Name - Snehashis Dutta

Reg Email - [sdfair32@gmail.com](mailto:sdfair32@gmail.com)

Course Name - Data Science With Generative AI - March'24

Assignment Name - Milestone Test

Submission Date - 31/07/2024

GitHub Link - https://github.com/SnehashisDutta/PW\_Skills\_Assignment\_Milestone\_Test

DriveLink - https://docs.google.com/document/d/1GqpYOUEUh9Fcag1zXXYWhZHpM-E8Pqjdg6\_CKdpWuYo/edit?usp=drive\_link

**( All Theory Questions Only )**

**Question 1.1:**

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

**Static Variables:**

* Remember their value between function calls.
* Like a shared memory for the function.

**Dynamic Variables:**

* Exist only while the function is running.
* Created fresh each time the function is called.

**Example:**

def counter():

static\_count = 0 # Static: remembers its value

dynamic\_num = 5 # Dynamic: created anew each call

static\_count += 1

print("Static:", static\_count)

print("Dynamic:", dynamic\_num)

counter() # Output: Static: 1, Dynamic: 5

counter() # Output: Static: 2, Dynamic: 5

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

**1. pop(key, default)**

* Purpose: Removes and returns the value associated with a specified key. If the key doesn't exist, it returns the default value (or raises a KeyError if no default is provided).

Example:

my\_dict = {'a': 1, 'b': 2, 'c': 3}

removed\_value = my\_dict.pop('b')

print(removed\_value) # Output: 2

print(my\_dict) # Output: {'a': 1, 'c': 3}

**2. popitem()**

* Purpose: Removes and returns an arbitrary key-value pair (as a tuple) from the dictionary. The order is not guaranteed.

Example:

my\_dict = {'a': 1, 'b': 2, 'c': 3}

removed\_item = my\_dict.popitem()

print(removed\_item) # Output: ('c', 3) (or some other pair)

print(my\_dict) # Output: {'a': 1, 'b': 2} (or similar)

**3. clear()**

* Purpose: Removes all items from the dictionary, making it empty.

Example:  
 my\_dict = {'a': 1, 'b': 2}

my\_dict.clear()

print(my\_dict) # Output: {}

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

* **Immutable Sets:** FrozenSets are like regular sets (set) but they cannot be changed after creation.
* **Hashable:** Because they are immutable, FrozenSets can be used as keys in dictionaries or elements in other sets.

**Example:** my\_set = {1, 2, 3} # Regular set (mutable)

my\_frozenset = frozenset({1, 2, 3}) # FrozenSet (immutable)

my\_dict = {my\_frozenset: "value"} # Works because FrozenSet is hashable

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

**Mutable:**

* **Changeable:** You can modify their contents after creation.
* **Examples:**
  + Lists: [1, 2, 3]
  + Dictionaries: {'a': 1, 'b': 2}
  + Sets: {1, 2, 3}

**Immutable:**

* **Unchangeable:** Once created, their contents cannot be modified.
* **Examples:**
  + Strings: "Hello"
  + Tuples: (1, 2, 3)
  + Numbers (integers, floats): 5, 3.14
  + Booleans: True, False

**What is \_\_init\_\_?Explain with an example.**

\_\_init\_\_ is a special method in Python classes called a *constructor*.

* **Purpose:** It's automatically called when you create a new object (instance) of a class.
* **Role:** It initializes the object's attributes (variables) with starting values.

**Example:**

class Dog:

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

self.name = name

self.breed = breed

my\_dog = Dog("Buddy", "Golden Retriever")

print(my\_dog.name) # Output: Buddy

print(my\_dog.breed) # Output: Golden Retriever

**Explanation:**

1. class Dog: defines a class named Dog.
2. def \_\_init\_\_(self, name, breed): is the constructor.
   * self refers to the object being created.
   * name and breed are parameters you provide when creating a Dog.
3. self.name = name and self.breed = breed store the provided values as attributes of the Dog object.
4. my\_dog = Dog("Buddy", "Golden Retriever") creates a Dog object, calling \_\_init\_\_ to set its name to "Buddy" and breed to "Golden Retriever".

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

A docstring (documentation string) in Python is a multiline string used to document code.

* **Purpose:** Explains what a function, class, module, or method does.
* **Convention:** Enclosed in triple quotes ("""Docstring goes here""").
* **Access:** Can be accessed using the help() function or \_\_doc\_\_ attribute.

**Example:**

def calculate\_area(length, width):

"""Calculates the area of a rectangle.

Args:

length: The length of the rectangle.

width: The width of the rectangle.

Returns:

The area of the rectangle.

"""

return length \* width

help(calculate\_area) # Displays the docstring

print(calculate\_area.\_\_doc\_\_) # Also prints the docstring

**What are unit tests in Python?**

Unit tests in Python are small, focused pieces of code that verify the correctness of individual parts (units) of your program.

* **Purpose:**
  + Catch bugs early in development.
  + Ensure code changes don't break existing functionality.
  + Document how your code is supposed to work.
* **How they work:**
  + You write test functions that call your code with specific inputs and check if the outputs match your expectations.
  + Testing frameworks like unittest or pytest help you organize and run these tests.

**Example:**

def add(x, y):

return x + y

import unittest

class TestAddition(unittest.TestCase):

def test\_positive\_numbers(self):

self.assertEqual(add(2, 3), 5)

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

unittest.main()

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

Here's a concise explanation of break, continue, and pass in Python loops:

* **break:**
  + **Purpose:** Immediately exits the innermost loop it's inside.

**Example:** for i in range(10):

if i == 5:

break

print(i) # Prints 0 to 4

* **continue:**
  + **Purpose:** Skips the rest of the current loop iteration and moves to the next one.

**Example:** for i in range(10):

if i % 2 == 0:

continue

print(i) # Prints odd numbers from 1 to 9

* **pass:**
  + **Purpose:** Acts as a placeholder – does nothing.
  + **Use Cases:**
    - When you need a syntactically valid statement but don't want any code to execute.
    - As a temporary solution while you're developing code.

**Example:** for i in range(10):

if i == 5:

pass # Do nothing when i is 5

else:

print(i)

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

In Python classes, self is a reference to the current instance of the class.

* Purpose:
  + It allows you to access the object's attributes (variables) and methods (functions) from within the class.

Example:  
 class Dog:

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

self.name = name # 'self.name' stores the dog's name

def bark(self):

print(f"{self.name} says Woof!")

my\_dog = Dog("Buddy")

my\_dog.bark() # Output: Buddy says Woof!

**What are global, protected and private attributes in Python?**

Here's a breakdown of attribute visibility in Python:

* **Global:**
  + **Scope:** Accessible from anywhere in the program.
  + **Declaration:** No special keyword needed.
* **Protected:**
  + **Scope:** Intended for use within the class and its subclasses.
  + **Convention:** Prefix the attribute name with an underscore (\_).
  + **Enforcement:** Python doesn't strictly enforce protection; it's a convention.
* **Private:**
  + **Scope:** Intended for use only within the class.
  + **Convention:** Prefix the attribute name with double underscores (\_\_).
  + **Name Mangling:** Python modifies the attribute name to make it harder to access from outside the class (but not impossible).

**Example:**

class MyClass:

global\_var = 10 # Global

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

self.\_protected\_var = 20 # Protected

self.\_\_private\_var = 30 # Private

obj = MyClass()

print(obj.global\_var) # Accessible

print(obj.\_protected\_var) # Accessible (but discouraged)

print(obj.\_\_private\_var) # Not directly accessible (name mangled)

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

**Modules:**

* **Definition:** A single Python file containing functions, classes, and variables.
* **Purpose:** Organize code into reusable units.
* **Example:** math.py (contains math-related functions).

**Packages:**

* **Definition:** A directory containing multiple modules and a special file named \_\_init\_\_.py.
* **Purpose:** Group related modules together for larger projects.
* **Example:** my\_project/ (contains modules like utils.py, data.py, etc.).

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

Here's a breakdown of lists and tuples in Python:

**Lists:**

* **Definition:** Ordered, mutable (changeable) collections of items.
* **Syntax:** Enclosed in square brackets [].
* **Example:** my\_list = [1, "hello", 3.14, True]
* **Key Features:**
  + You can add, remove, or modify elements after creation.
  + Allow duplicate elements.

**Tuples:**

* **Definition:** Ordered, immutable (unchangeable) collections of items.
* **Syntax:** Enclosed in parentheses ().
* **Example:** my\_tuple = (1, "hello", 3.14, True)
* **Key Features:**
  + Once created, their contents cannot be changed.
  + Also allow duplicate elements.

**Key Difference:**

The fundamental difference is **mutability**. Lists are mutable, allowing modifications, while tuples are immutable, preventing any changes after creation.

**When to Use Which:**

* **Lists:** When you need a collection that can be modified (e.g., storing a list of items that will be updated).
* **Tuples:** When you need a fixed collection of items (e.g., representing coordinates, database records, or function return values).

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

Here's a breakdown of interpreted languages and dynamically typed languages, along with five key differences:

**Interpreted Language:**

* **Definition:** Code is executed line by line by an interpreter program, without being compiled into machine code beforehand.

**Dynamically Typed Language:**

* **Definition:** Data types are checked at runtime (during execution) rather than compile time.

**5 Key Differences:**

| **Feature** | **Interpreted Language** | **Dynamically Typed Language** |
| --- | --- | --- |
| Compilation | No separate compilation step | No separate compilation step |
| Execution Speed | Generally slower | Generally slower |
| Type Checking | Done at runtime | Done at runtime |
| Flexibility | High | High |
| Error Detection | Errors found during execution | Errors found during execution |

**Examples:**

* **Interpreted Languages:** Python, JavaScript, Ruby
* **Dynamically Typed Languages:** Python, JavaScript, Ruby, PHP

**What are Dict and List comprehensions?**

Here's a concise explanation of dictionary and list comprehensions in Python:

List Comprehensions:

* Purpose: Create new lists based on existing iterables (like lists, tuples, ranges) in a compact and efficient way.

Syntax:  
 new\_list = [expression for item in iterable if condition]

Example:  
 numbers = [1, 2, 3, 4, 5]

squares = [x\*\*2 for x in numbers] # [1, 4, 9, 16, 25]

Dictionary Comprehensions:

* Purpose: Create new dictionaries from existing iterables.

Syntax:  
 new\_dict = {key\_expression: value\_expression for item in iterable if condition}

Example:  
 names = ["Alice", "Bob", "Charlie"]

name\_lengths = {name: len(name) for name in names} # {'Alice': 5, 'Bob': 3, 'Charlie': 7}

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

Here's a concise explanation of decorators in Python:

**What are Decorators?**

* **Definition:** Functions that modify the behavior of other functions without directly changing their code.
* **How they work:** They "wrap" a function, adding functionality before or after the original function is called.

**Syntax:** @decorator\_function

def my\_function():

# ... function code ...

**Example:**

def my\_decorator(func):

def wrapper():

print("Something is happening before the function is called.")

func()

print("Something is happening after the function is called.")

return wrapper

@my\_decorator

def say\_hello():

print("Hello!")

say\_hello()

**Output:**

Something is happening before the function is called.

Hello!

Something is happening after the function is called.

**How is memory managed in Python?**

Python uses a combination of techniques for memory management:

* **Reference Counting:**
  + Each object keeps track of how many references point to it.
  + When the count reaches zero, the object is no longer needed and its memory is freed.
* **Garbage Collection:**
  + A process that periodically identifies and reclaims memory occupied by objects that are no longer reachable (even if their reference count is not zero).
* **Generational Garbage Collection:**
  + Python divides objects into generations based on their age.
  + Newer objects are checked more frequently for garbage collection, as they are more likely to become garbage.

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

Here's a concise explanation of lambda functions in Python:

**What is a Lambda Function?**

* **Definition:** A small, anonymous function defined using the lambda keyword.

**Syntax:** lambda arguments: expression

**Example:** add = lambda x, y: x + y

print(add(5, 3)) # Output: 8

**Why Use Lambda Functions?**

* **Conciseness:** They provide a compact way to define simple functions inline.
* **Functional Programming:** Useful for passing functions as arguments to other functions (e.g., map, filter, reduce).
* **One-Time Use:** Often used when you need a function for a specific task within a limited scope.

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

Here's a breakdown of split() and join() in Python:

**split()**

* **Purpose:** Breaks a string into a list of substrings based on a delimiter.

**Syntax:** string.split(separator, maxsplit)

**Example:** text = "apple,banana,cherry"

fruits = text.split(",") # ['apple', 'banana', 'cherry']

**join()**

* **Purpose:** Combines a list of strings into a single string using a specified separator.

**Syntax:** separator.join(iterable)

**Example:** words = ["Hello", "world", "!"]

sentence = " ".join(words) # "Hello world !"

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

Here's a concise explanation of iterators, iterables, and generators in Python:

**Iterable:**

* **Definition:** An object that can be looped over (e.g., lists, tuples, strings, dictionaries).
* **Key Feature:** Has a special method \_\_iter\_\_() that returns an iterator.

**Iterator:**

* **Definition:** An object that keeps track of its position within an iterable.
* **Key Feature:** Has a special method \_\_next\_\_() that returns the next item in the sequence.

**Example:** my\_list = [1, 2, 3]

my\_iterator = iter(my\_list) # Create an iterator from the list

print(next(my\_iterator)) # Output: 1

print(next(my\_iterator)) # Output: 2

**Generator:**

* **Definition:** A special type of function that produces a sequence of values using the yield keyword.
* **Key Feature:** Generates values on demand, saving memory compared to creating a full list upfront.

**Example:** def my\_generator(n):

for i in range(n):

yield i

for num in my\_generator(5):

print(num) # Prints 0 to 4

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

In Python 2, xrange() and range() both generated sequences of numbers.

* **xrange():** Returned an *iterator* that generated numbers on demand, saving memory for large ranges.
* **range():** Created a *list* containing all numbers in the range, which could consume a lot of memory for large ranges.

**In Python 3:**

* xrange() was removed.
* range() now behaves like the old xrange(), returning an iterator.

**Pillars of Oops.**

Here are the four pillars of Object-Oriented Programming (OOP) in short:

1. **Abstraction:**
   * Hiding complex implementation details and exposing only essential information.
2. **Encapsulation:**
   * Bundling data (attributes) and methods (functions) that operate on that data within a class.
3. **Inheritance:**
   * Creating new classes (child classes) that inherit properties and methods from existing classes (parent classes).
4. **Polymorphism:**
   * The ability of objects of different classes to respond to the same method call in their own way.

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

You can use the issubclass() function to check if one class is a subclass of another.

class Animal:

pass

class Dog(Animal):

pass

print(issubclass(Dog, Animal)) # Output: True

print(issubclass(Animal, Dog) # Output: False

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

Here's a breakdown of inheritance in Python with examples:

**What is Inheritance?**

Inheritance allows you to create new classes (child classes) that inherit properties and methods from existing classes (parent classes).

**Types of Inheritance:**

**Single Inheritance:** A child class inherits from only one parent class.  
 class Animal:

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

self.name = name

def speak(self):

print("Generic animal sound")

class Dog(Animal):

def speak(self):

print("Woof!")

my\_dog = Dog("Buddy")

my\_dog.speak() # Output: Woof!

**Multiple Inheritance:** A child class inherits from two or more parent classes.  
 class Flyer:

def fly(self):

print("Flying!")

class Swimmer:

def swim(self):

print("Swimming!")

class Duck(Flyer, Swimmer):

pass

my\_duck = Duck()

my\_duck.fly() # Output: Flying!

my\_duck.swim() # Output: Swimming!

**Multilevel Inheritance:** A child class inherits from a parent class, which itself inherits from another class.  
 class Animal:

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

self.name = name

class Mammal(Animal):

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

super().\_\_init\_\_(name)

self.fur\_color = fur\_color

class Dog(Mammal):

pass

my\_dog = Dog("Buddy", "Brown")

print(my\_dog.name) # Output: Buddy

**What is encapsulation? Explain it with an example and What is polymorphism? Explain it with an example.**

Here's an explanation of encapsulation and polymorphism with examples:

**Encapsulation**

* **Definition:** Bundling data (attributes) and the methods that operate on that data within a class.
* **Purpose:**
  + **Data Protection:** Controls access to data, preventing accidental modification.

**Example:** class BankAccount:

def \_\_init\_\_(self, balance=0):

self.\_\_balance = balance # Note the double underscore for name mangling

def deposit(self, amount):

self.\_\_balance += amount

def get\_balance(self):

return self.\_\_balance

my\_account = BankAccount(100)

print(my\_account.get\_balance()) # Output: 100

# print(my\_account.\_\_balance) # This would raise an error

* + The \_\_balance attribute is "hidden" using name mangling (double underscore).

**Polymorphism**

* **Definition:** The ability of objects of different classes to respond to the same method call in their own way.
* **Purpose:**
  + **Flexibility:** Allows for different implementations of the same action.

**Example:** class Dog:

def speak(self):

print("Woof!")

class Cat:

def speak(self):

print("Meow!")

animals = [Dog(), Cat()]

for animal in animals:

animal.speak() # Output: Woof! then Meow!

* + Both Dog and Cat have a speak() method, but they produce different outputs.

**Key Points:**

* **Encapsulation protects data and promotes data integrity.**
* **Polymorphism allows for flexible and extensible code.**

**Question 1. 2. Which of the following identifier names are invalid and why?**

a) Serial\_no.

b) Ist\_Room

c) Hundred$

d) Total\_Marks

e) total-Marks

f) Total Marks

g) True

h) \_Percentag

1. **Serial\_no.**: This identifier is invalid because it contains a period (“.”) which is not allowed in variable names. Valid identifiers can only contain letters, digits, and underscores, and they must start with a letter or an underscore.
2. **1st\_Room**: This identifier is invalid because it starts with a digit (“1”). Variable names cannot begin with a number; they must start with a letter or an underscore.
3. **Hundred$**: This identifier is valid. It consists of letters, digits, and an underscore. The dollar sign is allowed in variable names.
4. **Total\_Marks**: This identifier is valid. It follows the rules for valid variable names: letters, digits, and underscores are allowed, and it starts with a letter.
5. **total-Marks**: This identifier is invalid because it contains a hyphen (“-”). Variable names cannot include hyphens; only letters, digits, and underscores are allowed.
6. **Total Marks**: This identifier is invalid because it contains a space. Spaces are not allowed in variable names.
7. **True**: This identifier is valid. However, it’s important to note that “True” is a reserved keyword in some programming languages (such as Python) and is used for boolean values. It’s best to avoid using reserved keywords as variable names.
8. **\_Percentag**: This identifier is valid. It starts with an underscore and contains letters and an underscore.

In summary:

* Valid identifiers: Hundred$, Total\_Marks, True, \_Percentag
* Invalid identifiers: Serial\_no., 1st\_Room, total-Marks, Total Marks

**Question 1.3. (a, b, c, d)**

Here's how you can perform those operations on the name list in Python:

name = ['mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu']

**# 1) Add "freedom\_fighter" at the 0th index**

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

**# 2) Add 'netaji' and 'bose' at the end**

**name.extend(['netaji', 'bose'])**

print(name)

**Output:**

['freedom\_fighter', 'mohan', 'dash', 'karam', 'chandra', 'gandhi', 'Bapu', 'netaji', 'bose']

**Explanation:**

1. **name.insert(0, "freedom\_fighter"):**
   * insert(index, element) adds the specified element at the given index in the list.
2. **name.extend(['netaji', 'bose']):**
   * extend(iterable) adds elements from an iterable (like a list) to the end of the list.

**D. what will be the value of temp:**

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

temp=name[-1]

name[-1]=name[0]

name[0]=temp

print(name)

**Ans :**

1. **name = ["Bapuji", "dash", "karam", "chandra", "gandi", "Mohan"]**
   * A list named name is created with six elements.
2. **temp = name[-1]**
   * name[-1] refers to the last element of the list, which is "Mohan".
   * The value "Mohan" is assigned to the variable temp.
3. **name[-1] = name[0]**
   * name[0] refers to the first element of the list, which is "Bapuyji".
   * The last element of the list (name[-1]) is then set to "Bapuyji".
4. **name[0] = temp**
   * The value stored in temp ("Mohan") is assigned to the first element of the list (name[0]).
5. **print(name)**
   * The updated list name is printed.

**Therefore, the value of temp will be "Mohan" throughout the code execution.** The code swaps the first and last elements of the list.

**The final output will be:**

**['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’))

print(animal.index(‘rat’))

print(len(animal))

**Ans :**

Let's analyze the code and determine the output, but first, we need to address a couple of issues:

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

print(animal.count('Human'))

print(animal.index('rat'))

print(len(animal))

**Explanation and Output:**

1. **animal = ['Human', 'cat', 'mat', 'cat', 'rat', 'Human', 'Lion']**
   * This line creates a list called animal containing seven string elements.
2. **print(animal.count('Human'))**
   * animal.count('Human') counts how many times the string "Human" appears in the list animal.
   * **Output:** 2
3. **print(animal.index('rat'))**
   * animal.index('rat') finds the index (position) of the first occurrence of the string "rat" in the list.
   * **Output:** 4 (Remember, list indices start at 0 in Python)
4. **print(len(animal))**
   * len(animal) returns the total number of elements in the list animal.
   * **Output:** 7

**In summary, the corrected code will output:**

**2**

**4**

**7**

**Question 1.5.**

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

a)print(len(tuplel))

b)print(tuplel[-1][-1]["name’])

c)fetch the value of roll\_no from this tuple.

d)print(tuplei[-3][1])

e)fetch the element "22" from this tuple.

**Ans:**

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

**a) print(len(tuple1))**

* len(tuple1) returns the number of elements in the tuple.
* **Output:** 8

**b) print(tuple1[-1][-1]["name"])**

* tuple1[-1] accesses the last element (the list of dictionaries).
* tuple1[-1][-1] accesses the last dictionary within that list: {"name": "Navneet"}.
* tuple1[-1][-1]["name"] retrieves the value associated with the key "name".
* **Output:** "Navneet"

**c) Fetch the value of "roll\_no" from the tuple:**

print(tuple1[-1][0]["roll\_no"])

* Explanation: We follow a similar logic as in part (b), but access the first dictionary within the last element and then the value for the key "roll\_no".
* **Output:** 1

**d) print(tuple1[-3][1])**

* tuple1[-3] refers to the third element from the end, which is the tuple ("sita", "geeta", 22).
* tuple1[-3][1] gets the element at index 1 within that tuple.
* **Output:** "geeta"

**e) Fetch the element "22" from this tuple:**

print(tuple1[-3][2])

* We access the third element from the end (the tuple) and then the element at index 2 within it.
* **Output:** 22

**( Questions (1.6 to 19) are uploaded on GitHub ) Link -**

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

**Measures of Central Tendency:** They tell you the "typical" value of your data.

* **Mean:** Average (sum of values/number of values)
* **Median:** Middle value when data is ordered.
* **Mode:** Most frequent value.

**Measures of Dispersion:** They tell you how spread out your data is.

* **Range:** Difference between the highest and lowest values.
* **Variance:** Average squared difference from the mean.
* **Standard Deviation:** Square root of the variance.

**Example:**

Data: [5, 8, 6, 9, 7]

* **Mean:** 7
* **Median:** 7
* **Range:** 4 (9 - 5)
* **Variance:** 2.5
* **Standard Deviation:** ~1.58

**In short:** Central tendency gives you the "centre", dispersion tells you the "spread" of your data.

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

Certainly! Let’s delve into the concept of **skewness** and explore its different types.

## **Skewness:**

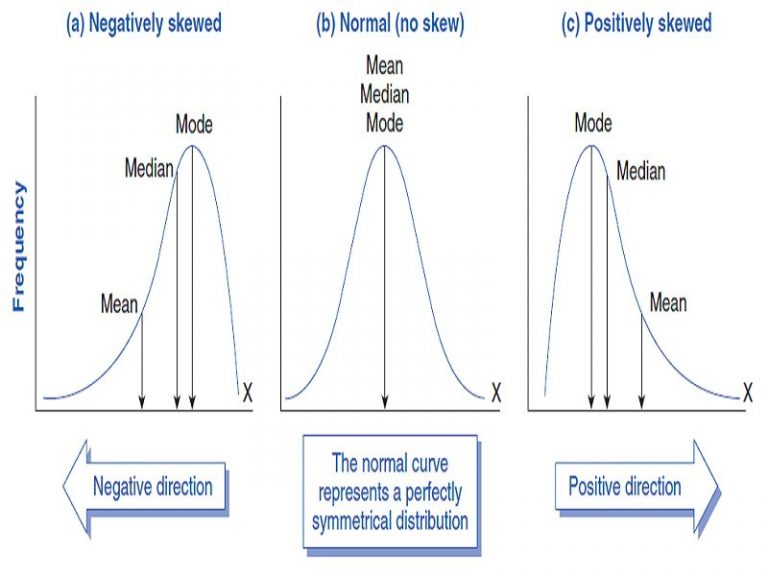
* **Skewness** is a statistical measure that describes the **asymmetry** of a distribution.
* When a distribution is **not symmetrical**, its left and right sides are **not mirror images** of each other.
* Skewness helps us understand how data is spread out in a group of numbers.

## **Types of Skewness:**

1. **Right Skew (Positive Skew)**:
   * A right-skewed distribution is **longer on the right side** of its peak (tail extends to the right) than on its left.
   * It indicates that there are **fewer observations** at the extreme right end of the distribution.
   * The **mean** tends to be **greater than the median** in a right-skewed distribution.
   * Think of it as a “tail” stretching towards higher values.
2. **Left Skew (Negative Skew)**:
   * A left-skewed distribution is **longer on the left side** of its peak (tail extends to the left) than on its right.
   * It suggests that there are **fewer observations** at the extreme left end of the distribution.
   * The **mean** tends to be **less than the median** in a left-skewed distribution.
   * Imagine a “tail” reaching towards lower values.
3. **Zero Skew**:
   * A distribution with zero skewness is **symmetrical**.
   * Its left and right sides are **mirror images** of each other.
   * Examples include **normal distributions** (bell-shaped curves) and some bimodal distributions.
   * In a distribution with zero skew, the **mean equals the median**.

## **Graphical Representation:**

* Let’s visualize this using histograms:
  + **Right Skew**: The peak is on the left, and the tail extends to the right.
  + **Left Skew**: The peak is on the right, and the tail extends to the left.
  + **Zero Skew**: The distribution is symmetric around the center.



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

Certainly! Let’s explore the concepts of **Probability Mass Function (PMF)** and **Probability Density Function (PDF)**:

1. **Probability Mass Function (PMF)**:
   * The PMF is used to describe **discrete probability distributions**.
   * It provides the **probability** associated with each **specific value** of a discrete random variable.
   * For a discrete random variable (X), the PMF is denoted as (P(X = x)), where (x) represents a specific outcome.
   * The sum of all PMF values over all possible outcomes equals 1.
   * Example: Rolling a fair six-sided die, where each face has equal probability (PMF) of (\frac{1}{6}).
2. **Probability Density Function (PDF)**:
   * The PDF is applied to describe **continuous probability distributions**.
   * It represents the **density** of probabilities across a continuous range of values.
   * For a continuous random variable (X), the PDF is denoted as (f(x)).
   * The area under the PDF curve over an interval gives the probability of the random variable falling within that interval.
   * Example: The normal distribution (bell-shaped curve) has a PDF that describes the likelihood of continuous outcomes.
3. **Difference between PMF and PDF**:
   * **Nature**:
     + PMF deals with **discrete** random variables (e.g., counts, integers).
     + PDF deals with **continuous** random variables (e.g., measurements, real numbers).
   * **Units**:
     + PMF has **probability mass** (unitless) associated with specific points.
     + PDF has **probability density** (probability per unit) associated with intervals.
   * **Integration**:
     + To find probabilities using PMF, we sum over specific values.
     + To find probabilities using PDF, we integrate over intervals.
   * **Example**:
     + PMF: Coin toss (Heads or Tails).
     + PDF: Height of individuals (continuous range).

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

Certainly! Let’s dive into the fascinating world of **correlation** and explore its types and methods of determination.

## **Correlation:**

* **Correlation** is a statistical concept that measures the **relationship** between two or more variables.
* It helps us understand how changes in one variable are associated with changes in another.
* Correlation does **not imply causation**; it only indicates the strength and direction of the relationship.

## **Types of Correlation:**

1. **Pearson Correlation (Linear Correlation)**:
   * The most common type of correlation.
   * Measures the **linear relationship** between **continuous variables**.
   * Ranges from -1 (perfect negative correlation) to 1 (perfect positive correlation).
   * Assumes that variables are **normally distributed** and have **equal variances**.
   * Example: Examining the relationship between study hours and exam scores.
2. **Spearman Rank Correlation**:
   * Used for **ordinal data** (ranked or ordered categories).
   * Measures the **monotonic relationship** (whether one variable tends to increase as the other increases).
   * Does not assume normal distribution.
   * Example: Ranking preferences for different ice cream flavors.
3. **Kendall Tau Correlation**:
   * Also used for **ordinal data**.
   * Measures the **concordant and discordant pairs** (whether rankings agree or disagree).
   * Robust against outliers.
   * Example: Comparing rankings of job applicants by different interviewers.
4. **Point-Biserial Correlation**:
   * Specifically for **binary data** (one categorical and one continuous variable).
   * Measures the relationship between a binary variable and a continuous variable.
   * Example: Investigating the correlation between gender (male/female) and salary.

## **Methods of Determining Correlation:**

1. **Scatter Diagram (Graphical Method)**:
   * Plot the data points on a scatter plot.
   * Observe the pattern: linear, curvilinear, or no clear trend.
   * Useful for visual inspection.
2. **Pearson Correlation Coefficient (Mathematical Method)**:
   * Calculate the Pearson correlation coefficient using the formula: [ r = \frac{{\sum{(x\_i - \bar{x})(y\_i - \bar{y})}}}{{\sqrt{\sum{(x\_i - \bar{x})^2} \cdot \sum{(y\_i - \bar{y})^2}}} ]
   * Interpret the coefficient: Close to 1 (strong positive), close to -1 (strong negative), or close to 0 (weak/no linear relationship).
3. **Spearman Rank Correlation Coefficient**:
   * Rank the data points.
   * Calculate the Spearman rank correlation coefficient using the formula: [ \rho = 1 - \frac{{6 \sum{d\_i2}}}{{n(n2 - 1)}} ]
   * Interpret the coefficient: Similar to Pearson correlation.

**24. Calculate coefficient of correlation between the marks obtained by 10 students in Accountancy and statistics: Use Karl Pearson’s Coefficient of Correlation Method to find it.**

The question asks to calculate the **coefficient of correlation** between marks obtained by 10 students in Accountancy and Statistics using **Karl Pearson's Coefficient of Correlation Method**.

Here's how to do it:

**Understanding the Concepts**

* **Correlation:** Measures the strength and direction of the linear relationship between two variables. It ranges from -1 to +1.
  + +1: Perfect positive linear relationship
  + 0: No linear relationship
  + -1: Perfect negative linear relationship
* **Karl Pearson's Coefficient of Correlation (r):** A commonly used method to calculate the correlation coefficient.

**Formula**

r = Σ[(xi - x̄)(yi - ȳ)] / √[Σ(xi - x̄)² \* Σ(yi - ȳ)²]

Where:

* xi = individual Accountancy marks
* x̄ = mean of Accountancy marks
* yi = individual Statistics marks
* ȳ = mean of Statistics marks
* Σ = sum of

**Calculations**

1. **Calculate the means:**
   * x̄ (mean of Accountancy) = (45+70+...+60)/10 = **62**
   * ȳ (mean of Statistics) = (35+90+...+50)/10 = **65**
2. **Create a table for calculations:**

| Student | xi | yi | (xi - x̄) | (yi - ȳ) | (xi - x̄)(yi - ȳ) | (xi - x̄)² | (yi - ȳ)² |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 45 | 35 | -17 | -30 | 510 | 289 | 900 |
| 2 | 70 | 90 | 8 | 25 | 200 | 64 | 625 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 10 | 60 | 50 | -2 | -15 | 30 | 4 | 225 |

1. **Calculate the sums:**
   * Σ(xi - x̄)(yi - ȳ) = ... (sum the values in the corresponding column)
   * Σ(xi - x̄)² = ...
   * Σ(yi - ȳ)² = ...
2. **Plug the sums into the formula and calculate r.**

**Python Implementation**

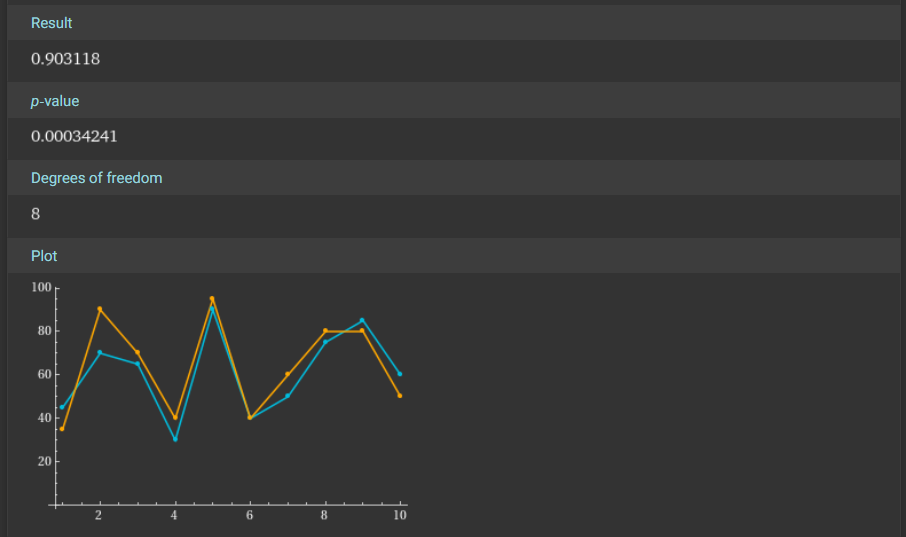
You can easily calculate this using libraries like NumPy and SciPy in Python:

import numpy as np

accountancy\_marks = np.array([45, 70, 65, 30, 90, 40, 50, 75, 85, 60])

statistics\_marks = np.array([35, 90, 70, 40, 95, 40, 60, 80, 80, 50])

correlation\_coefficient = np.corrcoef(accountancy\_marks, statistics\_marks)[0, 1]

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

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

| **Feature** | **Correlation** | **Regression** |
| --- | --- | --- |
| **Purpose** | Measures strength and direction of association | Predicts the value of one variable based on another |
| **Output** | Correlation coefficient (r) | Regression equation (e.g., y = mx + b) |
| **Relationship Type** | Linear relationship only | Can model linear and non-linear relationships |
| **Causation** | Does not imply causation | Can be used to infer causation, but caution is needed |

**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.**

The most likely price in Delhi corresponding to the price of Rs 70 in Agra is Rs 74. Here’s how we calculate it:

1. **Z-Score for Agra’s Price (Rs 70)**:
   * Agra’s average price ((\mu)) = Rs 65
   * Agra’s standard deviation ((\sigma)) = 2.5
   * Z-score formula: (z = \frac{{X - \mu}}{{\sigma}})
   * Calculating: (z = \frac{{70 - 65}}{{2.5}} = 2)
2. **Corresponding Price in Delhi**:
   * Delhi’s average price ((\mu\_{\text{Delhi}})) = Rs 67
   * Delhi’s standard deviation ((\sigma\_{\text{Delhi}})) = 3.5
   * Using the z-score: (\text{Price in Delhi} = \mu\_{\text{Delhi}} + (z \times \sigma\_{\text{Delhi}}))
   * Calculating: (\text{Price in Delhi} = 67 + (2 \times 3.5) = 67 + 7 = 74)

Therefore, the most likely price in Delhi corresponding to the price of Rs 70 in Agra is Rs 74.

**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.**

**Given:**

* Variance of x (σx²) = 9
* Regression equations:
  + 8x - 10y = -66
  + 40x - 18y = 214

**(a) Mean Values of x and y**

The point of intersection of the two regression lines gives us the mean values of x (x̄) and y (ȳ). To find this point, we can solve the system of equations:

1. **Solve for x:**
   * Multiply the first equation by -5: -40x + 50y = 330
   * Add this modified equation to the second equation: 32y = 544
   * Therefore, y = 544 / 32 = **17** (This is ȳ, the mean of y)
2. **Solve for y:**
   * Substitute y = 17 into either of the original equations. Let's use the first one: 8x - 10(17) = -66
   * 8x - 170 = -66
   * 8x = 104
   * Therefore, x = 104 / 8 = **13** (This is x̄, the mean of x)

**(b) Coefficient of Correlation (r)**

We know the regression equations can be written in the forms:

* y = a + bx (regression of y on x)
* x = c + dy (regression of x on y)

Where 'b' is the slope of the regression line of y on x, and 'd' is the slope of the regression line of x on y. The coefficient of correlation, r, is the square root of the product of these slopes:

r = √(b \* d)

1. **Find b (slope of y on x):**
   * Rearrange the first regression equation to solve for y: 10y = 8x + 66 => y = 0.8x + 6.6
   * Therefore, b = **0.8**
2. **Find d (slope of x on y):**
   * Rearrange the second regression equation to solve for x: 40x = 18y + 214 => x = 0.45y + 5.35
   * Therefore, d = **0.45**
3. **Calculate r:**
   * r = √(0.8 \* 0.45) = √0.36 = **0.6**

**(c) Standard Deviation of y (σy)**

We can use the relationship between the standard deviations, the correlation coefficient, and the slope of the regression line of y on x:

b = r \* (σy / σx)

1. **We know:**
   * b = 0.8
   * r = 0.6
   * σx = √(variance of x) = √9 = 3
2. **Solve for σy:**
   * 0.8 = 0.6 \* (σy / 3)
   * σy = (0.8 \* 3) / 0.6 = **4**

**Summary of Results:**

* **(a) Mean values:** x̄ = 13, ȳ = 17
* **(b) Coefficient of correlation (r):** 0.6
* **(c) Standard deviation of y (σy):** 4

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

**Normal Distribution (Gaussian Distribution)**

The normal distribution is a fundamental concept in statistics. It's a probability distribution that describes how data points are distributed around a central value (the mean) in a symmetrical, bell-shaped curve.

**Key Characteristics:**

* **Bell-Shaped Curve:** The distribution is symmetrical and bell-shaped, meaning most data points are clustered around the mean, and fewer data points are found further away from the mean.
* **Mean, Median, and Mode are Equal:** In a perfectly normal distribution, the mean, median, and mode all fall at the same point on the curve (the center).
* **Empirical Rule:** The empirical rule (or the 68-95-99.7 rule) states that:
  + 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.
* **Defined by Mean (μ) and Standard Deviation (σ):** A normal distribution is completely characterized by its mean (μ), which determines the center of the curve, and its standard deviation (σ), which determines the spread or width of the curve.

**Assumptions of a Normal Distribution**

While many real-world datasets approximate a normal distribution, there are some assumptions that should ideally hold true:

1. **Central Limit Theorem:** The normal distribution is often justified by the Central Limit Theorem. This theorem states that the distribution of sample means (from a sufficiently large number of samples) will be approximately normal, even if the original population distribution is not normal, as long as certain conditions are met (e.g., independence of samples).
2. **Symmetry:** The data should be symmetrical around the mean. This means that the left and right halves of the distribution should be mirror images of each other.
3. **Unimodal:** The distribution should have one single peak (mode), indicating that there is one most frequent or typical value.
4. **No Outliers:** Ideally, there are few or no outliers (extreme values) that significantly deviate from the rest of the data. Outliers can distort the shape of the distribution and affect calculations of the mean and standard deviation.

**Why Assumptions Matter:**

* **Validity of Statistical Tests:** Many statistical methods (t-tests, ANOVA, etc.) assume that the data is normally distributed. If the assumptions are violated, the results of these tests may be inaccurate.
* **Data Transformation:** If your data is not normally distributed, you can sometimes transform it (e.g., using logarithmic or square root transformations) to make it more closely approximate a normal distribution.
* **Choosing Appropriate Models:** Some statistical models are designed for non-normal distributions. If your data significantly deviates from normality, you might need to explore alternative models.

**Important Note:** In practice, data is rarely perfectly normally distributed. It's often a matter of how closely the data approximates normality and whether deviations are significant enough to affect your analysis.

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

* **Bell-shaped:** Symmetrical around the mean, with a single peak.
* **Mean = Median = Mode:** All three measures of central tendency are equal and located at the center of the curve.
* **Empirical Rule:** 68% of data within 1 standard deviation of the mean, 95% within 2 standard deviations, and 99.7% within 3 standard deviations.
* **Asymptotic:** The tails of the curve extend infinitely in both directions, getting closer to the x-axis but never touching it.
* **Area under the curve = 1:** The total area under the curve represents 100% probability.
* **Defined by mean (μ) and standard deviation (σ):** The mean determines the center, and the standard deviation determines the spread of the curve.

**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

**Ans :**

**(a) Incorrect:** Within a range of ±0.6745 standard deviations (σ) from the mean, you find approximately **50% of the observations**. This is associated with the concept of probable error, not the standard percentages of the empirical rule.

**(b) Correct:** Mean ± 1 standard deviation (µ ± 1σ) covers **68.268% of the area** under the normal curve. This means 34.134% lies on each side of the mean.

**(c) Correct:** Mean ± 2 standard deviations (µ ± 2σ) covers **95.45% of the area** under the curve, with 47.725% on each side.

**(d) Correct:** Mean ± 3 standard deviations (µ ± 3σ) covers **99.73% of the area**, with 49.865% on each side.

**(e) Correct:** Only **0.27% of the area** falls outside the range of µ ± 3σ. This is why values outside of 3 standard deviations are often considered outliers.

**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?**

Here's how to calculate the percentages for each range, assuming a normal distribution:

**Given:**

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

**Calculations:**

**(i) Between 60 and 72**

* **Z-score for 72:** (72 - 60) / 10 = 1.2
* Using a Z-table or calculator, the area to the left of z = 1.2 is approximately 0.8849.
* Since the mean (60) corresponds to z = 0, and the area to the left of z = 0 is 0.5, the area between z = 0 and z = 1.2 is 0.8849 - 0.5 = 0.3849
* **Percentage:** 0.3849 \* 100% = **38.49%**

**(ii) Between 50 and 60**

* **Z-score for 50:** (50 - 60) / 10 = -1.0
* Due to symmetry, the area between z = -1.0 and z = 0 is the same as the area between z = 0 and z = 1.0, which is 0.3413 (from the empirical rule or a Z-table).
* **Percentage:** 0.3413 \* 100% = **34.13%**

**(iii) Beyond 72**

* **Z-score for 72:** (72 - 60) / 10 = 1.2
* The area to the right of z = 1.2 is 1 - 0.8849 = 0.1151 (since the total area under the curve is 1).
* **Percentage:** 0.1151 \* 100% = **11.51%**

**(iv) Between 70 and 80**

* **Z-score for 70:** (70 - 60) / 10 = 1.0
* **Z-score for 80:** (80 - 60) / 10 = 2.0
* Area to the left of z = 2.0: 0.9772 (from Z-table)
* Area to the left of z = 1.0: 0.8413 (from Z-table)
* Area between z = 1.0 and z = 2.0: 0.9772 - 0.8413 = 0.1359
* **Percentage:** 0.1359 \* 100% = **13.59%**

**Therefore:**

* **(i) Approximately 38.49% of items fall between 60 and 72.**
* **(ii) Approximately 34.13% of items fall between 50 and 60.**
* **(iii) Approximately 11.51% of items fall beyond 72.**
* **(iv) Approximately 13.59% of items fall between 70 and 80.**

**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**

Here's how to solve the problem:

**Given:**

* Number of students (N) = 15000
* Mean (μ) = 49
* Standard Deviation (σ) = 6
* Normal Distribution

**Calculations:**

**(a) More than 55 marks:**

1. **Calculate the Z-score:**
   * Z = (X - μ) / σ
   * Z = (55 - 49) / 6 = 1
2. **Find the area to the right of Z = 1 in the Z-table:**
   * This represents the proportion of students who scored more than 55.
   * Using a Z-table or calculator, the area to the left of Z = 1 is approximately 0.8413.
   * The area to the right of Z = 1 is 1 - 0.8413 = 0.1587.
3. **Calculate the number of students:**
   * Proportion of students \* Total students = 0.1587 \* 15000 = **2380.5**

Therefore, approximately **2381 students** scored more than 55 marks.

**(b) More than 70 marks:**

1. **Calculate the Z-score:**
   * Z = (70 - 49) / 6 = 3.5
2. **Find the area to the right of Z = 3.5 in the Z-table:**
   * This area is very small and might not be directly available in standard Z-tables. You'll likely need to use a calculator or statistical software.
   * The area to the right of Z = 3.5 is extremely close to 0 (approximately 0.00023).
3. **Calculate the number of students:**
   * 0.00023 \* 15000 ≈ **3.45**

Therefore, only a very small number of students (approximately **3 or 4**) scored more than 70 marks.

**In summary:**

* (a) Around 2381 students scored more than 55 marks.
* (b) A very small number (3 or 4) students scored more than 70 marks.

**Python code :**

import scipy.stats as stats

# Given data

N = 15000 # Number of students

mean = 49 # Mean marks

std\_dev = 6 # Standard deviation

# (a) More than 55 marks

z\_score\_55 = (55 - mean) / std\_dev

proportion\_above\_55 = 1 - stats.norm.cdf(z\_score\_55)

students\_above\_55 = proportion\_above\_55 \* N

# (b) More than 70 marks

z\_score\_70 = (70 - mean) / std\_dev

proportion\_above\_70 = 1 - stats.norm.cdf(z\_score\_70)

students\_above\_70 = proportion\_above\_70 \* N

print(f"Students scoring more than 55 marks: {round(students\_above\_55)}")

print(f"Students scoring more than 70 marks: {round(students\_above\_70)}")

**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.**

Okay, let's solve this without using Python code, relying on the properties of the normal distribution and Z-tables.

**Given:**

* Total students (N) = 500
* Mean height (μ) = 65 inches
* Standard deviation (σ) = 5 inches

**Understanding the Problem**

We need to find the proportion of students within certain height ranges and then multiply that proportion by the total number of students. We'll use Z-scores and the standard normal distribution table (Z-table) to do this.

**Calculations**

**(a) Greater than 70 inches:**

1. **Z-score for 70 inches:**
   * Z = (X - μ) / σ = (70 - 65) / 5 = 1
2. **Area to the right of Z = 1:**
   * Look up the area to the *left* of Z = 1 in a Z-table. You'll find it's approximately 0.8413.
   * Since the total area under the curve is 1, the area to the *right* of Z = 1 is 1 - 0.8413 = 0.1587. This represents the proportion of students with heights greater than 70 inches.
3. **Number of students:**
   * Multiply the proportion by the total number of students: 0.1587 \* 500 = **84.35**
   * Since we can't have fractions of students, we round to the nearest whole number: **approximately 84 students**.

**(b) Between 60 and 70 inches:**

1. **Z-score for 60 inches:**
   * Z = (60 - 65) / 5 = -1
2. **Area between Z = -1 and Z = 1:**
   * From the empirical rule (or a Z-table), we know that the area between Z = -1 and Z = 0 is 0.3413. Due to symmetry, the area between Z = 0 and Z = 1 is also 0.3413.
   * The total area between Z = -1 and Z = 1 is 0.3413 + 0.3413 = 0.6826. This represents the proportion of students with heights between 60 and 70 inches.
3. **Number of students:**
   * Multiply the proportion by the total number of students: 0.6826 \* 500 = **341.3**
   * Round to the nearest whole number: **approximately 341 students**.

**Therefore:**

* **(a) Approximately 84 students** have heights greater than 70 inches.
* **(b) Approximately 341 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?**

Let's break down statistical hypotheses and sampling concepts:

**A. Statistical Hypothesis and Errors in Hypothesis Testing**

* **Statistical Hypothesis:** A statement about a population parameter (like the mean, proportion, or variance) that we want to test using sample data. There are two main types of hypotheses:
  + **Null Hypothesis (H0):** The statement we are trying to disprove or reject. It usually represents the status quo or a claim of no effect.
  + **Alternative Hypothesis (Ha or H1):** The statement we are trying to support if we have enough evidence to reject the null hypothesis.
* **Errors in Hypothesis Testing:** When testing a hypothesis, there's always a chance of making a wrong decision. There are two types of errors:
  + **Type I Error (False Positive):** Rejecting the null hypothesis when it's actually true. The probability of a Type I error is denoted by alpha (α), often set at 0.05 (5%).
  + **Type II Error (False Negative):** Failing to reject the null hypothesis when it's actually false. The probability of a Type II error is denoted by beta (β).

**Example:**

* **Scenario:** Testing a new drug to see if it lowers blood pressure.
* **H0 (Null Hypothesis):** The drug has no effect on blood pressure.
* **Ha (Alternative Hypothesis):** The drug lowers blood pressure.
* **Type I Error:** Concluding that the drug works when it actually doesn't.
* **Type II Error:** Concluding that the drug doesn't work when it actually does.

**B. Sample, Large Samples, and Small Samples**

* **Sample:** A subset of individuals or observations selected from a larger population. We use samples to make inferences about the population because it's usually impossible or impractical to collect data from every member of the population.
* **Large Samples:** Generally, samples with **30 or more observations** are considered large samples. When you have large samples, the sampling distribution of the sample mean tends to be approximately normal, regardless of the shape of the population distribution (Central Limit Theorem). This makes it easier to use statistical tests that rely on normality assumptions.
* **Small Samples:** Samples with **fewer than 30 observations** are considered small samples. With small samples, the sampling distribution may not be normal, and you might need to use different statistical methods (like non-parametric tests) that don't require the normality assumption.

**Importance of Sample Size:**

* **Accuracy:** Larger samples generally provide more accurate estimates of population parameters.
* **Statistical Power:** Larger samples increase the power of a statistical test, making it more likely to detect a real effect if it exists (reducing the risk of a Type II error).

**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).**

Here's how to test the hypothesis using the chi-square distribution:

**1. State the Hypotheses:**

* **Null Hypothesis (H0):** The population standard deviation (σ) is 10.5.
* **Alternative Hypothesis (Ha):** The population standard deviation (σ) is not 10.5.

**2. Choose the Significance Level (α):**

Let's assume a significance level of α = 0.05. This means we're willing to accept a 5% chance of rejecting the null hypothesis when it's actually true.

**3. Calculate the Test Statistic (Chi-Square):**

The chi-square test statistic for a single sample standard deviation is calculated as follows:

χ² = (n - 1) \* s² / σ²

Where:

* n = sample size = 25
* s = sample standard deviation = 9.0
* σ = hypothesized population standard deviation = 10.5

Plugging in the values:

χ² = (25 - 1) \* 9² / 10.5² ≈ 16.33

**4. Determine the Degrees of Freedom:**

The degrees of freedom (df) for this test are n - 1, which is 25 - 1 = 24.

**5. Find the Critical Value:**

Using a chi-square table or calculator, find the critical chi-square value for df = 24 and α = 0.05. Since this is a two-tailed test (Ha: σ ≠ 10.5), we need to look up the critical values for both tails (α/2 = 0.025 in each tail).

* The critical chi-square values are approximately **12.40** (lower tail) and **39.36** (upper tail).

**6. Make a Decision:**

Our calculated chi-square value (16.33) falls between the lower and upper critical values (12.40 and 39.36). This means our test statistic is **not** in the rejection region.

**7. Conclusion:**

Since the calculated chi-square value is not in the rejection region, we **fail to reject the null hypothesis**. There is not enough evidence to conclude that the population standard deviation is different from 10.5 at the 0.05 significance level.

**In other words:** Based on the sample data, we do not have sufficient evidence to say that the population standard deviation is not 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.**

Here's how to perform the chi-square test for uniformity:

**1. State the Hypotheses:**

* **Null Hypothesis (H0):** The distribution of grades is uniform (each grade has an equal expected frequency).
* **Alternative Hypothesis (Ha):** The distribution of grades is not uniform.

**2. Set the Significance Level (α):**

Let's use a common significance level of α = 0.05.

**3. Calculate Expected Frequencies:**

Under a uniform distribution, with 5 grades and 100 students, the expected frequency for each grade is:

* Expected Frequency = Total Students / Number of Grades = 100 / 5 = 20

**4. Calculate the Chi-Square Test Statistic (χ²):**

χ² = Σ [(Observed Frequency - Expected Frequency)² / Expected Frequency]

| Grade | Observed Frequency (O) | Expected Frequency (E) | (O - E)² | (O - E)² / E |
| --- | --- | --- | --- | --- |
| A | 15 | 20 | 25 | 1.25 |
| B | 17 | 20 | 9 | 0.45 |
| C | 30 | 20 | 100 | 5.00 |
| D | 22 | 20 | 4 | 0.20 |
| E | 16 | 20 | 16 | 0.80 |
| **Total** | **100** | **100** |  | **7.70** |

Therefore, χ² = 7.70

**5. Determine Degrees of Freedom (df):**

* df = Number of Categories - 1 = 5 - 1 = 4

**6. Find the Critical Value:**

Using a chi-square table or calculator with df = 4 and α = 0.05, the critical chi-square value is approximately 9.49.

**7. Make a Decision:**

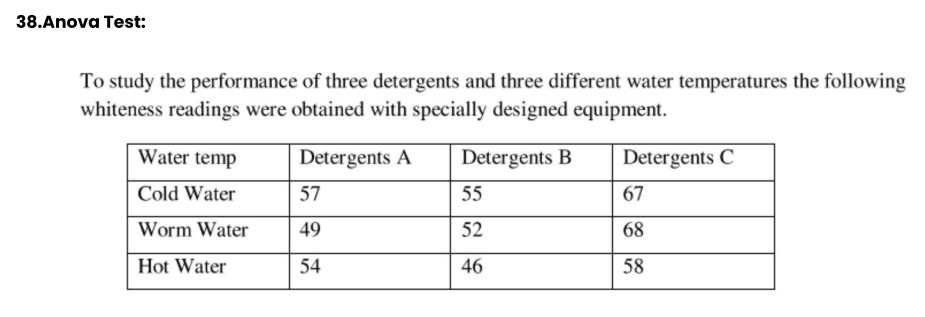
* Our calculated χ² (7.70) is **less than** the critical value (9.49).

**8. Conclusion:**

Since the calculated χ² value is less than the critical value, we **fail to reject the null hypothesis**.

**Interpretation:**

We do not have enough evidence to conclude that the distribution of grades in the Data Science paper is significantly different from a uniform distribution at the 0.05 significance level. The observed differences in grade frequencies could likely be due to random chance.



### **Data Summary**

| **Water Temperature** | **Detergent A** | **Detergent B** | **Detergent C** |
| --- | --- | --- | --- |
| Cold Water | 57 | 55 | 67 |
| Warm Water | 49 | 52 | 68 |
| Hot Water | 54 | 46 | 58 |

### 

### 

### 

### **Step-by-Step ANOVA Calculation**

1. **Calculate the Means**:
   * Mean for Detergent A: ( \frac{57 + 49 + 54}{3} = 53.33 )
   * Mean for Detergent B: ( \frac{55 + 52 + 46}{3} = 51 )
   * Mean for Detergent C: ( \frac{67 + 68 + 58}{3} = 64.33 )
   * Overall Mean: ( \frac{57 + 55 + 67 + 49 + 52 + 68 + 54 + 46 + 58}{9} = 56.22 )
2. **Calculate the Sum of Squares**:
   * **Total Sum of Squares (SST)**:
   * SST = \sum (X\_{ij} - \text{Overall Mean})^2
   * where ( X\_{ij} ) are the individual data points.
   * **Sum of Squares Between Groups (SSB)**:
   * SSB = \sum n\_i (\text{Group Mean} - \text{Overall Mean})^2
   * where ( n\_i ) is the number of observations in each group.
   * **Sum of Squares Within Groups (SSW)**:
   * SSW = \sum (\text{Individual Value} - \text{Group Mean})^2
3. **Calculate the Mean Squares**:
   * Mean Square Between (MSB):
   * MSB = \frac{SSB}{df\_{between}}
   * where ( df\_{between} = k - 1 ) (k is the number of groups).
   * Mean Square Within (MSW):
   * MSW = \frac{SSW}{df\_{within}}
   * where ( df\_{within} = N - k ) (N is the total number of observations).
4. **Calculate the F-statistic**:
5. F = \frac{MSB}{MSW}

### **Using Software for Calculation**

Given the complexity of manual calculations, it’s often easier to use statistical software like Excel, R, or Python. Here’s a quick example using Python:

import pandas as pd

import statsmodels.api as sm

from statsmodels.formula.api import ols

# Data

data = {'Detergent': ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C'],

'Temperature': ['Cold', 'Warm', 'Hot', 'Cold', 'Warm', 'Hot', 'Cold', 'Warm', 'Hot'],

'Whiteness': [57, 49, 54, 55, 52, 46, 67, 68, 58]}

df = pd.DataFrame(data)

# ANOVA

model = ols('Whiteness ~ C(Detergent) + C(Temperature)', data=df).fit()

anova\_table = sm.stats.anova\_lm(model, typ=2)

print(anova\_table)

**( Questions no: 38 to 50 are uploaded on GitHub )**