In [3]:

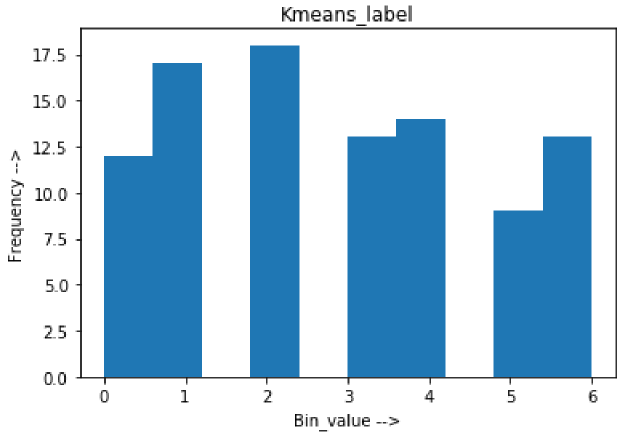
```
import pandas as pd
from copy import deepcopy
import numpy as np
from matplotlib import pyplot as plt
garbage_data = pd.read_csv("f.csv",index_col = ["bin_no"])
print(garbage_data)
print(garbage_data.head(6))
print(garbage_data.shape)
```

```
        region  latitude  longitude  percentage_full  config_status  \
bin_no
1           45      78.9      45.60               98              1
2           56      56.8     198.45               56              1
3           23      65.8     157.80               45              0
4           45      56.4     148.50               32              0
5           45      12.5     157.40               95              1
...        ...       ...        ...              ...            ...
2           45      78.9      45.60               98              1
3           56      56.8     198.45               56              1
4           23      65.8     157.80               45              0
5           45      56.4     148.50               32              0
6           45      12.5     157.40               95              1

        fill_time  clean_time
bin_no
1          123546      120000
2          125089      120500
3          124056      121000
4          124546      121500
5          124564      122000
...           ...         ...
2          123546      120500
3          125089      121000
4          124056      121500
5          124546      122000
6          124564      122500

[96 rows x 7 columns]
        region  latitude  longitude  percentage_full  config_status  \
bin_no
1           45      78.9      45.60               98              1
2           56      56.8     198.45               56              1
3           23      65.8     157.80               45              0
4           45      56.4     148.50               32              0
5           45      12.5     157.40               95              1
6           45      56.4     148.50               32              0

        fill_time  clean_time
bin_no
1          123546      120000
2          125089      120500
3          124056      121000
4          124546      121500
5          124564      122000
6          124546      122500
(96, 7)
```

In [4]:

```
from sklearn.cluster import KMeans
clusters = 7
kmeans = KMeans(n_clusters = clusters)
kmeans.fit(garbage_data)
print(kmeans.labels_)
plt.hist(kmeans.labels_)
plt.title('Kmeans_label')
plt.xlabel('Bin_value -->')
plt.ylabel('Frequency -->')
plt.show()
```

```
[3 0 3 4 1 1 6 0 2 5 2 0 3 4 6 1 2 0 3 4 6 1 4 0 3 1 2 2 4 5 6 1 2 2 4 5 6
 1 6 2 2 5 3 0 3 4 1 5 3 0 3 4 1 5 6 0 2 1 2 0 3 4 6 1 2 0 3 4 6 1 4 0 3 1
 2 2 4 5 6 1 2 2 4 5 6 1 6 2 2 5 0 3 4 6 1 1]
```



In [5]:

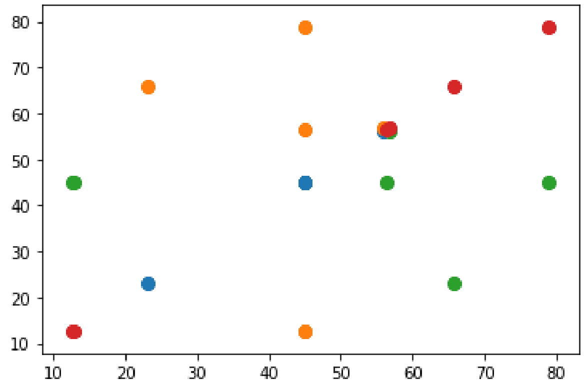
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(garbage_data)
```

Out[5]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [6]:

```
plt.scatter(garbage_data.iloc[:, 0], garbage_data.iloc[:, 0] , s=50, cmap='virdis')
plt.scatter(garbage_data.iloc[:, 0], garbage_data.iloc[:, 1] , s=50, cmap='virdis')
plt.scatter(garbage_data.iloc[:, 1], garbage_data.iloc[:, 0] , s=50, cmap='virdis')
plt.scatter(garbage_data.iloc[:, 1], garbage_data.iloc[:, 1] , s=50, cmap='virdis')
plt.show()
```



In [7]:

```
clusters = kmeans.cluster_centers_
```

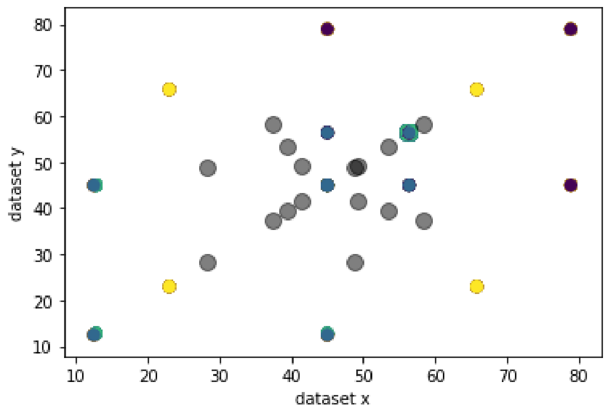
In [8]:

```
y_km = kmeans.fit_predict(garbage_data)
print(y_km)
```

```
[0 2 3 3 1 1 3 2 0 1 0 2 3 1 3 1 0 2 3 1 3 1 3 2 0 1 0 0 3 2 3 1 0 0 3 2 3
 1 3 0 0 1 0 2 3 3 1 1 0 2 3 3 1 1 3 2 0 1 0 2 3 1 3 1 0 2 3 1 3 1 3 2 0 1
 0 0 3 2 3 1 0 0 3 2 3 1 3 0 0 1 2 0 2 3 1 1]
```

In [9]:

```
plt.scatter(garbage_data.iloc[:, 0], garbage_data.iloc[:, 0], c=y_km, s=50, cmap='viridis')
plt.scatter(garbage_data.iloc[:, 0], garbage_data.iloc[:, 1], c=y_km, s=50, cmap='viridis')
plt.scatter(garbage_data.iloc[:, 1], garbage_data.iloc[:, 0], c=y_km, s=50, cmap='viridis')
plt.scatter(garbage_data.iloc[:, 1], garbage_data.iloc[:, 1], c=y_km, s=50, cmap='viridis')

plt.scatter(clusters[:, 0], clusters[:, 0], c='black', s=100, alpha=0.5);
plt.scatter(clusters[:, 0], clusters[:, 1], c='black', s=100, alpha=0.5);
plt.scatter(clusters[:, 1], clusters[:, 0], c='black', s=100, alpha=0.5);
plt.scatter(clusters[:, 1], clusters[:, 1], c='black', s=100, alpha=0.5);
plt.title('')
plt.xlabel('dataset x')
plt.ylabel('dataset y')
plt.show()
```



In [10]:

```
print("The centroids of clusters are given by :\n\n",clusters)
```

```
The centroids of clusters are given by :

 [[3.74137931e+01 5.83172414e+01 1.18065517e+02 7.22758621e+01
  5.51724138e-01 1.24035931e+05 1.21448276e+05]
 [4.92307692e+01 4.14153846e+01 1.68523077e+02 5.72692308e+01
  7.30769231e-01 1.24967038e+05 1.22153846e+05]
 [3.95000000e+01 5.34000000e+01 1.27325000e+02 6.75000000e+01
  5.00000000e-01 1.24178000e+05 1.20145833e+05]
 [4.88823529e+01 2.83294118e+01 1.64252941e+02 6.18235294e+01
  1.00000000e+00 1.25442294e+05 1.20735294e+05]]
```

In [11]:

```
x=clusters[:,[5]]
print("standard deviation of x : ", np.std(x), "\nthe member set of x differ from 6 minutes 16 sec
onds, which is difference of dustbin fill time from one dustbin to other bin")
print("Range of x : ",np.ptp(x))
```

```
standard deviation of x :  576.18850006617
the member set of x differ from 6 minutes 16 seconds, which is difference of dustbin fill time fro
m one dustbin to other bin
Range of x :  1406.3630831643095
```

In [12]:

```
y=clusters[:,[6]]
print("standard deviation of y : ", np.std(y), "\nthe member set of y differ from 7 minutes 53 sec
onds which is difference of dustbin clean time from one dustbin to other bin")
print("Range of y : ",np.ptp(y))
```

```
standard deviation of y :  753.9230037016786
the member set of y differ from 7 minutes 53 seconds which is difference of dustbin clean time
from one dustbin to other bin
Range of y :  2008.0128205128276
```

In [13]:

```
out = np.subtract(x,y)
print ("Output array:\n",out)
r = np.ptp(out)
print("range of final : ", r,"\nwhich is 23 minutes and 35 seconds, which is difference b/w
highest and lowest of data set difference \nIn our application more the range, greater possabality
of being filled for some time \nhence it should be minimized")
```

```
Output array:
 [[2587.65517241]
 [2813.19230769]
 [4032.16666667]
 [4707.        ]]
range of final :  2119.3448275862174
which is 23 minutes and 35 seconds, which is difference b/w highest and lowest of data set
difference
In our application more the range, greater possabality of being filled for some time
hence it should be minimized
```

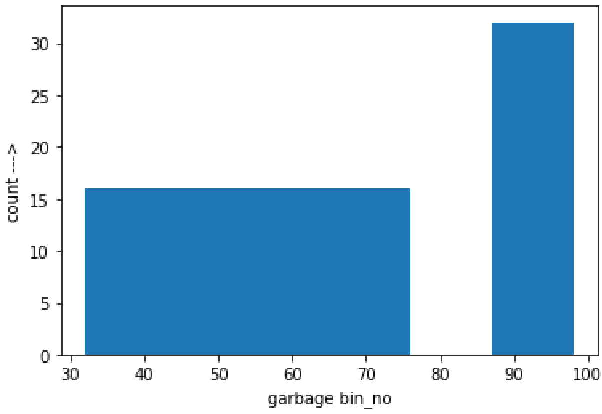
In [14]:

```
print("standard deviation of final averaged is : ", np.std(out),"\nwhich shows that dustbins are c
leaned before 9 minutes and 11 seconds before they are filled")
print("Hence we have to minimize standard deviation as much as possible to minimize the situations
\nwhere dustbin stays full for some time")
```

```
standard deviation of final averaged is :  871.6689826739457
which shows that dustbins are cleaned before 9 minutes and 11 seconds before they are filled
Hence we have to minimize standard deviation as much as possible to minimize the situations
where dustbin stays full for some time
```

In [15]:

```
import matplotlib.pyplot as plt
plt.hist(garbage_data['percentage_full'],6)
plt.ylabel('count --->')
plt.xlabel('garbage bin_no')
plt.show()
print("Hence we see that dustbin no 6 is nothing but, dustbin near cafe is getting filled soon,\nH
ence dustbin no:6 is having high anamoly and new dustbin allocation is required near by.")
```




```
Hence we see that dustbin no 6 is nothing but, dustbin near cafe is getting filled soon,
Hence dustbin no:6 is having high anamoly and new dustbin allocation is required near by.
```

In [16]:

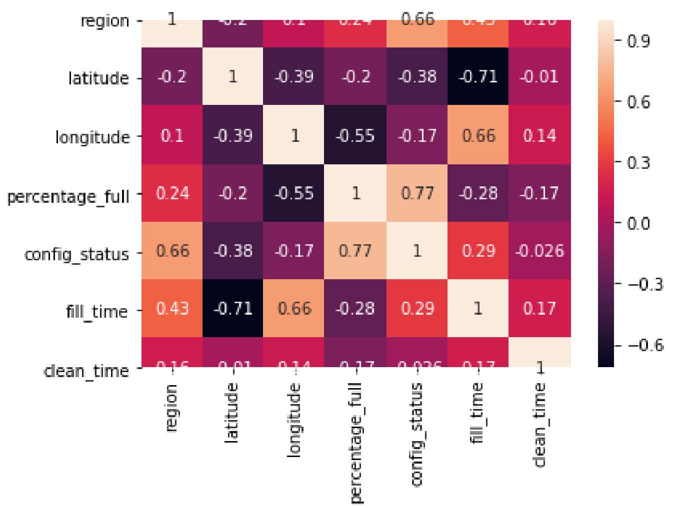
```
l0=clusters[:,[0]]
l1=clusters[:,[1]]
l2=clusters[:,[2]]
avg0 = sum(l0)/len(l0)
avg1 = sum(l1)/len(l1)
avg2 = sum(l2)/len(l2)
print("New dustbin allocation region is suggested by algorithm at :\n")
```

```
print("REGION     : ",avg0)
print("LATTITUDE : ",avg1)
print("LONGITUDE : ",avg2)
print("\nHence on new dustbin allocation at given landmark and once again collecting data and perf
orming cyclic clustering,\nstandard deviation and range can be reduced.")
```

```
New dustbin allocation region is suggested by algorithm at :

REGION     :  [43.75672882]
LATTITUDE :  [45.36550944]
LONGITUDE :  [144.54163384]

Hence on new dustbin allocation at given landmark and once again collecting data and performing cy
clic clustering,
standard deviation and range can be reduced.
```

In [17]:

```python
import seaborn as sns

# generating correlation heatmap
sns.heatmap(garbage_data.corr(), annot = True)

# posting correlation heatmap to output console
plt.show()
```



In [18]:

```python
print("Hence inorder to overcome this more number of attributes problem\nwe consider PCA method\nt
o reduce loss,remove noise and to restrict the attributes. ")
```

```
Hence inorder to overcome this more number of attributes problem
we consider PCA method
to reduce loss,remove noise and to restrict the attributes.
```

In [19]:

```python
print("We make use of RandomForestClassifier for attribute reduction ,pca analysis.")
```

```
We make use of RandomForestClassifier for attribute reduction ,pca analysis.
```

In [20]:

```python
import numpy as np
import pandas as pd
from copy import deepcopy
from matplotlib import pyplot as plt
names = ['bin_no','region','latitude','longitude','percentage_full','config_status','fill_time','cl
ean_time']
dataset= pd.read_csv("f2.csv",names=names)
dataset.head()
```

```
dataset.head()
```

Out[20]:

	bin_no	region	latitude	longitude	percentage_full	config_status	fill_time	clean_time
0	1	45	78.9	45.60	98	1	123546	120000
1	2	56	56.8	198.45	56	1	125089	120500
2	3	23	65.8	157.80	45	0	124056	121000
3	4	45	56.4	148.50	32	0	124546	121500
4	5	45	12.5	157.40	95	1	124564	122000

In [35]:

```
q='config_status'
X = dataset.drop(q, 1)
y = dataset[q]
```

In [36]:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [37]:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [38]:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
explained_variance = pca.explained_variance_ratio_
```

In [39]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

In [40]:

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The defa
ult value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [41]:

```
from sklearn.metrics import confusion_matrix
```

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)
q='region'
print("Accuracy in percentage = ",accuracy_score(y_test, y_pred)*100)
```

```
[[ 6  0]
 [ 0 14]]
Accuracy in percentage =  100.0
```

In [42]:

```python
print("We use elbow method to find the cluster choosen value is right\nAt what value dimensionality reduction can be done.")
```

```
We use elbow method to find the cluster choosen value is right
At what value dimensionality reduction can be done.
```

In [43]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [44]:

```python
df = pd.read_csv("f.csv")
df.head()
```

Out[44]:

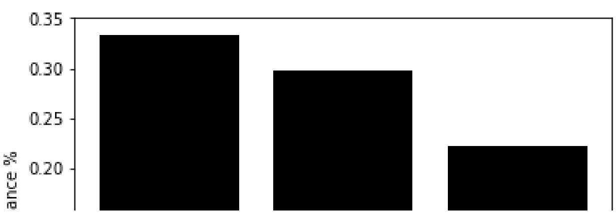
	bin_no	region	latitude	longitude	percentage_full	config_status	fill_time	clean_time
0	1	45	78.9	45.60	98	1	123546	120000
1	2	56	56.8	198.45	56	1	125089	120500
2	3	23	65.8	157.80	45	0	124056	121000
3	4	45	56.4	148.50	32	0	124546	121500
4	5	45	12.5	157.40	95	1	124564	122000

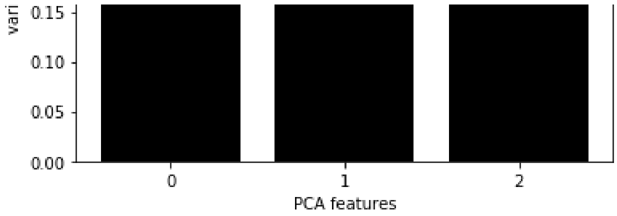
In [45]:

```python
X_std = StandardScaler().fit_transform(df)
pca = PCA(n_components=3)
principalComponents = pca.fit_transform(X_std)
```

In [46]:

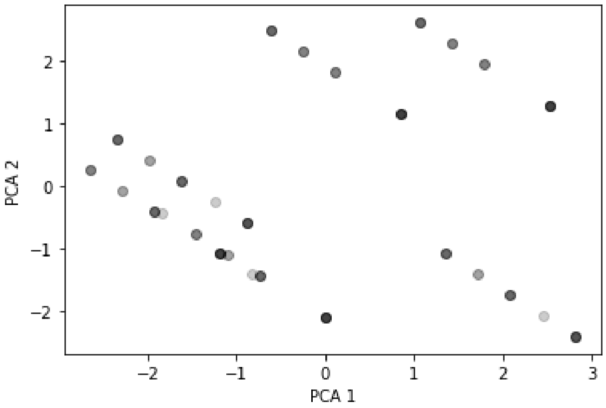
```python
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)
PCA_components = pd.DataFrame(principalComponents)
```





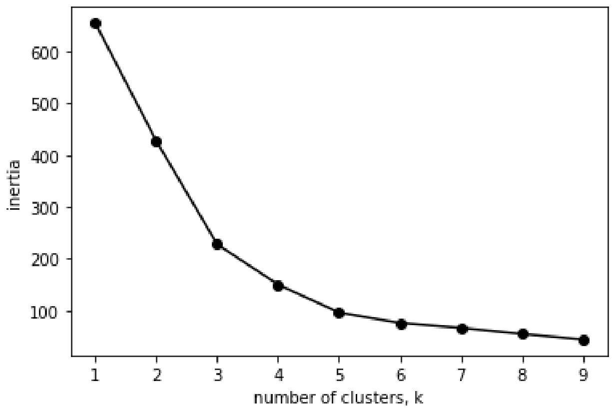
In [47]:

```
plt.scatter(PCA_components[0], PCA_components[1], alpha=0.2, color='black')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.show()
```



In [48]:

```
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
print("Hence choosen kvalue is correct, as afer 4 variance is neglected\nThis is obtained at PCA[3
] and hence we choose.")
```




```
Hence choosen kvalue is correct, as afer 4 variance is neglected
This is obtained at PCA[3] and hence we choose.
```

In [49]:

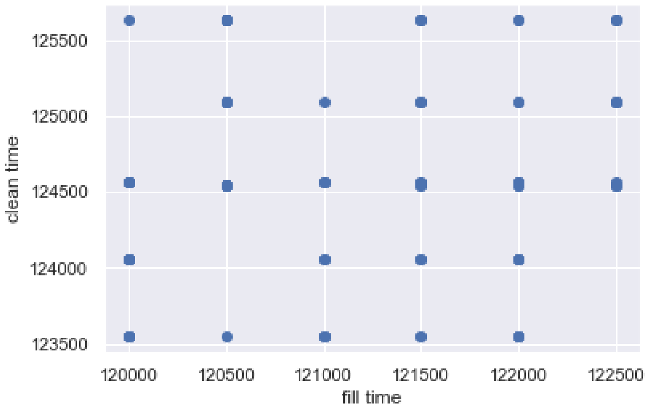
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("f.csv")
df.head()
```

Out[49]:

	bin_no	region	latitude	longitude	percentage_full	config_status	fill_time	clean_time
0	1	45	78.9	45.60	98	1	123546	120000
1	2	56	56.8	198.45	56	1	125089	120500
2	3	23	65.8	157.80	45	0	124056	121000
3	4	45	56.4	148.50	32	0	124546	121500
4	5	45	12.5	157.40	95	1	124564	122000

In [50]:

```
plt.scatter(df.iloc[:,7],df.iloc[:, 6])
plt.xlabel('fill time')
plt.ylabel('clean time')
plt.show()
```



In [51]:

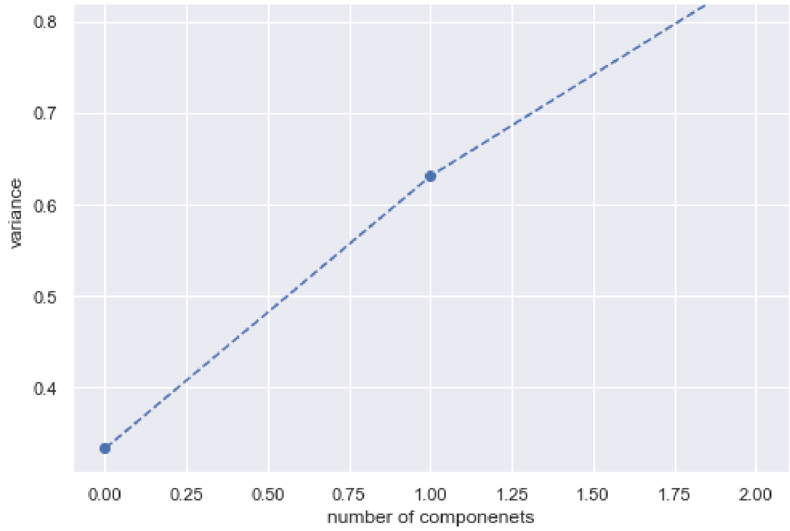
```
scaler=StandardScaler()
segmentation=scaler.fit_transform(df)
pca.fit(segmentation)
pca.explained_variance_ratio_
```

Out[51]:

```
array([0.3334838 , 0.29813883, 0.2217239 ])
```

In [52]:

```
plt.figure(figsize=(8,6))
plt.plot(pca.explained_variance_ratio_.cumsum() , marker='o', linestyle='--')
plt.xlabel('number of componenets')
plt.ylabel('variance')
plt.show()
```

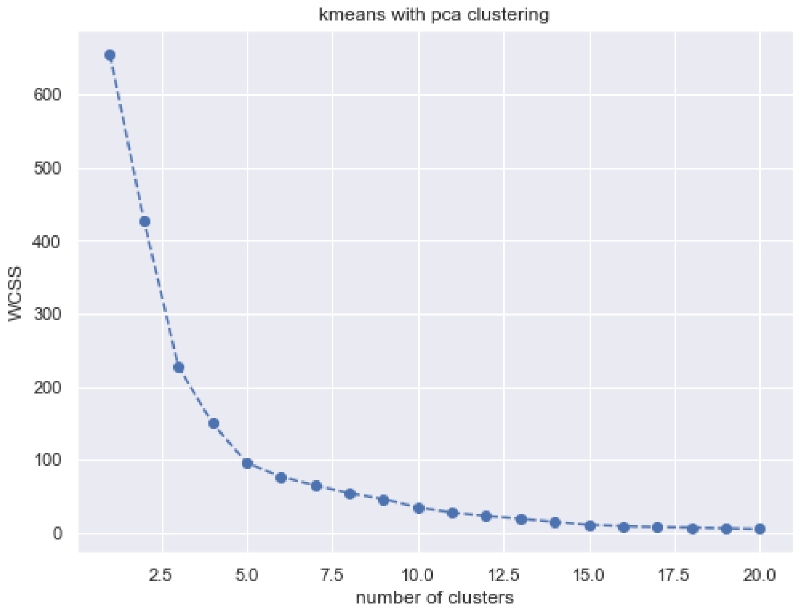


In [53]:

```
pca=PCA(n_components=3)
pca.fit(segmentation)
scores=pca.transform(segmentation)
wcss=[]
for i in range(1,21):
    kmeans_pca=KMeans(n_clusters=i,init='k-means++',random_state=42)
    kmeans_pca.fit(scores)
    wcss.append(kmeans_pca.inertia_)
```

In [54]:

```
plt.figure(figsize=(8,6))
plt.plot(range(1,21), wcss, marker='o', linestyle='--')
plt.xlabel('number of clusters')
plt.ylabel('WCSS')
plt.title('kmeans with pca clustering')
plt.show()
```



In [55]:

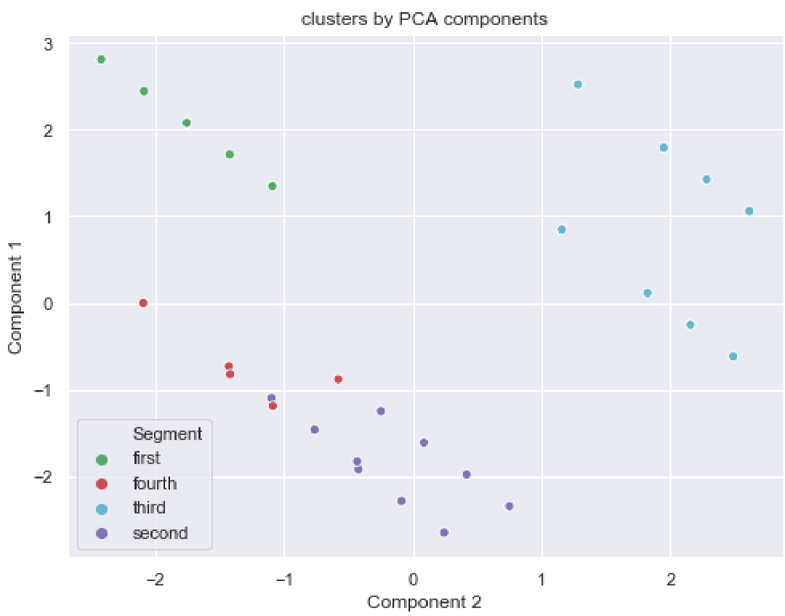
```
kmeans_pca=KMeans(n_clusters=4,init='k-means++',random_state=42)
kmeans_pca.fit(scores)
df_segm_pca_kmeans=pd.concat([df.reset_index(drop=True),pd.DataFrame(scores)],axis=1)
df_segm_pca_kmeans.columns.values[-3: ] = ['Component 1', 'Component 2', 'Component 3']
df_segm_pca_kmeans['Segment kmeans PCA']=kmeans_pca.labels_
df_segm_pca_kmeans.head()
```

Out[55]:

	bin_no	region	latitude	longitude	percentage_full	config_status	fill_time	clean_time	Component 1	Component 2	Component 3	Segn kme I
0	1	45	78.9	45.60	98	1	123546	120000	2.813597	-2.420856	0.439709	
1	2	56	56.8	198.45	56	1	125089	120500	-0.881464	-0.580190	-1.281238	
2	3	23	65.8	157.80	45	0	124056	121000	1.792680	1.948662	-0.597572	
3	4	45	56.4	148.50	32	0	124546	121500	0.116544	1.823008	-0.013177	
4	5	45	12.5	157.40	95	1	124564	122000	-1.463297	-0.764507	0.968915	

In [57]:

```
df_segm_pca_kmeans['Segment']=df_segm_pca_kmeans['Segment kmeans PCA'].map({0:'first',1:'second',2:
'third',3:'fourth'})
x_axis=df_segm_pca_kmeans['Component 2']
y_axis=df_segm_pca_kmeans['Component 1']
plt.figure(figsize=(8,6))
sns.scatterplot(x_axis,y_axis,hue=df_segm_pca_kmeans['Segment'],palette=['g','r','c','m'])
plt.title("clusters by PCA components")
plt.show()
print("Which is finally then fed to PCA 3d analysis")
```




```
Which is finally then fed to PCA 3d analysis
```

In [58]:

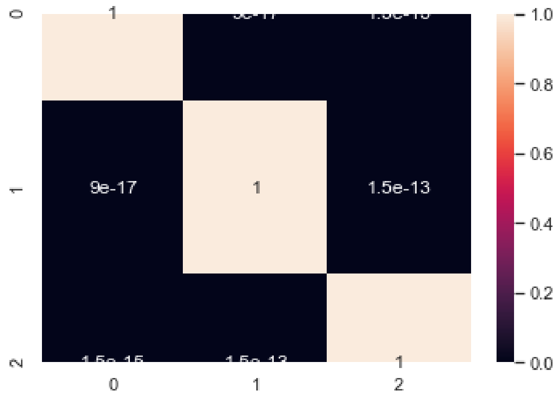
```
from sklearn.decomposition import PCA
garbage_data = pd.read_csv("f.csv")
pca = PCA(3)
pca.fit(garbage_data)
pca_data = pd.DataFrame(pca.transform(garbage_data))
print("After PCA :",pca_data.shape)
print("original data :",garbage_data.shape)
print(pca_data)
print(pca_data.head(6))
```

```
After PCA : (96, 3)
original data : (96, 8)
              0           1          2
0  -1450.406094 -607.553026  50.473751
1   -490.840744  706.994585 -36.092266
2   -339.228186 -431.187115 -47.939788
3    288.757361 -122.746933 -21.462589
4    769.260049 -260.956800   7.848135
..          ...         ...        ...
```

```
..          ...         ...        ...
91   -975.495747 -763.930849   52.154772
92    -15.930397  550.616761 -34.411245
93    135.682161 -587.564938 -46.258766
94    763.667708 -279.124756 -19.781568
95   1244.170396 -417.334623    9.529156

[96 rows x 3 columns]
             0           1          2
0 -1450.406094 -607.553026  50.473751
1  -490.840744  706.994585 -36.092266
2  -339.228186 -431.187115 -47.939788
3   288.757361 -122.746933 -21.462589
4   769.260049 -260.956800   7.848135
5  1238.578055 -435.502580 -18.100547
```

In [59]:

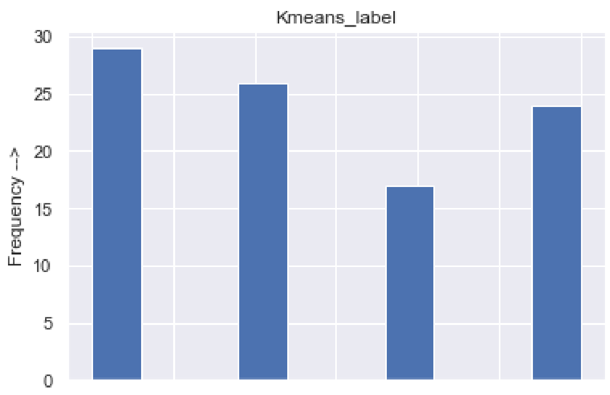
```python
import seaborn as sns

# generating correlation heatmap
sns.heatmap(pca_data.corr(), annot = True)

# posting correlation heatmap to output console
plt.show()
```

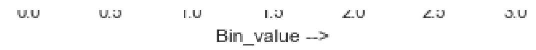


In [64]:

```python
from sklearn.cluster import KMeans
clusters = 4
kmeans = KMeans(n_clusters = clusters)
kmeans.fit(pca_data)
print(kmeans.labels_)
plt.hist(kmeans.labels_)
plt.title('Kmeans_label')
plt.xlabel('Bin_value -->')
plt.ylabel('Frequency -->')
plt.show()
```

```
[3 2 0 0 1 1 0 2 3 1 3 2 0 1 0 1 3 2 0 1 0 1 0 2 3 1 3 3 0 2 0 1 3 3 0 2 0
 1 0 3 3 1 3 2 0 0 1 1 3 2 0 0 1 1 0 2 3 1 3 2 0 1 0 1 3 2 0 1 0 1 0 2 3 1
 3 3 0 2 0 1 3 3 0 2 0 1 0 3 3 1 2 3 2 0 1 1]
```



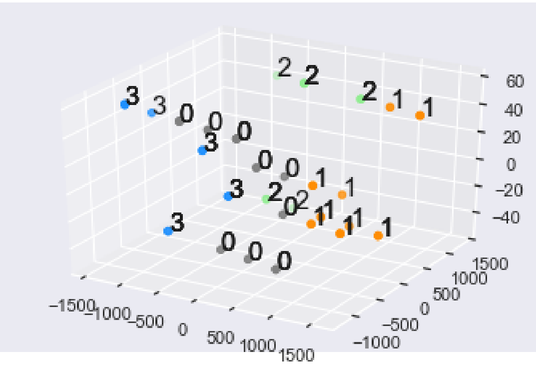


In [65]:

```
clusters=4
from matplotlib import colors as mcolors
import math

''' Generating different colors in ascending order
                                of their hsv values '''
colors = list(zip(*sorted((
                    tuple(mcolors.rgb_to_hsv(
                          mcolors.to_rgba(color)[:3])), name)
                     for name, color in dict(
                            mcolors.BASE_COLORS, **mcolors.CSS4_COLORS
                                                          ).items())))[1]


# number of steps to taken generate n(clusters) colors
skips = math.floor(len(colors[5 : -5])/clusters)
cluster_colors = colors[5 : -5 : skips]
```

In [66]:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
ax.scatter(pca_data[0], pca_data[1], pca_data[2],
           c = list(map(lambda label : cluster_colors[label],
                                            kmeans.labels_)))

str_labels = list(map(lambda label:'% s' % label, kmeans.labels_))

list(map(lambda data1, data2, data3, str_label:
        ax.text(data1, data2, data3, s = str_label, size = 16.5,
        zorder = 20, color = 'k'), pca_data[0], pca_data[1],
        pca_data[2], str_labels))
plt.show()
```

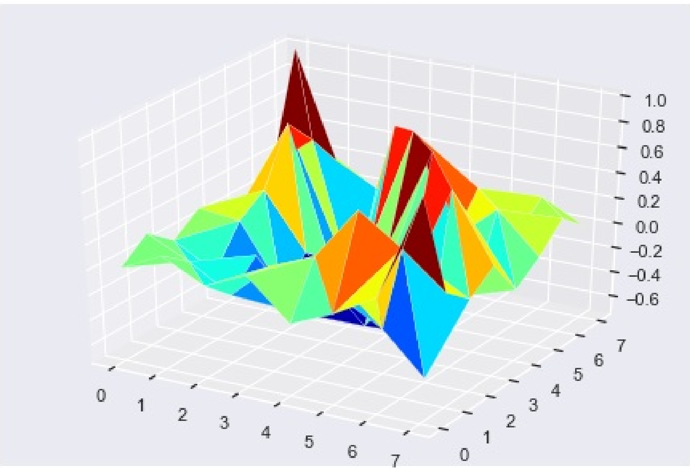


In [67]:

```
from matplotlib import cm

# generating correlation data
df = garbage_data.corr()
df.index = range(0, len(df))
df.rename(columns = dict(zip(df.columns, df.index)), inplace = True)
df = df.astype(object)

''' Generating coordinates with
corresponding correlation values '''
for i in range(0, len(df)):
    for j in range(0, len(df)):
        if i != j:
            df.iloc[i, j] = (i, j, df.iloc[i, j])
        else :
```

```
            df.iloc[i, j] = (i, j, 0)

df_list = []

# flattening dataframe values
for sub_list in df.values:
    df_list.extend(sub_list)

# converting list of tuples into trivariate dataframe
plot_df = pd.DataFrame(df_list)
print(plot_df)
fig = plt.figure()
ax = Axes3D(fig)

# plotting 3D trisurface plot
ax.plot_trisurf(plot_df[0], plot_df[1], plot_df[2],
                    cmap = cm.jet, linewidth = 0.2)

plt.show()
```

```
    0  1         2
0   0  0  0.000000
1   0  1  0.157860
2   0  2 -0.010418
3   0  3  0.136697
4   0  4 -0.171745
.. .. ..       ...
59  7  3  0.136697
60  7  4 -0.171745
61  7  5 -0.026135
62  7  6  0.168972
63  7  7  0.000000

[64 rows x 3 columns]
```



In [68]:

```
x1=plot_df[0]
x2=plot_df[1]
print("dustbin number causing anamoly : ",np.ptp(x1)-1)
print("standard deviation of x1 or x2 : ", np.std(x1), "\nThe member set of x1 or x2 differ from 2
minutes 29 seconds, which is difference of dustbin fill time from one dustbin to other bin")
print("Hence we observe reduction in standard deviation ,\nLast time before pca and additional dat
a of new dustbin it was 9 minutes 28 second \nHence new dustbin allocation has positive effect.")
```

```
dustbin number causing anamoly :  6
standard deviation of x1 or x2 :  2.29128784747792
The member set of x1 or x2 differ from 2 minutes 29 seconds, which is difference of dustbin fill t
ime from one dustbin to other bin
Hence we observe reduction in standard deviation ,
Last time before pca and additional data of new dustbin it was 9 minutes 28 second
Hence new dustbin allocation has positive effect.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: FutureWarning: Method .
ptp is deprecated and will be removed in a future version. Use numpy.ptp instead.
```

```
  return ptp(axis=axis, out=out, **kwargs)
```

In [ ]:

```

```