.log(X[c]),data_df['score'], alpha = 0.5)
    ax[i].set_xlabel(c, fontsize=15)
    ax[i].set_ylabel('score', fontsize=15)
    ax[i].set_title(c)

#plt.savefig("score_against_feature_log.png") #保存作的图
plt.show()
```

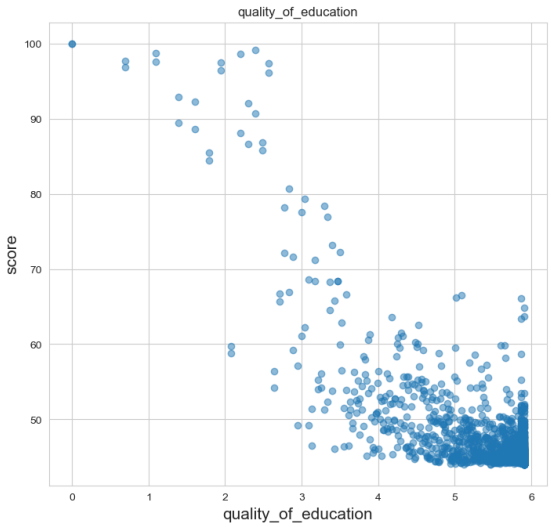

quality_of_faculty
score
quality_of_faculty

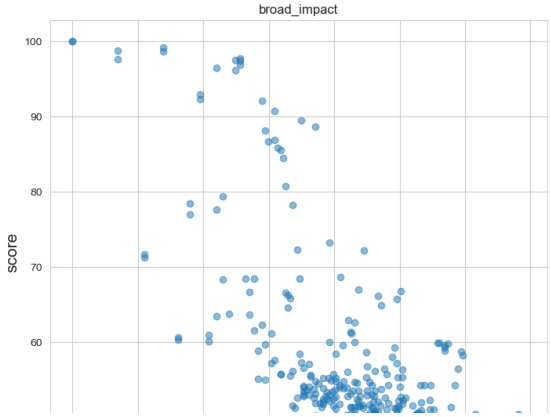

publications
score
publications

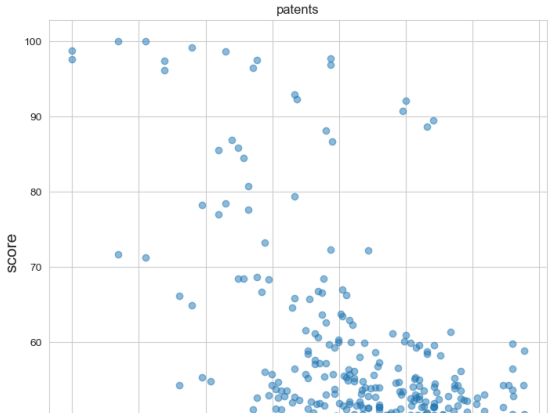

citations
score
citations

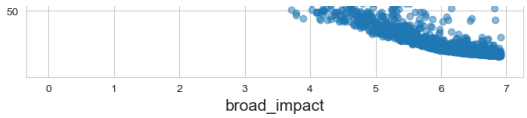

alumni_employment
score
alumni_employment

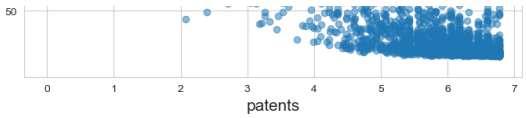

influence
score
influence


quality_of_education
score
quality_of_education


broad_impact
score


patents
score





In [43]:

```
for i, c in enumerate(feature_cols):
    r = rcc(np.array(data_df['score']), np.array(np.log(X[c])))
    print(c, "r = {:.4f}".format(r))
```

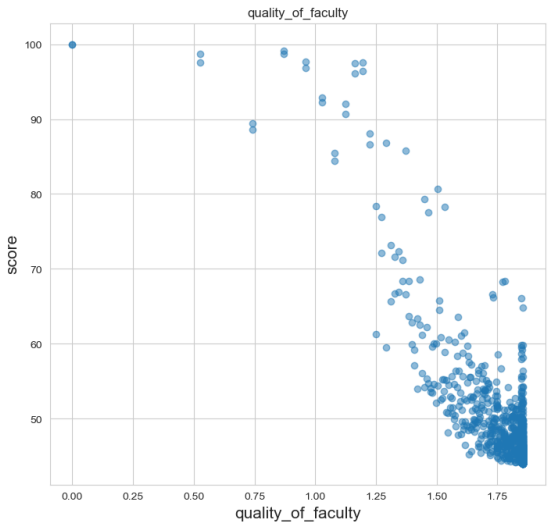
```
quality_of_faculty r = -0.8686
publications r = -0.7767
citations r = -0.7907
alumni_employment r = -0.7012
influence r = -0.8049
quality_of_education r = -0.7984
broad_impact r = -0.7880
patents r = -0.6777
```

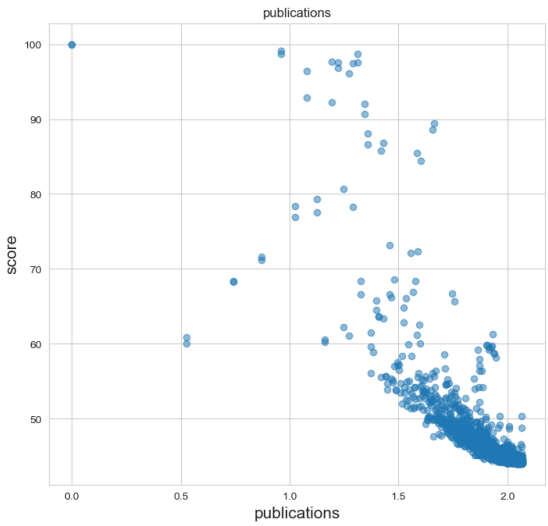
由图和相关系数可以发现处理过后的feature值和score之间的线性关系更强了。

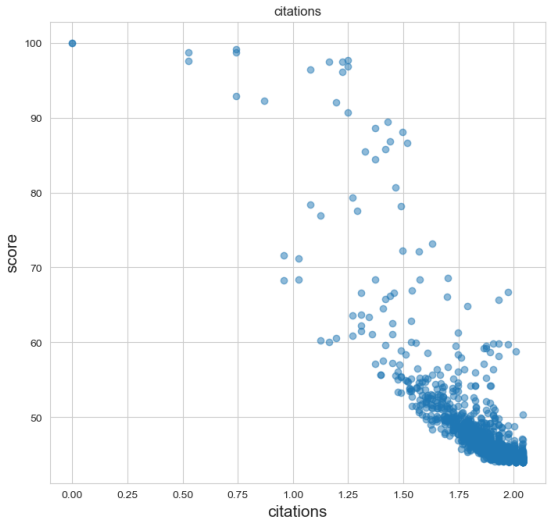
而quality_of_faculty与score之间的负线性关系是最强的，与我们之前改良后的模型是相符的。

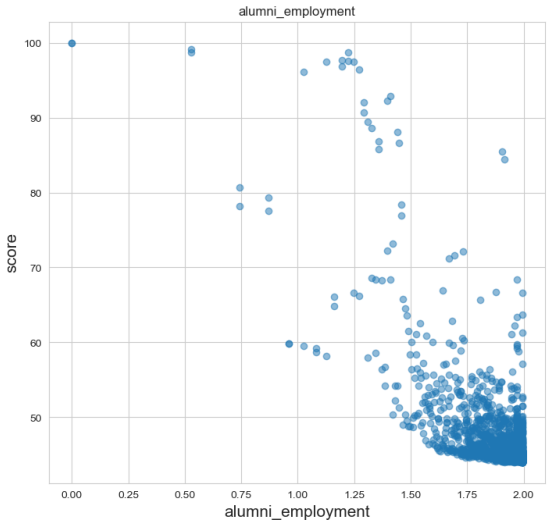
In [44]:

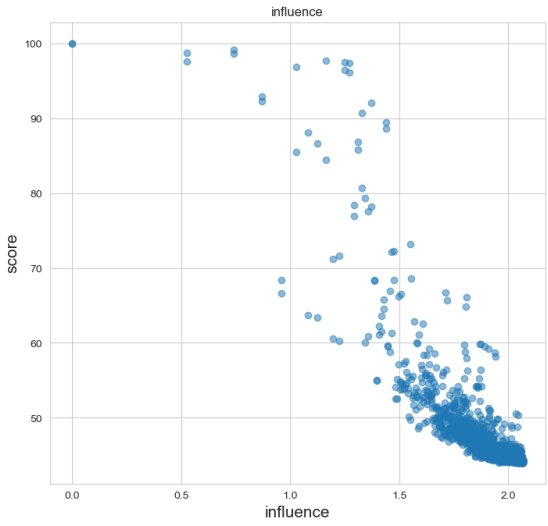
```
import matplotlib.pyplot as plt

figure, axes = plt.subplots(4, 2, figsize = (18, 36), dpi = 100)
ax = axes.flatten()
for i, c in enumerate(feature_cols):
    ax[i].scatter(X2[c],data_df['score'], alpha = 0.5)
    ax[i].set_xlabel(c, fontsize=15)
    ax[i].set_ylabel('score', fontsize=15)
    ax[i].set_title(c)

#plt.savefig("score_against_feature_log_log.png") #保存作的图
plt.show()
```

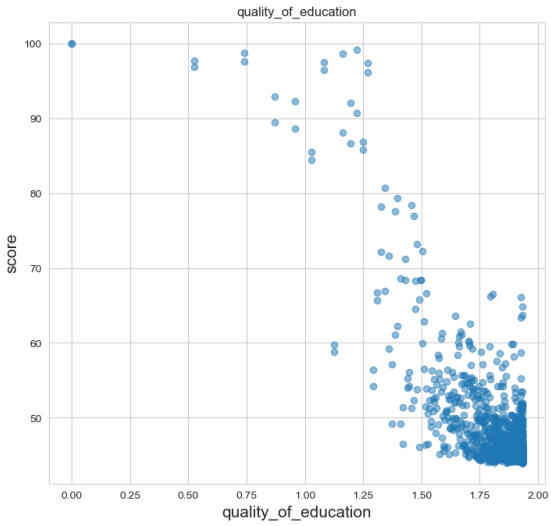

quality_of_faculty
score
quality_of_faculty

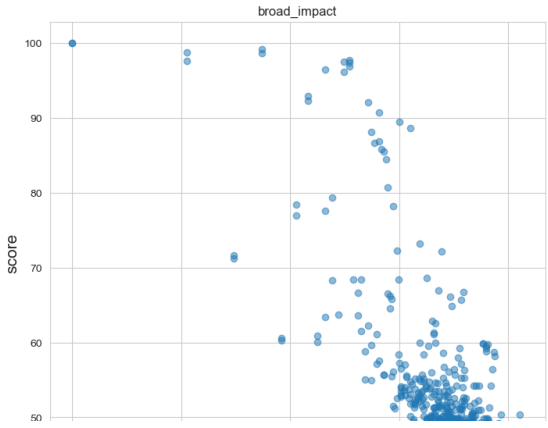

publications
score
publications

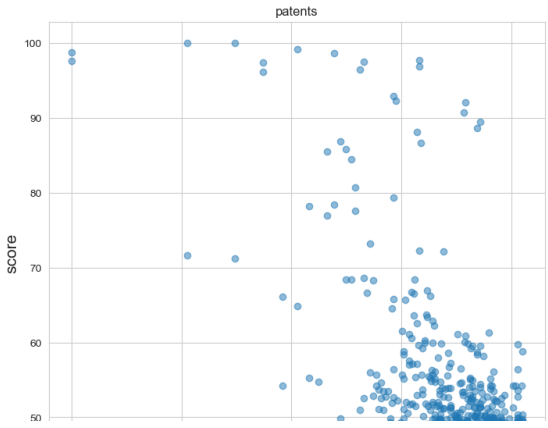

citations
score
citations


alumni_employment
score
alumni_employment


influence
score
influence


quality_of_education
score
quality_of_education


broad_impact
score


patents
score

In [45]:

```
for i, c in enumerate(feature_cols):
    r = rcc(np.array(data_df['score']), np.array(X2[c]))
    print(c, "r = {:.4f}".format(r))
```

```
quality_of_faculty r = -0.8795
publications r = -0.8122
citations r = -0.8449
alumni_employment r = -0.7458
influence r = -0.8629
quality_of_education r = -0.8364
broad_impact r = -0.8346
patents r = -0.7228
```

由图和相关系数可以发现再次log处理过后的feature值和score之间的线性关系更强了。

六、其他的回归模型

6.1 sklearn调用线性回归

使用的是优化过得特征集，可以发现结果是一样的。

In [46]:

```
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

regr = linear_model.LinearRegression()
regr.fit(train_x, train_y)
pred = regr.predict(test_x)
print('Coefficients: \n', regr.coef_)
print('Root Mean squared error: {:.4f}'.format(np.sqrt(mean_squared_error(test_y, pred))))
print('Coefficient of determination: {:.4f}'.format(r2_score(test_y, pred)))
```

```
Coefficients:
 [-15.88560002  -1.81866369  -1.33482863  -8.70003909  -6.84721445
  -7.78327306   1.17190408  -4.43475377]
Root Mean squared error: 1.5883
Coefficient of determination: 0.9344
```

6.2 RANSAC

Robust linear model estimation 忽视掉一些outlier数据

In [50]:

```
ransac = linear_model.RANSACRegressor()
ransac.fit(train_x, train_y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
pred = ransac.predict(test_x)
print("Estimated coefficients (RANSAC):")
print(ransac.estimator_.coef_)
print('Root Mean squared error: {:.4f}'.format(np.sqrt(mean_squared_error(test_y, pred))))
print('Coefficient of determination: {:.4f}'.format(r2_score(test_y, pred)))
```

```
Estimated coefficients (RANSAC):
[-1.93792258 -0.8587209  -1.10104245 -2.96879373  0.12754478 -1.92228381
 -7.96008537 -1.88272821]
Root Mean squared error: 3.8519
Coefficient of determination: 0.6140
```

In [49]:

```
ransac = linear_model.RANSACRegressor()
ransac.fit(train_x, train_y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
pred = ransac.predict(test_x)
print("Estimated coefficients (RANSAC):")
print(ransac.estimator_.coef_)
print('Root Mean squared error: {:.4f}'.format(np.sqrt(mean_squared_error(test_y, pred))))
print('Coefficient of determination: {:.4f}'.format(r2_score(test_y, pred)))
```

```
Estimated coefficients (RANSAC):
[-26.39746103  -2.0253391    0.04973992  -2.98036492  -0.15709491
  -0.72643539  -8.42402428  -1.77578437]
Root Mean squared error: 2.0071
Coefficient of determination: 0.8952
```

发现每一次RANSAC训练出来的model都是不一样的，可能的原因是其处理outlier的方式不同。

但由于我们的数据集中outlier其实是很少的（或者说应该没有），所以这个model并不适合。

6.3 SGDregression

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

In [51]:

```
model = linear_model.SGDRegressor(loss='squared_loss')
model.fit(train_x, train_y)
pred = model.predict(test_x)
print('Coefficients: \n', model.coef_)
print('Root Mean squared error: {:.4f}'.format(np.sqrt(mean_squared_error(test_y, pred))))
print('Coefficient of determination: {:.4f}'.format(r2_score(test_y, pred)))
```

```
Coefficients:
 [ -4.92900061   0.65909611  -2.89944401  -7.39733051 -11.79513618
 -10.40668586   0.98874031  -2.60706213]
Root Mean squared error: 1.9706
Coefficient of determination: 0.8990
```

用梯度下降法，损失函数为误差平方和，得到的结果和之前的线性回归是类似的。（略差一点）

6.4 决策树

In [52]:

```
from sklearn import tree
model = tree.DecisionTreeRegressor()
model.fit(train_x, train_y)
p = model.predict(test_x)
pred = model.predict(test_x)
print('Root Mean squared error: {:.4f}'.format(np.sqrt(mean_squared_error(test_y, pred))))
print('Coefficient of determination: {:.4f}'.format(r2_score(test_y, pred)))
```

```
Root Mean squared error: 0.7423
Coefficient of determination: 0.9857
```

震惊!决策树的效果比线性回归好!

6.5 随机森林

In [53]:

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(train_x, train_y)
p = model.predict(test_x)
pred = model.predict(test_x)
print('Root Mean squared error: {:.4f}'.format(np.sqrt(mean_squared_error(test_y, pred))))
print('Coefficient of determination: {:.4f}'.format(r2_score(test_y, pred)))
```

```
Root Mean squared error: 0.6022
Coefficient of determination: 0.9906
```

随机森林的效果比决策树更好诶，ensemble learning真强!

6.6 神经网络

In [57]:

```
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(learning_rate_init=0.01)
model.fit(train_x, train_y)
p = model.predict(test_x)
pred = model.predict(test_x)
print('Root Mean squared error: {:.4f}'.format(np.sqrt(mean_squared_error(test_y, pred))))
print('Coefficient of determination: {:.4f}'.format(r2_score(test_y, pred)))
```

```
Root Mean squared error: 1.3193
Coefficient of determination: 0.9547
```

修改了学习率后的神经网络效果不错。

七、加入离散国家特征

In [58]:

```
from sklearn.preprocessing import MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder
country_names = set(data_df['country'])
#生成一个有所有国家名的2d array
d2names = []
for c in country_names:
    newc =  [c]
    d2names.append(newc)

#制作onehotencoder
enc = OneHotEncoder()
enc.fit(d2names)

#将country一列转化为2d array
x = data_df['country']
d2x = []
for c in x:
    newx = [c]
    d2x.append(newx)

#生成数据
xcountry = enc.transform(d2x).toarray()

#乘上national_rank的log值
for i, c in enumerate(xcountry):
    num = data_df['national_rank'][200+i]
    xcountry[i] = xcountry[i] * math.log(num)
```

In [59]:

```
xcountry[1200]
```

Out[59]:

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 2.77258872, 0.        ])
```

In [60]:

```
all_x2 = deepcopy(all_x)
print(all_x2[1])
for i, c in enumerate(all_x2):
    for j, num in enumerate(c):
        all_x2[i][j] = np.log(np.log(num)+1)

print(all_x2[1])
all_x3 = np.c_[all_x2, xcountry]
all_x3.shape
```

```
[ 4.  5.  3.  2.  3. 11.  4.  6.]
[0.86974169 0.95913484 0.74127631 0.52658903 0.74127631 1.2231562
 0.86974169 1.02667203]
```

Out[60]:

```
(2000, 67)
```

In [61]:

```
train_x, test_x, train_y, test_y = train_test_split(all_x3, all_y, test_size = 0.2, random_state
= 2020)

LR = LinearRegression()
LR.fit(train_x, train_y)
p = LR.predict(test_x)
RMSE = rmse(test_y, p)
print("RMSE:", RMSE)
```

```
RMSE: 1.5574482028457648
```

RMSE值为2.044，比之前的模型的值更小了。

In [62]:

```
from copy import deepcopy
b = deepcopy(LR.B)
features = ['y-intercept']
for f in feature_names:
    features.append(f)
for c in country_names:
    features.append(c)

print(features)
print(b)

for i in range(len(b)):
    for j in range(len(b) - i - 1):
        if b[j]>b[j+1]:
            b[j], b[j+1] = b[j+1], b[j]
            features[j], features[j+1] = features[j+1], features[j]

for i in range(len(b)):
        print("{}:{:.5f}".format(features[i], b[i]))
```

```
['y-intercept', 'quality_of_faculty', 'publications', 'citations', 'alumni_employm
ent', 'influence', 'quality_of_education', 'broad_impact', 'patents', 'Iran', 'Ice
land', 'Slovak Republic', 'Colombia', 'Canada', 'Argentina', 'Norway', 'Romania',
'Bulgaria', 'France', 'Lithuania', 'Ireland', 'Mexico', 'Hungary', 'Switzerland',
'Egypt', 'Australia', 'Netherlands', 'New Zealand', 'India', 'Japan', 'Malaysia',
'Serbia', 'Slovenia', 'Austria', 'Estonia', 'Croatia', 'Denmark', 'Singapore', 'Sp
ain', 'Portugal', 'South Korea', 'Czech Republic', 'Thailand', 'Hong Kong', 'Leban
on', 'Puerto Rico', 'Italy', 'Belgium', 'Greece', 'South Africa', 'Poland', 'Unite
d Arab Emirates', 'China', 'Israel', 'Uganda', 'Russia', 'Finland', 'Taiwan', 'Uru
guay', 'Chile', 'Cyprus', 'Turkey', 'USA', 'Sweden', 'Germany', 'United Kingdom',
'Brazil', 'Saudi Arabia']
[ 1.32619190e+02 -1.46753553e+01 -2.68540954e-01 -1.88292125e+00
 -8.97270529e+00 -8.69769830e+00 -8.03775448e+00  1.29639341e+00
 -4.47825196e+00  4.00151751e-01 -6.63289324e-02  5.59681721e-02
 -3.37174128e-01  1.82120029e-01 -3.30716196e-15 -3.84460124e-04
 -1.29215199e-01  8.05756507e-02 -2.46138337e-01 -3.02411501e-15
  8.56868288e-15  3.11291939e-01 -4.67547953e-02  4.99470380e-01
  2.15510290e-15 -1.68927719e-02 -4.54652325e-02 -1.05442627e-01
  4.53470419e-01  2.25202167e-01  2.34872862e-01 -6.86837431e-15
  3.47445520e-02  4.46929192e-01  4.65539092e-02 -1.51939310e+00
  1.45491069e-01 -3.64683139e-02  5.66625436e-16  4.76490019e-15
 -7.53791426e-01 -2.51111122e-01  2.49977528e-01 -2.19130185e-01
 -1.83316874e-01  2.51039128e-02  8.21093552e-02 -2.25162733e-15
  1.09309144e+00 -1.11396437e+00 -4.82961151e-01 -1.98509034e-17
  3.06346499e+00 -3.91095877e-16  8.59864255e-01 -1.09004842e+00
  1.79326812e-02  8.10189515e-02 -3.47594095e-01 -6.98538675e-01
  2.15479246e-01  3.86495206e-01  3.37752111e-01 -7.62482588e-02
  0.00000000e+00  0.00000000e+00 -1.53097120e-01  1.48853988e-01]
quality_of_faculty:-14.67536
alumni_employment:-8.97271
influence:-8.69770
quality_of_education:-8.03775
patents:-4.47825
citations:-1.88292
Croatia:-1.51939
South Africa:-1.11396
Russia:-1.09005
South Korea:-0.75379
Chile:-0.69854
Poland:-0.48296
Uruguay:-0.34759
Colombia:-0.33717
publications:-0.26854
Czech Republic:-0.25111
France:-0.24614
Hong Kong:-0.21913
Lebanon:-0.18332
Brazil:-0.15310
Romania:-0.12922
New Zealand:-0.10544
Sweden:-0.07625
Iceland:-0.06633
Hungary:-0.04675
Netherlands:-0.04547
Singapore:-0.03647
Australia:-0.01689
Norway:-0.00038
Serbia:-0.00000
Argentina:-0.00000
Lithuania:-0.00000
Belgium:-0.00000
```

```
Israel:-0.00000
United Arab Emirates:-0.00000
Germany:0.00000
United Kingdom:0.00000
Spain:0.00000
Egypt:0.00000
Portugal:0.00000
Ireland:0.00000
Finland:0.01793
Puerto Rico:0.02510
Slovenia:0.03474
Estonia:0.04655
Slovak Republic:0.05597
Bulgaria:0.08058
Taiwan:0.08102
Italy:0.08211
Denmark:0.14549
Saudi Arabia:0.14885
Canada:0.18212
Cyprus:0.21548
Japan:0.22520
Malaysia:0.23487
Thailand:0.24998
Mexico:0.31129
USA:0.33775
Turkey:0.38650
Iran:0.40015
Austria:0.44693
India:0.45347
Switzerland:0.49947
Uganda:0.85986
Greece:1.09309
broad_impact:1.29639
China:3.06346
y-intercept:132.61919
```

In [63]:

```
VAR = var(test_y)
MSE = mse(test_y, p)
COD = cod(test_y, p)
print("var:{:.4f} mse:{:.4f}".format(VAR, MSE))
print("codfficient of determination:{:.4f}".format(COD))
```

```
var:38.4352 mse:2.4256
codfficient of determination:0.9369
```

误差更小了，对数据的解释程度更好了。

由此可见，这个ranking的评分对国家有偏好。在某一些国家，可能会获得特别的加分。

但是，这也存在过拟合的可能性，因为国家的分类太多了，而每个国家入选的学校并不一定很多。

八、总结

在基于回归分析的大学综合得分预测这个案例中，学习并应用了线性回归的模型，体会了使用log、归一化、标准化对特征值处理的效果，感受了不同回归模型带来的不同效果，尝试了用onehot对非数值特征进行处理。收获颇丰。