```
# Statistics.
```

```
#  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:
import numpy as np

# Set seed for reproducibility
np.random.seed(42)

# Generate 100 random integers between 90 and 130
int_list = np.random.randint(90, 131, size=100)
print("Generated List:", int_list)
```

```
Generated List: [128 118 104  97 110 128 108 112 100 100 113 125 129 113  92 111  91 113
 119 127  91 110 122 101 111 114 116 117 105 104  92 126  96 110  98 128
 107  93 114 103  98 115  91 109 117  96  97 124 103 106 125 129  93  91
  95  93 118 107 115 123  99 125 103 120 104  97 103 112 129 110 105 107
 113 115 114 130 118 104  90 114  96  98 113  90  97 113 100 106  97 124
 124 122  94 128 130 117  96  98  97 101]
```

```
#  (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
def calculate_mean(numbers):
    return np.mean(numbers)

mean_value = calculate_mean(int_list)
print("Mean:", mean_value)


def calculate_median(numbers):
    return np.median(numbers)

median_value = calculate_median(int_list)
print("Median:", median_value)
```

```
Mean: 108.94
Median: 109.5
```

```
#  (ii) Develop a program to compute the mode of a list of integers
from scipy import stats

def calculate_mode(numbers):
    mode_result = stats.mode(numbers, keepdims=True)
    return mode_result.mode[0] if mode_result.count[0] > 1 else None

mode_value = calculate_mode(int_list)
print("Mode:", mode_value)
```

```
Mode: 97
```

```
# (iii) Implement a function to calculate the weighted mean of a list of values and their corresponding weights.
def calculate_weighted_mean(values, weights):
    return np.average(values, weights=weights)

# Generate random weights for testing
weights = np.random.randint(1, 5, size=100)

weighted_mean_value = calculate_weighted_mean(int_list, weights)
print("Weighted Mean:", weighted_mean_value)
```

```
Weighted Mean: 109.27642276422765
```

```
#  (iv) Write a Python function to find the geometric mean of a list of positive numbers.
from scipy.stats import gmean

def calculate_geometric_mean(numbers):
    return gmean(numbers)

geometric_mean_value = calculate_geometric_mean(int_list)
print("Geometric Mean:", geometric_mean_value)
```

```
Geometric Mean: 108.31139336758402
```

```
#  (v) Create a program to calculate the harmonic mean of a list of values
from scipy.stats import hmean

def calculate_harmonic_mean(numbers):
    return hmean(numbers)

harmonic_mean_value = calculate_harmonic_mean(int_list)
print("Harmonic Mean:", harmonic_mean_value)
```

```
Harmonic Mean: 107.68717366618071
```

```
#  (vi) Build a function to determine the midrange of a list of numbers (average of the minimum and maximum).
def calculate_midrange(numbers):
    return (np.min(numbers) + np.max(numbers)) / 2

midrange_value = calculate_midrange(int_list)
print("Midrange:", midrange_value)
```

```
Midrange: 110.0
```

```
#  (vii) Implement a Python program to find the trimmed mean of a list, excluding a certain percentage of
# outliers.
from scipy.stats import trim_mean

def calculate_trimmed_mean(numbers, percentage):
    return trim_mean(numbers, proportiontocut=percentage/100)

trimmed_mean_value = calculate_trimmed_mean(int_list, 10)
print("Trimmed Mean:", trimmed_mean_value)
```

```
Trimmed Mean: 108.675
```

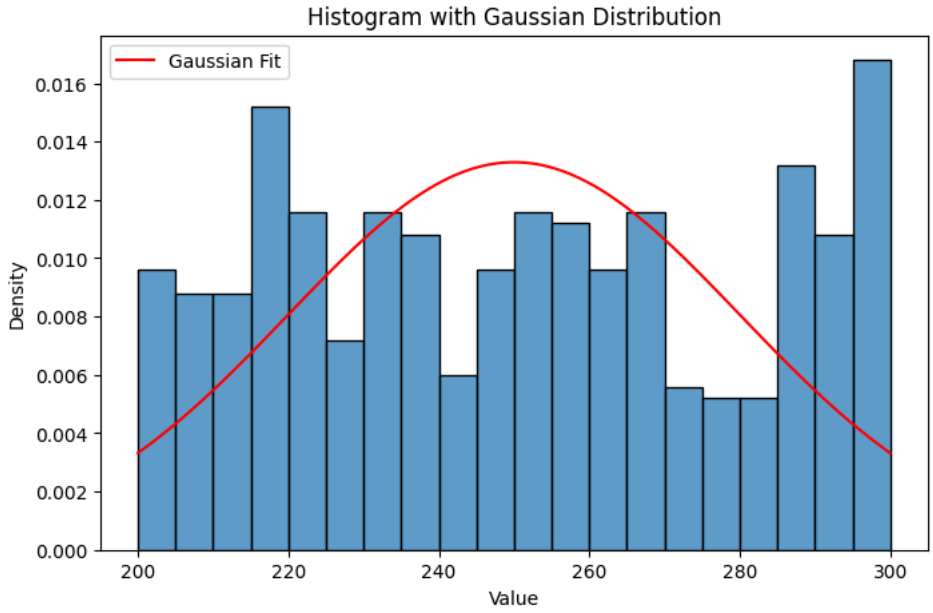
```
#  2. Generate a list of 500 integers containing values between 200 to 300 and store it in the variable `int_list2`.
# After generating the list, find the following:
# Generate 500 random integers between 200 and 300
int_list2 = np.random.randint(200, 301, size=500)
print("Generated List:", int_list2)
```

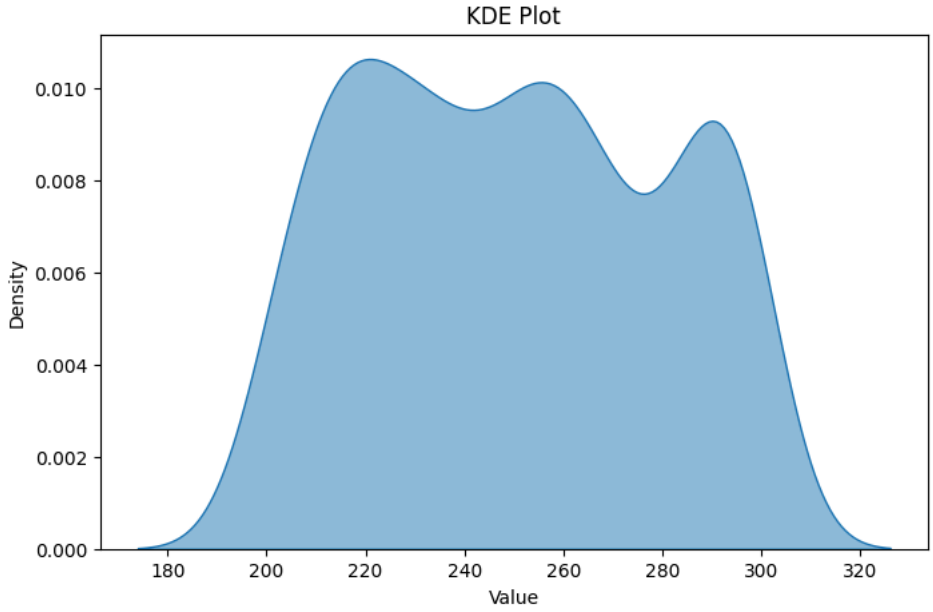
```
Generated List: [250 243 223 278 258 231 295 287 251 261 257 251 211 238 201 202 300 255
 280 258 201 201 291 253 286 300 295 296 200 218 201 252 243 289 231 269
 231 267 254 274 255 216 237 223 268 297 269 285 210 215 296 272 258 269
 279 292 202 219 258 235 218 289 266 218 219 295 270 251 232 239 238 281
 200 210 291 256 288 249 222 230 293 241 298 206 215 289 259 201 200 247
 211 268 236 231 208 298 218 247 279 202 219 223 253 232 223 274 271 235
 237 283 298 288 298 224 292 217 281 265 253 234 279 260 240 299 232 267
 232 213 220 247 219 207 206 266 216 232 247 275 258 285 221 229 237 250
 253 207 226 226 297 220 229 296 227 263 296 268 260 247 218 203 234 263
 248 216 243 291 229 292 245 205 298 236 223 292 245 252 294 298 259 296
 262 284 231 286 232 266 217 224 294 253 257 266 245 223 231 246 285 222
 265 226 201 289 216 232 208 242 247 238 292 241 225 298 249 224 223 212
 259 206 256 235 244 219 264 207 215 213 275 286 214 291 297 265 231 286
 262 285 250 224 257 262 261 221 257 257 285 248 251 241 269 214 253 259
 300 296 207 252 259 204 267 205 295 293 246 298 254 239 251 215 212 229
 218 216 262 218 291 257 254 289 300 289 261 222 208 211 200 257 200 233
 295 247 288 200 215 260 263 262 268 221 292 266 275 225 215 250 300 285
 256 228 277 291 268 246 293 261 268 275 215 289 289 247 284 238 299 232
 293 300 222 209 268 299 233 251 294 209 218 257 295 200 268 203 215 223
 279 201 291 231 290 283 223 211 249 234 232 232 260 250 242 300 211 266
 264 232 239 273 242 243 228 212 211 294 245 201 234 286 280 289 207 292
 225 273 289 233 206 267 257 274 228 235 288 220 235 209 300 272 223 263
 298 248 298 235 281 295 223 222 261 295 236 211 254 212 222 288 298 229
 216 261 283 288 285 212 258 218 248 299 211 260 218 275 208 270 227 277
 294 251 282 215 268 298 211 224 251 284 299 252 222 215 256 238 252 241
 257 238 213 294 204 234 286 292 274 217 275 208 273 257 216 206 245 212
 239 241 208 249 226 265 204 228 236 237 282 207 264 285 216 270 288 244
 203 235 269 230 218 260 253 238 290 273 289 218 238 266]
```

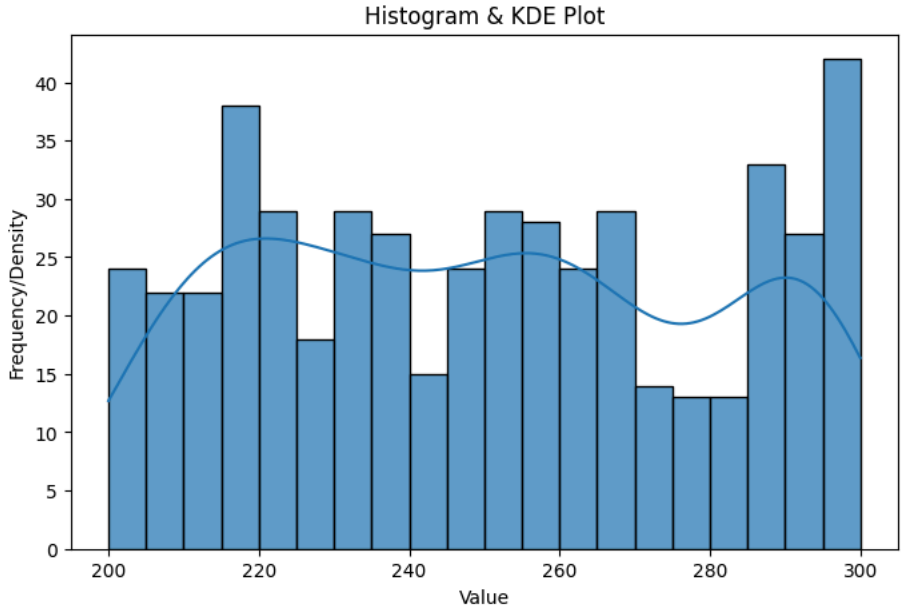
```
#  (i) Compare the given list of visualization for the given data:
#  1. Frequency & Gaussian distribution
#  2. Frequency smoothened KDE plot
#  3. Gaussian distribution & smoothened KDE plot

import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

def plot_histogram_with_gaussian(data, title):
    plt.figure(figsize=(8, 5))
```

```
    plt.figure(figsize=(8, 5))
    sns.histplot(data, bins=20, kde=False, edgecolor='black', alpha=0.7, stat='density')
    mu, sigma = np.mean(data), np.std(data)
    x = np.linspace(min(data), max(data), 100)
    plt.plot(x, norm.pdf(x, mu, sigma), color='red', label='Gaussian Fit')
    plt.title(title)
    plt.xlabel("Value")
    plt.ylabel("Density")
    plt.legend()
    plt.show()

plot_histogram_with_gaussian(int_list2, "Histogram with Gaussian Distribution")


def plot_kde(data, title):
    plt.figure(figsize=(8, 5))
    sns.kdeplot(data, fill=True, alpha=0.5)
    plt.title(title)
    plt.xlabel("Value")
    plt.ylabel("Density")
    plt.show()

plot_kde(int_list2, "KDE Plot")


def plot_gaussian_and_kde(data, title):
    plt.figure(figsize=(8, 5))
    sns.histplot(data, kde=True, bins=20, edgecolor='black', alpha=0.7)
    plt.title(title)
    plt.xlabel("Value")
    plt.ylabel("Frequency/Density")
    plt.show()

plot_gaussian_and_kde(int_list2, "Histogram & KDE Plot")
```

[4]








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

def calculate_range(numbers):
    return np.max(numbers) - np.min(numbers)

range_value = calculate_range(int_list2)
```

```
print("Range:", range_value)
```

```
Range: 100
```

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

def calculate_variance(numbers):
    return np.var(numbers, ddof=1)

def calculate_std_dev(numbers):
    return np.std(numbers, ddof=1)

variance_value = calculate_variance(int_list2)
std_dev_value = calculate_std_dev(int_list2)

print("Variance:", variance_value)
print("Standard Deviation:", std_dev_value)
```

```
Variance: 902.6090741482965
Standard Deviation: 30.043453099607184
```

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

def calculate_iqr(numbers):
    q1, q3 = np.percentile(numbers, [25, 75])
    return q3 - q1

iqr_value = calculate_iqr(int_list2)
print("Interquartile Range (IQR):", iqr_value)
```

```
Interquartile Range (IQR): 52.0
```

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

def calculate_coefficient_of_variation(numbers):
    return (np.std(numbers, ddof=1) / np.mean(numbers)) * 100

cv_value = calculate_coefficient_of_variation(int_list2)
print("Coefficient of Variation:", cv_value)
```

```
Coefficient of Variation: 12.017958101831763
```

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

def calculate_mad(numbers):
    return np.mean(np.abs(numbers - np.mean(numbers)))

mad_value = calculate_mad(int_list2)
print("Mean Absolute Deviation (MAD):", mad_value)
```

```
Mean Absolute Deviation (MAD): 25.932095999999998
```

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

def calculate_quartile_deviation(numbers):
    q1, q3 = np.percentile(numbers, [25, 75])
    return (q3 - q1) / 2

quartile_deviation_value = calculate_quartile_deviation(int_list2)
print("Quartile Deviation:", quartile_deviation_value)
```

```
Quartile Deviation: 26.0
```

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

def calculate_range_based_dispersion(numbers):
    return (np.max(numbers) - np.min(numbers)) / (np.max(numbers) + np.min(numbers))

range_dispersion_value = calculate_range_based_dispersion(int_list2)
print("Range-Based Coefficient of Dispersion:", range_dispersion_value)
```

```
Range-Based Coefficient of Dispersion: 0.2
```

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

import numpy as np

class DiscreteRandomVariable:
    def __init__(self, values, probabilities):
        self.values = np.array(values)
        self.probabilities = np.array(probabilities)

    def expected_value(self):
        return np.sum(self.values * self.probabilities)

    def variance(self):
        mean = self.expected_value()
        return np.sum(self.probabilities * (self.values - mean) ** 2)

# Example usage
values = [1, 2, 3, 4, 5]
probabilities = [0.1, 0.2, 0.3, 0.2, 0.2]

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

```
Expected Value: 3.2
Variance: 1.56
```

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

import numpy as np

# Simulate rolling a fair die 10,000 times
np.random.seed(42)
die_rolls = np.random.randint(1, 7, size=10000)

# Calculate Expected Value and Variance
expected_value = np.mean(die_rolls)
variance = np.var(die_rolls, ddof=1)

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

```
Expected Value: 3.4999
Variance: 2.91969195919592
```

```
#  5.Create a Python function to generate random samples from a given probability distribution (e.g.,
# binomial, Poisson) and calculate their mean and variance.

import numpy as np
from scipy.stats import binom, poisson

def generate_random_samples(distribution, params, size=1000):
    if distribution == "binomial":
        n, p = params
        return binom.rvs(n, p, size=size)
    elif distribution == "poisson":
        lambda_ = params[0]
        return poisson.rvs(lambda_, size=size)
    else:
        raise ValueError("Unsupported distribution")

# Generate Binomial Samples (n=10, p=0.5)
binomial_samples = generate_random_samples("binomial", (10, 0.5))

# Generate Poisson Samples (λ=4)
poisson_samples = generate_random_samples("poisson", (4,))

print("Binomial Mean:", np.mean(binomial_samples))
print("Binomial Variance:", np.var(binomial_samples, ddof=1))

print("Poisson Mean:", np.mean(poisson_samples))
print("Poisson Variance:", np.var(poisson_samples, ddof=1))
```

```
Binomial Mean: 5.022
Binomial Variance: 2.5060220220220217
Poisson Mean: 3.986
Poisson Variance: 4.2540580580580585
```

```
# 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.

import numpy as np
from scipy.stats import norm

# Generate 1000 samples from a normal distribution (mean=50, std=10)
np.random.seed(42)
normal_samples = np.random.normal(loc=50, scale=10, size=1000)

# Calculate mean, variance, and standard deviation
mean_value = np.mean(normal_samples)
variance_value = np.var(normal_samples, ddof=1)
std_dev_value = np.std(normal_samples, ddof=1)

print("Mean:", mean_value)
print("Variance:", variance_value)
print("Standard Deviation:", std_dev_value)
```

```
Mean: 50.193320558223256
Variance: 95.88638535851024
Standard Deviation: 9.792159381796756
```

```
#  7.Use seaborn library to load `tips` dataset. Find the following from the dataset for the columns `total_bill`
# and `tip`
import seaborn as sns
import numpy as np
import pandas as pd

# Load the Seaborn "tips" dataset
tips = sns.load_dataset("tips")

# Display the first few rows
print(tips.head())
```

```
   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
```

```
#  (i) Write a Python function that calculates their skewness.

from scipy.stats import skew

def calculate_skewness(column):
    return skew(tips[column])

total_bill_skewness = calculate_skewness("total_bill")
tip_skewness = calculate_skewness("tip")

print("Skewness of Total Bill:", total_bill_skewness)
print("Skewness of Tip:", tip_skewness)
```

```
Skewness of Total Bill: 1.1262346334818638
Skewness of Tip: 1.4564266884221506
```

```
#  (ii) Create a program that determines whether the columns exhibit positive skewness, negative skewness, or is approximately symmetric
def skewness_type(value):
    if value > 0:
        return "Positively Skewed"
    elif value < 0:
        return "Negatively Skewed"
    else:
        return "Approximately Symmetric"

print("Total Bill Skewness Type:", skewness_type(total_bill_skewness))
print("Tip Skewness Type:", skewness_type(tip_skewness))
```

```
Total Bill Skewness Type: Positively Skewed
Tip Skewness Type: Positively Skewed
```

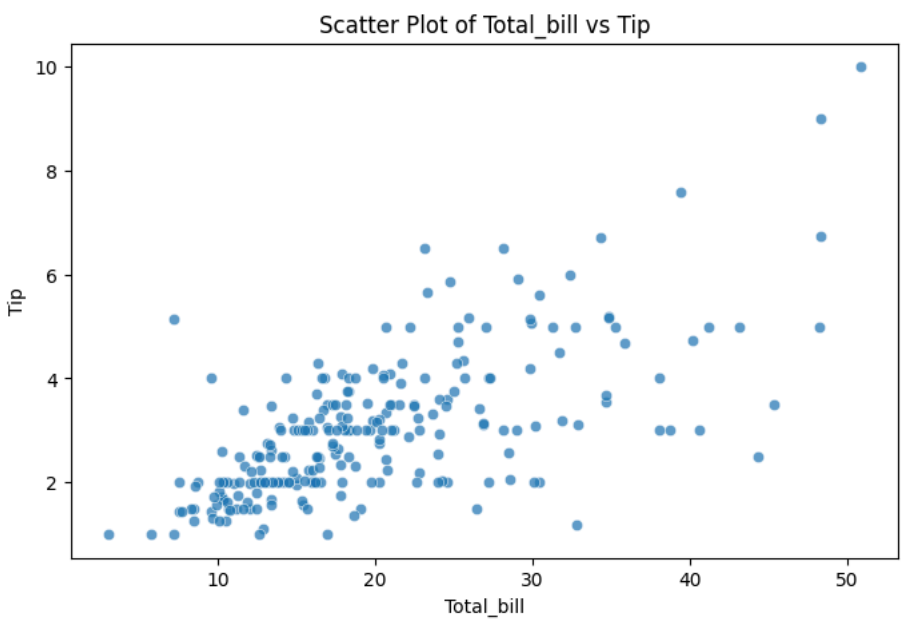
```
# (iii) Write a function that calculates the covariance between two columns
def calculate_covariance(column1, column2):
    return np.cov(tips[column1], tips[column2])[0, 1]

covariance_value = calculate_covariance("total_bill", "tip")
print("Covariance between Total Bill and Tip:", covariance_value)
```

```
Covariance between Total Bill and Tip: 8.323501629224854
```

```
#  (iv) Implement a Python program that calculates the Pearson correlation coefficient between two columns.

def calculate_pearson_correlation(column1, column2):
    return np.corrcoef(tips[column1], tips[column2])[0, 1]

pearson_correlation = calculate_pearson_correlation("total_bill", "tip")
print("Pearson Correlation Coefficient:", pearson_correlation)
```

```
Pearson Correlation Coefficient: 0.6757341092113641
```

```
#  (v) Write a script to visualize the correlation between two specific columns in a Pandas DataFrame using
# scatter plots

import matplotlib.pyplot as plt

def plot_scatter(column1, column2):
    plt.figure(figsize=(8, 5))
    sns.scatterplot(x=tips[column1], y=tips[column2], alpha=0.7)
    plt.xlabel(column1.capitalize())
    plt.ylabel(column2.capitalize())
    plt.title(f"Scatter Plot of {column1.capitalize()} vs {column2.capitalize()}")
    plt.show()

plot_scatter("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.
import math

def normal_pdf(x, mu=0, sigma=1):
    """
    Calculate the probability density function (PDF) of a normal distribution.

    Parameters:
    x (float): The point at which to evaluate the PDF
    mu (float): Mean of the distribution (default 0)
    sigma (float): Standard deviation of the distribution (default 1)

    Returns:
```

```
    float: The PDF value at point x
    """
    coefficient = 1 / (sigma * math.sqrt(2 * math.pi))
    exponent = -((x - mu) ** 2) / (2 * sigma ** 2)
    return coefficient * math.exp(exponent)

# Example usage:
print(normal_pdf(0))  # PDF at x=0 for standard normal distribution
print(normal_pdf(1, mu=0, sigma=1))  # PDF at x=1 for standard normal distribution
```

```
0.3989422804014327
0.24197072451914337
```

```
# 9. Create a program to calculate the cumulative distribution function (CDF) of exponential distribution.

import math

def exponential_cdf(x, lambd=1):
    """
    Calculate the cumulative distribution function (CDF) of an exponential distribution.

    Parameters:
    x (float): The point at which to evaluate the CDF
    lambd (float): Rate parameter (λ) of the exponential distribution (default 1)

    Returns:
    float: The CDF value at point x
    """
    if x < 0:
        return 0
    return 1 - math.exp(-lambd * x)

# Example usage:
print(exponential_cdf(1))  # CDF at x=1 for λ=1
print(exponential_cdf(2, lambd=0.5))  # CDF at x=2 for λ=0.5
```

```
0.6321205588285577
0.6321205588285577
```

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

import math

def poisson_pmf(k, lambd):
    """
    Calculate the probability mass function (PMF) of a Poisson distribution.

    Parameters:
    k (int): The number of occurrences
    lambd (float): Average rate of occurrence (λ)

    Returns:
    float: The PMF value for k occurrences
    """
    if k < 0:
        return 0
    return (math.exp(-lambd) * (lambd ** k)) / math.factorial(k)

# Example usage:
print(poisson_pmf(3, 2))  # PMF for k=3 with λ=2
print(poisson_pmf(5, 3.5))  # PMF for k=5 with λ=3.5
```

```
0.1804470443154836
0.1321685997861737
```

```
# 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 the following command:
#  ```python
#  import numpy as np
#  # 50 purchases out of 1000 visitors
#  old_layout = np.array([1] * 50 + [0] * 950)
#  # 70 purchases out of 1000 visitors
# new_layout = np.array([1] * 70 + [0] * 930)
#  ```
#  Apply z-test to find which layout is successful
import numpy as np
from statsmodels.stats.proportion import proportions_ztest

# Generate the data
old_layout = np.array([1] * 50 + [0] * 950)
```

```
new_layout = np.array([1] * 70 + [0] * 930)

# Perform z-test for two proportions
count = np.array([old_layout.sum(), new_layout.sum()])
nobs = np.array([len(old_layout), len(new_layout)])
z_stat, p_value = proportions_ztest(count, nobs, alternative='smaller')

print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Reject null hypothesis - new layout has significantly higher conversion rate")
else:
    print("Conclusion: Fail to reject null hypothesis - no significant difference in conversion rates")
```

```
Z-statistic: -1.8831
P-value: 0.0298
Conclusion: Reject null hypothesis - new layout has significantly higher conversion rate
```

```
# 12. A tutoring service claims that its program improves students' exam scores. A sample of students who
# participated in the program was taken, and their scores before and after the program were recorded.
#  Use the below code to generate samples of respective arrays of marks:
#  ```python
#  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.

import numpy as np
from statsmodels.stats.weightstats import ztest

# Generate 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])

# Perform paired z-test
z_stat, p_value = ztest(after_program, before_program, alternative='larger')

print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Reject null hypothesis - tutoring program significantly improves scores")
else:
    print("Conclusion: Fail to reject null hypothesis - no significant improvement from tutoring")
```

```
Z-statistic: 1.3600
P-value: 0.0869
Conclusion: Fail to reject null hypothesis - no significant improvement from tutoring
```

```
# 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 measurements before and after administering the drug.
#  Use the below code to generate samples of respective arrays of blood pressure:
#  ```python
#  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])
#  ```
#  Implement z-test to find if the drug really works or not.

import numpy as np
from statsmodels.stats.weightstats import ztest

# Generate 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])

# Perform paired z-test
z_stat, p_value = ztest(before_drug, after_drug, alternative='larger')

print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Reject null hypothesis - drug significantly reduces blood pressure")
else:
    print("Conclusion: Fail to reject null hypothesis - no significant effect from drug")
```

```
Z-statistic: 2.6396
P-value: 0.0042
Conclusion: Reject null hypothesis - drug significantly reduces blood pressure
```

```python
# 14.  A customer service department claims that their average response time is less than 5 minutes. A sample
# of recent customer interactions was taken, and the response times were recorded.
#  Implement the below code to generate the array of response time:
#  ```python
#  response_times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.4])
#  ```
#  Implement z-test to find the claims made by customer service department are tru or false

import numpy as np
from statsmodels.stats.weightstats import ztest

# Generate 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])

# Perform one-sample z-test against claim of 5 minutes
z_stat, p_value = ztest(response_times, value=5, alternative='smaller')

print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Reject null hypothesis - average response time is significantly less than 5 minutes")
else:
    print("Conclusion: Fail to reject null hypothesis - no evidence response time is less than 5 minutes")
```

```
Z-statistic: -3.1845
P-value: 0.0007
Conclusion: Reject null hypothesis - average response time is significantly less than 5 minutes
```

```python
# 15. A company is testing two different website layouts to see which one leads to higher click-through rates.
# Write a Python function to perform an A/B test analysis, including calculating the t-statistic, degrees of
# freedom, and p-value.
#  Use the following data:
#  ```python
#  layout_a_clicks = [28, 32, 33, 29, 31, 34, 30, 35, 36, 37]
#  layout_b_clicks = [40, 41, 38, 42, 39, 44, 43, 41, 45, 47]
import numpy as np
from scipy import stats

# Generate the data
layout_a_clicks = np.array([28, 32, 33, 29, 31, 34, 30, 35, 36, 37])
layout_b_clicks = np.array([40, 41, 38, 42, 39, 44, 43, 41, 45, 47])

# Perform independent two-sample t-test
t_stat, p_value = stats.ttest_ind(layout_b_clicks, layout_a_clicks)

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Reject null hypothesis - significant difference in click-through rates between layouts")
    if layout_b_clicks.mean() > layout_a_clicks.mean():
        print("Layout B performs significantly better than Layout A")
    else:
        print("Layout A performs significantly better than Layout B")
else:
    print("Conclusion: Fail to reject null hypothesis - no significant difference between layouts")
```

```
T-statistic: 7.2981
P-value: 0.0000
Conclusion: Reject null hypothesis - significant difference in click-through rates between layouts
Layout B performs significantly better than Layout A
```

```python
# 16. A pharmaceutical company wants to determine if a new drug is more effective than an existing drug in
# reducing cholesterol levels. Create a program to analyze the clinical trial data and calculate the t
# Use the following data of cholestrol level:
#  ```python
#  existing_drug_levels = [180, 182, 175, 185, 178, 176, 172, 184, 179, 183]
#  new_drug_levels = [170, 172, 165, 168, 175, 173, 170, 178, 172, 176

import numpy as np
from scipy import stats

# Generate the data
existing_drug_levels = np.array([180, 182, 175, 185, 178, 176, 172, 184, 179, 183])
new_drug_levels = np.array([170, 172, 165, 168, 175, 173, 170, 178, 172, 176])

# Perform independent two-sample t-test
t_stat, p_value = stats.ttest_ind(existing_drug_levels, new_drug_levels)
```

```
print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Reject null hypothesis - significant difference in cholesterol reduction")
    if new_drug_levels.mean() < existing_drug_levels.mean():
        print("New drug is significantly more effective at reducing cholesterol")
    else:
        print("Existing drug is significantly more effective at reducing cholesterol")
else:
    print("Conclusion: Fail to reject null hypothesis - no significant difference between drugs")
```

```
T-statistic: 4.1405
P-value: 0.0006
Conclusion: Reject null hypothesis - significant difference in cholesterol reduction
New drug is significantly more effective at reducing cholesterol
```

```
# 17.A school district introduces an educational intervention program to improve math scores. Write a Python
# function to analyze pre- and post-intervention test scores, calculating the t-statistic and p-value to
# determine if the intervention had a significant impact.
#  Use the following data of test score:
#  ```python
#  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]

import numpy as np
from scipy import stats

# Generate the data
pre_scores = np.array([80, 85, 90, 75, 88, 82, 92, 78, 85, 87])
post_scores = np.array([90, 92, 88, 92, 95, 91, 96, 93, 89, 93])

# Perform paired t-test
t_stat, p_value = stats.ttest_rel(post_scores, pre_scores)

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Reject null hypothesis - intervention had significant positive impact")
else:
    print("Conclusion: Fail to reject null hypothesis - no significant impact from intervention")
```

```
T-statistic: 4.4284
P-value: 0.0017
Conclusion: Reject null hypothesis - intervention had significant positive impact
```

```
# 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-statistic, and determine if there's a statistically
# significant difference between the average salaries of male and female employees.
#  Use the below code to generate synthetic data:
#  ```python
#  # 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

import numpy as np
from scipy import stats

# Generate synthetic data
np.random.seed(0)
male_salaries = np.random.normal(loc=50000, scale=10000, size=20)
female_salaries = np.random.normal(loc=55000, scale=9000, size=20)

# Perform independent t-test
t_stat, p_value = stats.ttest_ind(female_salaries, male_salaries)

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Significant salary difference between genders")
    if female_salaries.mean() > male_salaries.mean():
        print("Female salaries are significantly higher")
    else:
        print("Male salaries are significantly higher")
else:
    print("Conclusion: No significant salary difference between genders")
```

```
T-statistic: -0.0611
P-value: 0.9516
Conclusion: No significant salary difference between genders
```

```
# 19. A manufacturer produces two different versions of a product and wants to compare their quality scores.
# Create a Python function to analyze quality assessment data, calculate the t-statistic, and decide
# whether there's a significant difference in quality between the two versions.
#  Use the following data:
#  ```python
#  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, 88, 85]
#  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, 81, 82]

import numpy as np
from scipy import stats

# Generate the data
version1 = np.array([85, 88, 82, 89, 87, 84, 90, 88, 85, 86, 91, 83, 87, 84, 89, 86, 84, 88, 85, 86, 89, 90, 87, 88, 85])
version2 = np.array([80, 78, 83, 81, 79, 82, 76, 80, 78, 81, 77, 82, 80, 79, 82, 79, 80, 81, 79, 82, 79, 78, 80, 81, 82])

# Perform independent t-test
t_stat, p_value = stats.ttest_ind(version1, version2)

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Significant quality difference between versions")
    if version1.mean() > version2.mean():
        print("Version 1 has significantly higher quality scores")
    else:
        print("Version 2 has significantly higher quality scores")
else:
    print("Conclusion: No significant quality difference between versions")
```

```
T-statistic: 11.3258
P-value: 0.0000
Conclusion: Significant quality difference between versions
Version 1 has significantly higher quality scores
```

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

import numpy as np
from scipy import stats

# Generate the data
branch_a = np.array([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 = np.array([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])

# Perform independent t-test
t_stat, p_value = stats.ttest_ind(branch_a, branch_b)

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Significant satisfaction difference between branches")
    if branch_a.mean() > branch_b.mean():
        print("Branch A has significantly higher satisfaction")
    else:
        print("Branch B has significantly higher satisfaction")
else:
    print("Conclusion: No significant satisfaction difference between branches")
```

```
T-statistic: 5.4801
P-value: 0.0000
Conclusion: Significant satisfaction difference between branches
Branch A has significantly higher satisfaction
```

```
# 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:
#  ```python
#  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)

import numpy as np
from scipy.stats import chi2_contingency

# Generate the data
np.random.seed(0)
age_groups = np.random.choice(['18-30', '31-50', '51+'], size=500)
voter_preferences = np.random.choice(['Candidate A', 'Candidate B'], size=500)

# Create contingency table
contingency_table = np.zeros((3, 2), dtype=int)
for i, age in enumerate(['18-30', '31-50', '51+']):
    for j, candidate in enumerate(['Candidate A', 'Candidate B']):
        contingency_table[i,j] = np.sum((age_groups == age) & (voter_preferences == candidate))

# Perform chi-square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-square statistic: {chi2:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Significant association between age groups and voter preferences")
else:
    print("Conclusion: No significant association between age groups and voter preferences")
```

```
Chi-square statistic: 0.8780
P-value: 0.6447
Conclusion: No significant association between age groups and voter preferences
```

```
# 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 customers are
# located (East, West, North, South). The survey data is summarized 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 (rows) vs. Customer regions (columns)
#  data = np.array([[50, 30, 40, 20], [30, 40, 30, 50], [20, 30, 40, 30]]

import numpy as np
from scipy.stats import chi2_contingency

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

# Perform chi-square test
chi2, p_value, dof, expected = chi2_contingency(data)

print(f"Chi-square statistic: {chi2:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Significant relationship between satisfaction and region")
else:
    print("Conclusion: No significant relationship between satisfaction and region")
```

```
Chi-square statistic: 27.7771
P-value: 0.0001
Conclusion: Significant relationship between satisfaction and region
```

```
# 23.A company implemented an employee training program to improve job performance (Effective, Neutral,
# Ineffective). After the training, they 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
#  data = np.array([[50, 30, 20], [30, 40, 30], [20, 30, 40]])
#  ```
 # Sample data: Job performance levels before (rows) and after (columns) training

import numpy as np
from scipy.stats import chi2_contingency

# Sample data: Job performance before vs. after training
data = np.array([[50, 30, 20],
                 [30, 40, 30],
```

```
                [20, 30, 40]])

# Perform chi-square test
chi2, p_value, dof, expected = chi2_contingency(data)

print(f"Chi-square statistic: {chi2:.4f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("Conclusion: Significant difference in job performance after training")
else:
    print("Conclusion: No significant difference in job performance after training")
```

```
Chi-square statistic: 22.1617
P-value: 0.0002
Conclusion: Significant difference in job performance after 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
# difference in customer satisfaction scores.
#  Use the following data:
#  ```python
#  # 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

import numpy as np
from scipy import stats

# Sample data
standard = np.array([80, 85, 90, 78, 88, 82, 92, 78, 85, 87])
premium = np.array([90, 92, 88, 92, 95, 91, 96, 93, 89, 93])
deluxe = np.array([95, 98, 92, 97, 96, 94, 98, 97, 92, 99])
```