SHARDA UNIVERSITY
Beyond Boundaries

SHARDA SCHOOL OF BASIC SCIENCES AND RESEARCH

Department of Mathematics

LAB REPORT FILE

Course Title: Advanced Statistical Analysis

Course Code: VOM 202

Program-Bachelor of Science (Hons.)-Data Science

Semester: 4th

Session 2022-2023

Student Name: - Manish kumar

System ID: - 2021509877

Roll No. - 2107158014

Submitted to

Dr. Surya kant pal

Practical-1

❖ Aim:

Real life problem based on Discreet Probability Distributions (Uniform, binomial and Poisson) using python.

❖ Theory:

Python Bernoulli Distribution is a case of binomial distribution where we conduct a single experiment. This is a discrete probability distribution with probability p for value 1 and probability q=1-p for value 0. p can be for success, yes, true, or one. Similarly, q=1-p can be for failure, no, false, or zero.

❖ Discreet Uniform Probability Distributions:

A discrete uniform distribution is a probability distribution that has a finite set of equally likely outcomes. The probability of each outcome is the same, and the sum of all probabilities is equal to 1.

Implementing and visualizing uniform probability distribution in Python **using scipy** module.

❖ Types of discreet Probability Distributions:

- **Bernoulli Distribution:** A Bernoulli distribution is a discrete uniform distribution with only two possible outcomes, success (1) or failure (0).

- **Binomial Distribution:** A Binomial distribution is a discrete uniform distribution that models the number of successful outcomes in a fixed number of independent Bernoulli trials.

- **Poisson Distribution:** A Poisson distribution is a discrete uniform distribution that models the number of events that occur in a fixed interval of time or space.

- **Geometric Distribution:** A Geometric distribution is a discrete uniform distribution that models the number of independent Bernoulli trials needed to get the first success.

- **Hypergeometric Distribution:** A Hypergeometric distribution is a discrete uniform distribution that models the number of successful outcomes in a fixed number of trials, where the number of successes and failures are fixed and the trials are dependent.

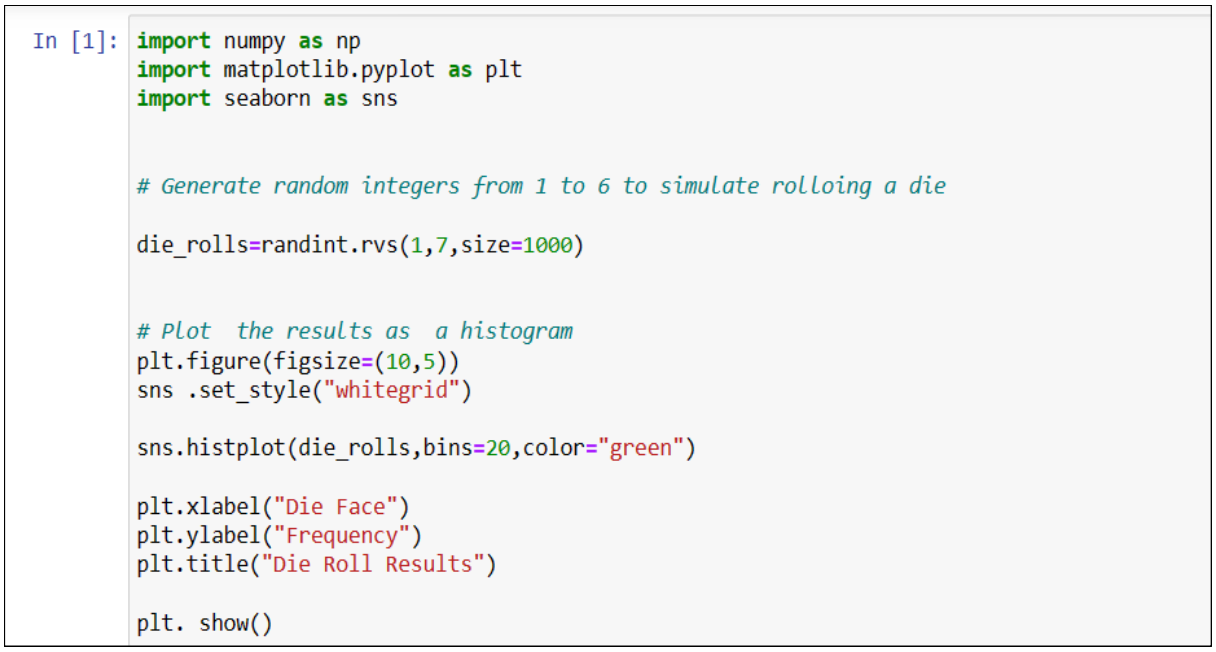
- **Negative Binomial Distribution:** A Negative Binomial distribution is a discrete uniform distribution that models the number of independent Bernoulli trials needed to get the rth success.

- **Uniform Distribution:** A Uniform distribution is a discrete uniform distribution that models the number of possible outcomes when all outcomes are equally likely.

Example 1:

A real-life example of a discrete uniform probability distribution is the rolling of a fair dice. Each outcome (rolling a 1, 2, 3, 4, 5, or 6) is equally likely, with probability of 1/6. Here's a code in Python that generates random numbers with a discrete uniform distribution:

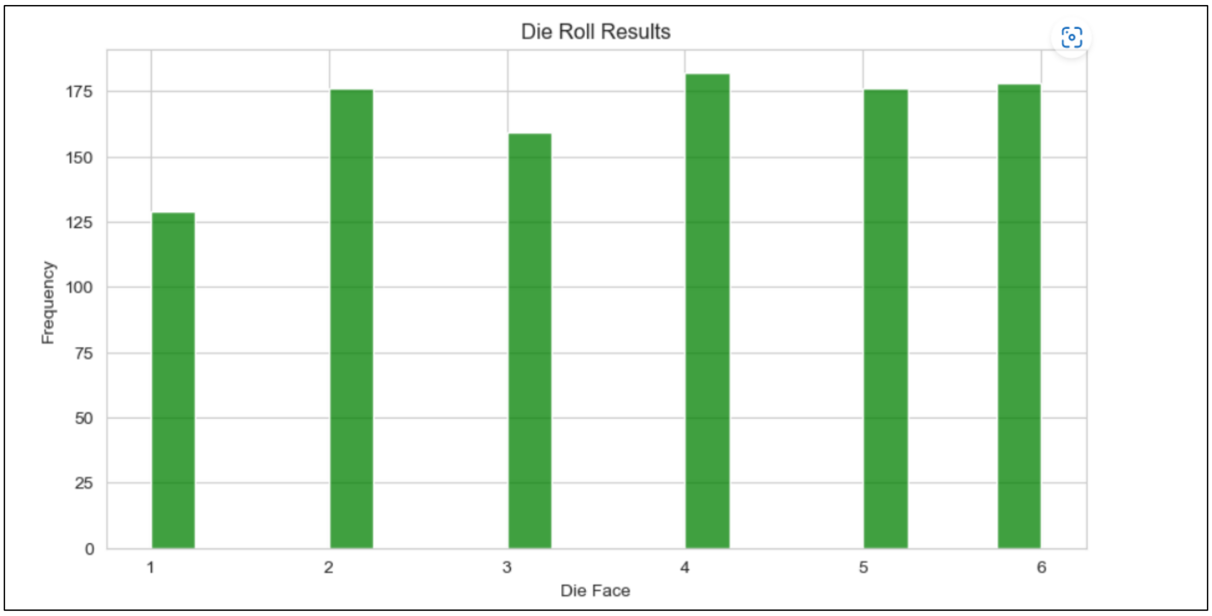
Code:



In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Generate random integers from 1 to 6 to simulate rolloing a die

die_rolls=randint.rvs(1,7,size=1000)


# Plot  the results as  a histogram
plt.figure(figsize=(10,5))
sns .set_style("whitegrid")

sns.histplot(die_rolls,bins=20,color="green")

plt.xlabel("Die Face")
plt.ylabel("Frequency")
plt.title("Die Roll Results")

plt. show()
```

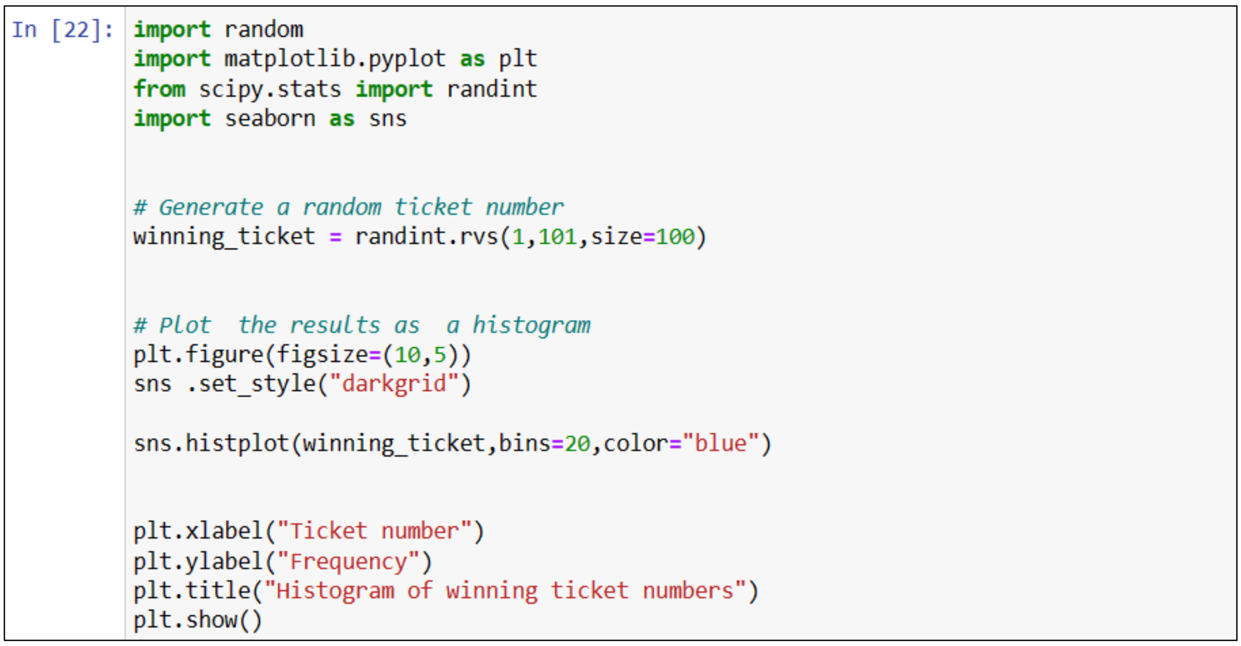
Output:



Example 2:

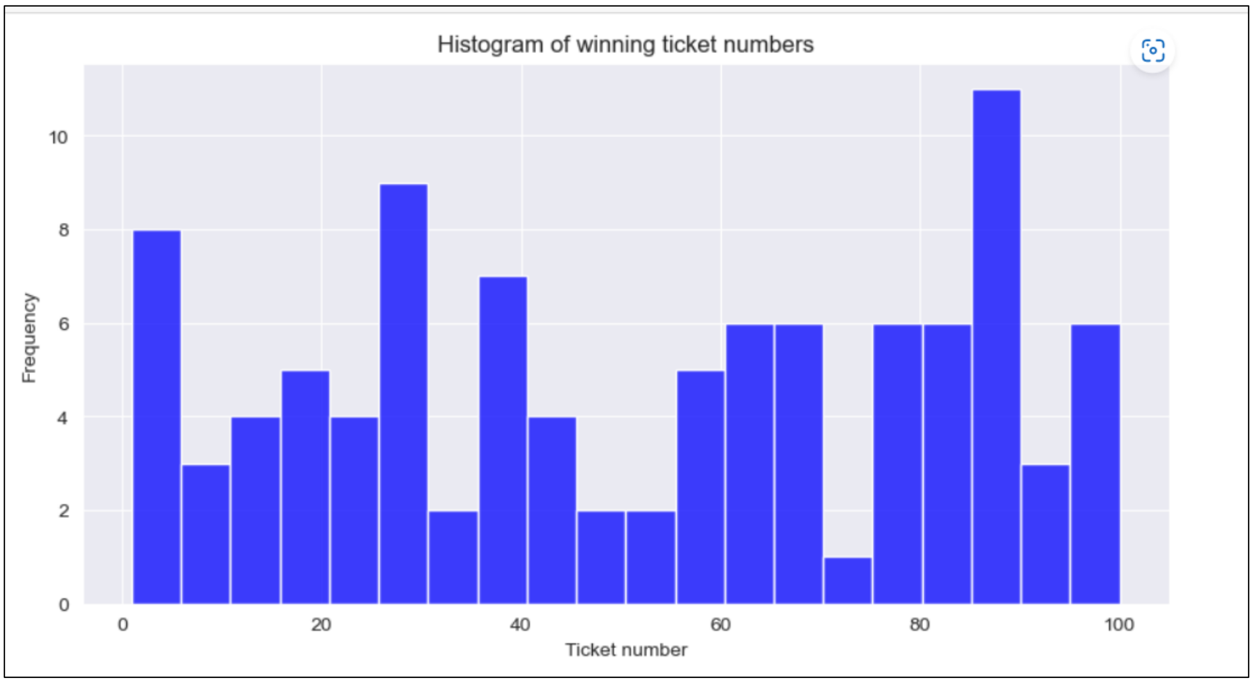
A real-life example of a discrete uniform probability distribution is the Choosing a random number between 1 and the number of entries in a raffle, where each ticket has an equal chance of being selected.

➢ **Code:**



```
In [22]: import random
         import matplotlib.pyplot as plt
         from scipy.stats import randint
         import seaborn as sns


         # Generate a random ticket number
         winning_ticket = randint.rvs(1,101,size=100)


         # Plot  the results as  a histogram
         plt.figure(figsize=(10,5))
         sns .set_style("darkgrid")

         sns.histplot(winning_ticket,bins=20,color="blue")


         plt.xlabel("Ticket number")
         plt.ylabel("Frequency")
         plt.title("Histogram of winning ticket numbers")
         plt.show()
```

➢ **Output:**



- **Discreet binomial Probability Distributions:**

The binomial distribution is a discrete probability distribution that describes the number of successes in a fixed number of independent trials, where each trial has the same probability of success. The distribution is defined by two parameters: the number of trials (n) and the probability of success in each trial (p). The probability of getting exactly k successes in n trials can be calculated using the following formula:

P(X = k) = (n choose k) * p^k * (1-p)^(n-k)

where:

P(X = k) is the probability of getting k successes in n trials
n is the total number of trials
k is the number of successes
p is the probability of success on each trial
(n choose k) is the binomial coefficient, which is calculated as n! / (k! * (n-k)!)

Here, (n choose k) is the binomial coefficient, which is the number of ways to choose k items from a set of n items. This formula gives the probability mass function for the binomial distribution, which gives the probability of getting exactly k successes in n trials.

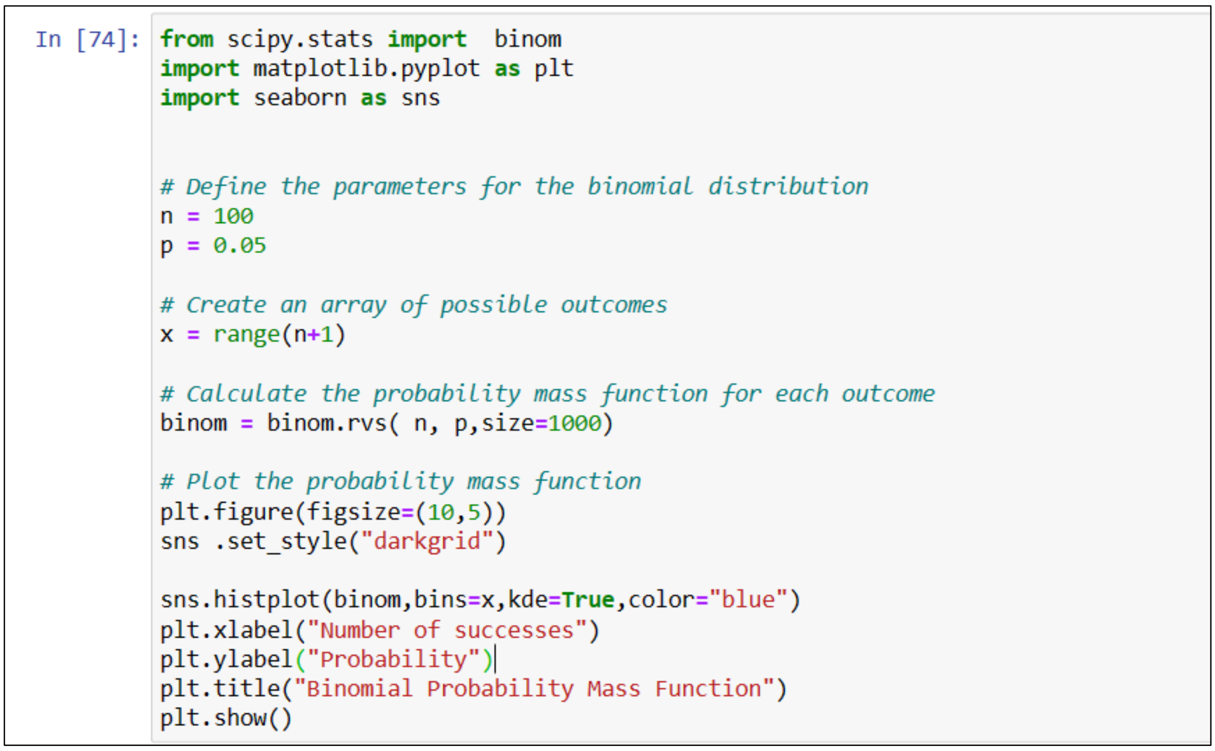
The binomial distribution is used in a variety of applications, such as:

- ✓ Modeling the number of successful trades in a fixed number of stock market trades with a certain probability of success
- ✓ Predicting the number of defective items in a sample of products from a manufacturing process that has a known defect rate
- ✓ Estimating the probability of a certain number of visitors clicking on an ad in a fixed number of online impressions, with a certain click-through rate

Example 1: Quality Control in Manufacturing

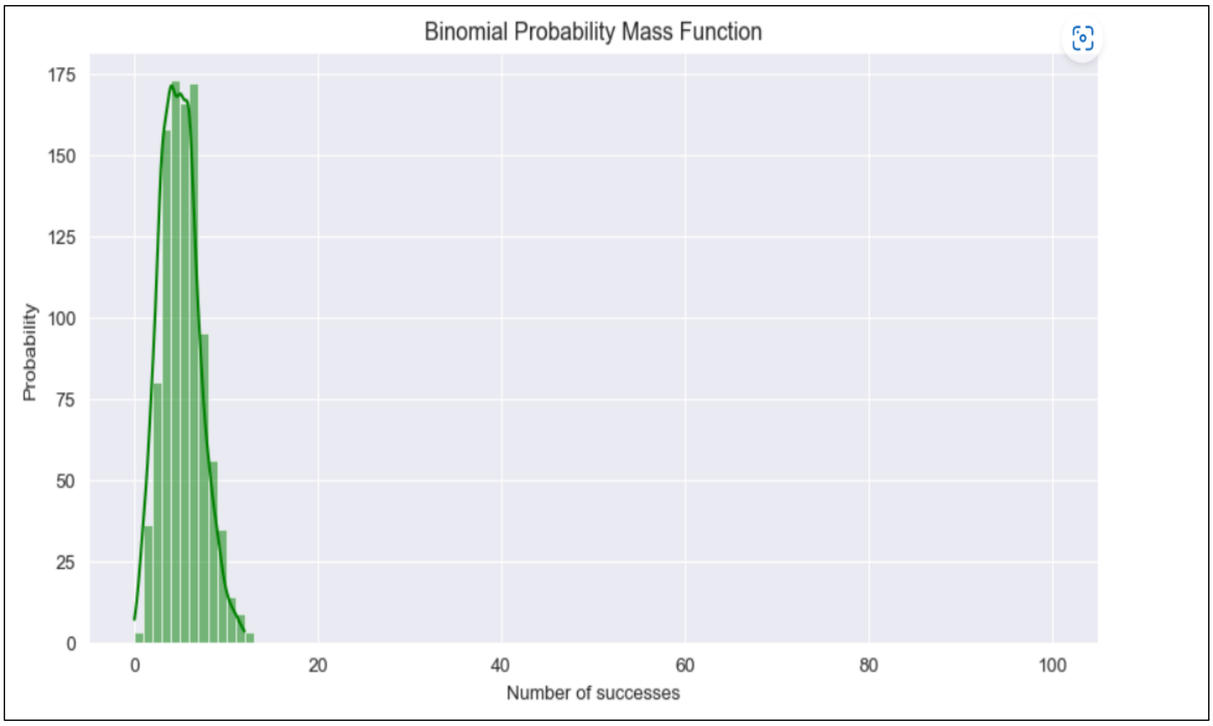
Suppose a factory produces light bulbs, and the probability that any one light bulb is defective is 0.05. The factory produces a batch of 100 light bulbs. What is the probability that exactly 5 of the light bulbs in the batch are defective?

➢ **Code:**



```
In [74]: from scipy.stats import  binom
         import matplotlib.pyplot as plt
         import seaborn as sns


         # Define the parameters for the binomial distribution
         n = 100
         p = 0.05

         # Create an array of possible outcomes
         x = range(n+1)

         # Calculate the probability mass function for each outcome
         binom = binom.rvs( n, p,size=1000)

         # Plot the probability mass function
         plt.figure(figsize=(10,5))
         sns .set_style("darkgrid")

         sns.histplot(binom,bins=x,kde=True,color="blue")
         plt.xlabel("Number of successes")
         plt.ylabel("Probability")
         plt.title("Binomial Probability Mass Function")
         plt.show()
```

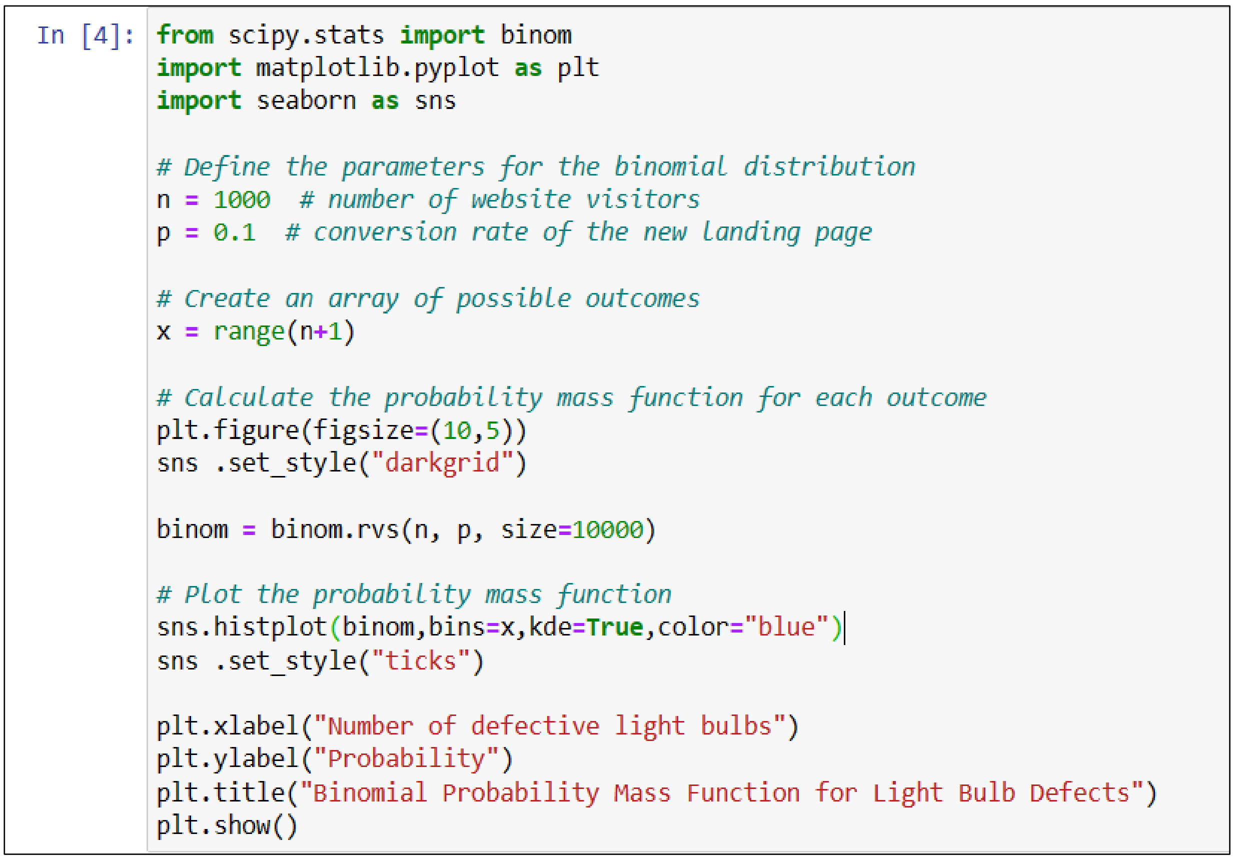
➢ **Output:**



Example 2: Website Conversion Rates

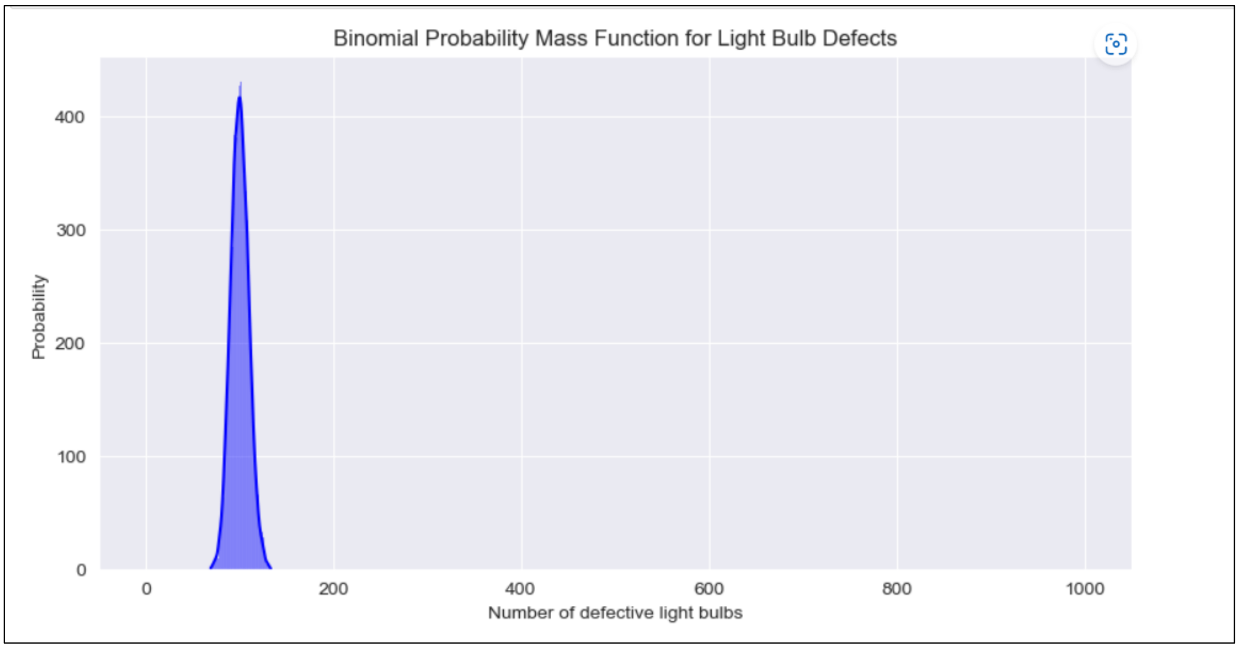
Suppose a website owner is running an experiment to test the effectiveness of a new landing page design. The website receives 1000 visitors, and the owner wants to know the probability that exactly 100 of the visitors will convert (i.e., take a desired action, such as making a purchase).

- **Code:**



```
In [4]: from scipy.stats import binom
        import matplotlib.pyplot as plt
        import seaborn as sns

        # Define the parameters for the binomial distribution
        n = 1000  # number of website visitors
        p = 0.1  # conversion rate of the new landing page

        # Create an array of possible outcomes
        x = range(n+1)

        # Calculate the probability mass function for each outcome
        plt.figure(figsize=(10,5))
        sns .set_style("darkgrid")

        binom = binom.rvs(n, p, size=10000)

        # Plot the probability mass function
        sns.histplot(binom,bins=x,kde=True,color="blue")
        sns .set_style("ticks")

        plt.xlabel("Number of defective light bulbs")
        plt.ylabel("Probability")
        plt.title("Binomial Probability Mass Function for Light Bulb Defects")
        plt.show()
```

- **Output:**



❖ Poisson Distributions:

The Poisson distribution is a discrete probability distribution that is often used to model the number of rare events that occur within a fixed interval of time or space, given a known average rate of occurrence.

The probability mass function for a Poisson distribution is given by:

P(X = k) = (e^-λ * λ^k) / k!

where X is the number of events, λ is the average rate of occurrence, e is the mathematical constant e, and k is the number of events that occur in the interval.

The Poisson distribution has only one parameter, λ, which is the mean and also the variance of the distribution. The Poisson distribution is characterized by the following properties:

- ✓ The number of events that occur in a given interval is independent of the number of events that occur in any other interval.
- ✓ The average rate of occurrence is constant over time or space.
- ✓ The probability of more than one event occurring in a very short interval is virtually zero.

The Poisson distribution is used in many fields, including:

- ✓ Epidemiology: to model the number of cases of a disease in a population over time.
- ✓ Finance: to model the number of loan defaults or insurance claims in a given period.
- ✓ Manufacturing: to model the number of defective products in a production run.

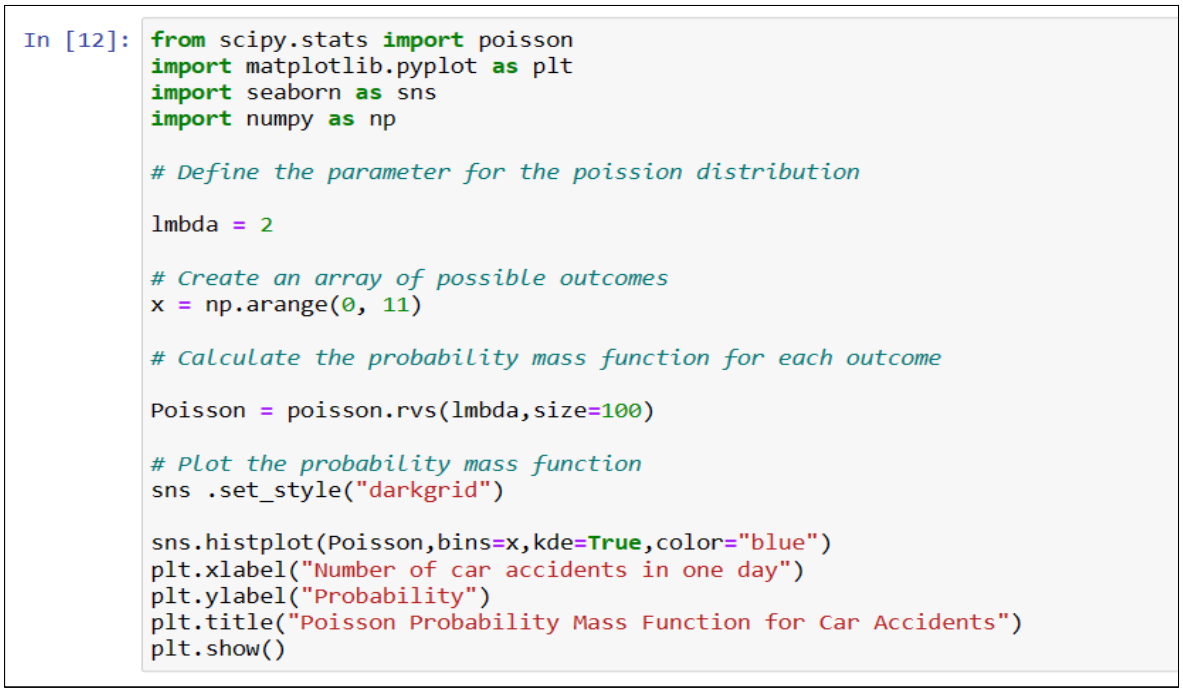
In Python, the **scipy.stats module** provides functions for working with the Poisson distribution. For example, the **scipy.stats.poisson.pmf** function can be used to calculate the probability mass function for a given value of λ and k, and the **scipy.stats.poisson.cdf** function can be used to calculate the cumulative distribution function for a given value of λ and k. The matplotlib library can be used to create plots of the Poisson distribution.

The Poisson distribution is used to model the number of events that occur in a fixed interval of time or space, given a known average rate of occurrence. Here are two examples of the Poisson distribution in real life, along with code to plot the probability mass functions for each example:

Example 1: Car Accidents

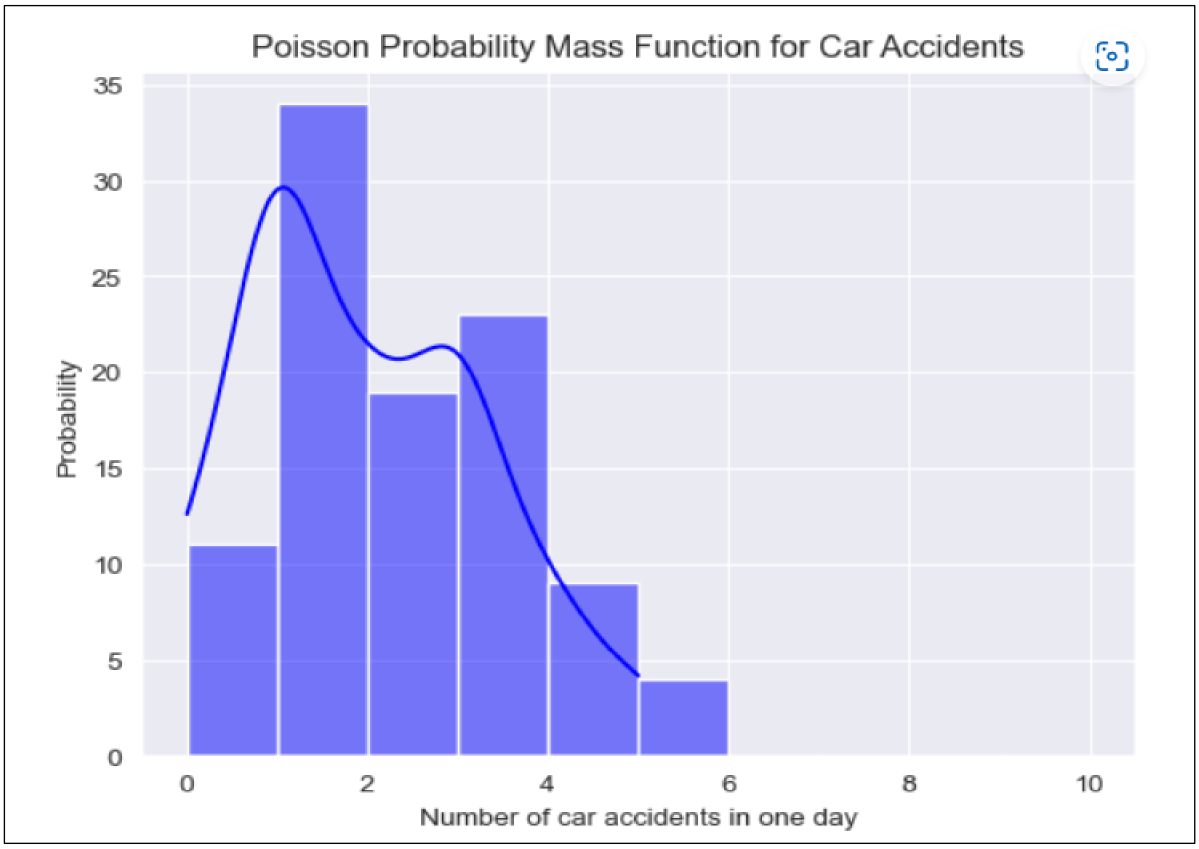
Suppose that in a given city, car accidents occur at an average rate of 2 per day. What is the probability that there will be exactly 4 car accidents in the city tomorrow?

➢ **Code:**



```
In [12]: from scipy.stats import poisson
         import matplotlib.pyplot as plt
         import seaborn as sns
         import numpy as np

         # Define the parameter for the poission distribution

         lmbda = 2

         # Create an array of possible outcomes
         x = np.arange(0, 11)

         # Calculate the probability mass function for each outcome

         Poisson = poisson.rvs(lmbda,size=100)

         # Plot the probability mass function
         sns .set_style("darkgrid")

         sns.histplot(Poisson,bins=x,kde=True,color="blue")
         plt.xlabel("Number of car accidents in one day")
         plt.ylabel("Probability")
         plt.title("Poisson Probability Mass Function for Car Accidents")
         plt.show()
```

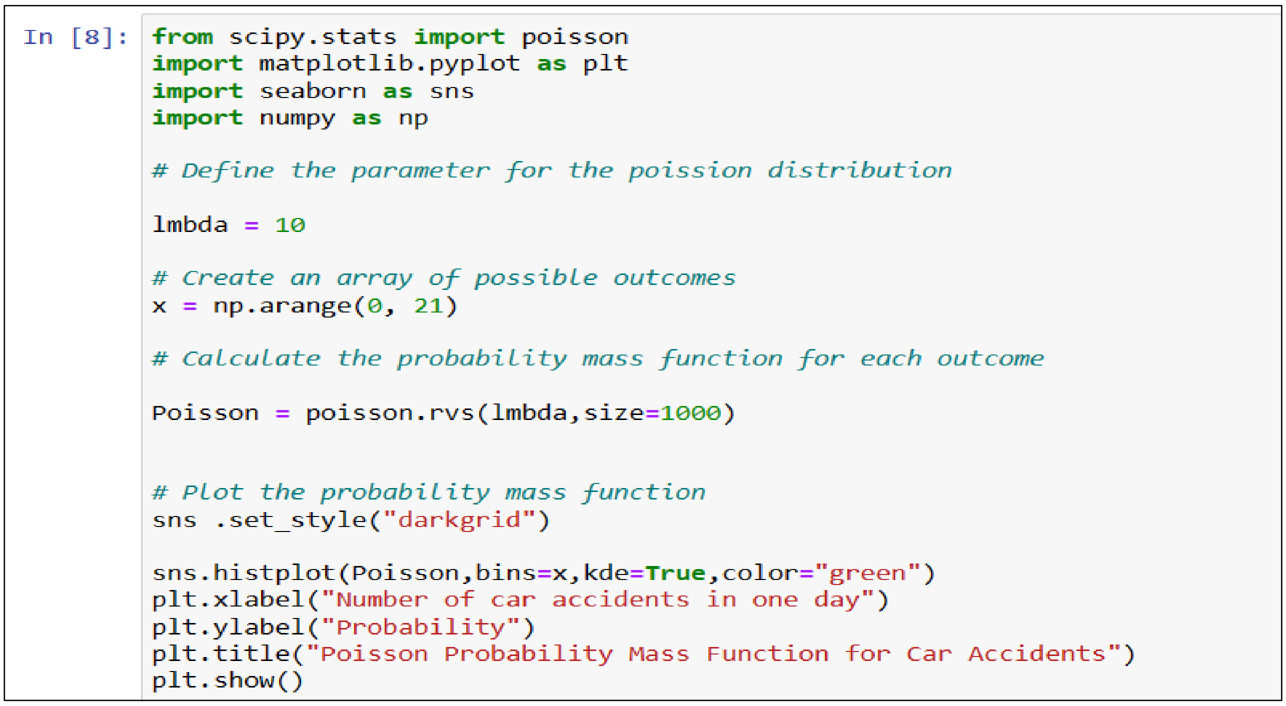
➢ **Output**:



Example 2: Call Center Wait Times

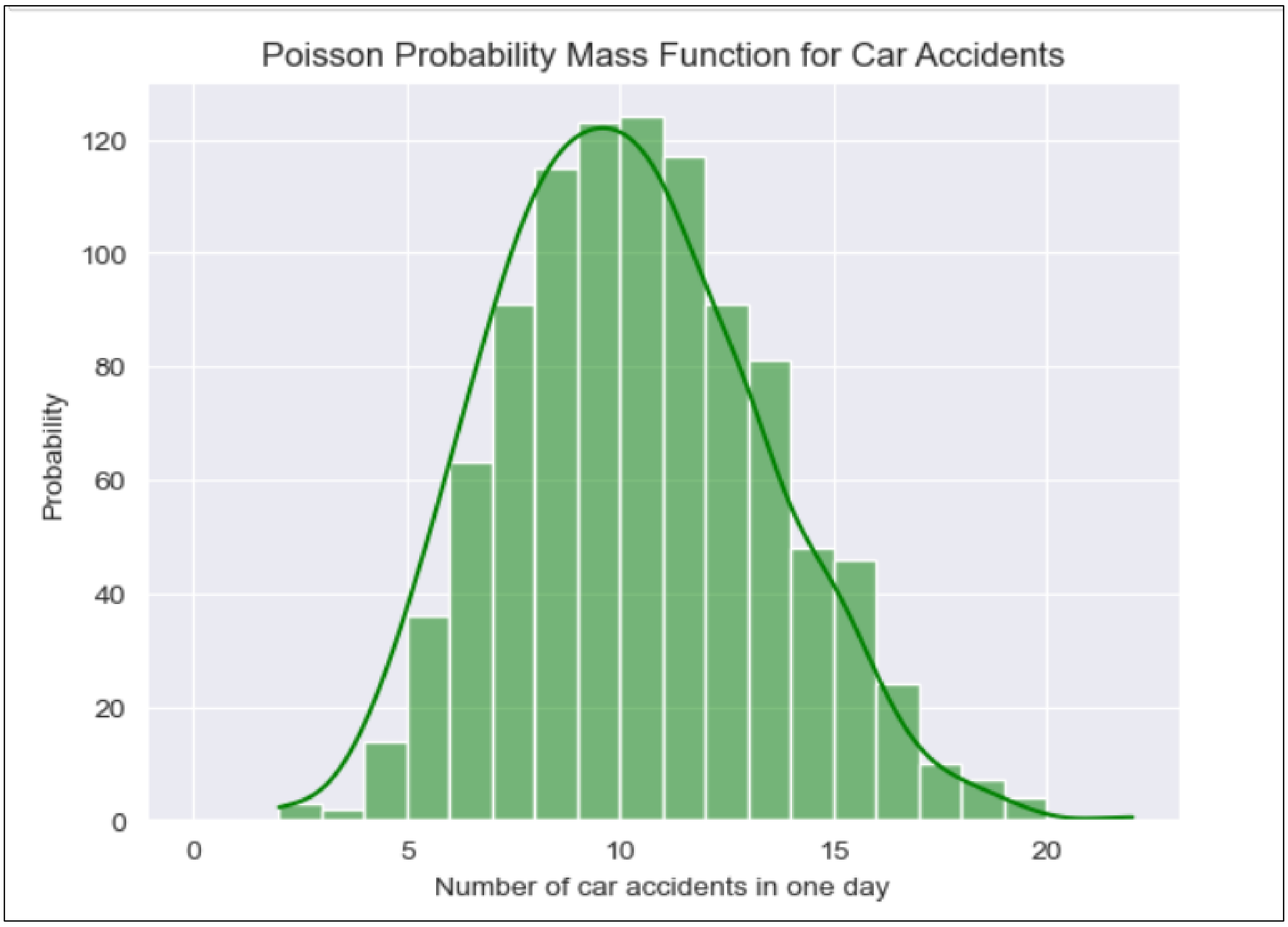
Suppose a call center receives an average of 5 calls per minute. What is the probability that the center will receive more than 10 calls in a 2-minute period?

➢ **Code:**



```
In [8]: from scipy.stats import poisson
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np

        # Define the parameter for the poission distribution

        lmbda = 10

        # Create an array of possible outcomes
        x = np.arange(0, 21)

        # Calculate the probability mass function for each outcome

        Poisson = poisson.rvs(lmbda,size=1000)


        # Plot the probability mass function
        sns .set_style("darkgrid")

        sns.histplot(Poisson,bins=x,kde=True,color="green")
        plt.xlabel("Number of car accidents in one day")
        plt.ylabel("Probability")
        plt.title("Poisson Probability Mass Function for Car Accidents")
        plt.show()
```

➢ **OUTPUT:**



Practical-2

❖ Aim:

Real life problem based on Continuous Probability Distributions (Normal, Exponential and Gamma) using python.

❖ Theory:

Normal Distribution:

The normal or Gaussian distribution is a continuous probability distribution characterized by a symmetric bell-shaped curve. A normal distribution is defined by its center (mean) and spread (standard deviation.). The bulk of the observations generated from a normal distribution lie near the mean, which lies at the exact center of the distribution: as a rule of thumb, about 68% of the data lies within 1 standard deviation of the mean, 95% lies within 2 standard deviations and 99.7% lies within 3 standard deviations.

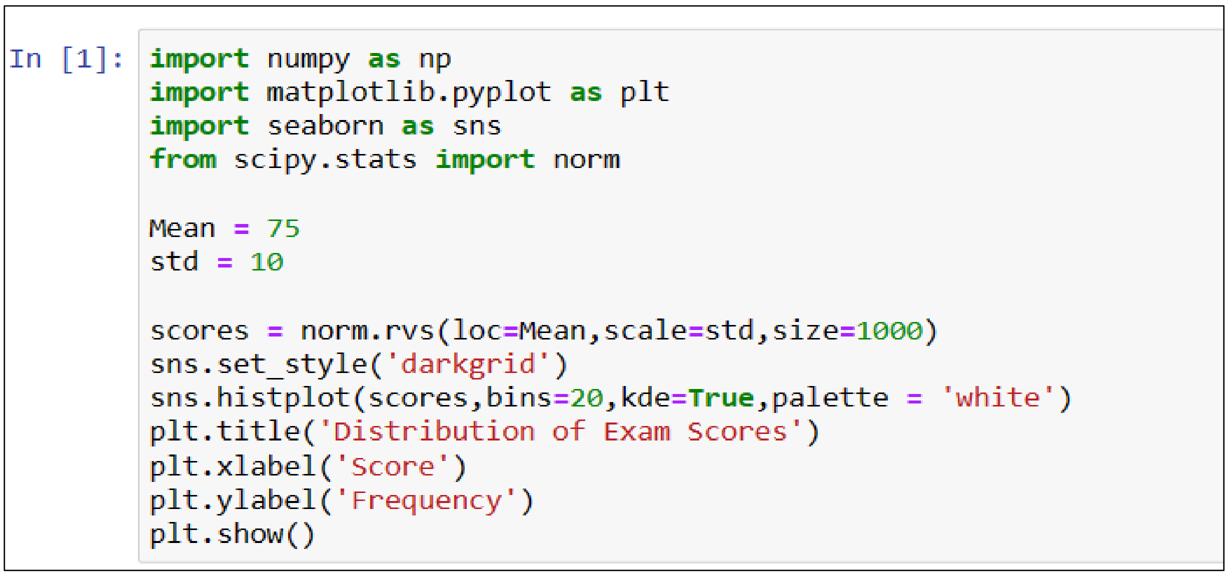
The normal distribution is perhaps the most important distribution in all of statistics. It turns out that many real world phenomena, like IQ test scores and human heights, roughly follow a normal distribution, so it is often used to model random variables. Many common statistical tests assume distributions are normal.

The scipy nickname for the normal distribution is norm. Let's investigate the normal distribution:

Example 1: Exam Scores

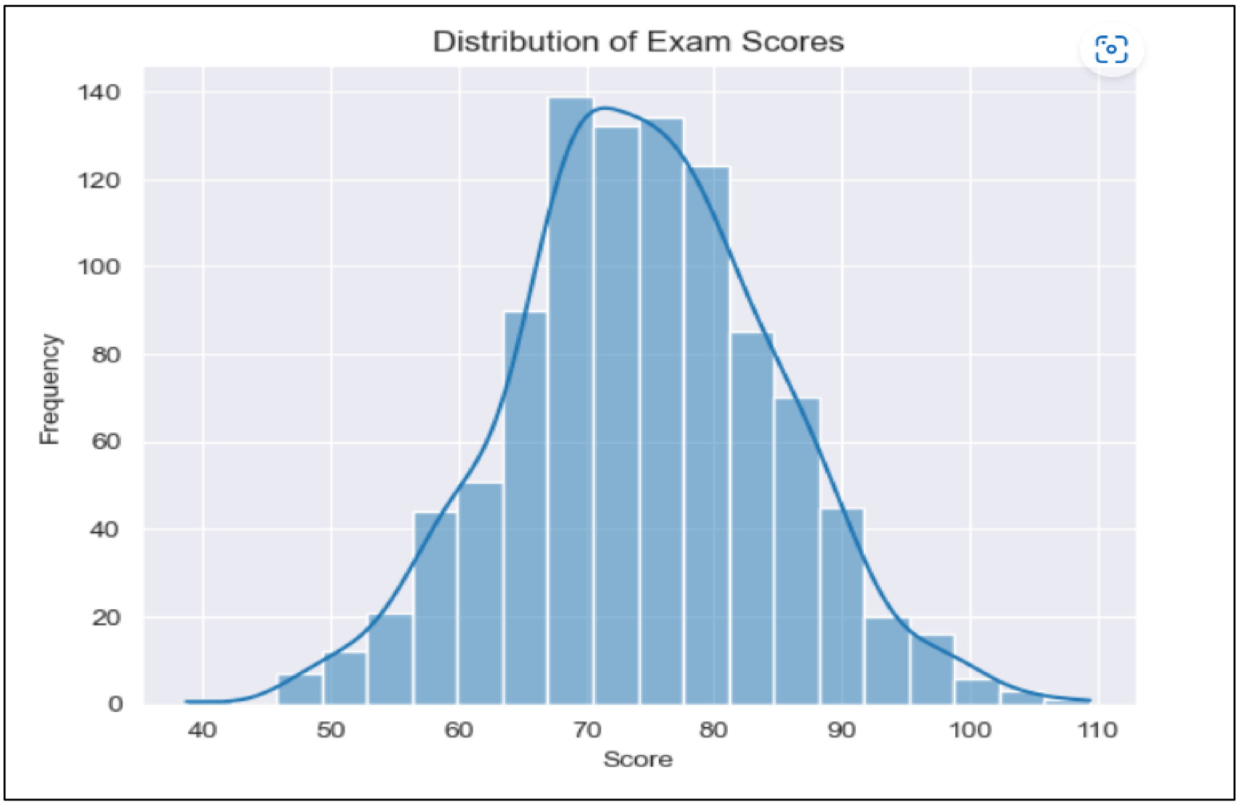
Suppose we want to model the distribution of exam scores for a particular class using a normal distribution. Let's say that the mean score on the exam is 75, and the standard deviation is 10. We can use Python to generate a random sample of exam scores from a normal distribution with these parameters, and plot the resulting distribution using a histogram.

➢ Code:



```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from scipy.stats import norm

        Mean = 75
        std = 10

        scores = norm.rvs(loc=Mean,scale=std,size=1000)
        sns.set_style('darkgrid')
        sns.histplot(scores,bins=20,kde=True,palette = 'white')
        plt.title('Distribution of Exam Scores')
        plt.xlabel('Score')
        plt.ylabel('Frequency')
        plt.show()
```

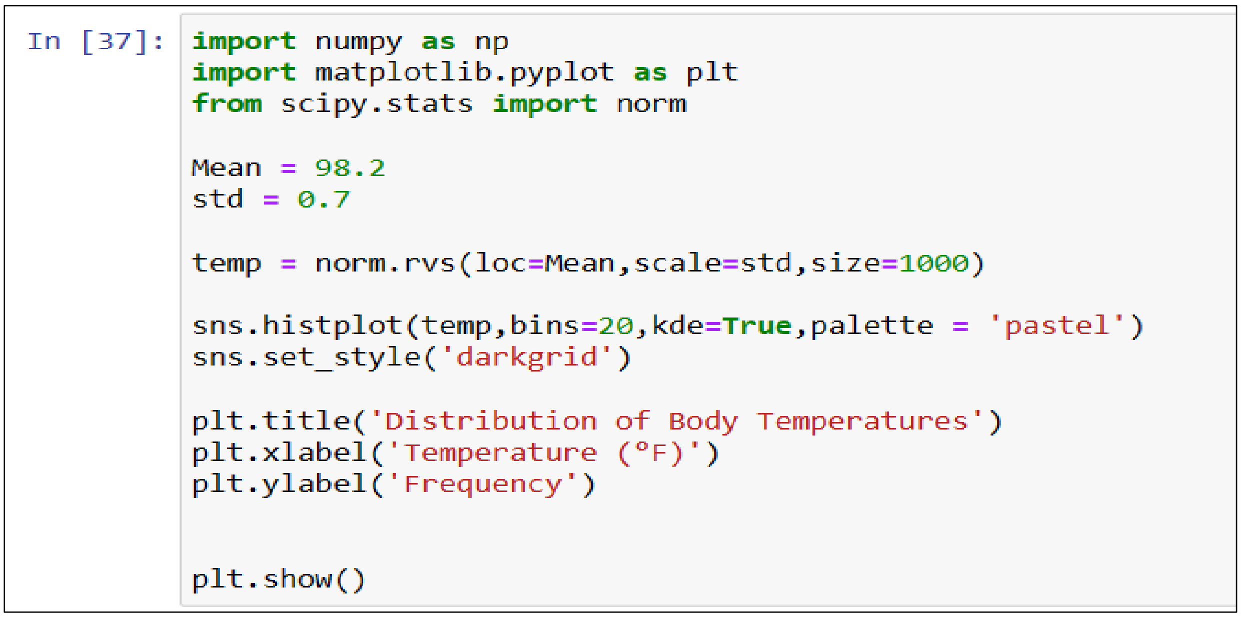
➢ **Output:**



Example 2: Body Temperature

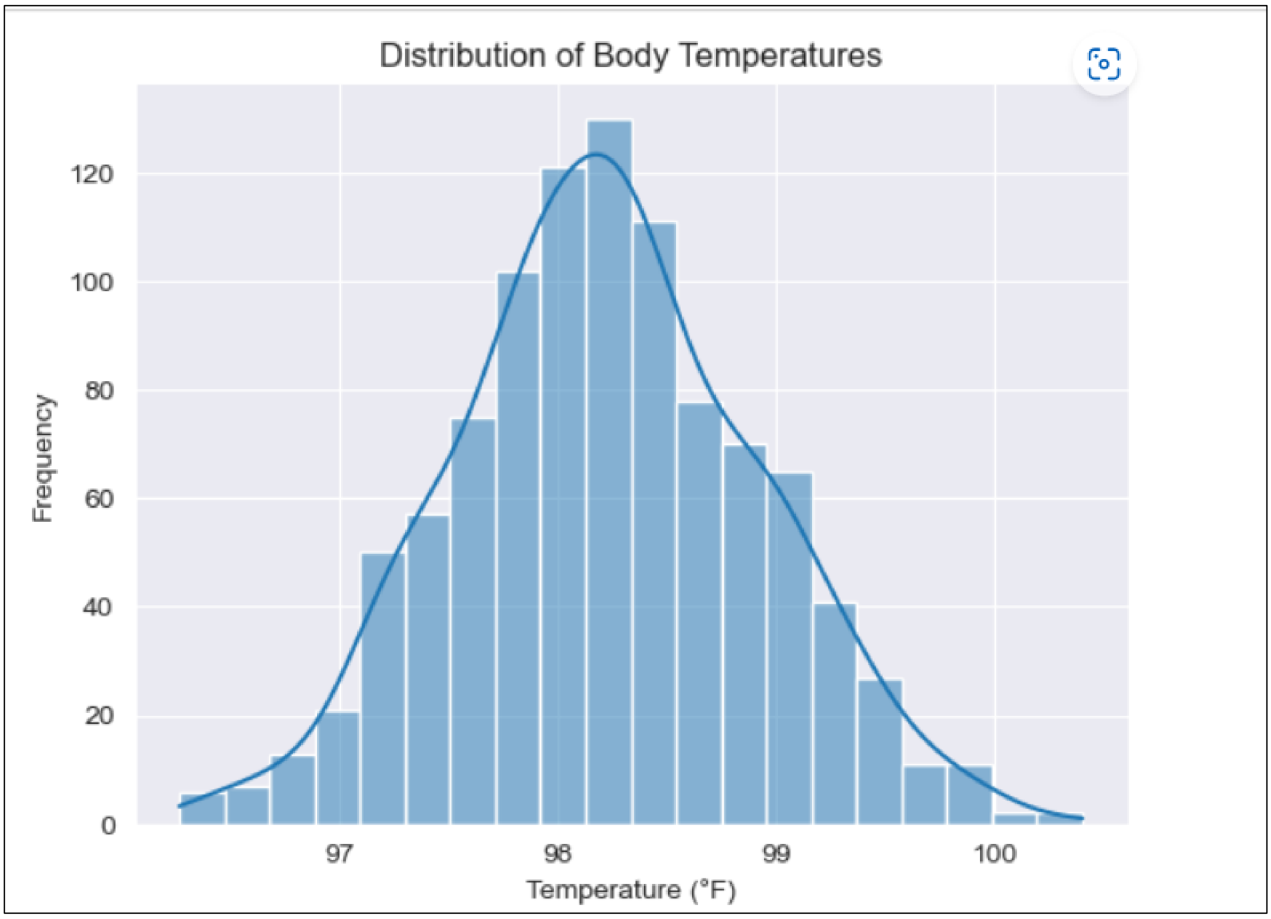
Suppose we want to model the distribution of body temperatures for a large population using a normal distribution. According to a study published in the Journal of the American Medical Association (JAMA), the mean body temperature of healthy adults is 98.2°F, with a standard deviation of 0.7°F. We can use the SciPy library to generate a random sample of body temperatures from a normal distribution with these parameters, and plot the resulting distribution using a histogram.

➢ **Code:**



```
In [37]: import numpy as np
         import matplotlib.pyplot as plt
         from scipy.stats import norm

         Mean = 98.2
         std = 0.7

         temp = norm.rvs(loc=Mean,scale=std,size=1000)

         sns.histplot(temp,bins=20,kde=True,palette = 'pastel')
         sns.set_style('darkgrid')

         plt.title('Distribution of Body Temperatures')
         plt.xlabel('Temperature (°F)')
         plt.ylabel('Frequency')


         plt.show()
```

➢ **Output:**



➢ Exponential Distribution:

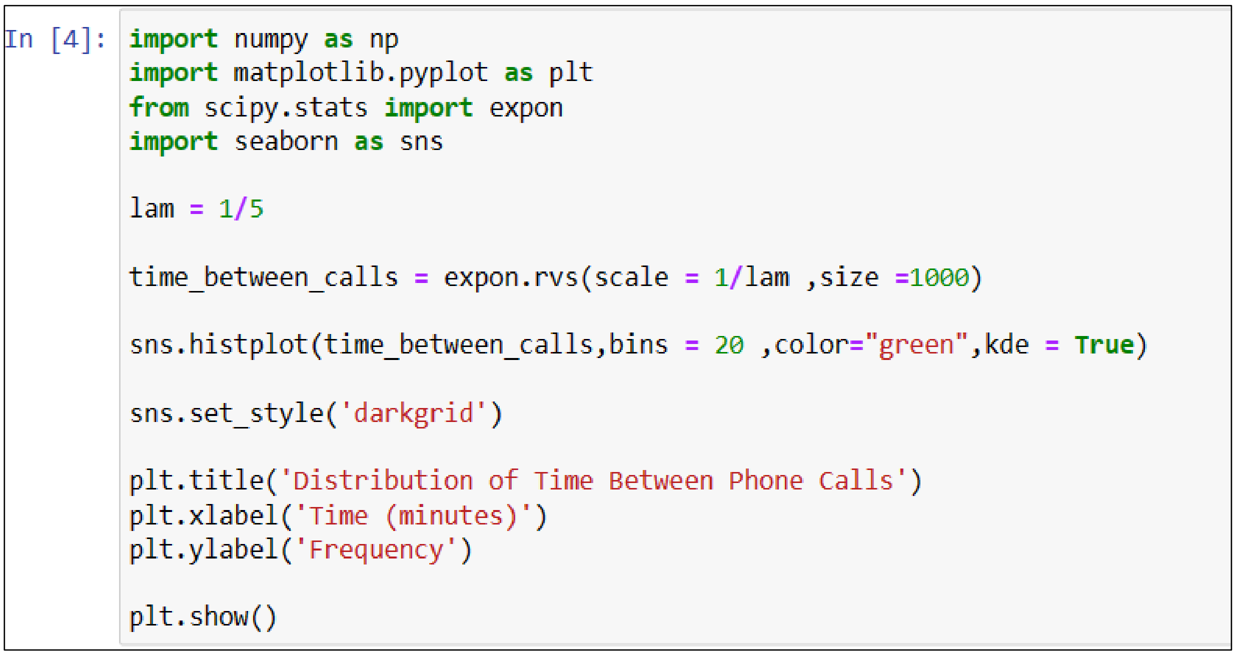
Exponential probability distribution is a type of continuous probability distribution that models the time between events in a Poisson process, where events occur independently and at a constant average rate. It is often used to model phenomena such as waiting times, survival times, and time between occurrences of rare events.

You can generate an exponentially distributed random variable using scipy.stats module's expon.rvs() method which takes shape parameter scale as its argument which is nothing but 1/lambda in the equation. To shift distributions use the loc argument, size decides the number of random variates in the distribution. If you want to maintain reproducibility, include a random_state argument assigned to a number.

Example 1: Interarrival Times

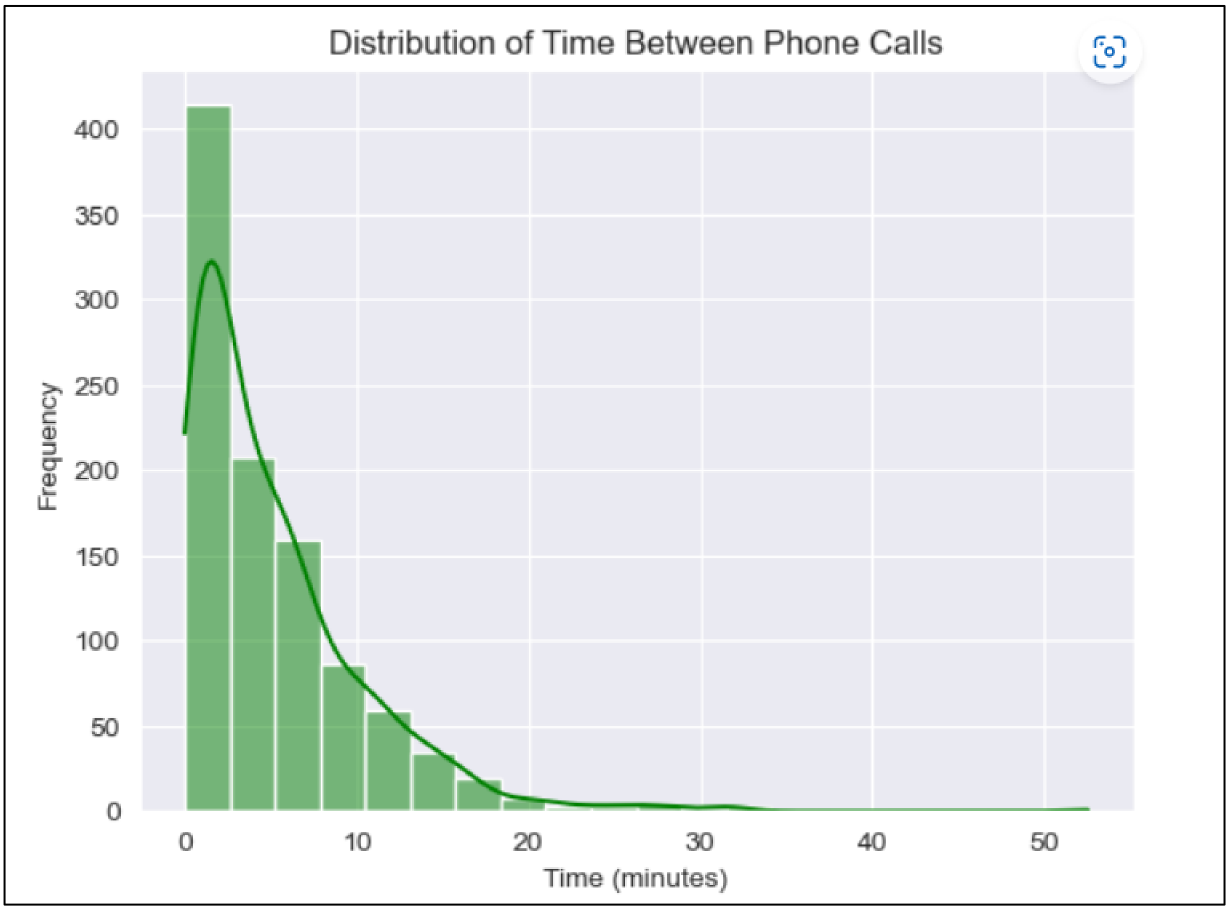
Suppose we want to model the distribution of interarrival times between customers at a store using an exponential distribution. Let's say that the average time between arrivals is 2 minutes. We can use the SciPy library to generate a random sample of interarrival times from an exponential distribution with this parameter, and plot the resulting distribution using a histogram

➢ Code:



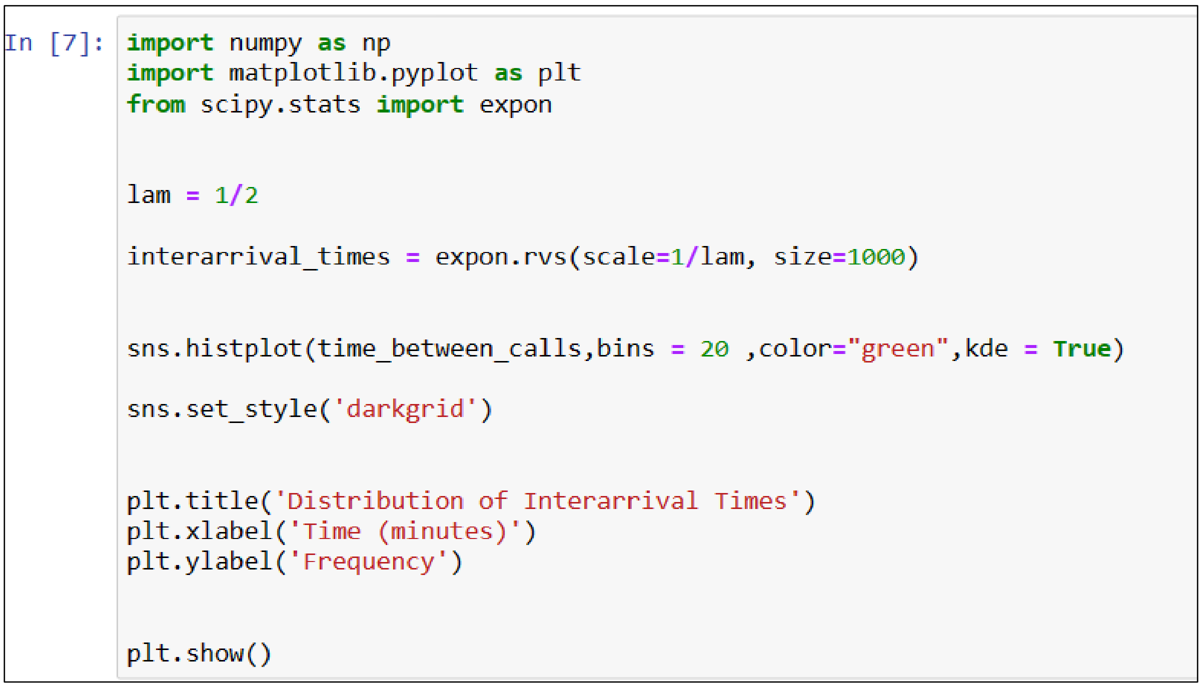
```
In [4]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.stats import expon
        import seaborn as sns

        lam = 1/5

        time_between_calls = expon.rvs(scale = 1/lam ,size =1000)

        sns.histplot(time_between_calls,bins = 20 ,color="green",kde = True)

        sns.set_style('darkgrid')

        plt.title('Distribution of Time Between Phone Calls')
        plt.xlabel('Time (minutes)')
        plt.ylabel('Frequency')

        plt.show()
```

➢ Output:



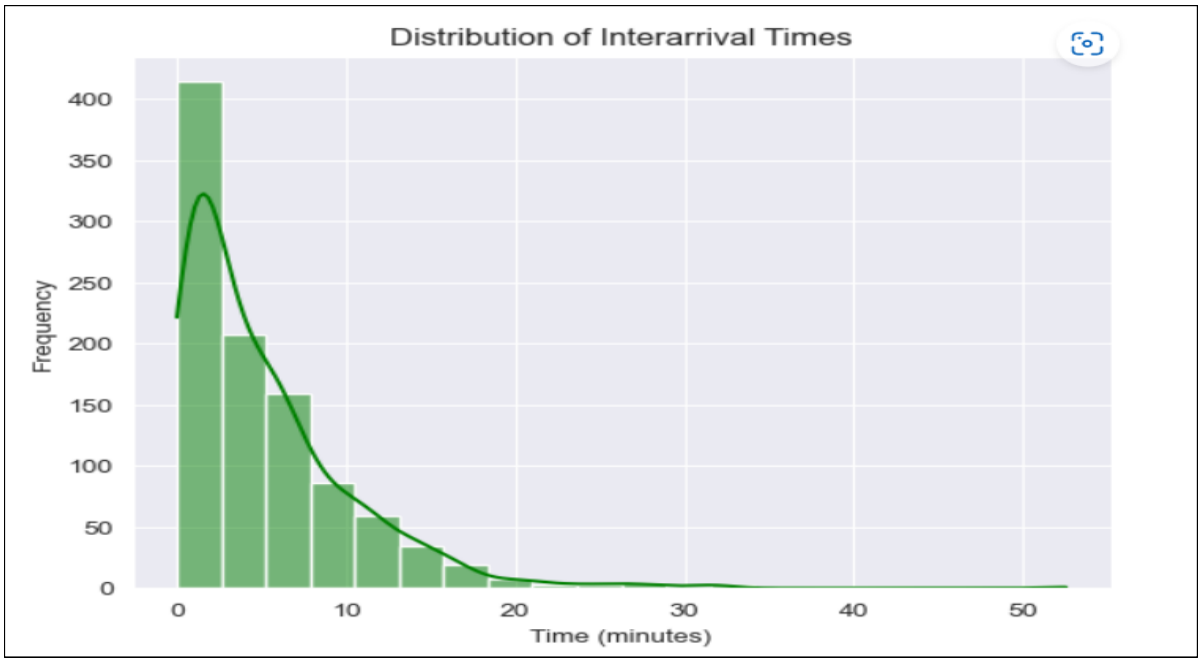
- **Example 2: Time Between Phone Calls**
  Suppose we want to model the distribution of time between phone calls received by a call center using an exponential distribution. Let's say that the average time between calls is 5 minutes. We can use the SciPy library to generate a random sample of time between calls from an exponential distribution with this parameter, and plot the resulting distribution using a histogram.

- **Code:**



```
In [7]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.stats import expon


        lam = 1/2

        interarrival_times = expon.rvs(scale=1/lam, size=1000)


        sns.histplot(time_between_calls,bins = 20 ,color="green",kde = True)

        sns.set_style('darkgrid')


        plt.title('Distribution of Interarrival Times')
        plt.xlabel('Time (minutes)')
        plt.ylabel('Frequency')


        plt.show()
```

- **Output:**



➢ Gamma Distribution:

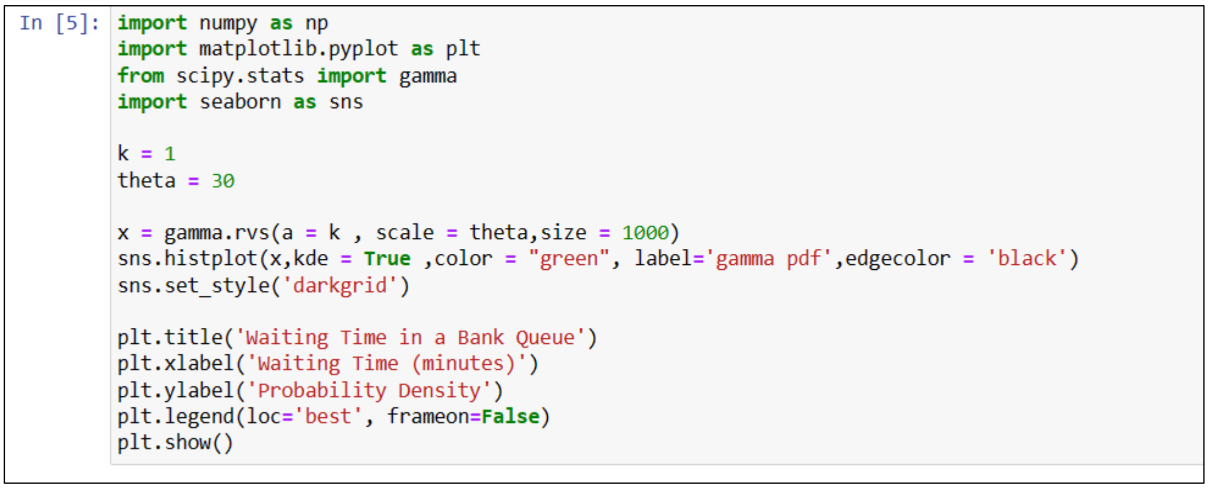
The gamma distribution is a two-parameter family of continuous probability distributions. It can be thought of as describing the waiting time until a certain number of events occur in a Poisson process with a given average rate.

The gamma distribution has two parameters, the shape k and scale theta. The PDF is f(x)=(x^{k-1}e^{-x/theta})/(Gamma(k)theta^k). Gamma refers to the gamma function.

➢ Example 1: Waiting Time in a Queue:

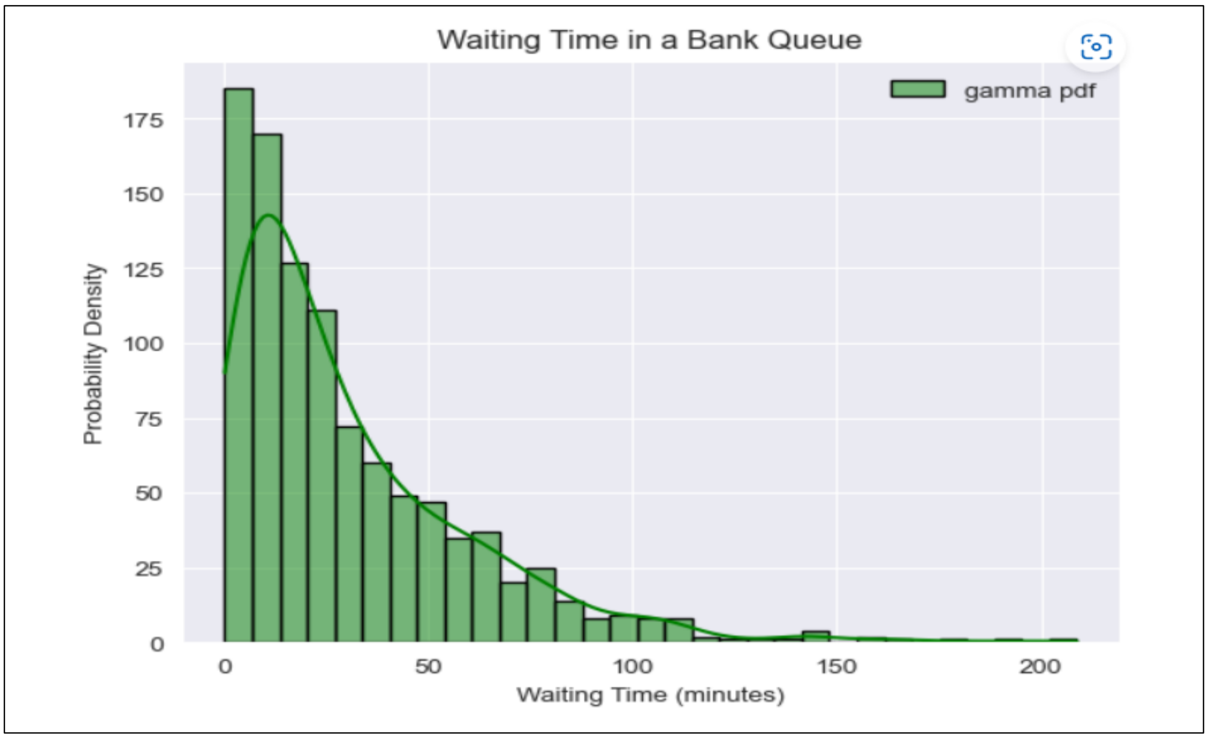
Suppose customers arrive at a bank according to a Poisson process with a rate of 10 customers per hour. The bank has only one teller, and the service time for each customer is exponentially distributed with a mean of 5 minutes. The waiting time for a customer in the queue can be modeled by a Gamma distribution with parameters k=1 and θ=30 minutes (which is the sum of the arrival and service times).

❖ Code:



```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.stats import gamma
        import seaborn as sns

        k = 1
        theta = 30

        x = gamma.rvs(a = k , scale = theta,size = 1000)
        sns.histplot(x,kde = True ,color = "green", label='gamma pdf',edgecolor = 'black')
        sns.set_style('darkgrid')

        plt.title('Waiting Time in a Bank Queue')
        plt.xlabel('Waiting Time (minutes)')
        plt.ylabel('Probability Density')
        plt.legend(loc='best', frameon=False)
        plt.show()
```

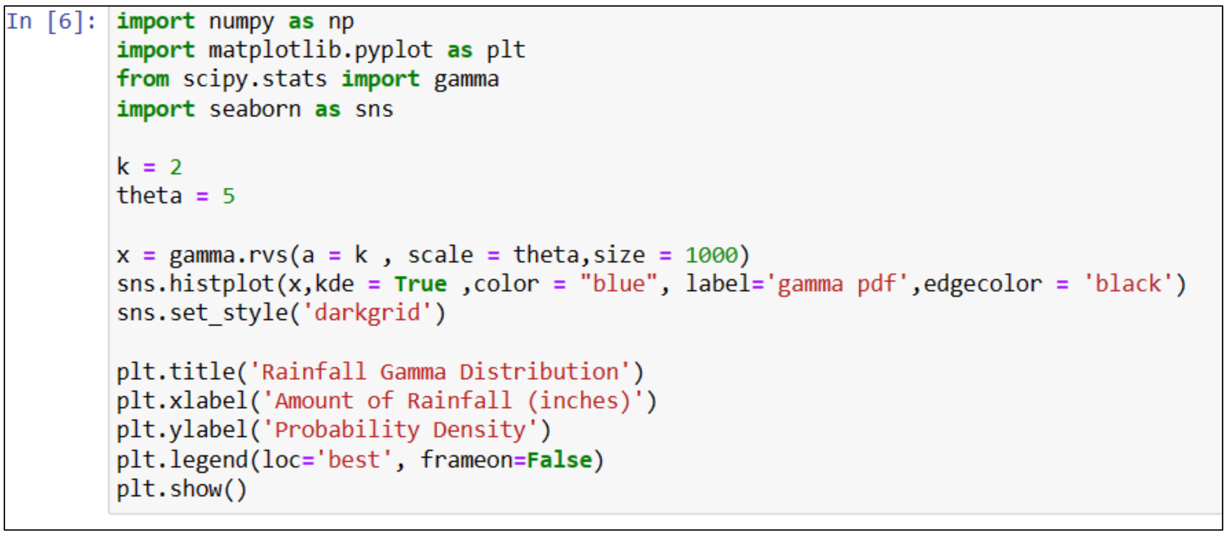
➢ Output:



➢ **Example 2: Rainfall:**

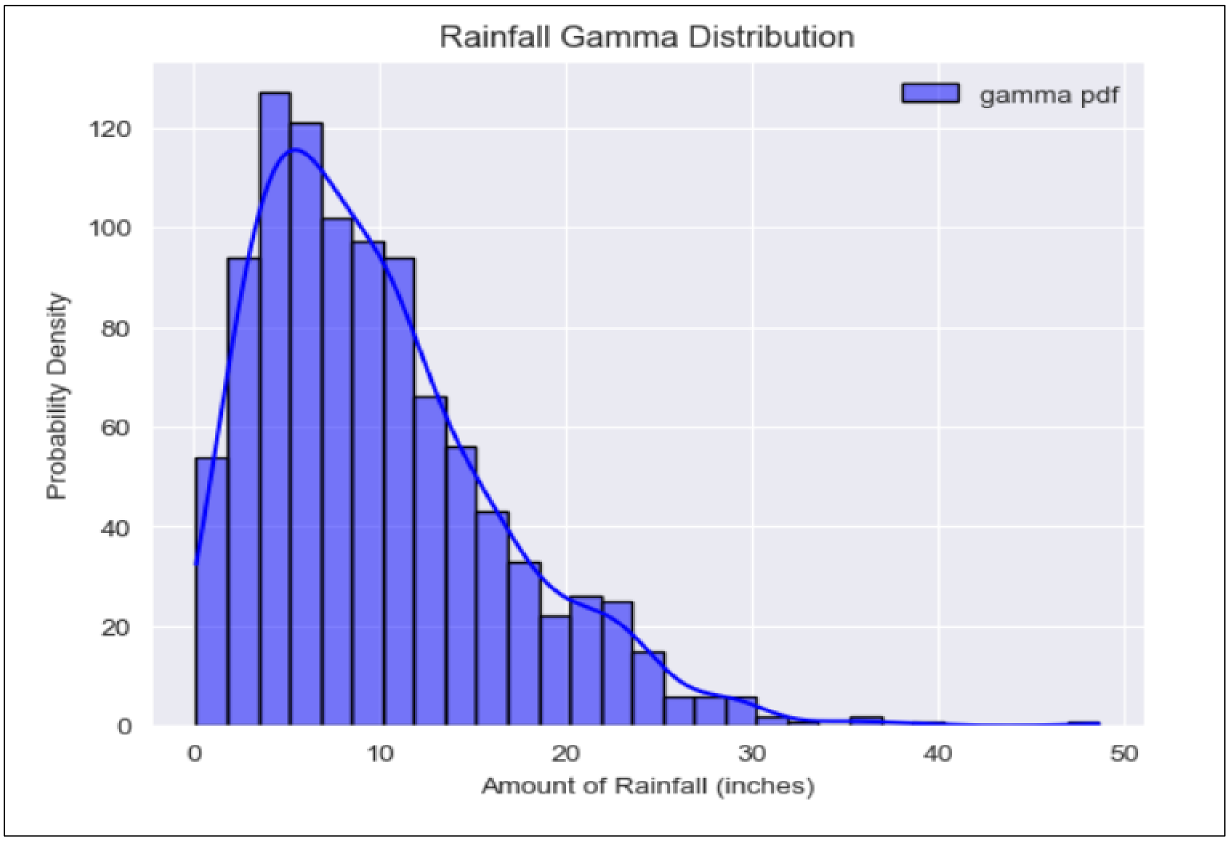
Suppose that the amount of rainfall in a certain region follows a Gamma distribution with parameters k=2 and θ=5 inches. This means that the rainfall is more likely to be moderate, with some heavy rainfall events occurring occasionally.

❖ **Code:**



```
In [6]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.stats import gamma
        import seaborn as sns

        k = 2
        theta = 5

        x = gamma.rvs(a = k , scale = theta,size = 1000)
        sns.histplot(x,kde = True ,color = "blue", label='gamma pdf',edgecolor = 'black')
        sns.set_style('darkgrid')

        plt.title('Rainfall Gamma Distribution')
        plt.xlabel('Amount of Rainfall (inches)')
        plt.ylabel('Probability Density')
        plt.legend(loc='best', frameon=False)
        plt.show()
```

❖ **Output:**



Practical-3

- **Aim:**

  Real life Problem based on sampling Distribution (t-test and z test) using python

- ## Theory:

  A t-test is a statistical hypothesis test that is used to determine if there is a significant difference between the means of two groups. It is a parametric test that assumes that the data is normally distributed and the variances of the two groups are equal. The t-test compares the means of the two groups and calculates the probability of obtaining such a difference by chance alone.

  There are several types of t-tests, but the most commonly used ones are:

  One-sample t-test: Used to compare the mean of a sample to a known value or population mean.

  Independent samples t-test: Used to compare the means of two independent groups.

  Paired samples t-test: Used to compare the means of two related groups, such as before-and-after measurements.

- ## T-test:

- ## One-sample t-test:

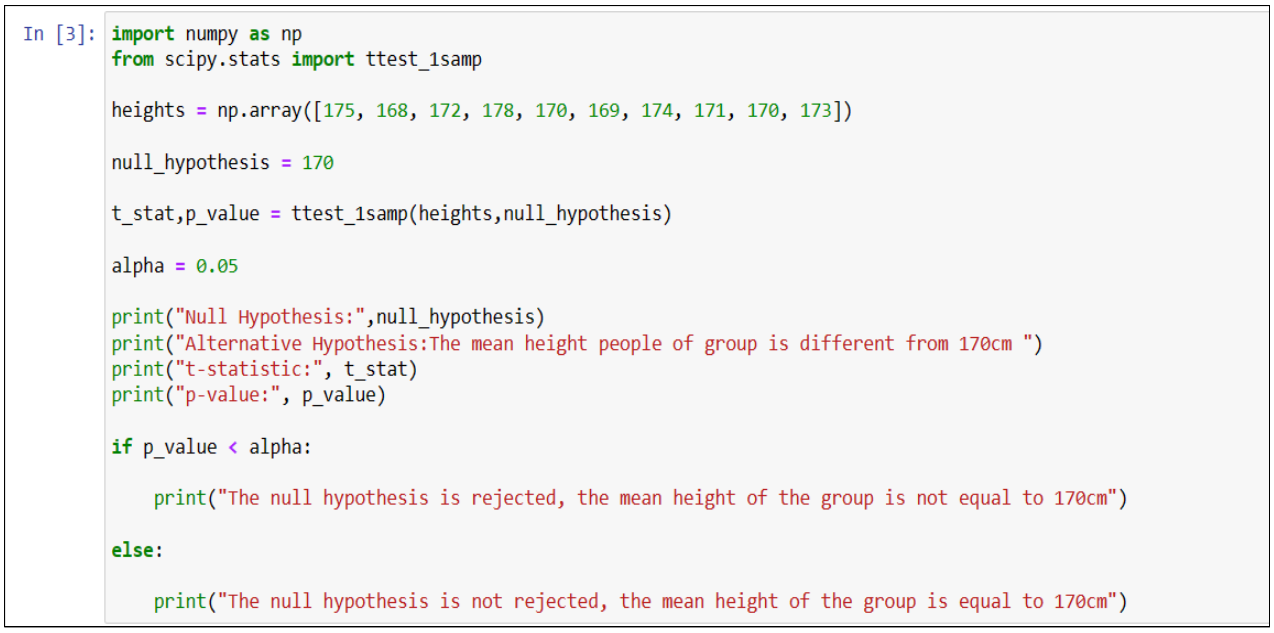
  One-sample t-test is used to determine whether a sample mean is significantly different from a known or hypothesized population mean.

  ### conditions for the one-sample t-test:

  - Random sampling: The sample should be randomly selected from the population of interest.
  - Normality: The data should be approximately normally distributed.
  - Independence: The observations in the sample should be independent.
  - Homogeneity of variance: The variance of the data within the sample should be approximately equal.
  - Interval or ratio scale: The data should be measured on an interval or ratio scale.
  - Sample size: The sample size should be sufficiently large (typically ≥ 30) for normality assumption.

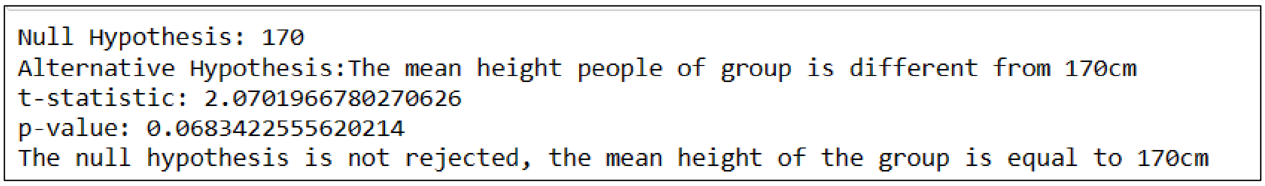
Example 1: Testing whether the mean height of a group of people is 170cm

Code:



```
In [3]: import numpy as np
        from scipy.stats import ttest_1samp

        heights = np.array([175, 168, 172, 178, 170, 169, 174, 171, 170, 173])

        null_hypothesis = 170

        t_stat,p_value = ttest_1samp(heights,null_hypothesis)

        alpha = 0.05

        print("Null Hypothesis:",null_hypothesis)
        print("Alternative Hypothesis:The mean height people of group is different from 170cm ")
        print("t-statistic:", t_stat)
        print("p-value:", p_value)

        if p_value < alpha:

            print("The null hypothesis is rejected, the mean height of the group is not equal to 170cm")

        else:

            print("The null hypothesis is not rejected, the mean height of the group is equal to 170cm")
```

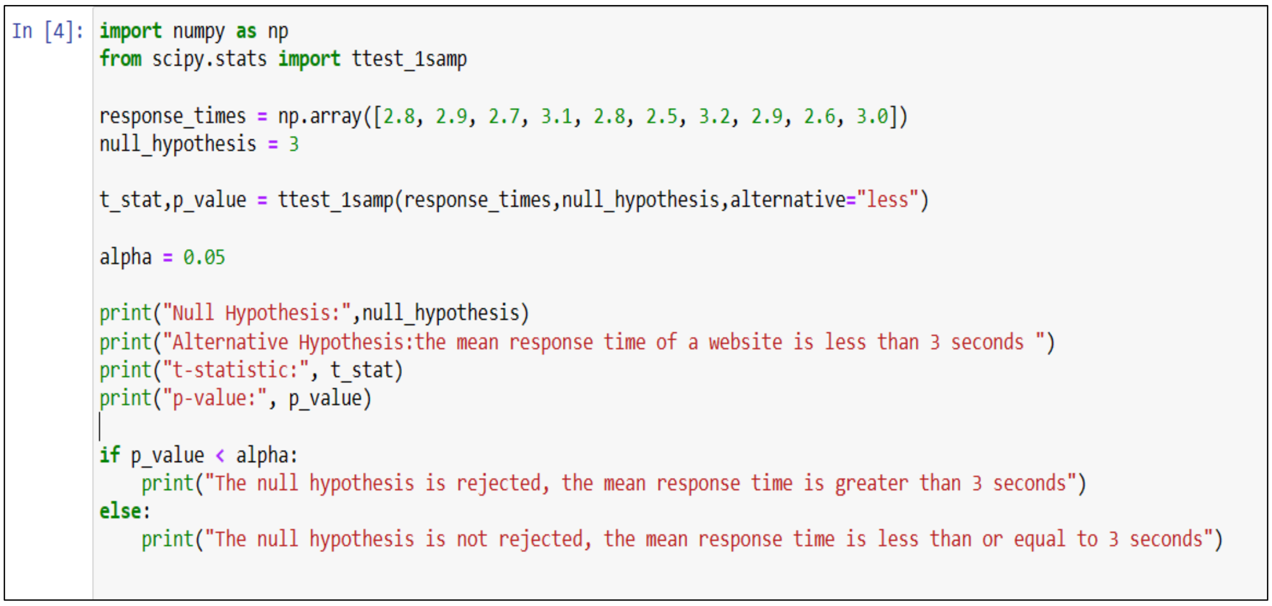
Output:



```
Null Hypothesis: 170
Alternative Hypothesis:The mean height people of group is different from 170cm
t-statistic: 2.0701966780270626
p-value: 0.0683422555620214
The null hypothesis is not rejected, the mean height of the group is equal to 170cm
```

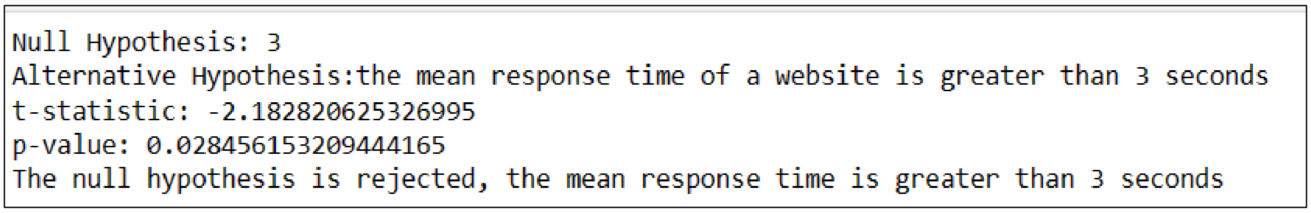
Example 2: Testing whether the mean response time of a website is less than 3 seconds.

Code:



```
In [4]: import numpy as np
        from scipy.stats import ttest_1samp

        response_times = np.array([2.8, 2.9, 2.7, 3.1, 2.8, 2.5, 3.2, 2.9, 2.6, 3.0])
        null_hypothesis = 3

        t_stat,p_value = ttest_1samp(response_times,null_hypothesis,alternative="less")

        alpha = 0.05

        print("Null Hypothesis:",null_hypothesis)
        print("Alternative Hypothesis:the mean response time of a website is less than 3 seconds ")
        print("t-statistic:", t_stat)
        print("p-value:", p_value)

        if p_value < alpha:
            print("The null hypothesis is rejected, the mean response time is greater than 3 seconds")
        else:
            print("The null hypothesis is not rejected, the mean response time is less than or equal to 3 seconds")
```

OUTPUT:



```
Null Hypothesis: 3
Alternative Hypothesis:the mean response time of a website is greater than 3 seconds
t-statistic: -2.182820625326995
p-value: 0.028456153209444165
The null hypothesis is rejected, the mean response time is greater than 3 seconds
```

❖ Independent samples t-test:

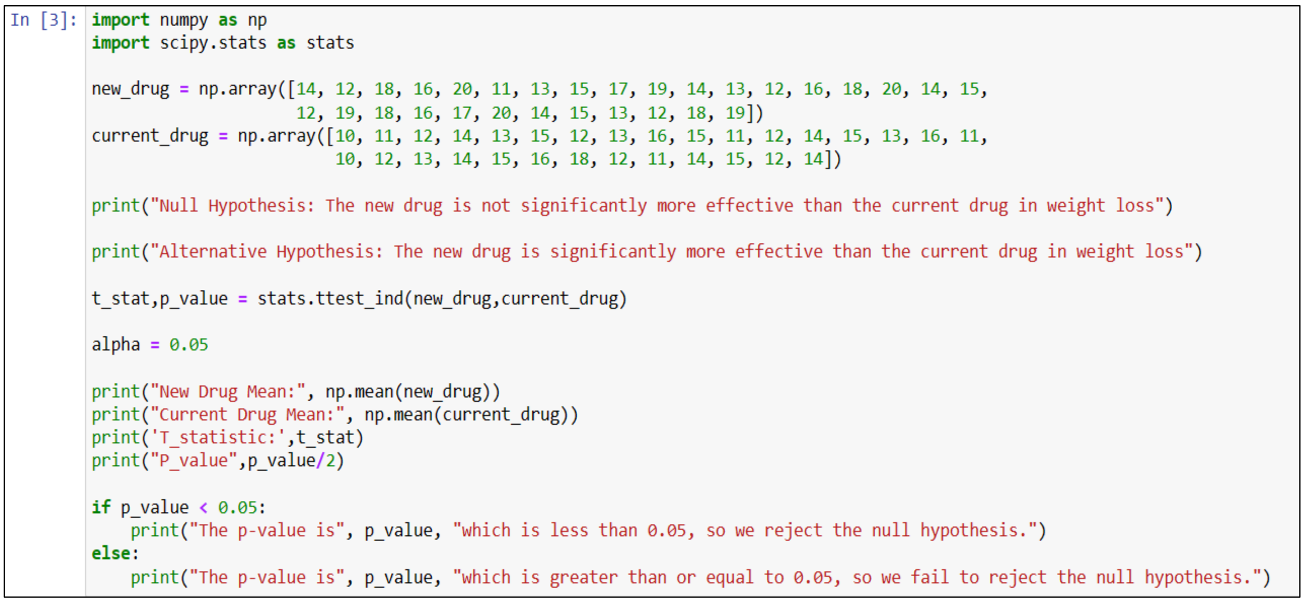
The independent samples t-test, also known as the unpaired or two-sample t-test, is a statistical test used to determine if there is a significant difference between the means of two separate and independent groups.

The conditions for conducting an independent samples t-test typically include:

- Random sampling: Both groups should be randomly sampled from their respective populations.
- Normality: The data for each group should be approximately normally distributed.
- Independence: Observations in each group should be independent.
- Homogeneity of variance: Variances of the data in both groups should be approximately equal.
- Interval or ratio scale: Data for both groups should be measured on an interval or ratio scale.
- Sample size: Sample sizes for both groups should be sufficiently large (typically ≥ 30) for normality assumption.

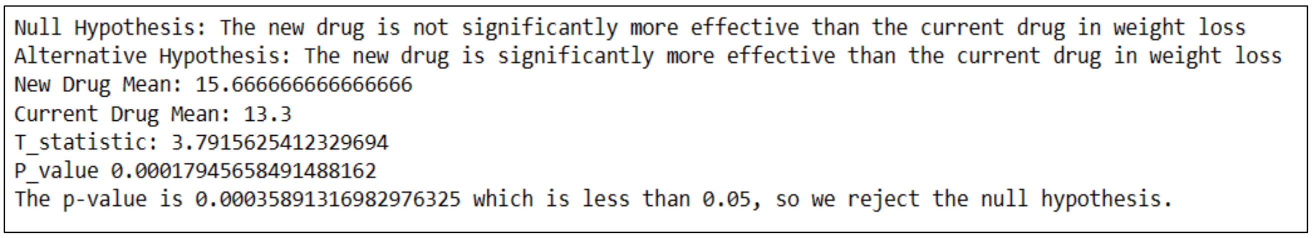
Example-1: Suppose a researcher wants to test whether a new weight loss drug is more effective than the current drug in the market. The researcher takes a random sample of 30 individuals who are given the new drug and another random sample of 30 individuals who are given the current drug. After a period of 6 months, the weight loss of both groups are recorded. The researcher wants to use an independent samples t-test to determine if the new drug is significantly more effective in weight loss compared to the current drug.

Code:



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

        new_drug = np.array([14, 12, 18, 16, 20, 11, 13, 15, 17, 19, 14, 13, 12, 16, 18, 20, 14, 15,
                             12, 19, 18, 16, 17, 20, 14, 15, 13, 12, 18, 19])
        current_drug = np.array([10, 11, 12, 14, 13, 15, 12, 13, 16, 15, 11, 12, 14, 15, 13, 16, 11,
                                 10, 12, 13, 14, 15, 16, 18, 12, 11, 14, 15, 12, 14])

        print("Null Hypothesis: The new drug is not significantly more effective than the current drug in weight loss")

        print("Alternative Hypothesis: The new drug is significantly more effective than the current drug in weight loss")

        t_stat,p_value = stats.ttest_ind(new_drug,current_drug)

        alpha = 0.05

        print("New Drug Mean:", np.mean(new_drug))
        print("Current Drug Mean:", np.mean(current_drug))
        print('T_statistic:',t_stat)
        print("P_value",p_value/2)

        if p_value < 0.05:
            print("The p-value is", p_value, "which is less than 0.05, so we reject the null hypothesis.")
        else:
            print("The p-value is", p_value, "which is greater than or equal to 0.05, so we fail to reject the null hypothesis.")
```

Output:

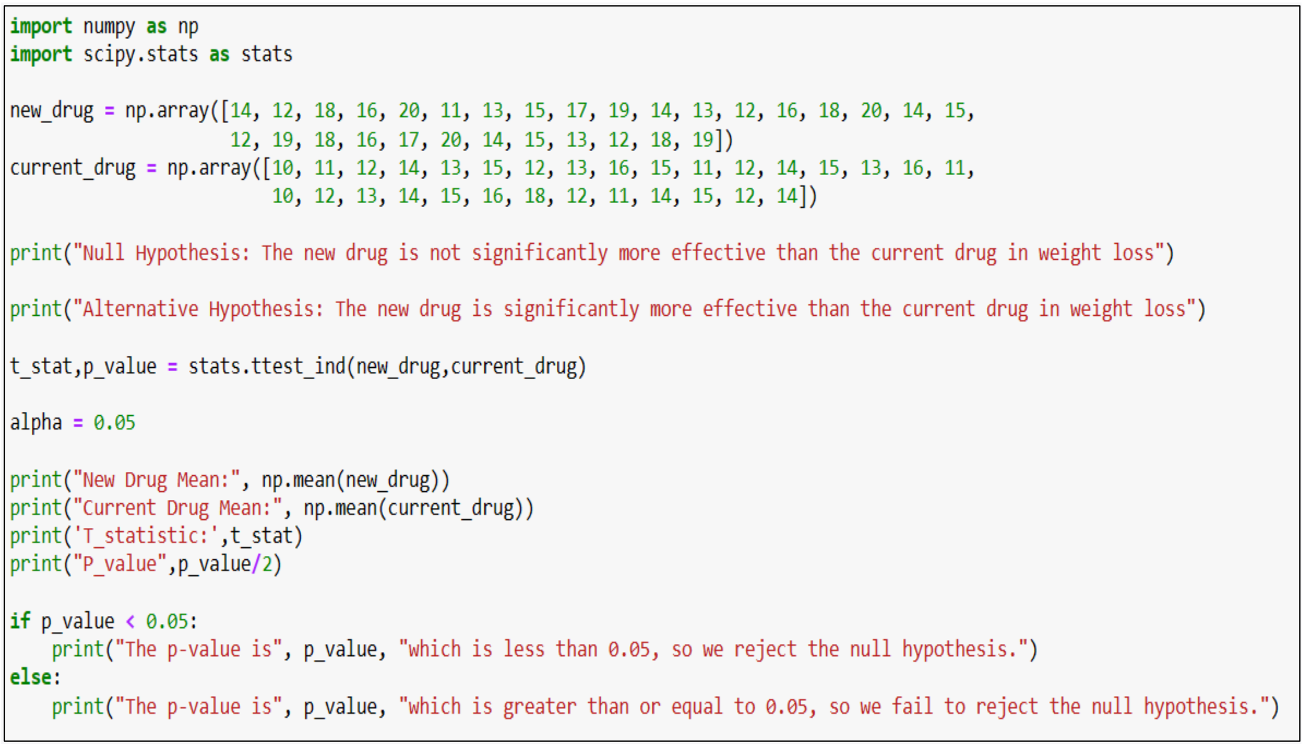


```
Null Hypothesis: The new drug is not significantly more effective than the current drug in weight loss
Alternative Hypothesis: The new drug is significantly more effective than the current drug in weight loss
New Drug Mean: 15.666666666666666
Current Drug Mean: 13.3
T_statistic: 3.7915625412329694
P_value 0.00017945658491488162
The p-value is 0.00035891316982976325 which is less than 0.05, so we reject the null hypothesis.
```

Example 2:

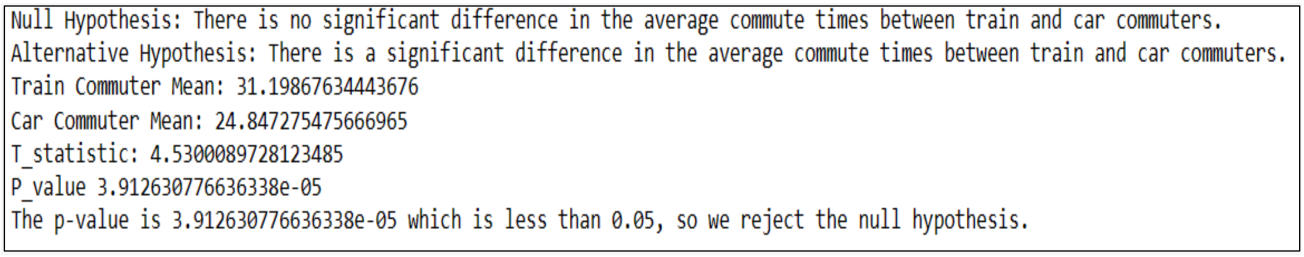
To conduct an independent samples t-test, we would need to collect data on commute times for both train and car commuters. The sample size for each group should be 25. We would then calculate the means and standard deviations of commute times for both groups, and perform the t-test to determine if there is a statistically significant difference in the means.

Code:



```python
import numpy as np
import scipy.stats as stats

new_drug = np.array([14, 12, 18, 16, 20, 11, 13, 15, 17, 19, 14, 13, 12, 16, 18, 20, 14, 15,
                     12, 19, 18, 16, 17, 20, 14, 15, 13, 12, 18, 19])
current_drug = np.array([10, 11, 12, 14, 13, 15, 12, 13, 16, 15, 11, 12, 14, 15, 13, 16, 11,
                         10, 12, 13, 14, 15, 16, 18, 12, 11, 14, 15, 12, 14])

print("Null Hypothesis: The new drug is not significantly more effective than the current drug in weight loss")

print("Alternative Hypothesis: The new drug is significantly more effective than the current drug in weight loss")

t_stat,p_value = stats.ttest_ind(new_drug,current_drug)

alpha = 0.05

print("New Drug Mean:", np.mean(new_drug))
print("Current Drug Mean:", np.mean(current_drug))
print('T_statistic:',t_stat)
print("P_value",p_value/2)

if p_value < 0.05:
    print("The p-value is", p_value, "which is less than 0.05, so we reject the null hypothesis.")
else:
    print("The p-value is", p_value, "which is greater than or equal to 0.05, so we fail to reject the null hypothesis.")
```

Output:



```
Null Hypothesis: There is no significant difference in the average commute times between train and car commuters.
Alternative Hypothesis: There is a significant difference in the average commute times between train and car commuters.
Train Commuter Mean: 31.19867634443676
Car Commuter Mean: 24.847275475666965
T_statistic: 4.5300089728123485
P_value 3.912630776636338e-05
The p-value is 3.912630776636338e-05 which is less than 0.05, so we reject the null hypothesis.
```

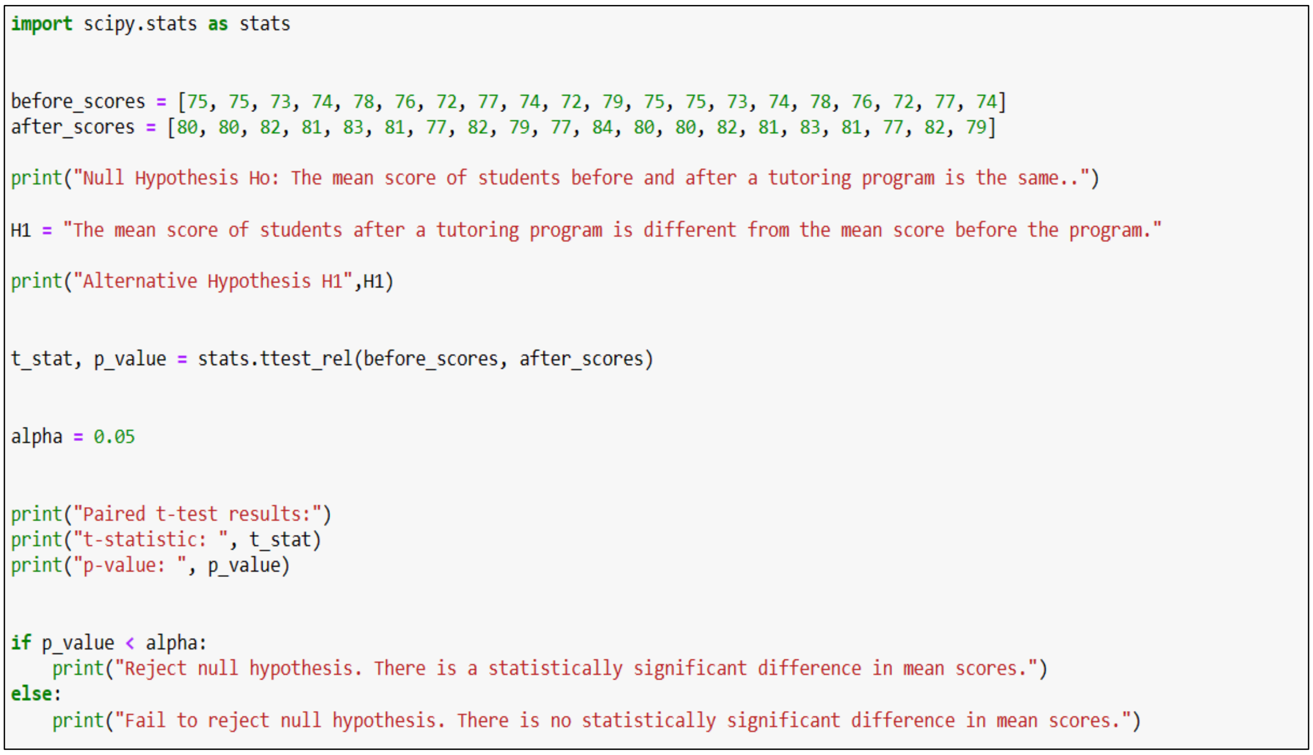
❖ Paired samples t-test:

The paired samples t-test is a statistical test used to compare means of related groups with continuous data, assuming normal distribution, equal variances, and paired observations. Used to compare the means of two related groups, such as before-and-after measurements.

Example 1:

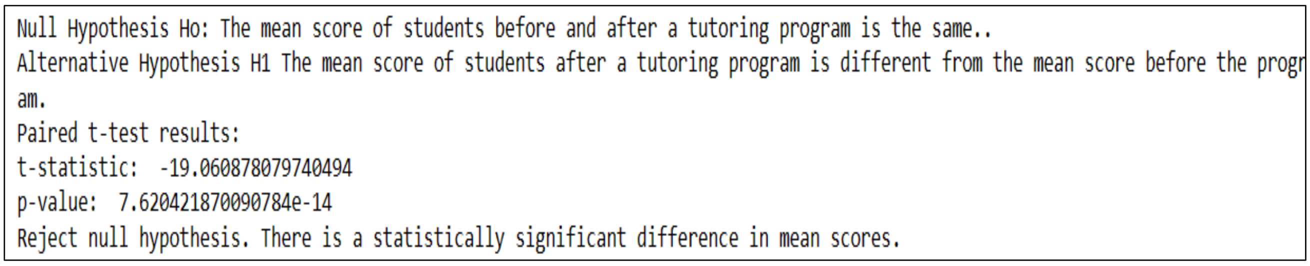
A paired t-test is conducted to compare the mean scores of the same group of students before and after the tutoring program. The data is collected for a sample of 20 students, and the mean score before the program is 75 with a standard deviation of 5, while the mean score after the program is 80 with a standard deviation of 6.

Code:



```python
import scipy.stats as stats


before_scores = [75, 75, 73, 74, 78, 76, 72, 77, 74, 72, 79, 75, 75, 73, 74, 78, 76, 72, 77, 74]
after_scores = [80, 80, 82, 81, 83, 81, 77, 82, 79, 77, 84, 80, 80, 82, 81, 83, 81, 77, 82, 79]

print("Null Hypothesis Ho: The mean score of students before and after a tutoring program is the same..")

H1 = "The mean score of students after a tutoring program is different from the mean score before the program."

print("Alternative Hypothesis H1",H1)


t_stat, p_value = stats.ttest_rel(before_scores, after_scores)


alpha = 0.05


print("Paired t-test results:")
print("t-statistic: ", t_stat)
print("p-value: ", p_value)


if p_value < alpha:
    print("Reject null hypothesis. There is a statistically significant difference in mean scores.")
else:
    print("Fail to reject null hypothesis. There is no statistically significant difference in mean scores.")
```

Output:

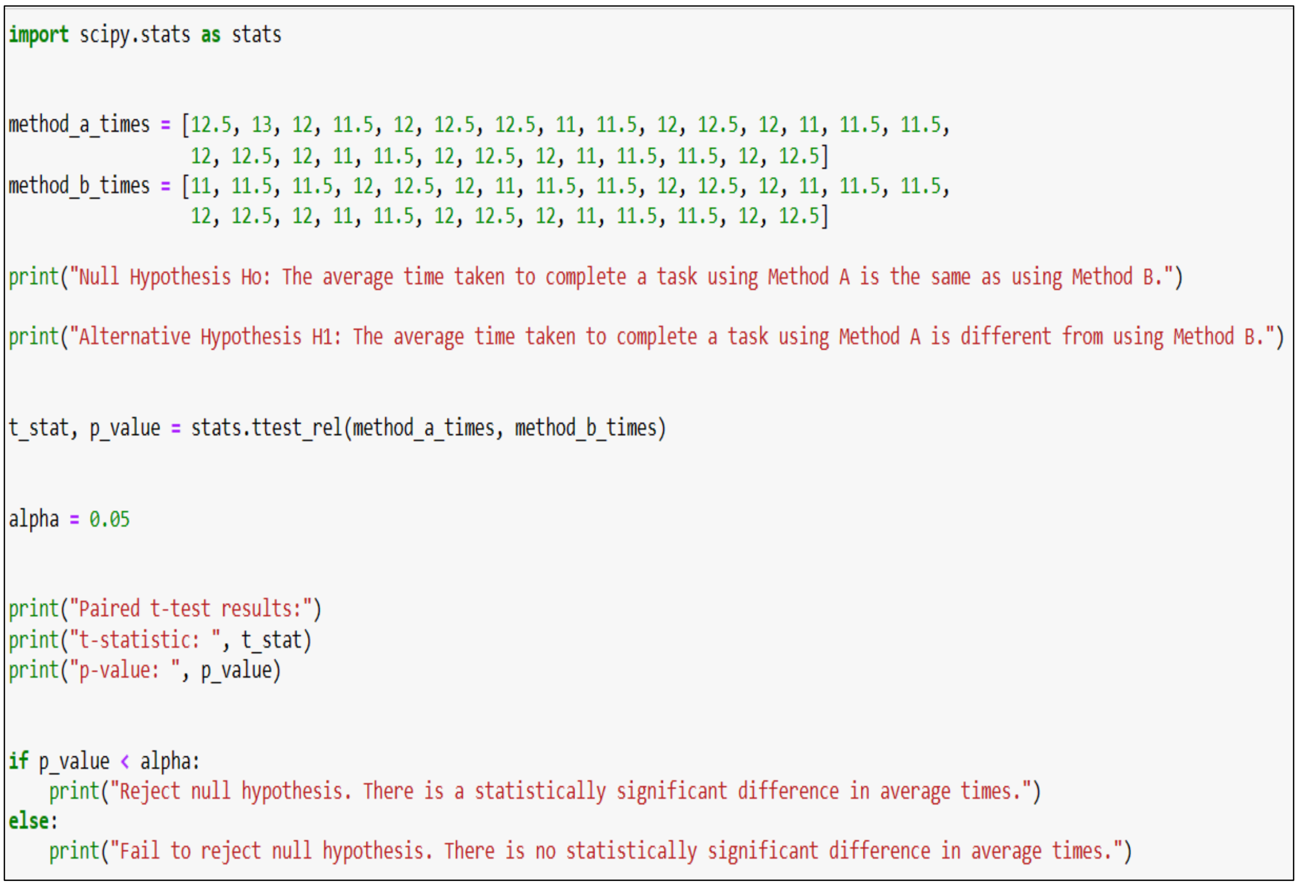


```
Null Hypothesis Ho: The mean score of students before and after a tutoring program is the same..
Alternative Hypothesis H1 The mean score of students after a tutoring program is different from the mean score before the progr
am.
Paired t-test results:
t-statistic:  -19.060878079740494
p-value:  7.620421870090784e-14
Reject null hypothesis. There is a statistically significant difference in mean scores.
```

Example 2:

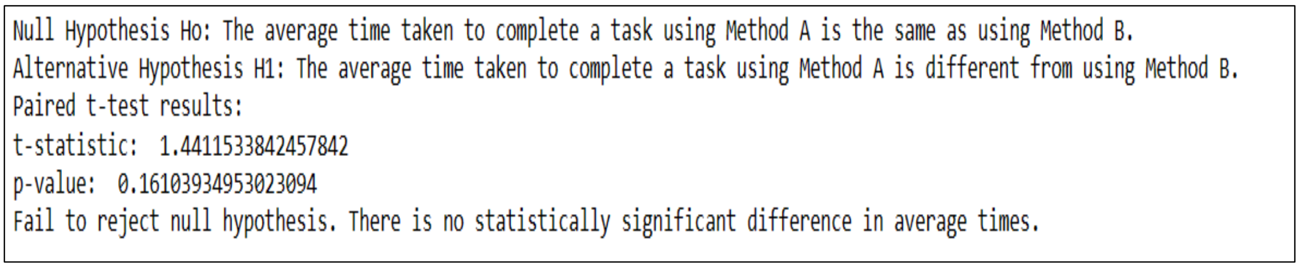
A paired t-test is conducted to compare the average time taken to complete a task using Method A and Method B. The data is collected for a sample of 30 participants, and the average time taken using Method A is 12.5 minutes with a standard deviation of 2.0 minutes, while the average time taken using Method B is 11.0 minutes with a standard deviation of 1.5 minutes.

Code:



```python
import scipy.stats as stats


method_a_times = [12.5, 13, 12, 11.5, 12, 12.5, 12.5, 11, 11.5, 12, 12.5, 12, 11, 11.5, 11.5,
                  12, 12.5, 12, 11, 11.5, 12, 12.5, 12, 11, 11.5, 11.5, 12, 12.5]
method_b_times = [11, 11.5, 11.5, 12, 12.5, 12, 11, 11.5, 11.5, 12, 12.5, 12, 11, 11.5, 11.5,
                  12, 12.5, 12, 11, 11.5, 12, 12.5, 12, 11, 11.5, 11.5, 12, 12.5]

print("Null Hypothesis Ho: The average time taken to complete a task using Method A is the same as using Method B.")

print("Alternative Hypothesis H1: The average time taken to complete a task using Method A is different from using Method B.")


t_stat, p_value = stats.ttest_rel(method_a_times, method_b_times)


alpha = 0.05


print("Paired t-test results:")
print("t-statistic: ", t_stat)
print("p-value: ", p_value)


if p_value < alpha:
    print("Reject null hypothesis. There is a statistically significant difference in average times.")
else:
    print("Fail to reject null hypothesis. There is no statistically significant difference in average times.")
```

Output:



```
Null Hypothesis Ho: The average time taken to complete a task using Method A is the same as using Method B.
Alternative Hypothesis H1: The average time taken to complete a task using Method A is different from using Method B.
Paired t-test results:
t-statistic:  1.4411533842457842
p-value:  0.16103934953023094
Fail to reject null hypothesis. There is no statistically significant difference in average times.
```

❖ Z-test:

A Z-test is a statistical test used to compare a sample mean to a known population mean, when the population standard deviation is known. It is used to determine whether there is a significant difference between the sample mean and the hypothesized population mean. The Z-test is based on the standard normal distribution and is used to make inferences about the population mean based on a sample.

The Z-test involves calculating a test statistic, which is a Z-score, using the formula:

$$Z = (X - \mu) / (\sigma / \text{sqrt}(n))$$

where:
X is the sample mean
μ is the hypothesized population mean
σ is the known population standard deviation
n is the sample size

➢ Conditions for a Z-test:

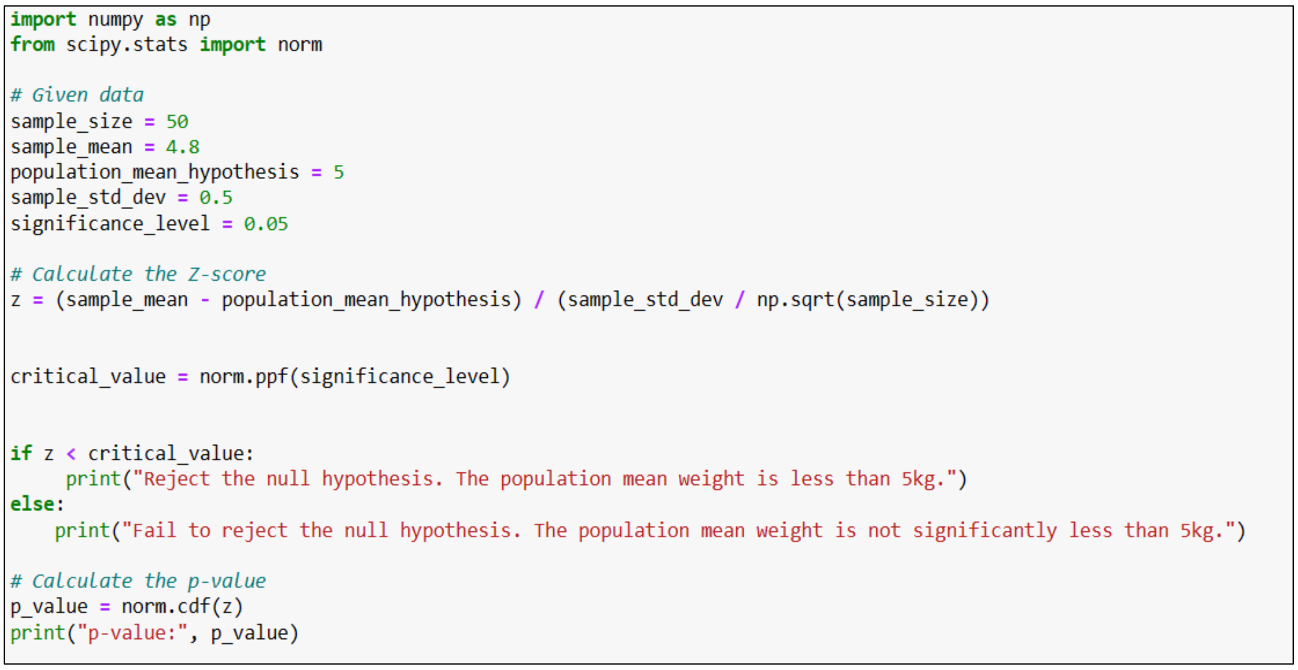
1. Normally distributed data.
2. Large sample size (typically $n \geq 30$).
3. Known population standard deviation.

❖ There are two main types of Z-test:

1. **One-sample Z-test:** This type of Z-test is used to compare a sample mean to a known population mean, when the population standard deviation is known. It is used to determine whether there is a significant difference between the sample mean and the hypothesized population mean.

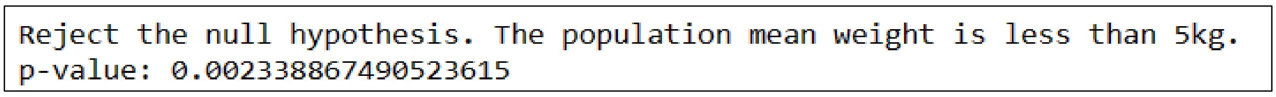
   Example suppose we want to test whether the mean weight of a population of bags of rice is less than a hypothesized value of 5kg. we randomly sample 50 bags of rice the population and weight them. The sample is found to be 4.8kg with a standard deviation of 0.5kg. we can use a one-sample z-test to determine whether this different is statistically significant at a 5% significance level.

Code:



```python
import numpy as np
from scipy.stats import norm

# Given data
sample_size = 50
sample_mean = 4.8
population_mean_hypothesis = 5
sample_std_dev = 0.5
significance_level = 0.05

# Calculate the Z-score
z = (sample_mean - population_mean_hypothesis) / (sample_std_dev / np.sqrt(sample_size))


critical_value = norm.ppf(significance_level)


if z < critical_value:
     print("Reject the null hypothesis. The population mean weight is less than 5kg.")
else:
    print("Fail to reject the null hypothesis. The population mean weight is not significantly less than 5kg.")

# Calculate the p-value
p_value = norm.cdf(z)
print("p-value:", p_value)
```

➢ **Output:**



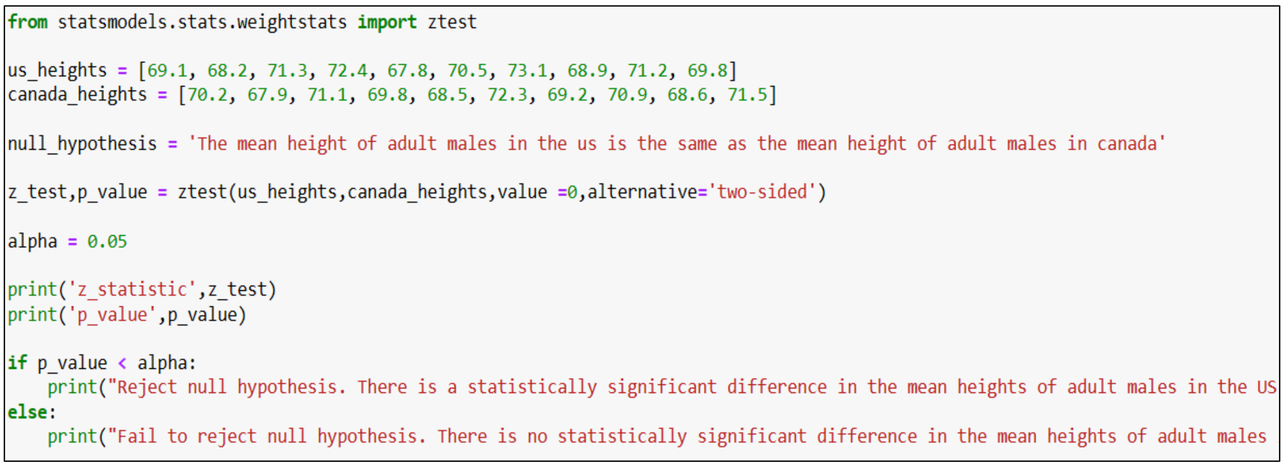
```
Reject the null hypothesis. The population mean weight is less than 5kg.
p-value: 0.002338867490523615
```

2. **Two-sample Z-test:** This type of Z-test is used to compare the means of two independent samples, when the population standard deviations are known. It is used to determine whether there is a significant difference between the means of the two samples.

❖ **Example**

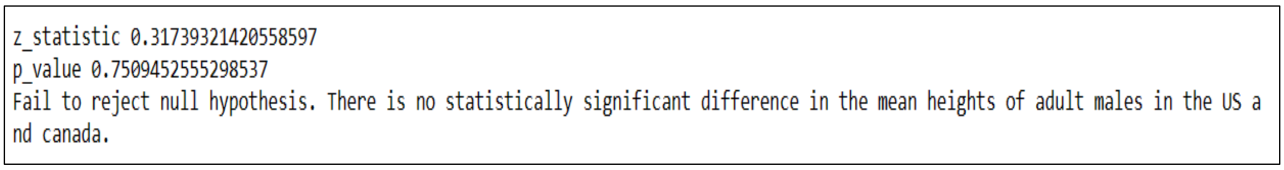
Suppose you want to test whether there is a significant difference between the mean height of adult males in the United States and the mean height of adult males in Canada. To do this, you could collect a sample of heights from each population and use a z-test to determine if the difference in sample means is statistically significant.

➢ **Code:**



```
from statsmodels.stats.weightstats import ztest

us_heights = [69.1, 68.2, 71.3, 72.4, 67.8, 70.5, 73.1, 68.9, 71.2, 69.8]
canada_heights = [70.2, 67.9, 71.1, 69.8, 68.5, 72.3, 69.2, 70.9, 68.6, 71.5]

null_hypothesis = 'The mean height of adult males in the us is the same as the mean height of adult males in canada'

z_test,p_value = ztest(us_heights,canada_heights,value =0,alternative='two-sided')

alpha = 0.05

print('z_statistic',z_test)
print('p_value',p_value)

if p_value < alpha:
    print("Reject null hypothesis. There is a statistically significant difference in the mean heights of adult males in the US
else:
    print("Fail to reject null hypothesis. There is no statistically significant difference in the mean heights of adult males
```

➢ **Output:**



```
z_statistic 0.31739321420558597
p_value 0.7509452555298537
Fail to reject null hypothesis. There is no statistically significant difference in the mean heights of adult males in the US a
nd canada.
```

Practical-4

❖ **Aim:**

Real life Problem based on Sampling Distribution (F, Chi-Square) using python.

❖ **Theory:**

F-test is a statistical hypothesis test used to compare the variances of two or more populations based on their sample variances. The F-test uses the F-distribution to calculate a test statistic that measures the ratio of the variances of the populations.

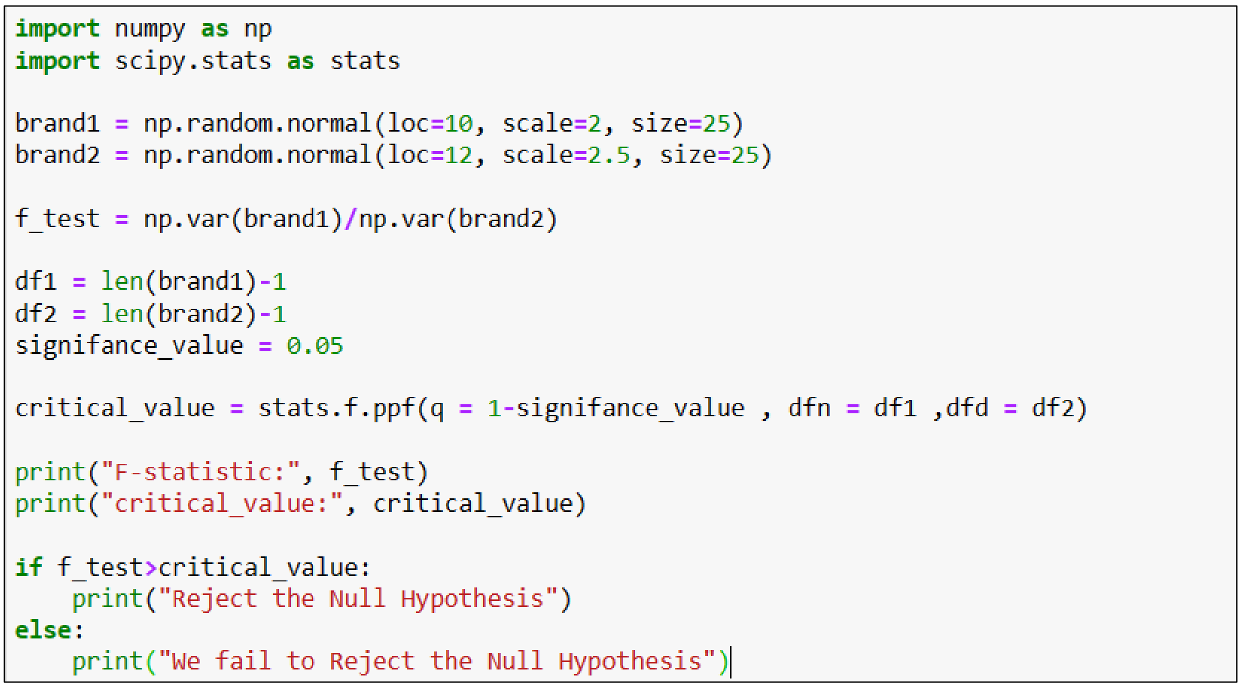
The F-test is used to test the null hypothesis that the variances of the populations are equal, against the alternative hypothesis that the variances are not equal. The F-test can be one-tailed or two-tailed, depending on the direction of the alternative hypothesis.

The conditions for using an F-test:

- The samples are independent of each other.
- The populations from which the samples are drawn are normally distributed.
- The variances of the populations from which the samples are drawn are unknown and assumed to be equal or unequal.
- The samples are randomly selected from the populations.
- The samples are approximately normally distributed.
- The sample size is small enough so that the distribution of the test statistic is approximately F-distributed.

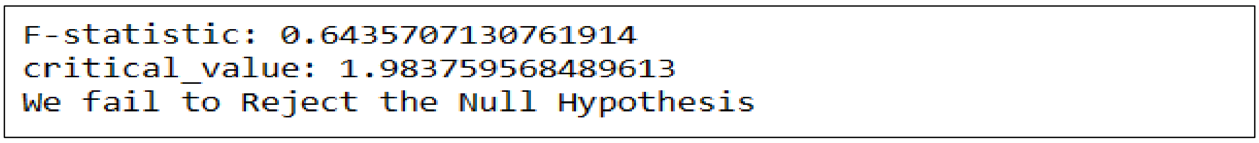
❖ **Example:** A company is testing two different brands of batteries to see if there is a significant difference in their lifetimes. They randomly select 25 batteries of each brand and record their lifetimes in hours. The company wants to use an F-test to compare the variances of the two samples.

➢ **Code:**



```python
import numpy as np
import scipy.stats as stats

brand1 = np.random.normal(loc=10, scale=2, size=25)
brand2 = np.random.normal(loc=12, scale=2.5, size=25)

f_test = np.var(brand1)/np.var(brand2)

df1 = len(brand1)-1
df2 = len(brand2)-1
signifance_value = 0.05

critical_value = stats.f.ppf(q = 1-signifance_value , dfn = df1 ,dfd = df2)

print("F-statistic:", f_test)
print("critical_value:", critical_value)

if f_test>critical_value:
    print("Reject the Null Hypothesis")
else:
    print("We fail to Reject the Null Hypothesis")
```

➢ **Output:**

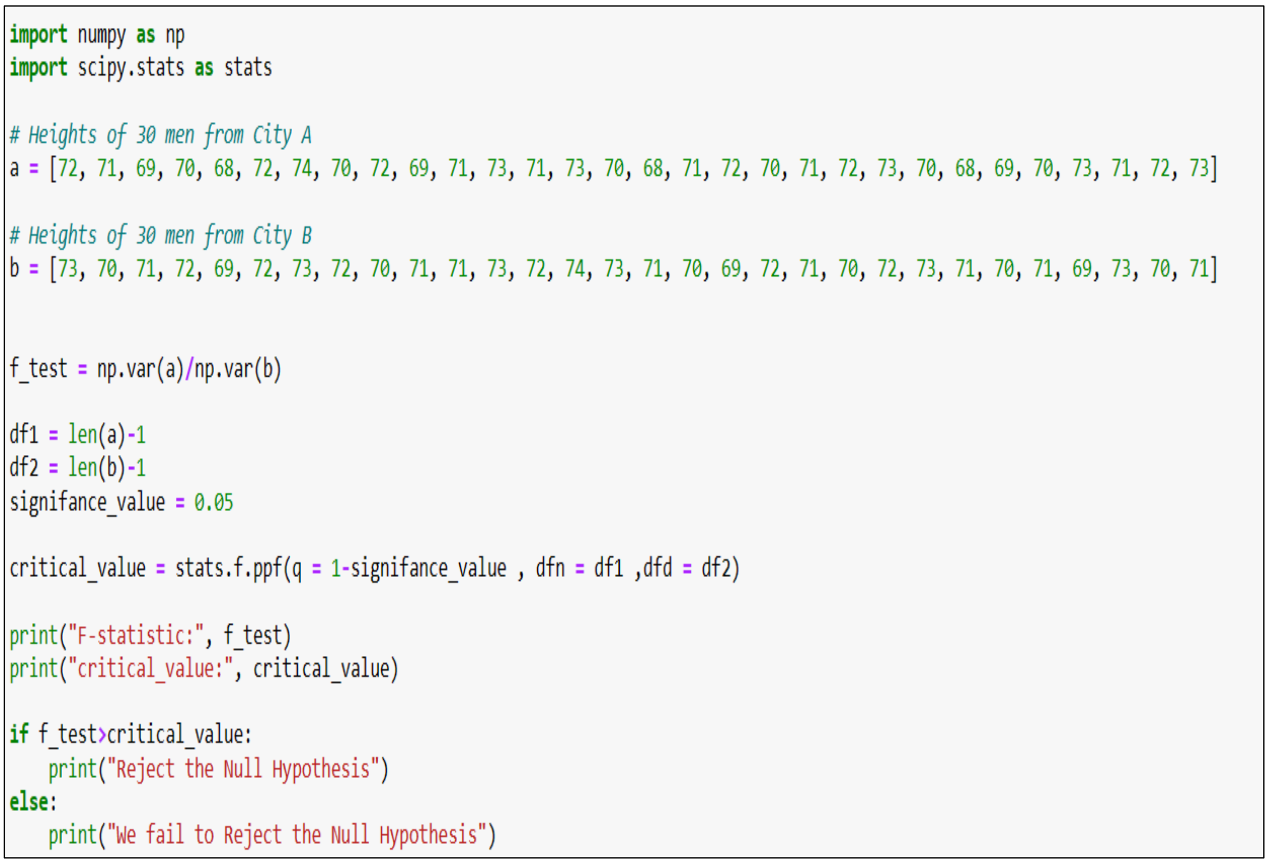


```
F-statistic: 0.6435707130761914
critical_value: 1.983759568489613
We fail to Reject the Null Hypothesis
```

❖ **Example:**

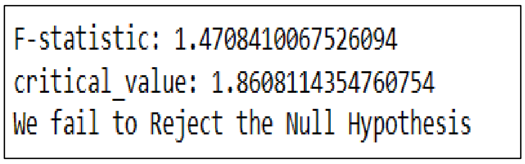
Suppose we have two sets of data, and we want to test whether their variances are equal. Here's an example where we have the heights (in inches) of 30 men from two different cities:

➢ **Code:**



```python
import numpy as np
import scipy.stats as stats

# Heights of 30 men from City A
a = [72, 71, 69, 70, 68, 72, 74, 70, 72, 69, 71, 73, 71, 73, 70, 68, 71, 72, 70, 71, 72, 73, 70, 68, 69, 70, 73, 71, 72, 73]

# Heights of 30 men from City B
b = [73, 70, 71, 72, 69, 72, 73, 72, 70, 71, 71, 73, 72, 74, 73, 71, 70, 69, 72, 71, 70, 72, 73, 71, 70, 71, 69, 73, 70, 71]


f_test = np.var(a)/np.var(b)

df1 = len(a)-1
df2 = len(b)-1
signifance_value = 0.05

critical_value = stats.f.ppf(q = 1-signifance_value , dfn = df1 ,dfd = df2)

print("F-statistic:", f_test)
print("critical_value:", critical_value)

if f_test>critical_value:
    print("Reject the Null Hypothesis")
else:
    print("We fail to Reject the Null Hypothesis")
```

➢ **Output:**



```
F-statistic: 1.4708410067526094
critical_value: 1.8608114354760754
We fail to Reject the Null Hypothesis
```

❖ **Chi-Square:**

Chi-square (χ^2) is a statistical distribution used to analyze categorical data. It is a continuous probability distribution that describes the distribution of the sum of squared standard normal deviates.

In statistics, the chi-square distribution is used to test the goodness of fit of an observed distribution to an expected distribution. It is also used in the chi-square test for independence to determine if there is a significant association between two categorical variables.

The chi-square distribution has a shape that is determined by its degrees of freedom (df), which represents the number of independent variables in the analysis. As the degrees of freedom increase, the chi-square distribution approaches a normal distribution.

The formula for calculating the chi-square test statistic is:

$$\chi^2 = \Sigma [(O - E)^2 / E]$$

where:

χ^2 is the chi-square test statistic

Σ is the sum of the values of $(O - E)^2 / E$ for all categories

O is the observed frequency for a category

E is the expected frequency for a category

The chi-square distribution is commonly used in a variety of statistical analyses, including goodness-of-fit tests, contingency table analyses, and logistic regression.

❖ **The conditions for using a chi-square test:**

- The data is categorical (nominal or ordinal).
- The sample is randomly selected from the population of interest.
- The sample size is large enough for the expected frequencies to be greater than 5.
- The observations in each category are independent of each other.
- The expected frequencies are not too small (i.e., less than 1).

❖ **Example:**

Goodness of fit:

Suppose we are analyzing the preferences of customers for different flavors of ice cream. We have collected data on the number of customers who prefer each flavor out of a total of 100 customers. We want to test if the distribution of ice cream flavor preferences in our sample matches a known distribution of flavors. We can use the chi-square goodness-of-fit test for this.

❖ **Code:**

```
import numpy as np
from scipy.stats import stats
from scipy.stats import chi2

observed = [25, 30, 20, 15, 10]
expected = [20, 30, 25, 15, 10]

# Calculate the chi-square test statistic and p-value
statistic, p_value = stats.chisquare(observed, expected)

# Print the results
print("Chi-square test statistic:", statistic)
print("p-value:", p_value)

## find the critical value

significance_value = 0.05

degree_of_freedom = len(expected)-1


critical_value = chi2.ppf(1 - significance_value, degree_of_freedom)

if statistic > critical_value:
    print("Reject the null hypothesis.")
else:
    print("Fail to reject the null hypothesis.")
```

❖ **Output:**

```
Chi-square test statistic: 2.25
p-value: 0.6898864931364932
Fail to reject the null hypothesis.
```

❖ **Example 2:** The distribution of the number of hours that students study on each day of the week is the expected distribution.

➢ **Code:**

```
import numpy as np
from scipy.stats import stats

## No of hours student study in a weekly basis daily
##monday,tuesday,Wednesday,thursday,Friday,Saturday,sunday

expected_data=[8,6,7,9,6,9,7]

observed_data=[7,8,6,9,9,6,7]

# Calculate the chi-square test statistic and p-value
statistic, p_value = stats.chisquare(observed_data, expected_data)

# Print the results
print("Chi-square test statistic:", statistic)
print("p-value:", p_value)

## find the critical value

significance_value = 0.05

degree_of_freedom = len(expected)-1


critical_value = chi2.ppf(1 - significance_value, degree_of_freedom)

if statistic > critical_value:
    print("Reject the null hypothesis.")

else:
    print("Fail to reject the null hypothesis.")
```

- **Output:**

```
Chi-square test statistic: 3.4345238095238093
p-value: 0.7526596580922865
Fail to reject the null hypothesis.
```

Practical-5

❖ **Aim:**

Real life Problem based on ANOVA (One Way) using python.

❖ **Theory:**

ANOVA stands for Analysis of Variance, which is a statistical method used to determine if there are significant differences between two or more groups of data.

1) **One-Way ANOVA:** This type of ANOVA is used to compare the means of two or more groups of data that are independent of each other. The conditions for using One-Way ANOVA are:

- The data must be continuous.
- The groups must be independent of each other.
- The variances of the groups must be equal.
- The data must be normally distributed.

❖ **Example**

Suppose we want to compare the mean salary of employees in three different departments in a company. We have collected random salary data for each department, and we want to determine if there is a significant difference in salaries between the departments.

➢ **Code:**

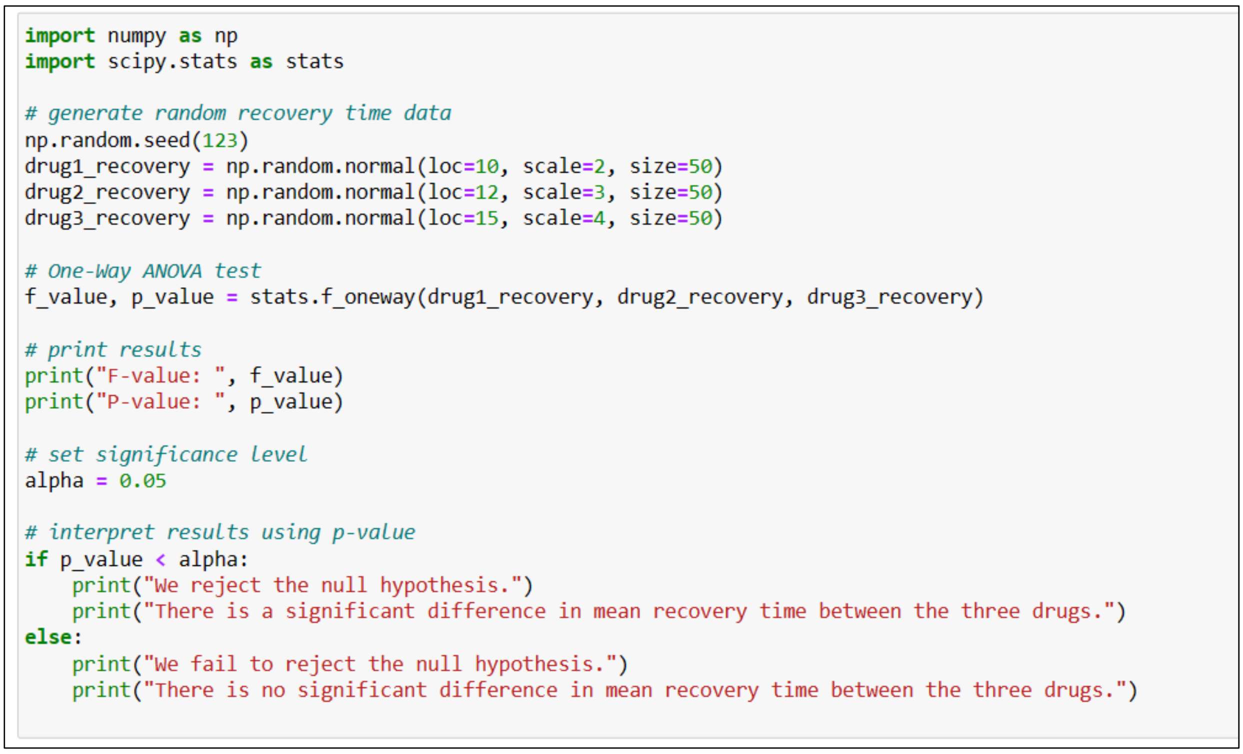
```python
import numpy as np
import scipy.stats as stats


np.random.seed(123)
sales_salary = np.random.normal(loc=50000, scale=10000, size=50)
marketing_salary = np.random.normal(loc=55000, scale=8000, size=50)
it_salary = np.random.normal(loc=60000, scale=12000, size=50)

# One-Way ANOVA test
f_value, p_value = stats.f_oneway(sales_salary, marketing_salary, it_salary)


print("F-value: ", f_value)
print("P-value: ", p_value)


alpha = 0.05


if p_value < alpha:
    print("We reject the null hypothesis.")
    print("There is a significant difference in mean salary between the three departments.")
else:
    print("We fail to reject the null hypothesis.")
    print("There is no significant difference in mean salary between the three departments.")
```

➢ **Output:**

```
F-value:  13.055870477967321
P-value:  6.037047433812819e-06
We reject the null hypothesis.
There is a significant difference in mean salary between the three departments.
```

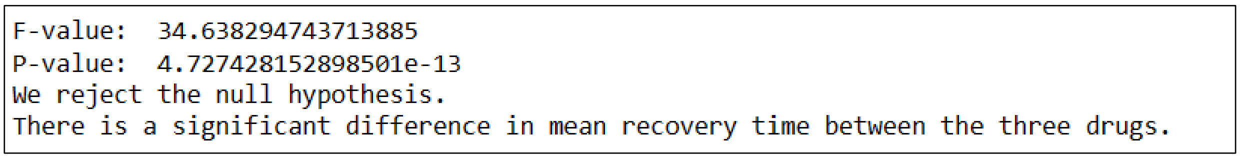
- **Example:**
  Suppose we want to compare the effectiveness of three different drugs in treating a certain disease. We have random collected data on the recovery time for patients taking each drug, and we want to determine if there is a significant difference in recovery times between the drugs.

- **Code:**



```python
import numpy as np
import scipy.stats as stats

# generate random recovery time data
np.random.seed(123)
drug1_recovery = np.random.normal(loc=10, scale=2, size=50)
drug2_recovery = np.random.normal(loc=12, scale=3, size=50)
drug3_recovery = np.random.normal(loc=15, scale=4, size=50)

# One-Way ANOVA test
f_value, p_value = stats.f_oneway(drug1_recovery, drug2_recovery, drug3_recovery)

# print results
print("F-value: ", f_value)
print("P-value: ", p_value)

# set significance level
alpha = 0.05

# interpret results using p-value
if p_value < alpha:
    print("We reject the null hypothesis.")
    print("There is a significant difference in mean recovery time between the three drugs.")
else:
    print("We fail to reject the null hypothesis.")
    print("There is no significant difference in mean recovery time between the three drugs.")
```

- **Output:**



```
F-value:  34.638294743713885
P-value:  4.727428152898501e-13
We reject the null hypothesis.
There is a significant difference in mean recovery time between the three drugs.
```

Practical-6

❖ **Aim:**

Real life Problem based on ANOVA (Two Way) using python.

❖ **Theory:**

ANOVA stands for Analysis of Variance, which is a statistical method used to determine if there are significant differences between two or more groups of data.

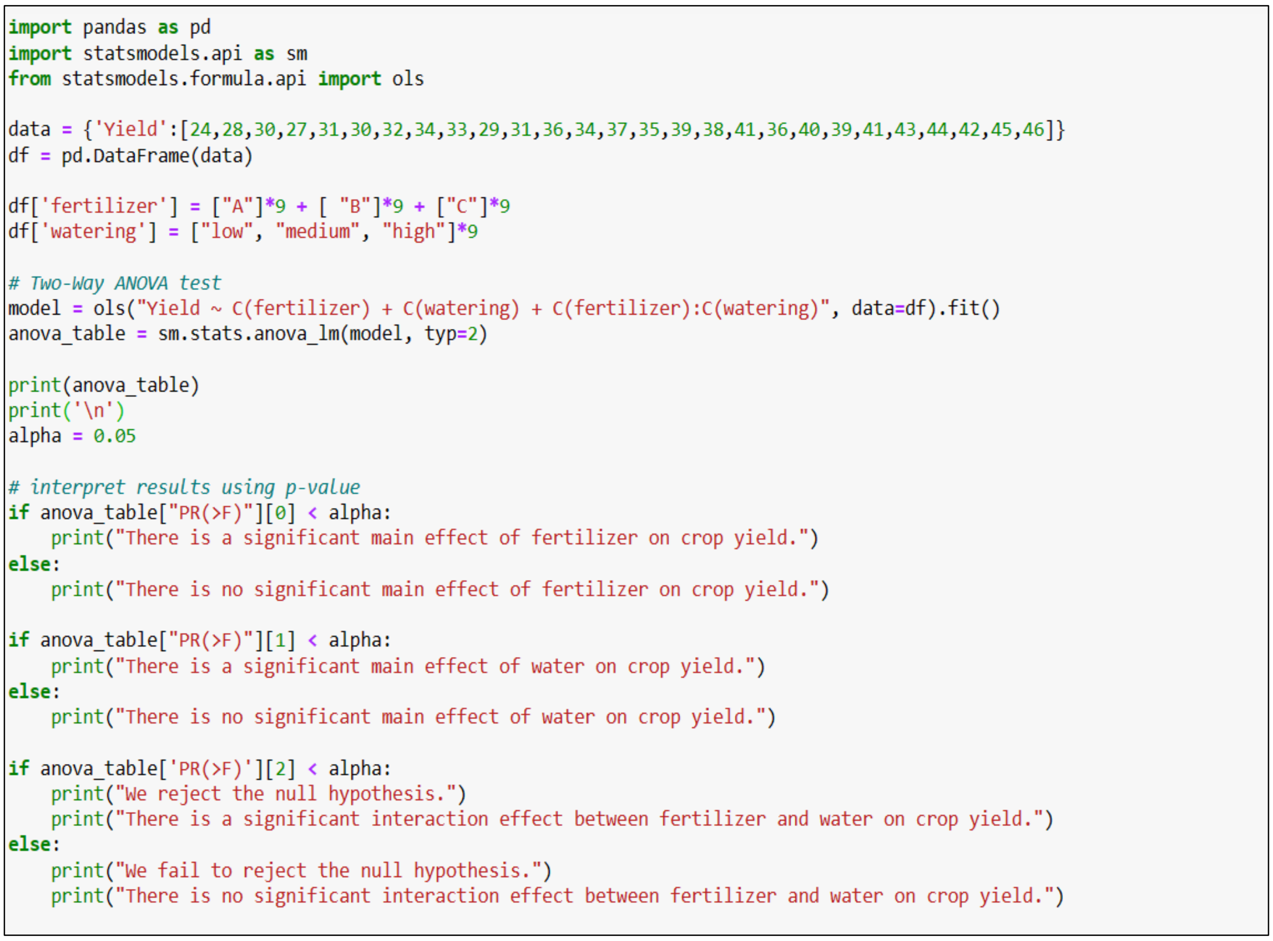
Two-Way ANOVA: This type of ANOVA is used to analyze the effects of two factors (independent variables) on a single dependent variable. The conditions for using Two-Way ANOVA are:

- The data must be continuous.
- The groups must be independent of each other.
- The variances of the groups must be equal.
- The data must be normally distributed.
- The factors must be independent of each other.

❖ **Example:**

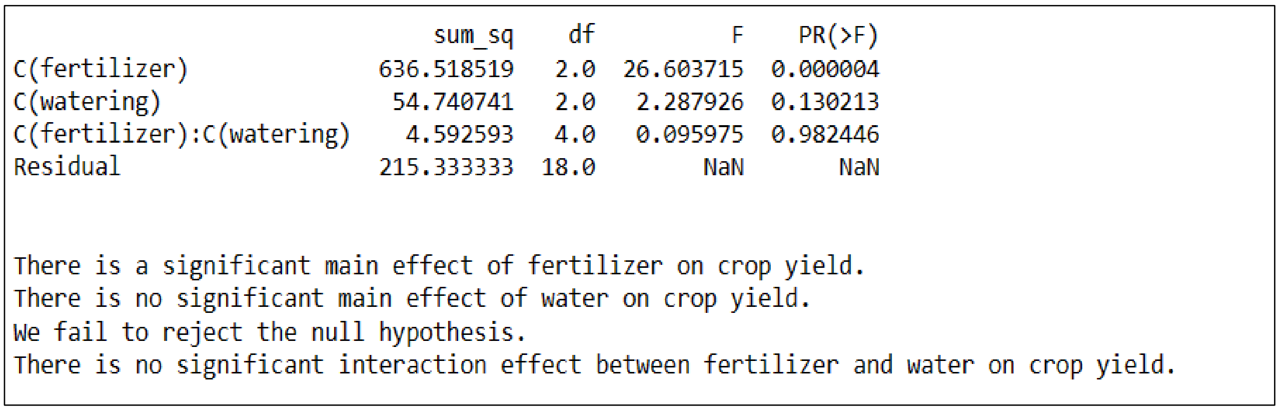
What is the effect of different levels of fertilizer and watering on plant growth? A study was conducted to compare the effectiveness of two different levels of fertilizer (low and high) and two different levels of watering (low and high) on plant growth. The null hypothesis is that there is no significant difference in mean plant growth between different levels of fertilizer and watering. The alternative hypothesis is that there is a significant difference in mean plant growth between different levels of fertilizer and watering.

➢ **Code:**



```python
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

data = {'Yield':[24,28,30,27,31,30,32,34,33,29,31,36,34,37,35,39,38,41,36,40,39,41,43,44,42,45,46]}
df = pd.DataFrame(data)

df['fertilizer'] = ["A"]*9 + [ "B"]*9 + ["C"]*9
df['watering'] = ["low", "medium", "high"]*9

# Two-Way ANOVA test
model = ols("Yield ~ C(fertilizer) + C(watering) + C(fertilizer):C(watering)", data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

print(anova_table)
print('\n')
alpha = 0.05

# interpret results using p-value
if anova_table["PR(>F)"][0] < alpha:
    print("There is a significant main effect of fertilizer on crop yield.")
else:
    print("There is no significant main effect of fertilizer on crop yield.")

if anova_table["PR(>F)"][1] < alpha:
    print("There is a significant main effect of water on crop yield.")
else:
    print("There is no significant main effect of water on crop yield.")

if anova_table['PR(>F)'][2] < alpha:
    print("We reject the null hypothesis.")
    print("There is a significant interaction effect between fertilizer and water on crop yield.")
else:
    print("We fail to reject the null hypothesis.")
    print("There is no significant interaction effect between fertilizer and water on crop yield.")
```

➢ **Output:**

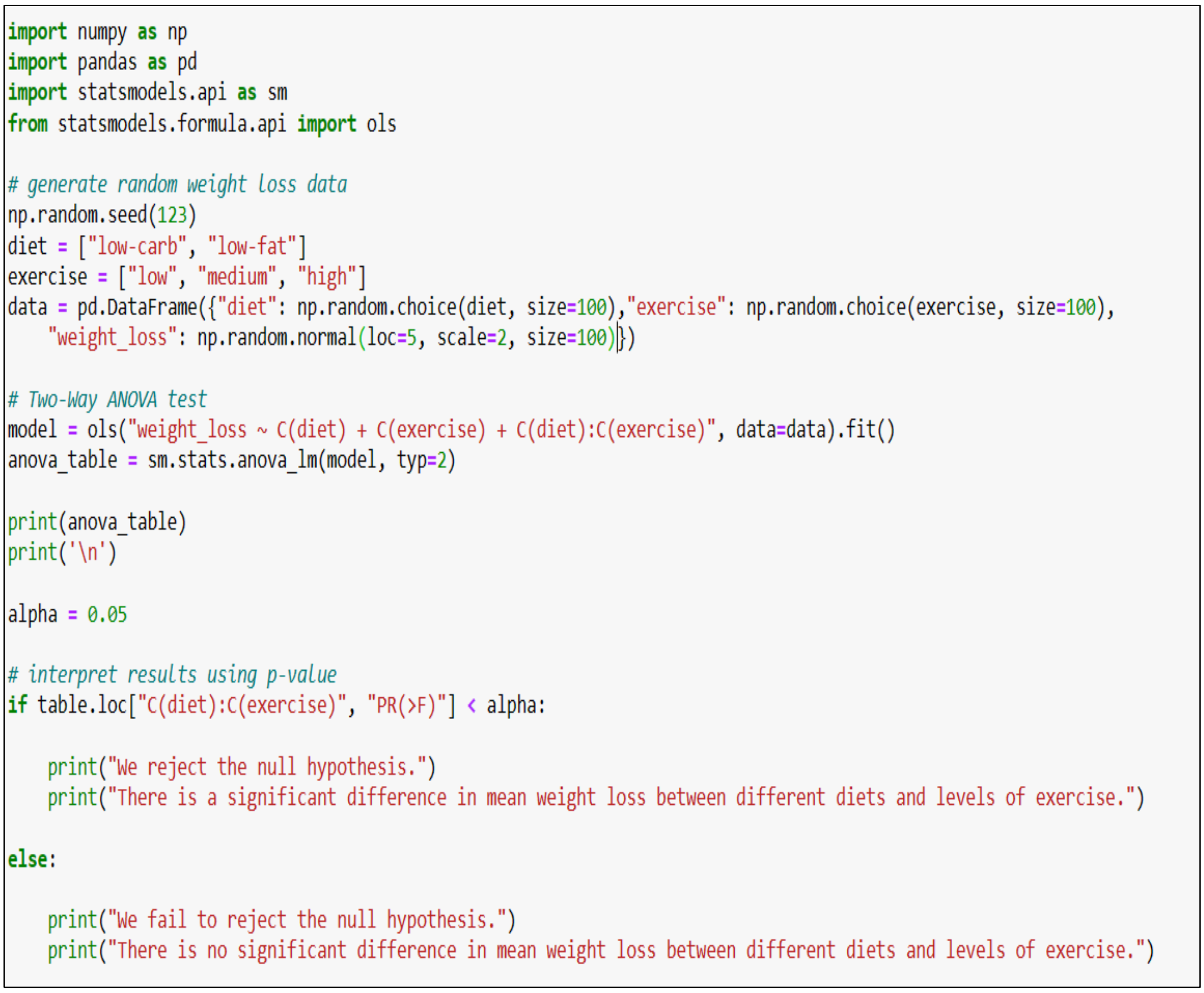


```
                            sum_sq    df          F    PR(>F)
C(fertilizer)           636.518519   2.0  26.603715  0.000004
C(watering)              54.740741   2.0   2.287926  0.130213
C(fertilizer):C(watering)  4.592593   4.0   0.095975  0.982446
Residual                215.333333  18.0        NaN       NaN


There is a significant main effect of fertilizer on crop yield.
There is no significant main effect of water on crop yield.
We fail to reject the null hypothesis.
There is no significant interaction effect between fertilizer and water on crop yield.
```

❖ **Example:**

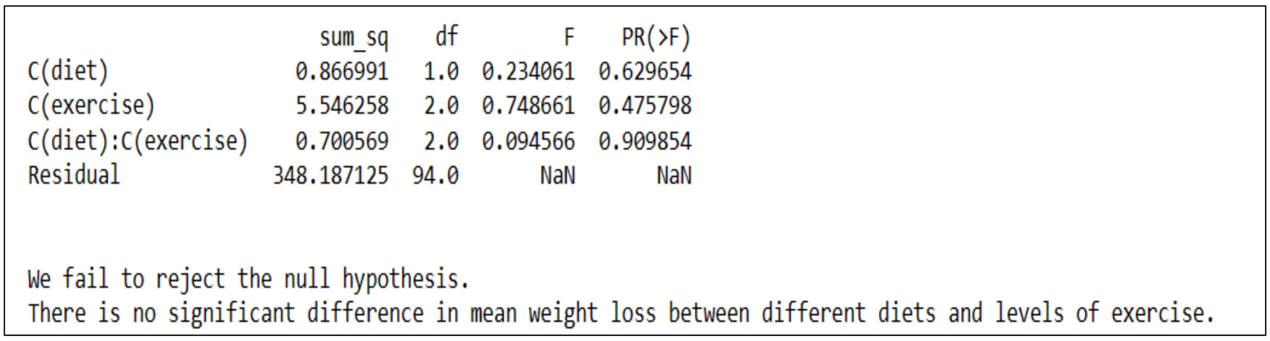
What is the effect of different diets and levels of exercise on weight loss? A study was conducted to compare the effectiveness of two different diets (low-carb and low-fat) and three different levels of exercise (low, medium, and high) on weight loss. Random data was collected on the weight loss for 100 participants, with each participant assigned to a specific diet and level of exercise. Perform a Two-Way ANOVA test to determine if there is a significant difference in mean weight loss between different diets and levels of exercise. State the null and alternative hypothesis and interpret the results of the test using a significance level of 0.05.

➢ **Code:**



```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# generate random weight loss data
np.random.seed(123)
diet = ["low-carb", "low-fat"]
exercise = ["low", "medium", "high"]
data = pd.DataFrame({"diet": np.random.choice(diet, size=100),"exercise": np.random.choice(exercise, size=100),
    "weight_loss": np.random.normal(loc=5, scale=2, size=100)})

# Two-Way ANOVA test
model = ols("weight_loss ~ C(diet) + C(exercise) + C(diet):C(exercise)", data=data).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

print(anova_table)
print('\n')

alpha = 0.05

# interpret results using p-value
if table.loc["C(diet):C(exercise)", "PR(>F)"] < alpha:

    print("We reject the null hypothesis.")
    print("There is a significant difference in mean weight loss between different diets and levels of exercise.")

else:

    print("We fail to reject the null hypothesis.")
    print("There is no significant difference in mean weight loss between different diets and levels of exercise.")
```

➢ **Output:**



```
                          sum_sq    df         F    PR(>F)
C(diet)                 0.866991   1.0  0.234061  0.629654
C(exercise)             5.546258   2.0  0.748661  0.475798
C(diet):C(exercise)     0.700569   2.0  0.094566  0.909854
Residual              348.187125  94.0       NaN       NaN


We fail to reject the null hypothesis.
There is no significant difference in mean weight loss between different diets and levels of exercise.
```

Practical-7

❖ Aim:

Real life Problem Based on Principle Component Analysis (PCA) in Python.

❖ Theory:

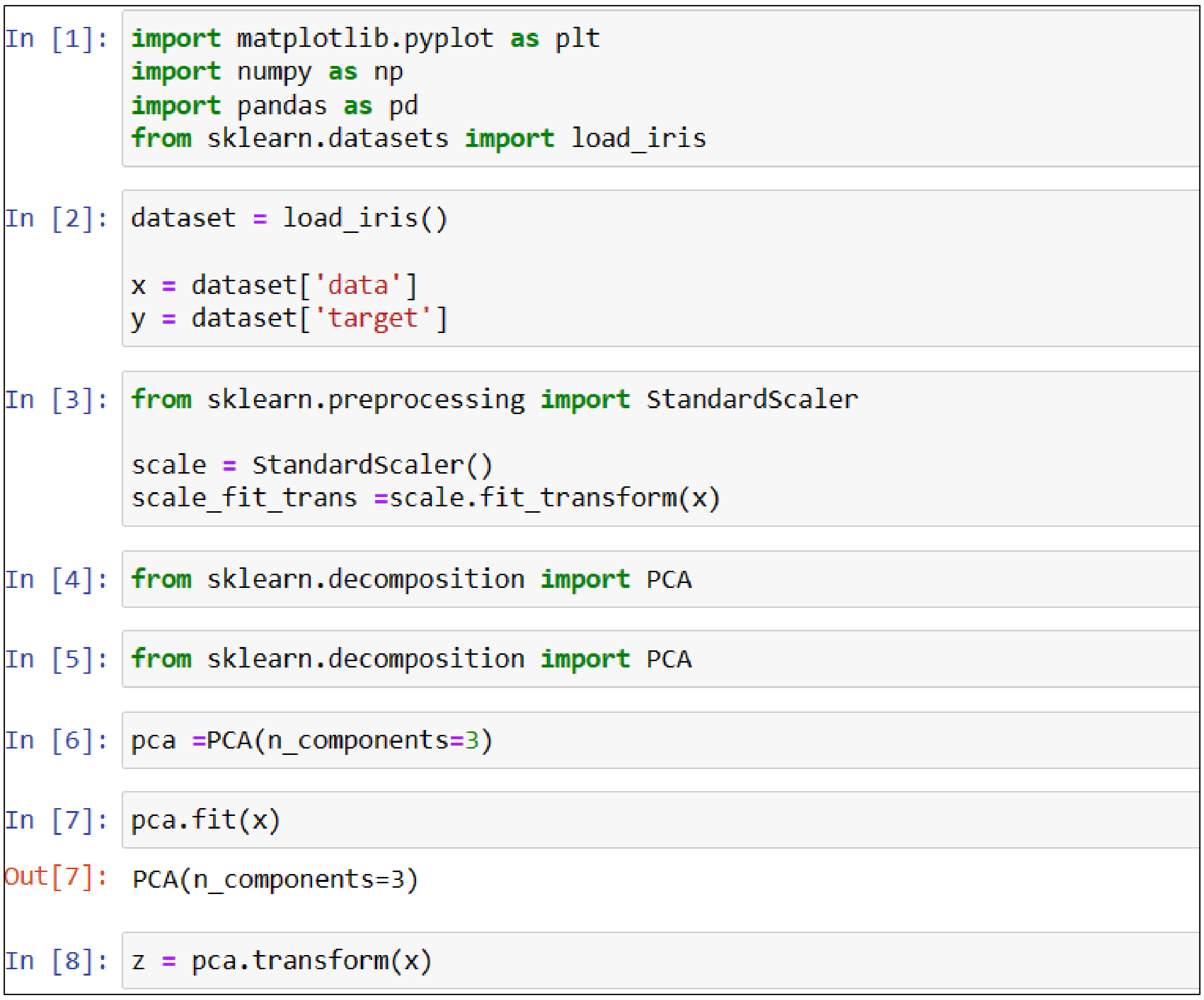
PCA is a statistical technique used for dimensionality reduction and feature extraction. It transforms correlated variables into uncorrelated principal components that explain the majority of variance in the data. It is used for dimensionality reduction, feature extraction, data visualization, noise reduction, and data compression. The data must be continuous and numeric for PCA to be appropriate.

❖ Example:

❖

Suppose we have a dataset containing the measurements of the sepal length, sepal width, petal length, and petal width of 150 iris flowers. We want to perform PCA to reduce the dimensionality of this dataset.

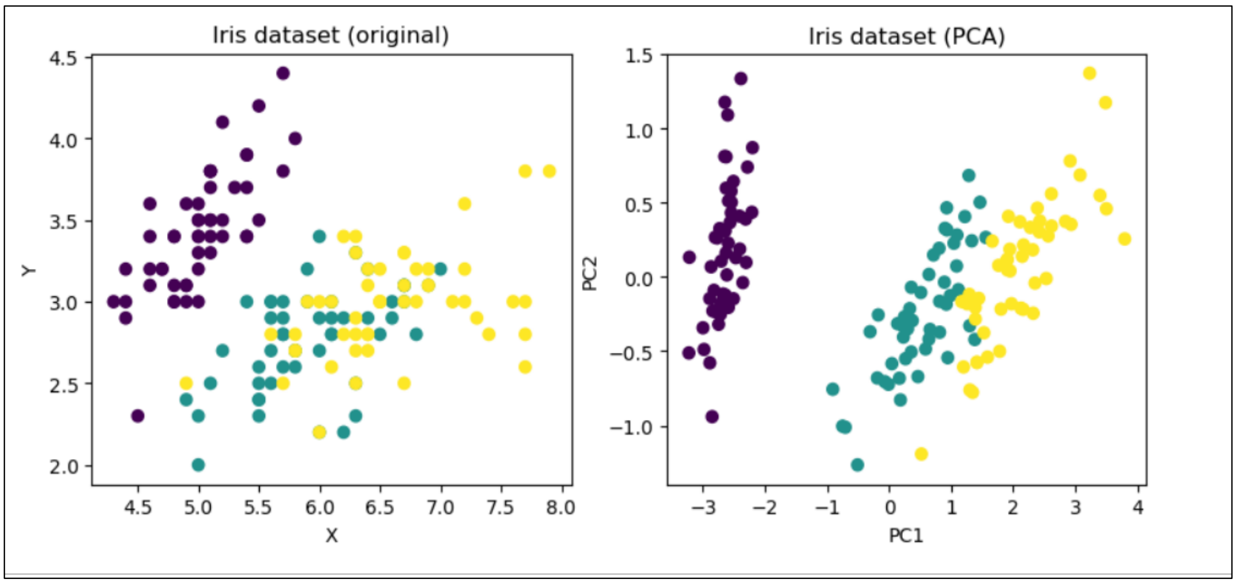
➢ Code:



```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        from sklearn.datasets import load_iris

In [2]: dataset = load_iris()

        x = dataset['data']
        y = dataset['target']

In [3]: from sklearn.preprocessing import StandardScaler

        scale = StandardScaler()
        scale_fit_trans =scale.fit_transform(x)

In [4]: from sklearn.decomposition import PCA

In [5]: from sklearn.decomposition import PCA

In [6]: pca =PCA(n_components=3)

In [7]: pca.fit(x)

Out[7]: PCA(n_components=3)

In [8]: z = pca.transform(x)
```

```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

ax[0].scatter(x[:, 0], x[:, 1], c=y)
ax[0].set_xlabel('X')
ax[0].set_ylabel('Y')
ax[0].set_title('Iris dataset (original)')

# Plot PCA results
ax[1].scatter(z[:,0], z[:,1], c=y)
ax[1].set_xlabel('PC1')
ax[1].set_ylabel('PC2')
ax[1].set_title('Iris dataset (PCA)')

plt.show()
```

➢ **Output:**



Practical-8

❖ Aim:

Real life Problem Based on Factor Analysis in Python.

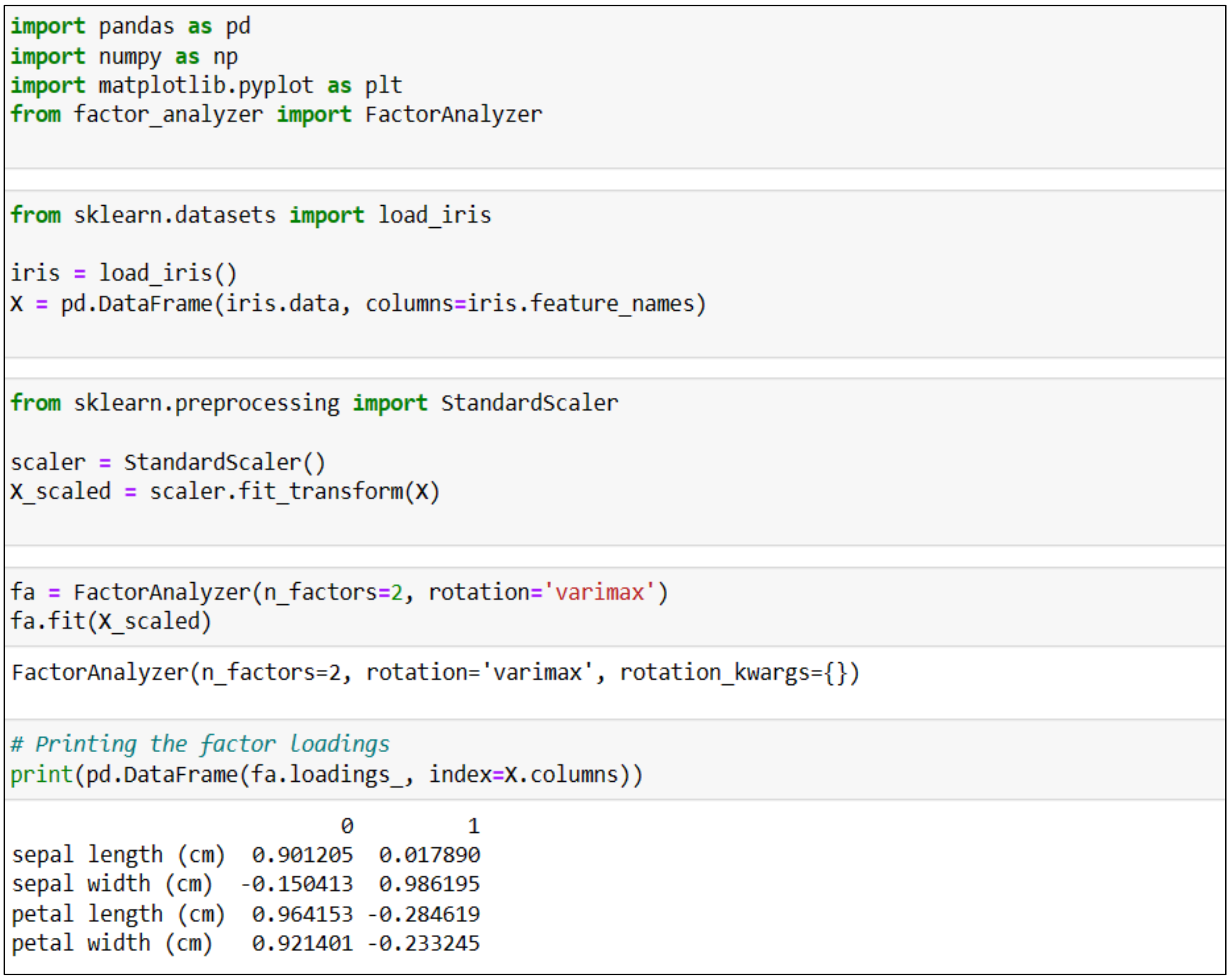
❖ Theory:

Factor analysis is a statistical method used to identify underlying factors that explain the variability among a set of observed variables. It helps researchers identify groups of variables that are related to each other and can be used to create composite variables or "factors" that represent the underlying dimensions of the data.

❖ Example:

For this example, we will be using the "iris" dataset from the scikit-learn library. This dataset contains measurements of sepal length, sepal width, petal length, and petal width for 150 iris flowers, along with their corresponding species.

➢ Code:



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from factor_analyzer import FactorAnalyzer
```

```python
from sklearn.datasets import load_iris

iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

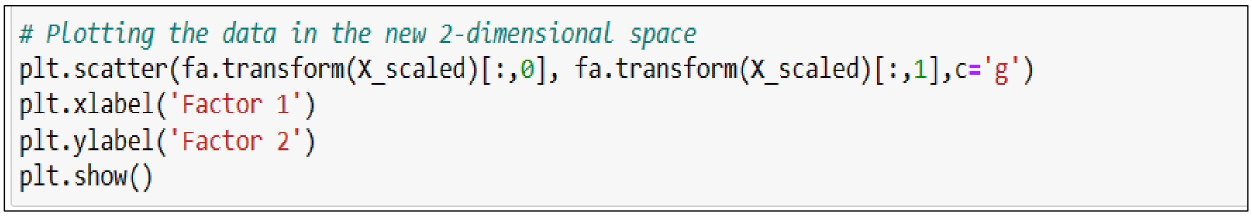
```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```python
fa = FactorAnalyzer(n_factors=2, rotation='varimax')
fa.fit(X_scaled)
```

```
FactorAnalyzer(n_factors=2, rotation='varimax', rotation_kwargs={})
```

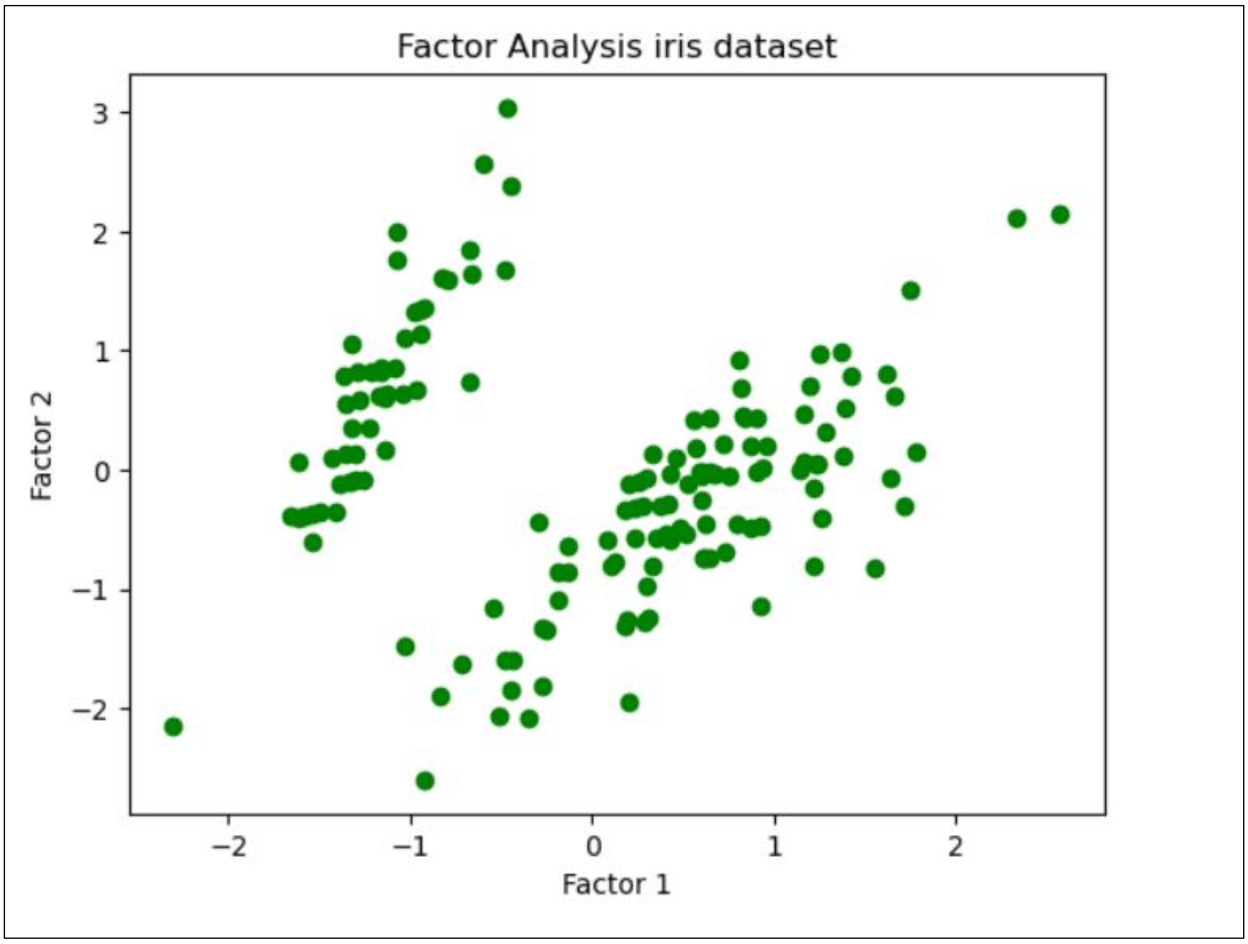
```python
# Printing the factor loadings
print(pd.DataFrame(fa.loadings_, index=X.columns))
```

```
                          0         1
sepal length (cm)  0.901205  0.017890
sepal width (cm)  -0.150413  0.986195
petal length (cm)  0.964153 -0.284619
petal width (cm)   0.921401 -0.233245
```



```python
# Plotting the data in the new 2-dimensional space
plt.scatter(fa.transform(X_scaled)[:,0], fa.transform(X_scaled)[:,1],c='g')
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.show()
```

➢ **Output:**



Factor Analysis iris dataset
Factor 2
Factor 1
3
2
1
0
−1
−2
−2
−1
0
1
2

Practical-9

❖ Aim:

Real life Problem Based on Cluster in Python.

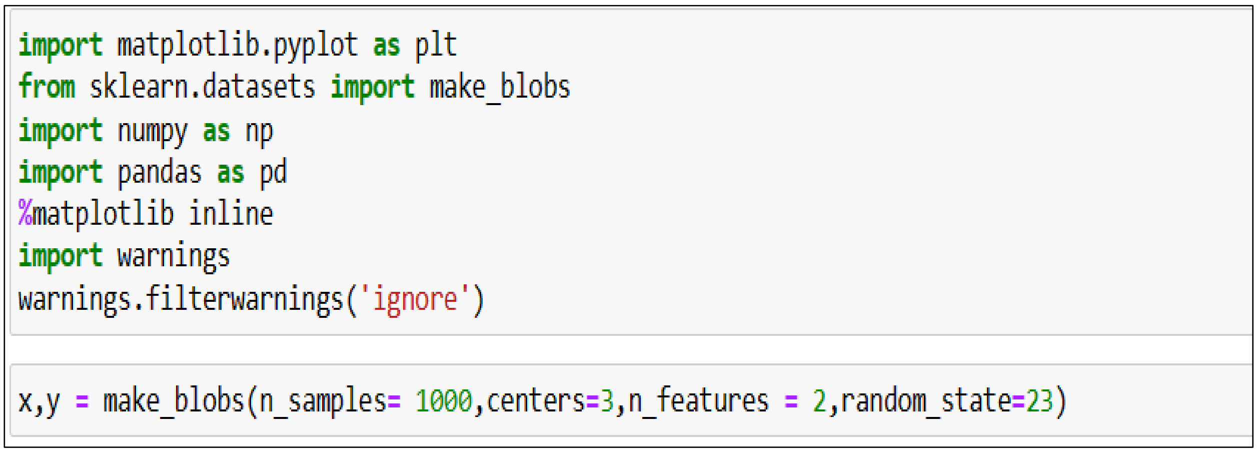
❖ Theory:

In machine learning, clustering is a type of unsupervised learning technique that groups data points based on their similarity. There are various types of clustering algorithms, each with its own strengths and weaknesses. Here are some common types of clustering algorithms used in machine learning:

❖ **K-Means Clustering:** K-Means clustering is a popular and simple algorithm that partitions the data into K clusters, where K is the number of clusters specified by the user. It works by randomly selecting K initial cluster centers, assigning each data point to the closest cluster, and then updating the cluster centers based on the mean of the data points in each cluster. K-Means clustering is used in various applications such as customer segmentation, market segmentation, and image segmentation.

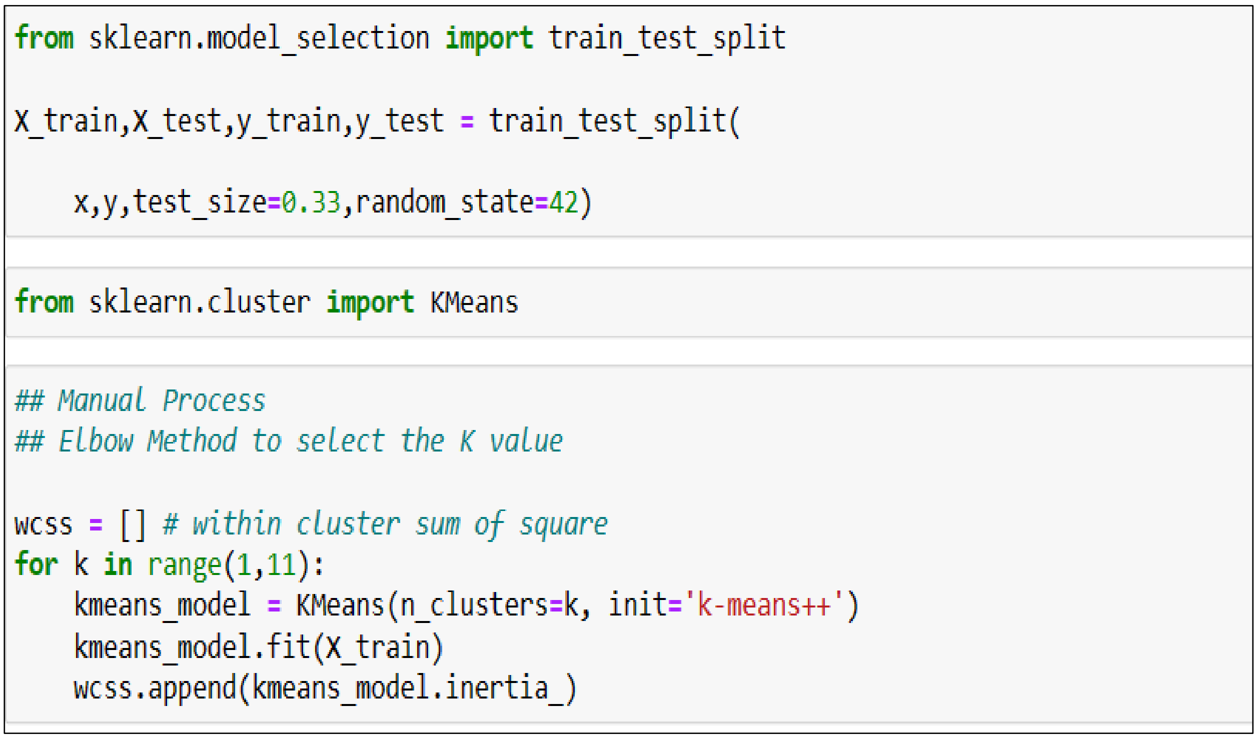
❖ Example:

➢ **Code:**



```python
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
import numpy as np
import pandas as pd
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```python
x,y = make_blobs(n_samples= 1000,centers=3,n_features = 2,random_state=23)
```



```python
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(

    x,y,test_size=0.33,random_state=42)
```

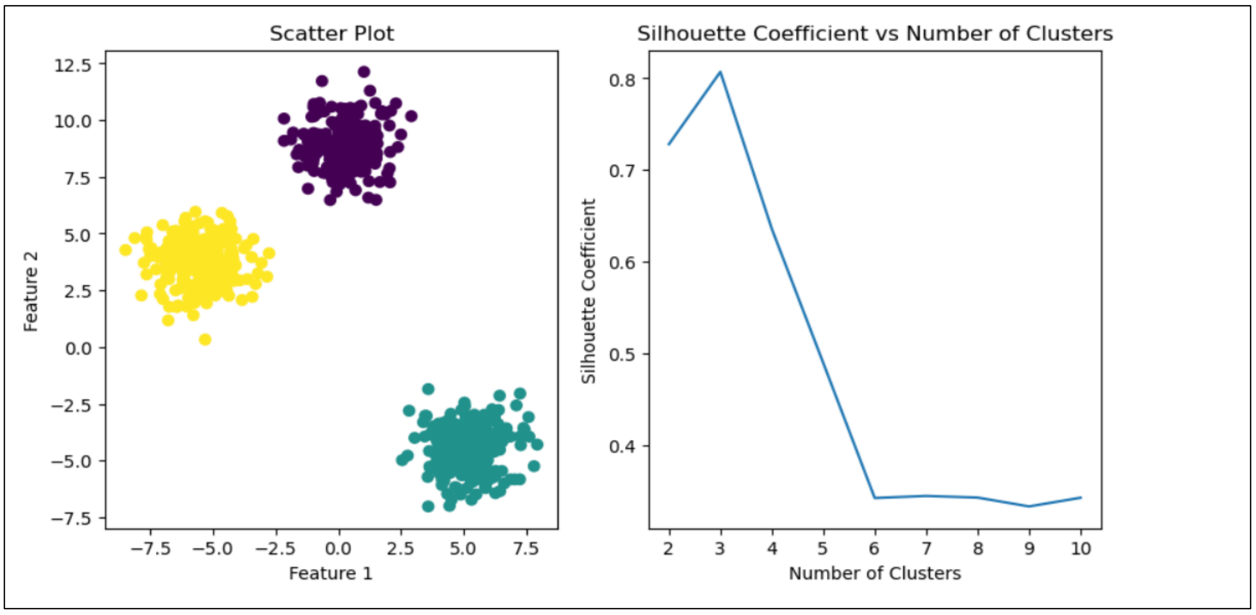
```python
from sklearn.cluster import KMeans
```

```python
## Manual Process
## Elbow Method to select the K value

wcss = [] # within cluster sum of square
for k in range(1,11):
    kmeans_model = KMeans(n_clusters=k, init='k-means++')
    kmeans_model.fit(X_train)
    wcss.append(kmeans_model.inertia_)
```

```
In [16]: from kneed import KneeLocator

In [17]: kl =KneeLocator(range(1,11),wcss,curve ="convex",direction='decreasing')

In [18]: kl.elbow

Out[18]: 3

In [19]: from sklearn.metrics import silhouette_score

In [20]: silhouette_coeff = [] # within cluster sum of square
         for k in range(2,11):
             kmeans_model = KMeans(n_clusters=k, init='k-means++')
             kmeans_model.fit(X_train)
             score = silhouette_score(X_train,kmeans_model.labels_)
             silhouette_coeff.append(score)

In [22]: plt.plot(range(2,11),silhouette_coeff)
         plt.xticks(range(2,11))
         plt.xlabel("Number of Cluters")
         plt.ylabel("Silhoutte Coeffecient")
         plt.show()
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

# Scatter plot
ax1.scatter(X_train[:,0], X_train[:,1], c=y_label)
ax1.set_xlabel("Feature 1")
ax1.set_ylabel("Feature 2")
ax1.set_title("Scatter Plot")

# Line chart
ax2.plot(range(2,11), silhouette_coeff)
ax2.set_xticks(range(2,11))
ax2.set_xlabel("Number of Clusters")
ax2.set_ylabel("Silhouette Coefficient")
ax2.set_title("Silhouette Coefficient vs Number of Clusters")

plt.show()
```

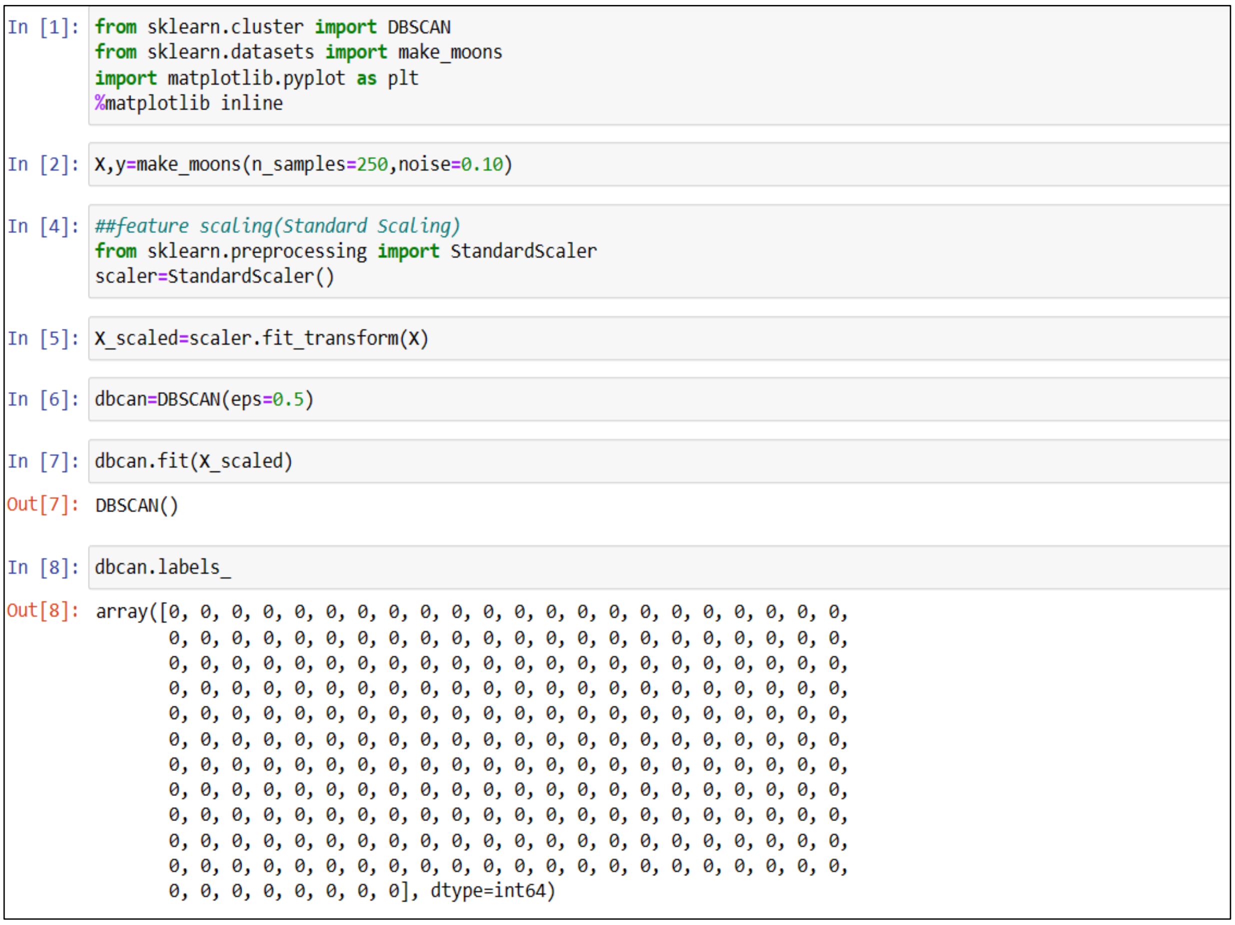
➢ **Output:**



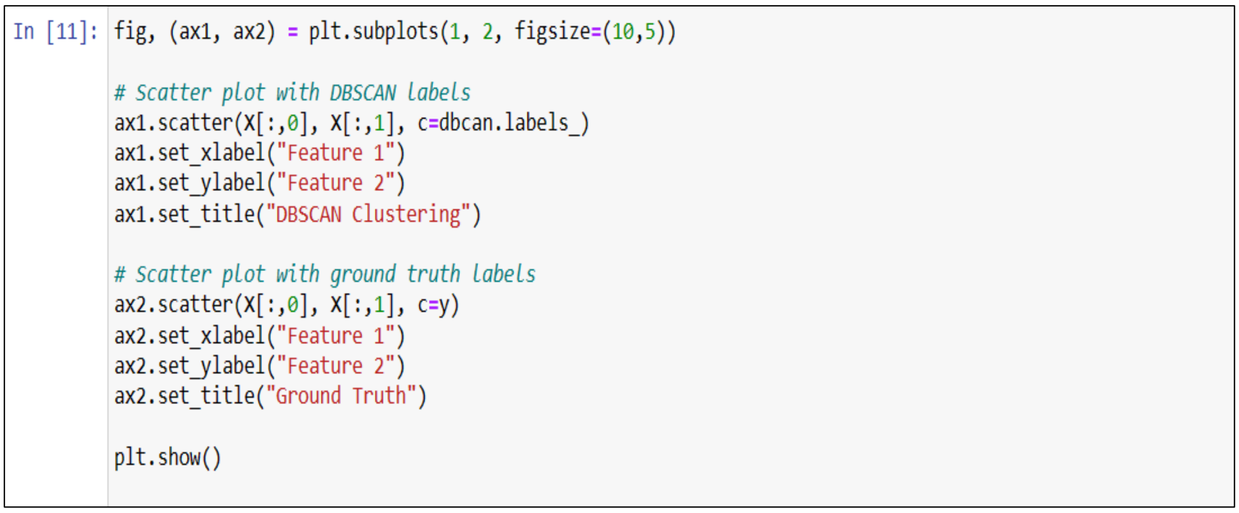
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):** DBSCAN is a density-based clustering algorithm that groups together data points that are close to each other and separates outliers as noise. It works by defining a radius around each data point and finding clusters based on the density of data points within that radius. DBSCAN is commonly used in applications such as anomaly detection, spatial clustering, and image segmentation.

- **Example:**

- **Code:**

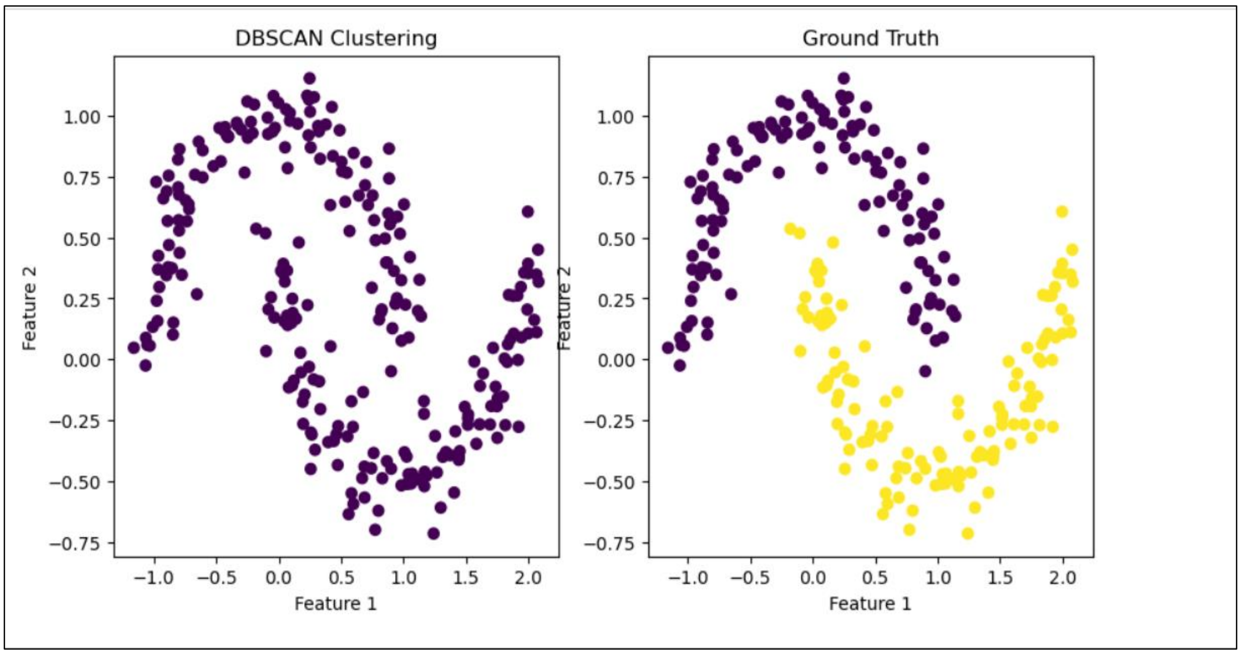


```
In [1]: from sklearn.cluster import DBSCAN
        from sklearn.datasets import make_moons
        import matplotlib.pyplot as plt
        %matplotlib inline

In [2]: X,y=make_moons(n_samples=250,noise=0.10)

In [4]: ##feature scaling(Standard Scaling)
        from sklearn.preprocessing import StandardScaler
        scaler=StandardScaler()

In [5]: X_scaled=scaler.fit_transform(X)

In [6]: dbcan=DBSCAN(eps=0.5)

In [7]: dbcan.fit(X_scaled)

Out[7]: DBSCAN()

In [8]: dbcan.labels_

Out[8]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```



```
In [11]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

         # Scatter plot with DBSCAN labels
         ax1.scatter(X[:,0], X[:,1], c=dbcan.labels_)
         ax1.set_xlabel("Feature 1")
         ax1.set_ylabel("Feature 2")
         ax1.set_title("DBSCAN Clustering")

         # Scatter plot with ground truth labels
         ax2.scatter(X[:,0], X[:,1], c=y)
         ax2.set_xlabel("Feature 1")
         ax2.set_ylabel("Feature 2")
         ax2.set_title("Ground Truth")

         plt.show()
```

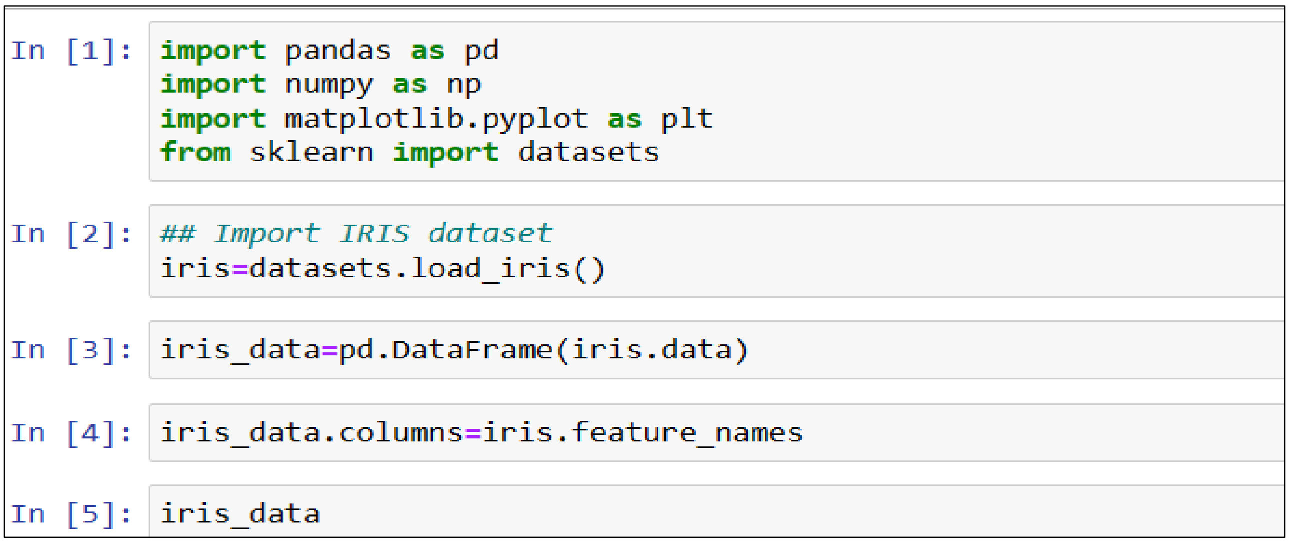
➢ **OUTPUT:**



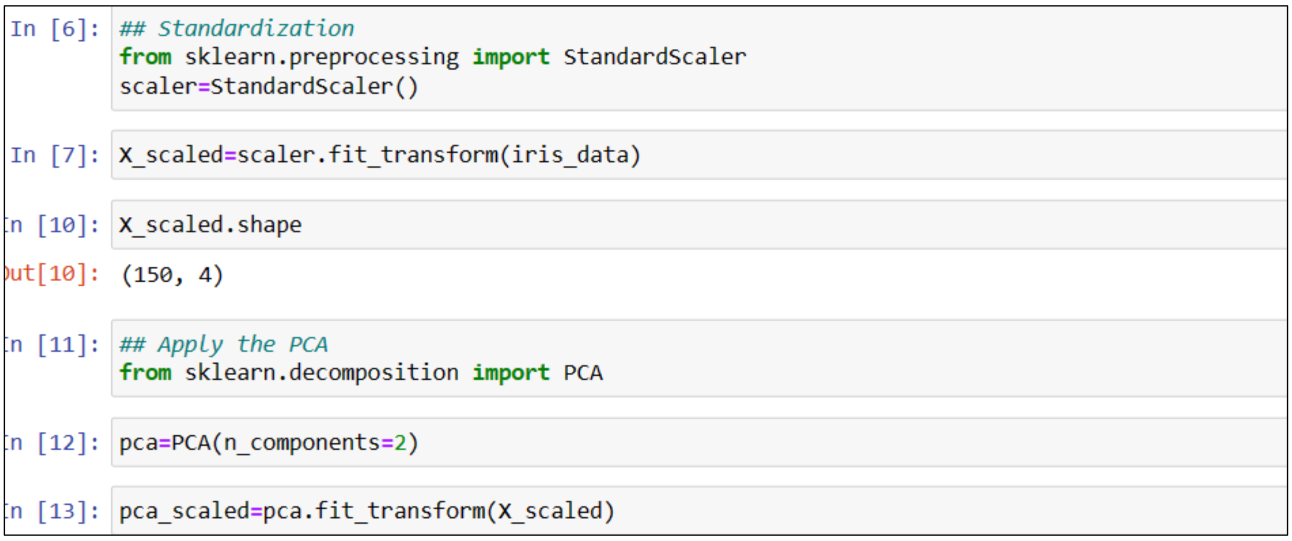
❖ **Hierarchical Clustering:** Hierarchical clustering is a family of clustering algorithms that creates a hierarchy of clusters using a bottom-up or top-down approach. It works by iteratively merging or splitting clusters based on the distance between them until a stopping criterion is met. Hierarchical clustering can be agglomerative (bottom-up) or divisive (top-down), and is often used in applications such as gene expression analysis, market segmentation, and image segmentation.

❖ **Example:**
The Iris dataset is a commonly used dataset in machine learning for classification and clustering tasks. It consists of 150 samples, each with 4 features: sepal length, sepal width, petal length, and petal width. The samples belong to 3 different species of iris flowers: setosa, versicolor, and virginica.

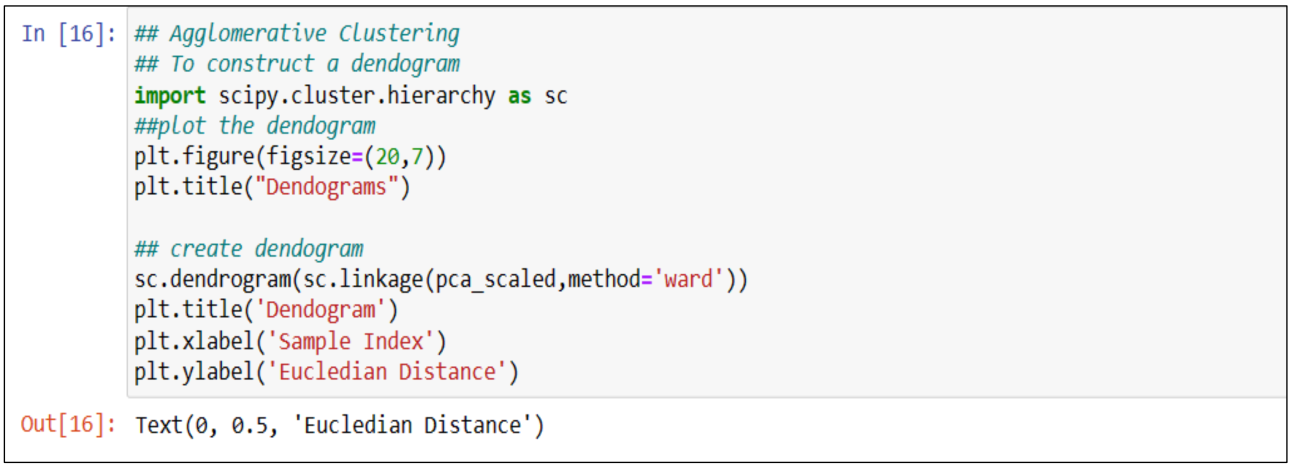
➢ **Code:**



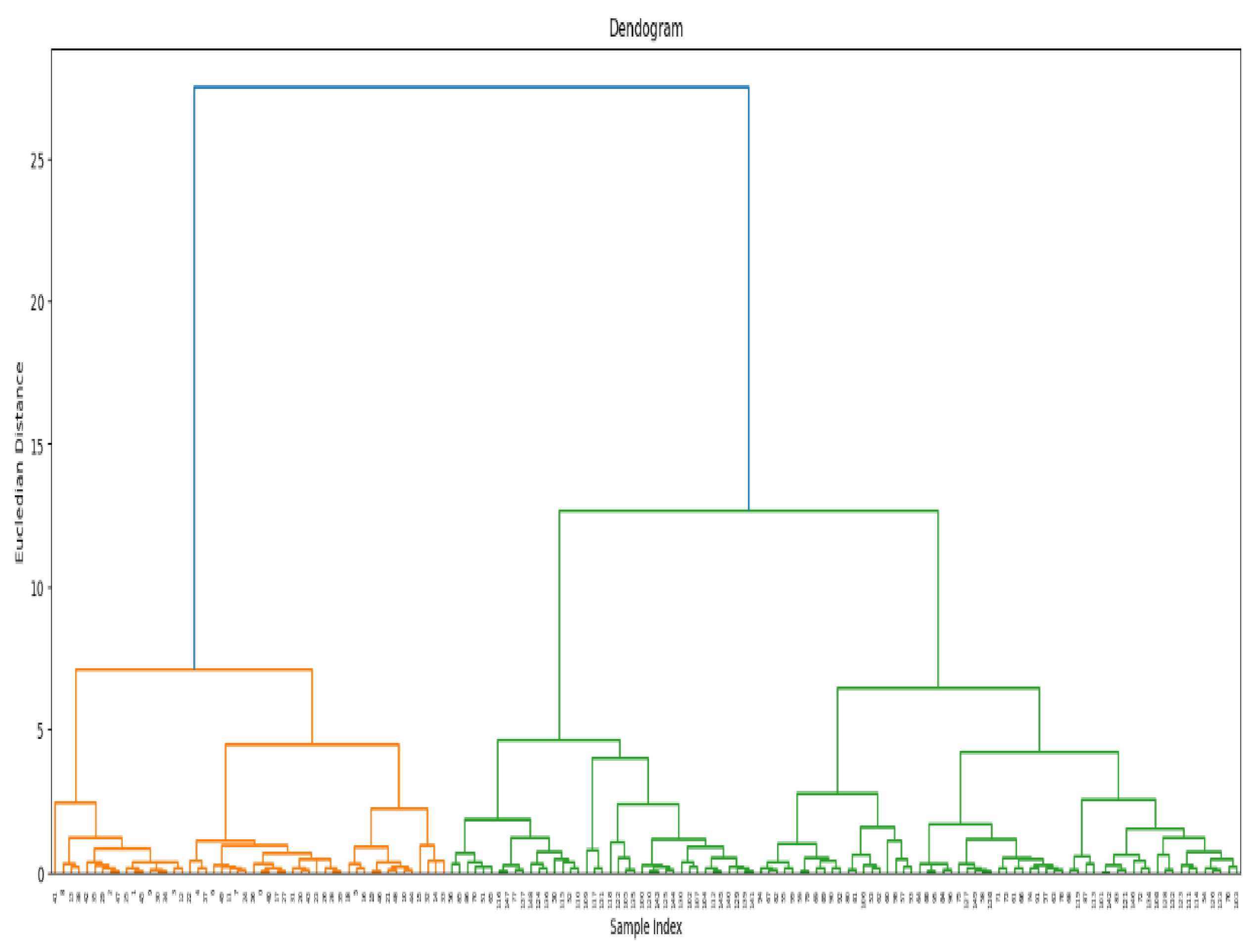
```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import datasets

In [2]: ## Import IRIS dataset
        iris=datasets.load_iris()

In [3]: iris_data=pd.DataFrame(iris.data)

In [4]: iris_data.columns=iris.feature_names

In [5]: iris_data
```

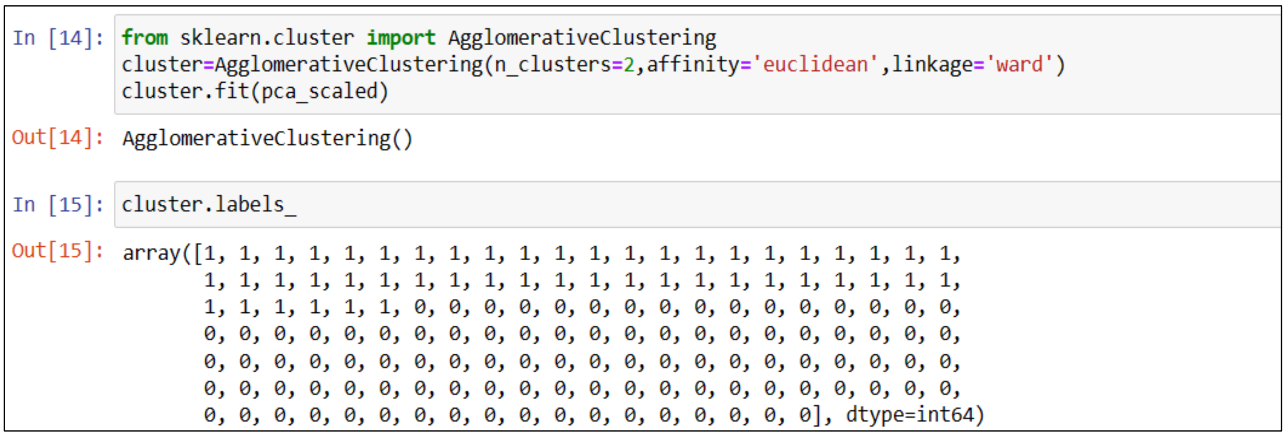


```
In [6]:  ## Standardization
         from sklearn.preprocessing import StandardScaler
         scaler=StandardScaler()

In [7]:  X_scaled=scaler.fit_transform(iris_data)

n [10]:  X_scaled.shape

ut[10]:  (150, 4)

n [11]:  ## Apply the PCA
         from sklearn.decomposition import PCA

n [12]:  pca=PCA(n_components=2)

n [13]:  pca_scaled=pca.fit_transform(X_scaled)
```

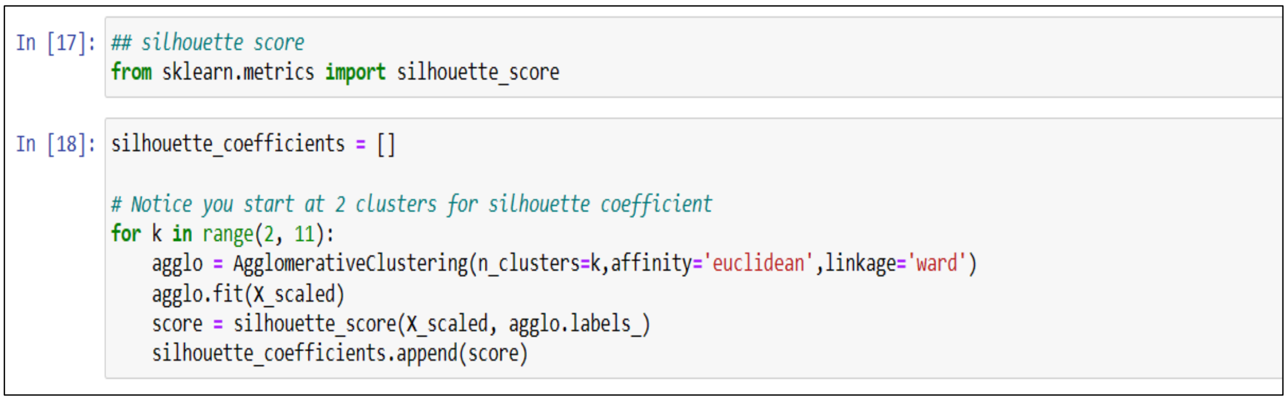


```
In [16]:  ## Agglomerative Clustering
          ## To construct a dendogram
          import scipy.cluster.hierarchy as sc
          ##plot the dendogram
          plt.figure(figsize=(20,7))
          plt.title("Dendograms")

          ## create dendogram
          sc.dendrogram(sc.linkage(pca_scaled,method='ward'))
          plt.title('Dendogram')
          plt.xlabel('Sample Index')
          plt.ylabel('Eucledian Distance')

Out[16]:  Text(0, 0.5, 'Eucledian Distance')
```

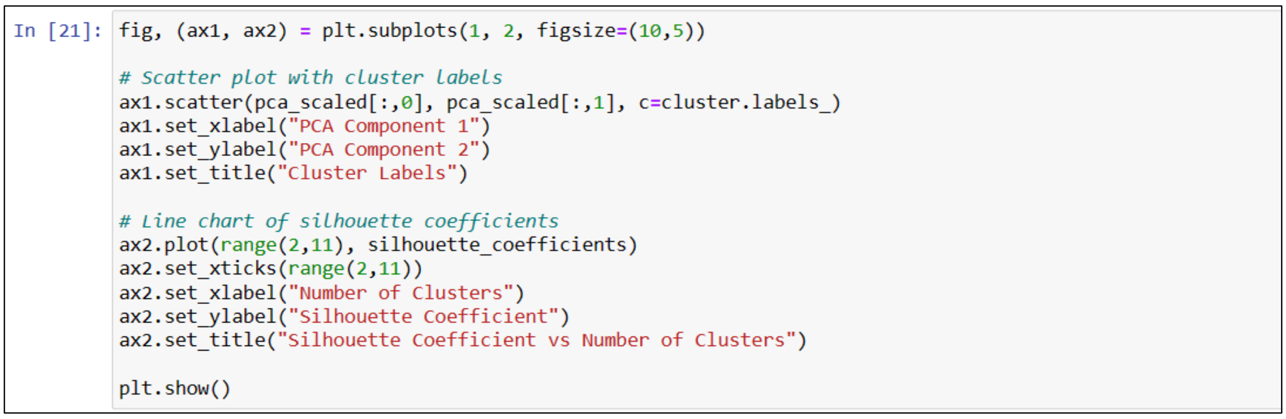




```
In [14]: from sklearn.cluster import AgglomerativeClustering
         cluster=AgglomerativeClustering(n_clusters=2,affinity='euclidean',linkage='ward')
         cluster.fit(pca_scaled)

Out[14]: AgglomerativeClustering()

In [15]: cluster.labels_

Out[15]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

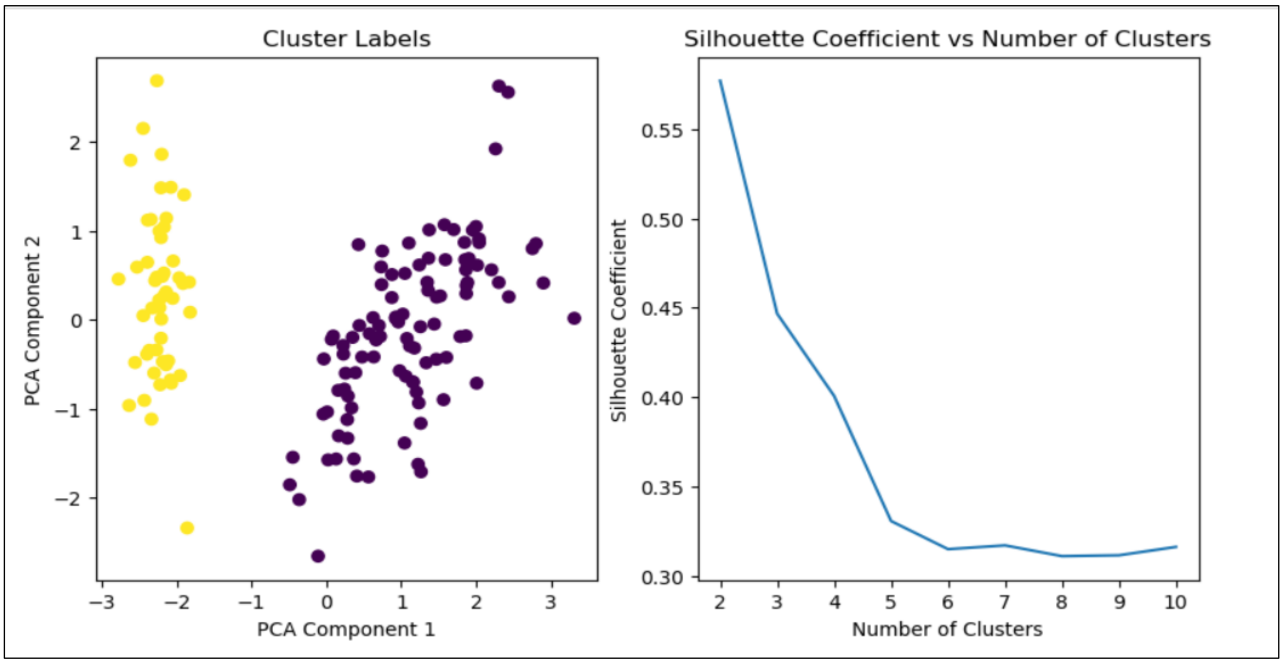


```
In [17]: ## silhouette score
         from sklearn.metrics import silhouette_score

In [18]: silhouette_coefficients = []

         # Notice you start at 2 clusters for silhouette coefficient
         for k in range(2, 11):
             agglo = AgglomerativeClustering(n_clusters=k,affinity='euclidean',linkage='ward')
             agglo.fit(X_scaled)
             score = silhouette_score(X_scaled, agglo.labels_)
             silhouette_coefficients.append(score)
```



```
In [21]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

         # Scatter plot with cluster labels
         ax1.scatter(pca_scaled[:,0], pca_scaled[:,1], c=cluster.labels_)
         ax1.set_xlabel("PCA Component 1")
         ax1.set_ylabel("PCA Component 2")
         ax1.set_title("Cluster Labels")

         # Line chart of silhouette coefficients
         ax2.plot(range(2,11), silhouette_coefficients)
         ax2.set_xticks(range(2,11))
         ax2.set_xlabel("Number of Clusters")
         ax2.set_ylabel("Silhouette Coefficient")
         ax2.set_title("Silhouette Coefficient vs Number of Clusters")

         plt.show()
```

- **Output:**



Practical-10

❖ Aim:

Real life problem based on Point Estimation and Interval Estimation using Python.

❖ Theory:

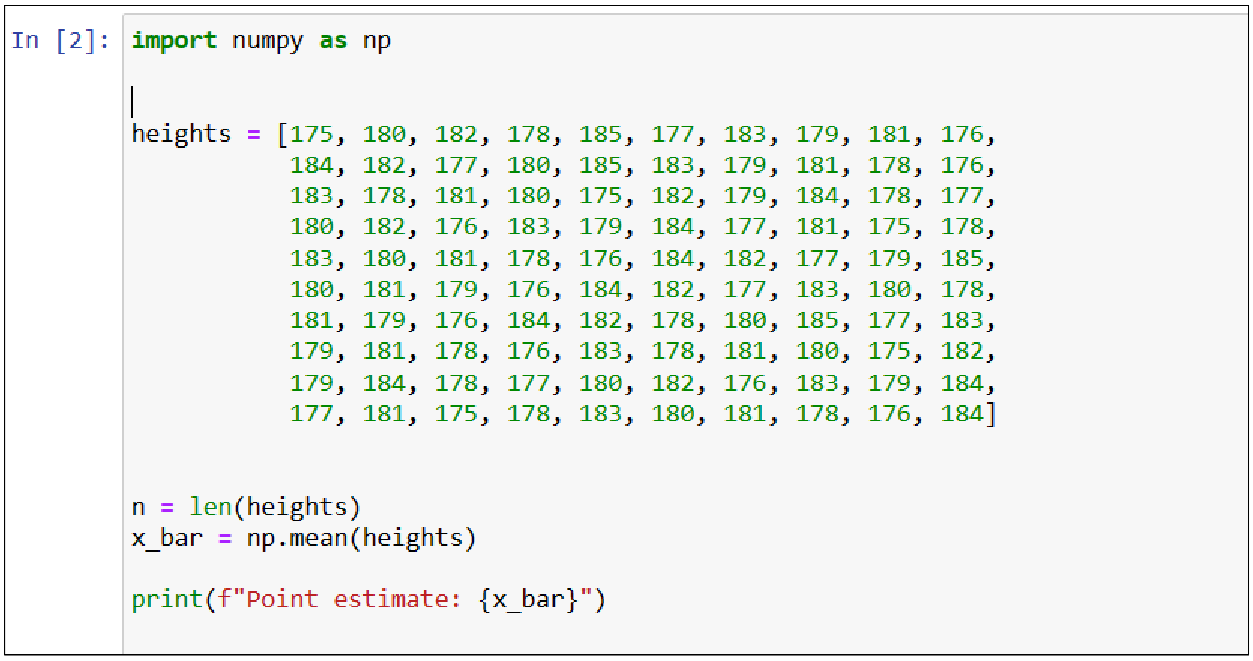
Point Estimation: Point estimation is a method of estimating a population parameter using a single value or point. The point estimate is typically calculated using a statistic that is calculated from the sample data. For example, the sample mean or sample proportion can be used as a point estimate of the population mean or population proportion, respectively. Point estimates can be biased or unbiased, and their accuracy and precision depend on the sample size and the variability of the population.

❖ Conditions for point estimation:

- The sample is selected randomly from the population.
- The sample is representative of the population.
- The sample size is large enough to ensure that the point estimate is unbiased and has a small standard error.

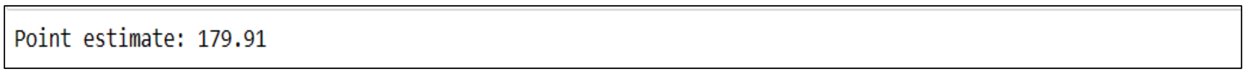
❖ **Problem:** Problem: A college wants to estimate the average height of its male students. The college takes a random sample of 100 male students and measures their heights in centimeters. Find a point estimate for the population mean height.

➢ Code:



```
In [2]: import numpy as np

        heights = [175, 180, 182, 178, 185, 177, 183, 179, 181, 176,
                   184, 182, 177, 180, 185, 183, 179, 181, 178, 176,
                   183, 178, 181, 180, 175, 182, 179, 184, 178, 177,
                   180, 182, 176, 183, 179, 184, 177, 181, 175, 178,
                   183, 180, 181, 178, 176, 184, 182, 177, 179, 185,
                   180, 181, 179, 176, 184, 182, 177, 183, 180, 178,
                   181, 179, 176, 184, 182, 178, 180, 185, 177, 183,
                   179, 181, 178, 176, 183, 178, 181, 180, 175, 182,
                   179, 184, 178, 177, 180, 182, 176, 183, 179, 184,
                   177, 181, 175, 178, 183, 180, 181, 178, 176, 184]


        n = len(heights)
        x_bar = np.mean(heights)

        print(f"Point estimate: {x_bar}")
```

➢ Output:



```
Point estimate: 179.91
```

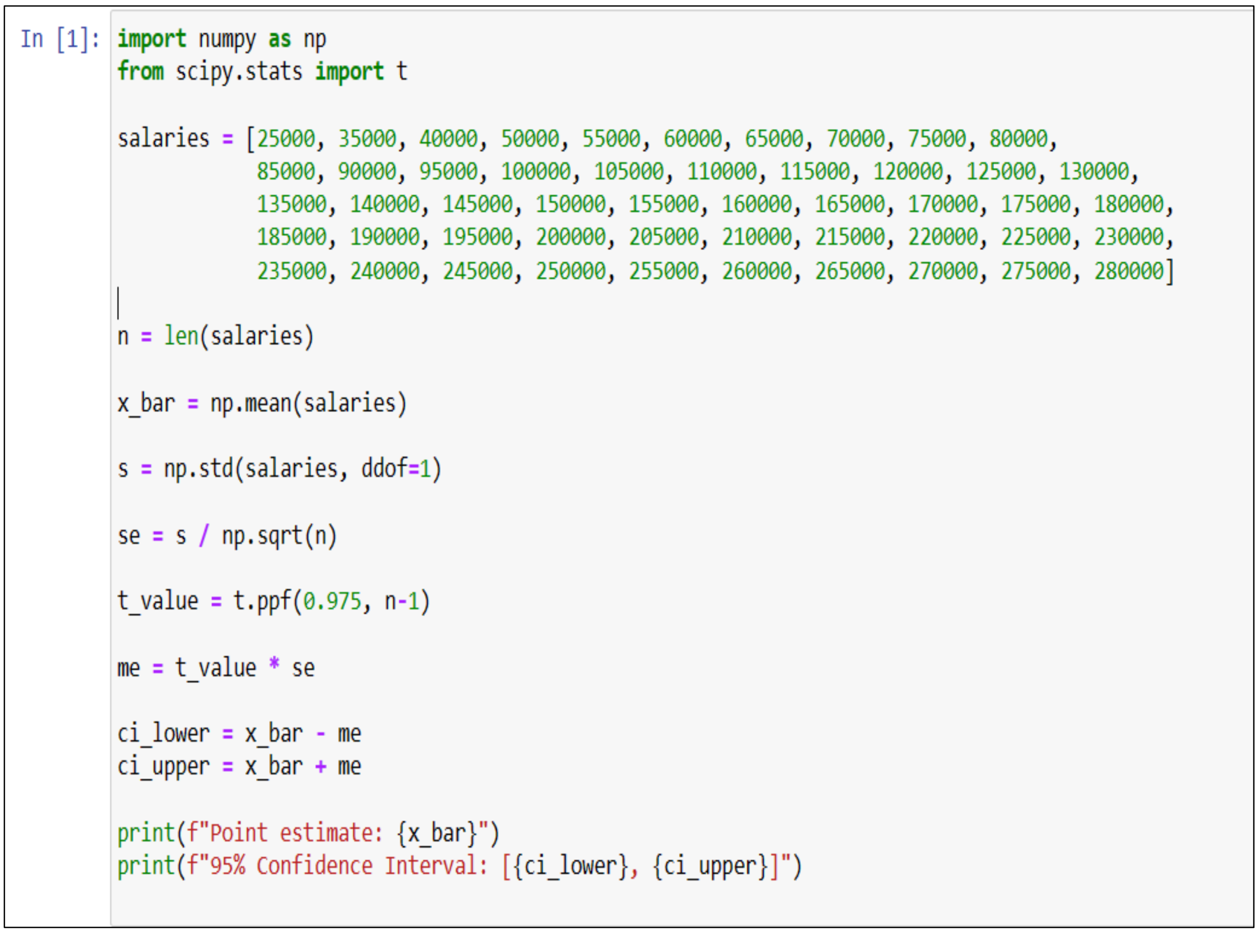
❖ **Interval Estimation:** Interval estimation is a method of estimating a population parameter by calculating a range of values that is likely to contain the true population parameter with a certain level of confidence. This range is called a confidence interval, and it is calculated based on the sample data and the desired level of confidence. The width of the confidence interval depends on the sample size, the variability of the population, and the desired level of confidence. A wider interval indicates more uncertainty, while a narrower interval indicates more precision.

❖ **Conditions for interval estimation:**

- The sample is selected randomly from the population.
- The sample is representative of the population.
- The population follows a normal distribution or the sample size is large enough to apply the Central Limit Theorem.
- The sample is independent and identically distributed (iid).
- The standard deviation or variance of the population is known, or the sample size is large enough to estimate it accurately using the sample data.

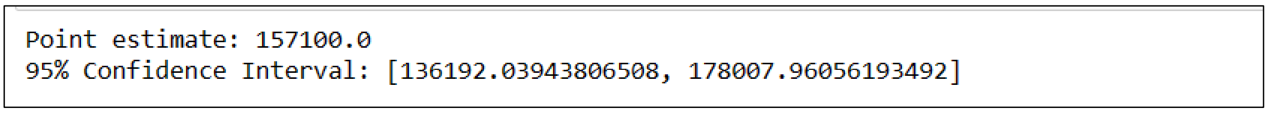
❖ **Problem:** A company wants to estimate the average income of its employees. The company has a dataset containing the salaries of 50 employees. Find a point estimate and a 95% confidence interval for the population mean income.

➢ **Code:**



```
In [1]: import numpy as np
        from scipy.stats import t

        salaries = [25000, 35000, 40000, 50000, 55000, 60000, 65000, 70000, 75000, 80000,
                    85000, 90000, 95000, 100000, 105000, 110000, 115000, 120000, 125000, 130000,
                    135000, 140000, 145000, 150000, 155000, 160000, 165000, 170000, 175000, 180000,
                    185000, 190000, 195000, 200000, 205000, 210000, 215000, 220000, 225000, 230000,
                    235000, 240000, 245000, 250000, 255000, 260000, 265000, 270000, 275000, 280000]

        n = len(salaries)

        x_bar = np.mean(salaries)

        s = np.std(salaries, ddof=1)

        se = s / np.sqrt(n)

        t_value = t.ppf(0.975, n-1)

        me = t_value * se

        ci_lower = x_bar - me
        ci_upper = x_bar + me

        print(f"Point estimate: {x_bar}")
        print(f"95% Confidence Interval: [{ci_lower}, {ci_upper}]")
```

➢ Output:



```
Point estimate: 157100.0
95% Confidence Interval: [136192.03943806508, 178007.96056193492]
```