HA #1

A couple of history

The dataset comes from a 1994 study about the population biology of blacklip abalone (Haliotis rubra) from the North Coast and Islands of Bass Strait. It was published to the UCI Machine Learning Repository in 1995 by Sam Waugh from the Department of Computer Science at the University of Tasmania (Australia). The original dataset was modified by removing objects with missing values and scaling continuous values by dividing by 200.

Relevant information paragraph

The general problem of the dataset is the age (and thus count of rings) prediction of the abalone using its phisical measurements. In order to get the age of abalone the scientist should cut the shell through the cone at first, stain it and then count up its rings using microscope. Thereby it is time-consuming and definitely dull task. In that case the problem is sufficiently substantial. So, the principal reason I choose this dataset is its striking canonicity. Saying that I mean there are both nominal and real valued features, the amount of features and objects is good enought and, in addition, there is absence of missing data. Thus, the abalone dataset is highly prepared and comfortable for educational purpose. So, I hope I will be enjoy working on that though my Core Data Analysis course. It is clear. It is pleasant to calculate. So I suppose it is like Dostoevsky in Russian literature or Van Gogh art, it is a kind of Computer Science genre classics. That is why I choose the ablone dataset.

Explanation of the data (Attributes)

There are 8 features. Next I will descrbe them all with its accepted values. Sex is the nominal feature with the value from the set {M, F, I(infant)}. All next features were scaled by dividing by 200. Length feature is the longest shell measurement which takes values from the interval (0.075, 0.815). Diameter is a continuous feature which is perpendicular to length. Its values are from 0.055 to 0.650. Height is the vertical shell measurement with meat in particular. Whole weight (whole_w in table) is an abalon's full weight with the value from 0.002 to 2.826. Cognitively to note that the black abalone has sufficiently large mass. Sometimes it reaches up to 565 grams and more. Shucked weight is the meat mass measurement (without the shell, which is logical). Shucked_w's values are between 0.001 and 1.488 (which is 0.2 and 297.6 grams, respectively, in original dataset). The Viscera weight comes after the bleeding, so it is a gut weight taking values from 0.001 to 0.760 (grams dividing by 200). Next measurement is a shell mass after being dried, which is important to note, because of the fact that similar abalones can collect significantly different amount of water. And the target variable is a rings count what refers to the age by adding 1.5 year for each ring.

There are 4177 instances with 8 features

The dataset is located in "abalone.csv" file. Each string means 1 object with features separating by "," symbol.

Statistics additionals

The following table shows some statistic information like mean, standart diviation and correlation with the target variable over all features.

```
            Length    Diam      Height    Whole     Shucked   Viscera   Shell
    Rings
    Mean   0.524    0.408    0.140    0.829    0.359    0.181    0.239
    9.934
    SD     0.120    0.099    0.042    0.490    0.222    0.110    0.139
    3.224
    Correl 0.557    0.575    0.557    0.540    0.421    0.504    0.628
    1.0
```

Finally, a few words about the **technologies** I use.

Core language is python v3.4. Pandas is the python's module for visualization and simple data processing. I use numpy for comfortable matrix operations and matplotlib for a graphical representation of data. In the end, sklearn python module is a powerful set of the most common machine learning tackle including the regression.

Next, I will try to comment on the nontrivial areas to clarify the situation.

In [1]:

```python
import sys
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from random import randint
%matplotlib inline
from sklearn.model_selection import KFold
```

In [4]:

```python
pass_data = pd.read_csv('abalone.csv', header=None)
pass_data.columns = ['Sex', 'Length', 'Diameter', 'Height', 'Whole_w', 'Shu
cked_w', 'Viscera_w', 'Shell_w', 'Rings']
pass_data #there is data sample
```

Out[4]:

	Sex	Length	Diameter	Height	Whole_w	Shucked_w	Viscera_w	Shell_w	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
5	I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.1200	8
6	F	0.530	0.415	0.150	0.7775	0.2370	0.1415	0.3300	20
7	F	0.545	0.425	0.125	0.7680	0.2940	0.1495	0.2600	16
8	M	0.475	0.370	0.125	0.5095	0.2165	0.1125	0.1650	9
9	F	0.550	0.440	0.150	0.8945	0.3145	0.1510	0.3200	19
10	F	0.525	0.380	0.140	0.6065	0.1940	0.1475	0.2100	14

	Sex	Length	Diameter	Height	Whole_w	Shucked_w	Viscera_w	Shell_w	Rings
11	M	0.430	0.350	0.110	0.4060	0.1675	0.0810	0.1350	10
12	M	0.490	0.380	0.135	0.5415	0.2175	0.0950	0.1900	11
13	F	0.535	0.405	0.145	0.6845	0.2725	0.1710	0.2050	10
14	F	0.470	0.355	0.100	0.4755	0.1675	0.0805	0.1850	10
15	M	0.500	0.400	0.130	0.6645	0.2580	0.1330	0.2400	12
16	I	0.355	0.280	0.085	0.2905	0.0950	0.0395	0.1150	7
17	F	0.440	0.340	0.100	0.4510	0.1880	0.0870	0.1300	10
18	M	0.365	0.295	0.080	0.2555	0.0970	0.0430	0.1000	7
19	M	0.450	0.320	0.100	0.3810	0.1705	0.0750	0.1150	9
20	M	0.355	0.280	0.095	0.2455	0.0955	0.0620	0.0750	11
21	I	0.380	0.275	0.100	0.2255	0.0800	0.0490	0.0850	10
22	F	0.565	0.440	0.155	0.9395	0.4275	0.2140	0.2700	12
23	F	0.550	0.415	0.135	0.7635	0.3180	0.2100	0.2000	9
24	F	0.615	0.480	0.165	1.1615	0.5130	0.3010	0.3050	10
25	F	0.560	0.440	0.140	0.9285	0.3825	0.1880	0.3000	11
26	F	0.580	0.450	0.185	0.9955	0.3945	0.2720	0.2850	11
27	M	0.590	0.445	0.140	0.9310	0.3560	0.2340	0.2800	12
28	M	0.605	0.475	0.180	0.9365	0.3940	0.2190	0.2950	15
29	M	0.575	0.425	0.140	0.8635	0.3930	0.2270	0.2000	11
...	...	...	...	...	...	...	...	...	...
4147	M	0.695	0.550	0.195	1.6645	0.7270	0.3600	0.4450	11
4148	M	0.770	0.605	0.175	2.0505	0.8005	0.5260	0.3550	11
4149	I	0.280	0.215	0.070	0.1240	0.0630	0.0215	0.0300	6
4150	I	0.330	0.230	0.080	0.1400	0.0565	0.0365	0.0460	7
4151	I	0.350	0.250	0.075	0.1695	0.0835	0.0355	0.0410	6
4152	I	0.370	0.280	0.090	0.2180	0.0995	0.0545	0.0615	7
4153	I	0.430	0.315	0.115	0.3840	0.1885	0.0715	0.1100	8
4154	I	0.435	0.330	0.095	0.3930	0.2190	0.0750	0.0885	6
4155	I	0.440	0.350	0.110	0.3805	0.1575	0.0895	0.1150	6
4156	M	0.475	0.370	0.110	0.4895	0.2185	0.1070	0.1460	8
4157	M	0.475	0.360	0.140	0.5135	0.2410	0.1045	0.1550	8
4158	I	0.480	0.355	0.110	0.4495	0.2010	0.0890	0.1400	8
4159	F	0.560	0.440	0.135	0.8025	0.3500	0.1615	0.2590	9
4160	F	0.585	0.475	0.165	1.0530	0.4580	0.2170	0.3000	11
4161	F	0.585	0.455	0.170	0.9945	0.4255	0.2630	0.2845	11

	Sex	Length	Diameter	Height	Whole_w	Shucked_w	Viscera_w	Shell_w	Rings
4162	M	0.385	0.255	0.100	0.3175	0.1370	0.0680	0.0920	8
4163	I	0.390	0.310	0.085	0.3440	0.1810	0.0695	0.0790	7
4164	I	0.390	0.290	0.100	0.2845	0.1255	0.0635	0.0810	7
4165	I	0.405	0.300	0.085	0.3035	0.1500	0.0505	0.0880	7
4166	I	0.475	0.365	0.115	0.4990	0.2320	0.0885	0.1560	10
4167	M	0.500	0.380	0.125	0.5770	0.2690	0.1265	0.1535	9
4168	F	0.515	0.400	0.125	0.6150	0.2865	0.1230	0.1765	8
4169	M	0.520	0.385	0.165	0.7910	0.3750	0.1800	0.1815	10
4170	M	0.550	0.430	0.130	0.8395	0.3155	0.1955	0.2405	10
4171	M	0.560	0.430	0.155	0.8675	0.4000	0.1720	0.2290	8
4172	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4173	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12

4177 rows × 9 columns

i form two groups of abalones consist of 100 elements each and analyse its "diameter" feature as x

In [4]:

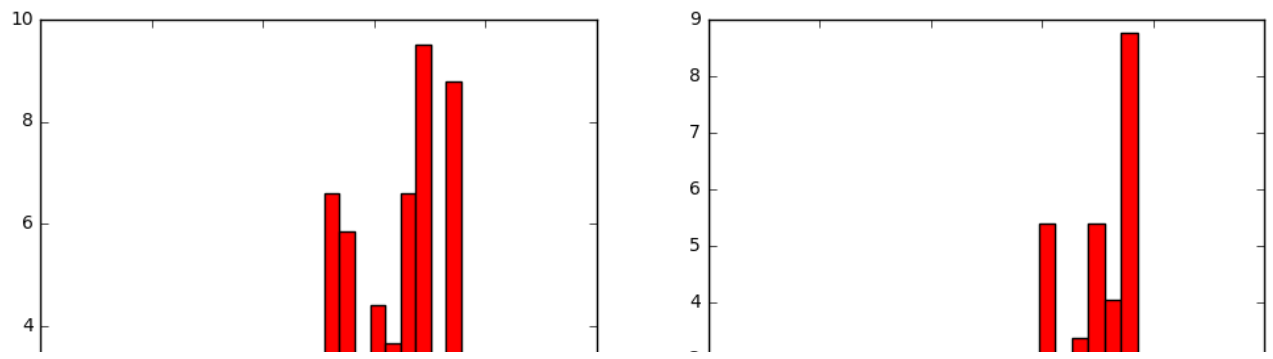
```
ab_diams1 = np.array(list(pass_data["Diameter"][:100]), dtype=float)
ab_diams2 = np.array(list(pass_data["Diameter"][100:200]), dtype=float)
print("mean of the first group", ab_diams1.mean(), "|", "mean of the second
group", ab_diams2.mean())
```

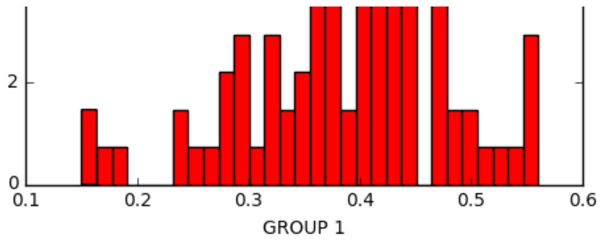
```
mean of the first group 0.3948 | mean of the second group 0.4009
```

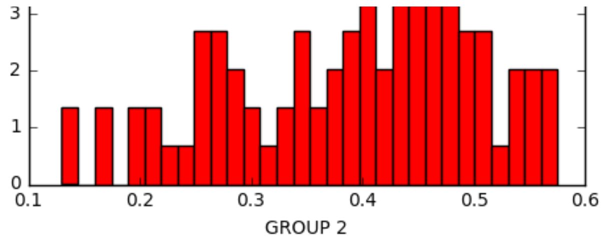
In [5]:

```
plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.xlabel("GROUP 1")
bins1 = plt.hist(ab_diams1, 30, normed=1, facecolor='red', alpha=1)

plt.subplot(122)
plt.xlabel("GROUP 2")
bins1 = plt.hist(ab_diams2, 30, normed=1, facecolor='red', alpha=1)
```







the question (it arose from the comparison of the means): whether snails in the second group have more wide shells in average than in the second or not? It is not clear from the graphs. lets check it using bootstrap (i form 2 groups of vectors(5000 each) of abalones shell diameters with the length of 100 randomly from the first and second groups and then analyse the density func of difference with means in diam5000_1 and diam5000_2). We can do it thanks to central limit theorem (there are groups of 5000 independent mean values which limit to Gaussian function)

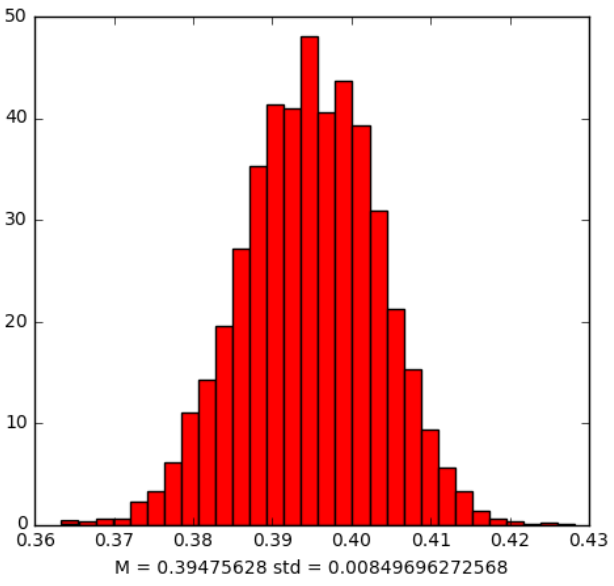
In [6]:

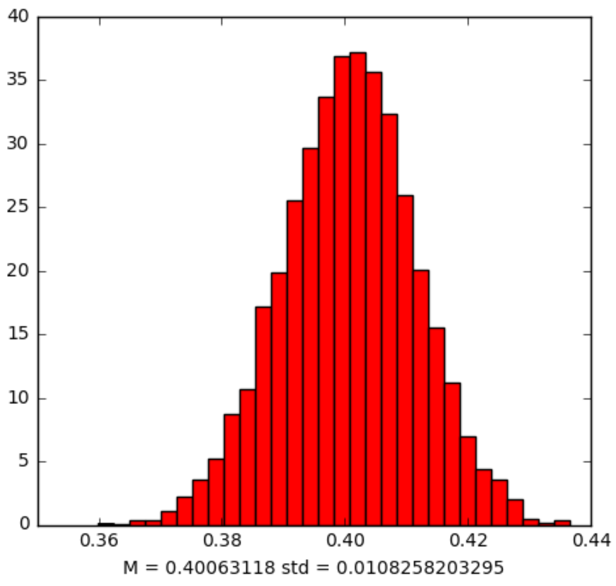
```
diam5000_1 = np.array([[ab_diams1[randint(0, 99)] for i in range(100)] for
j in range(5000)], dtype=float)
diam5000_2 = np.array([[ab_diams2[randint(0, 99)] for i in range(100)] for
j in range(5000)], dtype=float)
means5000_1 = np.array([diam5000_1[i].mean() for i in range(5000)])
means5000_2 = np.array([diam5000_2[i].mean() for i in range(5000)])
mean5000_1 = means5000_1.mean()
mean5000_2 = means5000_2.mean()
std5000_1 = means5000_1.std()
std5000_2 = means5000_2.std()
```

In [7]:

```
plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.xlabel("M = " + str(mean5000_1) + " std = " + str(std5000_1))
bins1 = plt.hist(means5000_1, 30, normed=1, facecolor='red', alpha=1)

plt.subplot(122)
plt.xlabel("M = " + str(mean5000_2) + " std = " + str(std5000_2))
bins2 = plt.hist(means5000_2, 30, normed=1, facecolor='red', alpha=1)
```





In [8]:

```
print("abalones group 1 95% confidence interval ", (mean5000_1 - 1.96 * std
5000_1, mean5000_1 + 1.96 * std5000_1))
print("abalones group 2 95% confidence interval ", (mean5000_2 - 1.96 * std
5000_2, mean5000_2 + 1.96 * std5000_2))
```

```
abalones group 1 95% confidence interval  (0.3781022330576666,
0.41141032694233332)
abalones group 2 95% confidence interval  (0.37941257215409091, 0.421849787
84590915)
```

thus they INTERSECT. This time we can say nothing to answer the question.

In [9]:

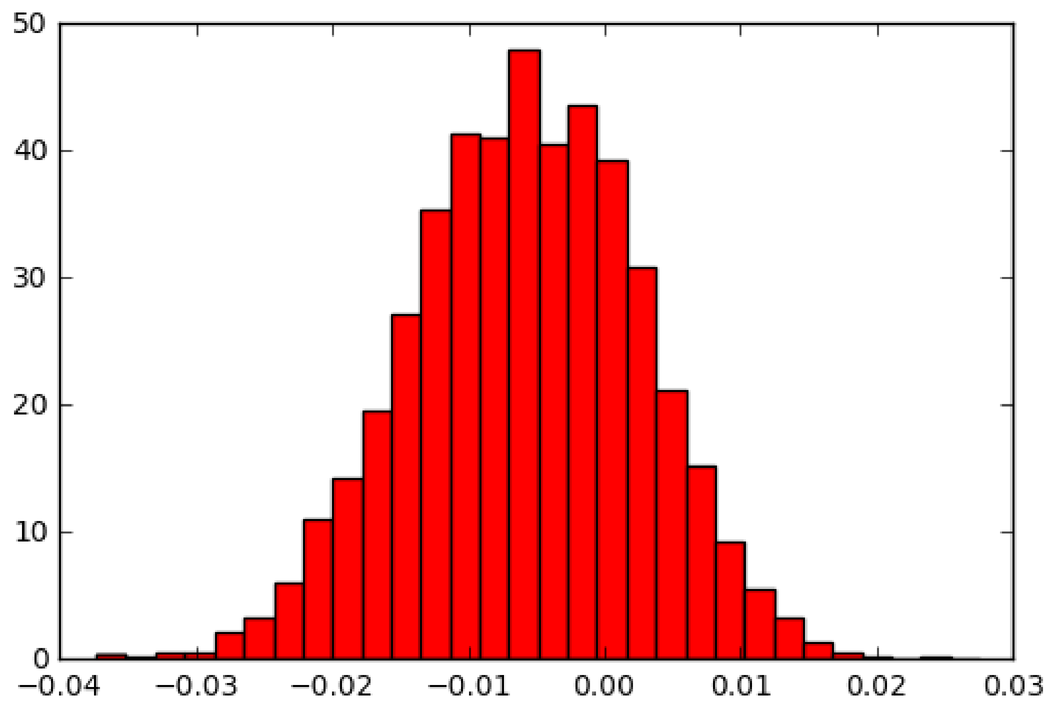
```
mean5000_diff = means5000_1 - mean5000_2
mean_diff = mean5000_1 - mean5000_2
int_ends = 1.96 * math.sqrt((math.pow(std5000_1, 2) / 5000) +
(math.pow(std5000_2, 2) / 5000))
# print("MEAN", mean5000_diff.mean(), "| STD", mean5000_diff.std())
print("95% confidence interval of the difference of the means:", (mean_diff
- int_ends,
                                                                   mean_diff
int_ends))
```

```
95% confidence interval of the difference of the means: (-
0.0062563673245446304, -0.0054934326754555131)
```

so, there is no 0 point in interval and we can say that abalones of 2 groups are not equal (i mean the difference is not 0) in shell diameter feature with the confidence level of 95%. In addition, we can say that group 2 has greater shells in average because the difference confidens interval shifted to the negative side (left with respect to zero point) according to bootstrap method. This bias can also be shown by following histogram.

In [10]:

```
bins = plt.hist(mean5000_diff, 30, normed=1, facecolor='red', alpha=1)
```



```
In [ ]:
```

```

```

```
In [ ]:
```

```

```

HA #2

```
In [11]:
```

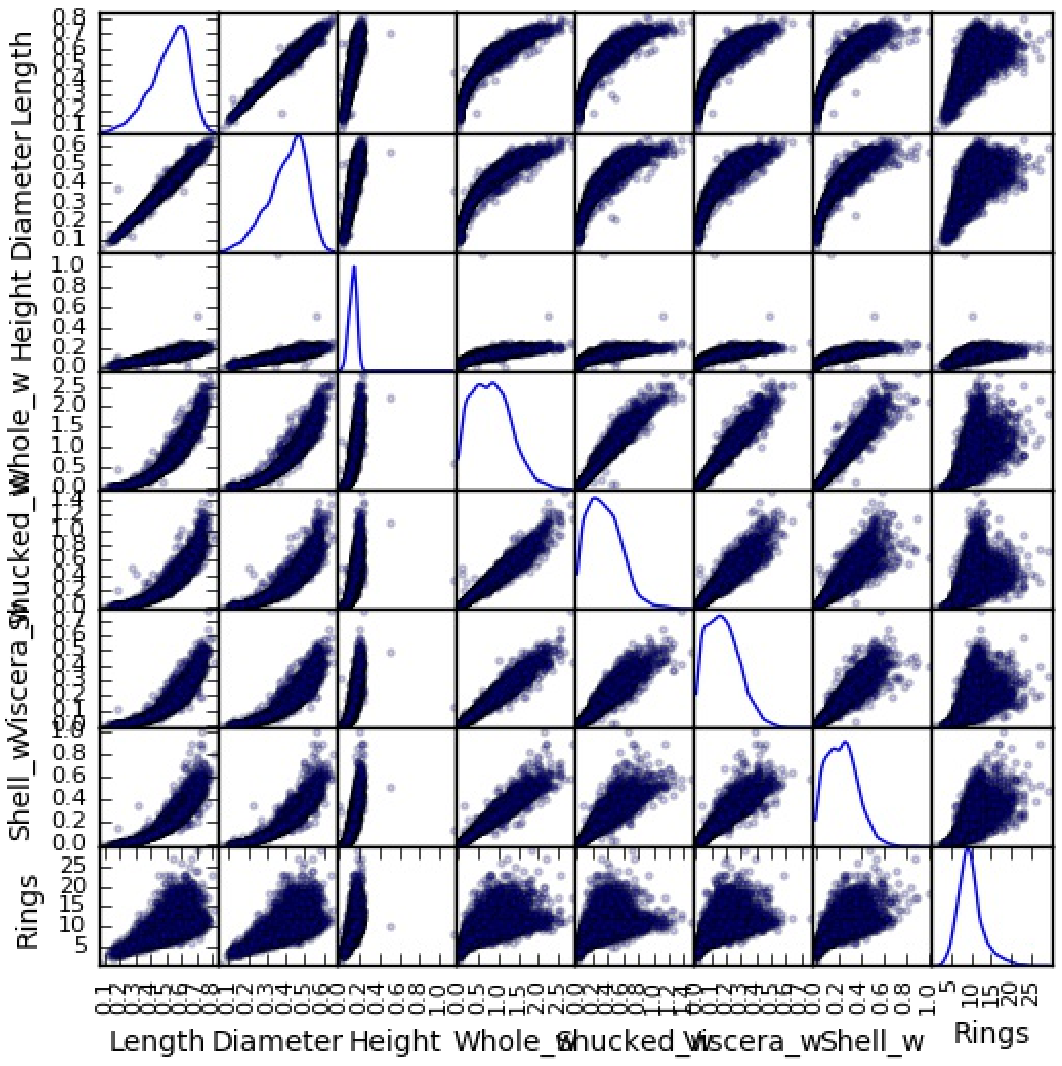
```python
from pandas.tools.plotting import scatter_matrix
```

```
In [12]:
```

```python
scatter_matrix(pass_data, alpha=0.2, figsize=(6, 6), diagonal='kde')
```

```
Out[12]:
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8721115a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87210a3a20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8721071438>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872102e828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720937240>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87208f2390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872083bf60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87207f9f98>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f87207c6128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872077edd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872128f358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87212ca048>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87213aa9e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872130f0b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720b010b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872079b8d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8720ac7b00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720a94dd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720a39a20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720a1eb70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87209ed0b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87209a95f8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87206fccf8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87206bf400>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8720688550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720648240>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720613470>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87205dc978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872059c0f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87205676d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87205290f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87204f0ba8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f87204adcf8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720474e48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720433b38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87203f1828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872033acf8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87202fc6a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87202c5da0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87202854a8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f87202515f8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87202112e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87201da518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f87201a4a20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8720165198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872012c780>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f872002f198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871ff78c50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f871ffb4da0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f75bef0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f717be0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f6e4e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f6b5358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f66fa90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f6400b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f5faa90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f871f5c8588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f5856d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f54f828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f510518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f4c8208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f495320>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f454080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f871f39e668>]],
dtype=object)
```



So, there are a lot of dependencies which look linear-like. In the following graph i've shown a Length-

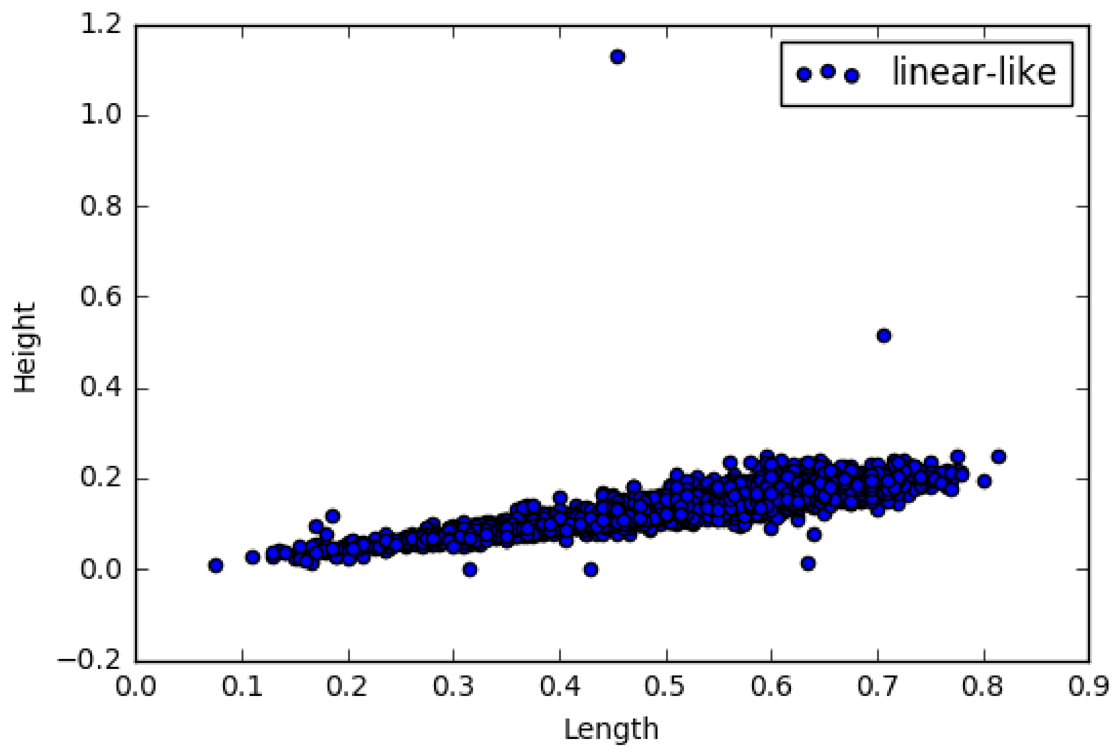
Height scatter plot which looks like a line most of all for me. What about non-linear, for instance, there is Whole_weight-Diameter plot, which looks like porabola. Also there are a lot of complex dependencies, like Shucked_weight-Rings. I have no idea what is it exactly, but i suppose it is not linear-like dependency (which is clear thanks to the fact that Rings count is the target variable of this data-set)

In [13]:

```
pass_data.plot.scatter(x="Length", y="Height", label="linear-like")
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f871eaa0780>
```

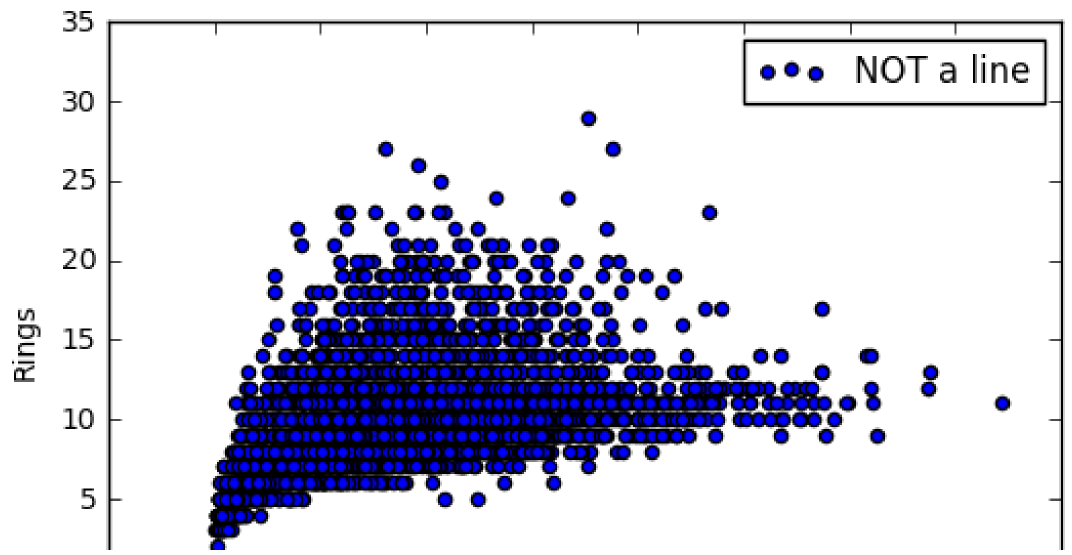


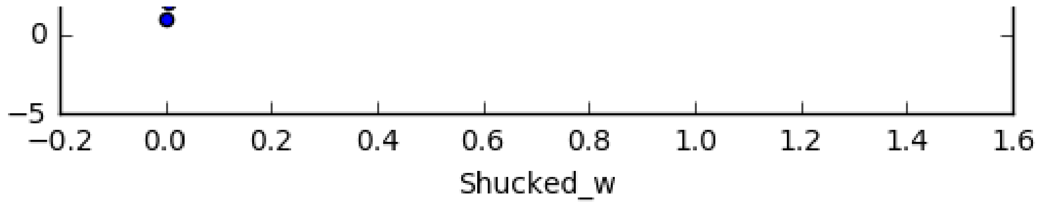
In [14]:

```
pass_data.plot.scatter(x="Shucked_w", y="Rings", label="NOT a line")
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f871d1f8828>
```





Lets build a simple linear regression estimating a and b coefficient of a model. x-asix means lenght, y-axis means height.

$$y_i = ax_i + b + \varepsilon_i$$

Using the following formula.

$$\hat{b} = \bar{y} - \hat{a}\bar{x}$$

$$\hat{a} = \frac{\sum_i^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_i^n (x_i - \bar{x})^2}$$

In [15]:

```
X = np.array(list(pass_data["Length"]), dtype=float)
Y = np.array(list(pass_data["Height"]), dtype=float)
X_mean = np.mean(X)
Y_mean = np.mean(Y)

est_a = np.sum((X - X_mean) * (Y - Y_mean)) / np.sum((X - X_mean) ** 2)
est_b = Y_mean - est_a * X_mean
print("slope, a = ", est_a, "intersept, b = ", est_b)
```

```
slope, a =  0.288227930552 intersept, b =  -0.0115127591617
```

Accordingly to slope we can say, that, when the length of an abalone increases by 1 mm we predict that the height will increase by approximately ~0.288 mm.

In [16]:

```
#make prediction
Y_pred = est_a * X  + est_b
#r coef
Correlation = np.corrcoef(Y, Y_pred)[0][1]
#R^2 calc
Determinacy_coef = Correlation ** 2 # Unfortunately, the specific func for
it is not found

print("r correlation coef =", Correlation, "| R^2 determinacy coef =", Dete
rminacy_coef)
```

```
r correlation coef = 0.827553609319 | R^2 determinacy coef = 0.684844976297
```

The sign of r means the sign of slope a. so if lenght of abalone grows the height grows too. The value of r ~0.827 is located near the 1. It indicates how closely spaced points to a straight line. 0.827 means very good (but not excellent) linear correlation. R^2 ~0.68 is a proportion of the variance of dispersion Y which is tacken into account by regressin of Y over X. 0.68 is ok for reasonable linear model (it should be at least 0.5)

NOT COMPULSORY доверительные интервалы построить создадим 5000 выборок по 4000 элементов и оценим по ним a и b и кореляцию

In [24]:

```
slope_set = []
intersept_set = []
correlation_set = []
abalone_set_size = pass_data.shape[0]

for i in range(5000):
    FOO = np.array(pass_data[["Length",
"Height"]].sample(n=abalone_set_size, replace=True), dtype=float)
    X = FOO[:, 0] #lenght
    Y = FOO[:, 1] #height
    X_mean = np.mean(X)
    Y_mean = np.mean(Y)
    est_a = np.sum((X - X_mean) * (Y - Y_mean)) / np.sum((X - X_mean) ** 2)
    est_b = Y_mean - est_a * X_mean
    Y_pred = est_a * X  + est_b
    Correlation = np.corrcoef(Y, Y_pred)[0][1]
    #fill sets with estimations
    slope_set.append(est_a)
    intersept_set.append(est_b)
    correlation_set.append(Correlation)
```

To make things easier i will be using a non-pivotal method building intervals

In [19]:

```
slope_set.sort()
print("95% confidence interval for slope using bootstrap:", slope_set[125],
slope_set[4875])
#cut off 5% of values (2.5% from each side)
intersept_set.sort()
print("95% confidence interval for intersept using bootstrap:", intersept_s
et[125], intersept_set[4875])
correlation_set.sort()
print("95% confidence interval for correlation using bootstrap:", correlati
on_set[125], correlation_set[4875])
```

```
95% confidence interval for slope using bootstrap: 0.283191585762 0.2932515
37416
95% confidence interval for intersept using bootstrap: -0.0140880689942 -0.
00863279091893
95% confidence interval for correlation using bootstrap: 0.722545435808 0.9
03258205528
```

In [ ]:

```
```

In [ ]:

```
```

HA #3

In [28]:

```
X = np.array(list(pass_data["Length"]), dtype=float)
y = np.array(list(pass_data["Height"]), dtype=float)
```

In [33]:

```
kf = KFold(n_splits=10, random_state=None, shuffle=True) # shuffling data i
s nice
                                                          # in case when we
dont predict future
MSE_over_each_fold = []
for train_index, test_index in kf.split(X):
#     print("TRAIN:", len(train_index), "TEST:", len(test_index))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    X_tr_mean = np.mean(X_train)
    y_tr_mean = np.mean(y_train)
    est_a = np.sum((X_train - X_tr_mean) * (y_train - y_tr_mean)) /
np.sum((X_train - X_tr_mean) ** 2)
    est_b = y_tr_mean - est_a * X_tr_mean
    y_pred = est_a * X_test + est_b
    MSE_over_each_fold.append(((y_test - y_pred) ** 2).mean())
```

In [34]:

```
MSE_over_each_fold
```

Out[34]:

```
[0.00028162194215044373,
 0.00031653097924926862,
 0.00028770995741071519,
 0.0027314475863249458,
 0.00030916170179851309,
 0.00025819003620261891,
 0.00028382631235782227,
 0.00024657466169481098,
 0.0002856960895263219,
 0.00051432500385893156]
```

In [36]:

```
np.array(MSE_over_each_fold).mean()
```

Out[36]:

```
0.00055150842705743925
```

In [ ]:

```

```