Clustering hospitals according to their bed capacity

In [1]:
```
import pandas as pd
import numpy as np


# Import Data, Select columns required in the model

files = "usa_hospital_beds.csv"
df = pd.read_csv(files, encoding = 'unicode_escape')

df.info ()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6626 entries, 0 to 6625
Data columns (total 18 columns):
X                                   6609 non-null float64
Y                                   6609 non-null float64
ID                                  6626 non-null int64
HOSPITAL_NAME                       6626 non-null object
HOSPITAL_TYPE                       6626 non-null object
ADDRESS                             6626 non-null object
CITY                                6626 non-null object
STATE                               6626 non-null object
ZIP_CODE                            6626 non-null int64
COUNTY_NAME                         6602 non-null object
STATE_NAME                          6602 non-null object
NUM_LICENSED_BEDS                   6403 non-null float64
NUM_STAFFED_BEDS                    6328 non-null float64
NUM_ICU_BEDS                        6626 non-null int64
ADULT_ICU_BEDS                      6626 non-null int64
BED_UTILIZATION                     5917 non-null float64
Potential_Increase_In_Bed_Capac     6626 non-null int64
AVG_VENTILATOR_USAGE                6567 non-null float64
dtypes: float64(6), int64(5), object(7)
memory usage: 931.9+ KB
```

In [2]:
```
# Change ID variable data type to into object
df['ID'] = df['ID'].astype(object)

# Change ZIP_CODE data type to string

df['ZIP_CODE'] = df['ZIP_CODE'].astype(object)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6626 entries, 0 to 6625
Data columns (total 18 columns):
X                                   6609 non-null float64
Y                                   6609 non-null float64
ID                                  6626 non-null object
HOSPITAL_NAME                       6626 non-null object
HOSPITAL_TYPE                       6626 non-null object
ADDRESS                             6626 non-null object
CITY                                6626 non-null object
STATE                               6626 non-null object
ZIP_CODE                            6626 non-null object
COUNTY_NAME                         6602 non-null object
STATE_NAME                          6602 non-null object
NUM_LICENSED_BEDS                   6403 non-null float64
```

```
NUM_STAFFED_BEDS                    6328 non-null float64
NUM_ICU_BEDS                        6626 non-null int64
ADULT_ICU_BEDS                      6626 non-null int64
BED_UTILIZATION                     5917 non-null float64
Potential_Increase_In_Bed_Capac     6626 non-null int64
AVG_VENTILATOR_USAGE                6567 non-null float64
dtypes: float64(6), int64(3), object(9)
memory usage: 931.9+ KB
```

In [3]:
```
# get more information from 'ID'
print(df['ID'].describe())
```

```
count     6626
unique    6626
top       6626
freq         1
Name: ID, dtype: int64
```

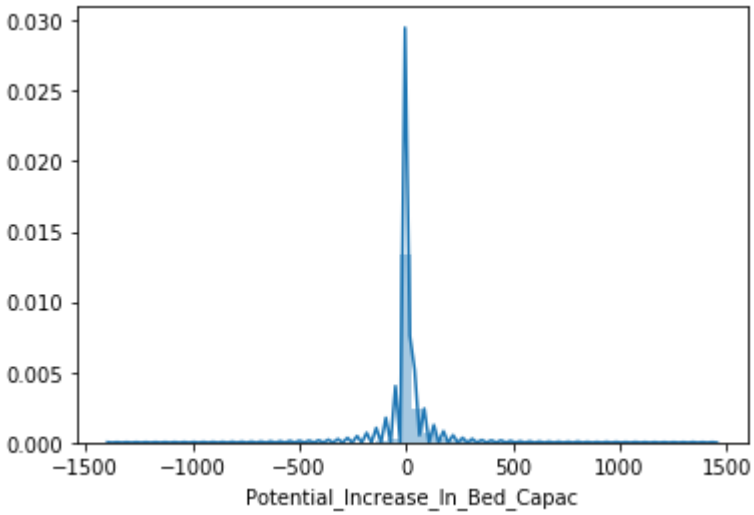
In [4]:
```
# have a look at the variables
print(df['Potential_Increase_In_Bed_Capac'].describe())
print("\n")
print(df['NUM_STAFFED_BEDS'].describe())
print("\n")
print(df['NUM_LICENSED_BEDS'].describe())
print("\n")
print(df['BED_UTILIZATION'].describe())
print("\n")
print(df['ADULT_ICU_BEDS'].describe())
print("\n")
print(df['AVG_VENTILATOR_USAGE'].describe())
```

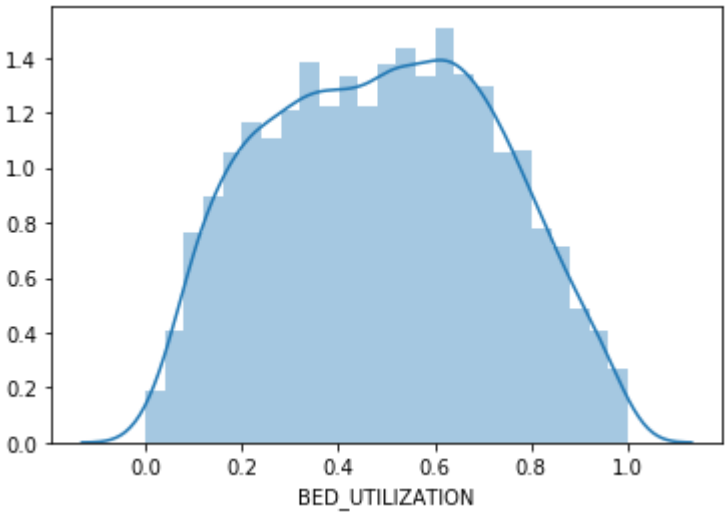
```
count    6626.000000
mean       21.176728
std        74.012221
min     -1389.000000
25%         0.000000
50%         0.000000
75%        23.000000
max      1446.000000
Name: Potential_Increase_In_Bed_Capac, dtype: float64


count    6328.000000
mean      127.668458
std       160.116186
min         1.000000
25%        25.000000
50%        65.000000
75%       166.000000
max      2753.000000
Name: NUM_STAFFED_BEDS, dtype: float64


count    6403.000000
mean      148.087303
std       182.684947
min         1.000000
25%        25.000000
50%        75.000000
75%       197.000000
max      2059.000000
Name: NUM_LICENSED_BEDS, dtype: float64


count    5917.000000
mean        0.491855
std         0.235353
```

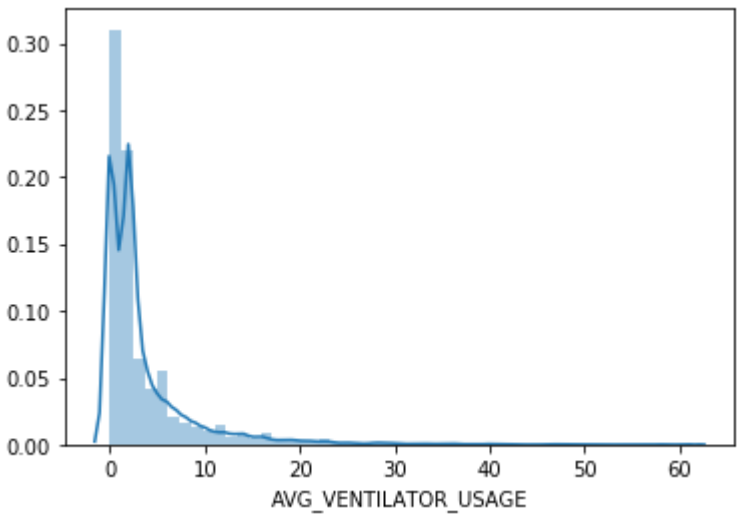
```
min          0.000721
25%          0.300923
50%          0.497300
75%          0.675295
max          1.000000
Name: BED_UTILIZATION, dtype: float64


count    6626.000000
mean       14.019771
std        22.457818
min         0.000000
25%         3.000000
50%         6.000000
75%        16.000000
max       310.000000
Name: ADULT_ICU_BEDS, dtype: float64


count    6567.000000
mean        3.550023
std         5.464410
min         0.000000
25%         0.000000
50%         2.000000
75%         4.000000
max        61.000000
Name: AVG_VENTILATOR_USAGE, dtype: float64
```

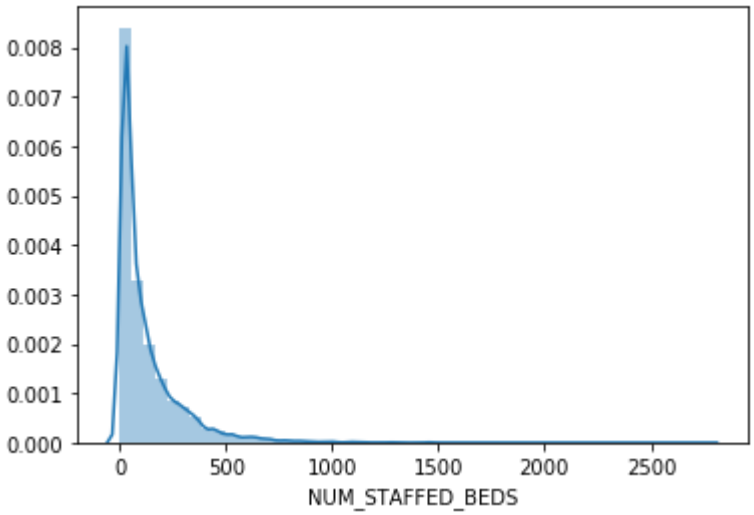
In [5]:

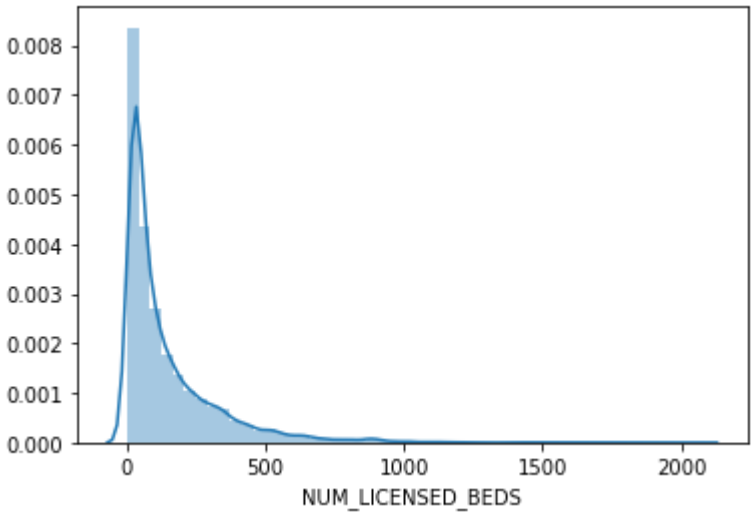
```
import seaborn as sns
import matplotlib.pyplot as plt
#Distribution of Potential_Increase_In_Bed_Capac
Potential_Increase_In_Bed_Capac_dist = sns.distplot(df['Potential_Increase_In_
plt.show ()
#Distribution of BED_UTILIZATION
BED_UTILIZATION_dist = sns.distplot (df['BED_UTILIZATION'].dropna())
plt.show ()
# Distribution of AVG_VENTILATOR_USAGE
average_dist= sns.distplot(df['AVG_VENTILATOR_USAGE'].dropna())
plt.show()
# Distribution of NUM_STAFFED_BEDS
average_dist= sns.distplot(df['NUM_STAFFED_BEDS'].dropna())
plt.show()
# Distribution of NUM_LICENSED_BEDS
average_dist= sns.distplot(df['NUM_LICENSED_BEDS'].dropna())
plt.show()
# Distribution of ADULT_ICU_BEDS
average_dist= sns.distplot(df['ADULT_ICU_BEDS'].dropna())
plt.show()
```

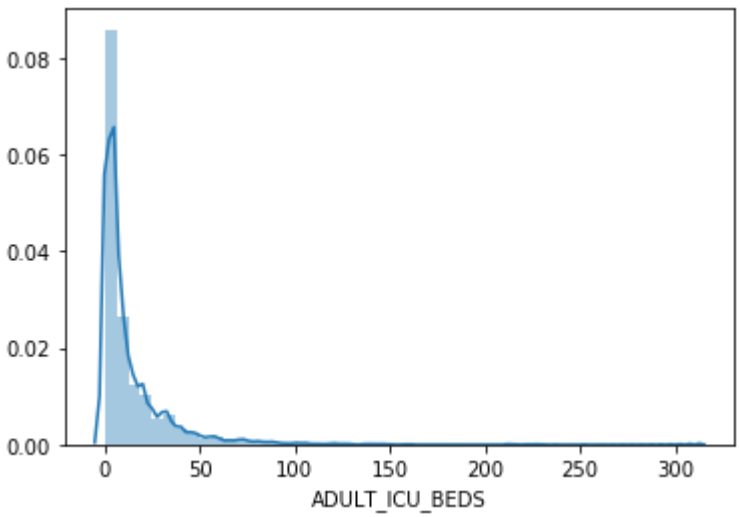

0.030
0.025
0.020
0.015
0.010
0.005
0.000
−1500
−1000
−500
0
500
1000
1500
Potential_Increase_In_Bed_Capac


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
BED_UTILIZATION


0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
60
AVG_VENTILATOR_USAGE







In [6]:

```
# Drop rows with missing values in NUM_STAFFED_BED column

df1 = df.dropna(how='any', subset=['NUM_STAFFED_BEDS'])

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6328 entries, 0 to 6610
Data columns (total 18 columns):
X                                   6314 non-null float64
Y                                   6314 non-null float64
ID                                  6328 non-null object
HOSPITAL_NAME                       6328 non-null object
HOSPITAL_TYPE                       6328 non-null object
```

```
ADDRESS                              6328 non-null object
CITY                                 6328 non-null object
STATE                                6328 non-null object
ZIP_CODE                             6328 non-null object
COUNTY_NAME                          6307 non-null object
STATE_NAME                           6307 non-null object
NUM_LICENSED_BEDS                    6328 non-null float64
NUM_STAFFED_BEDS                     6328 non-null float64
NUM_ICU_BEDS                         6328 non-null int64
ADULT_ICU_BEDS                       6328 non-null int64
BED_UTILIZATION                      5916 non-null float64
Potential_Increase_In_Bed_Capac      6328 non-null int64
AVG_VENTILATOR_USAGE                 6322 non-null float64
dtypes: float64(6), int64(3), object(9)
memory usage: 939.3+ KB
```

In [7]:

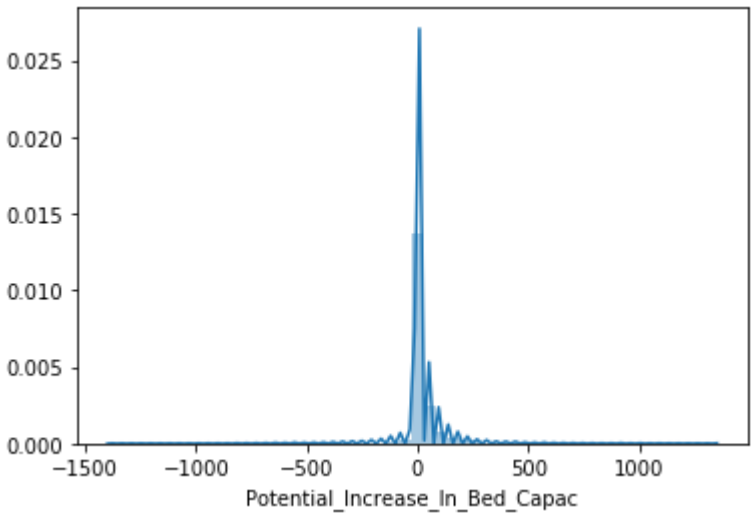
```
# Drop rows with missing values in AVG_VENTILATOR_USAGE and BED_UTILIZATION
df1 = df1.dropna(how='any', subset=['AVG_VENTILATOR_USAGE'])
df1 = df1.dropna(how='any', subset=['BED_UTILIZATION'])

# Convert the data types of NUM_LICENSED_BEDS and NUM_STAFFED_BEDS into int64
df1['NUM_LICENSED_BEDS'] = df1['NUM_LICENSED_BEDS'].astype('int64')
df1['NUM_STAFFED_BEDS'] = df1['NUM_STAFFED_BEDS'].astype('int64')

df1.info()
```

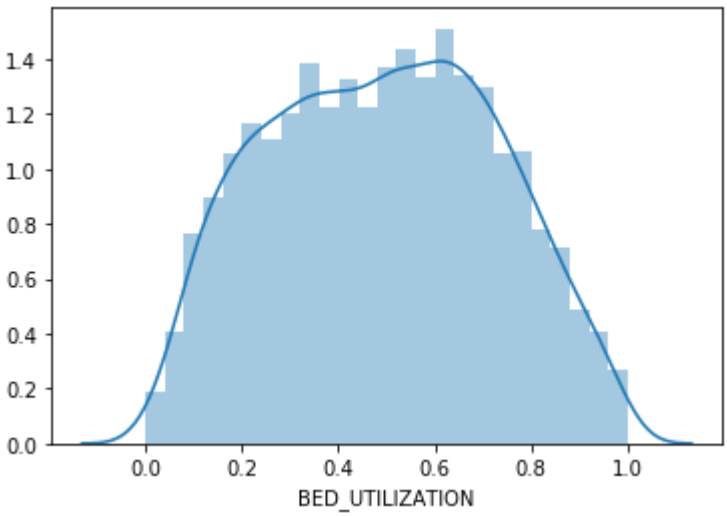
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5911 entries, 9 to 6544
Data columns (total 18 columns):
X                                    5899 non-null float64
Y                                    5899 non-null float64
ID                                   5911 non-null object
HOSPITAL_NAME                        5911 non-null object
HOSPITAL_TYPE                        5911 non-null object
ADDRESS                              5911 non-null object
CITY                                 5911 non-null object
STATE                                5911 non-null object
ZIP_CODE                             5911 non-null object
COUNTY_NAME                          5895 non-null object
STATE_NAME                           5895 non-null object
NUM_LICENSED_BEDS                    5911 non-null int64
NUM_STAFFED_BEDS                     5911 non-null int64
NUM_ICU_BEDS                         5911 non-null int64
ADULT_ICU_BEDS                       5911 non-null int64
BED_UTILIZATION                      5911 non-null float64
Potential_Increase_In_Bed_Capac      5911 non-null int64
AVG_VENTILATOR_USAGE                 5911 non-null float64
dtypes: float64(4), int64(5), object(9)
memory usage: 877.4+ KB
```

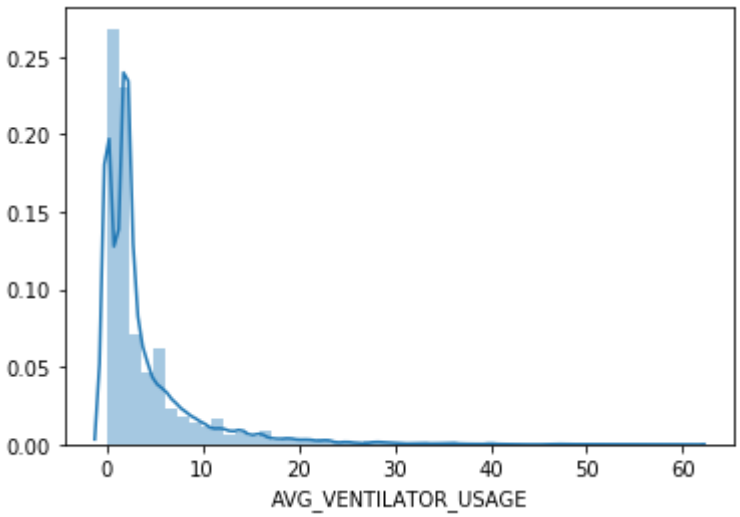
In [8]:

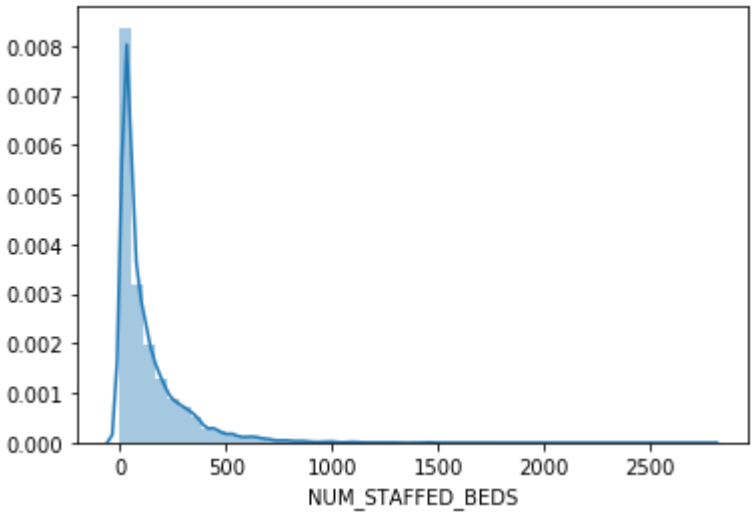
```
#Distribution of Potential_Increase_In_Bed_Capac
Potential_Increase_In_Bed_Capac_dist = sns.distplot(df1['Potential_Increase_I
plt.show ()
#Distribution of BED_UTILIZATION
BED_UTILIZATION_dist = sns.distplot (df1['BED_UTILIZATION'])
plt.show ()
# Distribution of AVG_VENTILATOR_USAGE
average_dist= sns.distplot(df1['AVG_VENTILATOR_USAGE'])
plt.show()
# Distribution of AVG_VENTILATOR_USAGE
average_dist= sns.distplot(df1['NUM_STAFFED_BEDS'])
plt.show()
# Distribution of NUM_LICENSED_BEDS
average_dist= sns.distplot(df1['NUM_LICENSED_BEDS'].dropna())
plt.show()
# Distribution of AVG_VENTILATOR_USAGE
```

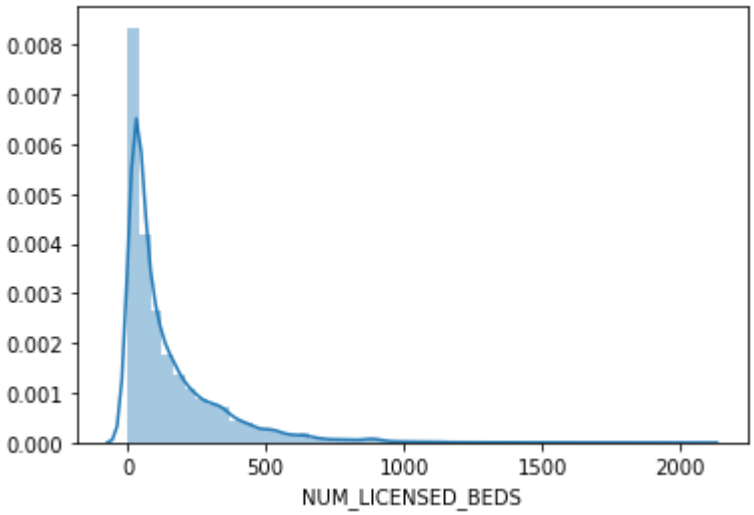
```
average_dist= sns.distplot(df1['ADULT_ICU_BEDS'])
plt.show()
```

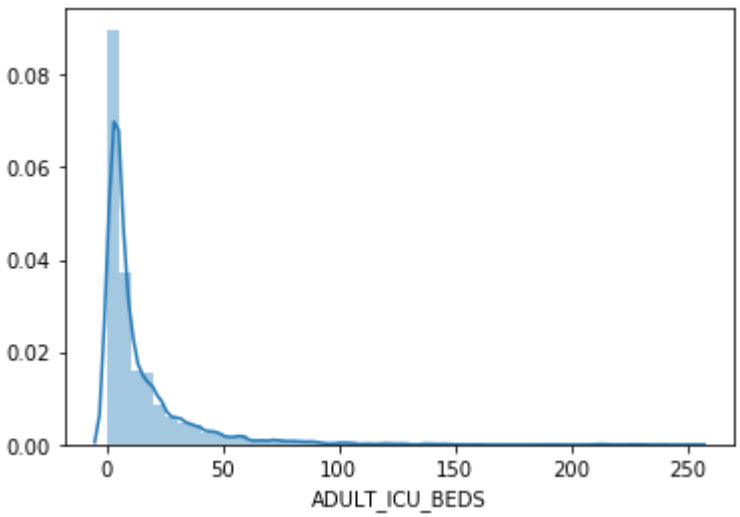












In [9]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5911 entries, 9 to 6544
Data columns (total 18 columns):
X                                  5899 non-null float64
Y                                  5899 non-null float64
ID                                 5911 non-null object
HOSPITAL_NAME                      5911 non-null object
HOSPITAL_TYPE                      5911 non-null object
ADDRESS                            5911 non-null object
CITY                               5911 non-null object
STATE                              5911 non-null object
ZIP_CODE                           5911 non-null object
```

```
COUNTY_NAME                          5895 non-null object
STATE_NAME                           5895 non-null object
NUM_LICENSED_BEDS                    5911 non-null int64
NUM_STAFFED_BEDS                     5911 non-null int64
NUM_ICU_BEDS                         5911 non-null int64
ADULT_ICU_BEDS                       5911 non-null int64
BED_UTILIZATION                      5911 non-null float64
Potential_Increase_In_Bed_Capac      5911 non-null int64
AVG_VENTILATOR_USAGE                 5911 non-null float64
dtypes: float64(4), int64(5), object(9)
memory usage: 877.4+ KB
```

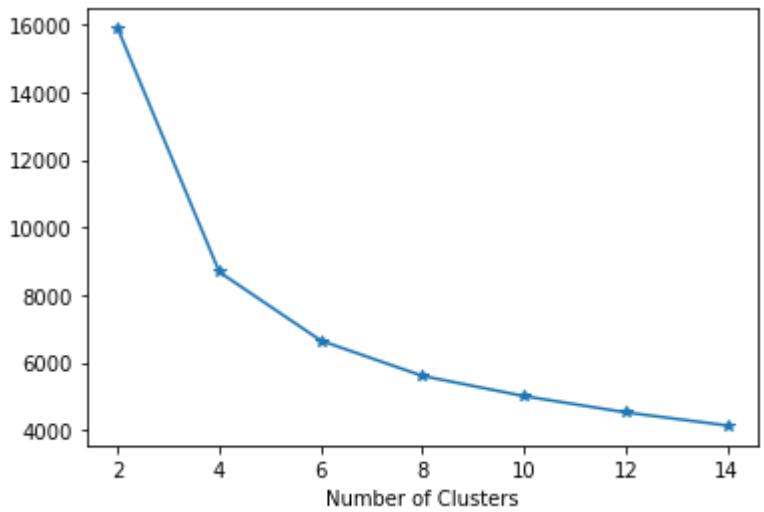
find out the optimal number of clusters

In [10]:

```
from sklearn.preprocessing import StandardScaler
# take 5 variables and drop the rest
data1 = df1[['NUM_STAFFED_BEDS', 'ADULT_ICU_BEDS','NUM_LICENSED_BEDS', 'BED_U
# convert data1 to matrix
X = data1.to_numpy()
# scaling
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [11]:

```
# For model 1 with data1
from sklearn.cluster import KMeans
rs = 42
# list to save the clusters and cost
clusters = []
inertia_vals = []

# this whole process should take a while
for k in range(2, 15, 2):
    # train clustering with the specified K
    model1 = KMeans(n_clusters=k, random_state=rs, n_jobs=10)
    model1.fit(X)

    # append model to cluster list
    clusters.append(model1)
    inertia_vals.append(model1.inertia_)

# plot the inertia vs K values
plt.plot(range(2,15,2), inertia_vals, marker='*')
plt.xlabel('Number of Clusters')
plt.show()
```



In [12]:
```
# Use silhouette score to see which cluster is optimal

from sklearn.metrics import silhouette_score

print(clusters[1])
print("Silhouette score for k=4", silhouette_score(X, clusters[1].predict(X)))
print("\n")
print(clusters[2])
print("Silhouette score for k=6", silhouette_score(X, clusters[2].predict(X)))
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=10, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
Silhouette score for k=4 0.4075185703084071


KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=6, n_init=10, n_jobs=10, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
Silhouette score for k=6 0.33978267202126294
```

Build a clustering model with normalised variables

In [13]:
```
# set the model based on optimal cluster
model1 = KMeans(n_clusters=4, random_state=rs)
model1.fit(X)
# sum of intra-cluster distances
print("Sum of intra-cluster distance:", model1.inertia_)
print("Centroid locations:")
for centroid in model1.cluster_centers_:
    print(centroid)
```

```
Sum of intra-cluster distance: 8680.47258286676
Centroid locations:
[-0.50692378 -0.382565   -0.51041723 -0.92840797 -0.40894495]
[3.26174958 3.16809162 3.10435427 1.04412083 3.34301599]
[-0.28347414 -0.35051005 -0.31073626  0.83096678 -0.34093951]
[0.96067444 0.807663   1.04895569 0.50369725 0.81497047]
```

In [14]:
```
model1 = KMeans(n_clusters=4, random_state=rs).fit(X)
y = model1.predict(X)
data1['Cluster_ID'] = y
# how many records are in each cluster
print("Cluster membership")
print(data1['Cluster_ID'].value_counts())
# pairplot the cluster distribution.
cluster_g = sns.pairplot(data1, hue='Cluster_ID',diag_kind='hist')
plt.show()
```

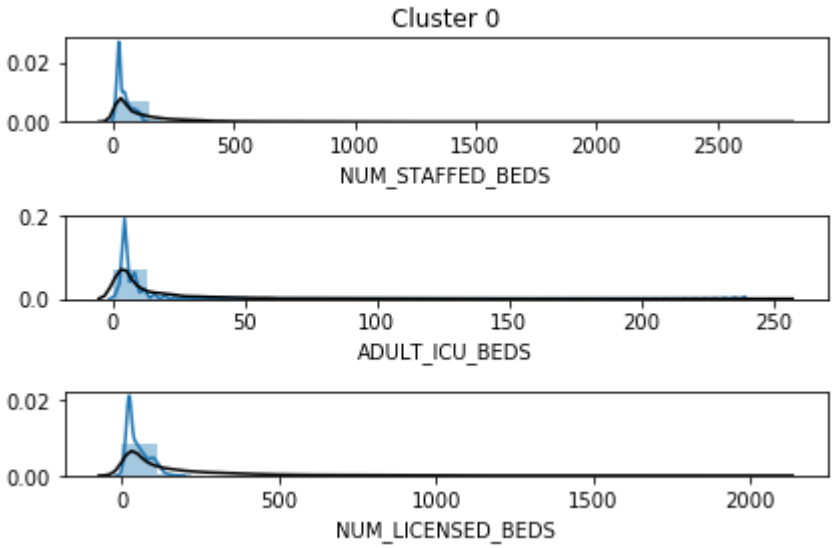
```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports un
til
Cluster membership
0    2611
2    1930
```

```
3    1129
1     241
Name: Cluster_ID, dtype: int64
```

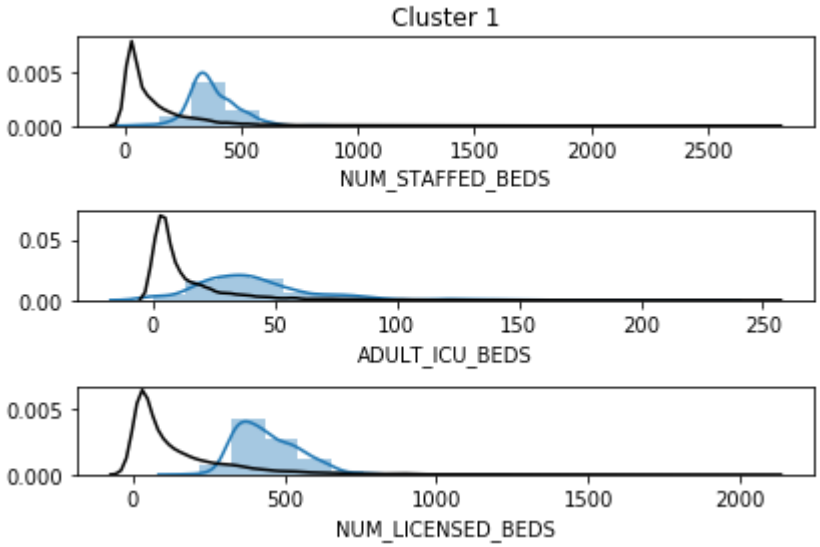
In [63]:

```
# prepare the column and bin size. Increase bin size to be more specific, but
cols_1 = ['NUM_STAFFED_BEDS', 'ADULT_ICU_BEDS','NUM_LICENSED_BEDS']
n_bins = 20

# inspecting cluster 0,1,2,3
clusters_to_inspect = [0,1,2,3]

for cluster in clusters_to_inspect:
    # inspecting cluster 0
    print("Distribution for cluster {}".format(cluster))

    # create subplots
    fig, ax = plt.subplots(nrows=3)
    ax[0].set_title("Cluster {}".format(cluster))


    for j, col in enumerate(cols_1):
        bins = np.linspace(min(data1[col]), max(data1[col]), 20)
        # plot distribution of the cluster using histogram
        sns.distplot(data1[data1['Cluster_ID'] == cluster][col], bins=bins, a
        # plot the normal distribution with a black line
        sns.distplot(data1[col], bins=bins, ax=ax[j], hist=False, color="k")

    plt.tight_layout()
    plt.show()
```

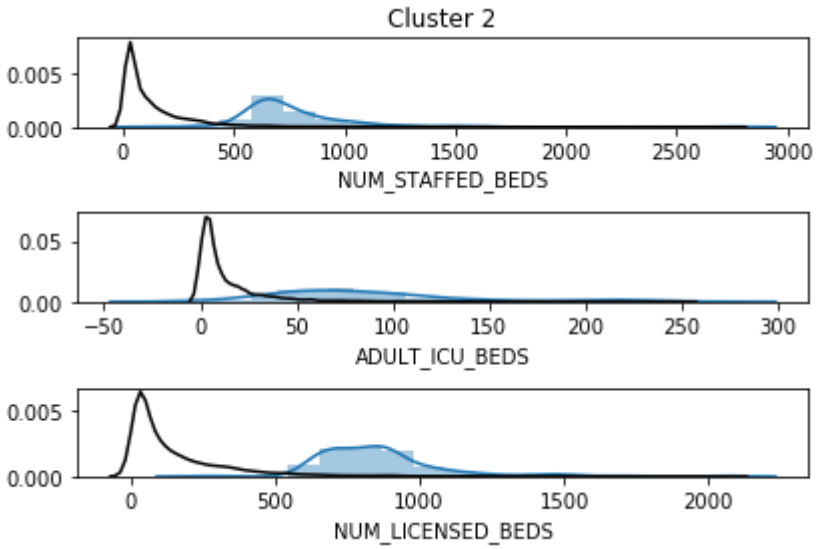
```
Distribution for cluster 0
```



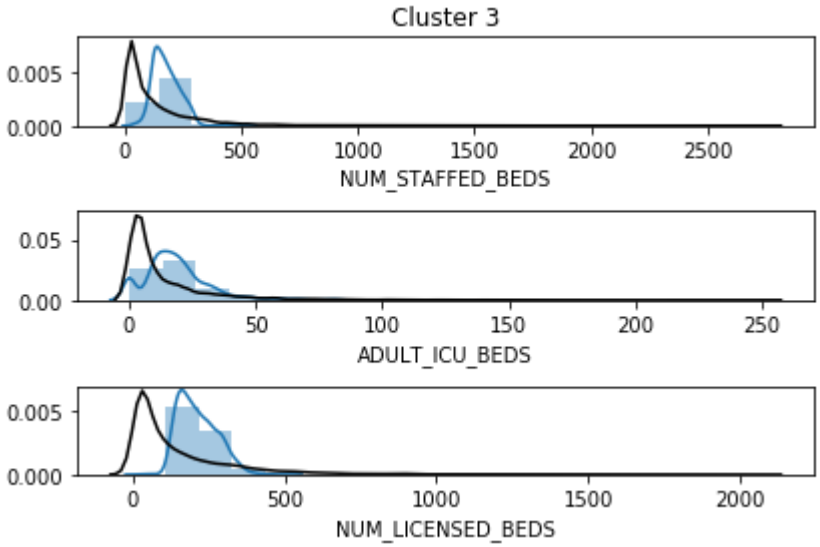

```
Distribution for cluster 1
```




```
Distribution for cluster 2
```



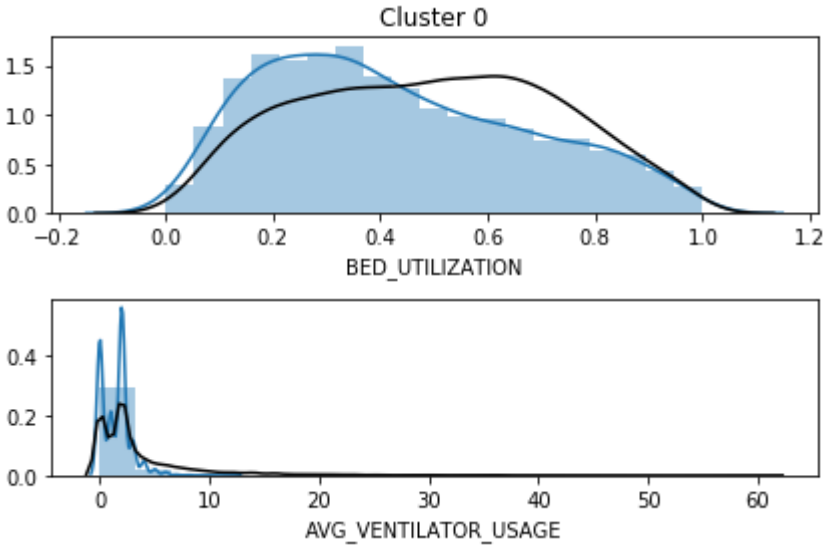

```
Distribution for cluster 3
```



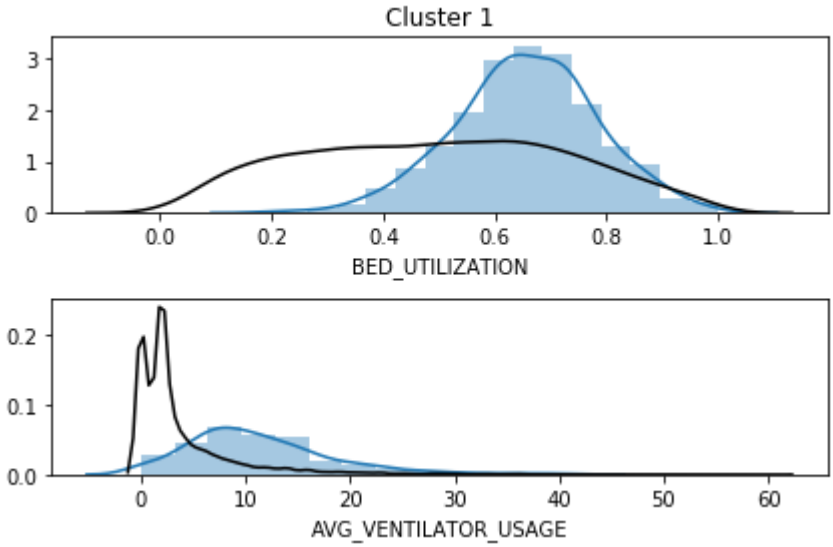
In [65]:

```
cols_2 = ['BED_UTILIZATION','AVG_VENTILATOR_USAGE']
n_bins = 20

# inspecting cluster 0 and 1
clusters_to_inspect = [0,1,2,3]

for cluster in clusters_to_inspect:
    # inspecting cluster 0
    print("Distribution for cluster {}".format(cluster))

    # create subplots
    fig, ax = plt.subplots(nrows=2)
    ax[0].set_title("Cluster {}".format(cluster))


    for j, col in enumerate(cols_2):
        bins = np.linspace(min(data1[col]), max(data1[col]), 20)
        # plot distribution of the cluster using histogram
        sns.distplot(data1[data1['Cluster_ID'] == cluster][col], bins=bins, a
        # plot the normal distribution with a black line
        sns.distplot(data1[col], bins=bins, ax=ax[j], hist=False, color="k")

    plt.tight_layout()
    plt.show()
```

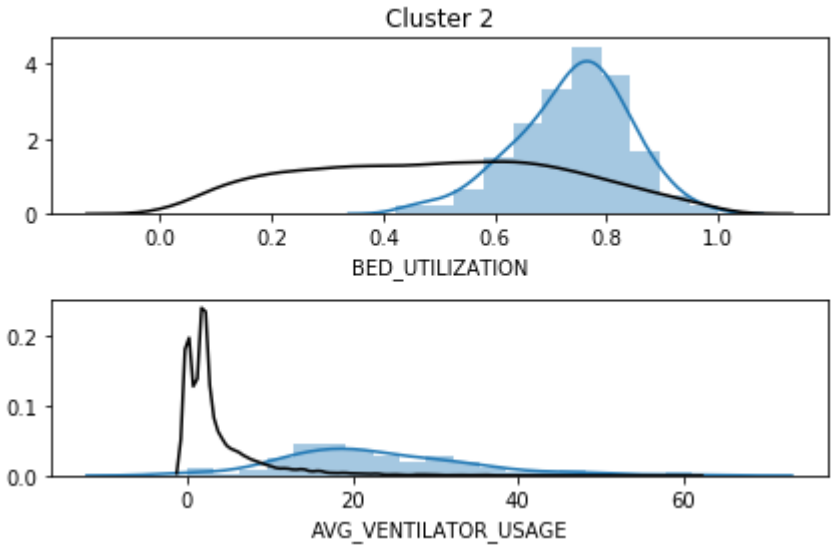
```
Distribution for cluster 0
```



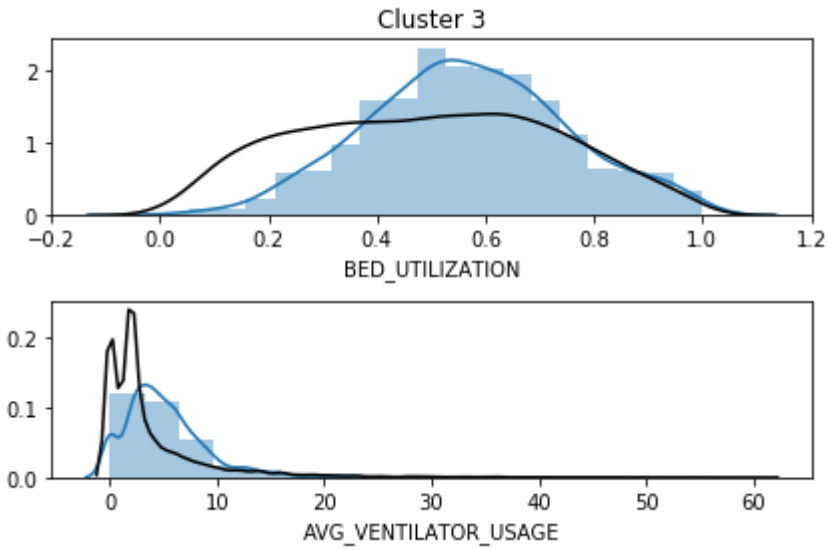

```
Distribution for cluster 1
```




```
Distribution for cluster 2
```



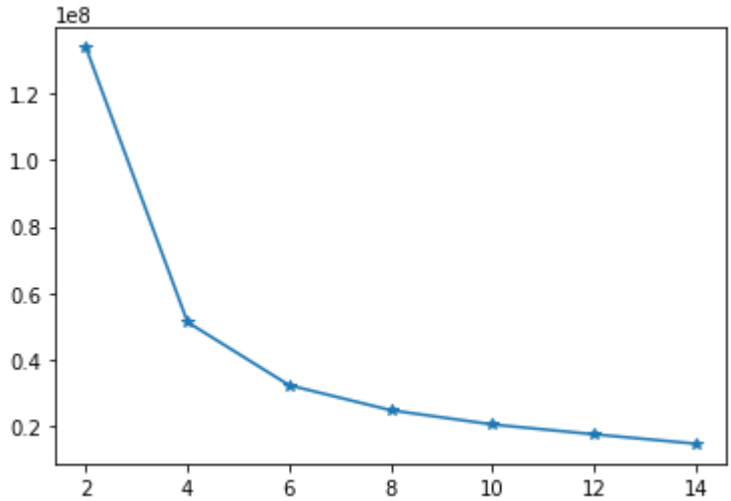

```
Distribution for cluster 3
```



Build a clustering model without variables scaling

In [16]:
```
# without scaling
X2 = data1.to_numpy()
```

```
rs = 42
# list to save the clusters and cost
clusters = []
inertia_vals = []

# this whole process should take a while
for k in range(2, 15, 2):
    # train clustering with the specified K
    model = KMeans(n_clusters=k, random_state=rs, n_jobs=10)
    model.fit(X2)

    # append model to cluster list
    clusters.append(model)
    inertia_vals.append(model.inertia_)

# plot the inertia vs K values
plt.plot(range(2,15,2), inertia_vals, marker='*')
plt.show()
```



In [17]:

```
# Use silhouette score to see which cluster is optimal

from sklearn.metrics import silhouette_score

print(clusters[1])
print("Silhouette score for k=4", silhouette_score(X2, clusters[1].predict(X2
print("\n")
print(clusters[2])
print("Silhouette score for k=6", silhouette_score(X2, clusters[2].predict(X2
print("\n")
print(clusters[3])
print("Silhouette score for k=8", silhouette_score(X2, clusters[3].predict(X2
```

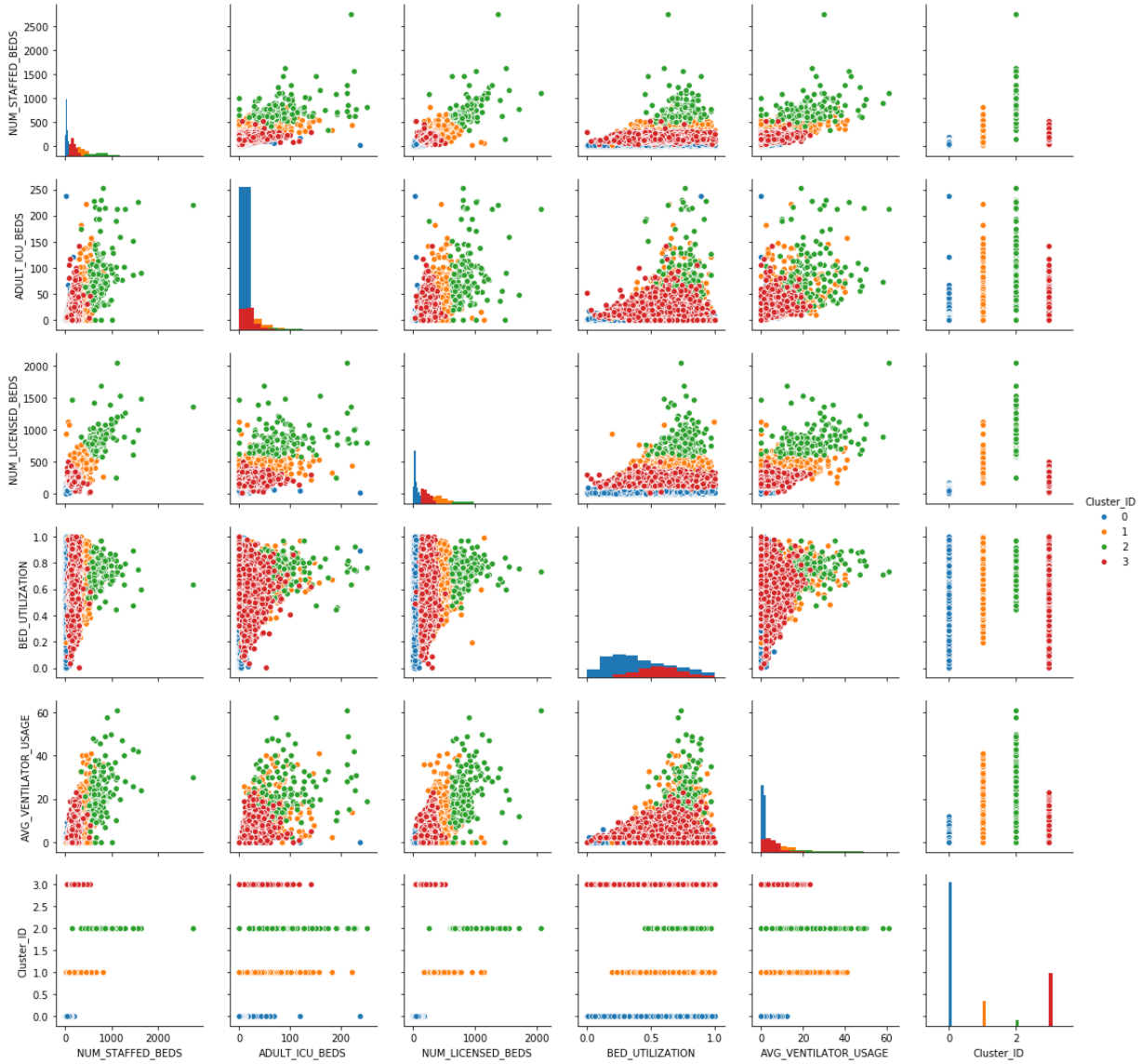
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=10, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
Silhouette score for k=4 0.6188727494017624


KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=6, n_init=10, n_jobs=10, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
Silhouette score for k=6 0.5785951655570027


KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=8, n_init=10, n_jobs=10, precompute_distances='auto',
       random_state=42, tol=0.0001, verbose=0)
Silhouette score for k=8 0.5321749843269643
```

In [18]:
```
# Build model without scaling
model2 = KMeans(n_clusters=4, random_state=rs)
model2.fit(X2)
# sum of intra-cluster distances
print("Sum of intra-cluster distance:", model2.inertia_)
print("Centroid locations:")
for centroid in model2.cluster_centers_:
    print(centroid)
```

```
Sum of intra-cluster distance: 51198466.11399235
Centroid locations:
[44.14730125  5.15102366 50.4408402   0.42776634  1.47248072  0.75059825]
[376.67032967  42.16169545 446.83673469   0.65503203  11.33124019
   2.67660911]
[7.66328859e+02 8.71073826e+01 8.63409396e+02 7.38847309e-01
 2.28456376e+01 1.10738255e+00]
[179.87390029  19.19721408 214.92741935   0.56584491   5.01906158
   2.04912023]
```

In [19]:
```
model2 = KMeans(n_clusters=4, random_state=rs).fit(X2)
y = model2.predict(X2)
data1['Cluster_ID'] = y
print("Cluster membership")
print(data1['Cluster_ID'].value_counts())
# pairplot the cluster distribution.
cluster_g = sns.pairplot(data1, hue='Cluster_ID',diag_kind='hist')
plt.show()
```

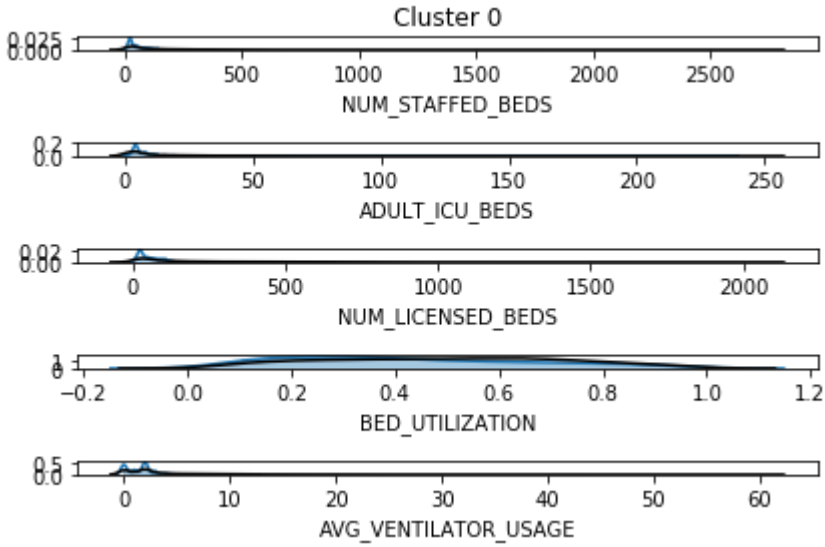
```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports un
til
Cluster membership
0    3756
3    1368
1     638
2     149
Name: Cluster_ID, dtype: int64
```



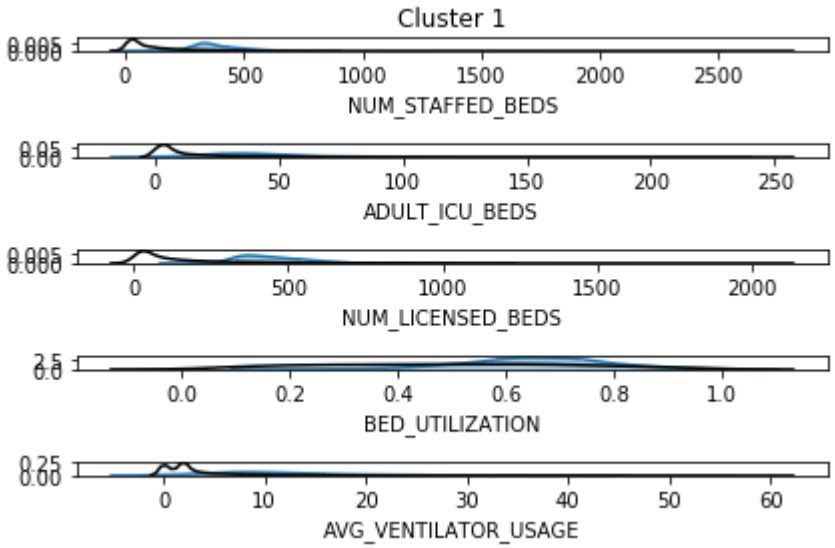
In [20]:

```
# prepare the column and bin size. Increase bin size to be more specific, but
cols = ['NUM_STAFFED_BEDS', 'ADULT_ICU_BEDS','NUM_LICENSED_BEDS', 'BED_UTILIZ
n_bins = 20

# inspecting cluster 0 and 1
clusters_to_inspect = [0,1]

for cluster in clusters_to_inspect:
    # inspecting cluster 0
    print("Distribution for cluster {}".format(cluster))

    # create subplots
    fig, ax = plt.subplots(nrows=5)
    ax[0].set_title("Cluster {}".format(cluster))

    for j, col in enumerate(cols):
        # create the bins
        bins = np.linspace(min(data1[col]), max(data1[col]), 20)
        # plot distribution of the cluster using histogram
        sns.distplot(data1[data1['Cluster_ID'] == cluster][col], bins=bins, a
        # plot the normal distribution with a black line
        sns.distplot(data1[col], bins=bins, ax=ax[j], hist=False, color="k")

    plt.tight_layout()
    plt.show()
```

```
Distribution for cluster 0
```




```
Distribution for cluster 1
```



The second clustering model

In [52]:
```
# selecet he variables which will use
# HOSPITAL_TYPE,'NUM_STAFFED_BEDS', 'ADULT_ICU_BEDS','NUM_LICENSED_BEDS', 'BE
df2 = df1.copy()
print(df2['HOSPITAL_TYPE'].unique())
df2.info()
```

```
['Short Term Acute Care Hospital' 'Critical Access Hospital'
 'Childrens Hospital' 'Long Term Acute Care Hospital'
 'Rehabilitation Hospital' 'Psychiatric Hospital'
 'Religious Non-Medical Health Care Institution']
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5911 entries, 9 to 6544
Data columns (total 18 columns):
X                                  5899 non-null float64
Y                                  5899 non-null float64
ID                                 5911 non-null object
HOSPITAL_NAME                      5911 non-null object
HOSPITAL_TYPE                      5911 non-null object
ADDRESS                            5911 non-null object
CITY                               5911 non-null object
STATE                              5911 non-null object
ZIP_CODE                           5911 non-null object
COUNTY_NAME                        5895 non-null object
STATE_NAME                         5895 non-null object
```

```
NUM_LICENSED_BEDS                   5911 non-null int64
NUM_STAFFED_BEDS                    5911 non-null int64
NUM_ICU_BEDS                        5911 non-null int64
ADULT_ICU_BEDS                      5911 non-null int64
BED_UTILIZATION                     5911 non-null float64
Potential_Increase_In_Bed_Capac     5911 non-null int64
AVG_VENTILATOR_USAGE                5911 non-null float64
dtypes: float64(4), int64(5), object(9)
memory usage: 877.4+ KB
```

In [53]:
```
# map HOSPITAL_TYPE - categorical value to numeric values
from sklearn.preprocessing import StandardScaler

HOSPITAL_map = {'Short Term Acute Care Hospital':1, 'Critical Access Hospital
df2['HOSPITAL_TYPE'] = df2['HOSPITAL_TYPE'].map(HOSPITAL_map)

data2 = df2[['HOSPITAL_TYPE','NUM_STAFFED_BEDS','ADULT_ICU_BEDS','NUM_LICENSE

# convert df to matrix
X1 = data2.to_numpy()

# scaling
scaler = StandardScaler()
X1= scaler.fit_transform(X1)
```

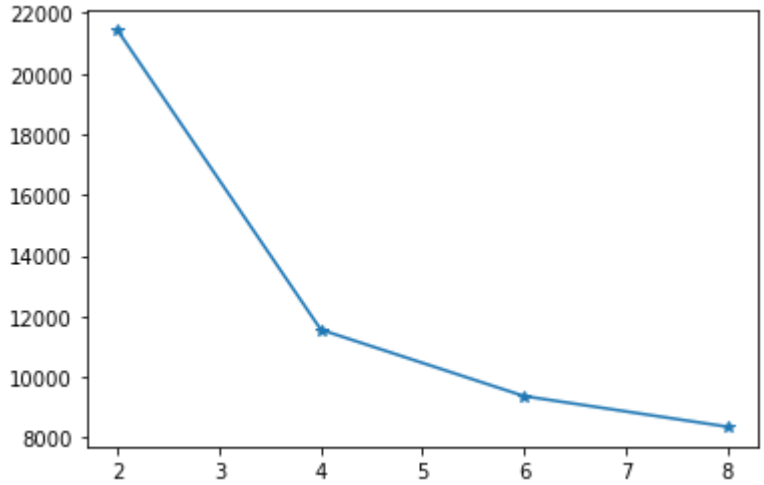
In [23]:
```
pip install kmodes
```

```
Requirement already satisfied: kmodes in /opt/conda/lib/python3.7/site-package
s (0.10.2)
Requirement already satisfied: scipy>=0.13.3 in /opt/conda/lib/python3.7/site-
packages (from kmodes) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-p
ackages (from kmodes) (0.14.1)
Requirement already satisfied: numpy>=1.10.4 in /opt/conda/lib/python3.7/site-
packages (from kmodes) (1.18.1)
Requirement already satisfied: scikit-learn>=0.19.0 in /opt/conda/lib/python3.
7/site-packages (from kmodes) (0.22.1)
Note: you may need to restart the kernel to use updated packages.
```

In [24]:
```
from kmodes.kmodes import KModes
from kmodes.kprototypes import KPrototypes
```

In [54]:
```
# list to save the clusters and cost
clusters = []
cost_vals = []

for k in range(2, 10, 2):
    model = KPrototypes(n_clusters=k, random_state=rs, n_jobs=20)
    model.fit_predict(X1, categorical=[1])

    # append model to cluster list
    clusters.append(model)
    cost_vals.append(model.cost_)
```

In [55]:
```
# plot K values

plt.plot(range(2,10,2), cost_vals, marker = '*')
plt.show()
```



In [56]:
```
X_num = [[row[0], row[2]] for row in X] # Variables of X with numeric datatyp
X_cat = [[row[1]] for row in X] # variables of X with categorical datatype
```

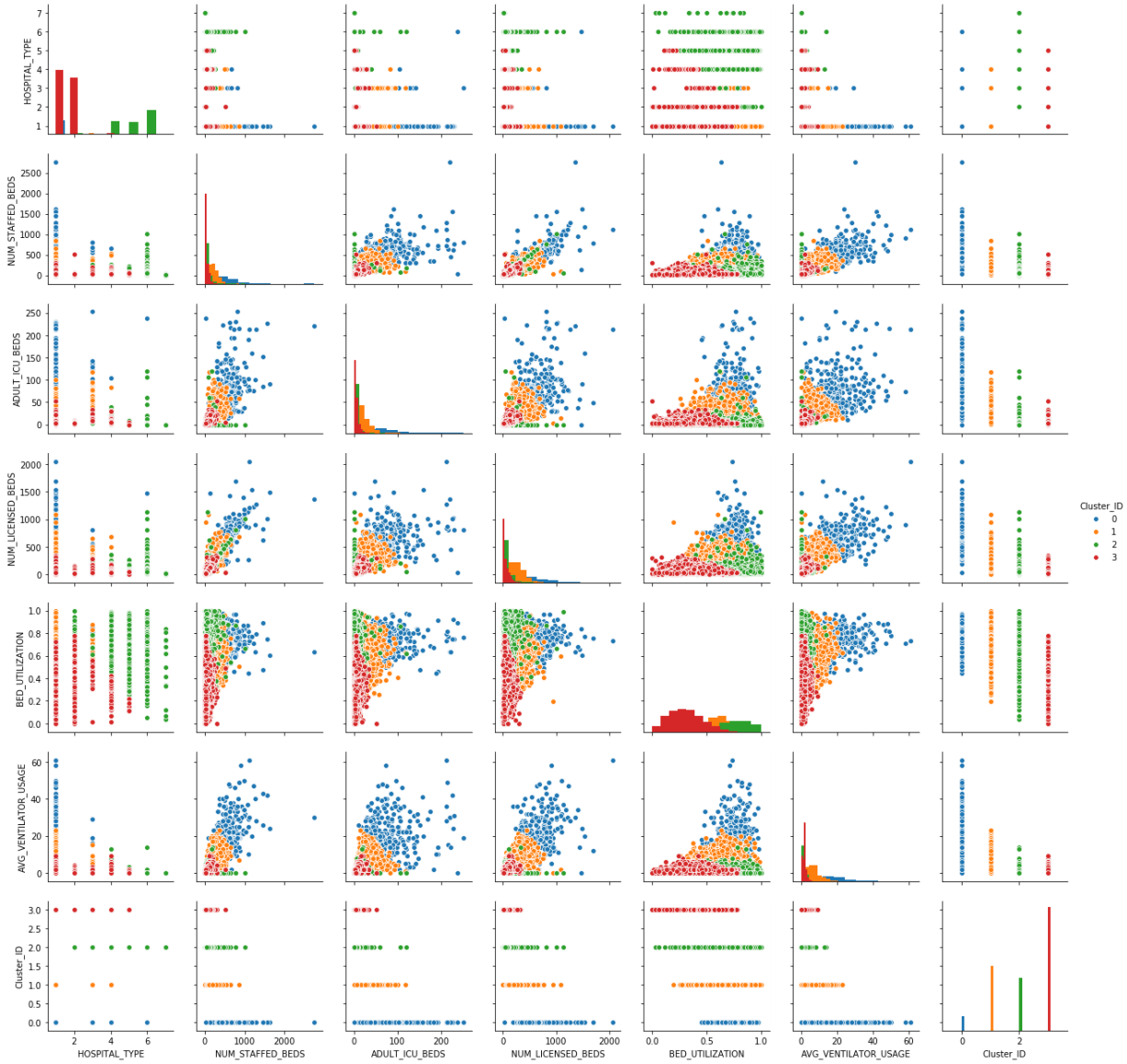
In [57]:
```
from sklearn.metrics import silhouette_score

# Calculate the average Silhouette Score k = 4
model = clusters[1]
silScoreNums = silhouette_score(X_num, model.fit_predict(X1,categorical=[1]),
silScoreCats = silhouette_score(X_cat, model.fit_predict(X1,categorical=[1]),
silScore = (silScoreNums + silScoreCats) / 2
print("The avg silhouette score for k=4: " + str(silScore))

#Calculate the average Silhouette Score k = 6
model = clusters[2]
silScoreNums = silhouette_score(X_num, model.fit_predict(X1,categorical=[1]),
silScoreCats = silhouette_score(X_cat, model.fit_predict(X1,categorical=[1]),
silScore = (silScoreNums + silScoreCats) / 2
print("The avg silhouette score for k=6: " + str(silScore))

#Calculate the average Silhouette Score k = 8
model = clusters[3]
silScoreNums = silhouette_score(X_num, model.fit_predict(X1,categorical=[1]),
silScoreCats = silhouette_score(X_cat, model.fit_predict(X1,categorical=[1]),
silScore = (silScoreNums + silScoreCats) / 2
print("The avg silhouette score for k=8: " + str(silScore))
```

```
The avg silhouette score for k=4: 0.13969625717049194
The avg silhouette score for k=6: 0.09202128113086472
The avg silhouette score for k=8: 0.020983715475672047
```

In [58]:
```
import seaborn as sns
import matplotlib.pyplot as plt
model = clusters[1]
y = model.fit_predict(X1, categorical=[1])
data2['Cluster_ID'] = y

print("Cluster membership")
print(data2['Cluster_ID'].value_counts())
cluster_g = sns.pairplot(data2, hue='Cluster_ID', diag_kind='hist')
plt.show()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:5: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
Cluster membership
3    2861
1    1499
2    1214
0     337
Name: Cluster_ID, dtype: int64
```

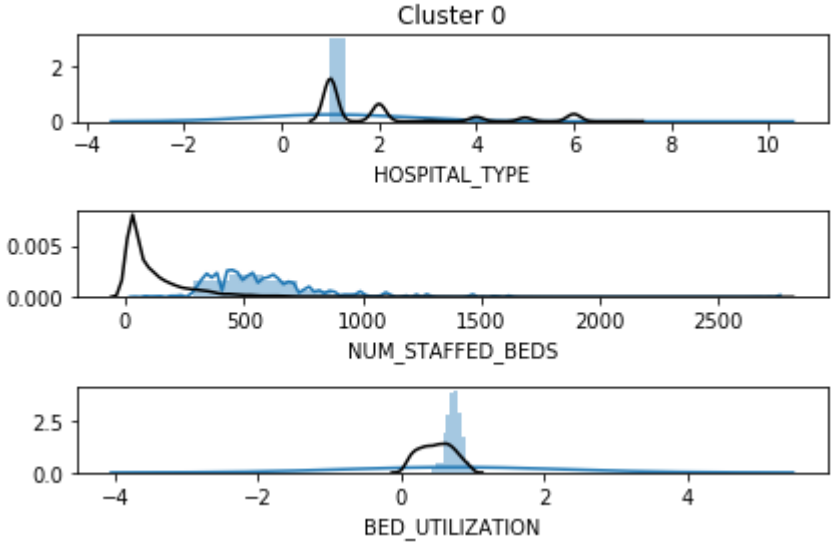


In [70]:

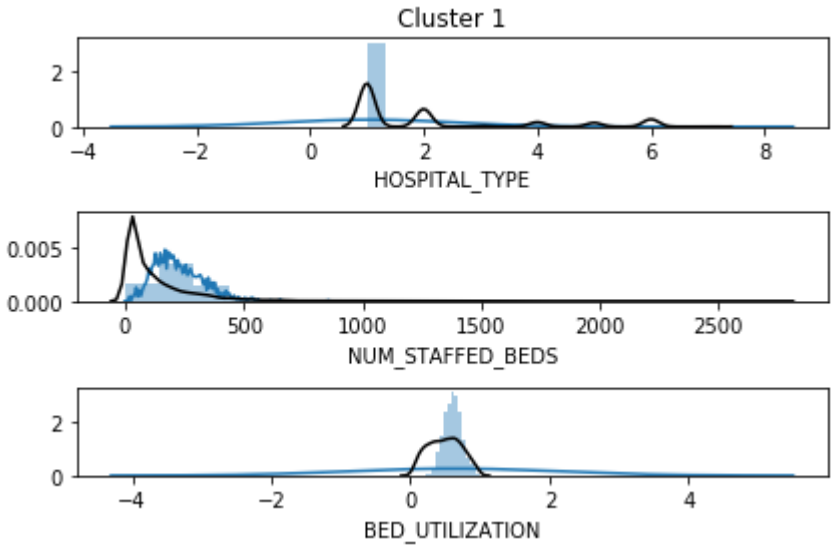
```
# plot distribution

cols_1 = ['HOSPITAL_TYPE','NUM_STAFFED_BEDS','BED_UTILIZATION']
n_bins = 20


clusters_to_inspect = [0,1,2,3]

for cluster in clusters_to_inspect:
    print("Distribution for cluster {}".format(cluster))
    fig, ax = plt.subplots(nrows=3)
    ax[0].set_title("Cluster {}".format(cluster))

    for j, col in enumerate(cols_1):
        bins = np.linspace(min(data2[col]), max(data2[col]), 20)
        sns.distplot(data2[data2['Cluster_ID'] == cluster][col], bins=bins, a
        sns.distplot(data2[col], bins=bins, ax=ax[j], hist=False, color="k")
```

```
    plt.tight_layout()
    plt.show()
```

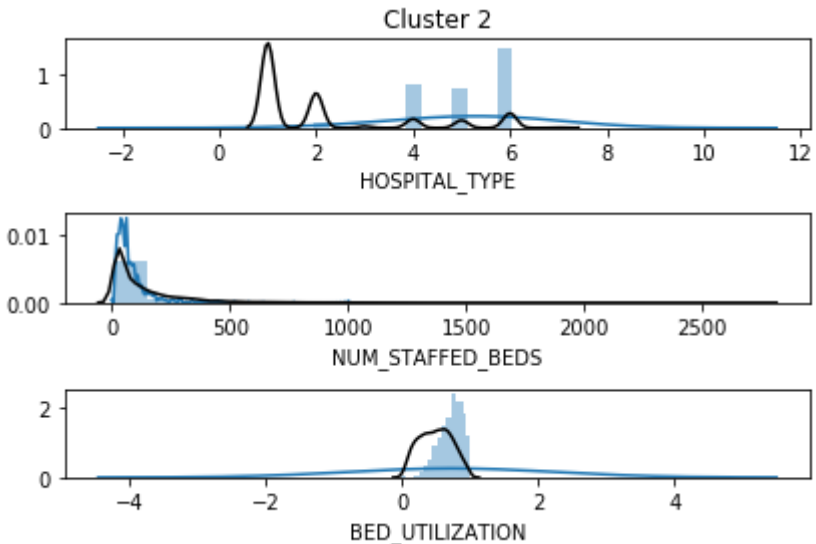
```
Distribution for cluster 0
```



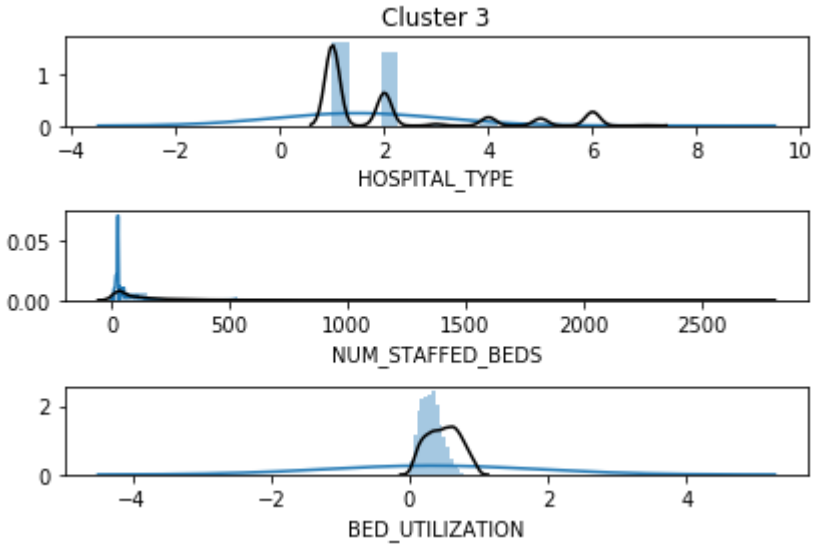

```
Distribution for cluster 1
```




```
Distribution for cluster 2
```



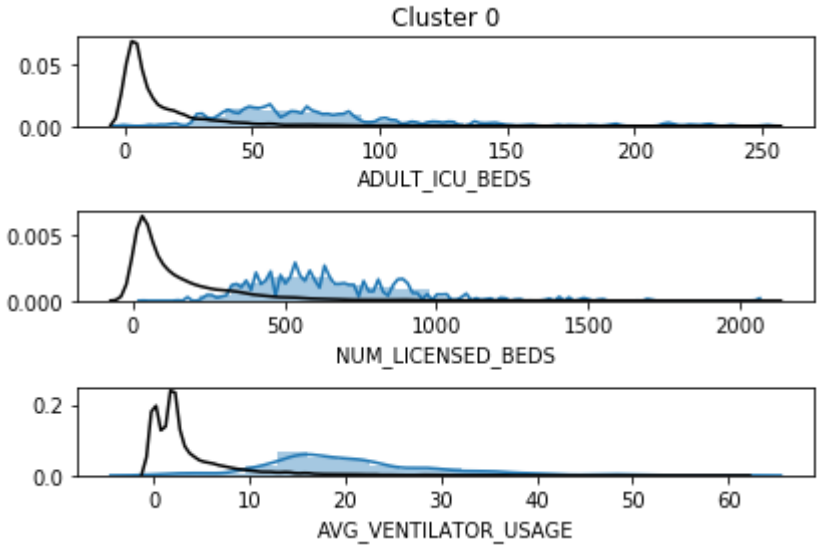

```
Distribution for cluster 3
```



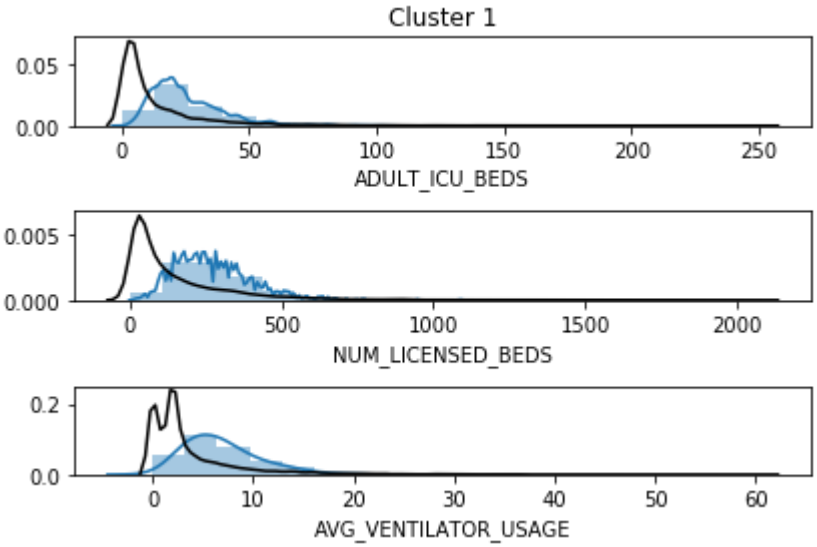
In [69]:

```
cols_2 = ['ADULT_ICU_BEDS','NUM_LICENSED_BEDS','AVG_VENTILATOR_USAGE']

n_bins = 20


clusters_to_inspect = [0,1,2,3]

for cluster in clusters_to_inspect:
    print("Distribution for cluster {}".format(cluster))
    fig, ax = plt.subplots(nrows=3)
    ax[0].set_title("Cluster {}".format(cluster))

    for j, col in enumerate(cols_2):
        bins = np.linspace(min(data2[col]), max(data2[col]), 20)
        sns.distplot(data2[data2['Cluster_ID'] == cluster][col], bins=bins, a
        sns.distplot(data2[col], bins=bins, ax=ax[j], hist=False, color="k")

    plt.tight_layout()
    plt.show()
```

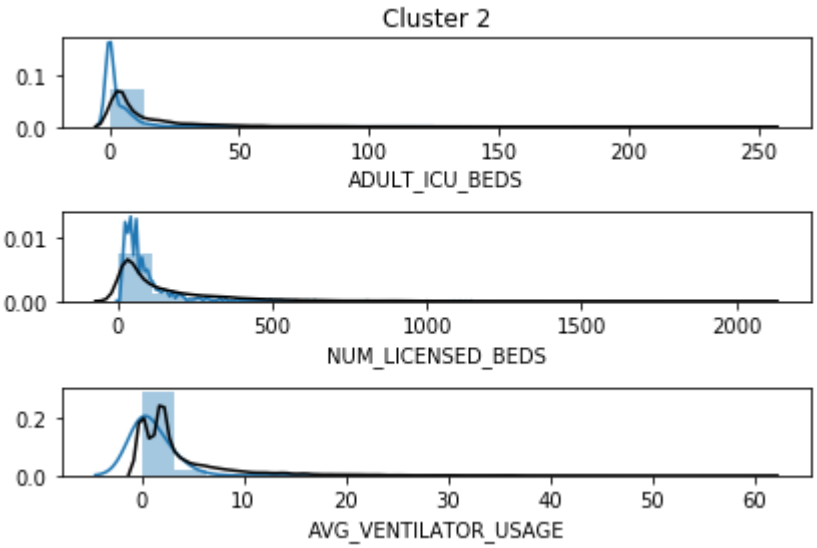
```
Distribution for cluster 0
```



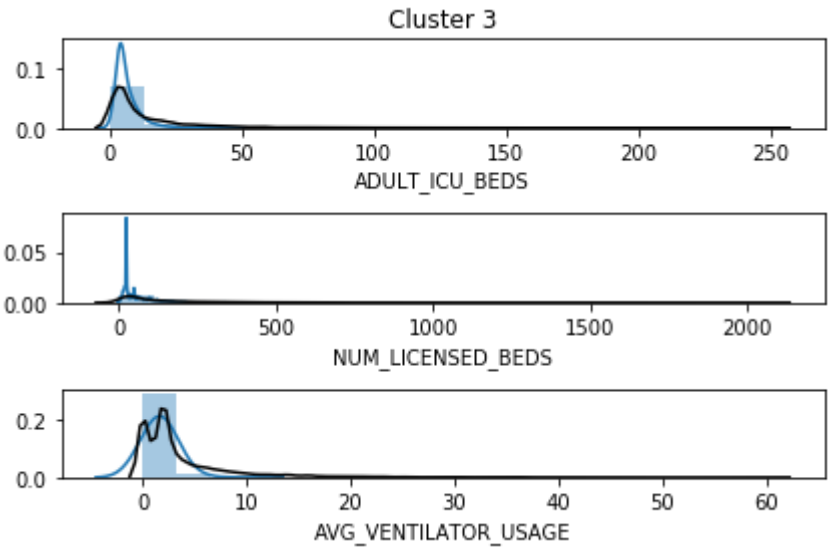

```
Distribution for cluster 1
```




```
Distribution for cluster 2
```




```
Distribution for cluster 3
```



In [ ]: