Make The Training Arrays and Give Them Corresponding Y-Cap Values

In [1]:

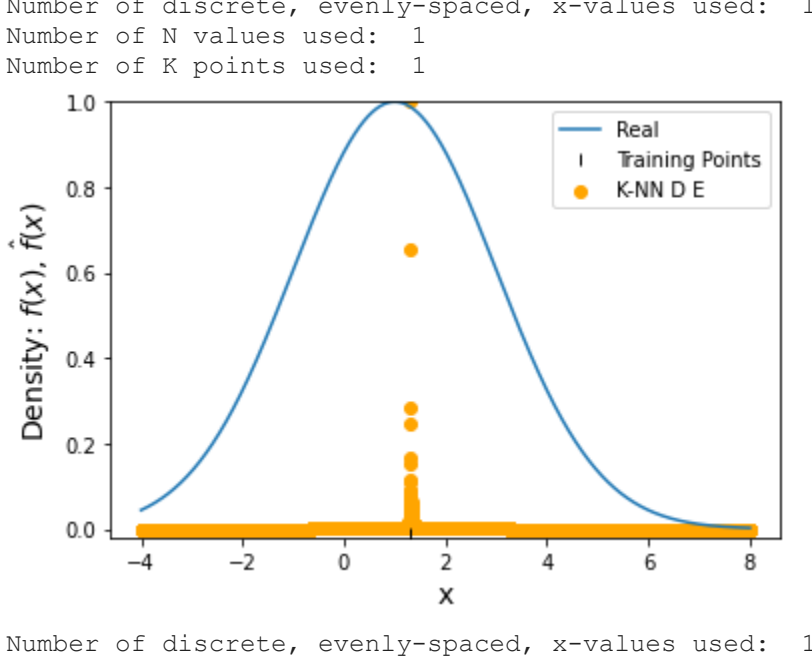
```python
import numpy as np
from multiprocessing import Process
import math
import matplotlib.pyplot as plt
from sklearn.neighbors import KernelDensity
from scipy.stats.distributions import norm
from sklearn.neighbors import NearestNeighbors

#define some kernels for future use
def gauss(x):
    return (1/math.sqrt(2*math.pi))*math.exp(-0.5*(x**2))

#set seed for reproducible results
np.random.seed(467)

#Generates a set of X values to
#be used as the X training set.
#Y-cap values will be generated
#separately if needed.
def generateXTrainGuassian(n):

    #random generation function
    x = lambda t: t + np.random.normal(1, 2)
    vfunc = np.vectorize(x)

    data = np.full(shape=n, fill_value=0, dtype=float)

    return vfunc(data)

#function for generating final number for the
#custom distribution array
def customFit():
    x = 0
    a = 0
    while (a == 0):
        x = np.random.uniform(-4,22)
        f1 = (.4 * ((x <= 3 and x >= 2)*(1) + 0))
        f2 = (.6 * ((x <= 20 and x >= 15)*(0.2) + 0))
        a = f1 + f2
    return (x)

#Generates a set of X values to
#be used as the X training set.
#Y-cap values will be generated
#separately if needed.
#uses a custom distribution
#of f(x) = 0.4f1(x) + 0.6f2(x)
#where f1(x) = 1 if x <= 3 and >=2; 0 otherwise
#where f2(x) = .2 if x <= 20, >= 15; 0 otherwise
def generateXTrainCustom(n):

    #random generation function
    x = lambda t: t + customFit()
    vfunc = np.vectorize(x)

    data = np.full(shape=n, fill_value=0, dtype=float)

    return vfunc(data)

#function to find the Kth nearest
#neighbor's distance from the
#targetted x point
def findV(x_o, k, x_train):
    dist = []

    #get all calculated distances
    for x in x_train:
        dist.append(abs(x_o-x))

    #sort distances ascending
    dist = np.sort(dist, kind='quick sort')

    #get the desired value
    return dist[k-1]
```

Question 1: Generating plots using K-NN density Estimation (Guassian distribution)

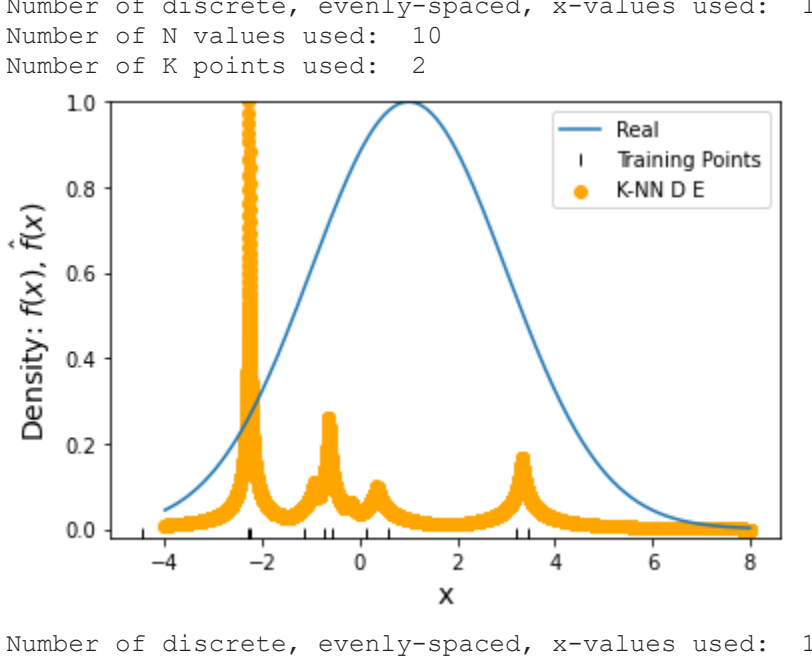
In [2]:

```python
#function to generate density plots
#consisting of the actual density,
#the points, and our estimated density
def generatePlots(x_train, k):

    #==========================================
    #Generating the Real Density happens here
    #==========================================
    x_values = np.linspace(-4, 8, 2000)
    mu = 1
    sig = 2
    y_carpet_values = np.exp(-np.power(x_values - mu, 2.) / (2 * np.power(sig, 2.)))

    #setup the axis labels
    plt.xlabel('x', fontsize=14)
    plt.ylabel('Density: $f(x)$, $\hat{f}(x)$', fontsize=14)

    #plot the rug values and the real distribution
    plt.plot(x_values, y_carpet_values)
    plt.plot(x_train, np.full_like(x_train, -0.01), '|k', markeredgewidth=1)
    plt.ylim(-0.02, 1)

    #==========================================
    #Plotting the estimation using my programming
    #Happens here
    #Equation: P = K/NV
    #Where P is density, K is neighbor count,
    # N is the total datapoints and V is the
    # hypervolume
    #==========================================
    dist = []
    a = np.arange(-4,8,.00120)

    print("Number of discrete, evenly-spaced, x-values used:  " + str(len(a)))
    print("Number of N values used:  " + str(len(x_train)))
    print("Number of K points used:  " + str(k))

    #get the hyperVolume (distance since it's 1D)
    for i in range(len(a)):
        dist.append((k/len(x_train))*(1/findV(a[i], k, x_train)))

    #plot the resultant estimate density
    dist = np.array(dist)
    dist = (dist - dist.min())/ (dist.max() - dist.min())
    plt.scatter(a, dist, c="orange")
    plt.legend(['Real', 'Training Points', 'K-NN D E'])
    plt.show()
    return


#make our training sets that we need here
x_1_train = generateXTrainGuassian(1)
x_10_train = generateXTrainGuassian(10)
x_1000_train = generateXTrainGuassian(1000)
x_50000_train = generateXTrainGuassian(50000)

#generate each plot
generatePlots(x_1_train, 1)
generatePlots(x_10_train, 2)
generatePlots(x_10_train, 10)
generatePlots(x_1000_train, 10)
generatePlots(x_1000_train, 100)
generatePlots(x_50000_train, 50000)
```

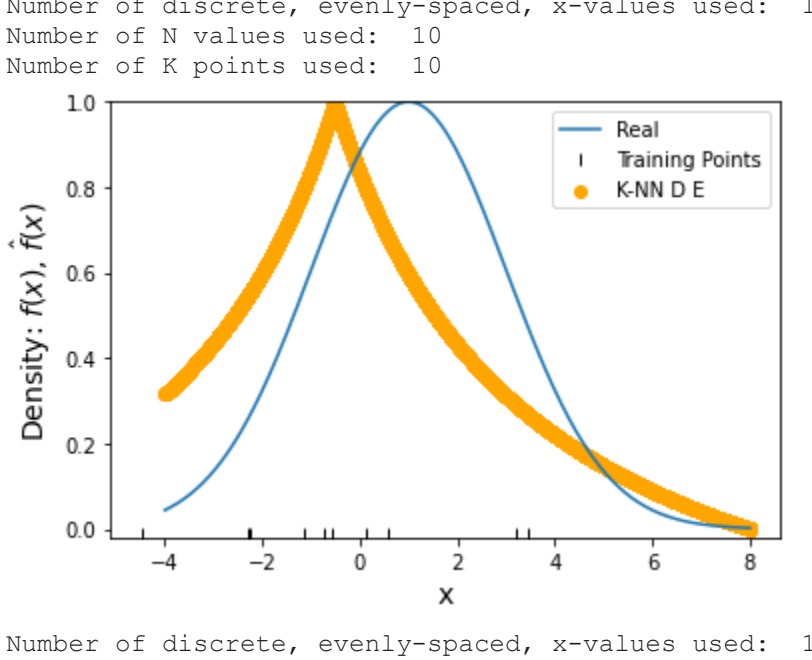
```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  1
Number of K points used:  1
```



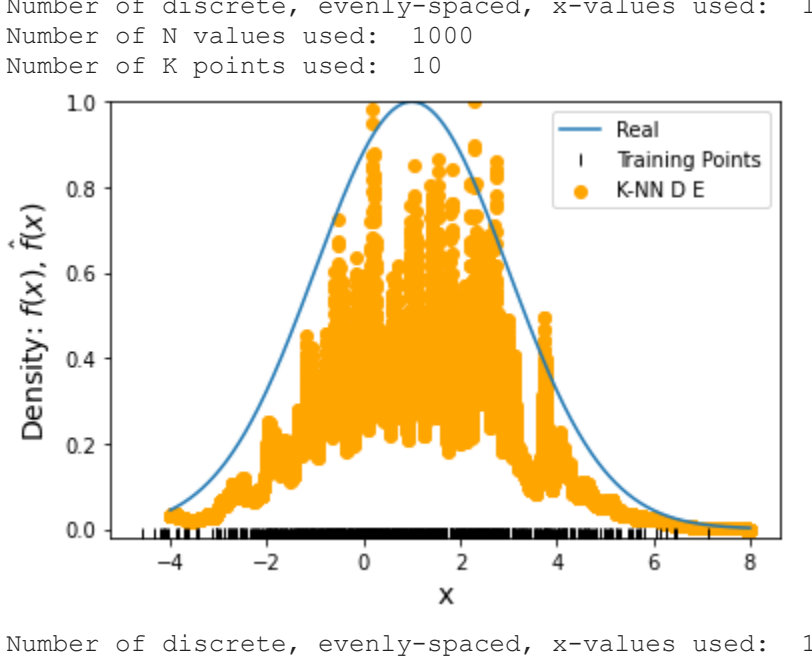

```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  10
Number of K points used:  2
```



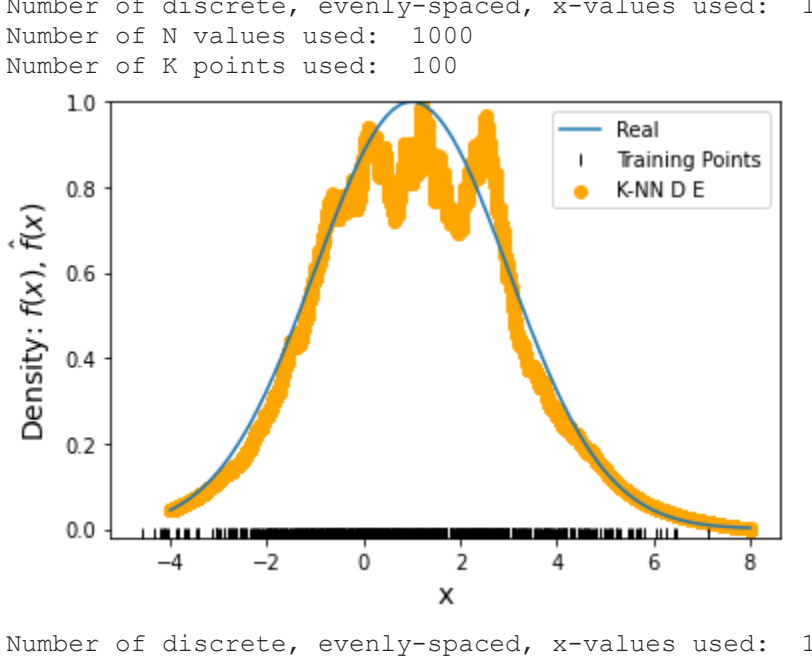

```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  10
Number of K points used:  10
```



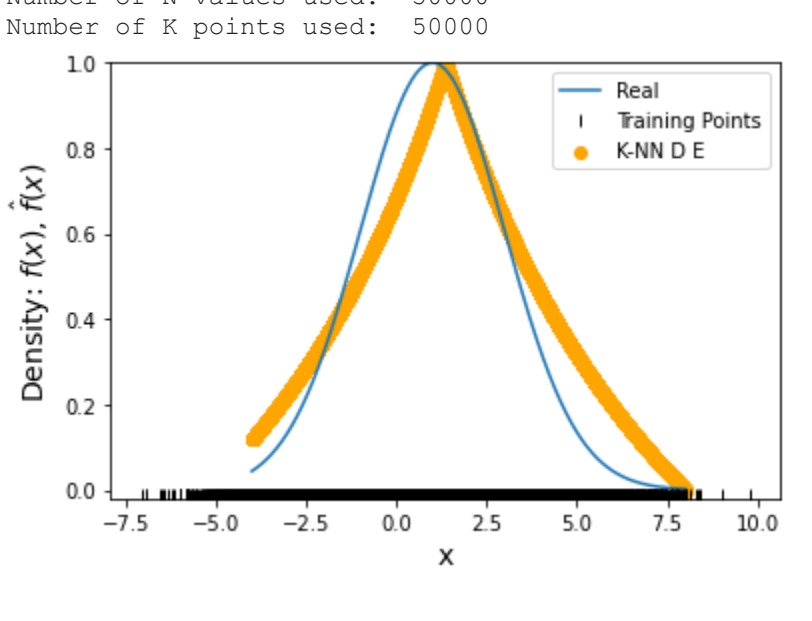

```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  1000
Number of K points used:  10
```




```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  1000
Number of K points used:  100
```




```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  50000
Number of K points used:  50000
```

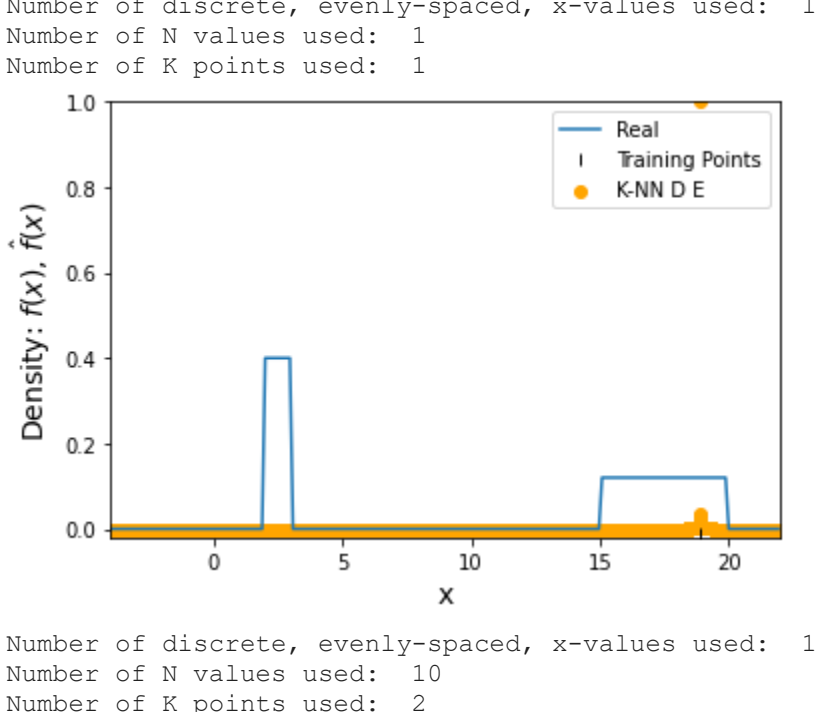


Question 2: Generating plots using K-NN density Estimation (custom distribution)

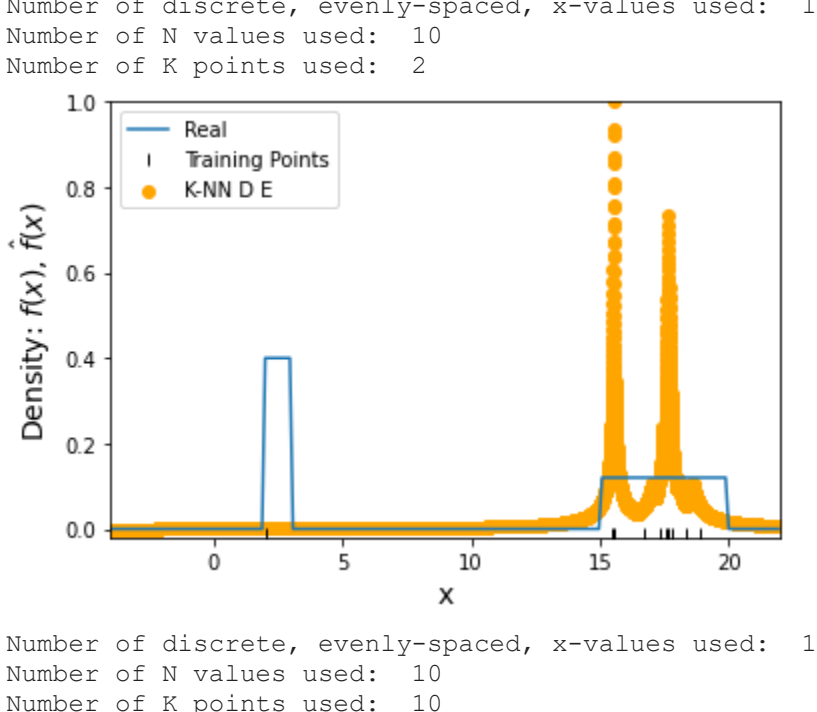
In [3]:

```python
#function to calc the y values for our
#custom distribution
#function for generating final number for the
#custom distribution array
def customFitPlot(x):
    f1 = (.4 * ((x <= 3 and x >= 2)*(1) + 0))
    f2 = (.6 * ((x <= 20 and x >= 15)*(0.2) + 0))
    a = f1 + f2
    return (a * (a > 0))

#function to generate density plots
#consisting of the actual density,
#the points, and our estimated density
def generateCustomPlots(x_train, k):

    #==========================================
    #Generating the Real Density happens here
    #==========================================
    x_values = np.linspace(-4, 8, 2000)
    mu = 1
    sig = 2

    #setup the axis labels
    plt.xlabel('x', fontsize=14)
    plt.ylabel('Density: $f(x)$, $\hat{f}(x)$', fontsize=14)

    #get the real dist for the custom set
    x_custom_values = np.arange(-4,22,.120)
    y_custom_values = np.full(shape=len(x_custom_values), fill_value=0, dtype=float)
    for i in range(len(x_custom_values)):
        y_custom_values[i] = customFitPlot(x_custom_values[i])

    #plot the rug values and the real distribution
    plt.plot(x_custom_values, y_custom_values)
    plt.plot(x_train, np.full_like(x_train, -0.01), '|k', markeredgewidth=1)
    plt.ylim(-0.02, 1)
    plt.xlim(-4, 22)

    #==========================================
    #Plotting the estimation using my programming
    #Happens here
    #Equation: P = K/NV
    #Where P is density, K is neighbor count, and
    #V is the hypervolume
    #==========================================
    dist = []
    a = np.arange(-4,22,.0026)

    print("Number of discrete, evenly-spaced, x-values used:  " + str(len(a)))
    print("Number of N values used:  " + str(len(x_train)))
    print("Number of K points used:  " + str(k))

    #get the hyperVolume (distance since it's 1D)
    for i in range(len(a)):
        dist.append((k/len(x_train))*(1/findV(a[i], k, x_train)))

    #plot the resultant estimate density
    dist = np.array(dist)
    dist = (dist - dist.min())/ (dist.max() - dist.min())
    plt.scatter(a, dist, c="orange")
    plt.legend(['Real', 'Training Points', 'K-NN D E'])
    plt.show()
    return


#make our training sets that we need here
x_1_custom_train = generateXTrainCustom(1)
x_10_custom_train = generateXTrainCustom(10)
x_1000_custom_train = generateXTrainCustom(1000)
x_50000_custom_train = generateXTrainCustom(50000)

#generate each plot
generateCustomPlots(x_1_custom_train, 1)
generateCustomPlots(x_10_custom_train, 2)
generateCustomPlots(x_10_custom_train, 10)
generateCustomPlots(x_1000_custom_train, 10)
generateCustomPlots(x_1000_custom_train, 100)
generateCustomPlots(x_50000_custom_train, 50000)
```

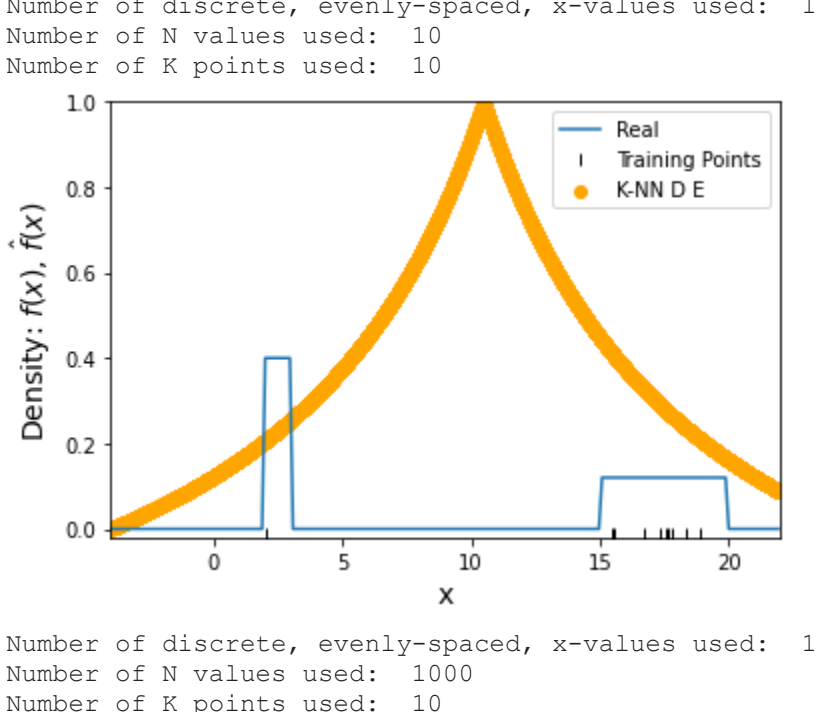
```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  1
Number of K points used:  1
```



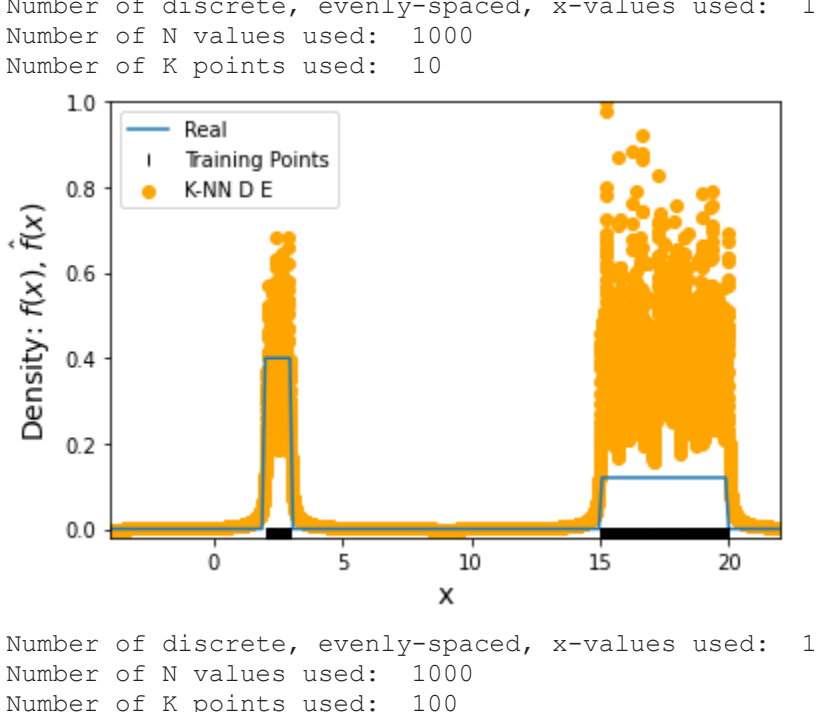

```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  10
Number of K points used:  2
```



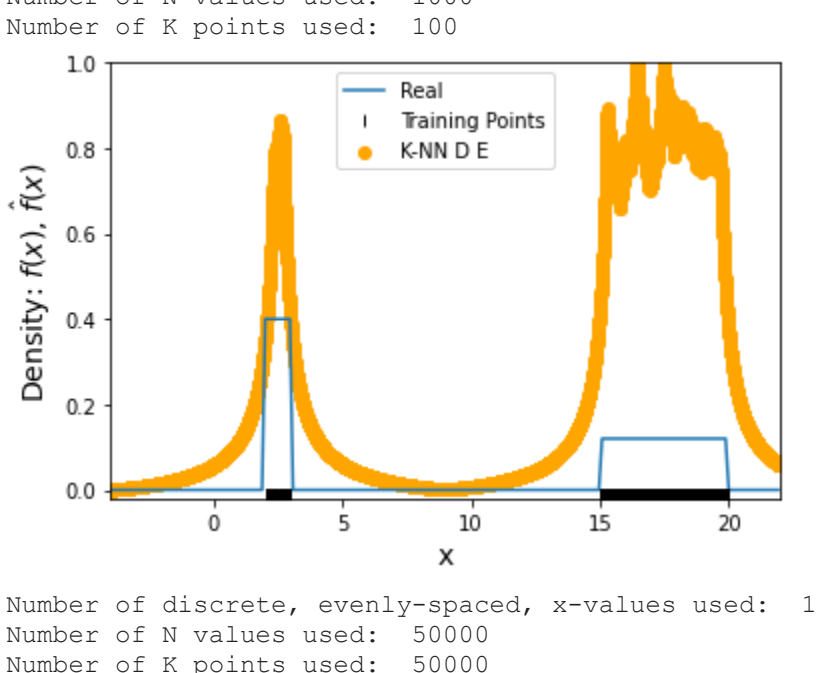

```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  10
Number of K points used:  10
```



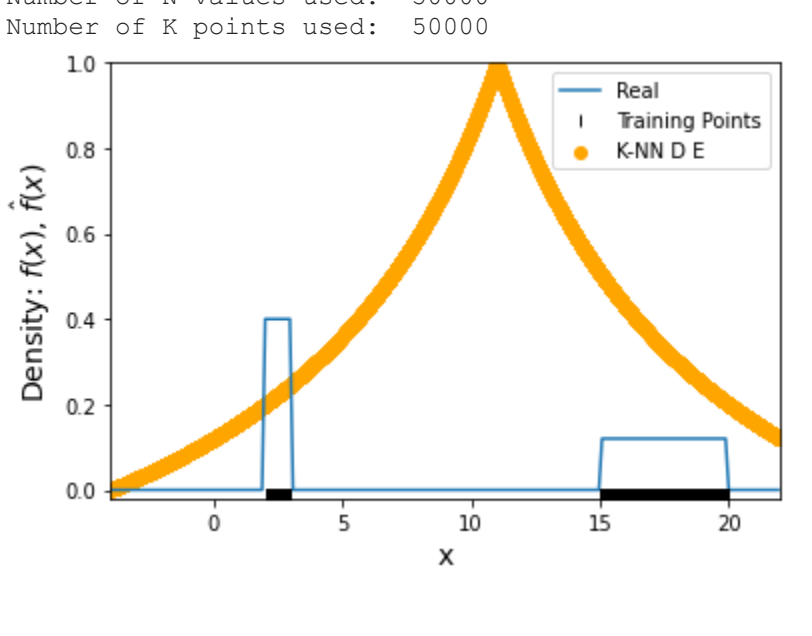

```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  1000
Number of K points used:  10
```




```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  1000
Number of K points used:  100
```



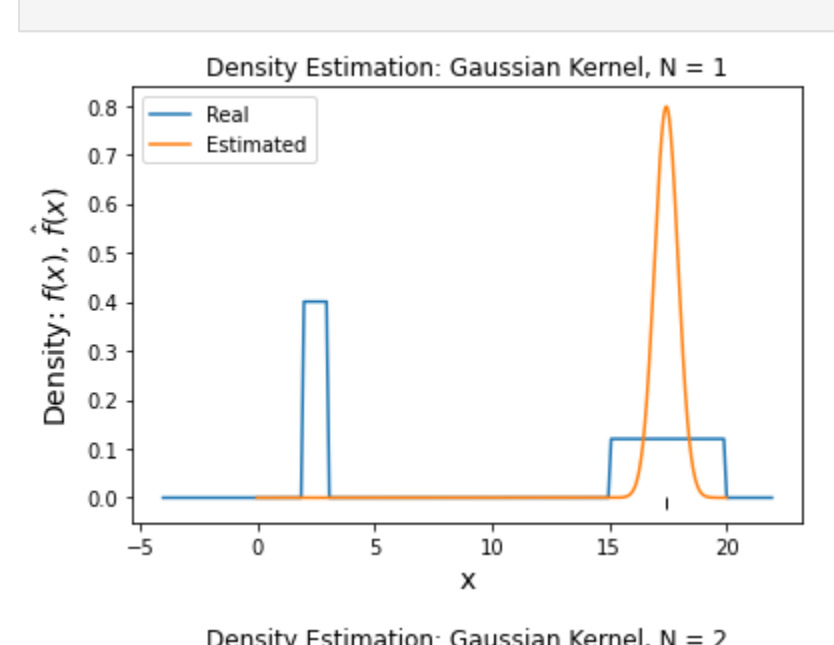

```
Number of discrete, evenly-spaced, x-values used:  10000
Number of N values used:  50000
Number of K points used:  50000
```

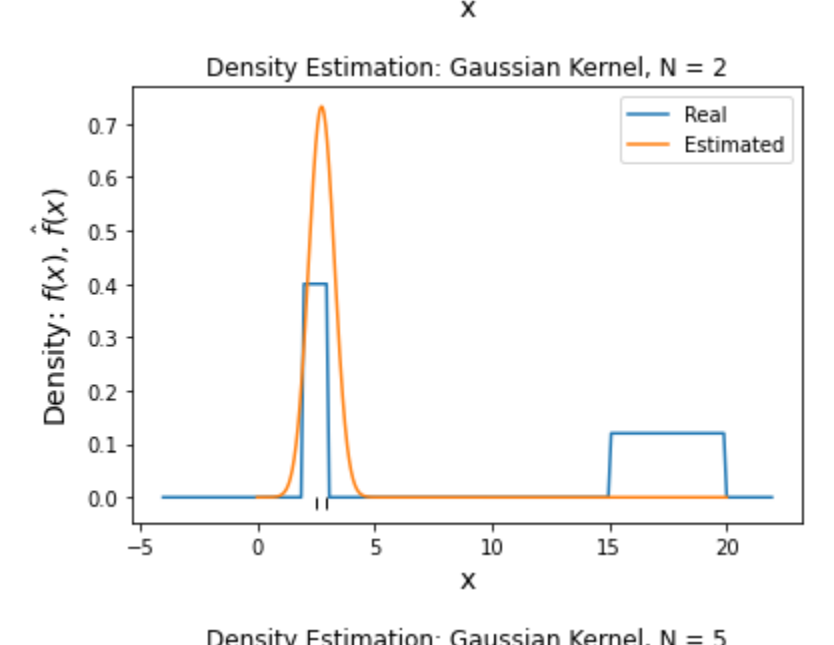


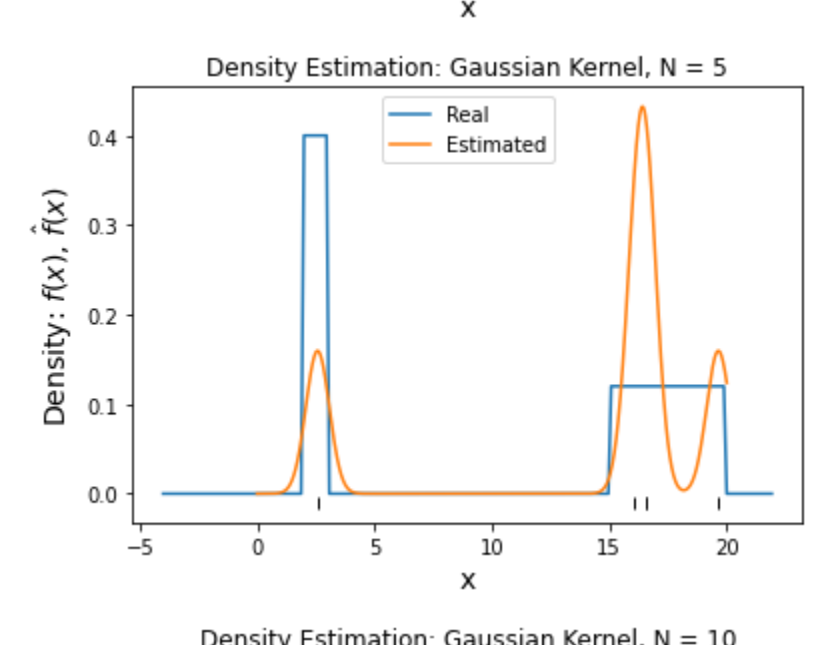
Question 3: Estimating mixture density using Gaussian kernel density estimation

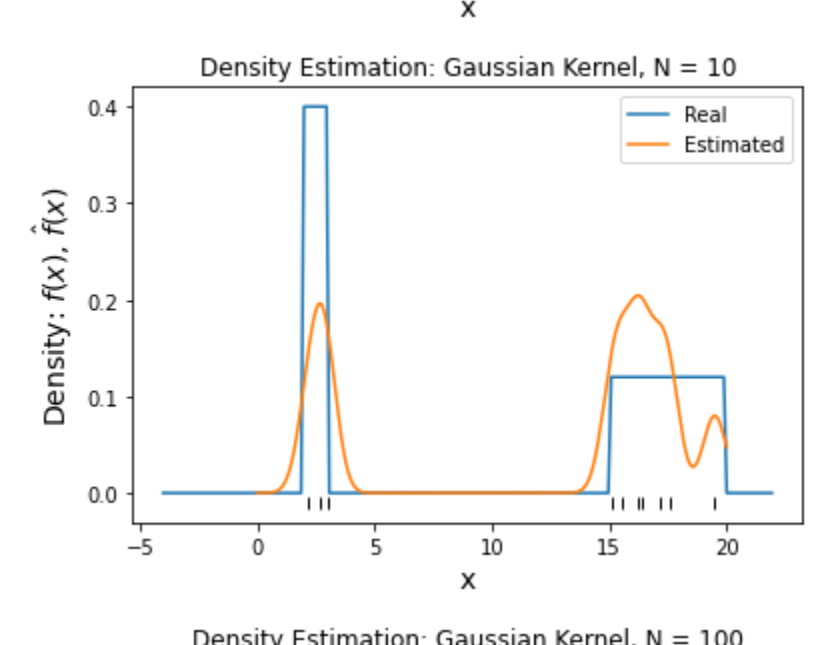
In [4]:

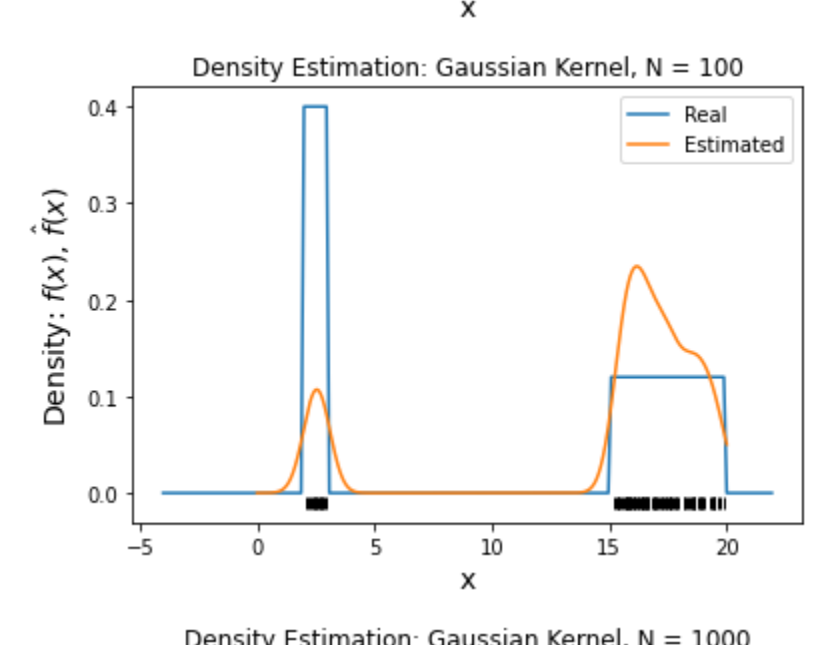
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special

# Function outputing using gaussian normal distribution equation
def KDE_gauss(u):
    return np.exp(-0.5 * u * u) / np.sqrt(2 * np.pi)

# Function to output custom disto for assignment
def customFitPlot(x):
    f1 = (.4 * ((x <= 3 and x >= 2)*(1) + 0))
    f2 = (.6 * ((x <= 20 and x >= 15)*(0.2) + 0))
    a = f1 + f2
    return (a * (a > 0))

# Function that takes in N, number of points used
# to find distibution using kernel density estimation
def KDE(N):
    shape, scale = 2,2
    NPOINTS = 1000
    LAMBDA = 0.5
    x_estimates = np.linspace(0,20, NPOINTS)
    values = np.empty(N * NPOINTS).reshape(N, NPOINTS)

    # Find number of points N, using custom distro
    X_for_plot = generateXTrainCustom(N)

    for i,xq in enumerate(X_for_plot):
        for j, x in enumerate(x_estimates):
            values[i,j] = KDE_gauss((xq - x)/LAMBDA)


    # get the real dist for the custom set
    x_custom_values = np.arange(-4,22,.120)
    y_custom_values = np.full(shape=len(x_custom_values), fill_value=0, dtype=float)
    for i in range(len(x_custom_values)):
        y_custom_values[i] = customFitPlot(x_custom_values[i])

    #plot the values for real distribution
    plt.plot(x_custom_values, y_custom_values, label="Real")

    # Plot the dashed line on X axis for points used
    # to determine distrobution
    plt.plot(X_for_plot, np.full_like(X_for_plot, -0.01), '|k', markeredgewidth=1)

    # Ploting all of the N values for finding real distobution
    plt.plot(x_estimates, np.sum(values, axis=0) / (N * LAMBDA), label="Estimated")

    #Setting Plot labels
    plt.xlabel('x', fontsize=14)
    plt.ylabel('Density: $f(x)$, $\hat{f}(x)$', fontsize=14)
    plt.legend()
    plt.title(f"Density Estimation: Gaussian Kernel, N = {N}")
    plt.show()


rng = np.random.default_rng(seed=101)

KDE(1)
KDE(2)
KDE(5)
KDE(10)
KDE(100)
KDE(1000)
```

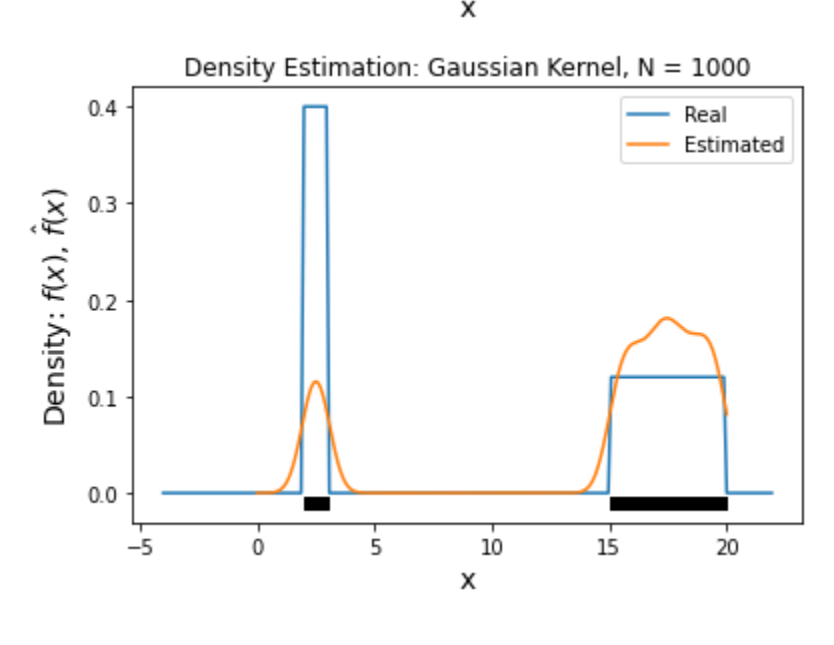












Question 4: Estimating mixture density using histograms

In [5]:

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special

# Function to use a histogram using N samples
# and bins number of bins to find distrobution
# of N points
def hist_method(N, bins):
    shape, scale = 2,1
    NPOINTS = 1000
    LAMBDA = 1

    # Find number of points N, using custom distro
    X_for_plot =  generateXTrainCustom(N)

    # get the real dist for the custom set
    x_custom_values = np.arange(-4,22,.120)
    y_custom_values = np.full(shape=len(x_custom_values), fill_value=0, dtype=float)
    for i in range(len(x_custom_values)):
        y_custom_values[i] = customFitPlot(x_custom_values[i])

    # Ploting all of the N values for finding real distobution
    plt.plot(x_custom_values,y_custom_values)

    # Ploting the histogram for the N points given using
    # number of bins selected
    plt.hist(X_for_plot, color = 'blue', bins = bins, density = True)

    #Setting Plot labels
    plt.xlabel('x', fontsize=14)
    plt.ylabel('Density: $f(x)$, $\hat{f}(x)$', fontsize=14)
    plt.title(f"Density Estimation: Histogram, N = {N} and bin = {bins}")
    plt.show()



hist_method(5,2 )
hist_method(5,5 )
hist_method(5,20 )

hist_method(10,2 )
hist_method(10,5 )
hist_method(10,20 )

hist_method(500,2 )
hist_method(500,5 )
hist_method(500,20 )
```