Cubic

November 21, 2020

0.0.1 Question 1: Cubic

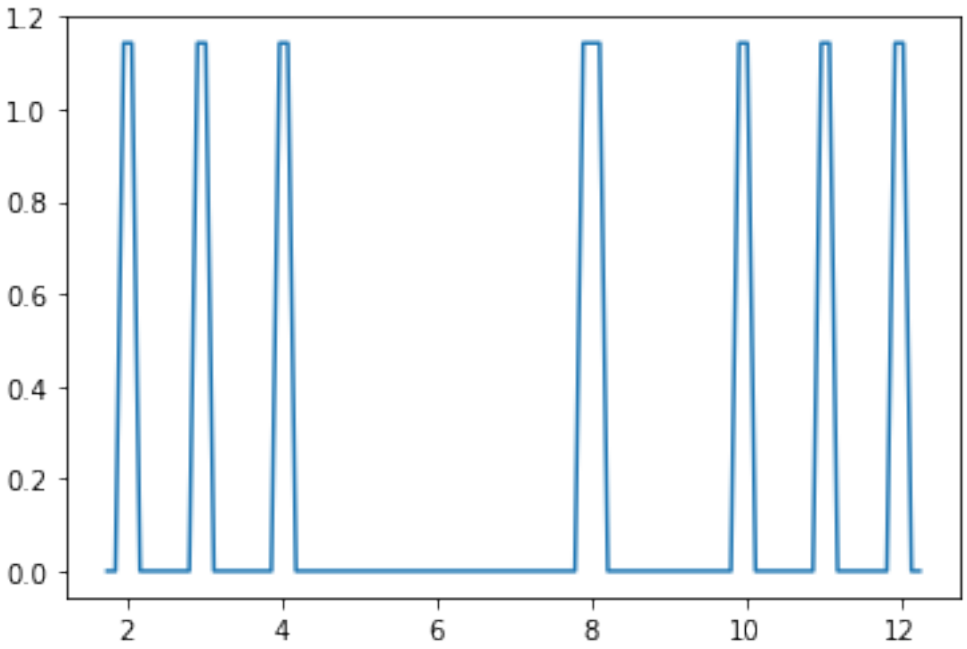
Plot the density using Parzen window (both cubic and Gaussian) for the data points (2, 3, 4, 8, 10, 11, 12). Vary h and observe the behavior of the computed density.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
#h=1.5
xi = [2,3,4,8,10,11,12]
```

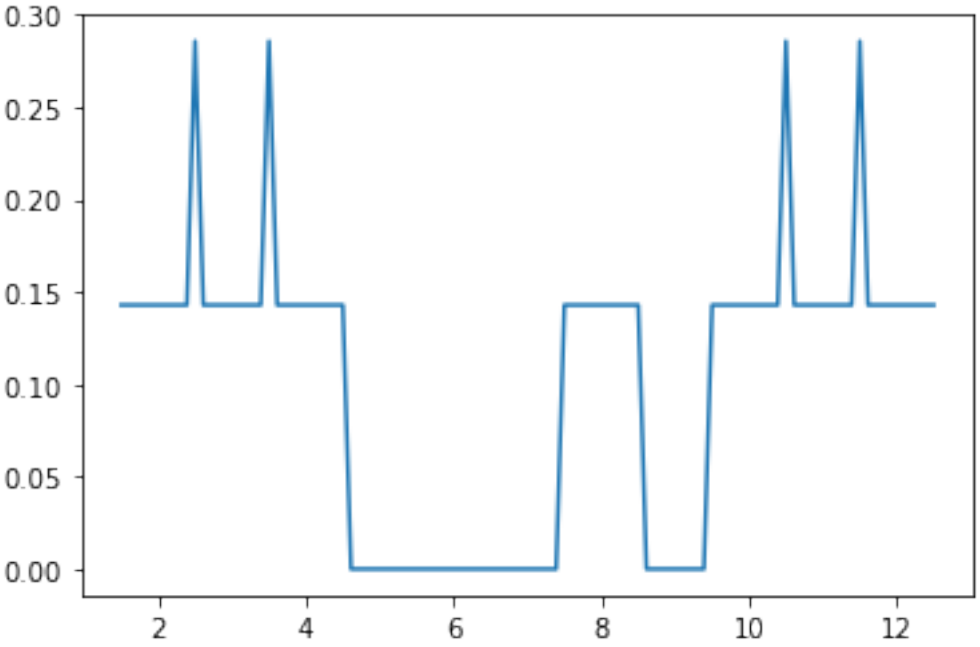
[3]:
```
def window_func(x,xi,h):
    out=abs((x-xi)/h)
    if(out<=h/2):
        output = 1
    else: output = 0
    return output
```

[4]:
```
#Package into a function

def cubic(h,data):
    x1 = np.linspace(min(data)-h/2,max(data)+h/2,100)
    final1=[]
    res1=0
    #h=1
    for i in x1:
        i=round(i,2)
        for itr in data:
            res1=res1+((1/h**3)*window_func(i,itr,h))
        result=(1/len(data))*res1
        res1=0
        final1.append(result)
        #print(final)
        #print(len(final))
    plt.plot(x1,final1)
    plt.show()
    #print(final1)
```

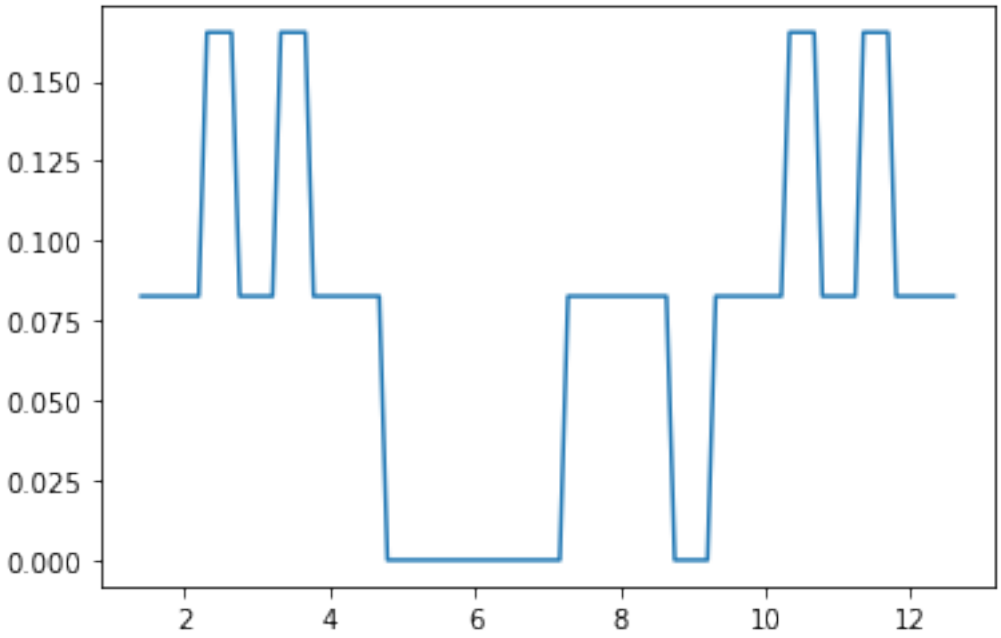
[7]:
```
cubic(0.5,xi)
```



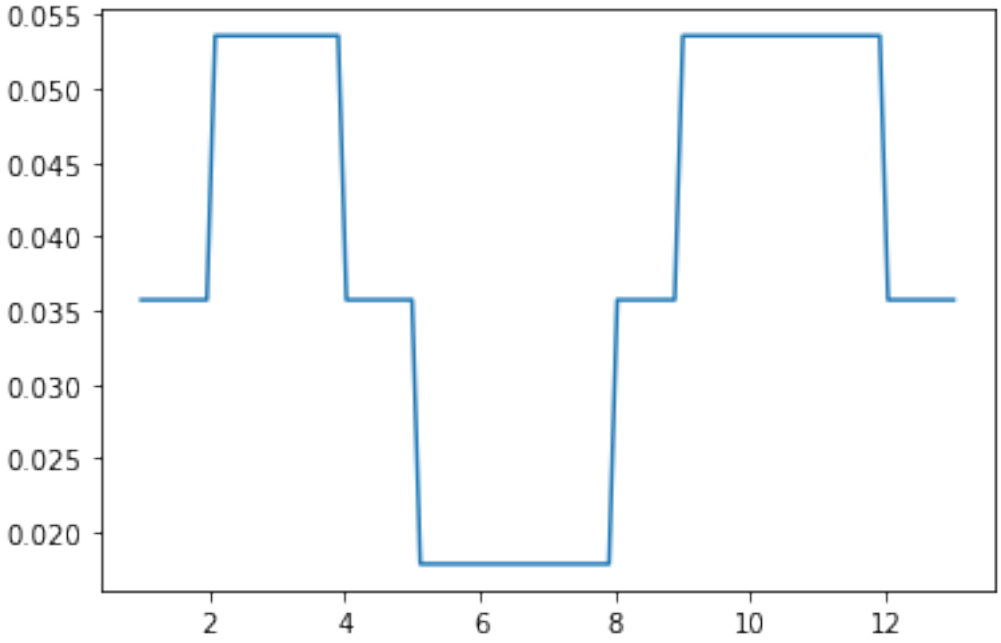
[8]:
```
cubic(1,xi)
```



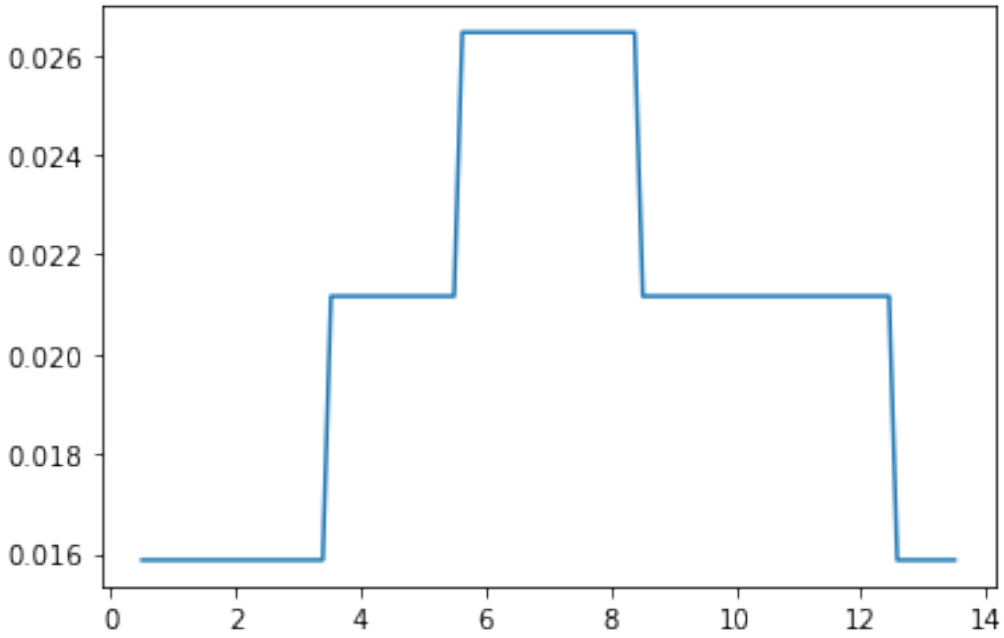
[9]:
```
cubic(1.2,xi)
```



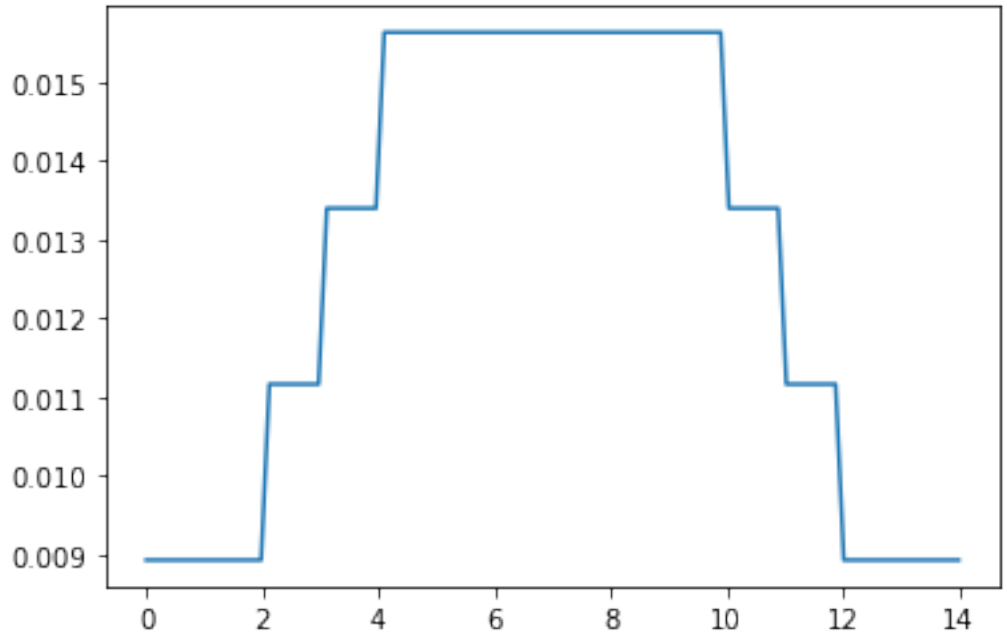
[10]:
```
cubic(2,xi)
```



[11]:
```
cubic(3,xi)
```


0.026
0.024
0.022
0.020
0.018
0.016
0
2
4
6
8
10
12
14

[13]:
```
cubic(4,xi)
```


0.015
0.014
0.013
0.012
0.011
0.010
0.009
0
2
4
6
8
10
12
14

[ ]:
```

```

Gaussian

November 21, 2020

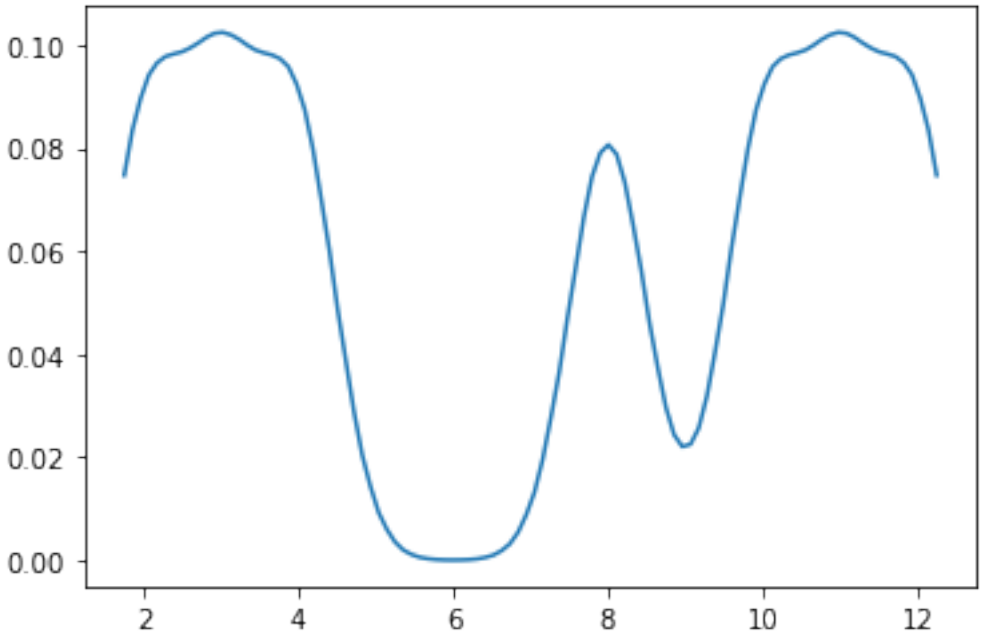
0.0.1 Question 1B: Gaussian

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

sample = [2,3,4,8,10,11,12]
```

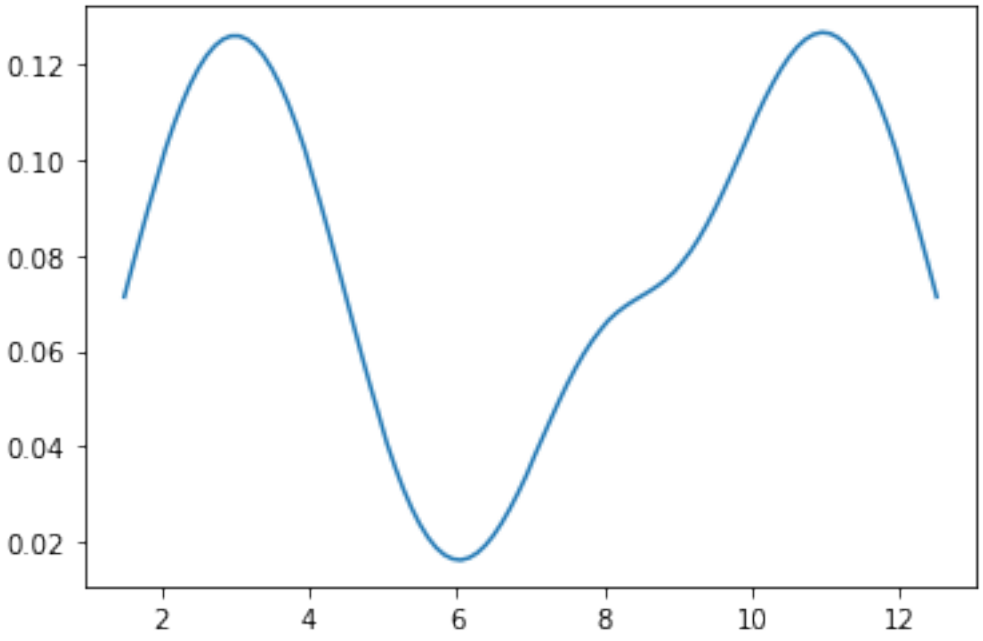
[2]:
```
def gauss_func(x,xi,h):
    out1=np.exp(-0.5*((x-xi)**2/h**2))
    #print(out1)
    final1=1/np.sqrt(2*3.14*h)
    #print(final1)
    output1=final1*out1
    #print(output1)
    #output1=int(output1)
    return output1
```

[3]:
```
def gaussian(data,h1):
    points1 = np.linspace(min(data)-h1/2,max(data)+h1/2,100)
    final0=[]
    res0=0
    for i in points1:
        i=round(i,2)
        for itr in data:
            res0=res0+gauss_func(i,itr,h1)
        result1=(1/len(data))*res0
        res0=0
        final0.append(result1)
    plt.plot(points1,final0)
    plt.show()
    #print(final0)
```

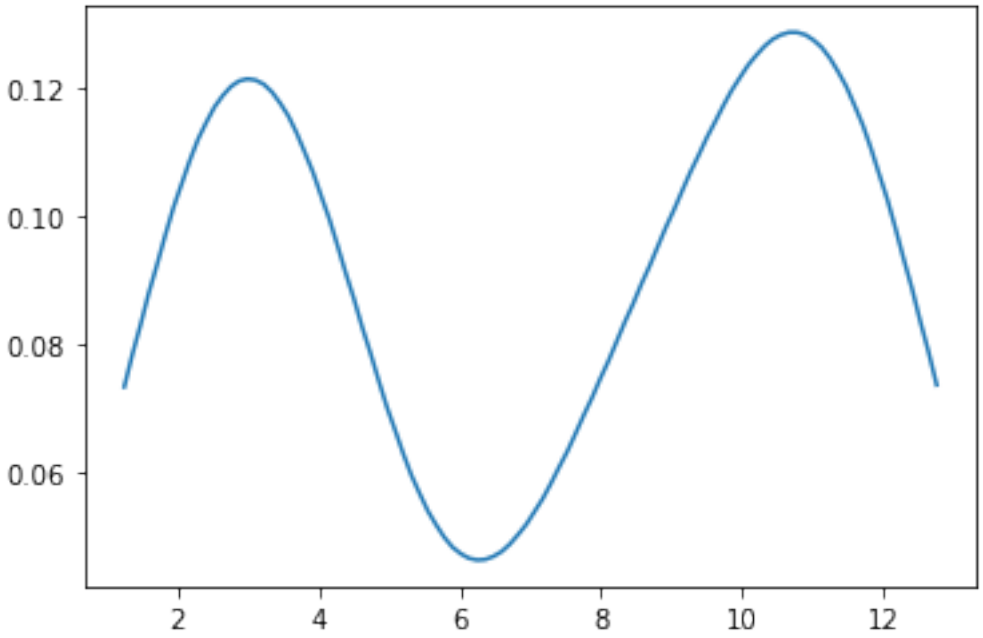
[4]:
```
gaussian(sample,0.5)
```



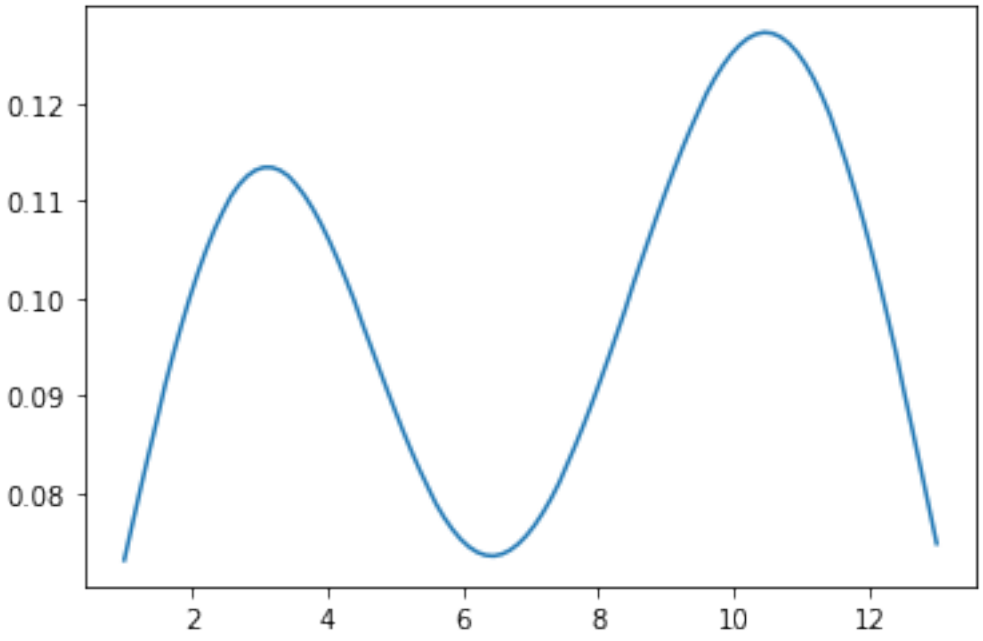
[5]:
```
gaussian(sample,1)
```



[6]:
```
gaussian(sample,1.5)
```



[7]:
```
gaussian(sample,2)
```



[ ]:
```

```

Question2

November 21, 2020

0.0.1 Question 2

Take a mixture of Gaussian and Uniform distribution (say by clearly separating the mean) and see that you can approximate the distribution a) by Parzen Window and b) KNN density. Vary k, h, and n and observe the accuracy of the computed density with true density (Use plot)

Parzen

[145]:
```
import numpy as np
import matplotlib.pyplot as plt
rand=np.random.default_rng()
```

[146]:
```
#x0=rand.normal(120,12,10)
#x0
```

[147]:
```
#xU=rand.uniform(-10,10,10)
```

[193]:
```
def Generate(n):
    mean=120
    std=12
    xG=rand.normal(120,12,n)
    xU=rand.uniform(-10,10,n)
    X=np.concatenate((xG,xU),None)
    np.random.shuffle(X)
    return X
```

[148]:
```
def gauss_func(x,xi,h):
    out1=np.exp(-0.5*((x-xi)**2/h**2))
    final1=1/np.sqrt(2*3.14*h)
    output1=final1*out1
    return output1
```

[188]:
```
def gaussian(data,h1):
    points1 = np.linspace(min(data)-h1/2,max(data)+h1/2,100)
    final0=[]
    res0=0
    for i in points1:
        i=round(i,2)
        for itr in data:
```

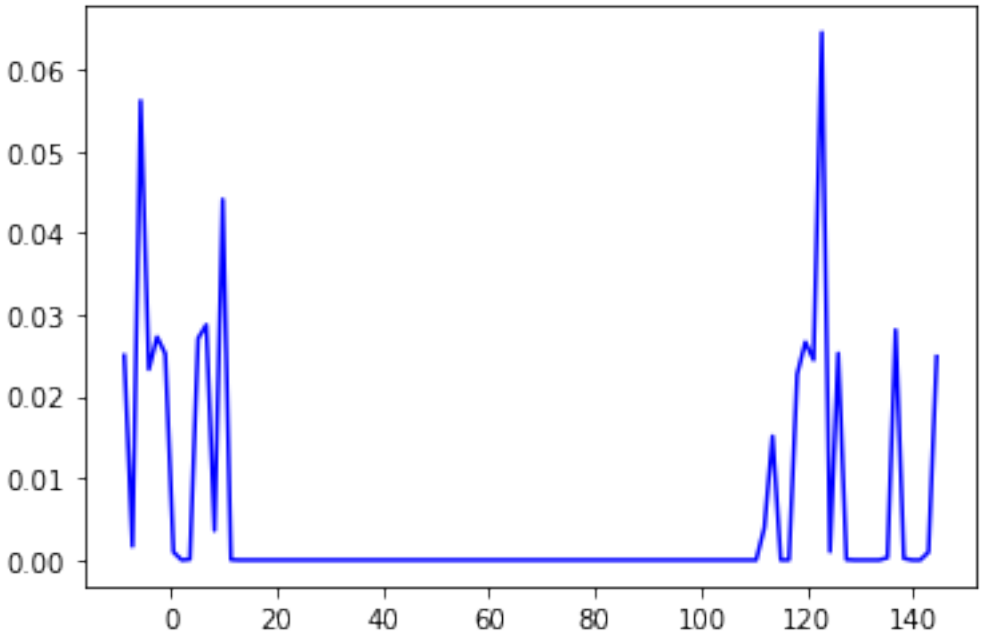
```
            res0=res0+gauss_func(i,itr,h1)
        result1=(1/len(data))*res0
        res0=0
        final0.append(result1)
    plt.plot(points1,final0,color='b')
    plt.show()
```

[194]:
```
X=Generate(10)
X
```

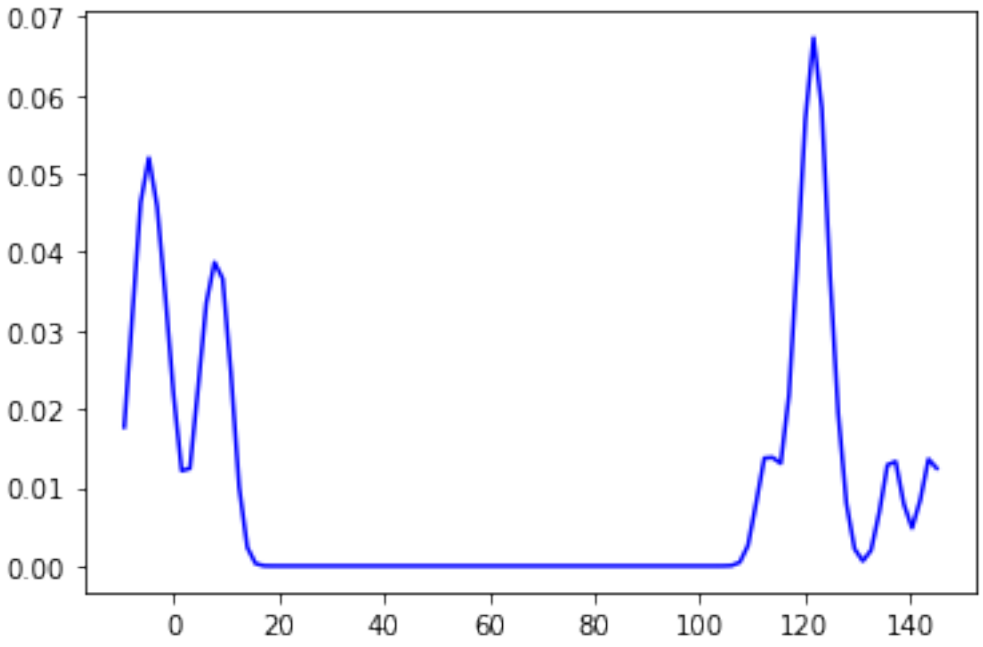
[194]:
```
array([  6.75477283, 122.51766461,  -4.44406447,  -8.48469725,
       136.59485768,   5.33462675, 122.09124055,  -5.53863106,
        -0.7588007 , 125.56561608, 119.37316231, 121.65849789,
       122.36077493,  -5.79341055,   9.46378731,  -2.68468481,
       112.87246914, 144.10900901,   9.47406876, 118.42999613])
```

Varying the h

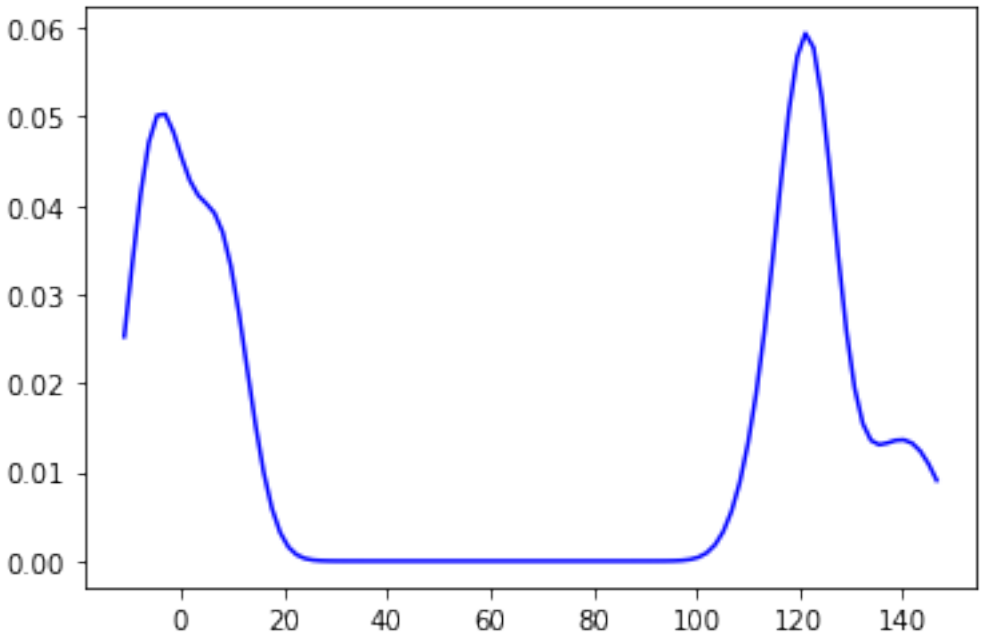
[195]:
```
gaussian(X,0.5)
```



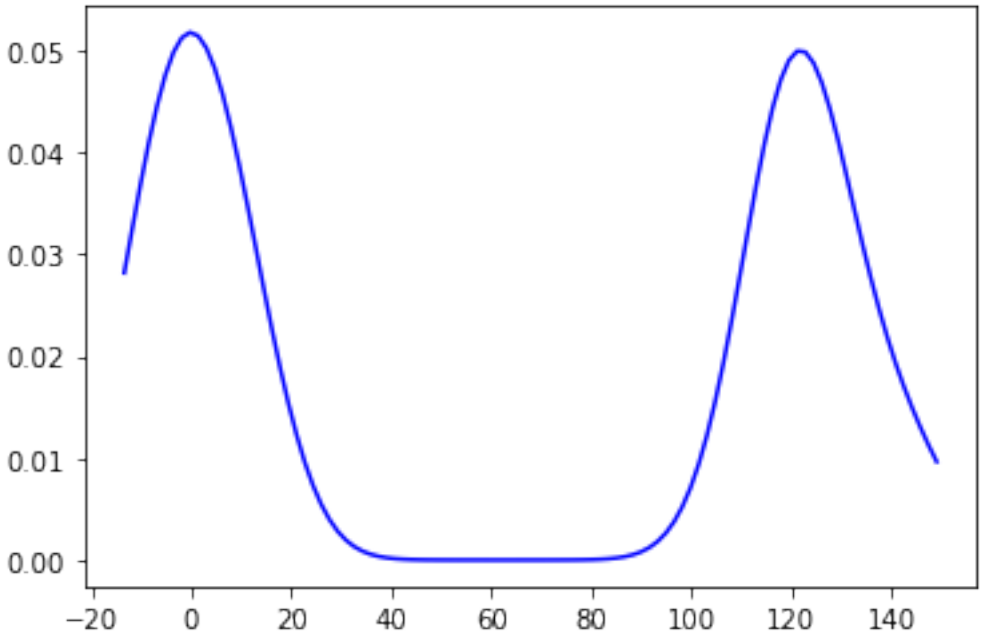
[196]:
```
gaussian(X,2)
```



[197]:
```
gaussian(X,5)
```



[198]:
```
gaussian(X,10)
```



KNN

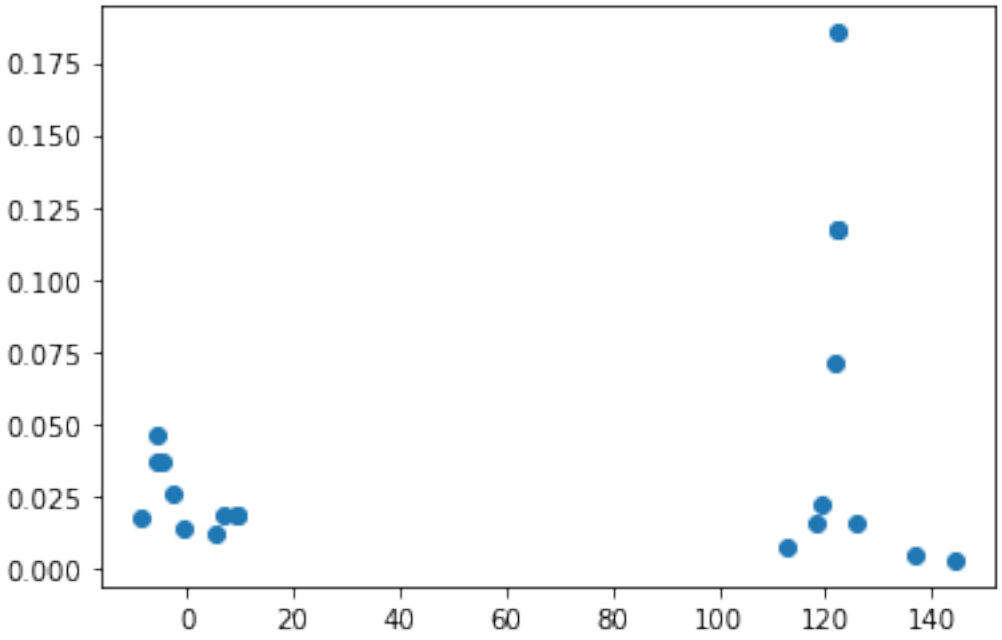
[164]:
```python
def probability_density_knn(sample, k, x):
    n = sample.size

    d = sample.ndim
    if d == 1:
        V = 2
    elif d == 2:
        V = np.pi
    elif d == 3:
        V = (4 / 3) * np.pi
    else:
        return

    # R is the distance from x to its kth nearest neighbour
    distance_from_x = np.absolute(sample - x)
    distance_from_x = np.sort(distance_from_x)
    R = distance_from_x[k]

    try:
        probability_density = (k / n) * (1 / (V * R) )
    except:
        print(f"k = {k}")
        print(f"n = {n}")
        print(f"V = {V}")
```

```
        print(f"R = {R}")
    return probability_density
```

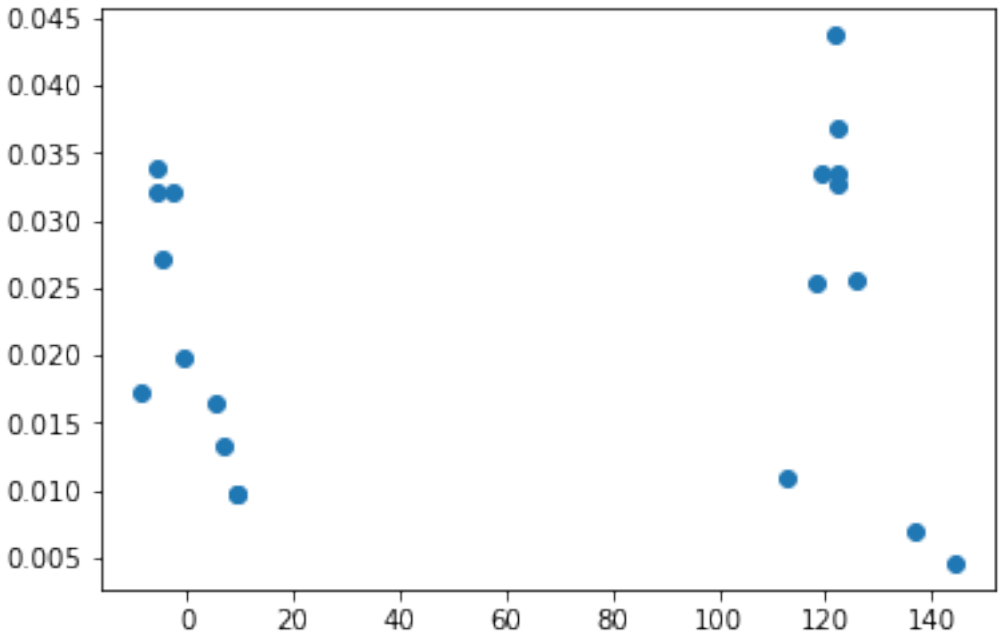
[206]:
```
def knn(sample, k):
    probability_density = [0]*len(sample)
    for i in range(sample.size):
        probability_density[i] = probability_density[i]␣
↪+probability_density_knn(sample, k, sample[i])
    plt.scatter(sample, probability_density)
    plt.show()
```

Varying the k

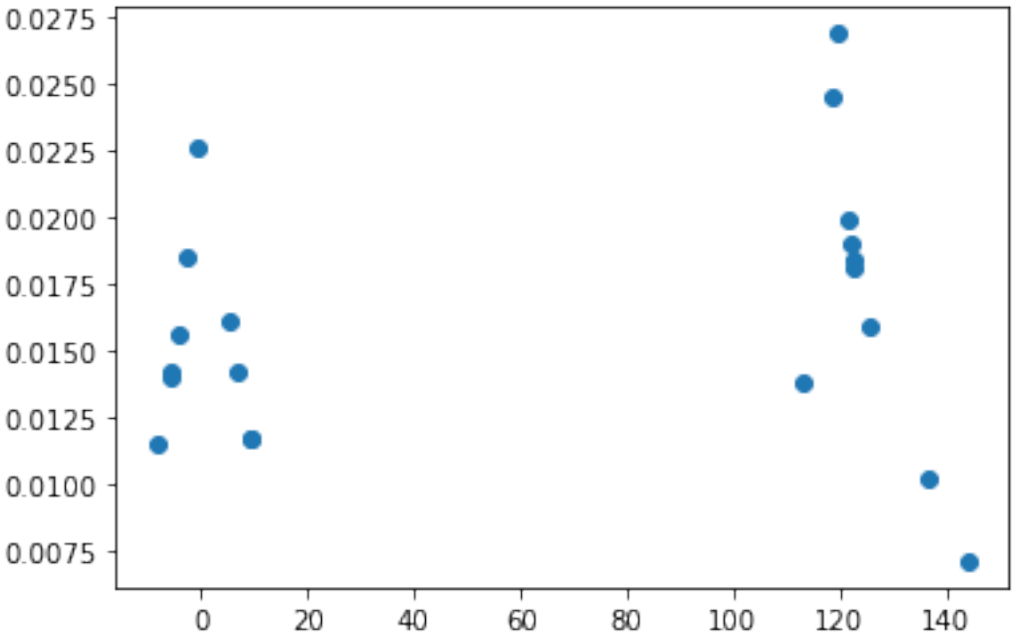
[207]:
```
knn(X, 2)
```



[208]:
```
knn(X,4 )
```



[209]:
```
knn(X, 7)
```



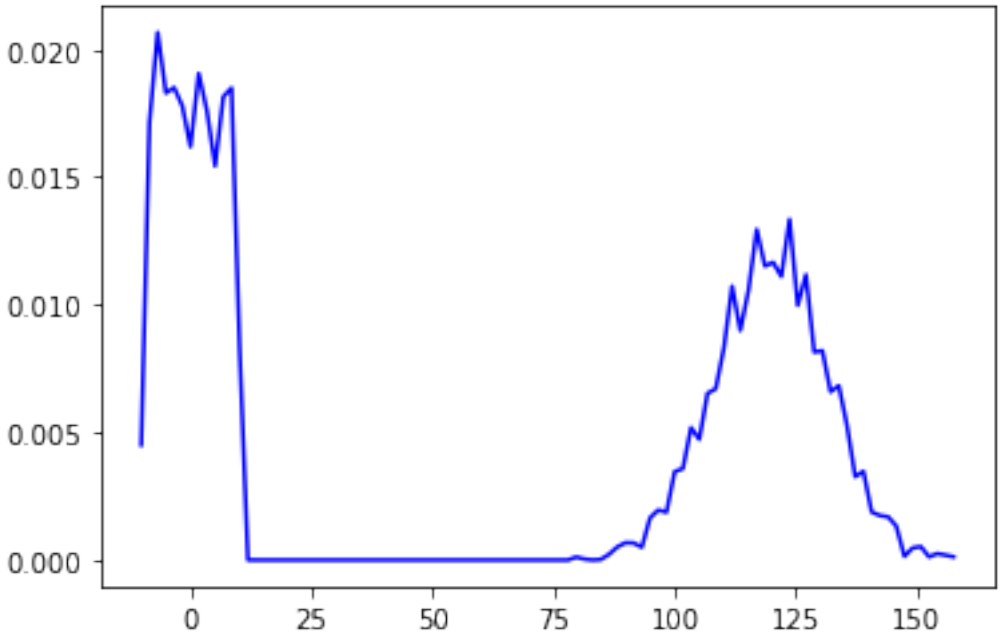
Increasing the sample size n to 4000 Parzen:

[199]:
```
X_new=Generate(2000)
X_new
```

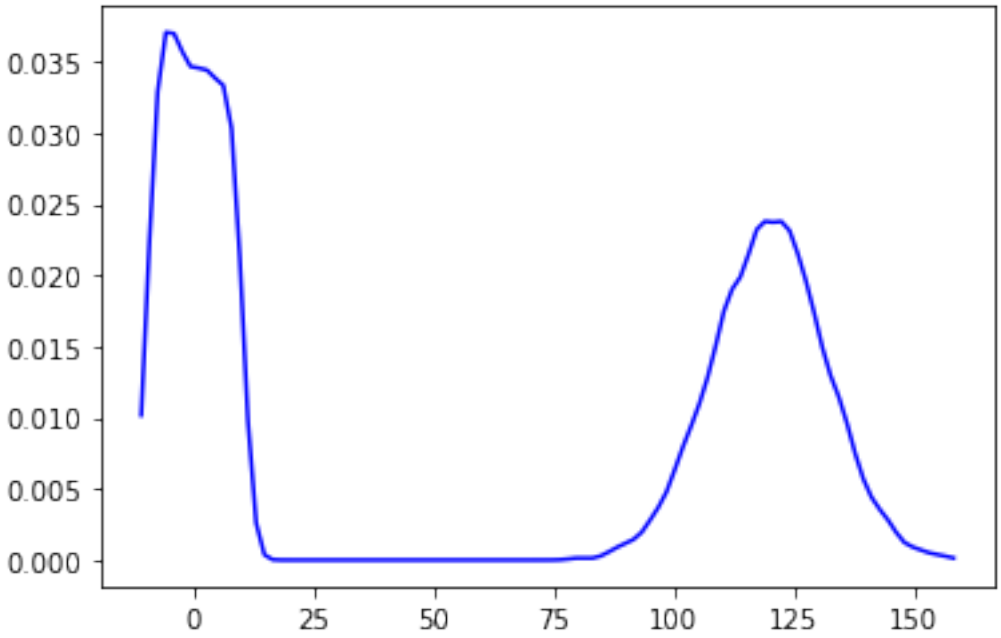
[199]:
```
array([ 9.11830890e+00, -3.51102848e-02, -7.43154823e+00, …,
        1.34803686e+02,  1.18110998e+02,  1.22604492e+02])
```

Parzen (varying values of h)

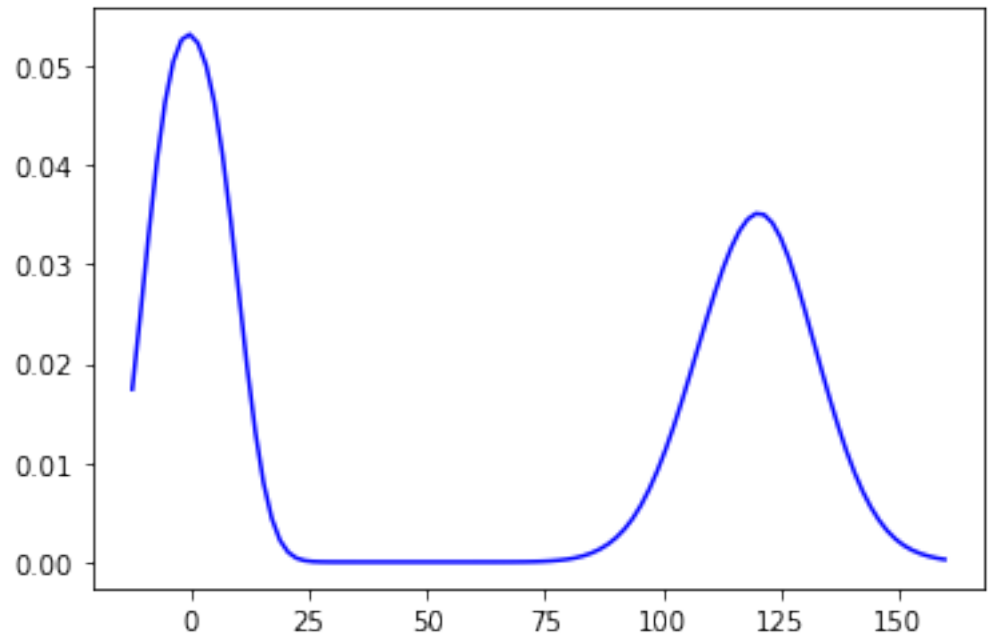
[200]:
```
gaussian(X_new,0.5)
```



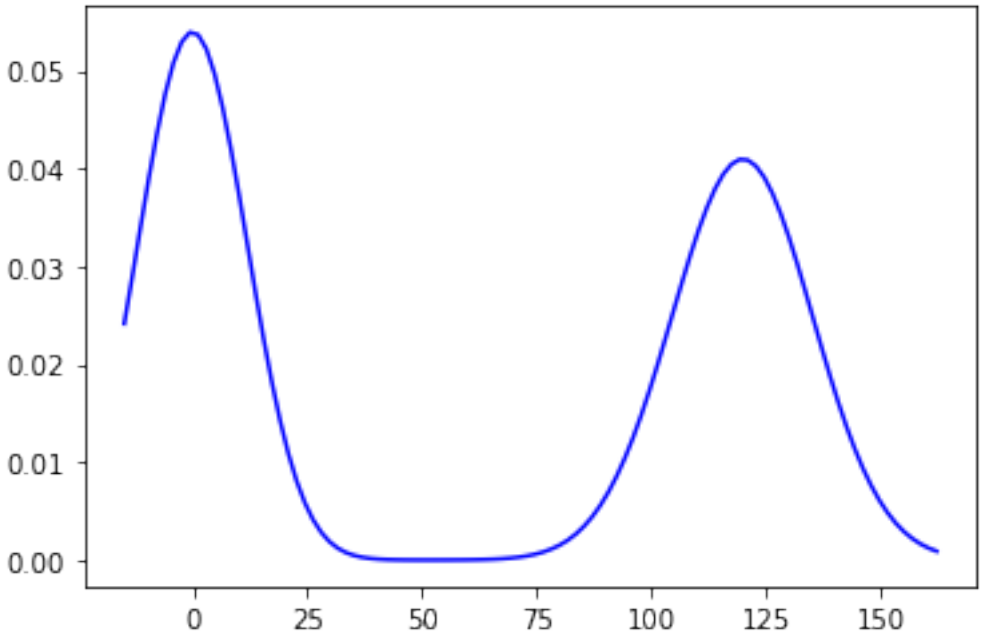
[201]:
```
gaussian(X_new,2)
```



[203]:
```
gaussian(X_new,5)
```

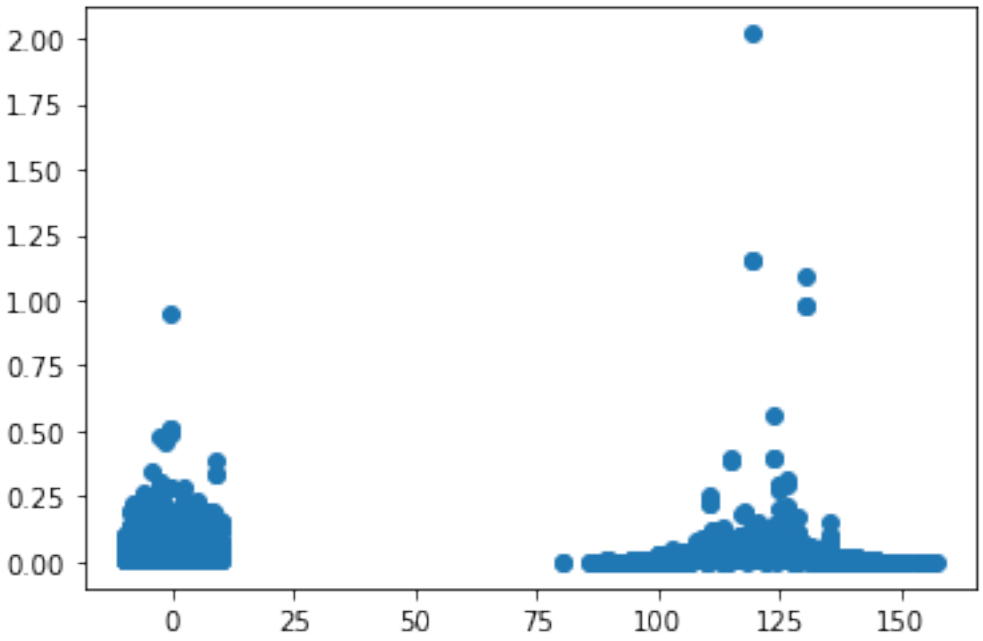


[204]:
```
gaussian(X_new,10)
```

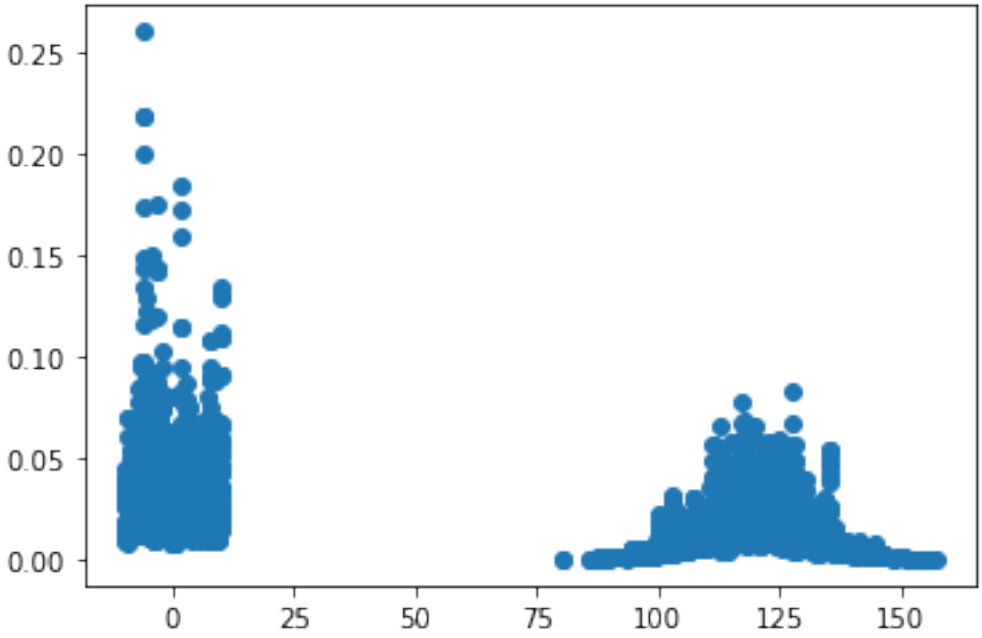


KNN (varying values of K):

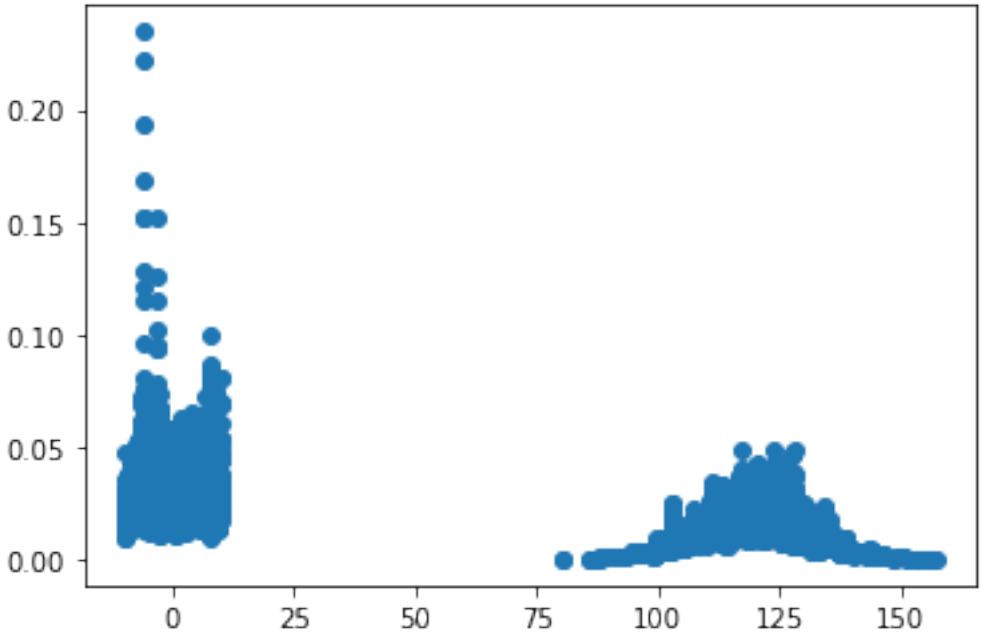
[214]:
```
knn(X_new,2)
```



[217]:
```
knn(X_new, 4)
```



[219]:
```
knn(X_new, 7)
```

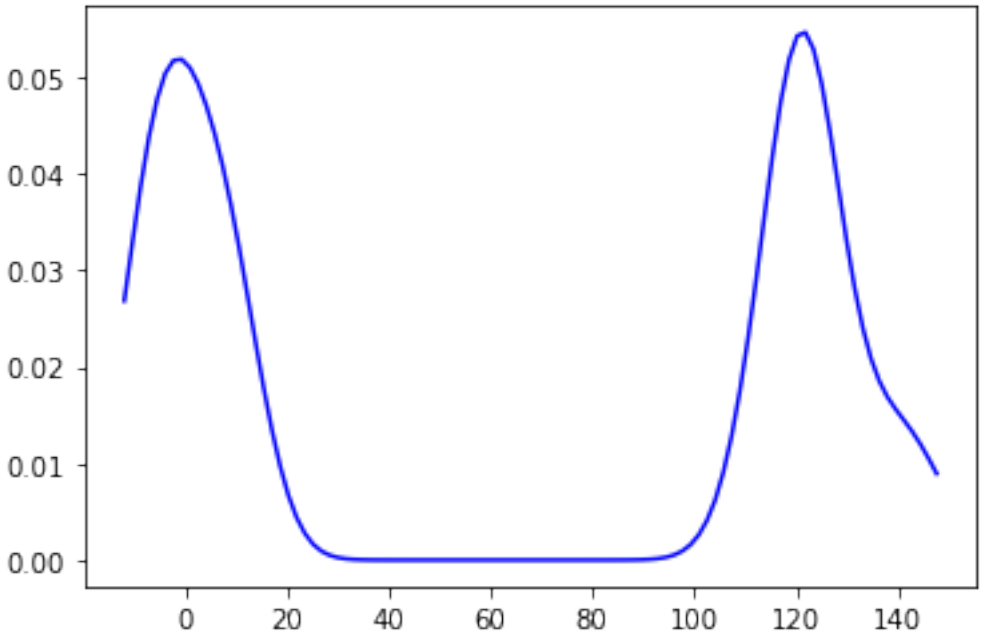


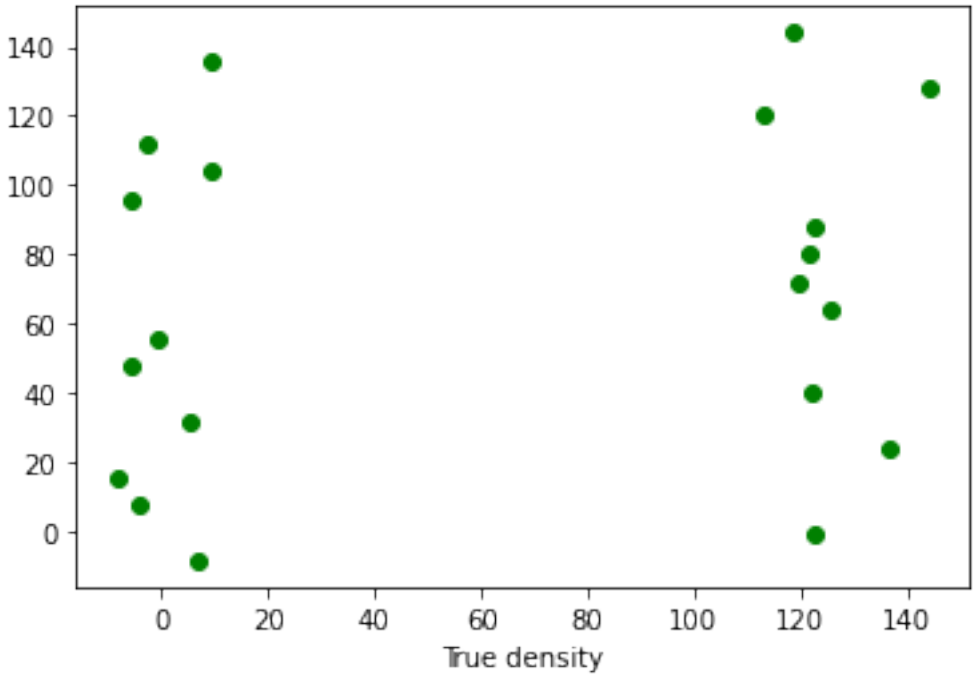
Comparing with true density Case1 (Parzen)

[220]:
```
X
```

[220]:
```
array([  6.75477283, 122.51766461,  -4.44406447,  -8.48469725,
       136.59485768,   5.33462675, 122.09124055,  -5.53863106,
        -0.7588007 , 125.56561608, 119.37316231, 121.65849789,
       122.36077493,  -5.79341055,   9.46378731,  -2.68468481,
       112.87246914, 144.10900901,   9.47406876, 118.42999613])
```

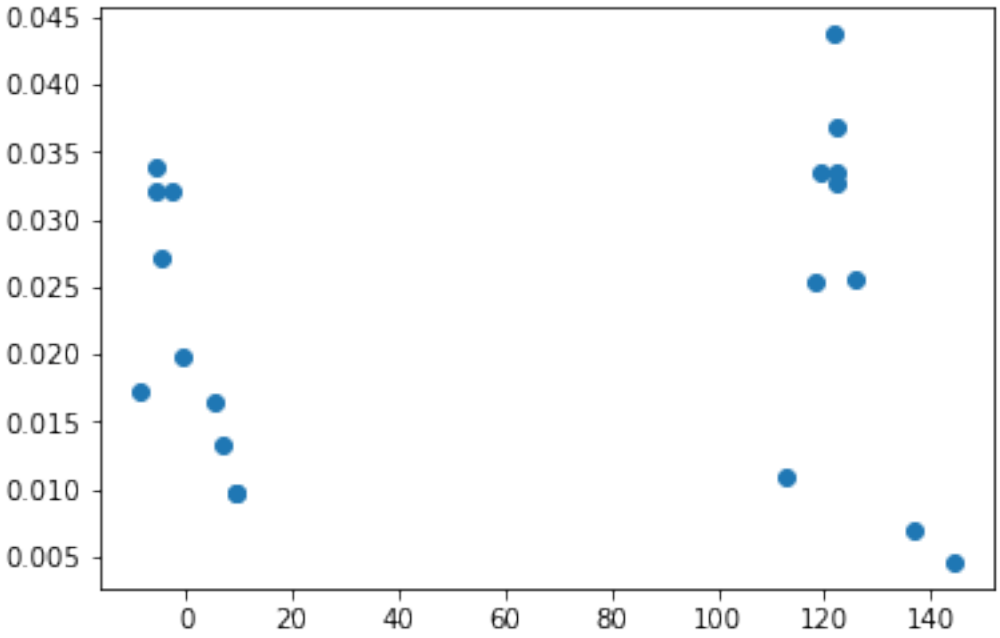
[276]:
```
gaussian(X,7)
plt.xlabel("True density")
plt.scatter(X,np.linspace(min(X),max(X),20),color='g')
plt.show()
```

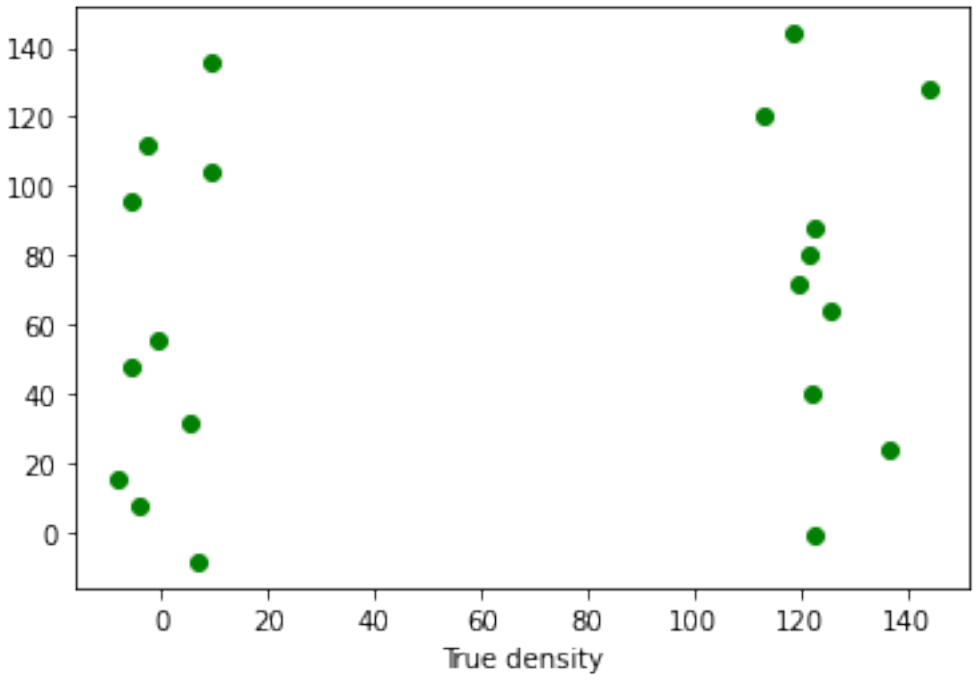




Case 1 (KNN)

[277]:
```
knn(X,4)
plt.xlabel("True density")
plt.scatter(X,np.linspace(min(X),max(X),20),color='g')
plt.show()
```



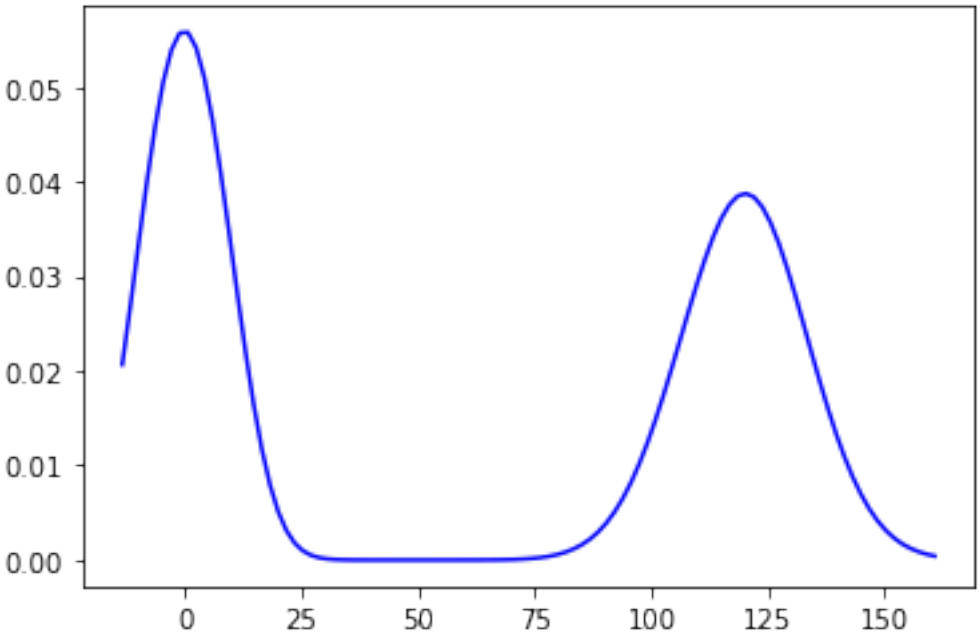


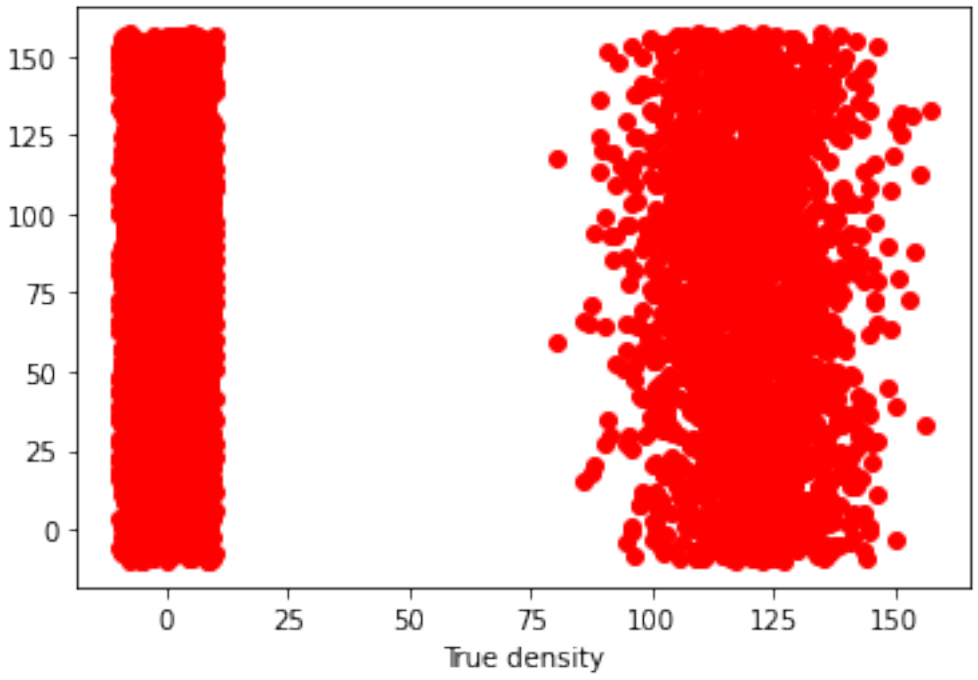
Case 2: (Parzen)

[263]:
```
X_new
```

[263]:
```
array([ 9.11830890e+00, -3.51102848e-02, -7.43154823e+00, …,
        1.34803686e+02,  1.18110998e+02,  1.22604492e+02])
```

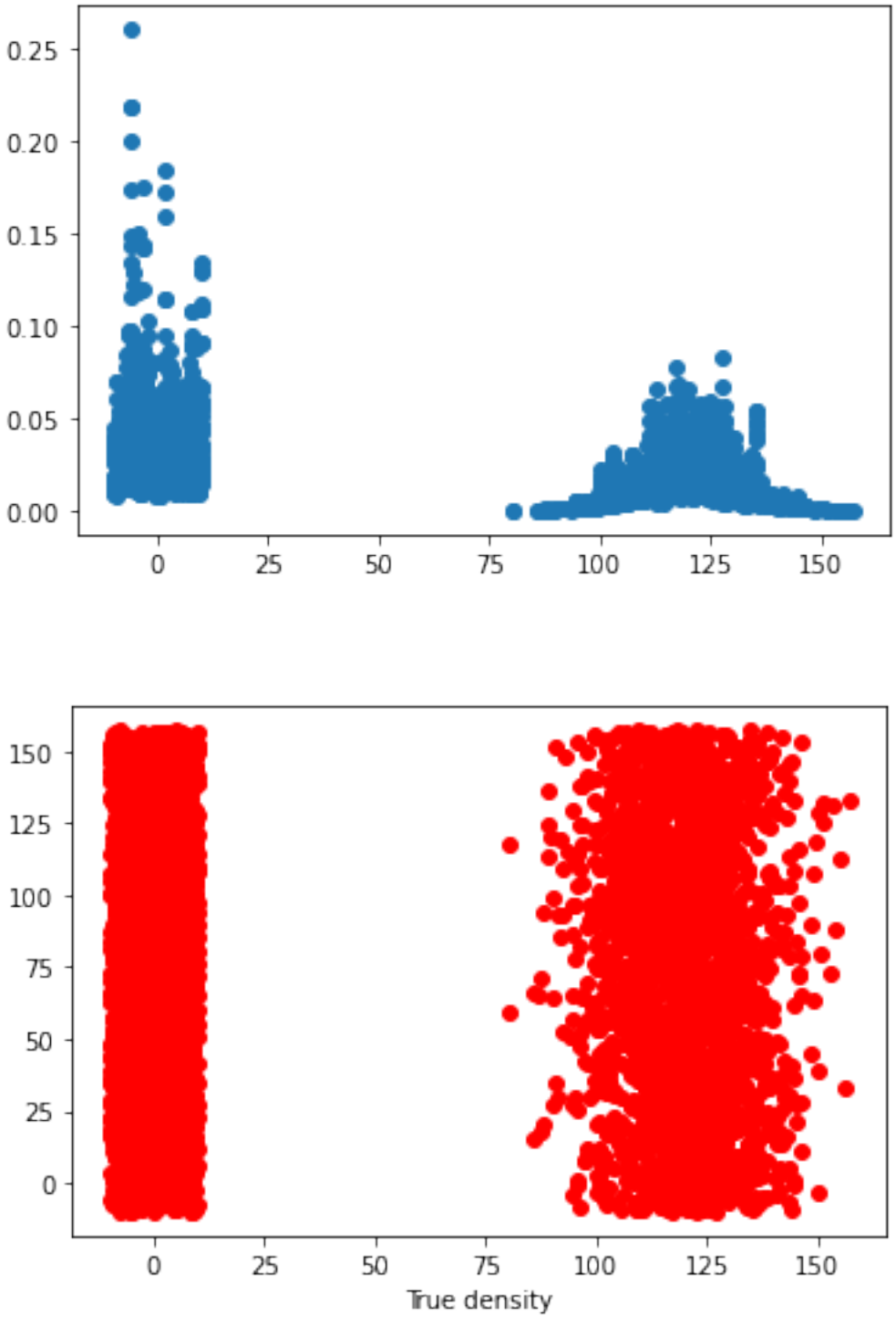
[275]:
```
gaussian(X_new,7)
plt.xlabel("True density")
plt.scatter(X_new,np.linspace(min(X_new),max(X_new),4000),color='r')
plt.show()
```


0.05
0.04
0.03
0.02
0.01
0.00
0
25
50
75
100
125
150


150
125
100
75
50
25
0
0
25
50
75
100
125
150
True density

Case 2: (KNN)

[274]:
```
knn(X_new, 4)
plt.xlabel("True density")
plt.scatter(X_new,np.linspace(min(X_new),max(X_new),4000),color='r')
plt.show()
```



[ ]:

Question3

November 21, 2020

0.0.1 Question 3:

Take Fisher Iris Dataset. Split it into 70% for training and 30% for testing. Assess the accuracy with KNN classifier for K = 1, 3, 5, 7, 9, 11. You can use built-in functions.

[2]:
```
import numpy as np
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

[3]:
```
#Load
iris = datasets.load_iris()
x = iris.data
y = iris.target
```

[4]:
```
columns = iris.feature_names
columns
```

[4]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[5]:
```
target = iris.target_names
target
```

[5]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

[11]:
```
iris_main = pd.DataFrame(x , columns=[columns])
iris_main
```

[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2

```
4                5.0               3.6                1.4               0.2
..               …                 …                  …                 …
145              6.7               3.0                5.2               2.3
146              6.3               2.5                5.0               1.9
147              6.5               3.0                5.2               2.0
148              6.2               3.4                5.4               2.3
149              5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

[12]:
```
iris_last = pd.DataFrame(y , columns=['target'])
iris_last
```

[12]:
```
     target
0         0
1         0
2         0
3         0
4         0
..      …
145       2
146       2
147       2
148       2
149       2

[150 rows x 1 columns]
```

[13]:
```
X = iris_main.copy()
Y = iris_last.copy()
```

[14]:
```
X_TRAIN, X_TEST, Y_TRAIN, Y_TEST = train_test_split(X, Y, test_size =0.3)
```

K=1

[15]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(X_TRAIN, Y_TRAIN)
```

```
<ipython-input-15-9d37b9d50445>:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  knn_classifier.fit(X_TRAIN, Y_TRAIN)
```

[15]:
```
KNeighborsClassifier(n_neighbors=1)
```

[85]:
```
Y_PRED_TEST = knn_classifier.predict(X_TEST)
```

[86]:
```
Y_PRED_TEST
```

[86]:
```
array([0, 0, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2, 1, 1, 0, 1, 1,
       2, 1, 2, 0, 2, 2, 1, 0, 1, 2, 2, 2, 0, 2, 2, 2, 0, 0, 0, 2, 2, 1,
       0])
```

[87]:
```
print(f'Test Accuracy = {accuracy_score(Y_TEST, Y_PRED_TEST)}')
```

```
Test Accuracy = 0.9333333333333333
```

K=3

[88]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_TRAIN, Y_TRAIN)
```

```
<ipython-input-88-0e5edd7f4ef4>:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  knn_classifier.fit(X_TRAIN, Y_TRAIN)
```

[88]:
```
KNeighborsClassifier(n_neighbors=3)
```

[89]:
```
Y_PRED_TEST = knn_classifier.predict(X_TEST)
print(f'Test Accuracy = {accuracy_score(Y_TEST, Y_PRED_TEST)}')
```

```
Test Accuracy = 0.9555555555555556
```

K=5

[90]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_TRAIN, Y_TRAIN)
Y_PRED_TEST = knn_classifier.predict(X_TEST)
print(f'Test Accuracy = {accuracy_score(Y_TEST, Y_PRED_TEST)}')
```

```
Test Accuracy = 0.9777777777777777

<ipython-input-90-1473a9ef10f9>:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  knn_classifier.fit(X_TRAIN, Y_TRAIN)
```

K=7

[91]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=7)
knn_classifier.fit(X_TRAIN, Y_TRAIN)
Y_PRED_TEST = knn_classifier.predict(X_TEST)
print(f'Test Accuracy = {accuracy_score(Y_TEST, Y_PRED_TEST)}')
```

```
Test Accuracy = 0.9777777777777777

<ipython-input-91-f37b932d61f3>:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  knn_classifier.fit(X_TRAIN, Y_TRAIN)
```

K=9

[92]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=9)
knn_classifier.fit(X_TRAIN, Y_TRAIN)
Y_PRED_TEST = knn_classifier.predict(X_TEST)
print(f'Test Accuracy = {accuracy_score(Y_TEST, Y_PRED_TEST)}')
```

```
Test Accuracy = 0.9777777777777777
```

```
<ipython-input-92-9c3e2db87573>:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  knn_classifier.fit(X_TRAIN, Y_TRAIN)
```

K=11

[93]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=11)
knn_classifier.fit(X_TRAIN, Y_TRAIN)
Y_PRED_TEST = knn_classifier.predict(X_TEST)
print(f'Test Accuracy = {accuracy_score(Y_TEST, Y_PRED_TEST)}')
```

```
Test Accuracy = 0.9555555555555556
```

```
<ipython-input-93-916f1888ddcc>:2: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  knn_classifier.fit(X_TRAIN, Y_TRAIN)
```