查看、准备数据

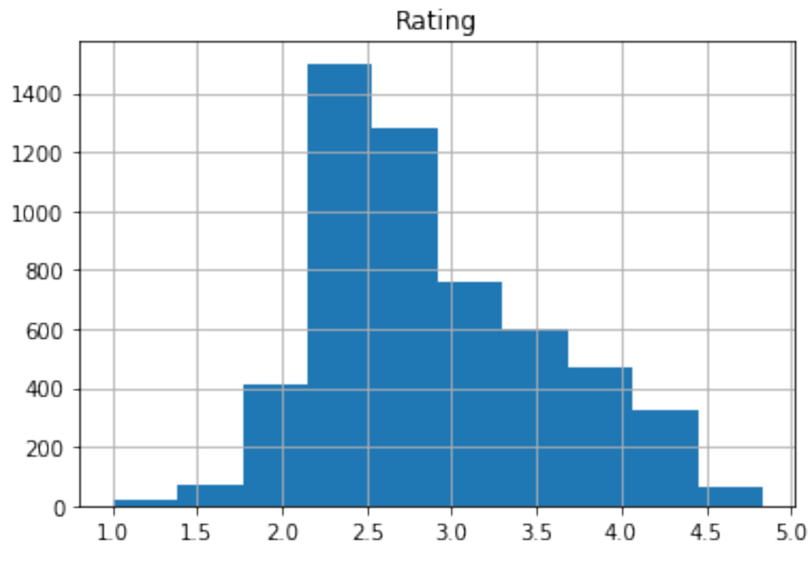
In [1]:
```
# Import essentail packages for image processing
import pandas as pd
import numpy as np
import cv2
from PIL import Image

#读取原有数据
MasterFile=pd.read_csv('/clubear/Lecture 2.1 - Linear Regression
by TensorFlow/data/faces/FaceScore.csv')
#查看数据
MasterFile[0:5]
```

Out[1]:

	Filename	Rating
0	ftw1.jpg	4.083333
1	ftw10.jpg	3.666667
2	ftw100.jpg	1.916667
3	ftw101.jpg	2.416667
4	ftw102.jpg	3.166667

In [2]:
```
MasterFile.hist()
```

Out[2]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f7f
1b6b8d50>]],
      dtype=object)
```



准备数据，将其变化为非线性回归问题

In [3]:
```
FileNames=MasterFile['Filename']
N=len(FileNames)
IMSIZE=128
X=np.zeros([N,IMSIZE,IMSIZE,3]) #画布，将要添加N列内含128列 128*3 的array
for i in range(N):
    MyFile=FileNames[i] #读取第i文件名称
    Im=Image.open('/clubear/Lecture 2.1 - Linear Regression by TensorFlow/data/faces/images/'+MyFile) #通过地址读取图片
    Im=Im.resize([IMSIZE,IMSIZE]) #使所有图片规整，方便看
    Im=np.array(Im)/255 #变成float32格式
    X[i,]=Im

#一个将faces里所有脸图片对应的array添加到X上的loop
Y=np.array(MasterFile['Rating']).reshape([N,1]) #将Y变为N*1的vector，每一行代表每一张图的打分
Y=(Y-np.mean(Y))/np.std(Y) #标准化Y
```

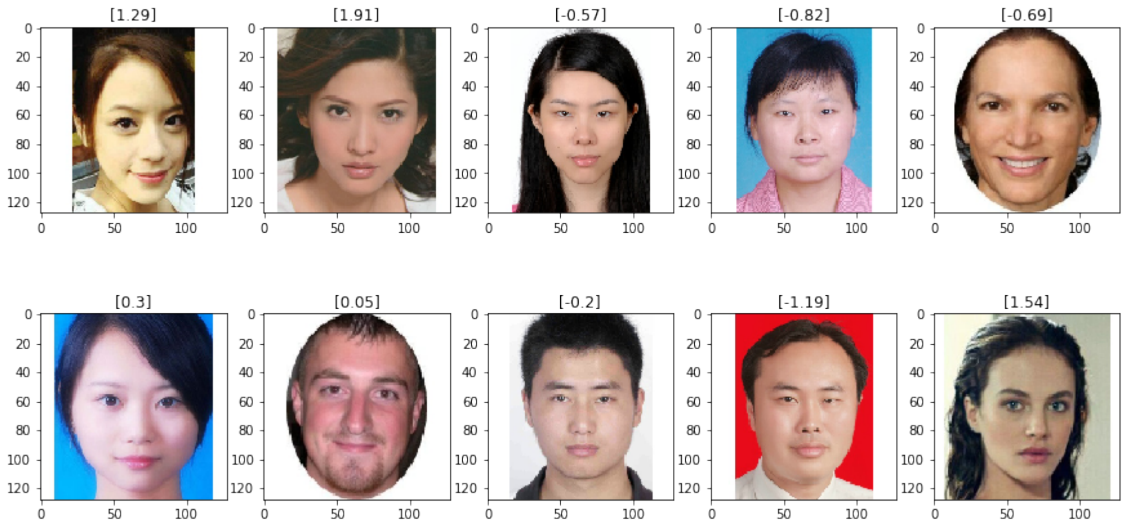
数据切分

In [4]:
```
from sklearn.model_selection import train_test_split
X0,X1,Y0,Y1=train_test_split(X,Y,test_size=0.3,random_state=233)
#将X，Y以7:3的比例分为train data和test data，保存这个seed保证下次还可以用这个dataset
```

颜值数据展示

In [5]:
```
from matplotlib import pyplot as plt
plt.figure()
fig,ax=plt.subplots(2,5)
fig.set_figheight(7.5)
fig.set_figwidth(15)
ax=ax.flatten() #把所有图片数据一行展示出来
for i in range(10):
    ax[i].imshow(X0[i,:,:,:])
    ax[i].set_title(np.round(Y0[i],2))
```

```
/root/miniconda3/envs/myconda/lib/python3.7/site-packages/matpl
otlib/text.py:1150: FutureWarning: elementwise comparison faile
d; returning scalar instead, but in the future will perform ele
mentwise comparison
  if s != self._text:

<Figure size 432x288 with 0 Axes>
```



搭建模型

In [6]:
```
from keras.layers import Dense, Flatten, Input
from keras import Model
from keras.applications import ResNet50


input_layer=Input([IMSIZE,IMSIZE,3])
x=input_layer
x=Flatten()(x)
x=Dense(1)(x)
output_layer=x
model=Model(input_layer,output_layer) #这个model使（N, 128, 128, 3
）的input经过卷积后压扁（（128, 128, 3） apply不同的
#weights到49152个nodes）、再全连接层（49152个nodes apply不同的weight
s到1个node）得到颜值打分。
model.summary()
```

```
Using TensorFlow backend.

Model: "model_1"
_________________________________________________________________
__
Layer (type)                 Output Shape              Param #
=================================================================
==
input_1 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
__
flatten_1 (Flatten)          (None, 49152)             0
_________________________________________________________________
__
dense_1 (Dense)              (None, 1)                 49153
=================================================================
==
Total params: 49,153
Trainable params: 49,153
Non-trainable params: 0
_________________________________________________________________
__
```

In [7]:
```
from keras.optimizers import Adam
model.compile(loss='mse',optimizer=Adam(lr=0.01),metrics=['mse']
) #选择恰当的loss function去判定这个model的好坏，使用Adam作为minimize
loss的方法
```

In [10]:
```
model.fit(X0,Y0,
          validation_data=[X1,Y1],
          batch_size=100,
          epochs=20)
#将training data feed into the model，并用validation data检测loss
```

```
Train on 3850 samples, validate on 1650 samples
Epoch 1/20
3850/3850 [==============================] - 1s 270us/step - lo
ss: 0.7196 - mse: 0.7196 - val_loss: 0.6925 - val_mse: 0.6925
```

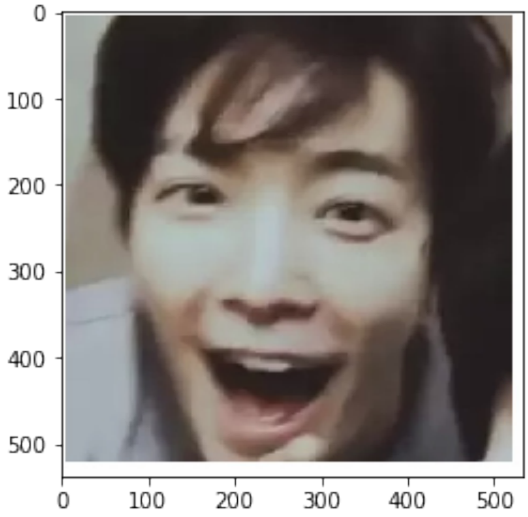
```
Epoch 2/20
3850/3850 [==============================] - 1s 357us/step - lo
ss: 0.7307 - mse: 0.7307 - val_loss: 0.6963 - val_mse: 0.6963
Epoch 3/20
3850/3850 [==============================] - 1s 280us/step - lo
ss: 0.6986 - mse: 0.6986 - val_loss: 0.7064 - val_mse: 0.7064
Epoch 4/20
3850/3850 [==============================] - 1s 286us/step - lo
ss: 0.7422 - mse: 0.7422 - val_loss: 0.8239 - val_mse: 0.8239
Epoch 5/20
3850/3850 [==============================] - 1s 319us/step - lo
ss: 0.7741 - mse: 0.7741 - val_loss: 0.9416 - val_mse: 0.9416
Epoch 6/20
3850/3850 [==============================] - 1s 305us/step - lo
ss: 0.6950 - mse: 0.6950 - val_loss: 0.7142 - val_mse: 0.7142
Epoch 7/20
3850/3850 [==============================] - 1s 327us/step - lo
ss: 0.6964 - mse: 0.6964 - val_loss: 0.6510 - val_mse: 0.6510
Epoch 8/20
3850/3850 [==============================] - 1s 315us/step - lo
ss: 0.7406 - mse: 0.7406 - val_loss: 0.6658 - val_mse: 0.6658
Epoch 9/20
3850/3850 [==============================] - 1s 341us/step - lo
ss: 0.6675 - mse: 0.6675 - val_loss: 0.6385 - val_mse: 0.6385
Epoch 10/20
3850/3850 [==============================] - 1s 335us/step - lo
ss: 0.6610 - mse: 0.6610 - val_loss: 0.7697 - val_mse: 0.7697
Epoch 11/20
3850/3850 [==============================] - 1s 338us/step - lo
ss: 0.6750 - mse: 0.6750 - val_loss: 0.6322 - val_mse: 0.6322
Epoch 12/20
3850/3850 [==============================] - 1s 328us/step - lo
ss: 0.7578 - mse: 0.7578 - val_loss: 0.9392 - val_mse: 0.9392
Epoch 13/20
3850/3850 [==============================] - 1s 308us/step - lo
ss: 0.6560 - mse: 0.6560 - val_loss: 0.7407 - val_mse: 0.7407
Epoch 14/20
3850/3850 [==============================] - 1s 321us/step - lo
ss: 0.6129 - mse: 0.6129 - val_loss: 1.2802 - val_mse: 1.2802
Epoch 15/20
3850/3850 [==============================] - 1s 306us/step - lo
ss: 0.7548 - mse: 0.7548 - val_loss: 0.6334 - val_mse: 0.6334
Epoch 16/20
3850/3850 [==============================] - 1s 298us/step - lo
ss: 0.6445 - mse: 0.6445 - val_loss: 1.1919 - val_mse: 1.1919
Epoch 17/20
3850/3850 [==============================] - 1s 319us/step - lo
ss: 0.9003 - mse: 0.9003 - val_loss: 0.8278 - val_mse: 0.8278
Epoch 18/20
3850/3850 [==============================] - 1s 310us/step - lo
ss: 0.6236 - mse: 0.6236 - val_loss: 0.6692 - val_mse: 0.6692
Epoch 19/20
3850/3850 [==============================] - 1s 317us/step - lo
ss: 0.6603 - mse: 0.6603 - val_loss: 0.6340 - val_mse: 0.6340
Epoch 20/20
```

```
3850/3850 [==============================] - 1s 286us/step - lo
ss: 0.7527 - mse: 0.7527 - val_loss: 1.0366 - val_mse: 1.0366
```

Out[10]: <keras.callbacks.callbacks.History at 0x7f7db006dad0>

In [11]:
```
MyPic=Image.open('DH.jpg')
plt.imshow(MyPic)
MyPic=MyPic.resize((IMSIZE,IMSIZE))
MyPic=np.array(MyPic)/255
MyPic=MyPic.reshape((1,IMSIZE,IMSIZE,3))      #
model.predict(MyPic)
```

Out[11]: array([[-1.0833082]], dtype=float32)



In [ ]: