Poverty Data for Central and South America Countries

In [6]:

```
import os
os.getcwd()
os.chdir("C:/Users/Sola Fide/Documents/Python Scripts/databank.worldbank.org")

import pandas
poverty = pandas.read_csv("poverty.csv")
poverty.head()
```

Out[6]:

	...	2005	2006	2007	2008	2009	2010
9475.0	...	39145491.0	39558750.0	39969903.0	40381860.0	40798641.0	41222875.0
88418.0	...	109747906.0	111382857.0	113139374.0	114972821.0	116815612.0	118617542.0
3346.0	...	13183505.0	13490041.0	13797629.0	14106687.0	14418033.0	14732261.0
555.0	...	9263409.0	9409479.0	9556958.0	9705130.0	9852953.0	9999617.0
511.0	...	3319301.0	3378600.0	3438398.0	3498679.0	3559401.0	3620506.0

In [22]:

```
# Data Indexing
pop = poverty[poverty.SeriesCode == 'SP.POP.TOTL']
pov = poverty[poverty.SeriesCode == 'SI.POV.DDAY']
incomeHigh10 = poverty[poverty.SeriesCode == 'SI.DST.10TH.10']
incomeLow10 = poverty[poverty.SeriesCode == 'SI.DST.FRST.10']
```

In [59]:
```
# Population Number Indexing
popN = pop.iloc[:,4:]
popN.describe()
```

Out[59]:

	1998	1999	2000	2001	2002	2003
count	1.900000e+01	1.900000e+01	1.900000e+01	1.900000e+01	1.900000e+01	1.900000
mean	2.309741e+07	2.346308e+07	2.382263e+07	2.417519e+07	2.452150e+07	2.486333
std	4.225042e+07	4.290780e+07	4.355577e+07	4.419308e+07	4.482037e+07	4.543654
min	2.302890e+05	2.390240e+05	2.473120e+05	2.549890e+05	2.622020e+05	2.691320
25%	4.080880e+06	4.129490e+06	4.174017e+06	4.213928e+06	4.249753e+06	4.283256
50%	8.026257e+06	8.182710e+06	8.339512e+06	8.496378e+06	8.653343e+06	8.810420
75%	1.917717e+07	1.950192e+07	1.982593e+07	2.014932e+07	2.047185e+07	2.079341
max	1.705165e+08	1.731531e+08	1.757864e+08	1.784194e+08	1.810456e+08	1.836273

In [27]:
```
# Poverty headcount ratio at $1.90 a day (2011 PPP) (% of population)
povN = pov.iloc[:,4:]
povN.describe()
```

Out[27]:

	1998	1999	2000	2001	2002	2003	2004	200
count	15.000000	12.000000	10.000000	12.000000	11.000000	11.000000	12.000000	12.0
mean	13.256667	16.389167	13.177000	17.277500	14.531818	12.689091	11.383333	12.6
std	6.352524	6.718820	9.647012	13.539626	7.425273	7.423150	6.504579	7.09
min	0.900000	4.780000	0.570000	0.750000	1.080000	1.310000	1.490000	1.54
25%	12.075000	12.572500	6.800000	9.547500	12.120000	9.380000	6.965000	8.90
50%	14.290000	15.095000	11.405000	15.020000	13.310000	12.240000	10.900000	11.9
75%	15.665000	22.210000	16.195000	18.840000	15.675000	16.350000	14.045000	15.9
max	25.930000	26.450000	29.670000	55.590000	29.060000	27.470000	26.320000	27.7

In [28]:
```
povN.head()
```

Out[28]:

	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	201
19	4.59	4.78	5.70	9.42	13.99	9.79	7.09	5.37	4.12	3.49	2.97	3.05	2.05	1.5
20	14.29	NaN	11.00	NaN	8.81	NaN	4.82	NaN	3.29	NaN	3.80	NaN	3.80	Na
21	13.38	NaN	10.10	NaN	NaN	NaN	NaN	NaN	11.51	NaN	NaN	NaN	NaN	11.
22	NaN	NaN	NaN	55.59	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
23	15.16	13.39	14.68	16.42	11.93	11.57	10.76	10.30	10.99	8.11	5.49	3.37	4.57	3.9

In [51]:
```
# Income share held by highest 10%
incHighN = incomeHigh10.iloc[:,4:]
incHighN.describe()
```

Out[51]:

	1998	1999	2000	2001	2002	2003	2004	200
count	14.000000	11.000000	10.000000	12.000000	11.000000	11.000000	12.000000	12.0
mean	40.149286	42.101818	41.817000	41.921667	41.930000	41.781818	39.900833	39.6
std	4.649116	3.703047	4.792517	4.516698	4.185984	3.970372	3.962152	4.61
min	32.510000	36.230000	33.050000	34.440000	34.860000	35.100000	35.460000	31.4
25%	36.827500	40.250000	38.602500	39.215000	38.995000	38.670000	35.672500	36.3
50%	39.290000	42.430000	42.525000	42.245000	42.510000	43.190000	40.485000	40.0
75%	44.262500	43.730000	45.332500	45.835000	45.390000	44.770000	42.945000	43.2
max	47.590000	47.850000	49.000000	47.820000	47.260000	46.000000	45.240000	46.2

In [52]:
```
# Income share held by lowest 10%
incLowN = incomeLow10.iloc[:,4:]
incLowN.describe()
```

Out[52]:

	1998	1999	2000	2001	2002	2003	2004	2005
count	14.000000	11.000000	10.000000	12.000000	11.000000	11.000000	12.000000	12.0
mean	1.042857	0.709091	0.984000	0.819167	0.874545	0.920909	1.030000	1.06
std	0.537393	0.269832	0.483372	0.338109	0.449185	0.444892	0.404924	0.63
min	0.220000	0.260000	0.130000	0.350000	0.270000	0.190000	0.180000	0.05
25%	0.577500	0.530000	0.942500	0.647500	0.575000	0.690000	0.865000	0.75
50%	1.050000	0.780000	1.050000	0.795000	0.870000	0.790000	0.965000	0.99
75%	1.402500	0.905000	1.240000	0.932500	0.990000	1.175000	1.285000	1.27
max	2.070000	1.070000	1.710000	1.640000	1.650000	1.830000	1.660000	2.53

In [57]:
```
# data sets (column mean)
povAvg = np.mean(povN)
incLowAvg = np.mean(incLowN)
incHighAvg = np.mean(incHighN)
```

In [36]:
```
# Loading stats fuction from scipy package
from scipy import stats

# Loading norm function from scipy package
from scipy.stats import norm
```

In [201]:
```
# cdf: Cumulative Distribution Function
norm.cdf(0)
```

Out[201]: 0.5

In [202]:
```
# cdf: Cumulative Distribution Function
norm.cdf([-1.96, 0, 1.96])
```

Out[202]:
```
array([ 0.0249979,  0.5      ,  0.9750021])
```

In [203]:
```
# ppf: Percent Point Function (Inverse of CDF)
norm.ppf([.025, .5, .975])
```

Out[203]:
```
array([-1.95996398,  0.        ,  1.95996398])
```

In [204]:
```
# cdf: Cumulative Distribution Function
stats.t.cdf(0, 30)
```

Out[204]: 0.5

In [205]:
```
# isf: Inverse Survival Function (Inverse of SF)
stats.t.isf([.1, .05, .01], 15)
```

Out[205]:
```
array([ 1.34060561,  1.75305036,  2.60248029])
```

In [206]:
```
# isf: Inverse Survival Function (Inverse of SF)
stats.t.isf([.1, .05, .01], [[15],[16]])
```

Out[206]:
```
array([[ 1.34060561,  1.75305036,  2.60248029],
       [ 1.33675717,  1.74588368,  2.58348719]])
```

In [39]:
```
norm.mean(), norm.std(), norm.var()
```

Out[39]:
```
(0.0, 1.0, 1.0)
```

In [40]:
```
# To find the median of a distribution we can use the percent point function ppf,
norm.ppf(0.5)
```

Out[40]:
```
0.0
```

In [119]:
```
# Filling the NA value with Mean value
povNF = povN.where(pandas.notnull(povN), povN.mean(), axis='columns')
#povN.head()
```

In [68]:
```
# Normalize population data into mean of 0 and std of 1.

population = popN.iloc[:,-1]  # 2014, 19 country total population

from sklearn import preprocessing
import numpy as np

pop_scaled = preprocessing.scale(population)
pop_scaled

pov_scaled = povN.iloc[:,-1]
```

Out[68]:
```
array([ 0.28794692,  1.93261499, -0.25016916, -0.3588612 , -0.4926632 ,
       -0.56283351, -0.41093694, -0.44794039, -0.44982204, -0.25244467,
        0.04273777,  0.04830644,  3.54282632, -0.55460391, -0.55911165,
       -0.35903764, -0.21533736, -0.43907283, -0.50159797])
```

In [67]:
```
np.mean(pop_scaled)
np.std(pop_scaled)
```

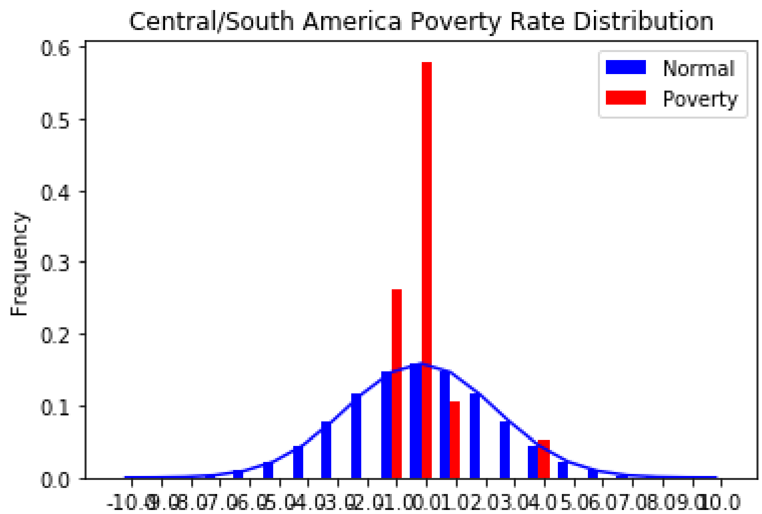
Out[67]:
```
0.99999999999999978
```

In [72]:
```
povAvgByCnty = np.mean(povN, axis=1)
pov_scaled = preprocessing.scale(povAvgByCnty)
pov_scaled
```

Out[72]:
```
array([-0.74828015, -0.63411621, -0.19038092,  3.67894106, -0.38933744,
        0.09656435,  0.722315  , -0.36990972,  0.04045376, -0.14226077,
        0.04950959, -0.24519632, -0.40103236,  0.06105132,  0.89383183,
        0.24356292, -0.98835866, -0.55871021, -1.11864708])
```

In [73]:

```python
# Normal distribution

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats



npoints = 20 # number of integer support points of the distribution minus 1
npointsh = npoints / 2
npointsf = float(npoints)
nbound = 4 #bounds for the truncated normal
normbound = (1 + 1 / npointsf) * nbound #actual bounds of truncated normal
grid = np.arange(-npointsh, npointsh+2, 1) #integer grid
gridlimitsnorm = (grid-0.5) / npointsh * nbound #bin limits for the truncnorm
gridlimits = grid - 0.5
grid = grid[:-1]
probs = np.diff(stats.truncnorm.cdf(gridlimitsnorm, -normbound, normbound))
gridint = grid
normdiscrete = stats.rv_discrete(
                        values=(gridint, np.round(probs, decimals=7)),
                        name='normdiscrete')


n_sample = len(pov_scaled)
#np.random.seed(87655678) #fix the seed for replicability
#rvs = normdiscrete.rvs(size=n_sample)
rvs = pov_scaled
rvsnd=rvs
f,l = np.histogram(rvs, bins=gridlimits)
sfreq = np.vstack([gridint, f, probs*n_sample]).T
fs = sfreq[:,1] / float(n_sample)
ft = sfreq[:,2] / float(n_sample)
nd_std = np.sqrt(normdiscrete.stats(moments='v'))

ind = gridint  # the x locations for the groups
width = 0.35       # the width of the bars

plt.subplot(111)
rects1 = plt.bar(ind, ft, width, color='b')
rects2 = plt.bar(ind+width, fs, width, color='r')
normline = plt.plot(ind+width/2.0, stats.norm.pdf(ind, scale=nd_std),
                    color='b')

plt.ylabel('Frequency')
plt.title('Central/South America Poverty Rate Distribution')
plt.xticks(ind+width, ind)
plt.legend((rects1[0], rects2[0]), ('Normal', 'Poverty'))

plt.show()
```



Analysing One Sample

In [78]:
```
# Random Variable
#  requires shape parameter of the t distribution,
#  which in statistics corresponds to the degrees of freedom, to 10
x= stats.t.rvs(10, size=10)
x
```

Out[78]:
```
array([ 0.12282784,  0.09821334,  1.31632051, -2.23480044,  1.09955852,
       -0.75610871, -0.30932222,  1.41815946,  1.08449065, -1.90839088])
```

In [80]:
```
# Poverty by Country (1998-2014)
stats.describe(povAvgByCnty)
```

Out[80]:
```
DescribeResult(nobs=19, minmax=(0.7125000000000002, 54.75), mean=13.3123501584
84447, variance=133.91390293515113, skewness=2.499337951809888, kurtosis=6.8560
764402030845)
```

In [101]:
```
# T-test
tStat, pval = stats.ttest_1samp(povAvgByCnty, 20)
print('t-statistic = %6.3f pvalue = %6.4f' % (tStat, pval))
```

```
t-statistic = -2.519 pvalue = 0.0214
```

In [94]:
```
# T-statistics and P-value
m = 20  # test mean value
n, (smin, smax), sm, sv, ss, sk = stats.describe(povAvgByCnty)

tt = (sm-m)/np.sqrt(sv/float(n))  # t-statistic for mean
pval = stats.t.sf(np.abs(tt), n-1)*2  # two-sided pvalue = Prob(abs(t)>tt)

sstr = 'mean = %6.4f, variance = %6.4f, skew = %6.4f, kurtosis = %6.4f'
print ('t-statistic = %6.3f pvalue = %6.4f' % (tt, pval))
```

```
t-statistic = -2.519 pvalue = 0.0214
```

In [113]:
```
# Percent Point Function (Inverse of cdf)
crit01, crit05, crit10 = stats.t.ppf([.99, .95, .90], 30)
print ('critical-t values from ppf at 1%%, 5%% and 10%% %8.4f %8.4f %8.4f'% (crit
```

```
critical-t values from ppf at 1%, 5% and 10%   2.4573   1.6973   1.3104
```

In [109]:
```
# ppf for Z value
crit01, crit05, crit10 = stats.norm.ppf([.99, .95, .90])
print ('critical-z values from ppf at 1%%, 5%% and 10%% %8.4f %8.4f %8.4f'% (crit
```

```
critical-z values from ppf at 1%, 5% and 10%   2.3263   1.6449   1.2816
```

In [131]:
```
# Normality Tests (skewtest)
x = stats.norm.rvs(size=30)
print ('normal skewtest teststat = %6.3f pvalue = %6.4f' % stats.skewtest(x))
```

```
normal skewtest teststat = -1.016 pvalue = 0.3095
```

In [132]:
```
# Normality Test (kurtosistest)
# ! runs only when n >=20 and only length-1 array
print ('normal kurtosistest teststat = %6.3f pvalue = %6.4f' % stats.kurtosistest
```

```
normal kurtosistest teststat =  0.293 pvalue = 0.7692
```

In [133]:
```
# Normality Test (skewtest + kurtosistest)
print ('normaltest teststat = %6.3f pvalue = %6.4f' % stats.normaltest(x))
```

```
normaltest teststat =  1.119 pvalue = 0.5716
```

Comparing Two Samples

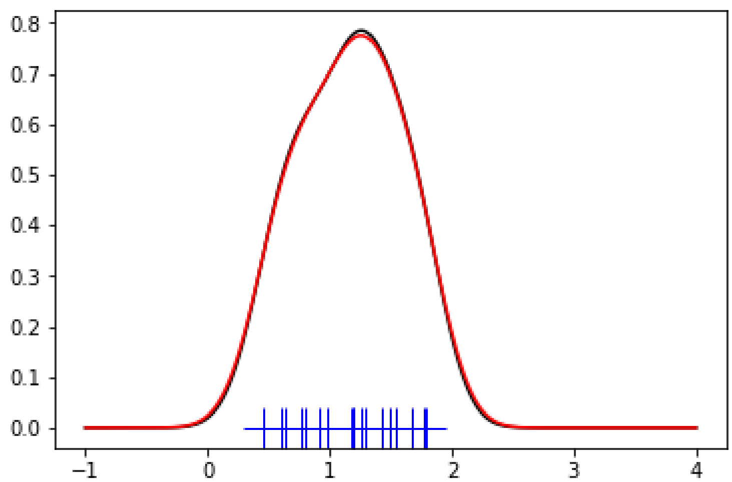
In [134]:
```
# incLowAvg = np.mean(incLowN)
# incHighAvg = np.mean(incHighN)
stats.ttest_ind(incLowAvg, incHighAvg)
```

Out[134]:
```
Ttest_indResult(statistic=-78.012586954984542, pvalue=4.402847288611941e-38)
```

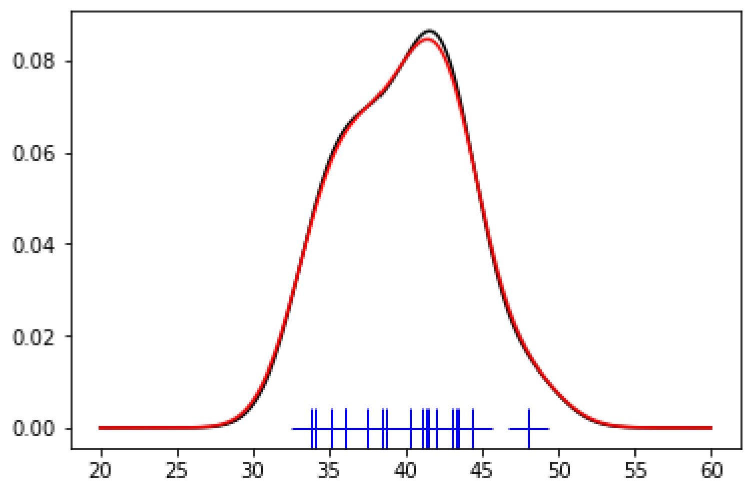
Kernel Density Estimation

In [148]:

```
'''
A histogram is a useful tool for visualization
(mainly because everyone understands it),
but doesn’t use the available data very efficiently.
'''

from scipy import stats
import matplotlib.pyplot as plt

# sample: last 20 years of income share data (row mean)
incLowAvg = np.mean(incLowN, axis=1)
incLowAvg = incLowAvg[~np.isnan(incLowAvg)]
x1 = incLowAvg

kde1 = stats.gaussian_kde(x1)
kde2 = stats.gaussian_kde(x1, bw_method='silverman')
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x1, np.zeros(x1.shape), 'b+', ms=20)  # rug plot
x_eval = np.linspace(-1, 4, num=200)
ax.plot(x_eval, kde1(x_eval), 'k-', label="Scott's Rule")
ax.plot(x_eval, kde2(x_eval), 'r-', label="Silverman's Rule")
plt.show()
```



In [145]:
```
# sample: last 20 years of income share data (row mean)
incHighAvg = np.mean(incHighN, axis=1)
incHighAvg = incHighAvg[~np.isnan(incHighAvg)]
x1 = incHighAvg

kde1 = stats.gaussian_kde(x1)
kde2 = stats.gaussian_kde(x1, bw_method='silverman')
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x1, np.zeros(x1.shape), 'b+', ms=20)  # rug plot
x_eval = np.linspace(20, 60, num=200)
ax.plot(x_eval, kde1(x_eval), 'k-', label="Scott's Rule")
ax.plot(x_eval, kde2(x_eval), 'r-', label="Silverman's Rule")
plt.show()
```



In [140]:
```
incHighAvg
```

Out[140]:
```
38    35.205882
39    38.648889
40    42.040000
41    48.015000
42    41.401765
43    43.065000
44    43.287500
45    36.089412
46    37.504000
47    40.290667
48    36.118750
49    38.363529
50    44.293333
51    34.110000
53    41.096429
54    43.355714
55    41.330667
56    33.824375
dtype: float64
```

In [141]:
```
incLowAvg
```

Out[141]:
```
57    1.180000
58    1.668889
59    1.285000
60    0.595000
61    0.798824
62    0.980000
63    0.768750
64    1.431176
65    1.762000
66    1.186667
67    0.458750
68    1.255294
69    0.910667
70    1.540000
72    0.627857
73    1.488571
74    1.191333
75    1.780625
dtype: float64
```

In [187]:
```
df = np.column_stack((povAvg,incHighAvg))
df
```

Out[187]:
```
array([[  4.88411765,  35.20588235],
       [  6.17      ,  38.64888889],
       [ 11.168     ,  42.04      ],
       [ 54.75      ,  48.015     ],
       [  8.92705882,  41.40176471],
       [ 14.4       ,  43.065     ],
       [ 21.448125  ,  43.2875    ],
       [  9.14588235,  36.08941176],
       [ 13.768     ,  37.504     ],
       [ 11.71      ,  40.29066667],
       [ 13.87      ,  36.11875   ],
       [ 10.55058824,  38.36352941],
       [  8.79533333,  44.29333333],
       [ 14.        ,  34.11      ],
       [ 23.38      ,          nan],
       [ 16.05571429,  41.09642857],
       [  2.18      ,  43.35571429],
       [  7.01933333,  41.33066667],
       [  0.7125    ,  33.824375  ]])
```

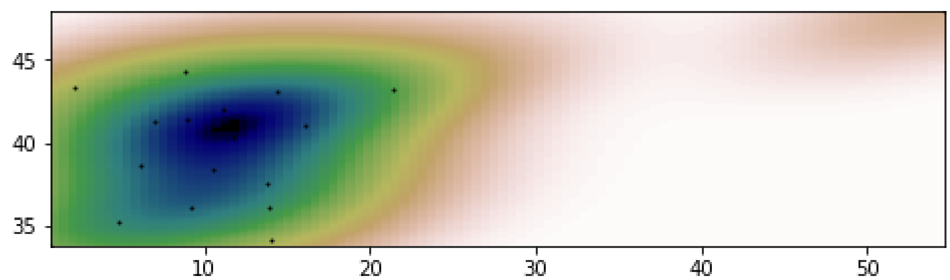
In [190]:
```
mask = ~np.any(np.isnan(df), axis=1)
df = df[mask]
df
```

Out[190]:
```
array([[  4.88411765,  35.20588235],
       [  6.17      ,  38.64888889],
       [ 11.168     ,  42.04      ],
       [ 54.75      ,  48.015     ],
       [  8.92705882,  41.40176471],
       [ 14.4       ,  43.065     ],
       [ 21.448125  ,  43.2875    ],
       [  9.14588235,  36.08941176],
       [ 13.768     ,  37.504     ],
       [ 11.71      ,  40.29066667],
       [ 13.87      ,  36.11875   ],
       [ 10.55058824,  38.36352941],
       [  8.79533333,  44.29333333],
       [ 14.        ,  34.11      ],
       [ 16.05571429,  41.09642857],
       [  2.18      ,  43.35571429],
       [  7.01933333,  41.33066667],
       [  0.7125    ,  33.824375  ]])
```

Multivariate Estimation

In [193]:

```
# Poverty Rate vs. Income Share Top 10% Population
m1 = df[:,0]
m2 = df[:,1]
xmin = m1.min()
xmax = m1.max()
ymin = m2.min()
ymax = m2.max()

# apply the KDE to the data
X, Y = np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
positions = np.vstack([X.ravel(), Y.ravel()])
values = np.vstack([m1, m2])
kernel = stats.gaussian_kde(values)
Z = np.reshape(kernel.evaluate(positions).T, X.shape)

# plot the estimated bivariate distribution as a colormap, and plot the individua
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111)

ax.imshow(np.rot90(Z), cmap=plt.cm.gist_earth_r,extent=[xmin, xmax, ymin, ymax])
ax.plot(m1, m2, 'k.', markersize=2)

ax.set_xlim([xmin, xmax])
ax.set_ylim([ymin, ymax])

plt.show()
```



In [195]:

```
from scipy.stats.stats import pearsonr
pearsonr(m1,m2)
```

Out[195]: (0.55476334137392136, 0.016869737454508715)