In [1]:

```
'''
K-Means
rnk, muk(random, 그 후 데이터에서 random) => Parameter, K => hyperparameter
data = point = instance => random(N)
'''
from random import randrange

K = 3
N = 100 # [?,?], [],[]

centroid = list()
for _ in range(K):
    centroid.append([randrange(1,100), randrange(1,100)])

data = list()
for _ in range(N):
    data.append([randrange(1,100), randrange(1,100)])
    #d1 = [t1]
```

In [2]:

```
data[0], centroid[2], data[0], centroid
```

Out[2]:

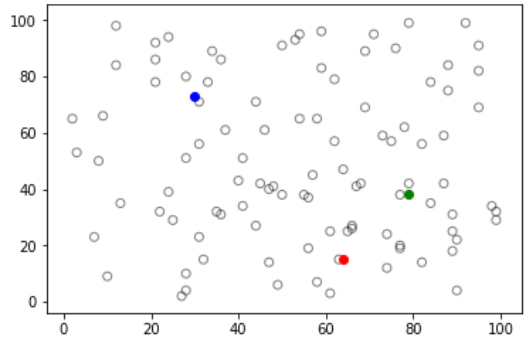
```
([25, 29], [30, 73], [25, 29], [[64, 15], [79, 38], [30, 73]])
```

In [4]:

```
import matplotlib.pyplot as plt

colorMap = ["r","g","b","k"]

for _ in data:
    plt.scatter(_[0], _[1], facecolor="none", alpha=.5, edgecolor=colorMap[-1])

for i, _ in enumerate(centroid):
    plt.scatter(_[0], _[1], color=colorMap[i])
plt.show
```

Out[4]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [5]:

```
# def distance(x1,x2):
#     return sqrt((x1[0]-x2[0])**2 + (x1[1]-x2[1])**2)

# def angle(x1,x2):
#     return (x1[0]*x2[0] + x1[1]*x2[1]) / (distance(x1,(0,0))*distance(x2,(0,0)))
```

In [6]:

```
from math import sqrt

def distance(x1,x2):
    _sum = 0
    for i in range(len(x1)):
        _sum += (x1[i]-x2[i])**2
    return sqrt(_sum)

def angle(x1,x2):
    _innerProduct = 0
    for i in range(len(x1)):
        _innerProduct += x1[i]*x2[i]
    x1VecLength = distance(x1, [0 for _ in range(len(x1))])
    x2VecLength = distance(x2, [0 for _ in range(len(x2))])
    return _innerProduct / (x1VecLength*x2VecLength)
```

In [7]:

```
distance((1,1),(1,1)), angle((1,1),(1,1))
```

Out[7]:

```
(0.0, 0.9999999999999998)
```

In [152]:

```
'''
EM algorithm
E - expectation = rnk assignment
M - Maximization = centroid update

def expectation(x,c):
    #rnk = list() => (0,0,1) assign
    candidates = list()
    for _ in c:
        candidates.append(distance(x, _))
    return candidates.index(min(candidates))

def expectation(x,c):
    #rnk = list() => (0,0,1) assign
    candidates = list()
    for _ in c:
        candidates.append(angle(x, _))
    return candidates.index(max(candidates))
'''
```

Out[152]:

```
'WnEM algorithmWnE - expectation = rnk assignmentWnM - Maximization = centroid upd
ateWn'
```

In [8]:

```
def expectation(x,c, opt=False):
    #rnk = list() => (0,0,1) assign
    candidates = list()
    nearest = distance if not opt else angle
    best = min if not opt else max
    for _ in c:
        candidates.append(nearest(x, _))
    return candidates.index(best(candidates))
```

In [9]:

```
data[0], centroid, expectation(data[0], centroid)
```

Out[9]:

```
([25, 29], [[64, 15], [79, 38], [30, 73]], 0)
```

In [157]:

```
data[0], centroid[2], data[0], centroid
```

Out[157]:

```
([5, 52], [4, 26], [5, 52], [[66, 45], [74, 38], [4, 26]])
```

In [16]:

```
def maximization(X):
    _sum = [0 for _ in range(len(X[0]))]
    D = len(X)
    for x in X:
        for i in range(len(x)):
            _sum[i] += x[i]
    return [_/D for _ in _sum]
#return (sum([x[0] for x in X]) / len(X), sum([x[1] for x in X]) / len(X))
```

In [159]:

```
'''
muk = 클러스터 내 데이터의 합 / 클러스터 내 데이터 갯수

rnk =
X0 => [0,0,0]
X1,
X2, ...         list

def sse(X, c):
    error = 0
    for x in X:
        error += distance(x,c)
    return error

def sse(X, c):
    error = 0
    for x in X:
        error += angle(x,c)
    return error
'''
```

Out[159]:

'Wnmuk = 클러스터 내 데이터의 합 / 클러스터 내 데이터 갯수Wn'

In [11]:

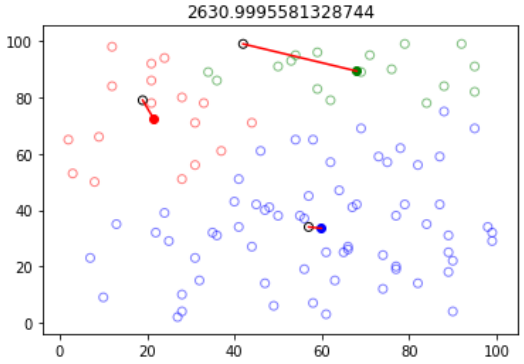
```
def sse(X, c, opt=False):
    error = 0
    nearest = distance if not opt else angle
    for x in X:
        error += nearest(x,c)
    return error
```

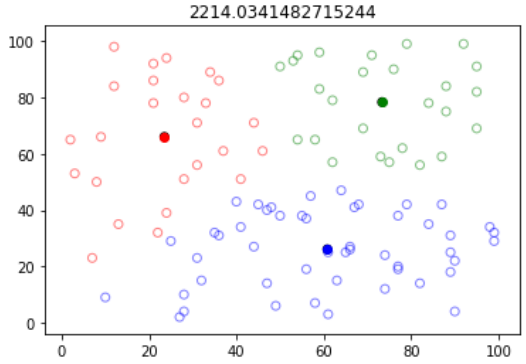
In [17]:

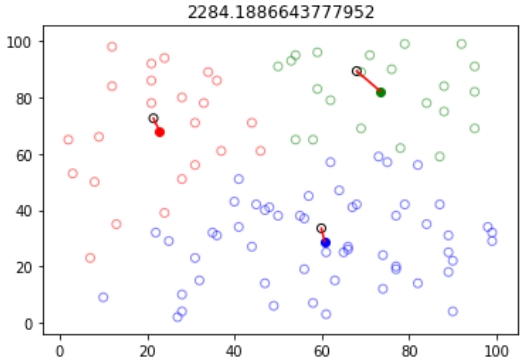
```
errorRate = list()

for _ in range(10):
    rnk = list(list(0 for _ in range(K)) for _ in range(N))

    for i in range(N):
            j = expectation(data[i], centroid)
            rnk[i][j] = 1

    _sse = 0

    for k in range(K):
        X = [data[i] for i in range(N) if rnk[i][k]]  #<-sub data
        _sse += sse(X, centroid[k])
        centroid[k] = maximization(X)
    errorRate.append(_sse)
```

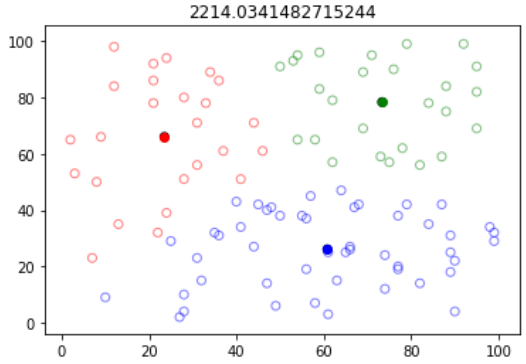
In [18]:

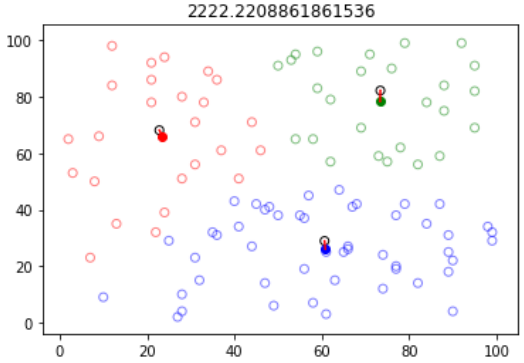
```
errorRate = list()
centroid = list()

for _ in range(K):
    centroid.append([randrange(1,100), randrange(1,100)])
for _ in range(10):
    rnk = list(list(0 for _ in range(K)) for _ in range(N))

    for i in range(N):
        j = expectation(data[i], centroid)
        rnk[i][j] = 1

    _sse = 0
    oldCentroid = list()

    for k in range(K):
        X = [data[i] for i in range(N) if rnk[i][k]]  #<-sub data
        _sse += sse(X, centroid[k])
        oldCentroid.append(centroid[k])
        centroid[k] = maximization(X)
    errorRate.append(_sse)

    for i in range(N):
    #plt.scatter(_[0], _[1], facecolor="none", alpha=.5, edgecolor=colorMap[-1])
        plt.scatter(data[i][0], data[i][1], facecolor="none", alpha=.5, edgecolor=colorMap[rnk[i
].index(max(rnk[i]))])

    for i, _ in enumerate(centroid):
        plt.plot((oldCentroid[i][0], _[0]), (oldCentroid[i][1], _[1]),"r-")
        plt.scatter(oldCentroid[i][0], oldCentroid[i][1], edgecolors="k", facecolor="none", colo
r=colorMap[i])
        plt.scatter(_[0], _[1], color=colorMap[i])
    plt.title(_sse)
    plt.show()

plt.plot(range(10), errorRate,  "r-")
plt.show()
#print(errorRate[-1])
```

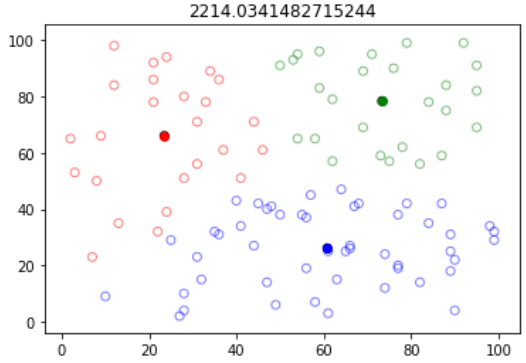

2630.9995581328744

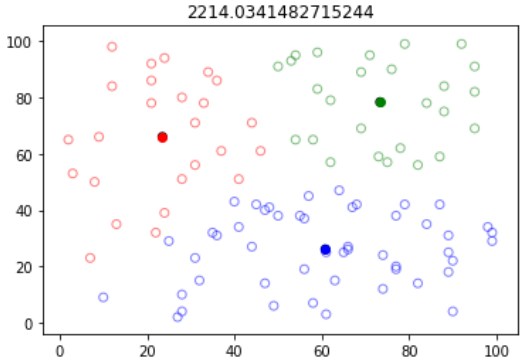

2214.0341482715244

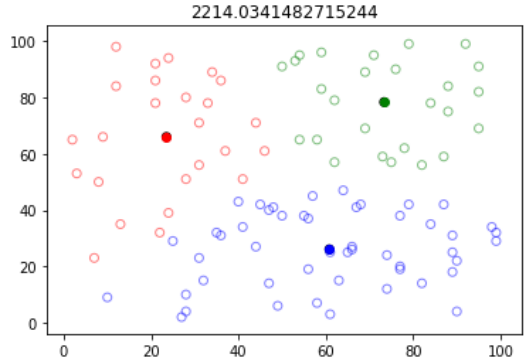

2284.1886643777952

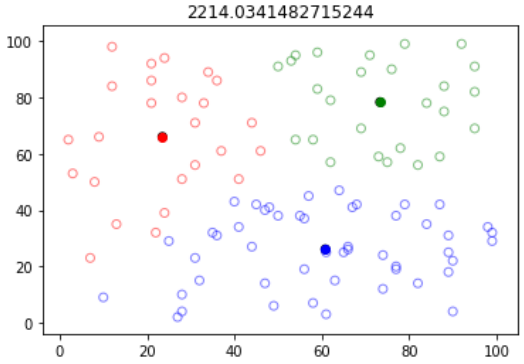

2214.0341482715244

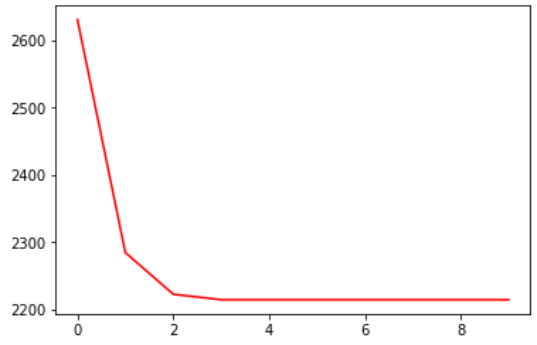

2222.2208861861536

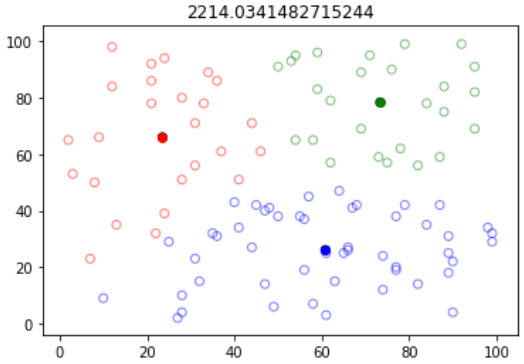

2214.0341482715244


2214.0341482715244


2214.0341482715244

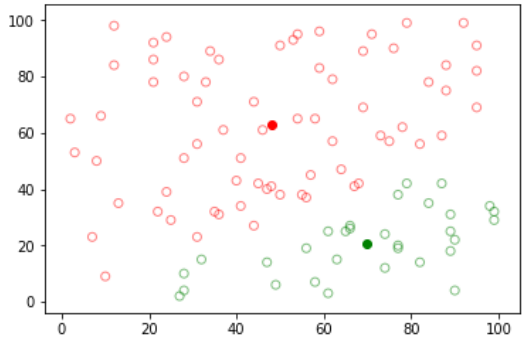

2214.0341482715244

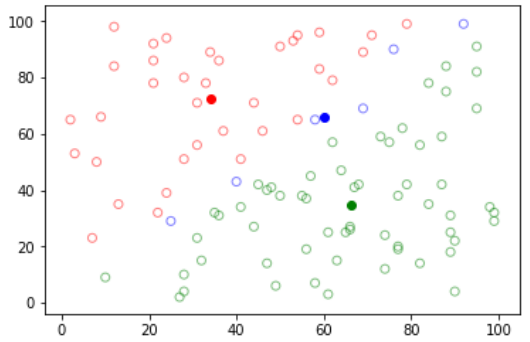


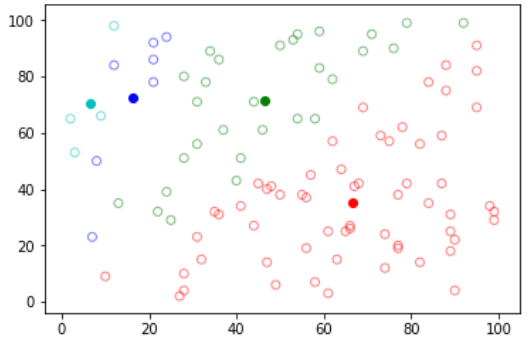

2214.0341482715244

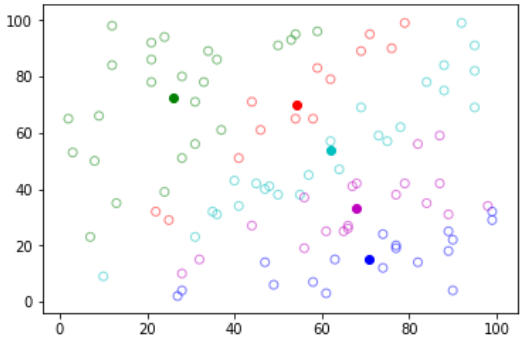
In [20]:

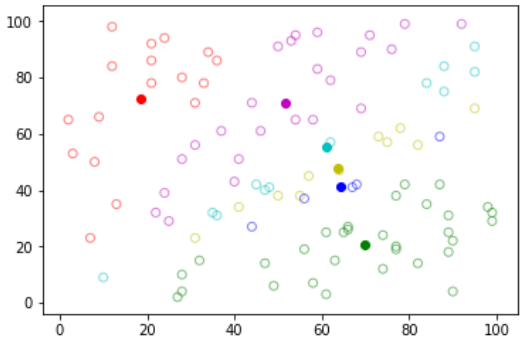
```
colorMap = ["r","g","b","c","m","y","C0","C1","C2","k"]

errorRate = list()
for K in range(2,10):
    centroid = list()
    for _ in range(K):
        centroid.append([randrange(1,100), randrange(1,100)])

    for _ in range(10):
        rnk = list(list(0 for _ in range(K)) for _ in range(N))

        for i in range(N):
            j = expectation(data[i], centroid, True)
            rnk[i][j] = 1

    _sse = 0
    oldCentroid = list()

    for k in range(K):
        X = [data[i] for i in range(N) if rnk[i][k]]  #<-sub data
        _sse += sse(X, centroid[k], True)
        oldCentroid.append(centroid[k])
        centroid[k] = maximization(X)
    errorRate.append(_sse)

    for i in range(N):
    #plt.scatter(_[0], _[1], facecolor="none", alpha=.5, edgecolor=colorMap[-1])
        plt.scatter(data[i][0], data[i][1], facecolor="none", alpha=.5, edgecolor=colorMap[rnk[i
].index(max(rnk[i]))])

    for i, _ in enumerate(centroid):
#         plt.plot((oldCentroid[i][0], _[0]), (oldCentroid[i][1], _[1]),"r-")
#         plt.scatter(oldCentroid[i][0], oldCentroid[i][1], edgecolors="k", facecolor="none", co
lor=colorMap[i])
        plt.scatter(_[0], _[1], color=colorMap[i])
    #plt.title(_sse)
    plt.show()

plt.plot(range(len(errorRate)), errorRate,  "r-")
plt.show
#print(errorRate[-1])
```

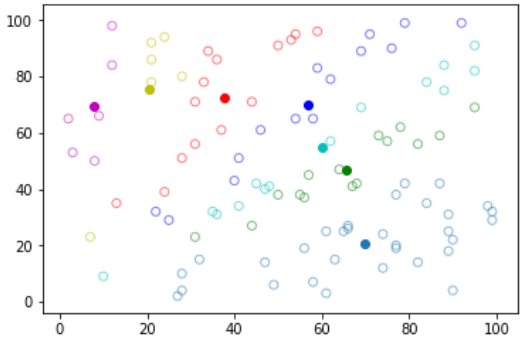


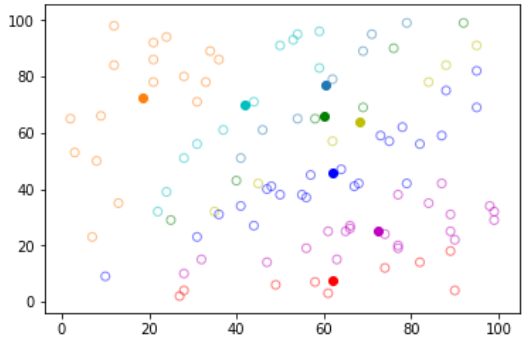


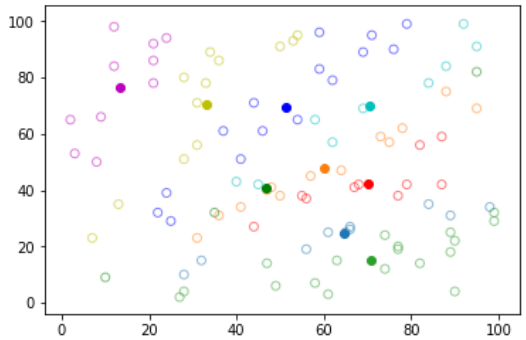






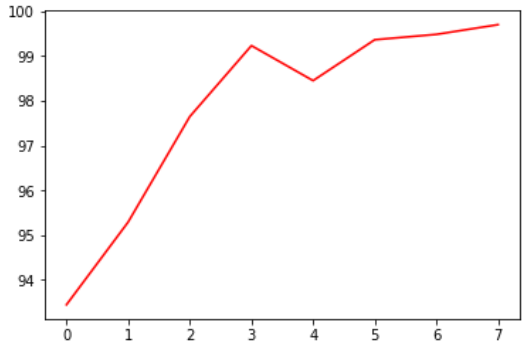






Out[20]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [ ]:

```
2158.953509681772
```

In [17]:

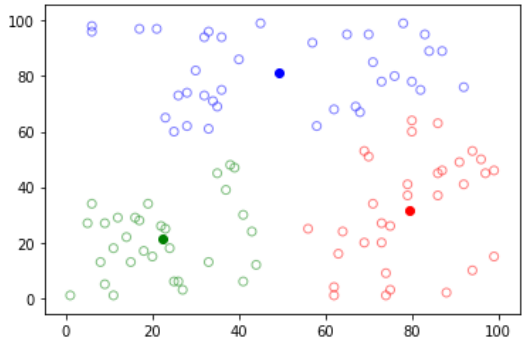
```
rnk
```

Out[17]:

```
[[0, 1, 0],
 [1, 0, 0],
 [0, 1, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 1],
 [0, 1, 0],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 1, 0],
 [0, 1, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 1, 0],
 [0, 0, 1],
 [0, 1, 0],
 [0, 1, 0],
 [0, 0, 1],
 [0, 1, 0],
 [0, 1, 0],
 [0, 0, 1],
 [0, 0, 1],
 [0, 0, 1],
 [0, 1, 0],
 [0, 0, 1],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 1],
 [0, 1, 0],
 [1, 0, 0],
 [0, 0, 1],
 [1, 0, 0],
 [0, 1, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 0, 1],
 [1, 0, 0],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 1],
 [1, 0, 0],
 [1, 0, 0],
 [0, 0, 1]]
```

In [18]:

```
import matplotlib.pyplot as plt

colorMap = ["r","g","b","k"]

#for _ in data:
for i in range(N):
    #plt.scatter(_[0], _[1], facecolor="none", alpha=.5, edgecolor=colorMap[-1])
    plt.scatter(data[i][0], data[i][1], facecolor="none", alpha=.5, edgecolor=colorMap[rnk[i].in
dex(max(rnk[i]))])

for i, _ in enumerate(centroid):
    plt.scatter(_[0], _[1], color=colorMap[i])
plt.show()
```



In [178]:

```
documents = [
    "This little kitty came to play when I was eating at a restaurant.",
    "Merley has the best squooshy kitten belly.",
    "Google Translate app is incredible",
    "If you open 100 tab in google you get a smiley face.",
    "Best cat photo I've ever taken.",
    "Climbing ninja cat.",
    "Impressed with google map feedback",
    "Key promoter extension for Google Chrome."
]
```

In [167]:

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from string import punctuation
import re
```

In [179]:

```
from collections import defaultdict

DTM = defaultdict(lambda:defaultdict(int))
for i,d in enumerate(documents):
    for t in word_tokenize(d.lower()):
        if t not in stopwords.words("english") and not re.search(r"[%s]" % re.escape(punctuatio
n), t):
            DTM[i][t] += 1
```

In [180]:

```
len(DTM), DTM[0]
```

Out[180]:

```
(8,
 defaultdict(int,
             {'little': 1,
              'kitty': 1,
              'came': 1,
              'play': 1,
              'eating': 1,
              'restaurant': 1}))
```

In [183]:

```
TDM = defaultdict(lambda:defaultdict(int))
for d, termList in DTM.items():
    print(d, termList)
    for t, f in termList.items():
        TDM[t][d] = f
```

```
0 defaultdict(<class 'int'>, {'little': 1, 'kitty': 1, 'came': 1, 'play': 1, 'eati
ng': 1, 'restaurant': 1})
1 defaultdict(<class 'int'>, {'merley': 1, 'best': 1, 'squooshy': 1, 'kitten': 1,
'belly': 1})
2 defaultdict(<class 'int'>, {'google': 1, 'translate': 1, 'app': 1, 'incredible':
1})
3 defaultdict(<class 'int'>, {'open': 1, '100': 1, 'tab': 1, 'google': 1, 'get':
1, 'smiley': 1, 'face': 1})
4 defaultdict(<class 'int'>, {'best': 1, 'cat': 1, 'photo': 1, 'ever': 1, 'taken':
1})
5 defaultdict(<class 'int'>, {'climbing': 1, 'ninja': 1, 'cat': 1})
6 defaultdict(<class 'int'>, {'impressed': 1, 'google': 1, 'map': 1, 'feedback':
1})
7 defaultdict(<class 'int'>, {'key': 1, 'promoter': 1, 'extension': 1, 'google':
1, 'chrome': 1})
```

In [182]:

```
len(TDM)
```

Out[182]: