Data Analytics Lab (2nd Week) - G Prashant (BS17B011)

Probability Distribution, Moments, Visualisation, Data Generation, Parameter Estimation, Hypothesis Testing, Correlation ¶

Basic moments of one-dimensional data and visualisation

Implement functions to compute median, mode, sample mean, sample variance, standard deviation of one-dimensional data "without using numpy".

Example : data1 = np.array([1,2,3,4,5])

data1_median = median(data1)

data1_mode = mode(data1)

data1_mean = mean(data1)

data1_variance = variance(data1)

data1_stddev = stddev(data1)

In [129]:

```
# import dependencies
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sb
import math
import pandas as pd
import random
from scipy import stats
```

In [158]:

```
random.seed(100)
#Generating Integer Dataset in the range 10 to 50
data1 = np.random.choice(np.array(range(10, 50)), 1000)
#Generating Integer Dataset in the range 15 to 100
data2 = np.random.choice(np.array(range(15, 100)), 1000)
#Generating Integer Dataset - Normal Distibuted with mean 20 and std 15
data3 = np.random.normal(20,15,1000).astype(int)
#Generating Integer Dataset - Normal Distibuted with mean 100 and std 20
data4 = np.random.normal(100,20,1000).astype(int)
```

Compute the above for different one-dimensional datasets - data1.txt, data2.txt, data3.txt, data4.txt and store as data1_median, data2_median etc. Verify the results using the numpy built in functions for mean,stddev and variance.

In [159]:

```
# mean
def mean(arr):
    n = len(arr)
    mean = sum(arr)/n
    return mean

# median
def median(arr):
    n = len(arr)
    arr = np.sort(arr)
    if n%2==0:
        median = (arr[n//2-1]+arr[n//2])/2
    else:
        median = arr[n//2]
    return median

# mode - finds more than 1 mode if exists
def mode(arr):
    n = len(arr)
    count = {}
    modes = []
    for ele in arr:
        count[ele] = sum(arr==ele)
    for ind in count:
        if count[ind]==max(count.values()):
            modes.append(ind)
    return modes

# variance
def variance(arr):
    n = len(arr)
    mu = mean(arr)
    var = 0
    for i in range(n):
        var += (arr[i]-mu)**2/n
    return var

#stddev
def stddev(arr):
    var = variance(arr)
    std = math.sqrt(var)
    return std
```

In [160]:

```
#calculating median, mode, mean, variance and standard deviations without using numpy f
unctions
#Data1
print("Data1\n")
data1_median = median(data1)
print("Median = "+ str(data1_median))

data1_mode = mode(data1)
print("Mode = "+ str(data1_mode))

data1_mean = mean(data1)
print("Mean = "+ str(data1_mean))

data1_variance = variance(data1)
print("Variance = "+ str(data1_variance))

data1_stddev = stddev(data1)
print("Standard Deviation = "+ str(data1_stddev))

#Data2
print("\nData2\n")
data2_median = median(data2)
print("Median = "+ str(data2_median))

data2_mode = mode(data2)
print("Mode = "+ str(data2_mode))

data2_mean = mean(data2)
print("Mean = "+ str(data2_mean))

data2_variance = variance(data2)
print("Variance = "+ str(data2_variance))

data2_stddev = stddev(data2)
print("Standard Deviation = "+ str(data2_stddev))

#Data3
print("\nData3\n")
data3_median = median(data3)
print("Median = "+ str(data3_median))

data3_mode = mode(data3)
print("Mode = "+ str(data3_mode))

data3_mean = mean(data3)
print("Mean = "+ str(data3_mean))

data3_variance = variance(data3)
print("Variance = "+ str(data3_variance))

data3_stddev = stddev(data3)
print("Standard Deviation = "+ str(data3_stddev))

#Data4
print("\nData4\n")
data4_median = median(data4)
print("Median = "+ str(data4_median))

data4_mode = mode(data4)
```

```
print("Mode = "+ str(data4_mode))

data4_mean = mean(data4)
print("Mean = "+ str(data4_mean))

data4_variance = variance(data4)
print("Variance = "+ str(data4_variance))

data4_stddev = stddev(data4)
print("Standard Deviation = "+ str(data4_stddev))
```

```
Data1

Median = 30.0
Mode = [25]
Mean = 29.852
Variance = 128.0960959999998
Standard Deviation = 11.31795458552471

Data2

Median = 58.0
Mode = [99, 38, 51, 85]
Mean = 57.677
Variance = 609.7146710000019
Standard Deviation = 24.69240107806452

Data3

Median = 20.0
Mode = [20]
Mean = 19.95
Variance = 232.67349999999968
Standard Deviation = 15.253638910109276

Data4

Median = 98.0
Mode = [94, 96]
Mean = 98.62
Variance = 402.65759999999955
Standard Deviation = 20.066330008250127
```

In [161]:

```
#Verification using numpy functions
#Data1
print("Data1\n")
np_median_1 = np.median(data1)
print("numpy Median = "+str(np_median_1))
np_mean_1 = np.mean(data1)
print("numpy Mean = "+str(np_mean_1))
np_var_1 = np.var(data1)
print("numpy Variance = "+str(np_var_1))
np_std_1 = np.std(data1)
print("numpy Standard Deviation = "+str(np_std_1))

#Data2
print("\nData2\n")
np_median_2 = np.median(data2)
print("numpy Median = "+str(np_median_2))
np_mean_2 = np.mean(data2)
print("numpy Mean = "+str(np_mean_2))
np_var_2 = np.var(data2)
print("numpy Variance = "+str(np_var_2))
np_std_2 = np.std(data2)
print("numpy Standard Deviation = "+str(np_std_2))

#Data3
print("\nData3\n")
np_median_3 = np.median(data3)
print("numpy Median = "+str(np_median_3))
np_mean_3 = np.mean(data3)
print("numpy Mean = "+str(np_mean_3))
np_var_3 = np.var(data3)
print("numpy Variance = "+str(np_var_3))
np_std_3 = np.std(data3)
print("numpy Standard Deviation = "+str(np_std_3))

#Data4
print("\nData4\n")
np_median_4 = np.median(data4)
print("numpy Median = "+str(np_median_4))
np_mean_4 = np.mean(data4)
print("numpy Mean = "+str(np_mean_4))
np_var_4 = np.var(data4)
print("numpy Variance = "+str(np_var_4))
np_std_4 = np.std(data4)
print("numpy Standard Deviation = "+str(np_std_4))
```

```
Data1

numpy Median = 30.0
numpy Mean = 29.852
numpy Variance = 128.09609600000002
numpy Standard Deviation = 11.317954585524719

Data2

numpy Median = 58.0
numpy Mean = 57.677
numpy Variance = 609.7146710000001
numpy Standard Deviation = 24.692401078064485

Data3

numpy Median = 20.0
numpy Mean = 19.95
numpy Variance = 232.6735
numpy Standard Deviation = 15.253638910109286

Data4

numpy Median = 98.0
numpy Mean = 98.62
numpy Variance = 402.6576
numpy Standard Deviation = 20.066330008250137
```

Hence, it can be noted that the values calculated with numpy functions matched with those that were calculated through appropriate formula.

Visualize all the datasets (1-4) separately using matplotlib scatter plot and histograms.

1. Comment on initial observations of distributions.
2. For each dataset, specify which descriptive statistics best describe the data.

Example :

data1 - mean

data2 - mean

data3 - mean

data4 - mean

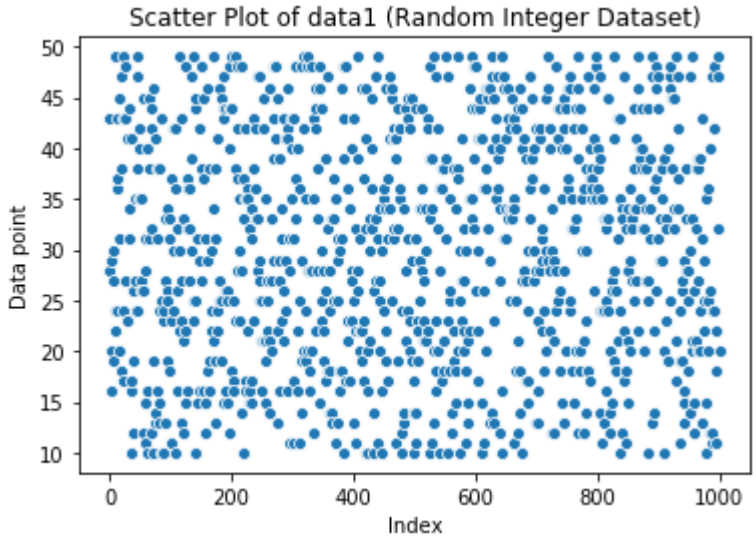
Hints: If you want to plot a known distribution (Normal,Unifrom, Exponential etc) using matplotlib, define it as a function and plot it. For this, you need to know the formula of the chosen distribution. If you dont know the exact distribution of a dataset, plot it as a histogram and smooth the data by converting bin edges to centres. Using seaborn library: Install seaborn library and use `distplot()` function to plot the distribution and verify the result.

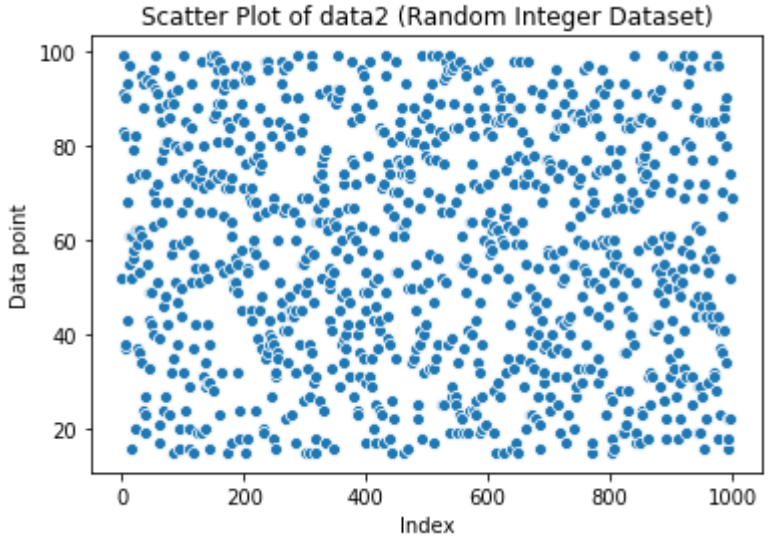
In [162]:

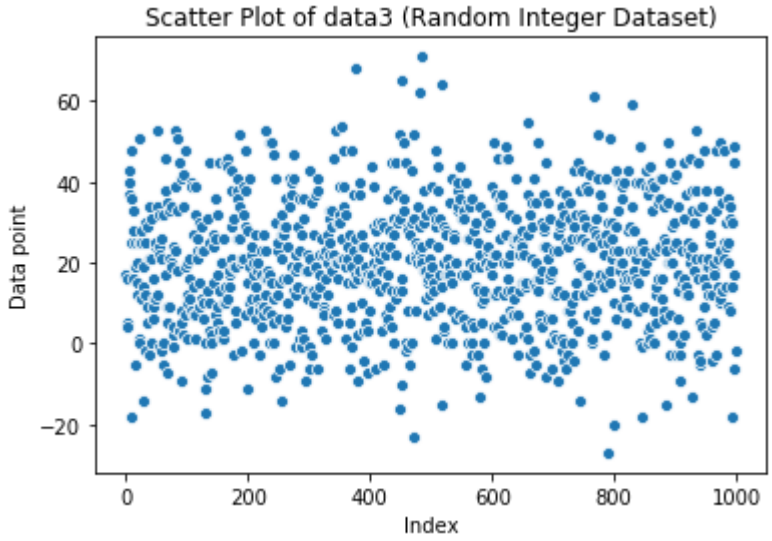
```
# visualization by scatter plot
plt.figure()
sb.scatterplot([i for i in range(len(data1))],data1)
plt.xlabel("Index")
plt.ylabel("Data point")
plt.title("Scatter Plot of data1 (Random Integer Dataset)")

plt.figure()
sb.scatterplot([i for i in range(len(data2))],data2)
plt.xlabel("Index")
plt.ylabel("Data point")
plt.title("Scatter Plot of data2 (Random Integer Dataset)")

plt.figure()
sb.scatterplot([i for i in range(len(data3))],data3)
plt.xlabel("Index")
plt.ylabel("Data point")
plt.title("Scatter Plot of data3 (Random Integer Dataset)")

plt.figure()
sb.scatterplot([i for i in range(len(data4))],data4)
plt.xlabel("Index")
plt.ylabel("Data point")
plt.title("Scatter Plot of data4 (Random Integer Dataset)")
```

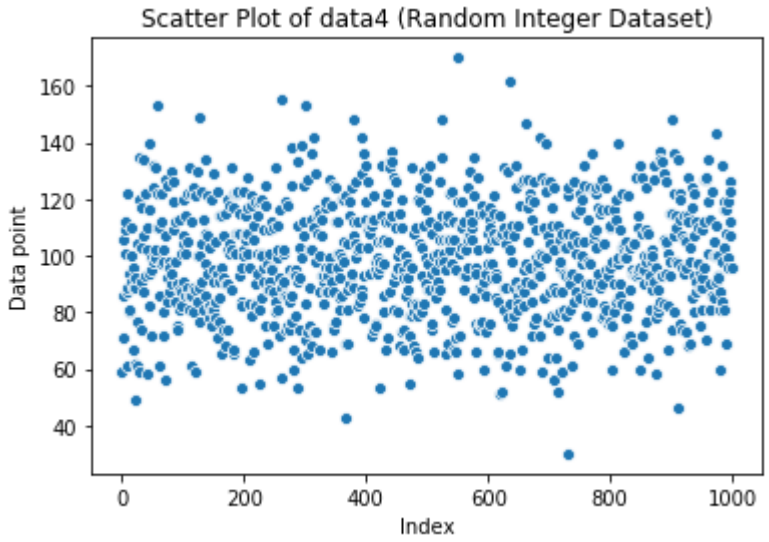
Out[162]:

```
Text(0.5, 1.0, 'Scatter Plot of data4 (Random Integer Dataset)')
```









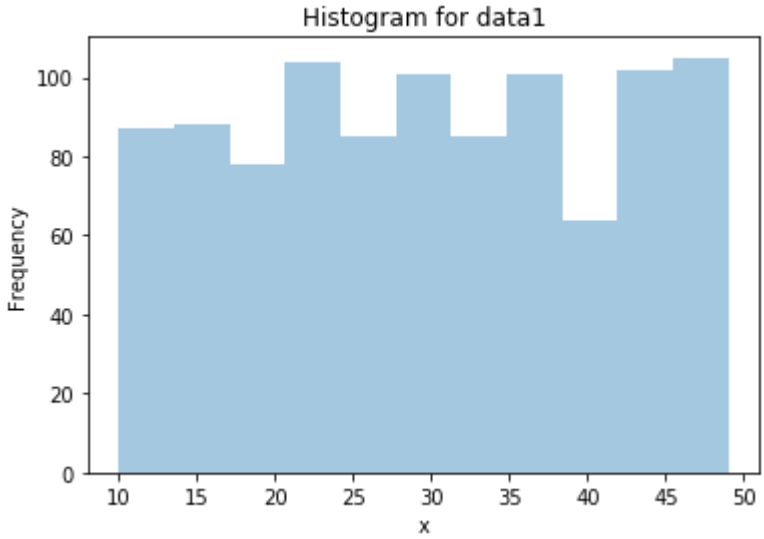
The scatter plots for uniformly distributed integer data (data1 and data2) are uniformly spread across the space within the given range, while in the scatter plots for normal distributed integer data (data3 and data4), most of the points lie near the mean

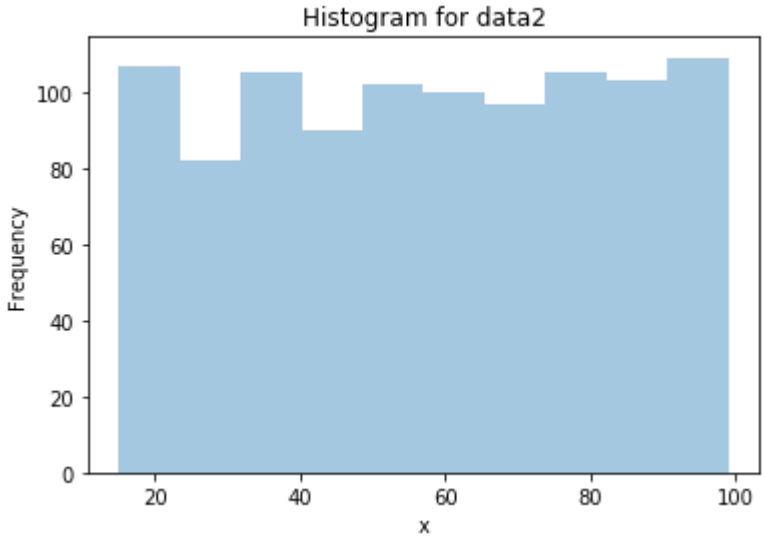
In [163]:

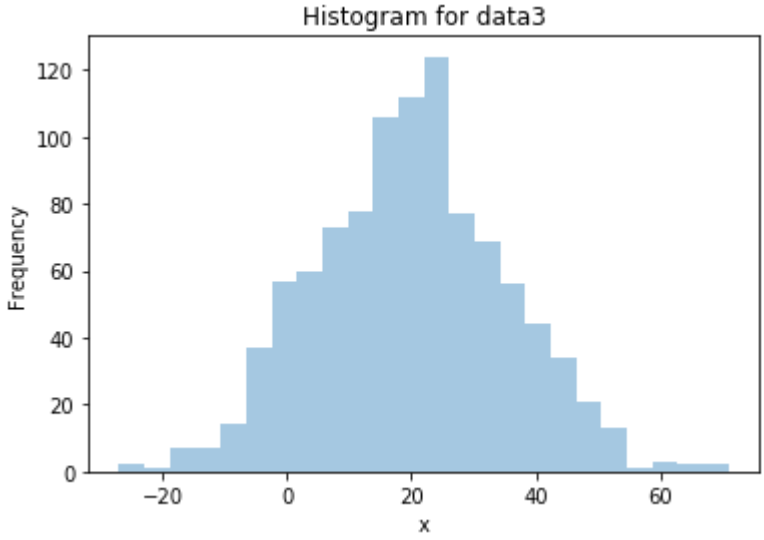
```
# histogram plot
plt.figure()
sb.distplot(data1,kde = False)
plt.title("Histogram for data1")
plt.xlabel("x")
plt.ylabel("Frequency")

plt.figure()
sb.distplot(data2,kde = False)
plt.title("Histogram for data2")
plt.xlabel("x")
plt.ylabel("Frequency")

plt.figure()
sb.distplot(data3,kde = False)
plt.title("Histogram for data3")
plt.xlabel("x")
plt.ylabel("Frequency")

plt.figure()
sb.distplot(data4,kde = False)
plt.title("Histogram for data4")
plt.xlabel("x")

plt.ylabel("Frequency")
```

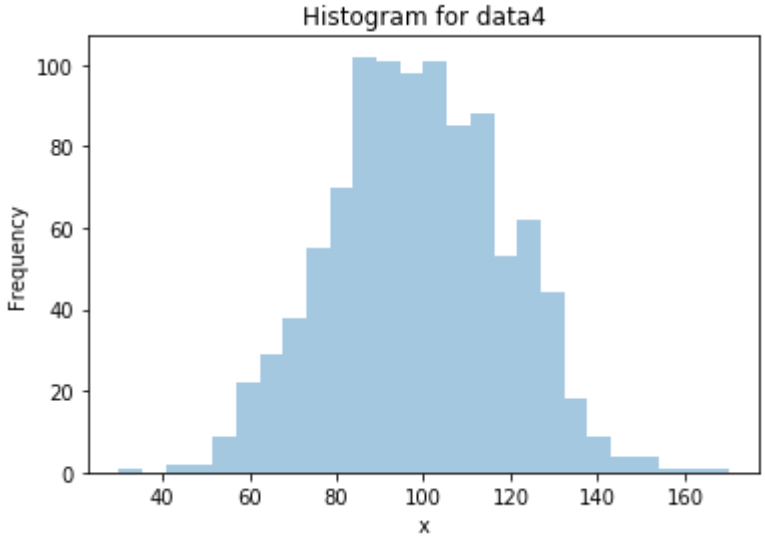
Out[163]:

```
Text(0, 0.5, 'Frequency')
```









In the plots shown above, histograms for data1 and data2 appear to be uniform and histograms for data3 and data4 appear to be a bell shaped Gaussian curve. As all the data (data1 to data4) were either sampled from uniform or Gaussian distributions, the best descriptive statistic that describes these datasets is the Mean

Probability Distribution parameters and visualisation

Uniform Distribution: A uniform distribution, also called a rectangular distribution, is a probability distribution that has constant probability. This distribution is defined by two parameters, a and b: a is the minimum and b is the maximum.

Formula:

$$f(x) = \frac{1}{b-a}, a \leq x \leq b$$

Parameters: $\text{Mean} = \frac{(a+b)}{2}; \text{Variance} = \frac{1}{12(b-a)^2}$

Gaussian Distribution: The normal distribution is a probability function that describes how the values of a variable are distributed. It is a symmetric distribution where most of the observations cluster around the central peak and the probabilities for values further away from the mean taper off equally in both directions. For mean, μ and standard deviation, σ, the formula is given as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Use the random.normal() method to get a Normal Data Distribution.

It has three parameters:

loc - (Mean) where the peak of the bell exists.

scale - (Standard Deviation) how flat the graph distribution should be.

size - The shape of the returned array.

Given a normal distribution of mean = 3 and standard deviation = 1.

1. Find $P(x \leq 3)$ from the given distribution.
2. Find the probability that $P(3 < x \leq 5)$ drawn from the given distribution. Write a function to estimate the above cumulative distribution function of a normal distribution. Verify the same.

In [164]:

```
# 1. Find  p(x<3)  from the given distribution
cdf_3 = 0.5 # As it is the mean
print("Theoretical value of p(x<=3) = "+str(cdf_3))

# 2. Find the probability that p(3<x<=5)  drawn from the given distribution
cdf_3_5 = 0.9545/2 +0.5-0.5    #As 5 is 2 standard deviations away from the mean (95.4
5% of data)
#cdf_3_5 = 0.5*(1+math.erf((5-3)/(math.sqrt(2)))) - cdf_3

print("Theoretical value of p(3<x<=5) = "+str(cdf_3_5))

# Function for estimating cdf
def estimateCDF(x, a, b):
    """
    PARAMETERS:
    x : input 1D data
    a, b: data points at which cdf is to be estimated

    RETURNS:
    cdf : cumulative distibution values s.t. P(a<x<=b)
    """
    n = len(x)
    ocurr = np.sum((x<=b)*(x>a))
    cdf = ocurr/n
    return cdf

#Generating random normal data with mean 3 and std 1
x = np.random.normal(3,1,100000)

#Estimating CDF
emp_3 = estimateCDF(x,-np.inf,3)
emp_3_5 = estimateCDF(x,3,5)

print("Empirical value of p(x<=3) = "+str(emp_3))
print("Empirical value of p(3<x<=5) = "+str(emp_3_5))
```

```
Theoretical value of p(x<=3) = 0.5
Theoretical value of p(3<x<=5) = 0.47724999999999995
Empirical value of p(x<=3) = 0.50128
Empirical value of p(3<x<=5) = 0.4761
```

It can be noted that the theoretical and the empirical values of the CDFs are almost equal

Maximum Likelihood Estimation for normal distribution :

A method of estimating the parameters of a distribution by maximizing a likelihood function, so that under the assumed statistical model the observed data is most probable. For normal distribution, $f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$, the goal is to determine μ and σ for our data so that we can match our data to its most likely Gaussian bell curve. The estimated mean, $\hat{\mu}$ for normal distribution is $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and the estimated standard deviation is $\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$.

Assuming that data1 has been sampled from normal distribution with unknown mean and standard deviation of 2, calculate 95% confidence interval of data1_mean.

In [13]:

```
#Random unknown mean
rand_mean = 10*np.random.random()

#Randomly Generating data with an unknown mean and std=2
data1 = np.random.normal(rand_mean,2,1000)
n = len(data1)
mu = mean(data1)
sig = stddev(data1)

#Estimating 95% confidence interval, where z = 1.96
conf = 1.96*sig/math.sqrt(n)

print("Sample mean = "+str(mu))
print("95% interval: (-1.96,1.96)")
print("Hence, the 95% confidence interval of the mean is "+str(-conf)+" to "+str(+conf
))
```

```
Sample mean = 6.40451230948348
95% interval: (-1.96,1.96)
Hence, the 95% confidence interval of the mean is -0.12243127465179618 to
0.12243127465179618
```

In other words, the population mean lies between (6.4045 - 0.1254) to (6.4045 + 0.1254)

Parameter estimation

In this section you will fit data to a standard distribution.

Normal distribution "DataSet1.txt" is to be fitted to normal distribution. Complete the function fitNormal() to estimate parameters.

In [15]:

```
#Using MLE to fit parameters of Normal Distribution
def fitNormal(x):
    """
    PARAMETERS:
    x : input 1D data

    RETURNS:
    mu : mean of distribution
    sig : standard deviation of distribution
    """
    n = len(x)
    mu = sum(x)/n
    sig = 0
    for ele in x:
        sig += 1/n*(ele-mu)**2
    sig = sig**0.5
    return [mu, sig]
```

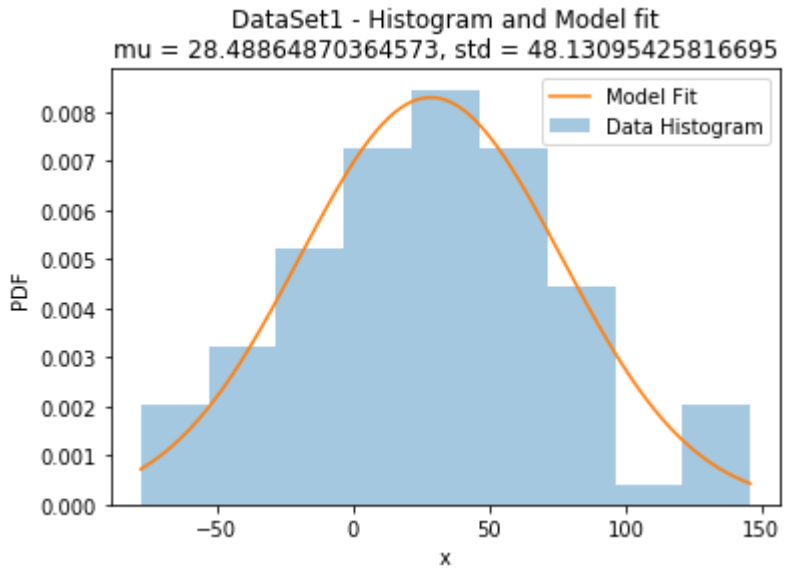
Read the file "DataSet1.txt" into a variable `data1`. Also plot a histogram of the same. Then pass it to `fitNormal()` to estimate parameters. Generate your own data using estimated parameters and plot the distribution. Use `numpy.linspace()` to generate 1000 points in the range of `data1`. The probabilities are estimated from Gaussian distribution formula. Make a lineplot in the same figure.

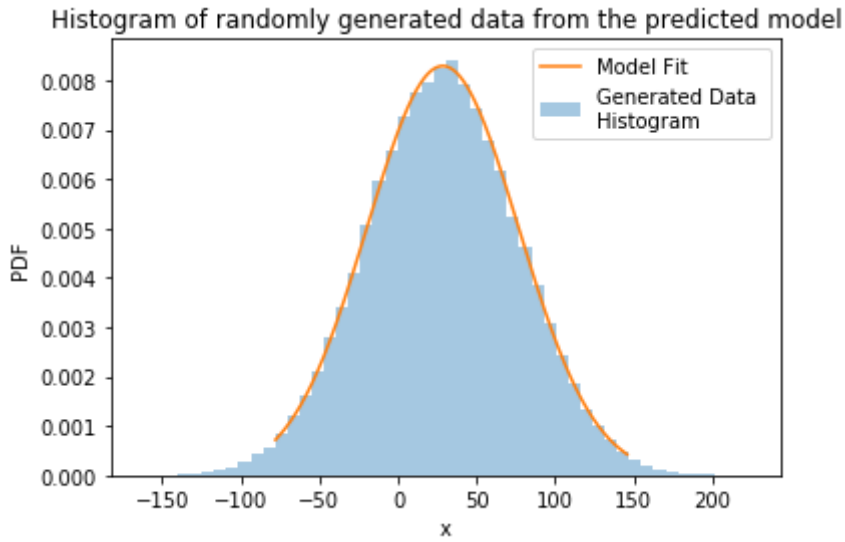
In [169]:

```
# Read datafile
data1 = np.genfromtxt('DataSet1.txt')

# Plot histogram of input data
#plt.hist(data1)
sb.distplot(data1,kde=False,norm_hist = True)

# Fit data to normal distribution
mu_mle,sig_mle = fitNormal(data1)

# Using linspace to generate 1000 points in the range of data1
x = np.linspace(min(data1),max(data1),1000)
p = (1/(sig_mle*(math.sqrt(2*math.pi))))*np.exp((-(x - mu_mle)**2)/(2*sig_mle**2))
plt.plot(x,p)
plt.ylabel("PDF")
plt.xlabel("x")
plt.title("DataSet1 - Histogram and Model fit\nmu = "+str(mu_mle)+", std = "+str(sig_ml
e))
plt.legend(["Model Fit","Data Histogram"])

#randomly generating 100000 data points from the Gaussian fit model
sample_1 = np.random.normal(mu_mle,sig_mle,100000)
plt.figure()
sb.distplot(sample_1,kde=False,norm_hist = True)
plt.plot(x,p)
plt.ylabel("PDF")
plt.xlabel("x")
plt.title("Histogram of randomly generated data from the predicted model")
plt.legend(["Model Fit","Generated Data \nHistogram"])
```

Out[169]:

```
<matplotlib.legend.Legend at 0x2a41ba47b08>
```





As shown in the above plots, the distribution of the Gaussian fit model closely resembles the the distribution of the data

"DataSet2.txt" is to be fitted to uniform distribution. Complete the function fitUniform() to estimate parameters. Generate and plot data as before.

In [170]:

```
#Using MLE to fit parameters of Uniform Distribution
def fitUniform(x):
    """
    PARAMETERS:
    x : input 1D data

    RETURNS:
    a : minimum value of distribution
    b : maximum value of distribution
    """
    a = min(x)
    b = max(x)
    return [a, b]
```

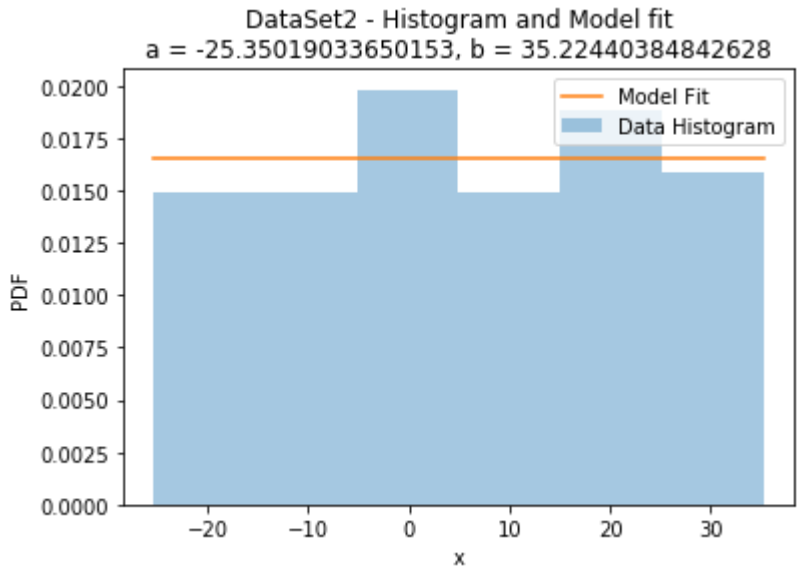
Read the file "DataSet2.txt" into a variable "data2". Also plot a histogram of the same. Then pass it to `fitUniform()` to estimate parameters. Generate your own data using estimated parameters and plot the distribution.

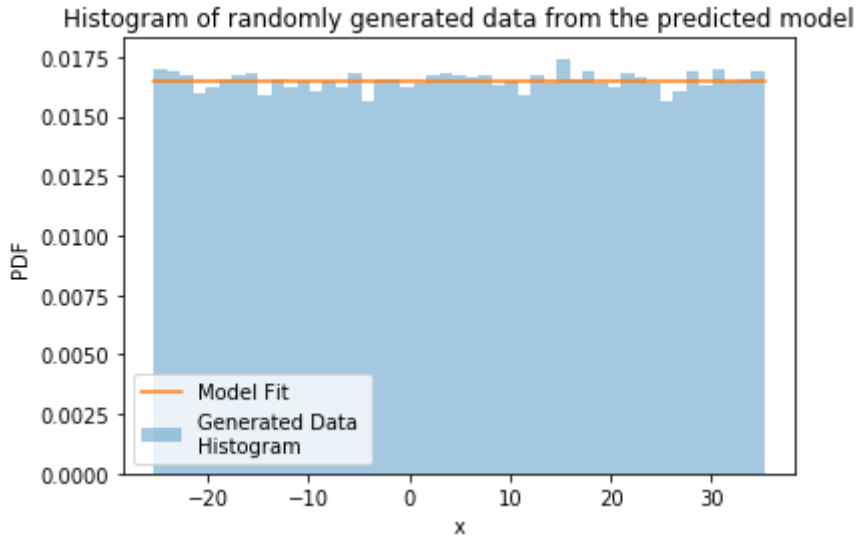
In [171]:

```
# Read datafile
data2 = np.genfromtxt('DataSet2.txt')

# Plot histogram of input data
#plt.hist(data2)
sb.distplot(data2,kde = False,norm_hist = True)


# Fit data to uniform distribution
a,b = fitUniform(data2)

# Generate data using estimated parameters for visualization and plot the same
x_u = np.linspace(min(data2),max(data2),1000)
p_u = np.array([1/(b-a)]*1000)
plt.plot(x_u,p_u)
plt.ylabel("PDF")
plt.xlabel("x")
plt.title("DataSet2 - Histogram and Model fit\na = "+str(a)+", b = "+str(b))
plt.legend(["Model Fit","Data Histogram"])

sample_2 = np.random.uniform(a,b,100000)
plt.figure()
sb.distplot(sample_2,kde=False,norm_hist = True)
plt.plot(x_u,p_u)
plt.ylabel("PDF")
plt.xlabel("x")
plt.title("Histogram of randomly generated data from the predicted model")
plt.legend(["Model Fit","Generated Data \nHistogram"])
```

Out[171]:

```
<matplotlib.legend.Legend at 0x2a4179b9d88>
```





As shown in the above plots, the distribution of the Uniform fit model resembles the the distribution of the data to some extent

Correlation

In this section you are required to measure correlation between two variables. The dataset "**AirQualityData.csv**" contains daily readings of PM_{10} and O_3 concentration in air along with temperature readings. To verify if the variation concentration of any of these pollutants is associated with temperature changes, measure the correlation between temperature and each of these pollutants. Complete the function `correlationCoeff()` below that measures the correlation coefficient between two given time-series data. For a given n two data sets x and y, the Pearson coefficient formula is given as

$$r_{xy} = \frac{cov(X,Y)}{\sigma(X) \cdot \sigma(Y)}$$

$cov(X,Y) = \frac{\Sigma_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1}$; $\sigma(X), \sigma(Y)$ are respective standard deviations

In [76]:

```
#Function to calculate pearson correlation coefficient of two variables X and Y
def correlationCoeff(x, y):
    """
    PARAMETERS:
    x : input 1D data
    y : input 1D data

    RETURNS:
    est : coefficient value
    """
    assert len(x)==len(y)
    sigma_x = 0
    sigma_y = 0
    mean_x = sum(x)/len(x)
    mean_y = sum(y)/len(y)
    cov = 0
    n = len(x)
    for i in range(n):
        sigma_x+=((x[i]-mean_x)**2)/n
        sigma_y+=((y[i]-mean_y)**2)/n
        cov+=(x[i]-mean_x)*(y[i]-mean_y)/(n-1)
    sigma_x = math.sqrt(sigma_x)
    sigma_y = math.sqrt(sigma_y)
    est = cov/(sigma_x*sigma_y)
    return est
```

Read the file "AirQualityData.csv" . Ignoring the "date" column, read columns "temp", "pm10" and "o3" as the dataframe as pass them to the function `correlationCoeff()` to estimate seperately the effect of "pm10" on "temp" and "o3" on "temp". Which of the pollutants has a greater impact on temperature? Compare the value that you get from the above function with the python built in function `df.corr()` .

In [77]:

```
# Read datafile
AirQualityData = pd.read_csv('AirQualityData.csv')

# Estimate correlation
corr_temp_pm10 = correlationCoeff(AirQualityData["temp"],AirQualityData["pm10"])
corr_temp_o3 = correlationCoeff(AirQualityData["temp"],AirQualityData["o3"])
print("Calculated Correlation Coefficient of temperature and pm10 = "+str(corr_temp_pm1
0))
print("Calculated Correlation Coefficient of temperature and ozone = "+str(corr_temp_o3
))

#Using df.corr function to calculate correlation matrix
print(AirQualityData.corr("pearson"))
```

```
Calculated Correlation Coefficient of temperature and pm10 = 0.36248659213
98137
Calculated Correlation Coefficient of temperature and ozone = 0.5703904221
495666
          temp      pm10        o3
temp  1.000000  0.362391  0.570239
pm10  0.362391  1.000000  0.333296
o3    0.570239  0.333296  1.000000
```

It can be noted that the calculated pearson correlation coefficients match with the those determined using pandas in built function. Also, temperature is more correlated with O3 than with pm10. Hence, O3 has a greater effect on temperature

Hypothesis Testing and Confidence Interval

P Value: A p-value for a statistical model is the probability that when the null hypothesis is true, the statistical summary is equal to or greater than the actual observed results. This is also termed ‘probability value’ or ‘asymptotic significance’. The null hypothesis states that two measured phenomena experience no relationship to each other. We denote this as H or H0. If one or more of these probabilities turn out to be less than or equal to α, the level of significance, we reject the null hypothesis.

Example: One such null hypothesis can be that the number of hours spent in the office affects the amount of salary paid. For a significance level of 5%, if the p-value falls lower than 5%, the null hypothesis is invalidated. Then it is discovered that the number of hours you spend in your office will not affect the amount of salary you will take home.

T Test: Such a test tells us whether a sample of numeric data strays or differs significantly from the population. It also talks about two samples- whether they’re different. In other words, it gives us the probability of difference between populations.

In [172]:

```
# Install scipy library
!pip install scipy
# import scipy.stats
from scipy import stats
# Use the function stats.ttest_ind() for DataSet1.txt and DataSet 2.txt
print("\n",stats.ttest_ind(data1,data2))
```

```
Requirement already satisfied: scipy in c:\users\dell\anaconda3\lib\site-p
ackages (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in c:\users\dell\anaconda3\li
b\site-packages (from scipy) (1.18.1)

 Ttest_indResult(statistic=4.4917061119380906, pvalue=1.1993223197767313e-
05)
```

As the p-value is 1.99e-05, which is less than 0.05, the two data samples do not come from the same distribution

In [106]:

```
#Example
#Generate 10,000 random data from a normal distribution of mean 0 and standard 1 with a
random seed value of 10.
np.random.seed(10)
x = np.random.normal(0,1,10000)
# Use KS test to compare the generated data to a normal distribution using kstest
print(stats.kstest(x,"norm"))
#if the p-value is less than 0.05 or 5% (for a 95% confidence level), then the null hyp
othesis is rejected,
#which means the generated data is not from the normal distribution
```

Out[106]:

```
KstestResult(statistic=0.008418461238842267, pvalue=0.4777893822483902)
```

As the p-value is 0.4778, which is greater than 0.05, the null hypothesis is valid, and x significantly resembles a normal distribution

In [24]:

```
#Example
# Generate 10,000 random data from a poisson distribution of lambda = 5
poiss_5 = np.random.poisson(5,10000)
# Generate 7,000 random data from a poisson distribution of lambda = 7
poiss_7 = np.random.poisson(7,7000)
# compare the mean of the two data sets using t test with a confidence level of 95%.
print(stats.ttest_ind(poiss_5,poiss_7))
```

Out[24]:

```
Ttest_indResult(statistic=-53.23089104317379, pvalue=0.0)
```

As the p-value is ≈ 0, which is less than 0.05, the two samples do not come from the same distribution

In [ ]:

```
```