Q1. Generate a list of 100 integers containing values between 90 to 130 and store it in the variable `int_list`. After generating the list, find the following:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
int_list=np.random.randint(90,130,100)
int_list
```

Out[2]:
```
array([ 92, 118, 102, 122, 123, 106, 100,  98, 112, 113, 121, 114, 106,
       103, 101, 123, 104, 102, 112, 110, 117,  95,  96, 121, 119, 125,
        91, 113, 114, 118,  91, 126, 108, 116, 114,  97,  94, 103, 123,
        92,  92, 105, 104, 114, 112,  95,  92, 112,  95, 116, 105,  99,
        97,  90, 122, 101,  91, 100,  95, 115, 112,  94, 101, 129,  97,
       100, 115, 121, 104, 121,  92, 127, 121,  93,  96, 113,  92, 100,
       126, 111,  96, 105, 129, 105,  95, 101, 115, 100,  98, 126, 110,
       129, 126,  90,  91, 128, 118, 111,  93, 106])
```

(i) Write a Python function to calculate the mean of a given list of numbers. Create a function to find the median of a list of numbers.

In [3]:
```
mean_int_list=np.mean(int_list)
median_int_list=np.median(int_list)
print("Mean:",mean_int_list)
print("Median:",median_int_list)
```

```
Mean: 107.44
Median: 105.5
```

(ii) Develop a program to compute the mode of a list of integers.

In [4]:
```
import statistics as stats
mode_int_list=stats.mode(int_list)
print("Mode:",mode_int_list)
```

```
Mode: 92
```

(iii)Implement a function to calculate the weighted mean of a list of values and their corresponding weights.

In [5]:
```
def calculate_weighted_mean(values, weights):
    if len(values) != len(weights):
        return "Error: The number of values and weights should be the same."
    weighted_sum = 0
    total_weight = 0
    for i in range(len(values)):
        weighted_sum += values[i] * weights[i]
        total_weight += weights[i]
    weighted_mean = weighted_sum / total_weight
    return weighted_mean

values = int_list
weights = int_list
result = calculate_weighted_mean(values, weights)
print("The weighted mean is:", round(result,2))
```

```
The weighted mean is: 108.7
```

In [6]:
```
#Using numpy:
weighted_mean=np.average(int_list,weights=int_list)
print("The weighted mean is:", round(weighted_mean,2))
```

```
The weighted mean is: 108.7
```

(iv) Write a Python function to find the geometric mean of a list of positive numbers.

In [7]:
```
def calculate_geometric_mean(values):
    geometric_mean = np.prod(values) ** (1 / len(values))
    return geometric_mean

values =[2,4,6,8,10]
result = calculate_geometric_mean(values)
print("The geometric mean is:", result)
```

```
The geometric mean is: 5.210342169394704
```

(v) Create a program to calculate the harmonic mean of a list of values.

In [8]:
```
def calculate_harmonic_mean(values):
    harmonic_mean = np.mean(1 / np.array(values))
    return harmonic_mean

values = [2,4,6,8,10]
result = calculate_harmonic_mean(values)
print("The harmonic mean is:", result)
```

```
The harmonic mean is: 0.22833333333333333
```

(vi) Build a function to determine the midrange of a list of numbers (average of the minimum and maximum).

In [9]:
```
def Calculate_midrange(l):
    max_=max(l)
    min_=min(l)
    midrange=(max_+min_)/2
    return midrange
Calculate_midrange(int_list)
```

Out[9]:
```
109.5
```

(vii)Implement a Python program to find the trimmed mean of a list, excluding a certain percentage of outliers.

```
In [10]: def calculate_trimmed_mean(num_list, percentage):
             sorted_list = sorted(num_list)
             exclude_count = round((percentage / 100) * len(sorted_list))
             trimmed_list = sorted_list[exclude_count:-exclude_count]
             trimmed_mean = sum(trimmed_list) / len(trimmed_list)
             return trimmed_mean
         calculate_trimmed_mean(int_list,10)
```

```
Out[10]: 107.0125
```

Q2. Generate a list of 500 integers containing values between 200 to 300 and store it in the variable "int_list2". After generating the list, find the following:

```
In [11]: int_list2=np.random.randint(200,300,500)
         len(int_list2)
```

```
Out[11]: 500
```

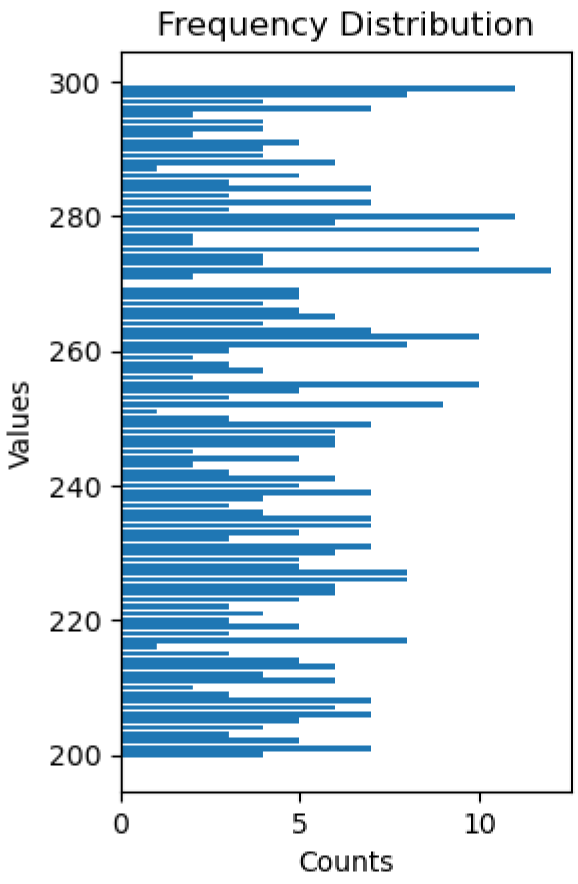
(i).Compare the given list of visualization for the given data:

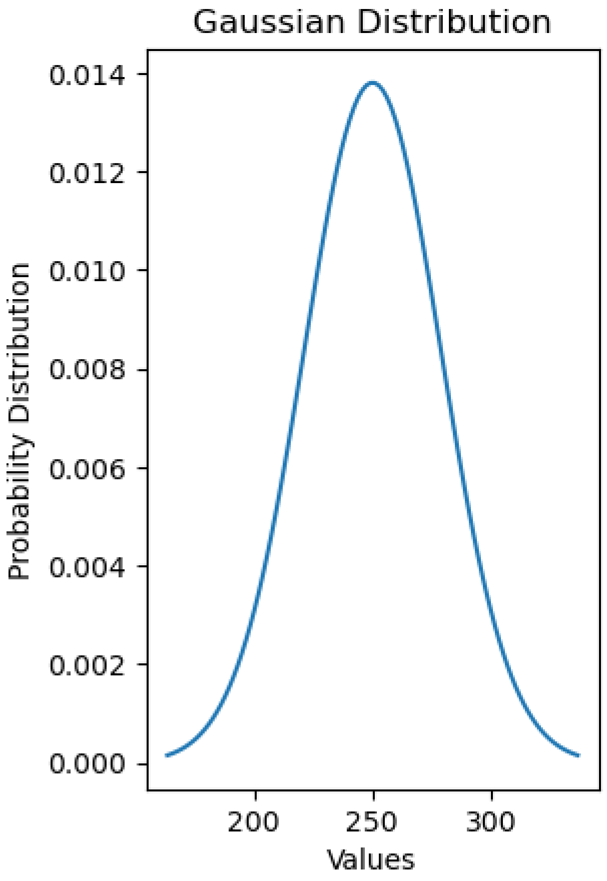
1. Frequency & Gaussian distribution: This visualization shows the frequency of data points along with a Gaussian distribution curve. It helps in understanding the distribution of the data and how closely it aligns with a normal distribution.

```
In [12]: import matplotlib.pyplot as plt
         data=int_list2
         value,counts=np.unique(data,return_counts=True)
         #Frequency distribution
         plt.subplot(1,2,1)
         plt.barh(value,counts)
         plt.ylabel("Values")
         plt.xlabel("Counts")
         plt.title("Frequency Distribution")
         plt.show()

         #Gaussian distribution:
         mu=np.mean(int_list2)
         sigma=np.std(int_list2)
         x=np.linspace(mu-3*sigma,mu+3*sigma,100)
         y=(1/(sigma*np.sqrt(2*np.pi)))*np.exp(-0.5*((x-mu)/sigma)**2)

         plt.subplot(1,2,2)
         plt.plot(x,y)
         plt.xlabel("Values")
         plt.ylabel("Probability Distribution")
         plt.title("Gaussian Distribution")
         plt.show()
```

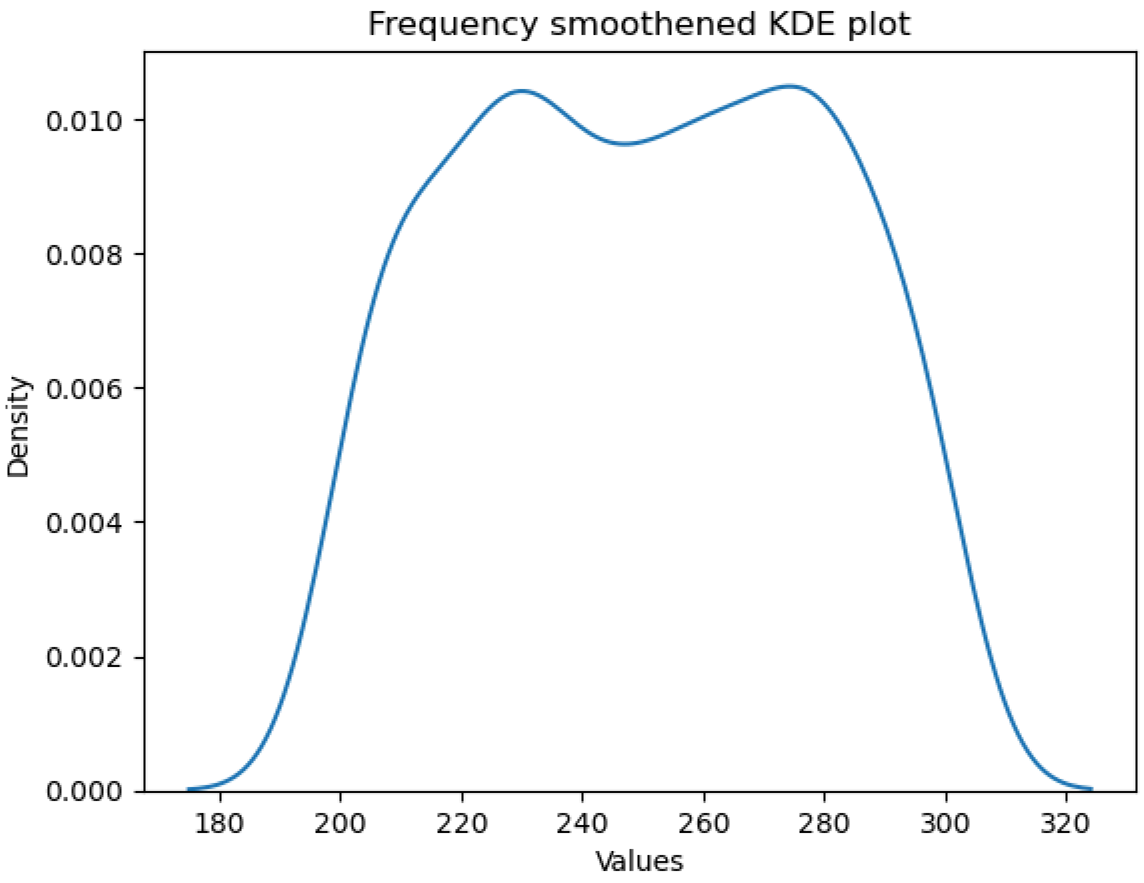

Frequency Distribution
Values
Counts
300
280
260
240
220
200
0
5
10


Gaussian Distribution
Probability Distribution
Values
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
200
250
300

2. Frequency smoothened KDE plot : This visualization represents the data using a Kernel Density Estimation (KDE) plot, which smoothes the data and provides a continuous density estimate. It shows the distribution of the data in a smooth curve, giving insights into the shape and density of the data.

In [13]:
```
import seaborn as sns
data=int_list2
values,counts=np.unique(data,return_counts=True)
#Smooth the frequency table by resampling the data:
smoothed_values = np.repeat(values, counts)

sns.kdeplot(smoothed_values)
plt.xlabel("Values")
plt.ylabel("Density")
plt.title("Frequency smoothened KDE plot")
plt.show()
```



3. Gaussian distribution & smoothened KDE plot: This visualization combines the Gaussian distribution curve with the smoothened KDE plot. It allows for a comparison between the actual data distribution and the estimated distribution based on the KDE.

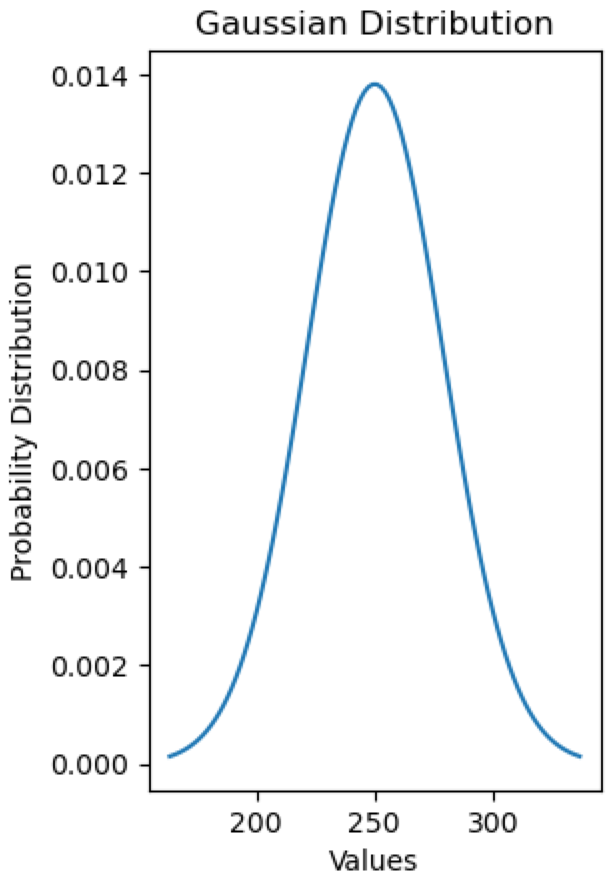
In [14]:
```
#Gaussian Distribution
```

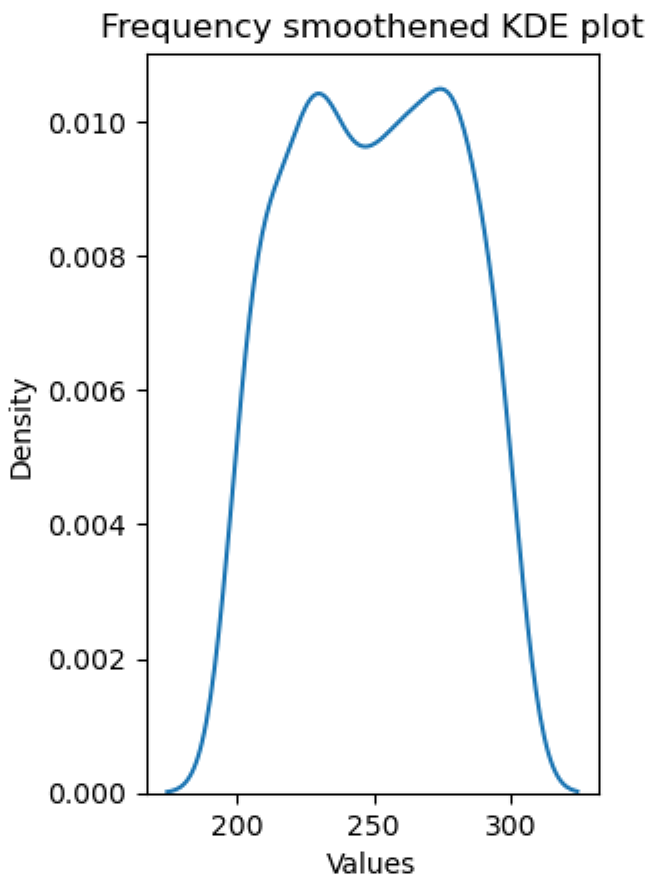
```
mu=np.mean(int_list2)
sigma=np.std(int_list2)
x=np.linspace(mu-3*sigma,mu+3*sigma,100)
y=(1/(sigma*np.sqrt(2*np.pi)))*np.exp(-0.5*((x-mu)/sigma)**2)

plt.subplot(1,2,1)
plt.plot(x,y)
plt.xlabel("Values")
plt.ylabel("Probability Distribution")
plt.title("Gaussian Distribution")
plt.show()

#Smoothened KDE plot
import seaborn as sns
data=int_list2
values,counts=np.unique(data,return_counts=True)
#Smooth the frequency table by resampling the data:
smoothed_values = np.repeat(values, counts)

plt.subplot(1,2,2)
sns.kdeplot(smoothed_values)
plt.xlabel("Values")
plt.ylabel("Density")
plt.title("Frequency smoothened KDE plot")
plt.show()
```





(ii) Write a python function to calculate the range of a given list of numbers

In [15]:
```
def range_(l):
    return max(l)-min(l)
print('Range:',range_(int_list2))
```

```
Range: 99
```

(iii)Create a program to find the variance and standard deviation of list of numbers.

In [16]:
```
def Variance(l,dof):
    n=len(l)
    #Find out mean
    mean=round(sum(l)/n,2)
    #Deviation
    deviation=[(x-mean)**2 for x in l]
    variance=sum(deviation)/(n-dof)
    return variance
Sample_Variance=Variance(int_list2,1)
Population_variance=Variance(int_list2,0)
print("Sample Variance:",round(Sample_Variance,2))
print("population Variance",round(Population_variance,2))
print("Standard_deviation:",np.sqrt(Variance(int_list2,0)).round(2))
```

```
Sample Variance: 837.24
population Variance 835.57
Standard_deviation: 28.91
```

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

```
In [17]: def IQR(l):
             q1,q3=np.percentile(int_list2,[25,75])
             return q3-q1
         print("IQR:",IQR(int_list2))
```

```
IQR: 49.0
```

(v).Build a program to calculate the coefficient of variation for a dataset.

```
In [18]: def coefficient_of_variation(data):
             mean = np.mean(data)
             std_dev = np.std(data)
             coefficient = (std_dev / mean) * 100
             return coefficient

         cv = coefficient_of_variation(int_list2).round(2)
         print("The coefficient of variation is:", cv)
```

```
The coefficient of variation is: 11.56
```

(vi) Write a python function to find the mean absolute deviation(MAD) of a list of numbers.

```
In [19]: def mean_absolute_deviation(numbers):
             mean = sum(numbers) / len(numbers)
             deviations = [abs(num - mean) for num in numbers]
             mad = sum(deviations) / len(numbers)
             return mad

         data = int_list2
         mad = mean_absolute_deviation(data).round(2)
         print("The Mean Absolute Deviation is:", mad)
```

```
The Mean Absolute Deviation is: 25.07
```

(vii) Create a program to calculate the quartile deviation of a list of values

```
In [20]: def quartile_deviation(data):
             q1 = np.percentile(data, 25)
             q3 = np.percentile(data, 75)
             deviation = (q3 - q1) / 2
             return deviation

         dataset = int_list2
         qd = quartile_deviation(dataset)
         print("The quartile deviation is:", qd)
```

```
The quartile deviation is: 24.5
```

(viii) Implement a function to find the range-based coefficient of dispersion for a dataset.

In [21]:
```python
def range_coefficient_of_dispersion(data):
    data_range = max(data) - min(data)
    data_mean = sum(data) / len(data)
    coefficient = (data_range / data_mean) * 100
    return coefficient

dataset = int_list2
rcd = range_coefficient_of_dispersion(dataset).round(2)
print("The range-based coefficient of dispersion is:", rcd)
```

```
The range-based coefficient of dispersion is: 39.58
```

3. Write a Python class representing a discrete random variable with methods to calculate its expected value and variance.

In [22]:
```python
class DiscreteRandomVariable:
    def __init__(self,values,probabilities):
        self.values=values
        self.probabilities=probabilities
    def expected_value(self):
        return sum(value*probability for value, probability in zip(self.valu

    def variance(self):
        expected_value=self.expected_value()
        return sum((value-expected_value)**2*probability for value,probabili
values = [1, 2, 3, 4]
probabilities = [0.2, 0.3, 0.4, 0.1]

rv = DiscreteRandomVariable(values, probabilities)
print("Expected Value:", round(rv.expected_value(),2))
print("Variance:", round(rv.variance(),2))
```

```
Expected Value: 2.4
Variance: 0.84
```

4. Implement a program to stimulate the rolling of a fair six-sided die and calculate the expected value and variance of the outcomes.

In [23]:
```python
import random

def roll_die():
    return random.randint(1, 6)

def simulate_rolls(num_rolls):
    rolls = [roll_die() for i in range(num_rolls)]
    return rolls

def calculate_expected_value(rolls):
    return sum(rolls) / len(rolls)

def calculate_variance(rolls):
    expected_value = calculate_expected_value(rolls)
    squared_diff = [(roll - expected_value) ** 2 for roll in rolls]
    return sum(squared_diff) / len(rolls)

rolls = simulate_rolls(100)

expected_value = calculate_expected_value(rolls)
variance = calculate_variance(rolls)

print("Expected Value:", expected_value)
print("Variance:", variance)
```

```
Expected Value: 3.5
Variance: 2.95
```

5. Create a Python function to generate random sample from a given probability distribution(eg. binomial,poisson) and calculate their mean and variance.

In [24]:
```
def generate_sample(distribution,size):
    if distribution=="binomial":
        sample=np.random.binomial(n=10,p=0.5,size=size)
    elif distribution=="poisson":
        sample=np.random.poisson(lam=5,size=size)
    else:
        return "Invalid distribution. Please choose either 'binomial' or 'po

    mean=np.mean(sample)
    variance=np.var(sample)
    return mean, variance
sample_mean, sample_variance = generate_sample("binomial", 1000)
print("Mean:", round(sample_mean,2))
print("Variance:", round(sample_variance,2))
```

```
Mean: 4.91
Variance: 2.28
```

6. Write a Python script to generate random numbers from a Gaussian(normal) distribution and compute the mean, variance and standard deviation of the samples.

In [25]:
```
def generate_gaussian_samples(mean, std_dev, size):
    samples = np.random.normal(mean, std_dev, size)
    sample_mean = np.mean(samples).round(2)
    sample_variance = np.var(samples).round(2)
    sample_std_dev = np.std(samples).round(2)
    return sample_mean, sample_variance, sample_std_dev

mean = 0
std_dev = 1
sample_size = 1000
sample_mean, sample_variance, sample_std_dev = generate_gaussian_samples(mea
print("Mean:", sample_mean)
print("Variance:", sample_variance)
print("Standard Deviation:", sample_std_dev)
```

```
Mean: 0.01
Variance: 1.01
Standard Deviation: 1.01
```

7. Use seaborn library to load 'tips' dataset. Find the following from the dataset for the columns "total_bill" and "tip"

In [26]:
```
import seaborn as sns
tips=sns.load_dataset("tips")
tips
```

Out[26]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

(i) Write a python function that calculate thir skewness.

In [27]:
```
def calculate_skewness():
    total_bill=tips["total_bill"]
    tip=tips["tip"]
    total_bill_skewness=total_bill.skew().round(2)
    tip_skewness=tip.skew().round(2)
    print("Skewness of total_bill column:", total_bill_skewness)
    print("Skewness of tip column:", tip_skewness)
calculate_skewness()
```

```
Skewness of total_bill column: 1.13
Skewness of tip column: 1.47
```

(ii) Create a program that determines whether the columns exhibit positive skewness, negative skewness or is approximately symmetric.

In [28]:
```
def determine_skewness():
    tips_dataset = sns.load_dataset("tips")
    total_bill_column = tips_dataset["total_bill"]
    tip_column = tips_dataset["tip"]
    total_bill_skewness = total_bill_column.skew()
    tip_skewness = tip_column.skew()

    if total_bill_skewness > 0:
        total_bill_skewness_type = "positive skewness"
    elif total_bill_skewness < 0:
        total_bill_skewness_type = "negative skewness"
    else:
        total_bill_skewness_type = "approximately symmetric"

    if tip_skewness > 0:
        tip_skewness_type = "positive skewness"
    elif tip_skewness < 0:
        tip_skewness_type = "negative skewness"
    else:
        tip_skewness_type = "approximately symmetric"

    print("Total_bill column has", total_bill_skewness_type)
    print("Tip column has", tip_skewness_type)

# Call the function to determine skewness
determine_skewness()
```

```
Total_bill column has positive skewness
Tip column has positive skewness
```

(iii)Write a function that calculate the covariance between two columns.

In [29]:
```
def calculate_covariance(column1, column2):
    tips_dataset = tips  # Replace "tips.csv" with your dataset file name
    covariance = tips_dataset["total_bill"].cov(tips_dataset["tip"]).round(2
    return covariance

# Example usage:
covariance = calculate_covariance("total_bill", "tip")
print("The covariance between 'total_bill' and 'tip' is:", covariance)
```

```
The covariance between 'total_bill' and 'tip' is: 8.32
```

(iv)Implement a python program that calculate the Pearson correlation coefficient between two columns.

In [32]:
```
def calculate_correlation(column1, column2):
    tips_dataset = tips
    correlation = tips_dataset[column1].corr(tips_dataset[column2])
    return correlation

correlation = calculate_correlation("total_bill", "tip").round(2)
print("The Pearson correlation coefficient between 'total_bill' and 'tip' is
```

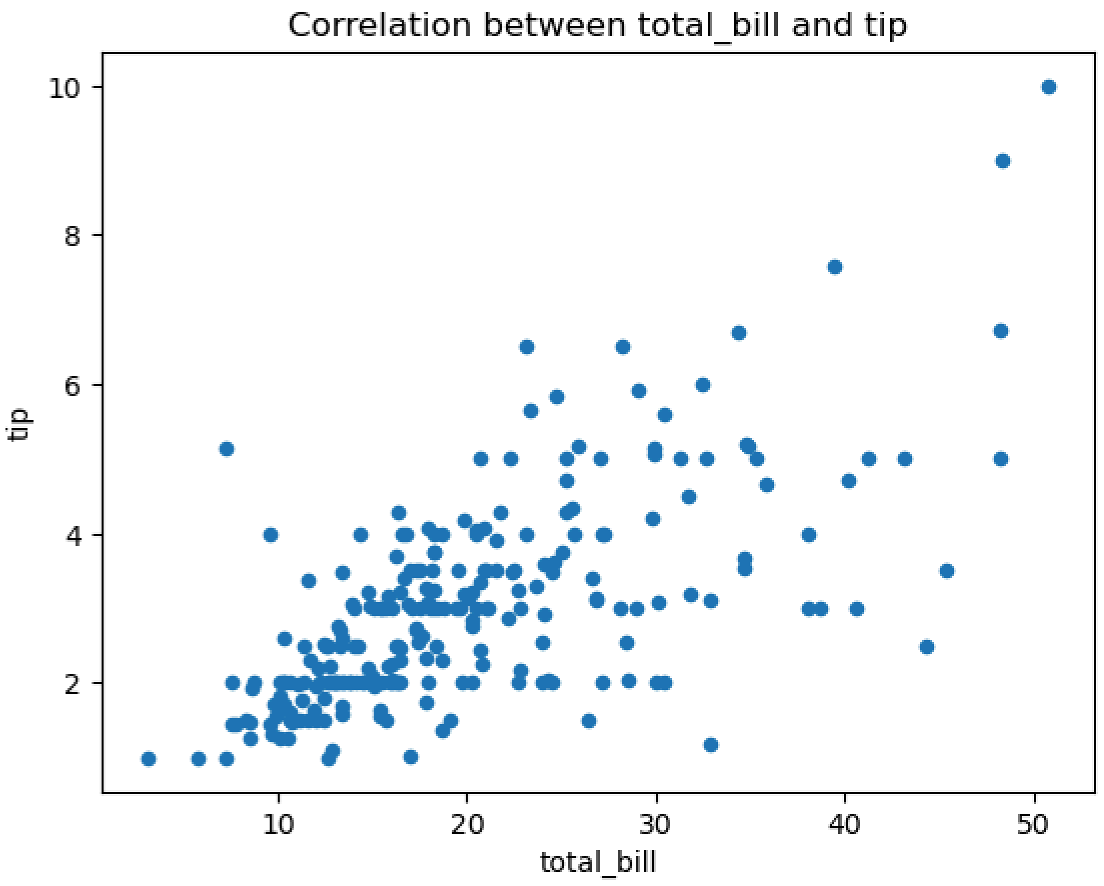
```
The Pearson correlation coefficient between 'total_bill' and 'tip' is: 0.68
```

(v) Write a script to visualize the correlation between two specific columns in a pandas DataFrame using scatter plots.

In [33]:
```
import matplotlib.pyplot as plt

def visualize_correlation(column1, column2):
    tips_dataset = tips
    tips_dataset.plot.scatter(x=column1, y=column2)
    plt.xlabel(column1)
    plt.ylabel(column2)
    plt.title("Correlation between " + column1 + " and " + column2)
    plt.show()

visualize_correlation("total_bill", "tip")
```



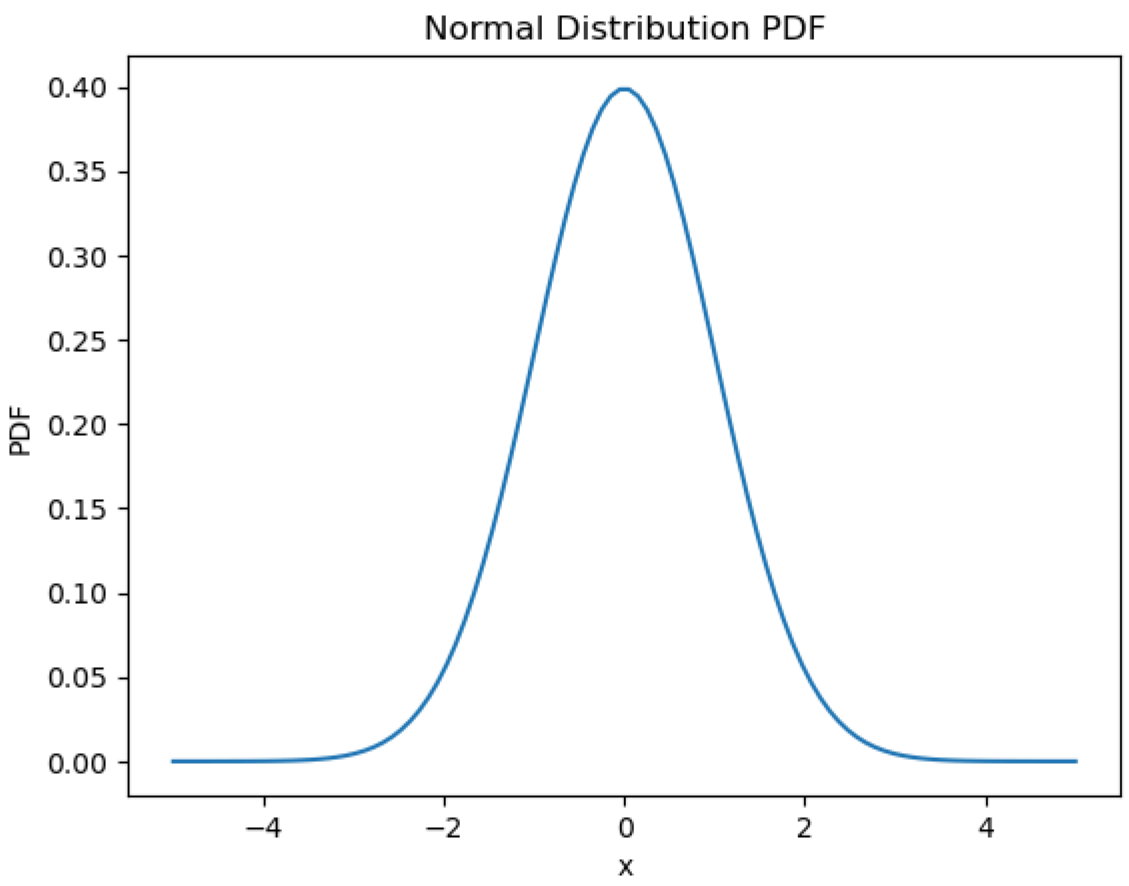
8. Write a Python function to calculate the probability density function(PDF) of a continuous random

variable for a given normal distribution.

In [4]:
```
from scipy.stats import norm
def calculate_pdf(x,mean,std_dev):
    pdf=norm.pdf(x,mean,std_dev)
    return pdf
x=np.linspace(-5,5,100)
mean=0
```

```python
std_dev=1
pdf=calculate_pdf(x,mean,std_dev)

plt.plot(x,pdf)
plt.xlabel("x")
plt.ylabel("PDF")
plt.title("Normal Distribution PDF")
plt.show()
```



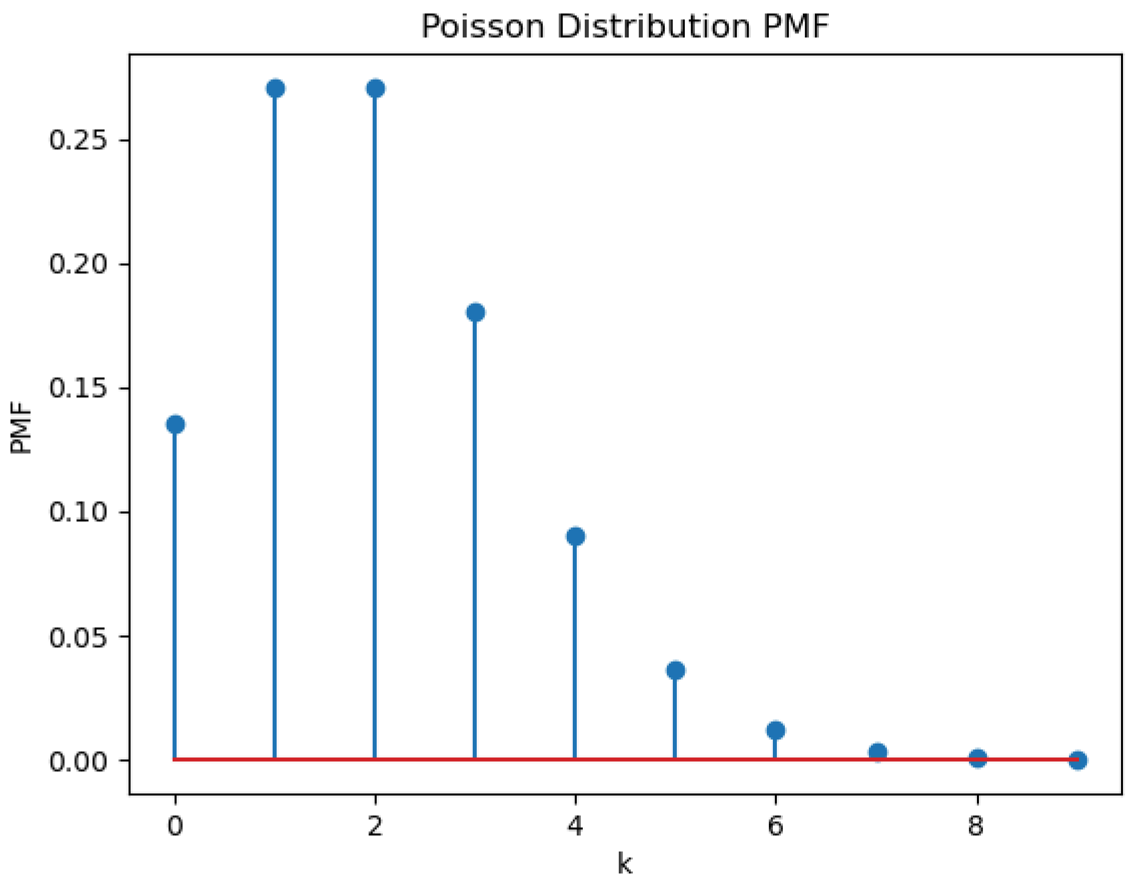
10. Write a Python function to calculate the probability mass function(PMF) of Poisson distribution.

In [5]:
```python
from scipy.stats import poisson

def calculate_pmf(k, lambda_val):
    pmf = poisson.pmf(k, lambda_val)
    return pmf

k = np.arange(0, 10)  #values of k
lambda_val = 2  #parameter lambda
pmf = calculate_pmf(k, lambda_val)

plt.stem(k, pmf)
plt.xlabel('k')
plt.ylabel('PMF')
plt.title('Poisson Distribution PMF')
plt.show()
```



11. A company wants to test if a new website layout leads to a higher conversion rate(percentage of visitors who make a purchase).They collect data from the old and new layouts to compare.

To generate the data use following command:

50 purchase out of 1000 visitors

old_layouts=np.array([1]*50+[0]*950)

70 purchase out of 1000 visitors

new_layouts=np.array([1]*70+[0]*930) Apply z_test to find which layout is successful.

In [2]:
```
from statsmodels.stats.proportion import proportions_ztest

#Define the data
old_layouts = np.array([1] * 50 + [0] * 950)
new_layouts = np.array([1] * 70 + [0] * 930)
```

```
#Perform the z-test
successes = np.array([np.sum(old_layouts), np.sum(new_layouts)])
nobs = np.array([len(old_layouts), len(new_layouts)])

z_score, p_value = proportions_ztest(successes, nobs)

#Interpret the results
if p_value < 0.05:
    print("The difference in conversion rates is statistically significant."
    if z_score < 0:
        print("The new layout has a higher conversion rate.")
    else:
        print("The old layout has a higher conversion rate.")
else:
    print("There is no statistically significant difference in conversion ra
```

```
There is no statistically significant difference in conversion rates.
```

12. A tutoring service claims that its program improves student's exam scores. A sample of students who participated in th program was taken, and their scores before and after the program we recorded. Use the below code to generate sample of respective array of marks: before_program=np.array([75,80,85,70,90,78,92,88,82,87 after_program=np.array([80,85,90,80,92,80,95,90,85,88]) Use z-test to find if the claims made by tutor are true or false.

In [3]:
```
import numpy as np
from scipy.stats import norm

#Define the data
before_program = np.array([75, 80, 85, 70, 90, 78, 92, 88, 82, 87])
after_program = np.array([80, 85, 90, 80, 92, 80, 95, 90, 85, 88])

#Calculate the mean and standard deviation of the differences
differences = after_program - before_program
mean_diff = np.mean(differences)
std_diff = np.std(differences, ddof=1) / np.sqrt(len(differences))

#Perform the z-test
z_score = (mean_diff - 0) / std_diff
p_value = 2 * (1 - norm.cdf(abs(z_score)))

#Interpret the results
if p_value < 0.05:
    print("The tutoring program has a statistically significant impact on ex
    if z_score > 0:
        print("The students' scores improved after the program.")
    else:
        print("The students' scores decreased after the program.")
```

```
else:
    print("There is no statistically significant impact of the tutoring prog
```

```
The tutoring program has a statistically significant impact on exam scores.
The students' scores improved after the program.
```

13.A pharmaceutical company wants to determine if a new drug is effective in reducing blood pressure. They conduct a study and record blood pressure measurement before and after administering the drug. Use the below code to generate sample of respective arrays of blood pressure: before_drug=np.array([145,150,140,135,155,160,152,148 after_drug=np.array([130,140,132,128,145,148,138,136,1 Implement z_test to find if the drug really works or not.

In [4]:
```
from scipy.stats import norm

# Define the data
before_drug = np.array([145, 150, 140, 135, 155, 160, 152, 148, 130, 138])
after_drug = np.array([130, 140, 132, 128, 145, 148, 138, 136, 125, 130])

# Calculate the mean and standard deviation of the differences
differences = after_drug - before_drug
mean_diff = np.mean(differences)
std_diff = np.std(differences, ddof=1) / np.sqrt(len(differences))

# Perform the z-test
z_score = (mean_diff - 0) / std_diff
p_value = 2 * (1 - norm.cdf(abs(z_score)))

# Interpret the results
if p_value < 0.05:
    print("The drug has a statistically significant effect in reducing blood
    if z_score < 0:
        print("The drug lowers blood pressure.")
    else:
        print("The drug increases blood pressure.")
else:
    print("There is no statistically significant effect of the drug in reduc
```

```
The drug has a statistically significant effect in reducing blood pressure.
The drug lowers blood pressure.
```

14.A customer service department claims that their average response time is less than 5 minutes. A sample of recent customer interaction was taken, and the response time were recorded. implement the below code to generate the array of response time: response_times=np.array([4.3,3.8,5.1,4.9,4.7,4.2,5.2,4.5,4 Implement z_test to find the claims made by customer service department are true or false.

In [5]:
```python
from scipy.stats import norm

#Define the data
response_times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.4]

#Calculate the sample mean and standard deviation
sample_mean = np.mean(response_times)
sample_std = np.std(response_times, ddof=1)

#Set the null hypothesis mean and significance level
null_mean = 5
alpha = 0.05

#Calculate the z-score
z_score = (sample_mean - null_mean) / (sample_std / np.sqrt(len(response_tim

#Calculate the p-value
p_value = 1 - norm.cdf(z_score)

#Interpret the results
if p_value < alpha:
    print("The claims made by the customer service department are false.")
else:
    print("The claims made by the customer service department are true.")
```

```
The claims made by the customer service department are true.
```

In [5]:
```python
'''15.A company is testing two different website layouts to see which one le
Write a Python function to perform an A/B test analysis, including calculati
freedom, and p-value:
Use the following data:
layouts_a_click=[28,32,33,29,31,34,30,35,36,37]
layouts_b_clicks=[40,41,38,42,39,44,43,41,45,47]'''
import scipy.stats as stats

def ab_test(layouts_a_click, layouts_b_clicks):
    t_stat, p_value = stats.ttest_ind(layouts_a_click, layouts_b_clicks)
    dof = len(layouts_a_click) + len(layouts_b_clicks) - 2
    return t_stat, dof, p_value

layouts_a_click = [28, 32, 33, 29, 31, 34, 30, 35, 36, 37]
layouts_b_clicks = [40, 41, 38, 42, 39, 44, 43, 41, 45, 47]

t_stat, dof, p_value = ab_test(layouts_a_click, layouts_b_clicks)
print("t_stats:",t_stat)
print("dof:",dof)
print("p_value:",p_value)
```

```
t_stats: -7.298102156175071
dof: 18
p_value: 8.833437608301987e-07
```

16.A pharmaceutical company wants to determine if a new drug is more effective than an existing drug in reducing cholestrol levels.Create a program to analyze

the clinical trial data and calculate the t- statistic and p-value for the treatment effect. Use the following data of cholestrol level: existing_drug_level= [180,182,175,185,178,172,184,179,183] new_drug_levels= [170,172,165,168,175,173,170,178,172,176]

In [6]:
```
import scipy.stats as stats

def analyze_clinical_trial(existing_drug_levels, new_drug_levels):
    t_stat, p_value = stats.ttest_ind(existing_drug_levels, new_drug_levels)
    return t_stat, p_value

existing_drug_levels = [180, 182, 175, 185, 178, 172, 184, 179, 183]
new_drug_levels = [170, 172, 165, 168, 175, 173, 170, 178, 172, 176]

t_stat, p_value = analyze_clinical_trial(existing_drug_levels, new_drug_leve
print("t_statistics:",t_stat)
print("p_value:",p_value)
```

```
t_statistics: 4.206274198750941
p_value: 0.0005935325314742236
```

17. A school district introduces an educational intervention program to improve math scores.Write a Python function to analyze pre-and post-intervention test score, calculating the t-statistics and p-value to determine if the intervention has a significant impact. Use the following data of test score: pre_intervention_scores= [80,85,90,75,88,82,92,78,85,87] post_intervention_score= [90,92,88,92,95,91,96,93,89,93]

In [7]:
```
import scipy.stats as stats

def analyze_intervention(pre_intervention_scores, post_intervention_scores):
    t_stat, p_value = stats.ttest_rel(pre_intervention_scores, post_interven
    return t_stat, p_value

pre_intervention_scores = [80, 85, 90, 75, 88, 82, 92, 78, 85, 87]
post_intervention_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]

t_stat, p_value = analyze_intervention(pre_intervention_scores, post_interve
print("t_statistics:",t_stat)
print("p_value:",p_value)
```

```
t_statistics: -4.42840883965761
p_value: 0.0016509548165795493
```

18. An HR department wants to investigate if there's a gender-based salary gap within the company. Develop a

program to analyze salary data, calculate the t-statistics, and determine if there's a statistically significant difference between the average salaries of male and female employees. Use the below code to generate synthetics data:

```
#Generate synthetic salary data for male and female employees
np.random.seed(0) #For reproducibility
male_salaries=np.random.normal(loc=50000,scale=10000,size=20)
female_salaries=np.random.normal(loc=55000,scale=9000,size=20)
```

In [3]:
```
import scipy.stats as stats

# Generate synthetic salary data for male and female employees
np.random.seed(0)  # For reproducibility
male_salaries = np.random.normal(loc=50000, scale=10000, size=20)
female_salaries = np.random.normal(loc=55000, scale=9000, size=20)

def analyze_salary_gap(male_salaries, female_salaries):
    t_stat, p_value = stats.ttest_ind(male_salaries, female_salaries)
    return t_stat, p_value

t_stat, p_value = analyze_salary_gap(male_salaries, female_salaries)
print("t_statistics:",t_stat)
print("p_value:",p_value)
```

```
t_statistics: 0.06114208969631383
p_value: 0.9515665020676465
```

19. A manufacturer produce two different versions of a product and wants to compare their quality score. Create a Python function to analyze quality assesment data, calculate the t-statistic, and decide whether there's significant difference in quality between the two versions. Use the following data:

```
version1_scores=
[85,88,82,89,87,84,90,88,85,86,91,83,87,84,89,86,84,88,85,86,89,90,87,
version2_scores=
[80,78,83,81,79,82,76,80,78,81,77,82,80,79,82,79,80,81,79,82,79,78,80,
```

In [2]:
```
import scipy.stats as stats

version1_scores = [85, 88, 82, 89, 87, 84, 90, 88, 85, 86, 91, 83, 87, 84, 
version2_scores = [80, 78, 83, 81, 79, 82, 76, 80, 78, 81, 77, 82, 80, 79, 

def analyze_quality_scores(version1_scores, version2_scores):
    t_stat, p_value = stats.ttest_ind(version1_scores, version2_scores)
    return t_stat, p_value

t_stat, p_value = analyze_quality_scores(version1_scores, version2_scores)
```

```
print("t_statistics:",t_stat)
print("p_value:",p_value)
```

```
t_statistics: 11.325830417646698
p_value: 3.6824250702873965e-15
```

20.A restaurant chain collects customer satisfaction scores for two different branches.Write a program to analyze the score, calculate the t-statistic, and determine if there's statistically significant difference in customer satisfaction between the branches. Use the below data of scores:

```
branch_a_scores=
[4,5,3,4,5,4,5,3,4,4,5,4,4,3,4,5,5,4,3,4,5,4,3,5,4,4,5,3,4,5,4]
branch_b_scores=
[3,4,2,3,4,3,4,2,3,3,4,3,3,2,3,4,4,3,2,3,4,3,2,4,3,3,4,2,3,4,3]
```

In [3]:
```
import scipy.stats as stats

branch_a_scores = [4, 5, 3, 4, 5, 4, 5, 3, 4, 4, 5, 4, 4, 3, 4, 5, 5, 4, 3,
branch_b_scores = [3, 4, 2, 3, 4, 3, 4, 2, 3, 3, 4, 3, 3, 2, 3, 4, 4, 3, 2,

def analyze_customer_satisfaction(branch_a_scores, branch_b_scores):
    t_stat, p_value = stats.ttest_ind(branch_a_scores, branch_b_scores)
    return t_stat, p_value

t_stat, p_value = analyze_customer_satisfaction(branch_a_scores, branch_b_sc

if p_value < 0.05:
    print("There is a statistically significant difference in customer satis
else:
    print("There is no statistically significant difference in customer sati
```

```
There is a statistically significant difference in customer satisfaction bet
ween the branches.
```

21.A political analyst wants to determine if there is a significant association between age groups and voter preferences(Candidate A or Candidate B).They collect data from a sample of 500 voters and classify them into different age groups and candidate preferences.Perform a Chi-Square test to determine if there is a significant association between age groups and voter preferences. Use the below code to generate data:

```
np.random.seed(0)
age_groups=np.random.choice(['18-30','31-50','51+','51+'],size=30)
voter_prferences=np.random.choice(["Candidate A","Candidate
B"],size=30)
```

In [4]:
```python
from scipy.stats import chi2_contingency

np.random.seed(0)
age_groups = np.random.choice(['18-30', '31-50', '51+', '51+'], size=30)
voter_preferences = np.random.choice(["Candidate A", "Candidate B"], size=30

#Create a contingency table
contingency_table = np.zeros((3, 2))
for i in range(len(age_groups)):
    if age_groups[i] == '18-30':
        row = 0
    elif age_groups[i] == '31-50':
        row = 1
    else:
        row = 2
    if voter_preferences[i] == 'Candidate A':
        col = 0
    else:
        col = 1
    contingency_table[row, col] += 1

#Perform Chi-Square test
chi2, p_value,_,_ = chi2_contingency(contingency_table)

if p_value < 0.05:
    print("There is a significant association between age groups and voter p
else:
    print("There is no significant association between age groups and voter
```

```
There is no significant association between age groups and voter preference
s.
```

22.A company conducted a customer satisfaction survey to determine if there is a significant relationship between product satisfaction levels (Satisfied, Neutral, Dissatisfied) and the region where customer are located(East,West,North,South).The survey data is summarised in a contingency table.Conduct a Chi-Square test to determine if there is a significant relationship between product satisfaction levels and customer regions.

Sample data:

```python
#Sample data: Product satisfaction levels(row) vs. Customer
regions(columns)
data=np.array([[50,30,40,20],[30,40,30,50],[20,30,40,30]])
```

In [6]:
```python
from scipy.stats import chi2_contingency

data = np.array([[50, 30, 40, 20], [30, 40, 30, 50], [20, 30, 40, 30]])

#Perform Chi-Square test
```

```python
chi2, p_value, _, _ = chi2_contingency(data)

if p_value < 0.05:
    print("There is a significant relationship between product satisfaction
else:
    print("There is no significant relationship between product satisfactior
```

```
There is a significant relationship between product satisfaction levels and
customer regions.
```

23.A company implemented an employee training program to improve job performance(Effective,Neutral, Ineffective).After the training, the collected data from a sample of employees and classified them based on their job performance before and after the training. Perform a Chi-Square test to determine if there is a significant difference between job performance levels before and after the training. Sample data:

```python
#Sample data:Job performance levels before(rows) and after(columns)
training
data=np.array([[50,30,20],[30,40,30],[20,30,40]])
```

In [7]:
```python
from scipy.stats import chi2_contingency

data = np.array([[50, 30, 20], [30, 40, 30], [20, 30, 40]])

# Perform Chi-Square test
chi2, p_value, _, _ = chi2_contingency(data)

if p_value < 0.05:
    print("There is a significant difference between job performance levels
else:
    print("There is no significant difference between job performance levels
```

```
There is a significant difference between job performance levels before and
after the training.
```

24.A company produces three different versions of a product:Standard,Premium, and Deluxe.The company wants to determine if there is a significant difference in customer satisfaction scores among the three product versions. They conducted a survey and collected customer satisfaction scores for each version from a random sample of customers. Perform an ANOVA test to determine if there is a significant Use the following data:

```
#Sample data: Customer satisfaction scores for each product version
standard_scores=[80,85,90,78,88,82,92,78,85,87]
premium_scores=[90,92,88,92,95,91,96,93,89,93]
deluxe_scores=[95,98,92,97,96,94,98,97,92,99]
```

In [8]:
```
from scipy.stats import f_oneway

standard_scores = [80, 85, 90, 78, 88, 82, 92, 78, 85, 87]
premium_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]
deluxe_scores = [95, 98, 92, 97, 96, 94, 98, 97, 92, 99]

#Perform ANOVA test
f_value, p_value = f_oneway(standard_scores, premium_scores, deluxe_scores)

if p_value < 0.05:
    print("There is a significant difference in customer satisfaction scores
else:
    print("There is no significant difference in customer satisfaction score
```

```
There is a significant difference in customer satisfaction scores among the
three product versions.
```

In [ ]: