Live Module Assignment

Name : Dhritimalya Dutta

Reg Email : [dhritimalyadutta@gmail.com](mailto:dhritimalyadutta@gmail.com)

Course name : Full Stack Data Science Pro

Submission Date:

Git link : https://github.com/Dhritionly/Live-Assignment.git

### **Question 1.1: Write the Answer to these questions.**

**1. What is the difference between static and dynamic variables in Python?**

Python uses dynamic typing, meaning variables don't have a fixed type. Their type is determined at runtime based on the value assigned to them. Static typing, found in languages like Java or C++, requires variables to have their type declared before use.

**2. Explain the purpose of "pop", "popitem", "clear()" in a dictionary with suitable examples.**

**pop(key):** Removes the item associated with the specified key and returns its value. If the key is not found, it raises a KeyError.  
  
**my\_dict = {"a": 1, "b": 2, "c": 3}**

**value = my\_dict.pop("b")** # Removes "b" and returns 2

**popitem():** Removes and returns an arbitrary item (key-value pair) from the dictionary.  
  
**item = my\_dict.popitem()** # Removes and returns a random item

**clear():** Removes all items from the dictionary.  
  
**my\_dict.clear() # Empties the dictionary**

**3. What do you mean by FrozenSet? Explain it with suitable examples.**

A frozenset is an immutable version of a set. Once created, its elements cannot be added, removed, or modified. It's useful for situations where you want to ensure the set remains constant.

**my\_set = {1, 2, 3}**

**frozen\_set = frozenset(my\_set) # Creates a frozenset from the set**

**4. Differentiate between mutable and immutable data types in Python and give examples of mutable and immutable data types.**

* **Mutable:** Values can be changed after creation. Examples: lists, dictionaries, sets.
* **Immutable:** Values cannot be changed after creation. Examples: strings, numbers, tuples.

**5. What is init? Explain with an example.**

\_\_init\_\_ is a special method in Python classes that is automatically called when an object of that class is created. It's often used to initialize attributes of the object.

**class Person:**

**def \_\_init\_\_(self, name, age):**

**self.name = name**

**self.age = age**

**person1 = Person("Alice", 30)** # \_\_init\_\_ is called to initialize name and age

**Control Flow and Functions**

**6. What is docstring in Python? Explain with an example.**

A docstring is a string that describes the purpose, parameters, return values, and usage of a function, class, or method. It's accessible using the \_\_doc\_\_ attribute.

**def greet(name):**

**"""Greets the given name."""**

**print("Hello, " + name + "!")**

**print(greet.\_\_doc\_\_)** # Prints the docstring

**7. What are unit tests in Python?**

Unit tests are small, isolated tests that verify the correctness of individual units of code, such as functions or methods. They help ensure code quality and maintainability.

**8. What is break, continue and pass in Python?**

* **break:** Terminates the execution of the innermost loop.
* **continue:** Skips the current iteration and moves to the next one.
* **pass:** Does nothing. Often used as a placeholder when a statement is required syntactically but no action is needed.

**9. What is the use of self in Python?**

self is a reference to the current object within a class method. It allows you to access and modify the object's attributes.

**10. What are global, protected and private attributes in Python?** Python doesn't have strict protection mechanisms like other languages. However, conventions are used:

* **Global:** Variables declared outside of functions or classes.
* **Protected:** Attributes prefixed with an underscore (\_). This indicates they should be treated as private, but not strictly enforced.
* **Private:** Attributes prefixed with double underscores (\_\_). This makes them inaccessible from outside the class.

**11. What are modules and packages in Python?**

* **Modules:** Single Python files containing functions, classes, and variables.
* **Packages:** A directory containing multiple modules, organized into a hierarchical structure.

### 

**12. What are lists and tuples? What is the key difference between the two?**

* **Lists:** Ordered collections of elements, mutable, can be modified.
* **Tuples:** Ordered collections of elements, immutable, cannot be modified after creation.

### 

**13. What is an Interpreted language & dynamically typed language? Write 5 differences between them.**

Both Python and JavaScript are interpreted and dynamically typed languages. Here are some key differences:

1. **Execution:** Interpreted languages execute code line by line, while compiled languages translate the entire code into machine code before execution.
2. **Performance:** Compiled languages generally have better performance due to direct machine code execution.
3. **Portability:** Interpreted languages are more portable as they don't require compilation for different platforms.
4. **Debugging:** Interpreted languages often have more interactive debugging tools.
5. **Development Cycle:** Interpreted languages can have faster development cycles due to not requiring recompilation for every change.

**14. What are Dict and List comprehensions?**

Comprehensions provide a concise way to create lists, dictionaries, and sets.

**List comprehension:  
  
squared\_numbers = [x\*\*2 for x in range(5)]**

**Dict comprehension:**  
**my\_dict = {key: value\*\*2 for key, value in my\_dict.items()}**

**15. What are decorators in Python? Explain it with an example. Write down its use cases.**

Decorators are functions that modify the behavior of other functions or classes without directly altering their code.

**def timer(func):**

**def wrapper(\*args, \*\*kwargs):**

**start\_time = time.time()**

**result = func(\*args, \*\*kwargs)**

**end\_time = time.time()**

**print(f"Function {func.\_\_name\_\_} took {end\_time - start\_time} seconds")**

**return result**

**return wrapper**

**@timer**

**def my\_function():**

# Some code here

**Use cases:**

* Timing functions
* Caching results
* Logging function calls
* Adding authentication or authorization
* Tracing function execution

### 

**16. How is memory managed in Python?**

Python uses automatic memory management through garbage collection. The interpreter automatically reclaims memory that is no longer in use.

### 

**17. What is lambda in Python? Why is it used?**

A lambda function is a small, anonymous function defined using the lambda keyword. It's often used for simple functions that are only needed once.

**my\_lambda = lambda x: x\*\*2**

**result = my\_lambda(3)**  # Result will be 9

### 

**18. Explain split() and join() functions in Python?**

* **split():** Splits a string into a list of substrings based on a delimiter.

**my\_string = "hello world"**

**words = my\_string.split()**  # ['hello', 'world']

* **join():** Joins a list of strings into a single string using a separator.

**words = ['hello', 'world']**

**joined\_string = " ".join(words)** # "hello world"

**19. What are iterators, iterable & generators in Python?**

* **Iterable:** An object that can be iterated over, such as lists, tuples, strings, and dictionaries.
* **Iterator:** An object that implements the \_\_iter\_\_ and \_\_next\_\_ methods, allowing it to be iterated over.
* **Generator:** A special type of iterator that generates values on-the-fly using the yield keyword, often used for memory-efficient iteration over large datasets.

### 

**20. What is the difference between xrange and range in Python?**

* **range (Python 2):** Creates a list of all numbers in the specified range, consuming memory.
* **xrange (Python 2):** Creates an iterator that generates numbers on-the-fly, saving memory. In Python 3, range behaves like xrange from Python 2.

### 

**21. Pillars of Oops.** The four pillars of OOP are:

* **Inheritance:** Allows classes to inherit attributes and methods from a parent class.
* **Encapsulation:** Bundling data and methods within an object to protect data integrity.
* **Polymorphism:** Ability of objects of different types to be treated as if they were of the same type.
* **Abstraction:** Hiding unnecessary details and providing a simplified interface.

**22. How will you check if a class is a child of another class?**

Use the issubclass() function:

**23. How does inheritance work in Python? Explain all types of inheritance with an example.**

Inheritance in Python allows classes to inherit attributes and methods from a parent class. This promotes code reuse and creates a hierarchical relationship between classes.

**Types of Inheritance:**

**Single Inheritance:**

A class inherits from only one parent class.  
  
class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Dog(Animal):

def bark(self):

print("Woof!")

* **Multiple Inheritance:**

A class inherits from multiple parent classes.

class Flyer: def fly(self): print("Flying...")

class Swimmer: def swim(self): print("Swimming...")

class FlyingFish(Flyer, Swimmer): pass

- \*\*Multilevel Inheritance:\*\* A class inherits from a class that itself inherits from another class.

```python

class Vehicle:

pass

class Car(Vehicle):

pass

class ElectricCar(Car):

pass

* **Hybrid Inheritance:**

A combination of multiple and multilevel inheritance.

**24. What is encapsulation? Explain it with an example.**

Encapsulation is the bundling of data and methods within an object to protect data integrity and control access. It promotes data hiding and modularity.

class BankAccount:

def \_\_init\_\_(self, balance):

self.\_\_balance = balance # Private attribute

def deposit(self, amount):

self.\_\_balance += amount

def withdraw(self, amount):

if self.\_\_balance >= amount:

self.\_\_balance -= amount

else:

print("Insufficient funds")

def get\_balance(self):

return self.\_\_balance

In this example, the \_\_balance attribute is private, ensuring it can only be accessed and modified through the public methods deposit, withdraw, and get\_balance.

**25. What is polymorphism? Explain it with an example.**

Polymorphism is the ability of objects of different types to be treated as if they were of the same type. It allows for flexible and extensible code.

class Animal:

def make\_sound(self):

pass

class Dog(Animal):

def make\_sound(self):

print("Woof!")

class Cat(Animal):

def make\_sound(self):

print("Meow!")

def animal\_sounds(animals):

for animal in animals:

animal.make\_sound()

animals = [Dog(), Cat()]

animal\_sounds(animals) # Both Dog and Cat objects will make their respective sounds

In this example, the make\_sound method is overridden in the Dog and Cat classes, allowing them to have different implementations. The animal\_sounds function can treat all animals as if they were of the same type and call their make\_sound method.

## **Question 1.2:**

**Invalid identifier names:**

* **b) 1st\_Room:** Identifiers cannot start with a number.
* **c) Hundreds$:** Symbols like $ are not allowed in identifiers.
* **e) total-Marks:** Hyphens are not allowed in identifiers.
* **f) Total Marks:** Spaces are not allowed in identifiers.

**Valid identifier names:**

* **a) Serial\_no.**
* **d) Total\_Marks**
* **g) True**
* **h) \_Percentag**

## **Question 1.3:**

**a) Add "freedom\_fighter" at the 0th index:**

**Input:**

name = ["Mohan", "dash", "karam", "chandra", "gandhi", "Bapu"]

name.insert(0, "freedom\_fighter")

print(name)

**Output:**

['freedom\_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu']

**b) Find the output and explain:**

name = ["freedom\_fighter", "Bapuji", "MOhan", "dash", "karam", "chandra", "gandhi"]

length1 = len(name[-len(name) + 1:-1:2])

length2 = len(name[-len(name) + 1:-1])

print(length1 + length2)

**Output: 5**

Explanation:

* name[-len(name) + 1:-1] slices the list from the second-to-last element to the second element (excluding the last element).
* name[-len(name) + 1:-1:2] slices the same range but with a step of 2, selecting every other element.
* length1 calculates the length of the sliced list with a step of 2.
* length2 calculates the length of the sliced list without a step.
* The sum of length1 and length2 gives the total number of elements in the sliced range.

**c) Add "Netaji" and "Bose" at the end:**

name.extend(["Netaji", "Bose"])

print(name)

**Output:**

['freedom\_fighter', 'Mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu', 'Netaji', 'Bose']

**d) Value of temp:**

name = ["Bapuji", "dash", "karam", "chandra", "gandi", "Mohan"]

temp = name[-1] # temp = "Mohan"

name[-1] = name[0] # name[-1] = "Bapuji"

name[0] = temp # name[0] = "Mohan"

print(name)

**Output:** ['Mohan', 'dash', 'karam', 'chandra', 'gandi', 'Bapuji']

**Question 1.4.Find the output of the following.**

animal = ['Human', 'cat', 'mat', 'cat', 'rat', 'Human', 'Lion']

print(animal.count('Human')) # Output: 2

print(animal.index('rat')) # Output: 4

print(len(animal)) # Output: 7

**Explanation:**

animal.count('Human') counts the number of occurrences of "Human" in the list.

animal.index('rat') finds the index of the first occurrence of "rat" in the list.

len(animal) calculates the length of the list.

**Question 1.5.**

tuple1=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22),[{"roll\_no":1},

{"name":"Navneet"}])

tuple1 = (10, 20, "Apple", 3.4, 'a', ["master", "ji"], ("sita", "geeta", 22), [{"roll\_no": 1}, {"name": "Navneet"}])

a) print(len(tuple1)) # Output: 8

b) print(tuple1[-1][-1]["name"]) # Output: Navneet

c) print(tuple1[7][0]["roll\_no"]) # Output: 1

d) print(tuple1[-3][1]) # Output: geeta

e) print(tuple1[-3][2]) # Output: 22

**Explanation:**

len(tuple1) calculates the length of the tuple.

tuple1[-1][-1]["name"] accesses the "name" key of the last element in the last nested list.

tuple1[7][0]["roll\_no"] accesses the "roll\_no" key of the first element in the last nested list.

tuple1[-3][1] accesses the second element of the third-to-last tuple.

tuple1[-3][2] accesses the third element of the third-to-last tuple.

**11. Define a Python module named constants.py containing constants like pi and the speed of light.**

PI = 3.14159 # Mathematical constant pi

SPEED\_OF\_LIGHT = 299792458 # Speed of light in meters per second (m/s)

**12. Write a Python module named calculator.py containing functions for addition, subtraction, multiplication, and division.**

**13. Implement a Python package structure for a project named ecommerce, containing modules for product management and order processing.**

# ecommerce/\_\_init\_\_.py

"""Package for managing an ecommerce system."""

# product\_management/\_\_init\_\_.py

"""Subpackage for managing products."""

# product\_management/product.py

class Product:

"""Represents a product."""

def \_\_init\_\_(self, product\_id, name, price, description, category):

"""Initializes a new Product object.

Args:

product\_id: The unique product ID.

name: The name of the product.

price: The price of the product.

description: The description of the product.

category: The product category.

"""

self.product\_id = product\_id

self.name = name

self.price = price

self.description = description

self.category = category

# product\_management/product\_category.py

class ProductCategory:

"""Represents a product category."""

def \_\_init\_\_(self, category\_id, name):

"""Initializes a new ProductCategory object.

Args:

category\_id: The unique category ID.

name: The name of the product category.

"""

self.category\_id = category\_id

self.name = name

# order\_processing/\_\_init\_\_.py

"""Subpackage for processing orders."""

# order\_processing/order.py

class Order:

"""Represents an order."""

def \_\_init\_\_(self, order\_id, customer\_id, order\_items, status):

"""Initializes a new Order object.

Args:

order\_id: The unique order ID.

customer\_id: The ID of the customer placing the order.

order\_items: A list of OrderItem objects.

status: The current status of the order (e.g., "pending", "processing", "shipped", "delivered").

"""

self.order\_id = order\_id

self.customer\_id = customer\_id

self.order\_items = order\_items

self.status = status

# order\_processing/order\_item.py

class OrderItem:

"""Represents an item in an order."""

def \_\_init\_\_(self, product, quantity):

"""Initializes a new OrderItem object.

Args:

product: The Product object associated with the order item.

quantity: The quantity of the product ordered.

"""

self.product = product

self.quantity = quantity

# utils/\_\_init\_\_.py

"""Subpackage for utility functions."""

# utils/database.py

import sqlite3

def connect\_to\_database(database\_name):

"""Connects to a SQLite database.

Args:

database\_name: The name of the database file.

Returns:

A connection object.

"""

conn = sqlite3.connect(database\_name)

return conn

def execute\_query(conn, query, params=None):

"""Executes a SQL query.

Args:

conn: The database connection object.

query: The SQL query.

params: A tuple of parameters for the query.

"""

cursor = conn.cursor()

cursor.execute(query, params)

conn.commit()

**14. Implement a Python module named string\_utils.py containing functions for string manipulation, such asreversing and capitalizing strings.**

def reverse\_string(text):

"""Reverses the order of characters in a string.

Args:

text: The string to reverse.

Returns:

The reversed string.

"""

return text[::-1]

def capitalize\_words(text):

"""Capitalizes the first letter of each word in a string.

Args:

text: The string to capitalize.

Returns:

The string with capitalized words.

"""

return text.title()

def remove\_whitespace(text):

"""Removes all whitespace characters from a string.

Args:

text: The string to remove whitespace from.

Returns:

The string with whitespace removed.

"""

return text.replace(" ", "")

def replace\_substring(text, old\_substring, new\_substring):

"""Replaces all occurrences of a substring with another substring.

Args:

text: The string to modify.

old\_substring: The substring to replace.

new\_substring: The replacement substring.

Returns:

The string with the substring replaced.

"""

return text.replace(old\_substring, new\_substring)

def count\_occurrences(text, substring):

"""Counts the number of occurrences of a substring in a string.

Args:

text: The string to search.

substring: The substring to count.

Returns:

The number of occurrences of the substring in the string.

"""

return text.count(substring)

def is\_palindrome(text):

"""Checks if a string is a palindrome (reads the same backward as forward).

Args:

text: The string to check.

Returns:

True if the string is a palindrome, False otherwise.

"""

return text.lower() == text.lower()[::-1]

**15. Write a Python module named file\_operations.py with functions for reading, writing, and appending data to a file.**

def read\_file(filename):

"""Reads the contents of a file and returns them as a string.

Args:

filename: The path to the file to read.

Returns:

The contents of the file as a string, or None if an error occurs.

"""

try:

with open(filename, 'r') as file:

return file.read()

except FileNotFoundError:

print(f"Error: File '{filename}' not found.")

return None

def write\_file(filename, content):

"""Writes content to a file, overwriting any existing content.

Args:

filename: The path to the file to write to.

content: The data to write to the file.

"""

try:

with open(filename, 'w') as file:

file.write(content)

print(f"Successfully wrote content to '{filename}'.")

except Exception as e:

print(f"Error writing to file '{filename}': {e}")

def append\_to\_file(filename, content):

"""Appends content to a file, adding it to the end of the existing content.

Args:

filename: The path to the file to append to.

content: The data to append to the file.

"""

try:

with open(filename, 'a') as file:

file.write(content)

print(f"Successfully appended content to '{filename}'.")

except Exception as e:

print(f"Error appending to file '{filename}': {e}")

import file\_operations

**16. Write a Python program to create a text file named "employees.txt" and write the details of employees,including their name, age, and salary, into the file.**

# Create a list of employees

employees = [

("Alice", 30, 50000),

("Bob", 25, 45000),

("Charlie", 35, 60000)

]

# Write employee details to the file

file\_content = "\n".join(f"{name},{age},{salary}" for name, age, salary in employees)

file\_operations.write\_file("employees.txt", file\_content)

**17. Develop a Python script that opens an existing text file named "inventory.txt" in read mode and displaysthe contents of the file line by line.**

import file\_operations

# Read the contents of the file

contents = file\_operations.read\_file("inventory.txt")

# Display the contents line by line

if contents:

lines = contents.splitlines()

for line in lines:

print(line)

**18. Create a Python script that reads a text file named "expenses.txt" and calculates the total amount spent on various expenses listed in the file.**

import file\_operations

def calculate\_total\_expenses(filename):

"""Calculates the total amount spent on various expenses listed in a file.

Args:

filename: The path to the file containing the expense data.

Returns:

The total amount spent.

"""

total\_expenses = 0

contents = file\_operations.read\_file(filename)

if contents:

lines = contents.splitlines()

for line in lines:

try:

amount = float(line)

total\_expenses += amount

except ValueError:

print(f"Error: Invalid expense amount: {line}")

return total\_expenses

# Calculate the total expenses

total\_expenses = calculate\_total\_expenses("expenses.txt")

# Print the result

print("Total expenses:", total\_expenses)

**19. Create a Python program that reads a text file named "paragraph.txt" and counts the occurrences of each word in the paragraph, displaying the results in alphabetical order.**

import file\_operations

def count\_word\_occurrences(filename):

"""Counts the occurrences of each word in a text file and displays the results in alphabetical order.

Args:

filename: The path to the text file.

"""

contents = file\_operations.read\_file(filename)

if contents:

words = contents.split()

word\_counts = {}

for word in words:

word\_counts[word] = word\_counts.get(word, 0) + 1

sorted\_word\_counts = sorted(word\_counts.items(), key=lambda item: item[0])

for word, count in sorted\_word\_counts:

print(f"{word}: {count}")

# Count word occurrences in the "paragraph.txt" file

count\_word\_occurrences("paragraph.txt")

**20. What do you mean by Measure of Central Tendency and Measures of Dispersion .How it can be calculated.**

## **Measures of Central Tendency and Dispersion**

### **Measures of Central Tendency**

Measures of central tendency are statistical metrics that represent the middle or typical value in a dataset. They provide a single value that summarizes the distribution of data.

**Common measures of central tendency:**

* **Mean:** The average value of a dataset. It's calculated by summing all the values and dividing by the total number of values.
* **Median:** The middle value in a dataset when the values are arranged in ascending or descending order. If the dataset has an even number of values, the median is the average of the two middle values.
* [1. www.studocu.com](https://www.studocu.com/in/document/university-of-delhi/bsc-h-biomedical-sciences/unit-i-descriptive-statistics/61403981)
* [www.studocu.com](https://www.studocu.com/in/document/university-of-delhi/bsc-h-biomedical-sciences/unit-i-descriptive-statistics/61403981)
* **Mode:** The most frequent value in a dataset. A dataset can have one mode (unimodal), multiple modes (multimodal), or no mode.

### **Measures of Dispersion**

Measures of dispersion, also known as measures of variability, quantify how spread out the data is. They indicate how much the values deviate from the central tendency.

**Common measures of dispersion:**

* **Range:** The difference between the largest and smallest values in a dataset.
* **Variance:** The average squared deviation from the mean. It measures how much the data points vary from the mean.
* **Standard Deviation:** The square root of the variance. It provides a measure of dispersion in the same units as the original data.
* **Interquartile Range (IQR):** The difference between the third quartile (Q3) and the first quartile (Q1). It measures the spread of the middle 50% of the data.

**Calculation:**

Here's a Python example using the statistics module to calculate these measures for a given dataset:

import statistics

data = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Mean

mean = statistics.mean(data)

print("Mean:", mean)

# Median

median = statistics.median(data)

print("Median:", median)

# Mode

mode = statistics.mode(data)

print("Mode:", mode)

# Range

data\_range = max(data) - min(data)

print("Range:", data\_range)

# Variance

variance = statistics.variance(data)

print("Variance:", variance)

# Standard Deviation

stdev = statistics.stdev(data)

print("Standard Deviation:", stdev)

# Interquartile Range (IQR)

q1, q3 = statistics.quantiles(data, [0.25, 0.75])

iqr = q3 - q1

print("Interquartile Range (IQR):", iqr)

**21. What do you mean by skewness.Explain its types.Use graph to show.**

## **Skewness: A Measure of Asymmetry**

**Skewness is a statistical measure that quantifies the asymmetry or lack of symmetry in a probability distribution.**

**It indicates whether the tail on one side of the distribution is longer or heavier than the other.**

**1. Skewness | Definition, Examples & Formula -**

**2. Right Skewed vs. Left Skewed Distribution**

### **Types of Skewness**

1. **Positive Skewness (Right-Skewed):**
   * **The tail on the right side of the distribution is longer.**
   * **The mean is greater than the median.**
   * **Example: Income distribution in many countries, where there are a few very high earners (long tail on the right) and many lower earners.**
2. **Negative Skewness (Left-Skewed):**
   * **The tail on the left side of the distribution is longer.**
   * **1. Right Skewed vs. Left Skewed Distribution**
   * **The mean is less than the median.**
   * **Example: The distribution of life expectancy, where there are a few individuals who live exceptionally long (long tail on the left).**
3. **Zero Skewness:**
   * **The distribution is symmetric.**
   * **The mean, median, and mode are equal.**
   * **Example: A normal distribution (bell curve).**

### **Graphical Representation**

### **Image of positively skewed distribution, a negatively skewed distribution, and a symmetric distribution**

**22. Explain PROBABILITY MASS FUNCTION (PMF) and PROBABILITY DENSITY FUNCTION (PDF). and what is the difference between them?**

### **Probability Mass Function (PMF)**

A Probability Mass Function (PMF) is used to describe the probability distribution of a discrete random variable. It gives the probability that a discrete random variable is exactly equal to some value.

* **Discrete random variables:** These variables can take on only a countable number of values, such as integers or specific outcomes from a dice roll.
* **PMF formula:** P(X = x) = f(x), where X is the random variable, x is a specific value, and f(x) is the PMF.

**Example:** Consider flipping a fair coin. The possible outcomes are "heads" and "tails." The PMF for this experiment would be:

* P(X = "heads") = 0.5
* P(X = "tails") = 0.5

### **Probability Density Function (PDF)**

A Probability Density Function (PDF) is used to describe the probability distribution of a continuous random variable. It represents the probability that a continuous random variable falls within a particular range of values.

* **Continuous random variables:** These variables can take on any value within a specified range, such as height, weight, or time.
* **PDF property:** The area under the PDF curve over a given interval equals the probability that the random variable falls within that interval.

**Example:** Consider the height of adult males. The PDF for this distribution would be a bell-shaped curve (normal distribution). To find the probability of a male being between 5'9" and 6'0" tall, you would calculate the area under the PDF curve between those heights.

### **Key Difference**

The main difference between PMF and PDF lies in the nature of the random variables they describe:

* **PMF:** For discrete random variables (e.g., coin flips, dice rolls)
* **PDF:** For continuous random variables (e.g., height, weight, time)

In essence, PMF gives the exact probability of a specific value, while PDF gives the probability of a value falling within a range.

**23. What is correlation. Explain its type in details.what are the methods of determining correlation**

**Correlation** is a statistical measure that quantifies the relationship between two variables. It indicates the extent to which changes in one variable are associated with changes in the other. Correlation can range from -1 to 1:

* **-1:** Perfect negative correlation: The variables move in opposite directions.
* **0:** No correlation: There is no relationship between the variables.
* **1:** Perfect positive correlation: The variables move in the same direction.

### **Types of Correlation**

1. **Linear Correlation:**
   * The relationship between the variables can be represented by a straight line.
   * Measured by Pearson's correlation coefficient (r).
2. **Non-linear Correlation:**
   * The relationship between the variables cannot be represented by a straight line.
   * Measured by Spearman's rank correlation coefficient or Kendall's tau.

### **Methods of Determining Correlation**

1. **Pearson's Correlation Coefficient (r):**
   * Measures the strength and direction of linear relationships between two quantitative variables.
   * Ranges from -1 to 1.
   * A value close to 1 indicates a strong positive correlation, a value close to -1 indicates a strong negative correlation, and a value close to 0 indicates no correlation.
2. **Spearman's Rank Correlation Coefficient (ρ):**
   * Measures the strength and direction of monotonic relationships between two variables, regardless of whether the relationship is linear.
   * Ranks the data for each variable and then calculates the correlation between the ranks.
   * Ranges from -1 to 1.
3. **Kendall's Tau:**
   * Another measure of rank correlation that is less sensitive to outliers than Spearman's rank correlation coefficient.
   * Also ranges from -1 to 1.

### **Visualizing Correlation**

Scatter plots can be used to visually inspect the relationship between two variables. If the points on the scatter plot cluster around a straight line, it suggests a linear correlation. If the points are scattered randomly, it suggests no correlation.

**Note:** Correlation does not imply causation. A strong correlation between two variables does not necessarily mean that one variable causes changes in the other. There

could be other factors influencing both variables

**24. Calculate coefficient of correlation between the marks obtained by 10 students in Accountancy and statistics:**

import statistics

# Data for Accountancy and Statistics marks

accountancy\_marks = [45, 70, 65, 30, 90, 40, 50, 75, 85, 60]

statistics\_marks = [35, 90, 70, 40, 95, 40, 60, 80, 80, 50]

# Calculate the mean of Accountancy and Statistics marks

mean\_accountancy = statistics.mean(accountancy\_marks)

mean\_statistics = statistics.mean(statistics\_marks)

# Calculate the numerator of the correlation coefficient formula

numerator = sum([(accountancy\_mark - mean\_accountancy) \* (statistics\_mark - mean\_statistics) for accountancy\_mark, statistics\_mark in zip(accountancy\_marks, statistics\_marks)])

# Calculate the denominator of the correlation coefficient formula

denominator = (sum([(accountancy\_mark - mean\_accountancy) \*\* 2 for accountancy\_mark in accountancy\_marks]) \* sum([(statistics\_mark - mean\_statistics) \*\* 2 for statistics\_mark in statistics\_marks])) \*\* 0.5

# Calculate the correlation coefficient

correlation\_coefficient = numerator / denominator

print("Correlation coefficient:", correlation\_coefficient)

This code calculates the correlation coefficient between the marks obtained by 10 students in Accountancy and Statistics using Karl Pearson's Coefficient of Correlation method. It first calculates the mean of both sets of marks. Then, it calculates the numerator and denominator of the correlation coefficient formula and finally calculates the correlation coefficient itself.

**25. Discuss the 4 differences between correlation and regression.**

Correlation and regression are both statistical techniques used to analyze relationships between variables, but they serve different purposes and have distinct characteristics.

### 1. Direction and Strength:

* Correlation: Measures the strength and direction of the relationship between two variables. It ranges from -1 (perfect negative correlation) to 1 (perfect positive correlation).
* Regression: Predicts the value of one variable (dependent variable) based on the value of another variable (independent variable). It doesn't necessarily measure the strength of the relationship.

### 2. Causation:

* Correlation: Does not imply causation. A strong correlation between two variables does not necessarily mean that one causes the other.
* Regression: Can be used to establish a causal relationship under certain conditions (e.g., controlled experiments, time-series analysis). However, correlation alone is not sufficient to prove causation.

### 3. Purpose:

* Correlation: Primarily used to measure the strength and direction of the relationship between variables.
* Regression: Used to predict the value of one variable based on the values of other variables.

### 4. Output:

* Correlation: Produces a single value (correlation coefficient) that measures the strength and direction of the relationship.
* Regression: Produces an equation (regression line or model) that can be used to predict the value of the dependent variable for given values of the independent variables.

In summary:

* Correlation measures the relationship between variables.
* Regression predicts the value of one variable based on another.
* Correlation does not imply causation, while regression can establish causation under certain conditions.
* Correlation produces a single value, while regression produces an equation.

**26. Find the most likely price at Delhi corresponding to the price of Rs. 70 at Agra from the following data: Coefficient of correlation between the prices of the two places +0.8.**

To find the most likely price at Delhi corresponding to the price of Rs. 70 at Agra, we can use the concept of linear regression based on the given correlation coefficient.

Here's the formula for predicting the value of one variable (y) based on the value of another variable (x) using the correlation coefficient (r) and the standard deviations (σx and σy) of the two variables:

y\_predicted = y\_mean + r \* (σy / σx) \* (x - x\_mean)

where:

* y\_predicted is the predicted value of the dependent variable (price in Delhi)
* y\_mean is the mean of the dependent variable (prices in Delhi)
* x is the given value of the independent variable (price in Agra)
* x\_mean is the mean of the independent variable (prices in Agra)
* σy is the standard deviation of the dependent variable (prices in Delhi)
* σx is the standard deviation of the independent variable (prices in Agra)

Assumptions:

* We assume that the relationship between the prices in Delhi and Agra is linear.
* We need the standard deviations of prices in both cities to calculate the predicted price accurately.

Since the data for prices in Delhi is not provided, we'll make some assumptions:

* Assume the mean price in Delhi is also Rs. 70.
* Assume the standard deviation of prices in Delhi is the same as the standard deviation in Agra.

Calculating the predicted price:

import numpy as np

# Given data

price\_agra = 70

correlation\_coefficient = 0.8

# Assume mean and standard deviation in Delhi are the same as in Agra

mean\_agra = np.mean([70, 60, 80, 90, 50])

std\_dev\_agra = np.std([70, 60, 80, 90, 50])

mean\_delhi = mean\_agra

std\_dev\_delhi = std\_dev\_agra

# Calculate the predicted price in Delhi

predicted\_price\_delhi = mean\_delhi + correlation\_coefficient \* std\_dev\_delhi \* ((price\_agra - mean\_agra) / std\_dev\_agra)

print("Predicted price in Delhi:", predicted\_price\_delhi)

Output:

Predicted price in Delhi: 70.0

**27. In a partially destroyed laboratory record of an analysis of correlation data, the following results only are**

**legible: Variance of x = 9, Regression equations are: (i) 8x−10y = −66; (ii) 40x − 18y = 214. What are (a) the**

**mean values of x and y, (b) the coefficient of correlation between x and y, (c) the σ of y.**

To analyze the given data, we can use the following equations:

Equation 1: 8x - 10y = -66 Equation 2: 40x - 18y = 214

a) Finding the mean values of x and y:

We can solve the given equations simultaneously to find the values of x and y.

Multiplying equation 1 by 5 and subtracting equation 2 from it, we get:

## 40x - 50y = -330 40x - 18y = 214

-32y = -544

Solving for y, we get:

y = 17

Substituting y = 17 into equation 1, we get:

8x - 10(17) = -66 8x - 170 = -66 8x = 104 x = 13

Therefore, the mean values of x and y are:

* x = 13
* y = 17

b) Finding the coefficient of correlation between x and y:

We can use the following formula to find the coefficient of correlation (r):

r = (n \* Σxy - Σx \* Σy) / sqrt((n \* Σx^2 - (Σx)^2) \* (n \* Σy^2 - (Σy)^2))

where:

* n is the number of data points
* Σxy is the sum of the product of x and y
* Σx is the sum of x
* Σy is the sum of y
* Σx^2 is the sum of the squares of x
* Σy^2 is the sum of the squares of y

Unfortunately, we don't have the complete data to calculate the exact values of Σxy, Σx^2, and Σy^2. However, we can use the given regression equations to estimate the correlation coefficient.

We know that the slope of the regression line is equal to the correlation coefficient multiplied by the ratio of the standard deviations of x and y:

b1 = r \* (σy / σx)

From equation 1, we can see that the slope is 8/10 = 0.8.

We also know that the variance of x (σx^2) is 9. Therefore, the standard deviation of x (σx) is √9 = 3.

Substituting these values into the equation above, we get:

0.8 = r \* (σy / 3)

Solving for σy, we get:

σy = 0.8 \* 3 / r

Since we don't have the exact value of r, we cannot calculate the exact value of σy. However, we can say that σy is proportional to 0.8/r.

c) Finding σy:

As discussed above, we cannot determine the exact value of σy without knowing the exact value of r. However, we can say that σy is proportional to 0.8/r.

**28. What is Normal Distribution? What are the four Assumptions of Normal Distribution? Explain in detail.**

## Normal Distribution (Gaussian Distribution)

Normal distribution, also known as Gaussian distribution or bell curve, is a probability distribution that is symmetrical around the mean and follows a bell-shaped curve. It's one of the most common probability distributions in statistics and is used to model many natural phenomena, such as height, weight, and IQ scores.

Characteristics of Normal Distribution:

* Symmetry: The distribution is symmetrical around the mean.
* Bell-shaped curve: The shape resembles a bell curve.
* Mean, median, and mode are equal: In a normal distribution, these three measures of central tendency coincide.
* Standard deviation: Controls the spread of the distribution. A larger standard deviation results in a wider curve, while a smaller standard deviation results in a narrower curve.

Four Assumptions of Normal Distribution

1. Normality: The data points are normally distributed. This means that the distribution of the data follows a bell-shaped curve.
2. Independence: The data points are independent of each other. This means that the value of one data point does not affect the value of another data point.
3. Homoscedasticity: The variance of the data points is constant across all values of the independent variable. This means that the spread of the data is the same at all points.
4. Linearity: There is a linear relationship between the dependent and independent variables. This means that the relationship can be modeled by a straight line.

Applications of Normal Distribution:

* Statistical inference: Many statistical tests and procedures assume that the data follows a normal distribution.
* Quality control: Normal distribution is used to monitor processes and identify deviations from expected values.
* Finance: Stock prices and returns are often assumed to follow a normal distribution.
* Social sciences: Many social phenomena, such as IQ scores and height, are approximately normally distributed.

Note: While many natural phenomena follow a normal distribution, it's important to check the assumptions of normality before applying statistical methods that rely on it. Techniques like histograms, Q-Q plots, and statistical tests can be used to assess normality.

**29.Write all the characteristics or Properties of the Normal Distribution Curve.**

## Characteristics of the Normal Distribution Curve

The normal distribution curve, also known as the Gaussian distribution or bell curve, has several key characteristics:

1. Symmetry: The curve is symmetrical around the mean, meaning the left and right sides are mirror images of each other.
2. Bell-shaped: The curve has a bell-like shape, with the highest point at the mean.
3. Mean, Median, and Mode are Equal: In a normal distribution, these three measures of central tendency coincide.
4. Standard Deviation: The standard deviation determines the shape of the curve. A larger standard deviation results in a wider curve, while a smaller standard deviation results in a narrower curve.
5. Empirical Rule (68-95-99.7 Rule):
   * Approximately 68% of the data falls within one standard deviation of the mean.
   * Approximately 95% of the data falls within two standard deviations of the mean.
   * Approximately 99.7% of the data falls within three standard deviations of the mean.
6. Area Under the Curve: The total area under the normal distribution curve is always equal to 1. This represents the probability of a random variable falling within a certain range.
7. Standardization:
8. Any normal distribution can be standardized using the z-score formula, which transforms the data into a standard normal distribution with a mean of 0 and a standard deviation of 1. This allows for easy comparison of data points from different normal distributions.
9. Central Limit Theorem:

The central limit theorem states that the distribution of sample means from a large sample size approaches a normal distribution, regardless of the underlying distribution of the population. This is a fundamental theorem in statistics and is used in many statistical tests and procedures.

**30.Which of the following options are correct about Normal Distribution Curve.**

**(a) Within a range 0.6745 of σ on both sides the middle 50% of the observations occur i,e. mean ±0.6745σ**

**covers 50% area 25% on each side.**

**(b) Mean ±1S.D. (i,e.μ ± 1σ) covers 68.268% area, 34.134 % area lies on either side of the mean.**

**(c) Mean ±2S.D. (i,e. μ ± 2σ) covers 95.45% area, 47.725% area lies on either side of the mean.**

**(d) Mean ±3 S.D. (i,e. μ ±3σ) covers 99.73% area, 49.856% area lies on the either side of the mean.**

**(e) Only 0.27% area is outside the range μ ±3σ.**

The correct options are:

- (b) Mean ±1S.D. (i,e.μ ± 1σ) covers 68.268% area, 34.134 % area lies on either side of the mean.

- (c) Mean ±2S.D. (i,e. μ ± 2σ) covers 95.45% area, 47.725% area lies on either side of the mean.

- (d) Mean ±3 S.D. (i,e. μ ±3σ) covers 99.73% area, 49.856% area lies on the either side of the mean.

- (e) Only 0.27% area is outside the range μ ±3σ.

These statements accurately reflect the properties of the normal distribution curve, as explained in the previous response.

**31. The mean of a distribution is 60 with a standard deviation of 10. Assuming that the distribution is normal,**

**what percentage of items be (i) between 60 and 72, (ii) between 50 and 60, (iii) beyond 72 and (iv) between**

**70 and 80?**

## 

Given:

* Mean (μ) = 60
* Standard Deviation (σ) = 10

Using the Empirical Rule (68-95-99.7 Rule):

* Approximately 68% of the data falls within one standard deviation of the mean.
* Approximately 95% of the data falls within two standard deviations of the mean.
* Approximately 99.7% of the data falls within three standard deviations of the mean.

Calculations:

(i) Between 60 and 72:

* This range is within one standard deviation of the mean (60 ± 10).
* Therefore, approximately 68% of the items fall within this range.

(ii) Between 50 and 60:

* This range is also within one standard deviation of the mean (60 - 10 to 60).
* Since the distribution is symmetrical, half of the 68% (34%) falls between the mean and one standard deviation below the mean.

(iii) Beyond 72:

* This range is more than one standard deviation above the mean.
* Since 68% of the data falls within one standard deviation, the remaining 32% falls outside.
* Due to symmetry, 16% falls above 72.

(iv) Between 70 and 80:

* This range is between one and two standard deviations above the mean.
* Using the empirical rule, we know that 95% of the data falls within two standard deviations.
* We already know that 68% falls within one standard deviation.
* So, the remaining 95% - 68% = 27% falls between one and two standard deviations.
* Since the distribution is symmetrical, half of this 27% (13.5%) falls between 70 and 80.

Summary:

* (i) Between 60 and 72: Approximately 68%
* (ii) Between 50 and 60: Approximately 34%
* (iii) Beyond 72: Approximately 16%
* (iv) Between 70 and 80: Approximately 13.5%

**32. 15000 students sat for an examination. The mean marks was 49 and the distribution of marks had a**

**standard deviation of 6. Assuming that the marks were normally distributed what proportion of students**

**scored (a) more than 55 marks, (b) more than 70 marks**

**Calculating z-scores:**

import scipy.stats as stats

# Given data

total\_students = 15000

mean\_marks = 49

standard\_deviation = 6

# Calculate z-scores for the given marks

z\_score\_55 = (55 - mean\_marks) / standard\_deviation

z\_score\_70 = (70 - mean\_marks) / standard\_deviation

**Calculating probabilities:**

# Calculate the probabilities using the standard normal distribution

probability\_more\_than\_55 = 1 - stats.norm.cdf(z\_score\_55)

probability\_more\_than\_70 = 1 - stats.norm.cdf(z\_score\_70)

**Calculating the number of students:**

Python

# Calculate the number of students

number\_of\_students\_more\_than\_55 = probability\_more\_than\_55 \* total\_students

number\_of\_students\_more\_than\_70 = probability\_more\_than\_70 \* total\_students

**Printing the results:**

Python

print("Proportion of students who scored more than 55 marks:", probability\_more\_than\_55)

print("Number of students who scored more than 55 marks:", number\_of\_students\_more\_than\_55)

print("Proportion of students who scored more than 70 marks:", probability\_more\_than\_70)

print("Number of students who scored more than 70 marks:", number\_of\_students\_more\_than\_70)

**Output:**

Proportion of students who scored more than 55 marks: 0.15865525393145707

Number of students who scored more than 55 marks: 2379.8288089718562

Proportion of students who scored more than 70 marks: 0.0002326290790355401

Number of students who scored more than 70 marks: 3.4894361855331013

**Explanation:**

* The z-score is calculated by subtracting the mean from the given mark and then dividing by the standard deviation.
* The stats.norm.cdf function from the scipy.stats module is used to calculate the cumulative probability of a value being less than or equal to the given z-score.
* To find the probability of scoring *more than* a certain mark, we subtract the cumulative probability from 1.
* Finally, the number of students is calculated by multiplying the proportion by the total number of students.

Therefore, approximately 15.87% of the students scored more than 55 marks, and approximately 0.0002% of the students scored more than 70 marks.

**33. If the height of 500 students are normally distributed with mean 65 inch and standard deviation 5 inch.**

**How many students have height : a) greater than 70 inch. b) between 60 and 70 inch.**

Calculating z-scores:

import scipy.stats as stats

# Given data

total\_students = 500

mean\_height = 65

standard\_deviation = 5

# Calculate z-scores for the given heights

z\_score\_70 = (70 - mean\_height) / standard\_deviation

z\_score\_60 = (60 - mean\_height) / standard\_deviation

Calculating probabilities:

# Calculate the probabilities using the standard normal distribution

probability\_greater\_than\_70 = 1 - stats.norm.cdf(z\_score\_70)

probability\_between\_60\_and\_70 = stats.norm.cdf(z\_score\_70) - stats.norm.cdf(z\_score\_60)

Calculating the number of students:

# Calculate the number of students

number\_of\_students\_greater\_than\_70 = probability\_greater\_than\_70 \* total\_students

number\_of\_students\_between\_60\_and\_70 = probability\_between\_60\_and\_70 \* total\_students

# Print the results

print("Proportion of students taller than 70 inches:", probability\_greater\_than\_70)

print("Number of students taller than 70 inches:", number\_of\_students\_greater\_than\_70)

print("Proportion of students between 60 and 70 inches:", probability\_between\_60\_and\_70)

print("Number of students between 60 and 70 inches:", number\_of\_students\_between\_60\_and\_70)

Output:

Proportion of students taller than 70 inches: 0.15865525393145707

Number of students taller than 70 inches: 79.32762696572854

Proportion of students between 60 and 70 inches: 0.6826894921370859

Number of students between 60 and 70 inches: 341.3447460685429

Explanation:

* The z-score is calculated by subtracting the mean from the given height and then dividing by the standard deviation.
* The stats.norm.cdf function from the scipy.stats module is used to calculate the cumulative probability of a value being less than or equal to the given z-score.
* To find the probability of being taller than 70 inches, we subtract the cumulative probability of being less than or equal to 70 inches from 1.
* To find the probability of being between 60 and 70 inches, we subtract the cumulative probability of being less than or equal to 60 inches from the cumulative probability of being less than or equal to 70 inches.
* Finally, the number of students is calculated by multiplying the proportion by the total number of students.

Therefore, approximately 15.87% of the students are taller than 70 inches, and approximately 68.27% of the students have heights between 60 and 70 inches.

**34. What is the statistical hypothesis? Explain the errors in hypothesis testing.b)Explain the Sample. What**

**are Large Samples & Small Samples?**

## Statistical Hypothesis

A statistical hypothesis is a statement about a population parameter. It is a claim or assumption that we want to test using statistical methods. There are two types of hypotheses:

* Null Hypothesis (H0): This is the hypothesis of no effect or no difference. It is the hypothesis that we aim to test.
* Alternative Hypothesis (H1 or Ha): This is the hypothesis that contradicts the null hypothesis. It is the hypothesis we accept if we reject the null hypothesis.

For example:

* Null Hypothesis: The average height of students in a class is 170 cm.
* Alternative Hypothesis: The average height of students in a class is not 170 cm.

## Errors in Hypothesis Testing

When conducting a hypothesis test, there is always a risk of making an incorrect decision. The two types of errors that can occur are:

1. Type I Error: Rejecting the null hypothesis when it is actually true. This is also known as a false positive. The probability of a Type I error is denoted by α.
2. Type II Error: Failing to reject the null hypothesis when it is false. This is also known as a false negative. The probability of a Type II error is denoted by β.

The goal of hypothesis testing is to minimize the probability of both Type I and Type II errors. However, there is often a trade-off between the two. Increasing the significance level (α) can reduce the probability of a Type II error but increases the probability of a Type I error.

## Sample

A sample is a subset of a population. It is used to make inferences about the population. The size of the sample can affect the accuracy of the inferences.

### Large Samples and Small Samples

* Large Sample: A large sample is one that contains a significant number of observations. Larger samples generally provide more accurate estimates of population parameters.
* Small Sample: A small sample contains a relatively small number of observations. Small samples may have more variability and may not provide as accurate estimates as large samples.

The appropriate sample size depends on factors such as the desired level of accuracy, the variability of the population, and the statistical method being used. Generally, larger samples are preferred for more accurate inference

**35.A random sample of size 25 from a population gives the sample standard derivation to be 9.0. Test the**

**hypothesis that the population standard derivation is 10.5.**

**Hint(Use chi-square distribution).**

To test the hypothesis that the population standard deviation is 10.5, we can use a chi-square test for variance.

**Here are the steps involved:**

1. **State the null and alternative hypotheses:**
   * Null hypothesis (H0): σ² = 10.5²
   * Alternative hypothesis (Ha): σ² ≠ 10.5²

**Calculate the test statistic:**  
import scipy.stats as stats

# Given data

sample\_size = 25

sample\_standard\_deviation = 9.0

population\_standard\_deviation = 10.5

# Calculate the test statistic (chi-square)

test\_statistic = (sample\_size - 1) \* (sample\_standard\_deviation \*\* 2) / (population\_standard\_deviation \*\* 2)

**Determine the degrees of freedom:**  
degrees\_of\_freedom = sample\_size - 1

1. **Set the significance level (α):** Choose a significance level (α) based on the desired level of confidence. A common choice is α = 0.05.

**Find the critical value from the chi-square distribution:**  
critical\_value = stats.chi2.ppf(1 - alpha, degrees\_of\_freedom)

1. **Compare the test statistic with the critical value:** If the test statistic is greater than the critical value, reject the null hypothesis. Otherwise, fail to reject the null hypothesis.

**Python code:**

import scipy.stats as stats

# Given data

sample\_size = 25

sample\_standard\_deviation = 9.0

population\_standard\_deviation = 10.5

alpha = 0.05

# Calculate the test statistic

test\_statistic = (sample\_size - 1) \* (sample\_standard\_deviation \*\* 2) / (population\_standard\_deviation \*\* 2)

# Calculate degrees of freedom

degrees\_of\_freedom = sample\_size - 1

# Find the critical value

critical\_value = stats.chi2.ppf(1 - alpha, degrees\_of\_freedom)

# Compare the test statistic with the critical value

if test\_statistic > critical\_value:

print("Reject the null hypothesis: Population standard deviation is not 10.5")

else:

print("Fail to reject the null hypothesis: Population standard deviation could be 10.5")

**Output:**

Fail to reject the null hypothesis: Population standard deviation could be 10.5

**Interpretation:**

Based on the chi-square test, there is not enough evidence to reject the null hypothesis at a significance level of 0.05. Therefore, we cannot conclude that the population standard deviation is significantly different from 10.5. It is possible that the sample standard deviation of 9.0 is within a reasonable range of variation for a population with a standard deviation of 10.5.

**37.100 students of a PW IOI obtained the following grades in Data Science paper :**

**Grade :[A, B, C, D, E]**

**Total Frequency :[15, 17, 30, 22, 16, 100]**

**Using the χ 2 test , examine the hypothesis that the distribution of grades is uniform.**

To test the hypothesis that the distribution of grades is uniform, we can use the chi-square goodness-of-fit test.

**Here are the steps involved:**

1. **State the null and alternative hypotheses:**
   * Null hypothesis (H0): The distribution of grades is uniform.
   * Alternative hypothesis (Ha): The distribution of grades is not uniform.

**Calculate the expected frequencies:** Since we are assuming a uniform distribution, the expected frequency for each grade is the total number of students divided by the number of grades.  
  
  
expected\_frequency = 100 / 5 # Assuming uniform distribution

**Calculate the chi-square test statistic:**  
import scipy.stats as stats

# Given data

observed\_frequencies = [15, 17, 30, 22, 16]

# Calculate the chi-square test statistic

test\_statistic = sum([(observed\_frequency - expected\_frequency) \*\* 2 / expected\_frequency for observed\_frequency in observed\_frequencies])

**Determine the degrees of freedom:**  
degrees\_of\_freedom = len(observed\_frequencies) - 1

1. **Set the significance level (α):** Choose a significance level (α) based on the desired level of confidence. A common choice is α = 0.05.

**Find the critical value from the chi-square distribution:**  
critical\_value = stats.chi2.ppf(1 - alpha, degrees\_of\_freedom)

1. **Compare the test statistic with the critical value:** If the test statistic is greater than the critical value, reject the null hypothesis. Otherwise, fail to reject the null hypothesis.

**Python code:**

import scipy.stats as stats

# Given data

observed\_frequencies = [15, 17, 30, 22, 16]

expected\_frequency = 100 / 5 # Assuming uniform distribution

# Calculate the chi-square test statistic

test\_statistic = sum([(observed\_frequency - expected\_frequency) \*\* 2 / expected\_frequency for observed\_frequency in observed\_frequencies])

# Calculate degrees of freedom

degrees\_of\_freedom = len(observed\_frequencies) - 1

# Set the significance level (alpha)

alpha = 0.05

# Find the critical value from the chi-square distribution

critical\_value = stats.chi2.ppf(1 - alpha, degrees\_of\_freedom)

# Compare the test statistic with the critical value

if test\_statistic > critical\_value:

print("Reject the null hypothesis: The distribution of grades is not uniform.")

else:

print("Fail to reject the null hypothesis: The distribution of grades could be uniform.")

**Output:**

Fail to reject the null hypothesis: The distribution of grades could be uniform.

**Interpretation:**

Based on the chi-square test, there is not enough evidence to reject the null hypothesis at a significance level of 0.05. Therefore, we cannot conclude that the distribution of grades is significantly different from a uniform distribution. It is possible that the observed differences in frequencies are due to chance

**38.Anova Test:**

## 

Understanding the Problem:

We have a dataset with two factors: detergent type and water temperature. We want to determine if either factor or their interaction significantly affects the whiteness readings.

Steps:

1. State the Hypotheses:
   * Null Hypothesis (H0): There is no significant difference in whiteness readings due to detergent type, water temperature, or their interaction.
   * Alternative Hypothesis (H1): At least one of the factors (detergent type, water temperature, or their interaction) has a significant effect on whiteness readings.
2. Perform Two-Way ANOVA:  
   Use statistical software (like R, Python, or Excel) to perform a two-way ANOVA analysis on the given data. This will provide the F-values and p-values for each factor and their interaction.
3. Interpret the Results:
   * If the p-value for a factor is less than the chosen significance level (e.g., 0.05), we reject the null hypothesis for that factor, indicating a significant effect.
   * If the p-value for the interaction is less than the significance level, it suggests that the effect of one factor depends on the level of the other factor.

Note: Without the actual calculations from the software, I cannot provide specific p-values or conclusions. However, the above steps outline the general procedure for conducting a two-way ANOVA to analyze the given data.

Additional Considerations:

* Assumptions: Ensure that the data meets the assumptions of ANOVA, such as normality, homogeneity of variances, and independence.
* Post-hoc tests: If a significant effect is found, you may need to perform post-hoc tests (e.g., Tukey's HSD) to determine which specific groups or levels are significantly different.

By following these steps and using appropriate statistical software, you can analyze the given data and determine if there are significant differences in whiteness readings due to detergent type, water temperature, or their interaction.

## 

### **39. How would you create a basic Flask route that displays "Hello, World!" on the homepage?**

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello, World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

### **40. Explain how to set up a Flask application to handle form submissions using POST requests.**

`` from flask import Flask, request, render\_template

app = Flask(name)

@app.route('/') def index(): return render\_template('index.html')

@app.route('/submit', methods=['POST']) def submit(): name = request.form['name'] message = request.form['message'] return f"You submitted: Name: {name}, Message: {message}"

if name == 'main': app.run()

This code creates a Flask application with two routes:

- `/`: Renders an HTML template `index.html` that contains a form with `name` and `message` fields.

- `/submit`: Handles the POST request from the form and retrieves the submitted data using `request.form`. It then returns a message with the submitted name and message.

**41. Write a Flask route that accepts a parameter in the URL and displays it on the page.**

@app.route('/hello/<name>')

def hello\_name(name):

return f"Hello, {name}!"

This route accepts a parameter name in the URL and displays it in the response. You can access the parameter using the name variable within the function.

### **42. How can you implement user authentication in a Flask application?**

There are several ways to implement user authentication in a Flask application, including:

* Using a built-in authentication system like Flask-Login or Flask-Principal.
* Building your own authentication system using sessions and cookies.
* Integrating with a third-party authentication service like Google, Facebook, or OAuth2.

### **43. Describe the process of connecting a Flask app to a SQLite database using SQLAlchemy.**

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI']

1. g

= 'sqlite:///your\_database.db'

db = SQLAlchemy(app)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

email = db.Column(db.String(120), unique=True, nullable=False)

# Create

the database tables

with app.app\_context():

db.create\_all()

This code connects the Flask application to a SQLite database using SQLAlchemy. It defines a User model with id, username, and email columns. The db.create\_all() method creates the database tables based on the defined models.

### 44. How would you create a RESTful API endpoint in Flask that returns JSON data?

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/api/users', methods=['GET'])

def get\_users():

users = [

{'id': 1, 'name': 'Alice'},

{'id': 2, 'name': 'Bob'}

]

return jsonify(users)

This code creates a GET endpoint /api/users that returns a list of users as JSON data. You can replace the hardcoded list of users with data from a database or other sources.

### **45. Explain how to use Flask-WTF to create and validate forms in a Flask application.**

from flask import Flask, render\_template, request

from flask\_wtf import FlaskForm

from wtforms import StringField, SubmitField

from wtforms.validators import DataRequired

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'your\_secret\_key'

wtf = FlaskForm()

class ContactForm(FlaskForm):

name = StringField('Name', validators=[DataRequired()])

email = StringField('Email', validators=[DataRequired()])

message

= StringField('Message', validators=[DataRequired()])

submit = SubmitField('Submit')

@app.route('/', methods=['GET', 'POST'])

def index():

form = ContactForm()

if form.validate\_on\_submit():

# Handle form submission

name = form.name.data

email = form.email.data

message = form.message.data

# Process the submitted data

return 'Form submitted successfully!'

return render\_template('index.html', form=form)

This code uses Flask-WTF to create a form with name, email, and message fields. The DataRequired validator ensures that these fields are not empty. The form is rendered in the index.html template, and the submitted data is processed when the form is submitted.

### **46. How can you implement file uploads in a Flask application?**

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_)

@app.route('/', methods=['GET', 'POST'])

def upload\_file():

if request.method == 'POST':

file = request.files['file']

if file and file.filename:

file.save('uploads/' + file.filename)

return 'File uploaded successfully!'

return render\_template('upload.html')

This code creates a form for uploading files. When the form is submitted, the uploaded file is saved to the uploads directory.

### **47. Describe the steps to create a Flask blueprint and why you might use one.**

A Flask blueprint is a way to organize your application into smaller, reusable modules. This can be useful for larger applications with multiple features or when you want to share code between different applications.

To create a blueprint, you define a class that inherits from flask.Blueprint. You can then register the blueprint with the main application using the app.register\_blueprint method.

### **48. How would you deploy a Flask application to a production server using Gunicorn and Nginx?**

Gunicorn is a Python WSGI HTTP server that can be used to deploy Flask applications. Nginx is a high-performance web server that can act as a reverse proxy for your Flask application.

To deploy a Flask application using Gunicorn and Nginx, you can follow these steps:

1. Install Gunicorn and Nginx on your production server.
2. Create a Gunicorn configuration file (e.g., gunicorn.conf) with the following contents:

workers = 4

bind = '0.0.0.0:8000'

1. Create an Nginx configuration file (e.g., nginx.conf) with the following contents:

http {

upstream app {

server unix:/path/to/your/app.sock;

}

server {

listen 80;

location / {

proxy\_pass http://app;

}

}

}

1. Run Gunicorn with the configuration file:

gunicorn --config gunicorn.conf your\_app:app

1. Start Nginx using the nginx command.

### **49. Make a fully functional web application using flask, MongoDB. Signup, Signin page. And after successfully login. Say hello Geeks message at webpage.**

from flask import Flask, render\_template, request, redirect, url\_for

from flask\_mongoengine import MongoEngine

from werkzeug.security import generate\_password\_hash, check\_password\_hash

app = Flask(\_\_name\_\_)

app.config['MONGODB\_DB']

= 'your\_database\_name'

db = MongoEngine()

db.init\_app(app)

class User(db.Document):

username = db.StringField(required=True, unique=True)

email = db.StringField(required=True, unique=True)

password = db.StringField(required=True)

@app.route('/signup', methods=['GET', 'POST'])

def signup():

if request.method == 'POST':

username = request.form['username']

email = request.form['email']

password = request.form['password']

user = User(username=username, email=email, password=generate\_password\_hash(password))

user.save()

return redirect(url\_for('login'))

return render\_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user

= User.objects.get(username=username)

if user and check\_password\_hash(user.password, password):

# Successful login

return redirect(url\_for('hello'))

else:

# Login failed

return "Invalid username or password"

return render\_template('login.html')

@app.route('/hello')

def hello():

return "Hello, Geeks!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

This code creates a Flask application with MongoDB integration for user signup and login. It defines a User model with username, email, and password fields. The

**50.Machine Learning**

### What is the difference between Series & Dataframes?

* Series: A one-dimensional labeled array capable of holding any data type (integers, strings, floats, etc.). Think of it as a single column in a spreadsheet.
* DataFrame: A two-dimensional, size-mutable labeled data structure with columns of potentially different data types. Essentially, a tabular data structure similar to a spreadsheet with rows and columns.

### Creating a MySQL Database and Populating a Table with Pandas

1. Establish MySQL Connection (Replace placeholders with your credentials):

import mysql.connector

mydb = mysql.connector.connect( host="your\_host", user="your\_username", password="your\_password", database="Travel\_Planner" )

mycursor = mydb.cursor()

2. \*\*Create Table (if it doesn't exist):\*\*

```sql

CREATE TABLE IF NOT EXISTS bookings (

user\_id INT,

flight\_id INT,

hotel\_id INT,

activity\_id INT,

booking\_date DATE

);

Insert Dummy Data:  
  
 SQL  
INSERT INTO bookings (user\_id, flight\_id, hotel\_id, activity\_id, booking\_date)

VALUES (1, 123, 456, 789, '2024-09-01'),

(2, 456, 789, 1011, '2024-08-15'),

(3, 789, 1213, 1314, '2024-07-20');

Read Data Using Pandas:  
  
  
import pandas as pd

query = "SELECT \* FROM bookings"

df = pd.read\_sql(query, mydb)

print(df.to\_string()) # Display DataFrame content

mydb.close() # Close connection

### Difference Between loc and iloc

* loc: Label-based indexing and selection for both rows and columns. Selects rows and/or columns by their labels or boolean conditions.
* iloc: Integer-based indexing and selection for rows and columns. Selects rows and/or columns by their position (zero-based indexing).

### Supervised vs. Unsupervised Learning

* Supervised Learning: Involves training a model on labeled data (data with known outcomes). The model learns to map inputs to desired outputs (e.g., classification, regression).
* Unsupervised Learning: Deals with unlabeled data (data without known outcomes). The model discovers patterns or structures within the data itself.

### Bias-Variance Tradeoff

* Bias: The tendency of a model to underfit the data, meaning it may not capture the underlying relationships fully.
* Variance: The tendency of a model to overfit the data, meaning it captures noise or random errors in the training data, leading to poor performance on unseen data.

There's a trade-off between bias and variance:

* Reducing bias often leads to higher variance (e.g., simpler models may not capture the true relationship between features and targets).
* Reducing variance often leads to higher bias (e.g., too complex models may overfit the training data).

### Precision and Recall

* Precision: Measures the proportion of true positives (correctly identified positive cases) out of all predicted positive cases.
* Recall: Measures the proportion of true positives (correctly identified positive cases) out of all actual positive cases.

Relationship to Accuracy

* Accuracy: Ratio of correctly classified instances (true positives + true negatives) to the total number of instances.

Differences:

* Accuracy doesn't differentiate between true positives and false positives, while precision and recall do.
* Precision focuses on the number of positive predictions that are truly positive.

### Overfitting and Prevention

Overfitting: A model that performs well on the training data but poorly on unseen data. This occurs when the model learns the noise in the training data rather than the underlying patterns.

Prevention:

* Regularization: Adding a penalty term to the loss function to discourage complex models.
* Cross-validation: Splitting the data into training and validation sets to evaluate model performance on unseen data.
* Early stopping: Stopping the training process when performance on the validation set starts to deteriorate.
* Feature engineering: Selecting or creating relevant features that capture the underlying patterns in the data.

### Cross-Validation

Cross-validation is a technique used to evaluate the performance of a machine learning model on unseen data. It involves splitting the dataset into multiple folds, training the model on a subset of the folds, and evaluating its performance on the remaining fold. This process is repeated multiple times to obtain a more accurate estimate of the model's performance.

### Classification vs. Regression

* Classification: The task of predicting categorical outcomes (e.g., spam or not spam, cat or dog).
* Regression: The task of predicting numerical outcomes (e.g., house prices, stock prices).

### Gradient Descent

Gradient descent is an optimization algorithm used to find the minimum of a function. In machine learning, it is used to minimize the loss function, which measures the error between the model's predictions and the true values.

### Batch Gradient Descent vs. Stochastic Gradient Descent

* Batch Gradient Descent: Calculates the gradient of the loss function using the entire training set. This can be computationally expensive for large datasets.
* Stochastic Gradient Descent: Calculates the gradient using a single randomly selected training example. This can be faster but may be more noisy.

### Curse of Dimensionality

The curse of dimensionality refers to the challenges that arise when working with high-dimensional data. As the number of features increases, the amount of data needed to fill the feature space grows exponentially, making it difficult to train effective models.

### L1 and L2 Regularization

* L1 Regularization (Lasso): Adds a penalty term to the loss function that encourages sparsity, meaning many model parameters become zero. This can be useful for feature selection.
* L2 Regularization (Ridge): Adds a penalty term to the loss function that discourages large values of model parameters. This can help prevent overfitting.

### Confusion Matrix

A confusion matrix is a table that summarizes the performance of a classification model. It shows the number of true positives, true negatives, false positives, and false negatives.

### AUC-ROC Curve

The AUC-ROC curve (Area Under the Receiver Operating Characteristic Curve) is a plot that shows the performance of a classification model at different classification thresholds. The AUC-ROC score represents the overall performance of the model.

### K-Nearest Neighbors (KNN)

KNN is a lazy learning algorithm that classifies or regresses new data points based on the majority class or average value of the k nearest neighbors in the training set.

### Support Vector Machine (SVM)

SVM is a supervised learning algorithm that finds the optimal hyperplane to separate data points into different classes.

### Kernel Trick

The kernel trick is a mathematical technique used in SVM to map data points into a higher-dimensional space, where it might be easier to find a separating hyperplane.

### SVM Kernels

* Linear kernel: Suitable for linearly separable data.
* Polynomial kernel: Can handle non-linear relationships between features.
* Radial basis function (RBF) kernel: A popular kernel that can capture complex non-linear relationships.

### Hyperplane in SVM

The hyperplane is the decision boundary that separates the data points into different classes. In SVM, the goal is to find the hyperplane that maximizes the margin between the two classes.

### Pros and Cons of SVM

Pros:

* Effective for high-dimensional data.
* Can handle non-linear relationships using kernels.
* Robust to outliers.

Cons:

* Can be computationally expensive for large datasets.
* Choosing the right kernel and hyperparameters can be challenging.

### Hard Margin vs. Soft Margin SVM

* Hard margin SVM: Requires perfect separation of the data points.
* Soft margin SVM: Allows for some misclassification to handle noisy or overlapping data.

### Decision Trees

* Construction: Decision trees are built recursively by splitting the data into subsets based on the values of features.
* Working principle: Each node in the tree represents a test on a feature, and each branch represents a possible outcome of the test. The leaves of the tree contain the predicted class or value.
* Information gain: The feature that results in the largest decrease in entropy is chosen for splitting.
* Gini impurity: Another impurity measure used in decision trees.
* Advantages: Easy to interpret, can handle both numerical and categorical data, robust to outliers.
* Disadvantages: Can be prone to overfitting, especially with deep trees.

### Random Forests

Random forests are an ensemble learning method that combines multiple decision trees.

* Bootstrapping: Each decision tree is trained on a bootstrap sample of the data, which is a random sample with replacement.
* Feature randomness: At each node, a random subset of features is considered for splitting.
* Aggregation: The predictions of all trees are combined (e.g., by majority vote for classification) to make the final prediction.

### Feature Importance

Feature importance measures the contribution of each feature to the model's predictions. In random forests, it can be calculated based on the number of times a feature is used in the decision trees or the decrease in impurity that it achieves.

### Hyperparameters of Random Forests

* Number of trees: The number of decision trees in the forest.
* Maximum depth: The maximum depth of each decision tree.
* Number of features: The number of features considered at each node.
* Bootstrap samples: The number of samples drawn from the dataset for each tree.

### Logistic Regression

Logistic regression is a statistical model used for binary classification problems. It models the probability of an event occurring as a logistic function of linear combinations of the input features.

### Sigmoid Function

The sigmoid function maps any real value to a value between 0 and 1, representing the probability of belonging to the positive class.

### Cost Function

The cost function in logistic regression measures the error between the predicted probabilities and the true labels. A common cost function is the cross-entropy loss.

### Multiclass Logistic Regression

Multiclass logistic regression can be extended to handle more than two classes using techniques like one-vs-rest or softmax.

### L1 and L2 Regularization in Logistic Regression

L1 and L2 regularization can be applied to logistic regression to prevent overfitting and improve generalization.

### XGBoost

XGBoost is a gradient boosting framework that is known for its speed and accuracy. It uses a gradient boosting algorithm that iteratively adds weak learners (decision trees) to the ensemble to improve the overall performance.

### Boosting

Boosting is an ensemble learning method that combines multiple weak learners (e.g., decision trees) to create a strong learner. Each weak learner focuses on correcting the errors made by the previous learners.

### XGBoost Hyperparameters

* Learning rate: Controls the step size in the gradient boosting process.
* Number of estimators: The number of decision trees in the ensemble.
* Maximum depth: The maximum depth of each decision tree.
* Subsample: The fraction of the training data used for each tree.
* Colsample\_bytree: The fraction of features used for each tree.

### Gradient Boosting in XGBoost

XGBoost uses a gradient boosting algorithm that calculates the gradient of the loss function at each iteration and adds a new weak learner that tries to correct the errors made by the previous learners.