Generate a list of 100 integers containing values between 90 to 130 and store it in the variable "int_list".

```
import random
```

```
int_list= []
for i in range(100):
  int_list.append(random.randint(90,130))
print(int_list)
```

```
[113, 125, 130, 129, 96, 115, 110, 128, 119, 100, 105, 108, 99, 103, 95, 129, 112, 130, 93, 111, 129, 94, 101, 121, 124, 129, 96, 10
```

```
# Calculate the Mean and Median
```

```
import statistics
```

```
mean_val=statistics.mean(int_list)
print("The Mean value is:",mean_val)

median_val=statistics.median(int_list)
print("The Median value is:",median_val)
```

```
The Mean value is: 109.84
The Median value is: 110.0
```

```
# Calculate the Mode
```

```
mode_val=statistics.mode(int_list)
print("The Mode value is:",mode_val)
```

```
The Mode value is: 96
```

```
# Implement a function to calculate weighted mean
```

```
def weighted_mean(data, weights):
  weighted_sum = 0
  total_weight = 0
  for i in range(len(data)):
    weighted_sum += data[i] * weights[i]
    total_weight += weights[i]
  return weighted_sum / total_weight
```

```
weights=[random.randint(1,10) for i in range(100)]
weighted_mean_val=weighted_mean(int_list,weights)
print("The Weighted Mean value is:",weighted_mean_val)
```

```
The Weighted Mean value is: 109.69947275922671
```

```
# Implement a function to calculate Geometric Mean
```

```
import math
```

```
def geometric_mean(number):
  product = 1
  for num in number:
    product *= num
  return math.sqrt(product)
```

```
geometric_mean_val=geometric_mean(int_list)
print("The Geometric Mean value is:",geometric_mean_val)
```

```
The Geometric Mean value is: 8.17407316233081e+101
```

```
# Implement a function to calculate Harmonic Mean
```

```
def harmonic_mean(number):
  sum = 0
  for num in number:
    sum += 1 / num
  return len(number) / sum
```

```
harmonic_mean_val=harmonic_mean(int_list)
print("The Harmonic Mean value is:",harmonic_mean_val)
```

```
The Harmonic Mean value is: 108.57545713592394
```

```
# Implement a function to determine the midrange of a list of numbers
```

```
def mid_range(numbers):
  numbers.sort()
  return (numbers[len(numbers) // 2] + numbers[len(numbers) // 2 - 1]) / 2
```

```
midrange_val=mid_range(int_list)
print("The Mid_range value is:",midrange_val)
```

```
The Mid_range value is: 110.0
```

```
# Implement a Python program to find the trimmed mean of a list
```

```
def trimmed_mean(numbers, percentage):
  numbers.sort()
  trim_size = int(len(numbers) * percentage / 100)
  return sum(numbers[trim_size:-trim_size]) / (len(numbers) - 2 * trim_size)
```

```
trimmed_mean_val=trimmed_mean(int_list,10)
print("The Trimmed Mean value is:",trimmed_mean_val)
```

```
The Trimmed Mean value is: 109.7
```

Generate a list of 500 integers containing values between 200 to 300 and store it in the variable "int_list2".

```
import random
```

```
int_list2= []
for i in range(500):
  int_list2.append(random.randint(200,300))
print(int_list2)
```

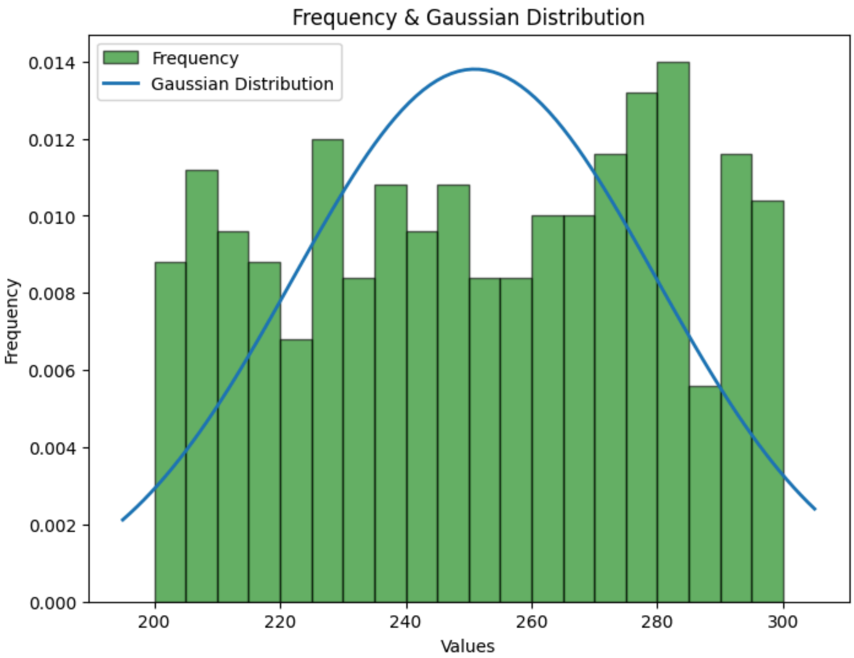
```
[278, 275, 245, 232, 243, 215, 209, 275, 219, 293, 248, 247, 275, 217, 231, 258, 229, 220, 264, 261, 242, 202, 278, 293, 253, 282, 2
```

```
# Compare the given list of visualization for the given data:
```

```
#Frequency & Gaussian distribution
```

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
```

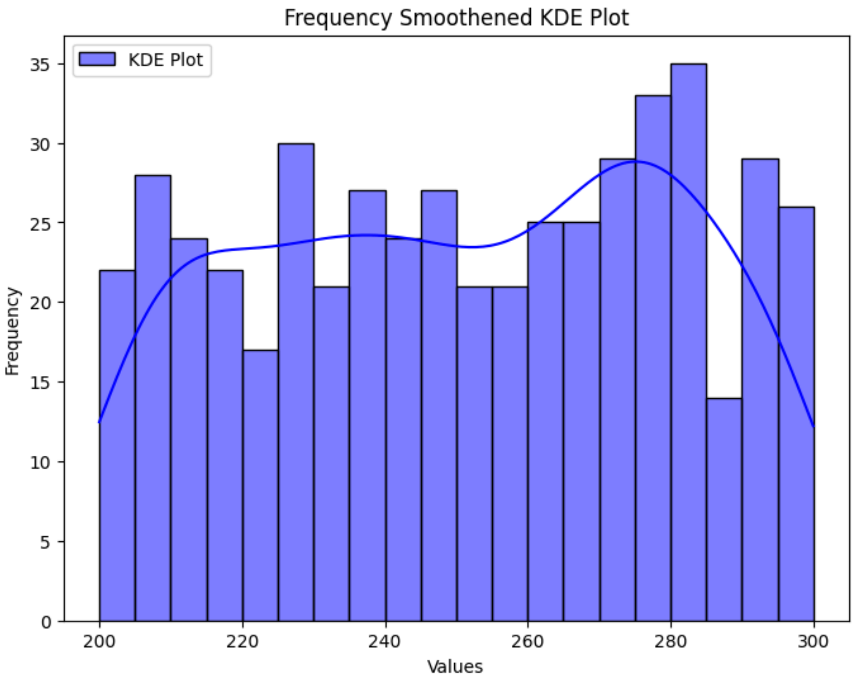
```
plt.figure(figsize=(8, 6))
plt.hist(int_list2, bins=20,density=True,alpha=0.6,color="green", edgecolor='black',label="Frequency")

mean,std=np.mean(int_list2),np.std(int_list2)
xmin,xmax=plt.xlim()
x=np.linspace(xmin,xmax,100)
p=stats.norm.pdf(x,mean,std)

plt.plot(x,p,linewidth=2,label="Gaussian Distribution")
plt.title("Frequency & Gaussian Distribution")
plt.xlabel("Values")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```




```
# Frequency smoothened KDE plot
```

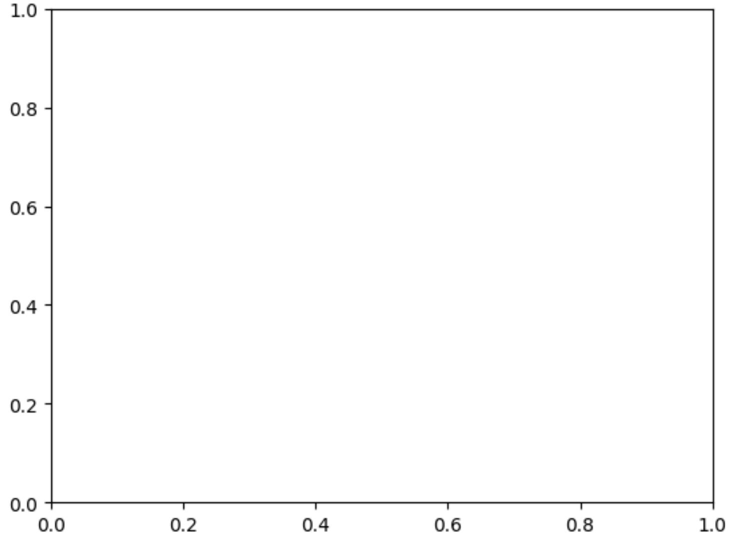
```
import seaborn as sns
```

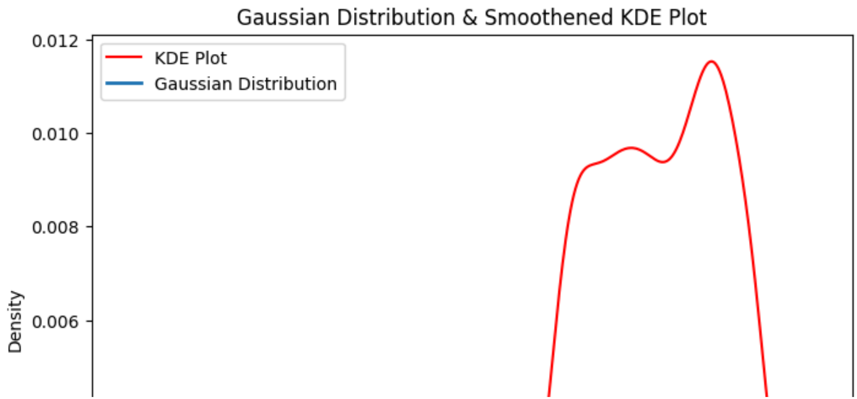
```
plt.figure(figsize=(8, 6))
sns.histplot(int_list2, bins=20, kde=True, color="blue",edgecolor="black", label="KDE Plot")
plt.title("Frequency Smoothened KDE Plot")
plt.xlabel("Values")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

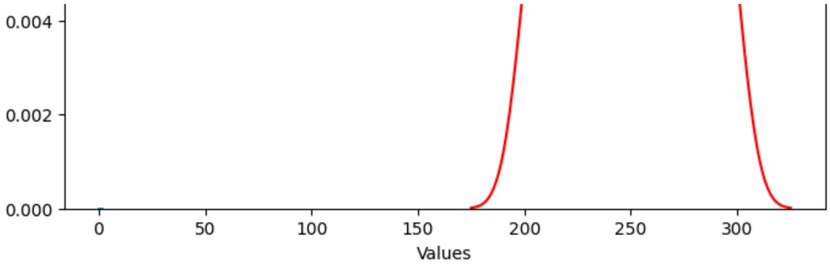


```
#Gaussian distribution & smoothened KDE plot
```

```
mean,std=np.mean(int_list2),np.std(int_list2)
xmin,xmax=plt.xlim()
x=np.linspace(xmin,xmax,100)
p=stats.norm.pdf(x,mean,std)

plt.figure(figsize=(8, 6))
sns.kdeplot(int_list2,color="red",label="KDE Plot")
plt.plot(x,p,linewidth=2,label="Gaussian Distribution")
plt.title("Gaussian Distribution & Smoothened KDE Plot")
plt.xlabel("Values")
plt.ylabel("Density")
plt.legend()
plt.show()
```




Gaussian Distribution & Smoothened KDE Plot
KDE Plot
Gaussian Distribution
Density
0.012
0.010
0.008
0.006




```
#Write a Python function to calculate the range of a given list of numbers.
```

```
def cal_range(num):
  if not num:
    return None
  return max(num)-min(num)
```

```
range_of_list=cal_range(int_list2)
print(f"The range of the list is:{range_of_list}")
```

```
The range of the list is:100
```

```
# Create a program to find the variance and standard deviation of a list of numbers.
```

```
import math
```

```
def cal_variance(num):
  if not num:
    return None

  mean=sum(num)/len(num)

  variance=sum((x-mean)**2 for x in num)/len(num)
  return variance
```

```
def cal_std(num):
  variance=cal_variance(num)
  if variance is None:
    return None

  standard_deviation=math.sqrt(variance)
  return standard_deviation
```

```
variance_of_list=cal_variance(int_list2)
standard_deviation_of_list=cal_std(int_list2)

print(f"The variance of the list is:{variance_of_list}")
print(f"The standard deviation of the list is:{standard_deviation_of_list}")
```

```
The variance of the list is:835.6816960000001
The standard deviation of the list is:28.908159678540592
```

```
#Implement a function to compute the interquartile range (IQR) of a list of values.
```

```
def cal_iqr(num):
  if not num:
    return None

  sorted_num=sorted(num)
  n=len(sorted_num)

  def median(data):
    mid=len(data)//2
    if len(data)%2==0:
      return (data[mid-1]+data[mid])/2
    else:
      return data[mid]

  q1=median(sorted_num[:n//2])
  q3=median(sorted_num[n//2:])

  iqr= q3-q1
  return iqr
```

```
interquartile_range=cal_iqr(int_list2)
print(f"The interquartile range of the list is:{interquartile_range}")
```

```
The interquartile range of the list is:49.0
```

```
#Build a program to calculate the coefficient of variation for a dataset.
```

```
import math
```

```
def cal_mean(num):
  if not num:
    return None

  return sum(num)/len(num)

def cal_variance(num):
  if not num:
    return None
  variance=sum((x-mean)**2 for x in num)/len(num)
  return variance

def cal_std(num):
  variance=cal_variance(num)
  if variance is None:
    return None

  standard_deviation=math.sqrt(variance)
  return standard_deviation


def cal_coeff(num):
  mean=cal_mean(num)
  standard_deviation=cal_std(num)

  if mean==0:
    return None

  return (standard_deviation/mean)*100
```

```
coefficient_of_variation=cal_coeff(int_list2)
print(f"The coefficient of variation is:{coefficient_of_variation:.2f}%")
```

```
The coefficient of variation is:11.52%
```

```
#Write a Python function to find the mean absolute deviation (MAD) of a list of numbers.
```

```
from mmap import MADV_DODUMP
def cal_mad(num):
  if not num:
    return None

  mean=sum(num)/len(num)
  mad=sum(abs(x-mean)for x in num)/len(num)
  return mad
```

```
mad_of_list=cal_mad(int_list2)
print(f"The mean absolute deviation of the list is:{mad_of_list}")
```

```
The mean absolute deviation of the list is:25.20134400000001
```

```
#Create a program to calculate the quartile deviation of a list of values.
```

```
 def median(data):
    mid=len(data)//2
    if len(data)%2==0:
      return (data[mid-1]+data[mid])/2
    else:
      return data[mid]
```

```
def quartile(data):
  sorted_data=sorted(data)
  n=len(sorted_data)

  Q1=median(sorted_data[:n//2])

  if n%2==0:
    Q3=median(sorted_data[n//2:])
  else:
    Q3=median(sorted_data[n//2+1:])

  return Q1,Q3

def quartile_deviation(data):
  Q1,Q3=quartile(data)
  quart_dev=(Q3-Q1)/2
  return quart_dev
```

```
result=quartile_deviation(int_list2)
print(f"The quartile deviation of the list is:{result}")
```

```
The quartile deviation of the list is:24.5
```

```
#Implement a function to find the range-based coefficient of dispersion for a dataset.
```

```
def median(data):
    mid=len(data)//2
    if len(data)%2==0:
      return (data[mid-1]+data[mid])/2
    else:
      return data[mid]
```

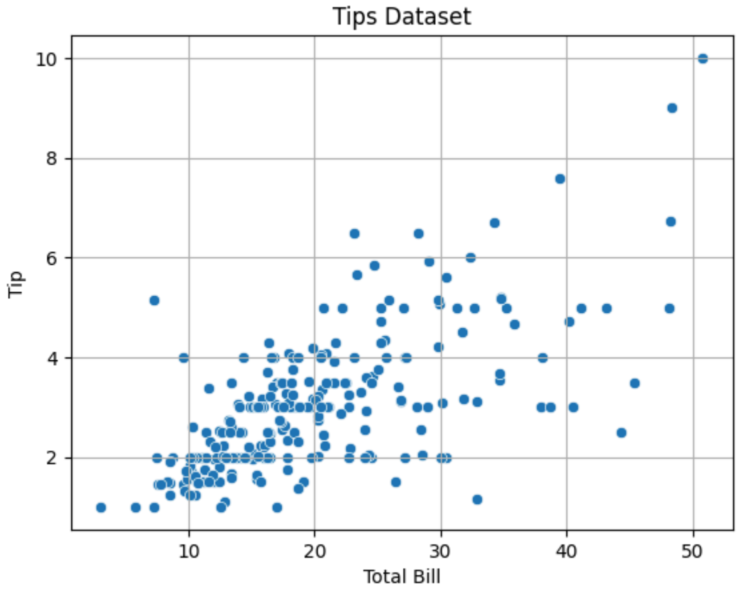
```
def range_coeff(data):
  max_val=max(data)
  min_val=min(data)
  range_val=max_val-min_val
  med_val=median(data)

  range_of_dispersion=(range_val/med_val)*100
  return range_of_dispersion
```

```
result=range_coeff(int_list2)
print(f"The range-based coefficient of dispersion:{result:.2f}%")
```

```
The range-based coefficient of dispersion:41.07%
```

Use seaborn library to load "tips" dataset and the dataset for the columns "total_bill" and "tip"

```
import seaborn as sns
import matplotlib.pyplot as plt

tips=sns.load_dataset("tips")

data=tips[["total_bill","tip"]]

sns.scatterplot(x="total_bill",y="tip",data=data)
plt.title("Tips Dataset")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.grid(True)
plt.show()
```



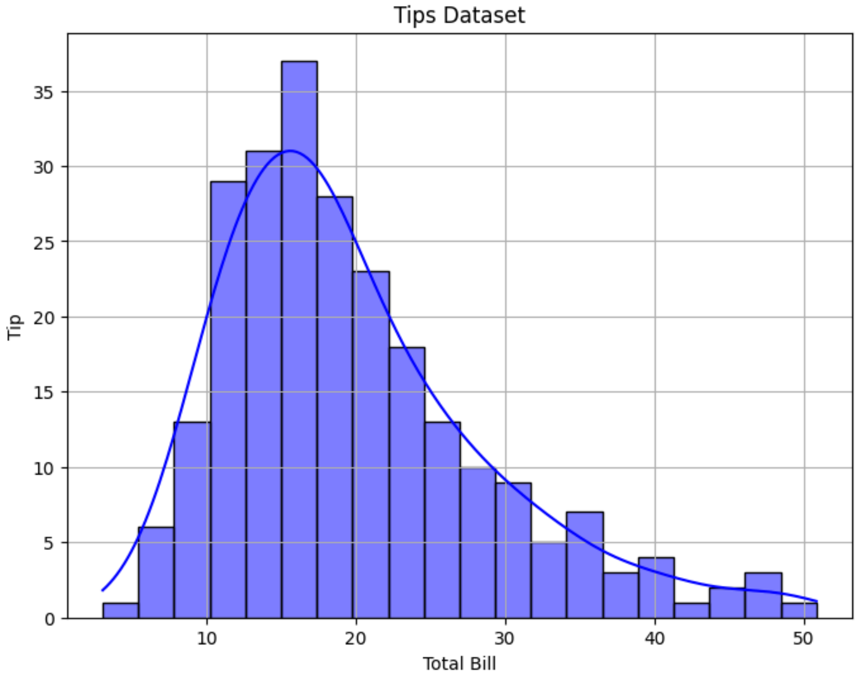

```
#Write a Python function that calculates their skewness.
```

```
import numpy as np
```

```
def skewness(data):
  n=len(data)
  mean=np.mean(data)
  std_dev=np.std(data)

  skewness=(n/((n-1)*(n-2)))*(np.sum((data-mean)**3)/((n-1)*(n-2)*(std_dev**3)))
  return skewness
```

```
result=skewness(tips["total_bill"])
print(f"Skewness of the Dataset:{result:.4f}")

plt.figure(figsize=(8, 6))
sns.histplot(tips["total_bill"], bins=20, kde=True, color="blue",edgecolor="black")
plt.title("Tips Dataset")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.grid(True)
plt.show()
```

```
Skewness of the Dataset:0.0000
```




```
#Create a program that determines whether the columns exhibit positive skewness, negative skewness, or is approximately symmetric
```

```
import numpy as np
```

```
def skewness(data):
```

```
  n=len(data)
  mean=np.mean(data)
  std_dev=np.std(data)

  skewness=(n/((n-1)*(n-2)))*(np.sum((data-mean)**3)/((n-1)*(n-2)*(std_dev**3)))
  return skewness

  def skew_type(skew):
    if skew>0.5:
      return "Positive Skewed"
    elif skew<-0.5:
      return "Negative Skewed"
    else:
      return "Approximately Symmetric"
```

```
skew_positive=skewness(tips["total_bill"])
skew_negative=skewness(tips["total_bill"])
skew_symmetric=skewness(tips["total_bill"])
```