My_Project

May 7, 2017

```
In [7]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.preprocessing import MinMaxScaler
        fn = "D:/My Project-Spring 2017/feactures_thresh_2000-tab.txt"
        data = pd.read_table(fn)
        labels = pd.read_csv("D:/My Project-Spring 2017/labels.csv")
        labels= pd.DataFrame(labels)
```

```
In [8]: data = pd.DataFrame(data)
        df = pd.DataFrame(labels)
        new_data = []
        i = 0
        k = 0
        my_data_ids = list( map(str, data["ID"]))

        while  k < len(labels) and i < len(data):
            id = str(labels['id'][i])
            if id in my_data_ids:
                indx = my_data_ids.index(id)
                result = list(data.loc[indx][1:])
                new_data = new_data + [[id] + result + [labels['cancer'][i]] ]
                i += 1
                k = k+1
            else:
                k = k+1
```

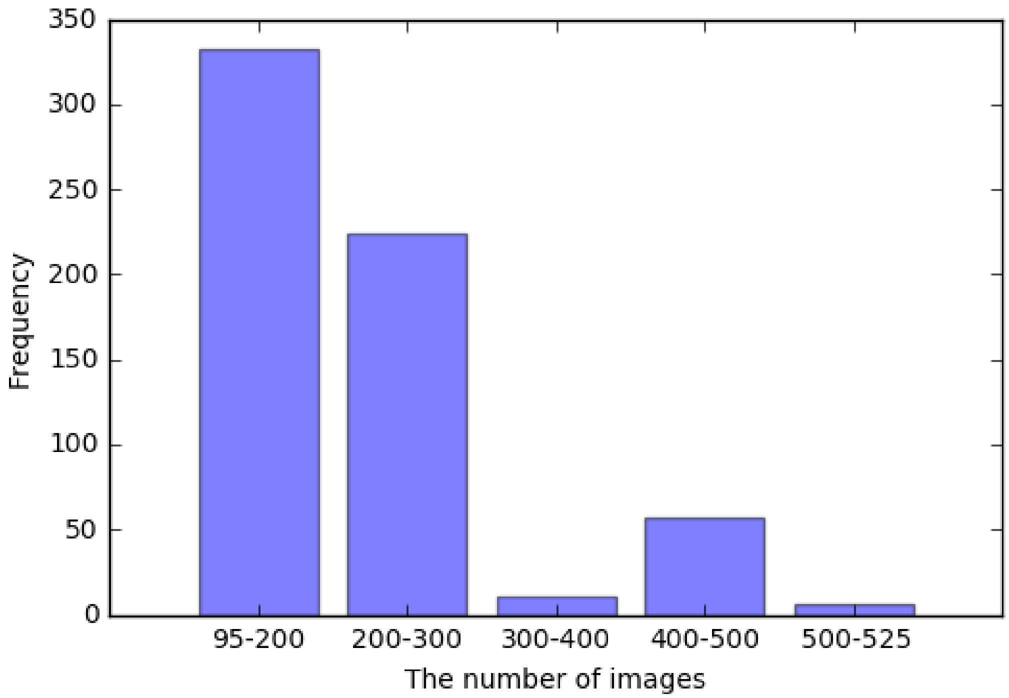
```
In [9]: names = ['id', 'no_images', 'area', 'max', 'HU', 'mu', 'sigma', 'cancer']
        df = pd.DataFrame(new_data, columns = ('id', 'no_images', 'area', 'max', 'H
        len(df)
```

```
Out[9]: 631
```

```
In [118]: x = ('95-200', '200-300', '300-400', '400-500', ' 500-525')
          y= pd.DataFrame(df)
          y = y['no_images']
          print (min(y), max(y))
          y = round(y/100)*100
```

```python
import collections
counter = collections.Counter(y)
print (counter.values())
plt.bar(range(len(x)), counter.values(),  align='center', alpha=0.5)
plt.xticks(range(len(x)), x)

plt.xlabel('The number of images')
plt.ylabel("Frequency")
plt.show()
```

```
95 525
dict_values([333, 224, 11, 57, 6])
```




```python
In [10]: # Just to check if we new data macthes with old ones
         ans = True
         for i in range(0, len(df)):
             if df.loc[i][0] == labels.loc[i][0]:
                 if df.loc[i][-1] == labels.loc[i][-1]:
                     continue
                 else:
                     ans = False

             else:
```

```
            print ("opps for ", i)
            break
    print (ans)
```

```
True
```

In [11]: df

Out[11]:

	id	no_images	area	max
0	0015ceb851d7251b8f399e39779d1e7d	195	26.225641	0.435897
1	0030a160d58723ff36d73f41b170ec21	265	1.158491	0.150943
2	003f41c78e6acfa92430a057ac0b306e	233	0.000000	0.000000
3	006b96310a37b36cccb2ab48d10b49a3	173	0.000000	0.000000
4	008464bb8521d09a42985dd8add3d0d2	146	16.328767	0.541096
5	0092c13f9e00a3717fdc940641f00015	171	3.608187	0.169591
6	00986bebc45e12038ef0ce3e9962b51a	123	0.000000	0.000000
7	00cba091fa4ad62cc3200a657aeb957e	134	0.000000	0.000000
8	00edff4f51a893d80dae2d42a7f45ad1	135	0.000000	0.000000
9	0121c2845f2b7df060945b072b2515d7	191	121.628272	0.434555
10	013395589c01aa01f8df81d80fb0e2b8	217	0.000000	0.000000
11	01de8323fa065a8963533c4a86f2f6c1	231	13.380952	0.718615
12	01e349d34c06410e1da273add27be25c	159	0.000000	0.000000
13	01f1140c8e951e2a921b61c9a7e782c2	241	880.327801	0.705394
14	024efb7a1e67dc820eb61cbdaa090166	175	130.000000	0.617143
15	0257df465d9e4150adef13303433ff1e	186	0.000000	0.000000
16	0268f3a7a17412178cfb039e71799a80	159	0.000000	0.000000
17	026be5d5e652b6a7488669d884ebe297	106	0.000000	0.000000
18	02801e3bbcc6966cb115a962012c35df	205	0.000000	0.000000
19	028996723faa7840bb57f57e28275e4c	183	0.267760	0.136612
20	0334c8242ce7ee1a6c1263096e4cc535	147	2.748299	0.190476
21	03fb0d0fdb187ee1160f09386b28c3f2	149	0.000000	0.000000
22	03ff23e445787886f8b0cb192b3c154d	135	0.000000	0.000000
23	043ed6cb6054cc13804a3dca342fa4d0	160	3324.375000	0.562500
24	0482c444ac838adc5aa00d1064c976c1	223	65.681614	0.367713
25	04a3187ec2ed4198a25033071897bffc	147	0.000000	0.000000
26	04a52f49cdbfb8b99789b9e93f1ad319	145	0.000000	0.000000
27	04a8c47583142181728056310759dea1	151	0.000000	0.000000
28	04cfc5efa4c8c2a8944c8b9fa6cb04d1	161	0.000000	0.000000
29	04e5d435fa01b0958e3274be73312cac	140	184.342857	0.357143
..	...	...	...	...
601	6e5f12931ef179cc21382a59f5acab86	127	0.000000	0.000000
602	6e6d5603fb8fcf523f86ac0856e50236	172	0.552326	0.552326
603	6ee742b62985570a1f3a142eb7e49188	226	576.495575	0.482301
604	6f38eb7988753c6a978d0da80dbc014b	152	0.000000	0.000000
605	6f43af3f636f37b9695b58378f9265cc	248	0.000000	0.000000
606	6faabf4152bf0ebfd91f686bc37a1f16	134	154.559701	0.268657
607	6fd3af9174242c1b393fe4ba515e7a26	122	169.540984	0.319672

608	6fd582d25eeb2250c2b0996c4216deb9	125	0.000000	0.000000
609	700bdc2723c2ac75a8a8376d9ca170ad	159	0.000000	0.000000
610	70287a7720e0d90249ac7b3978b7ca40	134	0.000000	0.000000
611	7050f8141e92fa42fd9c471a8b2f50ce	172	11.110465	0.622093
612	7051fc0fcf2344a2967d9a1a5478208e	140	16.900000	0.464286
613	713d8136c360ad0f37d6e53b61a7891b	161	0.000000	0.000000
614	71665cc6a7ee85268ca1da69c94bbaeb	234	218.393162	0.431624
615	7180c83eb184d5c9dfcbda228ab91213	141	1.418440	0.085106
616	718f43ecf121c79899caba1e528bd43e	119	0.000000	0.000000
617	71e09cd11d743964f1abf442c34f2c9d	134	0.000000	0.000000
618	721949894f5309ed4975a67419230a3c	245	1.306122	0.436735
619	722429bc9cb25d6f4b7a820c14bf2ab1	116	0.000000	0.000000
620	7239b3a904f39b25c4e303c10a24621a	103	0.000000	0.000000
621	72609c2be68be9d7c9cde3d0127c05ac	162	6.771605	0.185185
622	72a1e35c34052e163f61585ba0c9daf4	156	0.000000	0.000000
623	72b080b50118e9ddb795890eb1f13684	155	0.000000	0.000000
624	72ed4046708e5607eb0a5703905438ee	155	0.000000	0.000000
625	72fd04cf3099b148d9ad361efb988866	219	0.000000	0.000000
626	73280f6a624b3bf7a766c70b31dfc56b	154	0.000000	0.000000
627	733205c5d0bbf19f5c761e0c023bf9a0	131	8.022901	0.114504
628	7367ede966b44c6dce30b83345785671	267	111.067416	0.550562
629	7395f64fba89c2463a1b13c400adf876	259	0.000000	0.000000
630	73b28e2eadad587c9a8ac6c7186dd51b	159	6.836478	0.496855

	HU	mu	sigma	cancer
0	44.682051	0.179501	0.794980	1
1	22.415094	0.111439	0.555196	0
2	20.686695	0.139469	0.746667	0
3	9.404624	0.085476	0.521393	1
4	42.068493	0.063953	0.437583	1
5	33.269006	0.084927	0.524975	0
6	0.000000	0.032570	0.239367	0
7	0.000000	0.131626	0.739179	0
8	0.000000	0.043789	0.387303	1
9	80.774869	0.091854	0.498592	0
10	22.410138	0.295574	1.056444	0
11	44.935065	0.333027	1.226209	0
12	12.446541	0.105530	0.656555	0
13	146.020747	0.322544	1.665774	0
14	511.994286	0.050781	0.407039	0
15	0.000000	0.000084	0.009568	1
16	0.000000	0.194824	0.908341	0
17	15.264151	0.105469	0.675777	0
18	0.000000	0.000034	0.007042	1
19	40.896175	0.068691	0.474446	1
20	49.850340	0.162430	0.871358	0
21	0.000000	0.015884	0.192657	0
22	11.940741	0.106369	0.552477	0

```
23   214.906250  0.000000  0.000000       0
24   161.937220  0.133541  0.735602       0
25    11.258503  0.065155  0.431642       0
26     0.000000  0.124817  0.580352       0
27     0.000000  0.066650  0.435453       1
28    20.155280  0.106937  0.660640       0
29   125.528571  0.055313  0.388186       0
..          ...       ...       ...     ...
601    0.000000  0.028301  0.248462       0
602   30.383721  0.156784  0.948409       0
603  130.353982  0.219109  1.044102       1
604   10.526316  0.086338  0.463163       0
605   20.391129  0.179844  0.844417       1
606  117.462687  0.149693  0.738610       0
607  100.688525  0.042397  0.301850       1
608    0.000000  0.075123  0.438257       0
609   10.182390  0.143597  0.645223       0
610   12.179104  0.209431  0.928365       0
611   17.331395  0.143047  0.630558       0
612   58.485714  0.000187  0.016686       0
613   19.875776  0.144199  0.736229       1
614  131.217949  0.003273  0.097328       1
615   26.212766  0.078621  0.444034       0
616    0.000000  0.033562  0.245448       0
617    0.000000  0.041878  0.316300       0
618   29.632653  0.198608  0.872928       1
619    0.000000  0.059921  0.395656       0
620    0.000000  0.022118  0.205104       0
621   38.777778  0.108948  0.485989       0
622    0.000000  0.000542  0.028163       0
623   22.922581  0.165226  0.789167       0
624    0.000000  0.054123  0.378494       0
625   29.794521  0.107410  0.555371       1
626    0.000000  0.069721  0.546760       0
627  117.381679  0.068066  0.457826       1
628  138.097378  0.276417  1.400465       0
629    6.397683  0.232712  1.227217       1
630   50.433962  0.202473  0.852514       0

[631 rows x 8 columns]
```

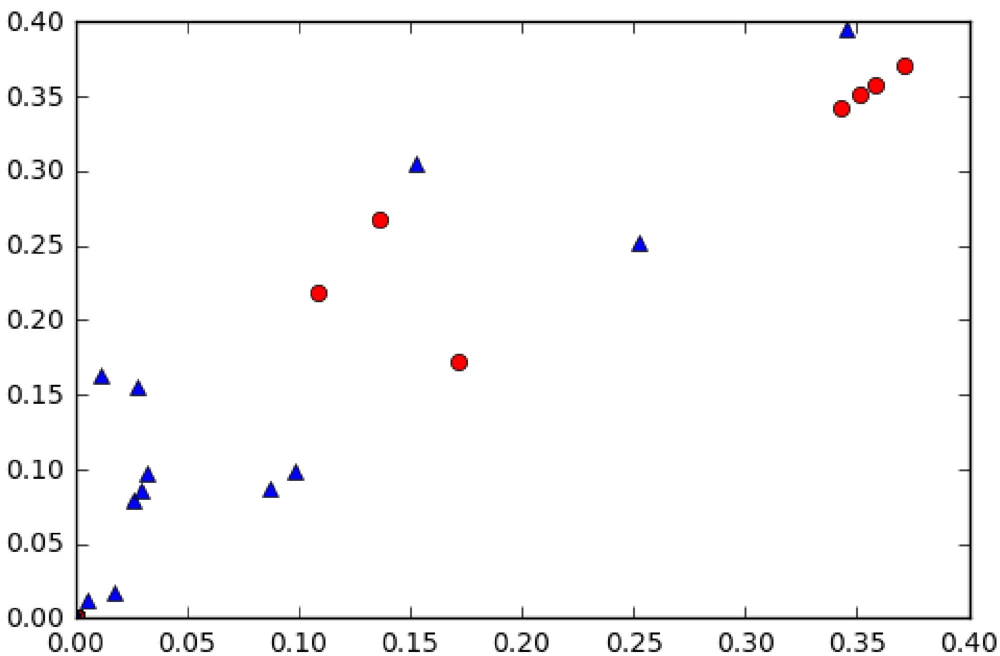
In [19]:
```python
print ('Max and Min of area: ', max(df['area']), 'and ', min(df['area']))
print ('Max and Min of max: ', max(df['max']), 'and ', min(df['max']))
print ('Max and Min of HU: ', max(df['HU']), 'and ', min(df['HU']))
print (sum(df['no_images']))
```

```
Max and Min of area:  3324.375 and  0.0
Max and Min of max:  0.811428571 and  0.0
```

```
Max and Min of HU:  1374.512987 and  0.0
112190
```

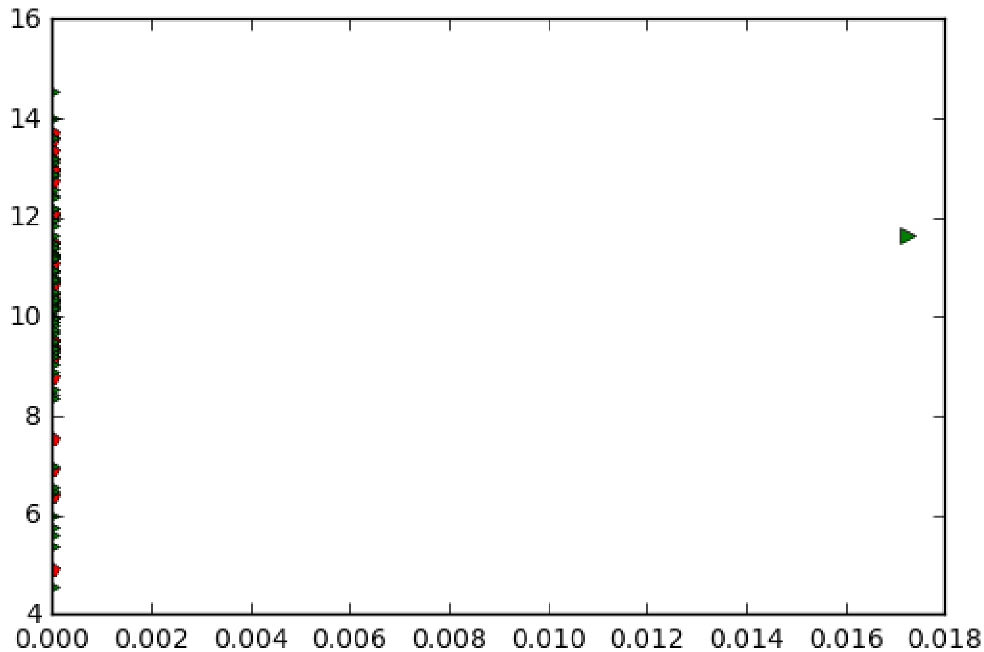
```
In [12]: df1 = pd.DataFrame(df)
         df1= df1[(df1.area <= .4) & (df1['max']<=.40)]
         print (len(df1))

         n1 = df1[df1['cancer'] == 1]
         n0 = df1[df1['cancer'] == 0]

         plt.plot(n1['max'], n1['area'], 'ro')
         plt.plot(n0['max'], n0['area'], 'b^')

         plt.show()
```

```
427
```



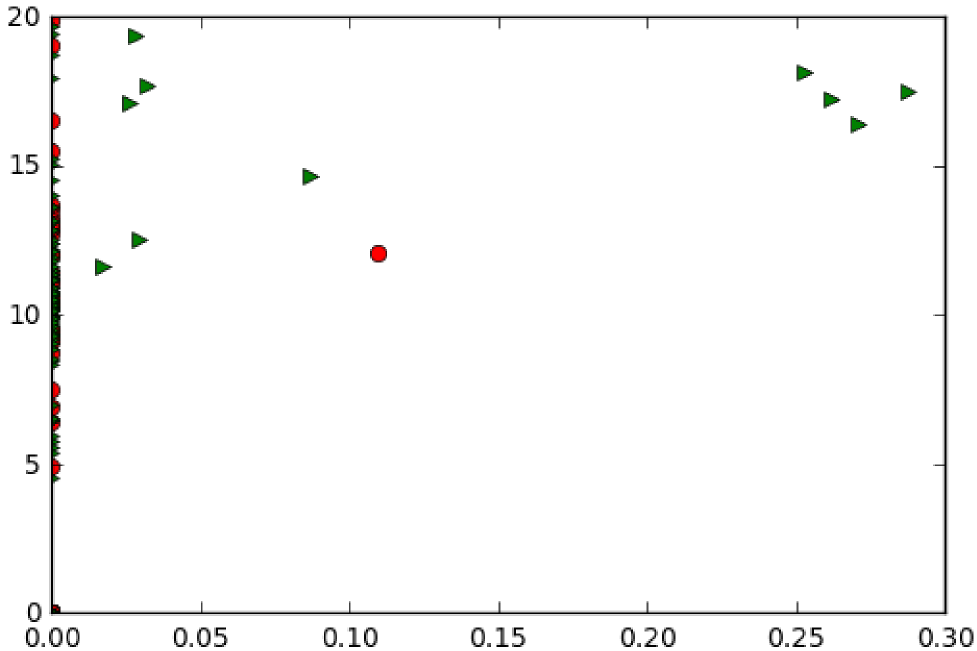

```
In [450]: df2 = pd.DataFrame(df)
          df2 = df2[(df2.area>=0) & (df2.area<=.03)]
          df2 = df2[(df2.HU>=.02) & (df2.HU<=15)]
          g0 = df2[df2['cancer'] == 0]
          g1 = df2[df2['cancer'] == 1]
          plt.plot(g1['area'], g1['HU'], 'rp')
          plt.plot(g0['area'], g0['HU'], 'g>')
```

```
plt.show()

print ("The length is %d" %len(df2))
```



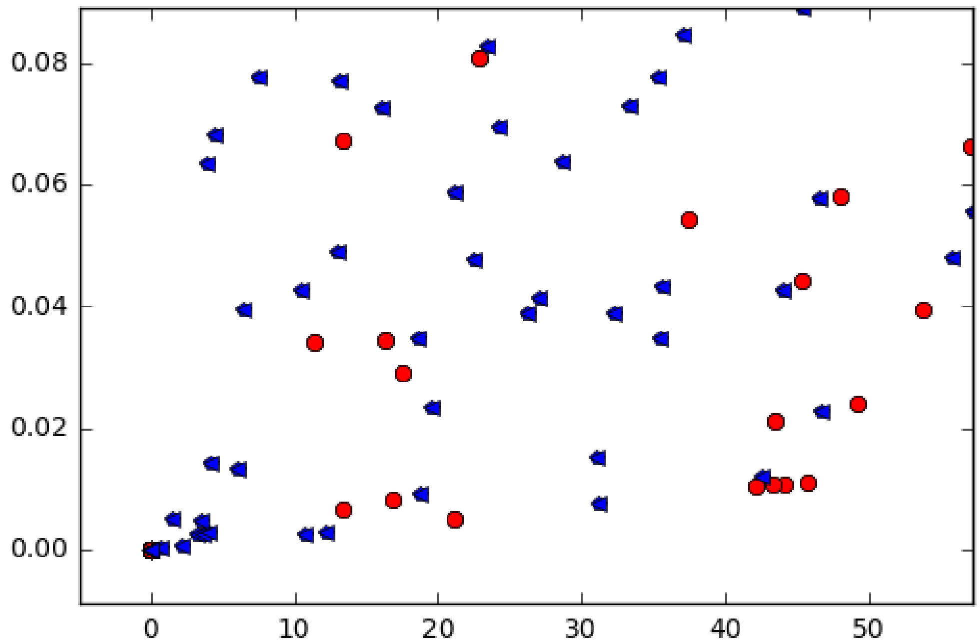

```
The length is 113
```

```
In [452]: df22 = pd.DataFrame(df)
          df22 = df22[(df22['max']>=0) & (df22['max']<=.4)]
          df22 = df22[(df22.HU>=.0) & (df22.HU<=20)]
          g0 = df22[df22['cancer'] == 0]
          g1 = df22[df22['cancer'] == 1]
          plt.plot(g1['max'], g1['HU'], 'ro')
          plt.plot(g0['max'], g0['HU'], 'g>')
          plt.show()

          print ("The length is ", len(df2))
```

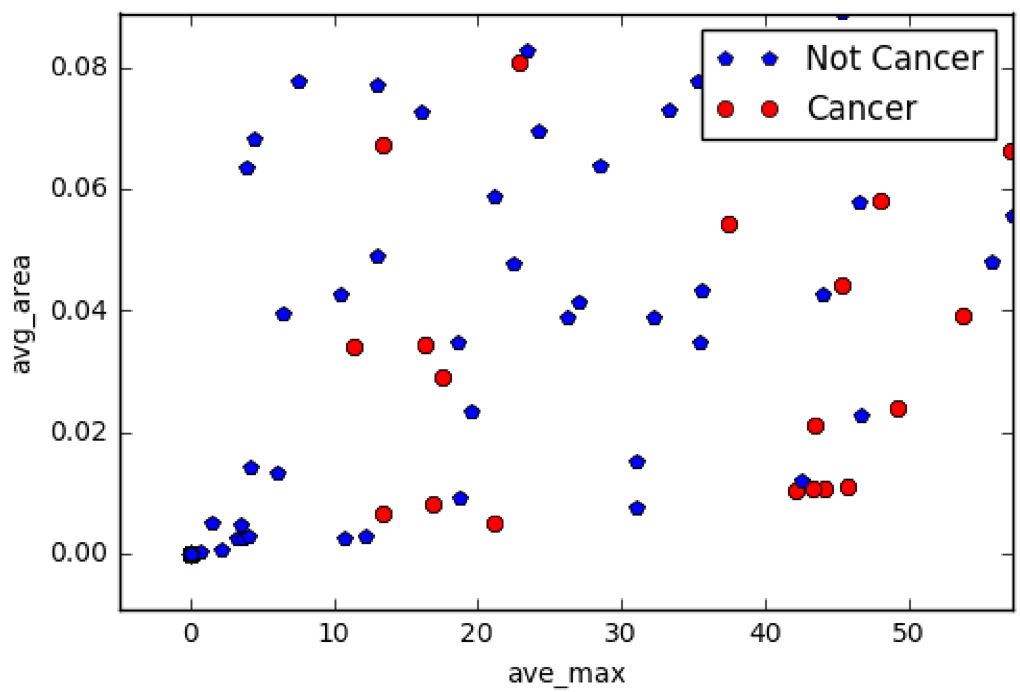



```
The length is  113
```

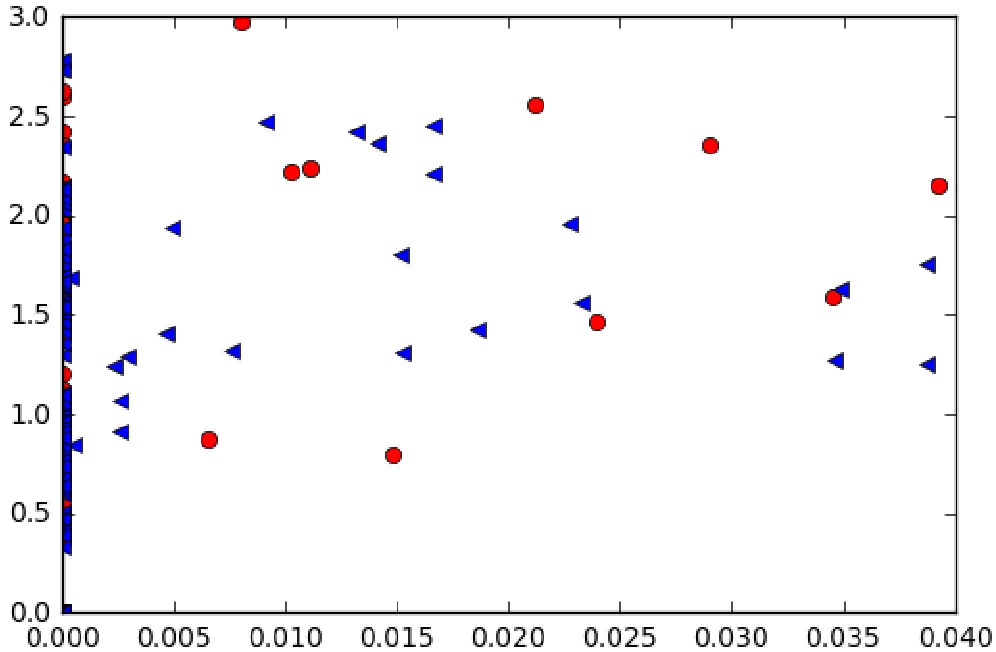
```
In [406]: max(df['area'])

Out[406]: 3324.375

In [334]: scaler = MinMaxScaler()

In [335]: ndf= pd.DataFrame(df)

In [336]: ndf[['area', 'max', 'HU']] = scaler.fit_transform( ndf[['area', 'max', 'H

In [337]: ndf[['area', 'max', 'HU']] = ndf[['area', 'max', 'HU']] *100

In [338]: max(ndf['max'])

Out[338]: 100.0

In [341]: ndf1 = ndf[(ndf['area'] <= 3) ]
          n1 = ndf1[ndf1['cancer'] == 1]
          n0 = ndf1[ndf1['cancer'] == 0]
          plt.plot(n1['max'], n1['area'], 'ro')
          plt.plot(n0['max'], n0['area'], 'b<')

          plt.show()
```



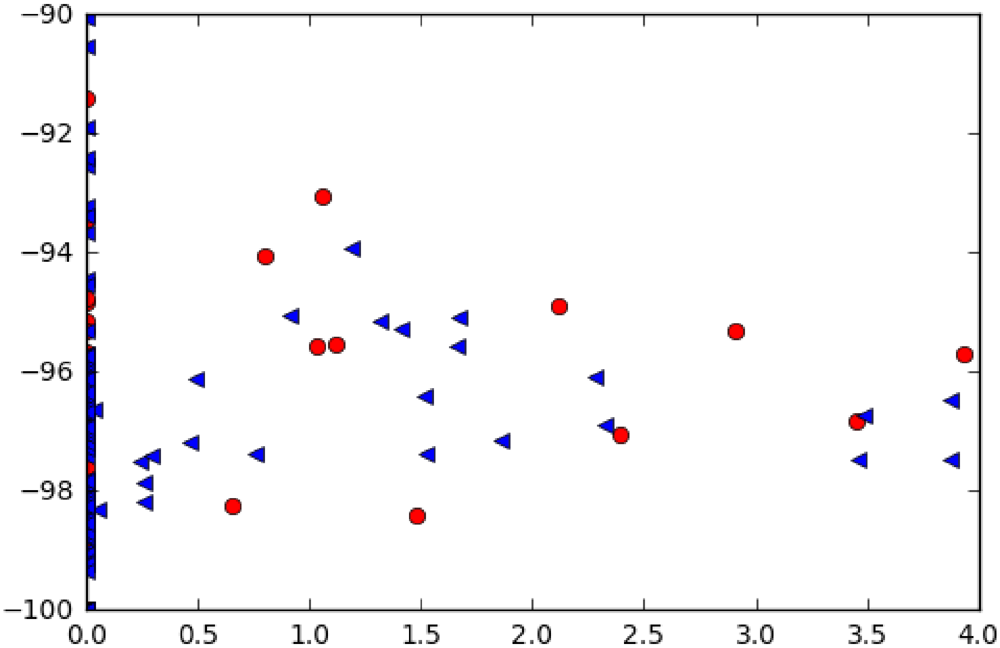

```
In [351]: ndfn = pd.DataFrame(ndf)
          ndfn=ndfn[(ndfn['max']<=60) & (ndfn.area<=.1)]
          n1 = ndfn[ndfn['cancer'] == 1]
          n0 = ndfn[ndfn['cancer'] == 0]

          N1, = plt.plot(n1['max'], n1['area'], 'ro', label='Cancer')
          N0, = plt.plot(n0['max'], n0['area'], 'bp', label='Not Cancer')
          plt.legend(handles=[N0, N1])
          plt.xlim(-5, max(ndfn['max']))
          plt.ylim(-.009, max(ndfn['area']))
          plt.xlabel("ave_max")
          plt.ylabel("avg_area")

          plt.show()
```



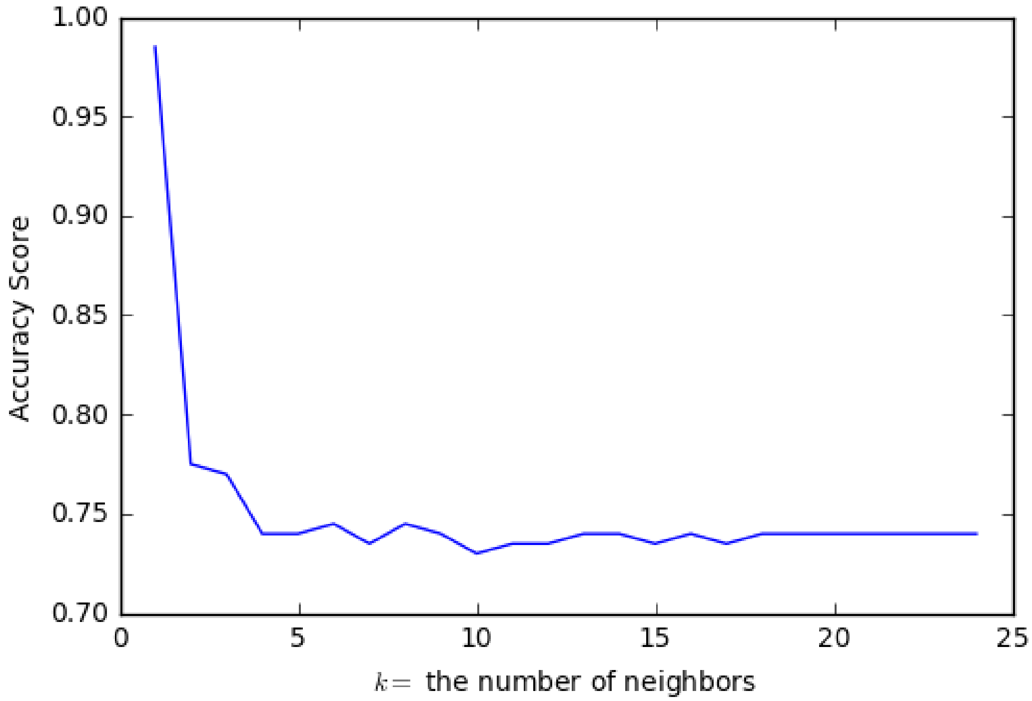

```
In [107]: ndfn = pd.DataFrame(ndf)
          ndfn=ndfn[(ndfn['area']<=.04) & (ndfn.HU<=3)]
          n1 = ndfn[ndfn['cancer'] == 1]
          n0 = ndfn[ndfn['cancer'] == 0]

          plt.plot(n1['area'], n1['HU'], 'ro')
          plt.plot(n0['area'], n0['HU'], 'b<')

          plt.show()
```




```
In [482]: mdf = pd.DataFrame(df)
          mdf[[ 'HU']] = 200*scaler.fit_transform( mdf[['HU']] )-100
          mdf[['area']] = 10000*scaler.fit_transform( mdf[['area']] )

In [ ]:

In [483]: nmdf = pd.DataFrame(mdf)
          nmdf=nmdf[(nmdf['area']<=4) & (nmdf.HU<=-90)]
          n1 = nmdf[nmdf['cancer'] == 1]
          n0 = nmdf[nmdf['cancer'] == 0]

          plt.plot(n1['area'], n1['HU'], 'ro')
          plt.plot(n0['area'], n0['HU'], 'b<')

          plt.show()
```




```
In [359]: from sklearn.neighbors import KNeighborsClassifier
          X = ndf[['area', 'max', 'HU', 'mu', 'sigma']]
          Y = ndf['cancer']
          X_trainig = X.loc[1:400]
          Y_training = Y.loc[1:400]
          X_test = X.loc[401:600]
          Y_test = Y.loc[401:600]
          accuracy_list = []
          for k in range(1,25):
              neigh = KNeighborsClassifier(n_neighbors=k)
              neigh.fit(X, Y)
              KNeighborsClassifier(...)
              predicted = pd.DataFrame(neigh.predict(X_test))
              accuracy_list = accuracy_list + [neigh.score(X_test, Y_test)]

In [360]: plt.plot(range(1,25), accuracy_list)
          plt.xlabel("$k = $ the number of neighbors")
          plt.ylabel("Accuracy Score")
          plt.show()
```




```
In [313]: predicted = pd.DataFrame(neigh.predict(X_test))
          miss_num = 0
          print(neigh.score(X_test, Y_test))

0.74


In [296]: from sklearn.metrics import accuracy_score
          predicted = pd.DataFrame(neigh.predict(X_test))
          accuracy = accuracy_score(Y_test, predicted)
          error_rate = 1 - accuracy

In [297]: error_rate

Out[297]: 0.015000000000000013

In [305]:

Out[305]: [1, 3, 5, 7, 9]

In [ ]:
```