RAMANUJAN COLLEGE

University Of Delhi

DATA PRIVACY ASSIGNMENT

Submitted to:- Dr. Ashmita Patel Mam

Name:- Ravikant Maurya

Course:- B.Sc.(Hons.) Computer Science

Exam Roll No. :- 20220570026

College Roll No.:- 20221433

Q.1:- Write a program to perform encryption and decryption using Caesar cipher (substitutional cipher).

Explanation:-

The Caesar cipher is a simple substitution cipher named after Julius Caesar, who reportedly used it for secure communication. It involves **shifting the letters of the alphabet** by a fixed number of places.

- ***Encryption:*** Transforming the original message (plaintext) into a coded message (ciphertext) by shifting each letter.
- ***Decryption:*** Reversing the process by shifting the letters back to their original positions.

Key Concept:-

The Caesar cipher uses a key (the shift value) to determine how far each letter should be moved. For example:

- A shift of **3** means:
  - A → D, B → E, C → F, and so on.

If we reach the end of the alphabet, it wraps around. For example:

X → A, Y → B, Z → C.

Algorithm for Caesar Cipher

Encryption:-

1. Choose a key (e.g., 3).
2. Replace each letter in the plaintext with the letter located **key positions ahead** in the alphabet.
3. Leave non-alphabetic characters (like spaces or punctuation) unchanged.

Decryption:-

1. Use the same key.
2. Replace each letter in the ciphertext with the letter located **key positions**
   Encryption:

Limitations of Caesar Cipher

1. **Vulnerable to brute-force attacks**: Since there are only 25 possible shifts, it's easy to break by trying all shifts.
2. **No complexity**: Patterns in the plaintext are preserved, making it easier to analyze using frequency analysis.

Code:-

```
# Function to encrypt the plaintext
def encrypt(plaintext, shift):
    """
    Encrypts the given plaintext using a Caesar cipher with the specified shift.
    Parameters:
        - plaintext: The text to be encrypted.
        - shift: The number of positions each character should be shifted.
    Returns:
        - encrypted_text: The resulting encrypted text.
    """
    encrypted_text = ""
    for char in plaintext:
        if char.isalpha():  # Check if the character is a letter
            # Determine the base ASCII value (uppercase: 65, lowercase: 97)
            shift_base = 65 if char.isupper() else 97
```

```
            # Perform the Caesar cipher shift within the alphabet range
            encrypted_text += chr((ord(char) - shift_base + shift) % 26 +
shift_base)
        else:
            # If not a letter, keep the character unchanged (e.g., spaces,
punctuation)
            encrypted_text += char
    return encrypted_text

# Function to decrypt the ciphertext
def decrypt(ciphertext, shift):
    """
    Decrypts the given ciphertext using a Caesar cipher with the specified shift.
    Parameters:
        - ciphertext: The encrypted text to be decrypted.
        - shift: The number of positions each character was shifted during
encryption.
    Returns:
        - decrypted_text: The resulting decrypted text.
    """
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():  # Check if the character is a letter
            # Determine the base ASCII value (uppercase: 65, lowercase: 97)
            shift_base = 65 if char.isupper() else 97
            # Reverse the Caesar cipher shift to decrypt
            decrypted_text += chr((ord(char) - shift_base - shift) % 26 +
shift_base)
        else:
            # If not a letter, keep the character unchanged
            decrypted_text += char
    return decrypted_text

# Example usage with user input
if __name__ == "__main__":
    # Get user input
    plaintext = input("Enter the text to encrypt: ")  # The text to be encrypted
    shift = int(input("Enter the shift value (integer): "))  # The Caesar cipher
shift value

    # Encrypt the input text
    encrypted = encrypt(plaintext, shift)
    print(f"Encrypted Text: {encrypted}")  # Display the encrypted text

    # Decrypt the text back
```

```
    decrypted = decrypt(encrypted, shift)
    print(f"Decrypted Text: {decrypted}")  # Verify that the decrypted text
matches the original
```

Output:-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE  GITLENS  SEARCH ERROR

PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> python -u "c
:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433\Practical_01.py"
Enter the text to encrypt: Ravikant Maurya
Enter the shift value (integer): 6
Encrypted Text: Xgboqgtz Sgaxeg
Decrypted Text: Ravikant Maurya
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Q.2:- Write a program to perform encryption and decryption using Rail Fence Cipher (transpositional cipher).

Explanation:-

The **Rail Fence Cipher** is a simple transposition cipher that rearranges the letters of plaintext according to a pattern determined by a number of "rails." It is a form of **transpositional cipher** because it reorders the characters rather than substituting them.

How it Works:

1. **Encryption**:
    - The plaintext is written in a zigzag pattern across a fixed number of rails (rows).
    - After completing the zigzag, the characters are read row by row to form the ciphertext.

2. **Decryption**:
    - The ciphertext is split back into rows (rails).
    - These rows are then reconstructed in the zigzag pattern to retrieve the original plaintext.

Steps for Encryption:

1. Choose the number of rails (e.g., 3 rails).
2. Write the plaintext in a zigzag pattern row by row.
3. Read the characters row by row to get the ciphertext.

Steps for Decryption:

1. Determine the length of each rail based on the ciphertext and the zigzag pattern.
2. Place the characters of the ciphertext row by row according to the rail lengths.

Advantages of Rail Fence Cipher:

- Simple to implement and understand.
- Provides basic security by reordering characters.

Disadvantages:

- Easy to break with pattern analysis or brute force since the keyspace (number of rails) is small.
- Not suitable for modern cryptographic needs.

Code:-

```
# Function to encrypt the plaintext using Rail Fence Cipher
def encrypt(plaintext, rails):
    """
    Encrypts the plaintext using the Rail Fence Cipher with the specified number
of rails.
    Parameters:
        - plaintext: The text to encrypt.
        - rails: The number of rails (zigzag rows) in the cipher.
    Returns:
        - encrypted_text: The resulting encrypted text.
```

```
    """
    # Create a 2D list (matrix) to represent the rail fence
    fence = [['' for _ in range(len(plaintext))] for _ in range(rails)]

    # Variables to track the current row and direction of movement (zigzag
pattern)
    row, step = 0, 1

    # Populate the fence with plaintext characters in a zigzag manner
    for col in range(len(plaintext)):
        fence[row][col] = plaintext[col]  # Place character in the current row
and column

        # Change direction at the top or bottom rail
        if row == 0:
            step = 1  # Move down
        elif row == rails - 1:
            step = -1  # Move up

        row += step  # Move to the next row based on the direction

    # Concatenate all characters from the fence row by row to form the encrypted
text
    encrypted_text = ''.join(''.join(row) for row in fence)
    return encrypted_text

# Function to decrypt the ciphertext using Rail Fence Cipher
def decrypt(ciphertext, rails):
    """
    Decrypts the ciphertext encrypted using the Rail Fence Cipher with the
specified number of rails.
    Parameters:
        - ciphertext: The encrypted text to decrypt.
        - rails: The number of rails (zigzag rows) used during encryption.
    Returns:
        - decrypted_text: The resulting decrypted text.
    """
    # Create a 2D list (matrix) to represent the rail fence
    fence = [['' for _ in range(len(ciphertext))] for _ in range(rails)]

    # Variables to track the current row and direction of movement (zigzag
pattern)
    row, step = 0, 1

    # Mark the zigzag positions with a placeholder ('*')
```

```
    for col in range(len(ciphertext)):
        fence[row][col] = '*'

        if row == 0:
            step = 1  # Move down
        elif row == rails - 1:
            step = -1  # Move up

        row += step  # Move to the next row based on the direction

    # Fill the zigzag positions with characters from the ciphertext
    index = 0
    for row in range(rails):
        for col in range(len(ciphertext)):
            if fence[row][col] == '*':  # If marked, place the next ciphertext
character
                fence[row][col] = ciphertext[index]
                index += 1

    # Read the characters in the zigzag pattern to reconstruct the plaintext
    decrypted_text = ''
    row, step = 0, 1
    for col in range(len(ciphertext)):
        decrypted_text += fence[row][col]  # Append character to the decrypted
text

        if row == 0:
            step = 1  # Move down
        elif row == rails - 1:
            step = -1  # Move up

        row += step  # Move to the next row based on the direction

    return decrypted_text

# Example usage with user input
if __name__ == "__main__":
    # Get user input
    plaintext = input("Enter the text to encrypt: ")  # Text to be encrypted
    rails = int(input("Enter the number of rails: "))  # Number of rails (rows)
for the zigzag pattern

    # Encrypt the input text
    encrypted = encrypt(plaintext, rails)
    print(f"Encrypted Text: {encrypted}")  # Display the encrypted text
```

```
    # Decrypt the text back
    decrypted = decrypt(encrypted, rails)
    print(f"Decrypted Text: {decrypted}")  # Verify that the decrypted text
matches the original plaintext
```

Output:-

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya
_20221433> python -u "c:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Priv
acy_Ravikant Maurya_20221433\Practical_02.py"
Enter the text to encrypt: Ravikant Maurya
Enter the number of rails: 7
Encrypted Text: RrauyvaaiMk atn
Decrypted Text: Ravikant Maurya
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya
_20221433>
```

Q.3:- Write a Python program that defines a function and takes a password string as input and returns its SHA-256 hashed representation as a hexadecimal string.

Explanation:-

SHA-256 Hashed Representation in Hexadecimal

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that takes an input (message) and produces a fixed-size, 256-bit (32-byte) output. The hashed output is often represented as a **hexadecimal string** for readability and convenience.

Steps to Obtain SHA-256 Hexadecimal Representation:

1. **Hashing Process**:
    - SHA-256 processes the input data in blocks of 512 bits.

    - The algorithm applies a series of logical operations, bit manipulations, and modular additions to compress the input data into a fixed 256-bit hash value.
2. **Binary to Hexadecimal Conversion**:
    - The 256-bit hash (binary format) is divided into 32 groups of 8 bits.
    - Each 8-bit group is converted to a **2-digit hexadecimal number**.
    - The result is a 64-character hexadecimal string because 32×2=6432 \times 2 = 6432×2=64.

Properties of SHA-256 Hexadecimal String:

- **Fixed Length**: Always 64 characters long, regardless of input size.
- **Unique Representation**: Even a single-bit change in the input drastically changes the hash (known as the **avalanche effect**).
- **Irreversible**: It is computationally infeasible to retrieve the original input from the hash.

Applications:

1. **Password Hashing**: Storing hashed passwords for secure authentication.
2. **Data Integrity**: Verifying that data has not been altered (e.g., digital signatures).
3. **Blockchain**: SHA-256 is widely used in blockchains like Bitcoin for creating hashes of transactions and blocks.

Code:-

```
import hashlib  # Importing the hashlib library for cryptographic hashing

# Function to compute SHA-256 hash of a password
def sha256_hash(password):
    """
    Computes the SHA-256 hash of the given password.
    Parameters:
        - password: The string input that needs to be hashed.
    Returns:
        - The hexadecimal representation of the SHA-256 hash.
    """
    # Create a new sha256 hash object using hashlib
    sha256_hash = hashlib.sha256()
```

```
    # Update the hash object with the password, which is converted to bytes (UTF-
8 encoding)
    # Note: Hashing functions operate on binary data, so the input string must be
encoded.
    sha256_hash.update(password.encode('utf-8'))

    # Compute and return the hexadecimal representation of the hash
    return sha256_hash.hexdigest()

# Example usage with user input
if __name__ == "__main__":
    # Prompt the user to input a password
    password = input("Enter your password: ")

    # Compute the SHA-256 hash of the input password
    hashed_password = sha256_hash(password)

    # Display the hashed password to the user
    print(f"SHA-256 Hash: {hashed_password}")
```

Output:-

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> p
ython -u "c:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_2022
1433\tempCodeRunnerFile.py"
Enter your password: Ravikant@6386
SHA-256 Hash: 3634dbd9eda2add5e01934328fe486432acba78df4b422cd3c3dc0cba6cbdeb6
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Q.4:-Write a Python program that reads a file containing a list of usernames and passwords, one pair per line (separated by a comma). It checks each password to see if it has been leaked in a data breach. You can use the "Have I Been

Pwned"API(https://haveibeenpwned.com/API/v3) to check if a password has been leaked.

Explanation:-

What is Have I Been Pwned?

Have I Been Pwned (HIBP) is a free online service created by security expert **Troy Hunt** in December 2013. The platform helps individuals and organizations determine if their email addresses, usernames, or passwords have been exposed in data breaches.

Key Features of Have I Been Pwned:

1. **Email/Username Check**:
   - Users can enter their email address or username to check if it has been involved in a data breach.
   - The platform checks the input against a database of publicly known breaches and informs the user if their data is compromised.
2. **"Pwned" Meaning**:
   - "Pwned" is internet slang for "owned," meaning that the data has been compromised or leaked in a security breach.
3. **Password Checking**:
   - HIBP includes a feature called **"Pwned Passwords"**, allowing users to check if their password has been exposed in breaches without revealing it (using hashing techniques).
4. **Notification Service**:
   - Users can subscribe to get notifications if their email address is found in future breaches.
5. **API Access**:
   - HIBP provides APIs for developers to integrate breach-checking features into their applications.

How Does It Work?

1. The service aggregates data from publicly disclosed breaches and compiles it into a searchable database.

2. When a user searches their email address or password, HIBP compares the input against this database.
3. It uses secure methods (e.g., **hashing** and **k-anonymity**) to ensure user privacy.

Code:-

```
import hashlib
import requests
import csv

# Function to create a CSV file with usernames and passwords
def create_csv(filename, data):
    """
    Creates a CSV file with the provided username-password data.

    Parameters:
        - filename: Name of the CSV file to create (string).
        - data: List of dictionaries with 'username' and 'password' keys.
    """
    # Define the column headers for the CSV file
    fieldnames = ['username', 'password']

    # Open the CSV file in write mode with UTF-8 encoding and no extra blank
lines between rows
    with open(filename, mode='w', newline='', encoding='utf-8') as csv_file:
        # Create a DictWriter object that will write the dictionaries as rows
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

        # Write the header row to the CSV file (the field names)
        writer.writeheader()

        # Loop through the data list and write each username-password pair to the
CSV
        for row in data:
            writer.writerow(row)

    # Inform the user that the file has been created successfully
    print(f"CSV file '{filename}' created successfully!")

# Example usage
if __name__ == "__main__":
    # Sample data: list of dictionaries with 'username' and 'password' keys
```

```
    users = [
        {"username": "user1", "password": "password123"},
        {"username": "user2", "password": "securepass456"},
        {"username": "user3", "password": "admin@2023"},
        {"username": "user4", "password": "Ravi@2023"},
        {"username": "user5", "password": "Ram@2023"},
        {"username": "user6", "password": "sahil23@2023"},
        {"username": "user7", "password": "ravi@2023"},
        {"username": "user8", "password": "Ravi@123"},
        {"username": "user9", "password": "Ravikant@63806"},
        {"username": "user10", "password": "abc@12345678"},
        {"username": "user11", "password": "abcd@1234"},
        {"username": "user12", "password": "ram@2024"},
        {"username": "user13", "password": "765der@"},
        {"username": "user14", "password": "admin@2023"},
        {"username": "user15", "password": "admin@2023"}
    ]

    # File name for the CSV (this will be the name of the file saved on disk)
    csv_filename = "Generally_used_passwords.csv"

    # Call the function to create the CSV file with the sample data
    create_csv(csv_filename, users)


# Function to check if the password is part of a data breach using "Have I Been
Pwned" API
def check_password_breach(password):
    """
    Checks whether the given password has been part of a known data breach
    using the "Have I Been Pwned" API.

    Parameters:
        - password: The password to check.

    Returns:
        - True if the password has been found in a data breach,
        - False if not, or None if there's an error.
    """
    # Compute the SHA-1 hash of the password
    sha1_hash = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

    # Send the first 5 characters to the API
    first_five = sha1_hash[:5]
    remaining_hash = sha1_hash[5:]
```

```
    # Send GET request to the API with the first 5 characters of the hash
    url = f"https://api.pwnedpasswords.com/range/{first_five}"
    response = requests.get(url)

    # Check if the remaining hash is in the response (indicating a match)
    if response.status_code == 200:
        if remaining_hash in response.text:
            return True  # Password found in breach
        else:
            return False  # Password not found in breach
    else:
        print("Error fetching data from Have I Been Pwned API")
        return None


# Function to check usernames and passwords from a file
def check_file_for_breaches(file_path):
    """
    Reads a CSV file with usernames and passwords, checks each password
    for breaches using the "Have I Been Pwned" API, and prints the results.

    Parameters:
        - file_path: The path to the CSV file containing username-password pairs.
    """
    # Open the CSV file for reading
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip the header row

        # Loop through each row (username and password pair)
        for row in reader:
            username, password = row

            # Check the password against the API
            print(f"Checking password for {username}...")
            if check_password_breach(password):
                print(f"Password for {username} has been pwned!")
            else:
                print(f"Password for {username} has NOT been pwned.")

# Example usage with user input
if __name__ == "__main__":
    # Path to the file containing usernames and passwords (use the file created
earlier)
```

```
file_path = "Generally_used_passwords.csv"

# Check each username and password pair for data breaches
check_file_for_breaches(file_path)
```

Generally_used_passwords.csv:-

Generally_used_passwords.csv

```
username,password
user1,password123
user2,securepass456
user3,admin@2023
user4,Ravi@2023
user5,Ram@2023
user6,sahil23@2023
user7,ravi@2023
user8,Ravi@123
user9,Ravikant@63806
user10,abc@12345678
user11,abcd@1234
user12,ram@2024
user13,765der@
user14,admin@2023
user15,admin@2023
```

Output:-

```
        > python -u "c:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant
 Maurya_20221433\Practical_04.py"
CSV file 'Generally_used_passwords.csv' created successfully!
Checking password for user1...
Password for user1 has been pwned!
Checking password for user2...
Password for user2 has NOT been pwned.
Checking password for user3...
Password for user3 has been pwned!
Checking password for user4...
Password for user4 has been pwned!
Checking password for user5...
Password for user5 has been pwned!
Checking password for user6...
Password for user6 has NOT been pwned.
Checking password for user7...
Password for user7 has been pwned!
Checking password for user8...
Password for user8 has been pwned!
Checking password for user9...
Password for user9 has NOT been pwned.
Checking password for user10...
Password for user10 has been pwned!
Checking password for user11...
Password for user11 has been pwned!
Checking password for user12...
Password for user12 has NOT been pwned.
Checking password for user13...
Password for user13 has NOT been pwned.
Checking password for user14...
Password for user14 has been pwned!
Checking password for user15...
Password for user15 has been pwned!
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Q.5:- Write a Python program that generates a password using a random combination of words from a dictionary file.

Explanation:-

1. **Purpose**:
    - The program generates a secure and memorable password by combining random words from a dictionary file.
2. **Input**:
    - A dictionary file containing a list of words, with one word per line.

- Number of words to use for the password (`num_words`).
- An optional separator (e.g., `-`, `_`, or a space) to combine the words.

3. **Process**:
   - Reads the dictionary file and loads all words into a list, ignoring empty lines and extra spaces.
   - Ensures the dictionary contains enough words for the requested password length.
   - Randomly selects unique words using the `random.sample()` function.
4. **Password Creation**:
   - Combines the selected words into a single string using the specified separator.
5. **Error Handling**:
   - If the dictionary file is missing, it displays an error message.
   - If the dictionary does not have enough words, it raises a `ValueError` with a helpful message.
6. **Output**:
   - The generated password is displayed, e.g., `apple-banana-cherry-grape`.
7. **Usage**:
   - Place the program and dictionary file in the same directory.
   - Customize the number of words and separator as needed.
   - Run the program to generate a secure password.

Code:-

```
import random

# Predefined list of dictionary words (can be extended as needed)
dictionary_words = [
    "apple", "banana", "cherry", "date", "elderberry", "fig", "grape",
"honeydew",
    "kiwi", "lemon", "mango", "nectarine", "orange", "papaya", "quince",
"raspberry",
    "strawberry", "tangerine", "ugli", "vanilla", "watermelon", "xylophone",
"yellow",
    "zucchini", "avocado", "blueberry", "cantaloupe", "dragonfruit", "apricot",
"blackberry",
    "coconut", "plum", "pear", "persimmon", "pomegranate", "jackfruit", "lychee",
"fig",
```

```
    "clementine", "pineapple", "watercress", "spinach", "lettuce", "tomato",
"onion",
    "garlic", "carrot", "broccoli", "cauliflower", "potato", "sweetpotato",
"beet", "asparagus",
    "peas", "artichoke", "celery", "pumpkin", "radish", "cucumber", "ginger",
"chili",
    "cilantro", "oregano", "parsley", "basil", "thyme", "rosemary", "sage",
"tarragon",
    "mint", "dill", "curry", "nutmeg", "cinnamon", "cardamom", "clove",
"paprika",
    "turmeric", "saffron", "vanilla", "chocolate", "coffee", "tea", "latte",
"mocha",
    "espresso", "cappuccino", "macchiato", "lemonade", "lime", "grapefruit",
"orangeade",
    "applejuice", "carrotjuice", "tomatojuice", "gingerale", "soda", "fizz",
"milk",
    "yogurt", "cheese", "butter", "cream", "icecream", "frozenyogurt", "popcorn",
"chips",
    "pretzel", "cookie", "brownie", "cake", "pie", "donut", "muffin", "cupcake",
"pancake",
    "waffle", "toast", "sandwich", "burger", "pizza", "pasta", "noodle",
"spaghetti",
    "lasagna", "meatball", "hotdog", "steak", "chicken", "fish", "salmon",
"tuna",
    "lobster", "shrimp", "crab", "clam", "oyster", "mussels", "sushi", "sashimi"
]

# Function to create a .txt file with dictionary words
def create_dict_txt(filename, words):
    """
    Creates a .txt file with the provided list of words.

    Parameters:
        - filename: The name of the .txt file to create (string).
        - words: List of words to write into the .txt file.
    """
    # Open the file in write mode and write each word from the dictionary list to
it
    with open(filename, 'w') as file:
        for word in words:
            file.write(word + '\n')  # Write each word on a new line

    # Print confirmation message after creating the file
    print(f"File '{filename}' created with dictionary words.")
```

```
# Example usage
if __name__ == "__main__":
    # File name for the .txt file to store the dictionary words
    txt_filename = "dictionary_words.txt"

    # Create the .txt file with the list of dictionary words
    create_dict_txt(txt_filename, dictionary_words)

    # Ask the user for the number of words they want in the password
    # This input will determine how many random words will be chosen for the
password
    num_words = int(input("Enter the number of words you want in your password:
"))

    # Function to generate a password from a dictionary file
    def generate_password(dictionary_file, num_words, separator="-"):
        """
        Generate a random password using words from the dictionary file.

        Parameters:
        - dictionary_file: Path to the text file containing a list of words.
        - num_words: The number of random words to include in the password
(default: 4).
        - separator: The separator between words (default: "-").

        Returns:
        - A string representing the generated password.
        """
        # Read words from the dictionary file
        try:
            # Open the dictionary file and read the words into a list
            with open(dictionary_file, 'r') as file:
                words = [line.strip() for line in file.readlines()]

            # If the dictionary file is empty, raise an exception
            if len(words) == 0:
                raise ValueError("Dictionary file is empty.")

            # Randomly select the specified number of words from the dictionary
            selected_words = random.sample(words, num_words)

            # Join the selected words using the provided separator
            password = separator.join(selected_words)
            return password
```

```
        except FileNotFoundError:
            print(f"Error: The file '{dictionary_file}' was not found.")
        except Exception as e:
            print(f"An error occurred: {e}")

    # Generate the password using the provided dictionary file and number of
words
    dictionary_file = "dictionary_words.txt"  # Replace with the actual
dictionary file path
    password = generate_password(dictionary_file, num_words, separator="-")

    # If a password was generated, print it
    if password:
        print(f"Generated Password: {password}")
```

Dictionary file:-

dictionary_words.txt

```
plum
pear
persimmon
pomegranate
jackfruit
lychee
fig
clementine
pineapple
watercress
spinach
lettuce
tomato
onion
garlic
carrot
broccoli
cauliflower
```

```
potato
sweetpotato
beet
asparagus
peas
artichoke
celery
pumpkin
radish
cucumber
ginger
chili
cilantro
oregano
parsley
basil
thyme
rosemary
sage
tarragon
mint
```

Output:-

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20
3\Practical_05.py"
File 'dictionary_words.txt' created with dictionary words.
Enter the number of words you want in your password: 7
Generated Password: cookie-raspberry-cream-radish-brownie-vanilla-pear
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Q.6:- Write a Python program that simulates a brute-force attack on a password by trying out all possible character combinations.

Explanation:-

1. **Purpose**:
   - The program demonstrates a brute-force attack by systematically guessing a password through all possible character combinations.
2. **Character Set**:
   - The character set includes lowercase letters (`a-z`), uppercase letters (`A-Z`), digits (`0-9`), and special characters (e.g., `!@#$`).
   - This ensures that all common password components are considered.
3. **Password Length**:
   - The program attempts passwords starting from length 1 up to the specified maximum length (`max_length`).
4. **Combination Generation**:
   - It uses `itertools.product()` to generate all possible combinations of the character set for a given length.
   - Each combination is converted into a string and treated as a password guess.
5. **Password Matching**:
   - For each generated password, the program checks if it matches the target password.
   - If a match is found, it stops and returns the guessed password.
6. **Output**:

- If the password is guessed, it displays the guessed password.
- If no match is found within the `max_length` limit, it indicates failure.

7. **Performance**:
   - Brute-force attacks are computationally expensive. The time taken increases exponentially with the password length and character set size.
8. **Usage**:
   - Replace `target_password` with the password to guess and set `max_length` to the maximum allowed password length.
   - Run the script to see the brute-force process in action.

Code:-

```
import itertools
import string
import time

# Function to perform brute-force attack on the given password
def brute_force_attack(target_password):
    # Define the character set to use for the attack:
    # This includes lowercase letters, uppercase letters, digits, and punctuation
    charset = string.ascii_letters + string.digits + string.punctuation

    # Start a timer to measure how long the attack takes
    start_time = time.time()

    # Iterate over all possible lengths for the password (from 1 to the length of
the target password)
    for length in range(1, len(target_password) + 1):
        # Generate all possible combinations of characters of the given length
        for attempt in itertools.product(charset, repeat=length):
            # Convert the tuple (which is generated by itertools.product) to a
string
            attempt_password = ''.join(attempt)

            # Check if the generated password matches the target password
            if attempt_password == target_password:
                # Calculate the elapsed time for the brute-force attack
                elapsed_time = time.time() - start_time
                # Output the found password and the time it took to find it
                print(f"Password found: {attempt_password}")
```

```
                print(f"Time taken: {elapsed_time:.2f} seconds")
                return  # Exit the function since the password has been found

    # If no match was found, print that the password could not be cracked
    print("Password not found.")

# Example usage
if __name__ == "__main__":
    # Take the target password as user input
    target_password = input("Enter the password to simulate brute-force attack:
")

    # Notify user that the brute-force attack is starting
    print("Starting brute-force attack...")
    # Call the brute-force attack function to try and guess the password
    brute_force_attack(target_password)
```

Output:-

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> python -u "c:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_2022143
3\Practical_06.py"
Enter the password to simulate brute-force attack: p123
Starting brute-force attack...
Password found: p123
Time taken: 3.03 seconds
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Q.7:- Demonstrate the usage/sending of a digitally signed document.

Explanation:-

Demonstration of Sending a Digitally Signed Document

Digitally signing a document involves using cryptographic methods to ensure authenticity, integrity, and non-repudiation. Here's a step-by-step explanation:

Step 1: Create the Document

- The sender creates or prepares the document to be sent. This could be a PDF, text file, or any other type of file.

Step 2: Generate a Key Pair

1. The sender generates a public-private key pair using cryptographic software like OpenSSL or libraries such as `cryptography` in Python.
    - **Private Key**: Used to sign the document.
    - **Public Key**: Shared with the recipient to verify the signature.

Step 3: Create a Hash of the Document

1. The sender uses a cryptographic hash function (e.g., SHA-256) to compute a unique hash of the document.
2. The hash represents the content of the document and changes if the document is altered.

Step 4: Sign the Hash

1. The sender encrypts the hash using their private key to create the digital signature.
2. The encrypted hash is unique to the document and the sender's private key.

Step 5: Attach the Digital Signature

- The sender attaches the digital signature to the document, typically alongside the original document, or as a separate file.

Step 6: Send the Document

- The sender transmits the document and the digital signature to the recipient via email, file-sharing services, or other means.

Step 7: Verify the Signature

1. The recipient uses the sender's public key to decrypt the digital signature.
2. The recipient computes a hash of the received document.
3. If the decrypted signature hash matches the hash of the received document:
    - **Authenticity**: The document came from the sender.
    - **Integrity**: The document was not altered during transmission.

Key Points

1. **Digital Signature**: Provides authenticity and integrity by signing the document hash with a private key.
2. **Hash Function**: Ensures that even minor changes in the document will invalidate the signature.
3. **Public Key**: Used by the recipient to verify the authenticity of the signature.
4. **Applications**:
    - Signing legal documents.
    - Verifying software downloads.
    - Ensuring email authenticity (e.g., S/MIME).

This approach secures document exchange and prevents tampering

Code:-

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes

# Generate a pair of private and public keys
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
public_key = private_key.public_key()

# The message/document to be signed
document = b"This is a sensitive document."

# Sign the document
signature = private_key.sign(
    document,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
```

```
)

print("Document signed successfully.")

# Verify the signature
try:
    public_key.verify(
        signature,
        document,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    print("Signature verified successfully.")
except Exception as e:
    print("Verification failed:", e)
```

Output:-

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> python -u "c:\Users
Privacy_Ravikant Maurya_20221433\Practical_07.py"
Document signed successfully.
Signature verified successfully.
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Q.8:- Students needs to conduct a data privacy audit of an organization to identify potential vulnerabilities and risks in their data privacy practices.

Explanation:-

Define Audit Categories and Questions

- `audit_criteria`: A dictionary containing categories (e.g., `data_collection`, `data_storage`) and a list of questions

under each category. These questions assess data privacy practices within the organization.

3. Collect User Responses

- `collect_responses()`: This function:
  - Prompts the user to answer audit questions for each category.
  - Ensures responses are only "Yes" or "No" (with input validation).
  - Stores the responses in a `responses` dictionary where each category has its own list of answers.
- Example:
  - **Category**: `data_collection`
  - **Question**: "Are users informed about data collection?" → Response: "Yes"

4. Analyze Responses

- `analyze_responses()`: This function:
  - Goes through all the responses.
  - Identifies any questions that were answered with "No", which could indicate vulnerabilities or gaps in the organization's data privacy practices.
  - Creates a `vulnerabilities` dictionary to list which specific questions in each category had "No" answers.

5. Generate a Report

- `generate_report(vulnerabilities)`: This function:
  - Generates a detailed JSON report with the timestamped filename (e.g., `data_privacy_audit_report_2024-11-26_10-30-00.json`).
  - The report contains:
    - `timestamp`: The exact time the audit was conducted.

        - `audit_results`: The user's responses to the questions.
        - `vulnerabilities`: The identified issues (questions with "No" responses).
    - Saves the report as a JSON file.

6. Display Summary of Findings

- `display_summary(vulnerabilities)`: This function:
    - Summarizes the audit findings.
    - Displays how many vulnerabilities were found in each category and lists the specific questions that were flagged as issue.

Summary of Key Functions:

- **`collect_responses()`**: Collects answers to the audit questions.
- **`analyze_responses()`**: Analyzes the responses and identifies vulnerabilities.
- **`generate_report()`**: Creates a detailed JSON report of the audit.
- **`display_summary()`**: Prints a summary of the findings, including flagged issues.

Code:-

```
import os
import json
import datetime

# Define audit categories with a list of questions for each category
# These categories help assess different aspects of data privacy compliance.
audit_criteria = {
    "data_collection": [
        "Are users informed about data collection?",
        "Is data collection limited to what's necessary?",
        "Are consent mechanisms in place?"
```

```
    ],
    "data_storage": [
        "Is data encrypted at rest?",
        "Is sensitive data stored securely?",
        "Are backup policies in place?"
    ],
    "data_access": [
        "Is access to data restricted based on roles?",
        "Are access logs maintained and monitored?",
        "Are strong authentication mechanisms used?"
    ],
    "compliance": [
        "Is the organization GDPR compliant?",
        "Are data retention policies clearly defined?",
        "Is there a process for handling data subject requests?"
    ]
}

# Dictionary to store the user responses for each audit question
responses = {}

def collect_responses():
    """
    Prompts the user to answer audit questions for each category.
    Stores responses (Yes/No) in the `responses` dictionary.
    """
    print("Starting Data Privacy Audit...\n")
    for category, questions in audit_criteria.items():
        print(f"Category: {category.upper()}")
        category_responses = []
        for question in questions:
            # Validate user input to ensure it is either "Yes" or "No"
            response = input(f" - {question} (Yes/No): ").strip().lower()
            while response not in ["yes", "no"]:
                print("Please enter 'Yes' or 'No'.")
                response = input(f" - {question} (Yes/No): ").strip().lower()
            category_responses.append({"question": question, "response":
response})
        responses[category] = category_responses
        print("\n")

def analyze_responses():
    """
    Analyzes the collected responses to identify vulnerabilities.
    Returns a dictionary containing the categories and their associated issues.
```

```
    """
    print("\nAnalyzing Responses...\n")
    vulnerabilities = {}
    for category, answers in responses.items():
        # Identify questions where the response was "No" as vulnerabilities
        category_vulnerabilities = [item["question"] for item in answers if
item["response"] == "no"]
        vulnerabilities[category] = category_vulnerabilities

    return vulnerabilities

def generate_report(vulnerabilities):
    """
    Generates a JSON report with audit results and identified vulnerabilities.
    Saves the report to a file with a timestamped filename.
    """
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    report_filename = f"data_privacy_audit_report_{timestamp}.json"

    report_content = {
        "timestamp": timestamp,
        "audit_results": responses,
        "vulnerabilities": vulnerabilities
    }

    with open(report_filename, "w") as report_file:
        json.dump(report_content, report_file, indent=4)

    print(f"\nAudit report generated: {report_filename}")
    return report_filename

def display_summary(vulnerabilities):
    """
    Displays a summary of the findings, including the number of issues identified
    in each category and the specific questions flagged as vulnerabilities.
    """
    print("\nSummary of Findings:")
    for category, issues in vulnerabilities.items():
        print(f" - {category.upper()}: {len(issues)} issues identified.")
        for issue in issues:
            print(f"    * {issue}")

# Main script execution
if __name__ == "__main__":
    collect_responses()  # Collect responses from the user
```

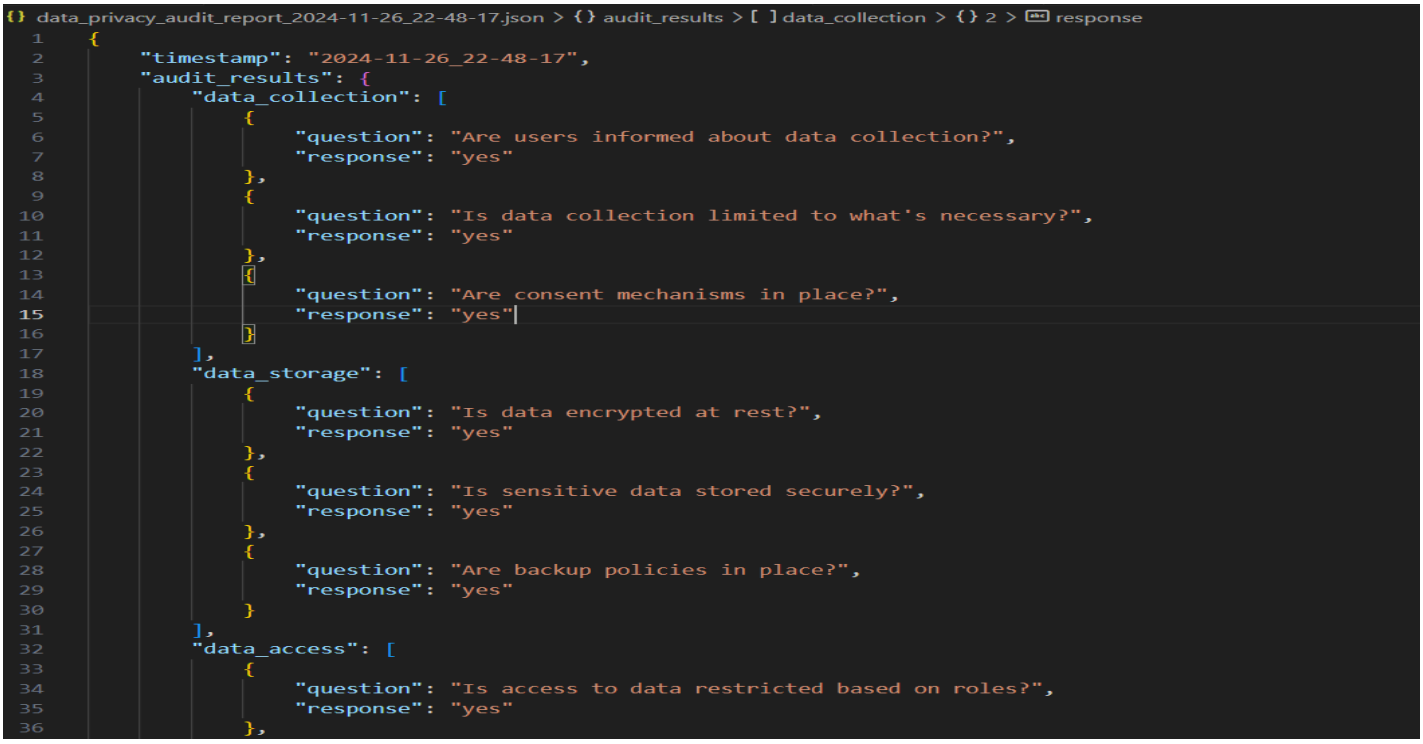
```
    vulnerabilities = analyze_responses()  # Analyze responses for
vulnerabilities
    display_summary(vulnerabilities)  # Display a summary of findings
    generate_report(vulnerabilities) # Generate a detailed report in JSON format
```

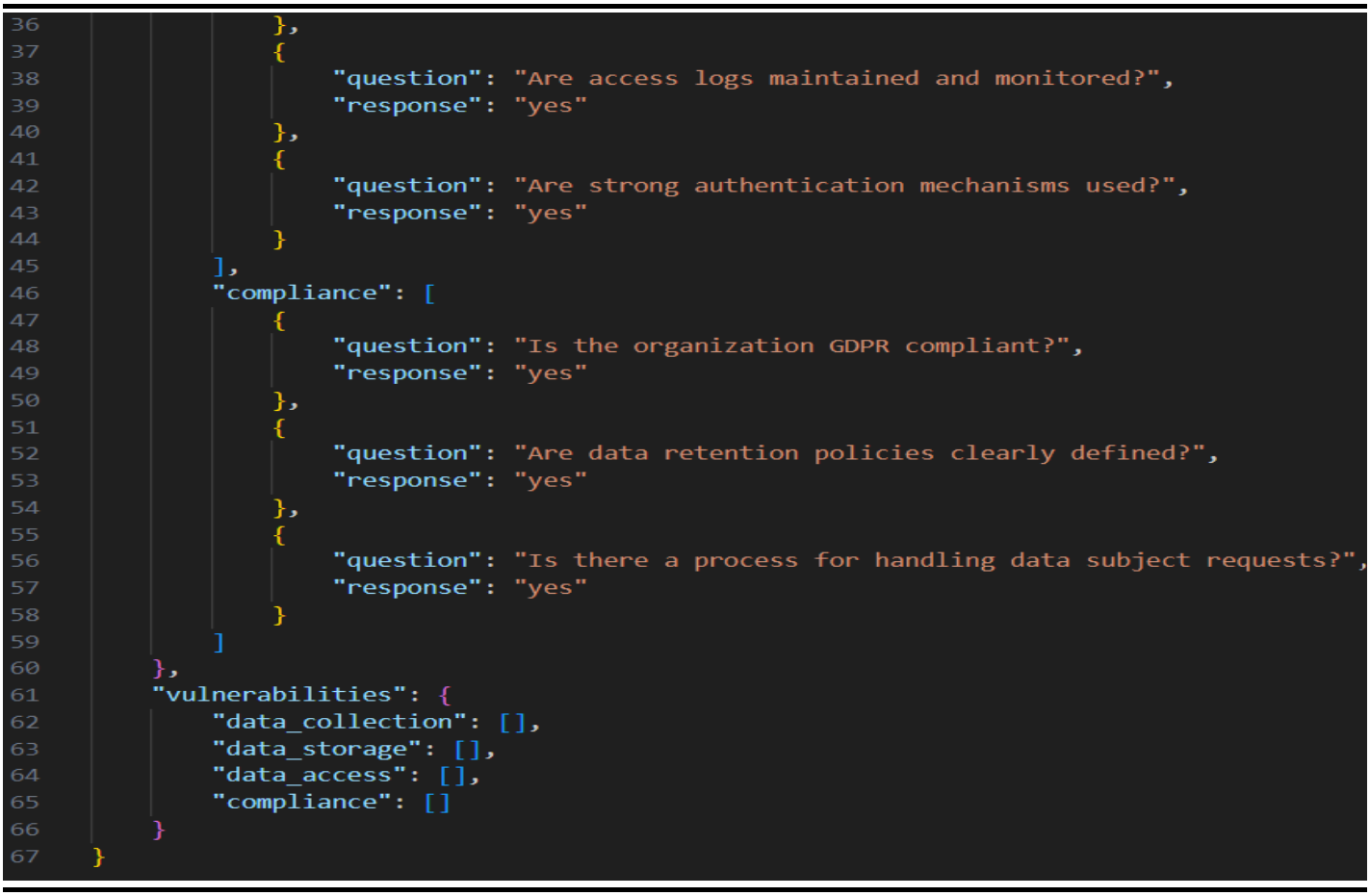
Output:-

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> python -u "c:\Users\RAVIKANT7229\OneDrive\Desktop\Ass
Privacy_Ravikant Maurya_20221433\tempCodeRunnerFile.py"
Starting Data Privacy Audit...

Category: DATA_COLLECTION
 - Are users informed about data collection? (Yes/No): yes
 - Is data collection limited to what's necessary? (Yes/No): yes
 - Are consent mechanisms in place? (Yes/No): yes


Category: DATA_STORAGE
 - Is data encrypted at rest? (Yes/No): yes
 - Is sensitive data stored securely? (Yes/No): yes
 - Are backup policies in place? (Yes/No): yes


Category: DATA_ACCESS
 - Is access to data restricted based on roles? (Yes/No): yes
 - Are access logs maintained and monitored? (Yes/No): yes
 - Are strong authentication mechanisms used? (Yes/No): yes


Category: COMPLIANCE
 - Is the organization GDPR compliant? (Yes/No): yes
 - Are data retention policies clearly defined? (Yes/No): yes
 - Is there a process for handling data subject requests? (Yes/No): yes
```

```
Category: COMPLIANCE
  - Is the organization GDPR compliant? (Yes/No): yes
  - Are data retention policies clearly defined? (Yes/No): yes
  - Is there a process for handling data subject requests? (Yes/No): yes



Analyzing Responses...


Summary of Findings:
  - DATA_COLLECTION: 0 issues identified.
  - DATA_STORAGE: 0 issues identified.
  - DATA_ACCESS: 0 issues identified.
  - COMPLIANCE: 0 issues identified.

Audit report generated: data_privacy_audit_report_2024-11-26_22-32-32.json
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Audit Report:-


{} data_privacy_audit_report_2024-11-26_22-48-17.json > {} audit_results > [ ] data_collection > {} 2 > abc response
{
"timestamp": "2024-11-26_22-48-17",
"audit_results": {
"data_collection": [
{
"question": "Are users informed about data collection?",
"response": "yes"
},
{
"question": "Is data collection limited to what's necessary?",
"response": "yes"
},
{
"question": "Are consent mechanisms in place?",
"response": "yes"
}
],
"data_storage": [
{
"question": "Is data encrypted at rest?",
"response": "yes"
},
{
"question": "Is sensitive data stored securely?",
"response": "yes"
},
{
"question": "Are backup policies in place?",
"response": "yes"
}
],
"data_access": [
{
"question": "Is access to data restricted based on roles?",
"response": "yes"
},


{
"question": "Are access logs maintained and monitored?",
"response": "yes"
},
{
"question": "Are strong authentication mechanisms used?",
"response": "yes"
}
],
"compliance": [
{
"question": "Is the organization GDPR compliant?",
"response": "yes"
},
{
"question": "Are data retention policies clearly defined?",
"response": "yes"
},
{
"question": "Is there a process for handling data subject requests?",
"response": "yes"
}
]
},
"vulnerabilities": {
"data_collection": [],
"data_storage": [],
"data_access": [],
"compliance": []
}
}

Q.9:- Students needs to explore the requirements of the Data Protection Regulations and develop a plan for ensuring compliance with the regulation.

Explanation:-

1. Regulation Data Definition:

- A dictionary called `regulation_requirements` is defined for two data protection regulations: **GDPR** and **HIPAA**.
- For each regulation, specific categories (e.g., **data_processing**, **data_security**) are provided, listing actions or requirements that organizations must adhere to for compliance.

2. Compliance Plan Development:

- **Function `develop_compliance_plan`**:
  - This function takes the regulation name (like "GDPR") and a list of selected categories (e.g., `data_processing`, `data_security`) as input.
  - It checks whether the regulation exists in the `regulation_requirements` dictionary.
  - Based on the user's input, it collects and returns a compliance plan containing the selected categories and their respective actions.

3. Compliance Report Generation:

- **Function `generate_report`**:
  - This function generates a compliance report by saving the selected compliance plan in a JSON file.
  - The report file is named with a timestamp to ensure uniqueness (e.g., `compliance_plan_GDPR_2024-11-25_14-32-00.json`).
  - The report contains the regulation name, timestamp, and the compliance plan.

4. Plan Display and Report Generation:

- The selected compliance plan is generated using the `develop_compliance_plan` function.
- The compliance plan is then displayed, showing the categories and corresponding actions or requirements.
- If the plan is valid (i.e., the user selected valid categories), the compliance report is generated and saved in a JSON file using the `generate_report` function.

Code:-

```
import json
from datetime import datetime

# Define data protection regulation requirements
# Each regulation (e.g., GDPR, HIPAA) has a dictionary of categories and specific
requirements/actions.
regulation_requirements = {
    "GDPR": {  # General Data Protection Regulation
        "data_processing": [
            "Ensure lawful, fair, and transparent processing of personal data.",
            "Obtain explicit consent from data subjects.",
            "Provide data subjects with access to their data and the right to
correct or delete it."
        ],
        "data_security": [
            "Implement appropriate technical and organizational measures.",
            "Ensure encryption and pseudonymization of data.",
            "Maintain data integrity and confidentiality."
        ],
        "compliance_monitoring": [
            "Conduct regular data protection impact assessments (DPIA).",
            "Maintain records of processing activities.",
            "Appoint a Data Protection Officer (DPO) if required."
        ]
    },
    "HIPAA": {  # Health Insurance Portability and Accountability Act
        "privacy_rule": [
            "Ensure protected health information (PHI) is safeguarded.",
            "Provide patients with rights over their PHI.",
            "Limit disclosures of PHI to the minimum necessary."
        ],
```

```
        "security_rule": [
            "Implement administrative safeguards (e.g., training, risk
analysis).",
            "Establish physical safeguards (e.g., facility access controls).",
            "Use technical safeguards (e.g., encryption, access control)."
        ],
        "breach_notification_rule": [
            "Notify affected individuals within 60 days of discovering a
breach.",
            "Report breaches affecting more than 500 individuals to the
Department of Health and Human Services."
        ]
    }
}

# Function to develop a compliance plan
def develop_compliance_plan(regulation, selected_requirements):
    """
    Generates a compliance plan based on the selected categories for a specific
regulation.
    Parameters:
        - regulation: The name of the regulation (e.g., GDPR, HIPAA).
        - selected_requirements: List of selected categories to include in the
plan.
    Returns:
        - A dictionary containing the compliance plan for the selected
categories.
    """
    if regulation not in regulation_requirements:
        print(f"Regulation '{regulation}' not recognized.")
        return

    print(f"\nDeveloping Compliance Plan for {regulation}...\n")
    selected_plan = {}
    for category, requirements in regulation_requirements[regulation].items():
        # Add only the categories selected by the user to the plan
        if category in selected_requirements:
            selected_plan[category] = requirements

    return selected_plan

# Function to generate a compliance plan report
def generate_report(regulation, compliance_plan):
    """
    Saves the compliance plan to a JSON file with a timestamped filename.
```

```
    Parameters:
        - regulation: The name of the regulation (e.g., GDPR, HIPAA).
        - compliance_plan: The generated compliance plan dictionary.
    Returns:
        - The filename of the generated report.
    """
    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    report_filename = f"compliance_plan_{regulation}_{timestamp}.json"

    # Create a report containing the timestamp, regulation, and the compliance
plan
    report_content = {
        "timestamp": timestamp,
        "regulation": regulation,
        "compliance_plan": compliance_plan
    }

    # Write the report to a JSON file
    with open(report_filename, "w") as report_file:
        json.dump(report_content, report_file, indent=4)

    print(f"\nCompliance plan report generated: {report_filename}")
    return report_filename

# Main execution
if __name__ == "__main__":
    # Display available regulations
    print("Available Regulations:")
    for regulation in regulation_requirements.keys():
        print(f" - {regulation}")

    # Prompt the user to select a regulation
    regulation = input("\nEnter the regulation to comply with (e.g., GDPR,
HIPAA): ").strip().upper()

    if regulation in regulation_requirements:
        # Display categories for the selected regulation
        print(f"\nCategories for {regulation}:")
        for category in regulation_requirements[regulation].keys():
            print(f" - {category}")

        # Prompt the user to select categories for the compliance plan
        selected_categories = input(
            "\nEnter the categories to include in the compliance plan (comma-
separated): "
```

```
        ).strip().split(",")

        # Normalize user input (strip spaces and convert to lowercase for
comparison)
        selected_categories = [cat.strip().lower() for cat in
selected_categories]

        # Generate the compliance plan for the selected categories
        compliance_plan = develop_compliance_plan(regulation,
selected_categories)

        if compliance_plan:
            # Display the compliance plan
            print("\nCompliance Plan:")
            for category, actions in compliance_plan.items():
                print(f" - {category.capitalize()}:")
                for action in actions:
                    print(f"   * {action}")

            # Generate a report for the compliance plan
            generate_report(regulation, compliance_plan)
    else:
        # Display an error message if the regulation is not recognized
        print(f"Regulation '{regulation}' is not supported.")
```

Output:-

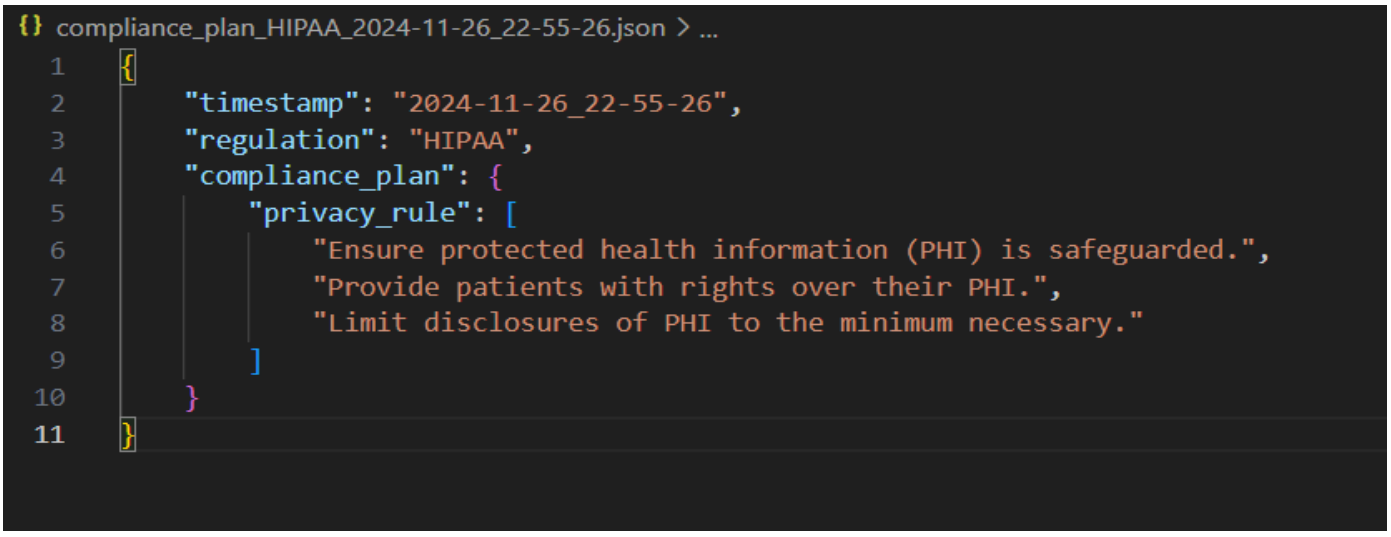
```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> py
 - data_processing
 - data_security
 - compliance_monitoring

Enter the categories to include in the compliance plan (comma-separated): data_security

Developing Compliance Plan for GDPR...


Compliance Plan:
 - Data_security:
   * Implement appropriate technical and organizational measures.
   * Ensure encryption and pseudonymization of data.
   * Maintain data integrity and confidentiality.

Compliance plan report generated: compliance_plan_GDPR_2024-11-26_22-54-38.json
```

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> python -u "c:\Users
ical_9.py"
Available Regulations:
 - GDPR
 - HIPAA

Enter the regulation to comply with (e.g., GDPR, HIPAA): HIPAA

Categories for HIPAA:
 - privacy_rule
 - security_rule
 - breach_notification_rule

Enter the categories to include in the compliance plan (comma-separated): privacy_rule

Developing Compliance Plan for HIPAA...


Compliance Plan:
 - Privacy_rule:
   * Ensure protected health information (PHI) is safeguarded.
   * Provide patients with rights over their PHI.
   * Limit disclosures of PHI to the minimum necessary.

Compliance plan report generated: compliance_plan_HIPAA_2024-11-26_22-55-26.json
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```

Compliance plan report:-

compliance_plan_GDPR_2024-11-26_22-54-38.json


```json
{
    "timestamp": "2024-11-26_22-54-38",
    "regulation": "GDPR",
    "compliance_plan": {
        "data_security": [
            "Implement appropriate technical and organizational measures.",
            "Ensure encryption and pseudonymization of data.",
            "Maintain data integrity and confidentiality."
        ]
    }
}
```



Q.10:- Students needs to explore ethical considerations in data privacy, such as the balance between privacy and security, the impact of data collection and analysis on marginalized communities,and the role of data ethics in technology development.

Explanation:-

1. Defining Ethical Considerations (ethical_topics)

- **Purpose:** The `ethical_topics` dictionary defines different categories of data ethics and the questions related to each category. Each category includes a list of relevant ethical questions.
- **Categories:**
  - **Privacy vs Security:** Questions regarding the balance between user privacy and organizational security needs.
  - **Impact on Marginalized Communities:** Focuses on the disproportionate effect data practices might have on underrepresented groups.

    - **Role of Data Ethics in Technology Development:** Discusses the ethical principles in the design and deployment of technologies like AI and machine learning.

2. `collect_responses()` Function

- **Purpose:** This function is responsible for collecting the user's responses to the ethical questions.
- **Steps:**
    - Prints a header message indicating the start of the process ("Exploring Ethical Considerations in Data Privacy").
    - Iterates through each topic and its associated questions.
    - For each question, it prompts the user to input a response, which is then stored as a dictionary of questions and responses.
    - After collecting all responses for a topic, a separator line is printed for better readability.
- **Returns:** A dictionary called `responses` where each key is a topic, and the value is a list of dictionaries containing the questions and user-provided responses.

3. `generate_report()` Function

- **Purpose:** This function generates a JSON file to store the collected responses and generates a timestamped report file.
- **Steps:**
    - Creates a unique filename for the report using the current timestamp (to avoid overwriting files).
    - Opens the generated file in write mode and uses `json.dump()` to save the responses in a structured JSON format.
    - The `indent=4` argument ensures that the JSON is nicely formatted with indents for readability.
- **Returns:** The filename of the generated JSON report.

4. `display_summary()` Function

- **Purpose:** This function provides a user-friendly summary of all collected responses.
- **Steps:**
    - Prints a summary heading for ethical considerations.

    - Iterates through the collected responses and prints each topic followed by its associated questions and user responses.
    - After printing all responses, it adds a separator for visual clarity.
- **Display:** This function is mainly for displaying the responses on the console in a readable format.

Key Concepts:

- **User Interaction:** The script interacts with the user by asking ethical questions and collecting responses via `input()`.
- **Data Storage:** Responses are stored in a structured way using a dictionary and are saved in a JSON file for later reference.
- **Timestamping:** The generated report filename includes a timestamp to ensure each report has a unique name.
- **JSON Format:** The responses are saved in JSON format, making it easy to read, share, or process later.

Code:-

```
import json
from datetime import datetime

# Define ethical considerations categories and questions
# Each category addresses a critical aspect of data ethics, with multiple
questions under each category.
ethical_topics = {
    "Privacy vs Security": [
        # Questions related to balancing user privacy with organizational or
societal security needs
        "How should organizations balance individual privacy with the need for
security?",
        "What measures can be implemented to protect privacy without compromising
security?",
        "What are examples of when privacy and security have conflicted?"
    ],
    "Impact on Marginalized Communities": [
        # Questions exploring how data practices might disproportionately affect
underrepresented groups
```

```
        "How can data collection practices disproportionately affect marginalized communities?",
        "What steps can be taken to ensure inclusivity and fairness in data analysis?",
        "Are there cases where biased data has caused harm? If so, how could it have been prevented?"
    ],
    "Role of Data Ethics in Technology Development": [
        # Questions on ethical principles in the design and deployment of data technologies
        "What ethical principles should guide the development of data-driven technologies?",
        "How can organizations ensure accountability in the use of AI and machine learning?",
        "What are the consequences of neglecting data ethics in technology development?"
    ]
}

# Function to collect responses from users
def collect_responses():
    """
    Prompts the user to provide responses to each question under each ethical topic.
    Returns:
        - responses: A dictionary containing topics and user-provided answers.
    """
    print("\nExploring Ethical Considerations in Data Privacy...\n")
    responses = {}
    for topic, questions in ethical_topics.items():
        print(f"Topic: {topic}")
        topic_responses = []  # List to store responses for this topic
        for question in questions:
            print(f"\n - {question}")
            response = input("Your response: ").strip()  # Prompt user for their thoughts
            topic_responses.append({"question": question, "response": response})
        responses[topic] = topic_responses  # Save all responses for this topic
        print("\n" + "-" * 50 + "\n")  # Add a visual separator for clarity
    return responses

# Function to generate a summary report
def generate_report(responses):
    """
```

```
    Creates a JSON file summarizing the user's responses to the ethical
questions.
    Parameters:
        - responses: A dictionary containing the topics and their corresponding
responses.
    Returns:
        - report_filename: The name of the generated JSON file.
    """
    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")  # Current timestamp
for unique filename
    report_filename = f"ethical_considerations_summary_{timestamp}.json"  #
Construct the filename

    # Save responses to a JSON file
    with open(report_filename, "w") as report_file:
        json.dump(responses, report_file, indent=4)  # Indented for readability

    print(f"\nSummary report generated: {report_filename}")  # Inform the user
about the report
    return report_filename

# Function to display a summary of discussions
def display_summary(responses):
    """
    Displays the collected responses in a user-friendly format.
    Parameters:
        - responses: A dictionary containing topics and user responses.
    """
    print("\nSummary of Ethical Considerations:")
    for topic, answers in responses.items():
        print(f"\nTopic: {topic}")  # Display topic name
        for answer in answers:
            print(f" - {answer['question']}")  # Display the question
            print(f"   * {answer['response']}")  # Display the user's response
    print("\n" + "-" * 50 + "\n")  # Add a visual separator for clarity

# Main script execution
if __name__ == "__main__":
    # Collect responses from the user
    responses = collect_responses()
    # Display a summary of the responses
    display_summary(responses)
    # Generate a report and save it as a JSON file
    generate_report(responses)
```

Output:-

```
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433> python -u "c:\Users\RAVIKANT7229\OneDrive\Desktop\Assi
gn_Privacy_Ravikant Maurya_20221433\Practical_10.py"

Exploring Ethical Considerations in Data Privacy...

Topic: Privacy vs Security

 - How should organizations balance individual privacy with the need for security?
Your response: Organizations should balance individual privacy with security by implementing measures that protect sensitive data while e
nsuring transparency and accountability. Privacy should be respected through data minimization, encryption, and consent, while security m
easures, such as strong authentication and threat detection, safeguard against potential breaches without infringing on personal rights.

 - What measures can be implemented to protect privacy without compromising security?
Your response:

 - What are examples of when privacy and security have conflicted?
Your response: Organizations can protect privacy by encrypting sensitive data, using anonymization techniques, and ensuring access contro
ls are in place. Simultaneously, security can be maintained through robust threat detection systems, secure authentication, and regular a
udits.

--------------------------------------------------

Topic: Impact on Marginalized Communities

 - How can data collection practices disproportionately affect marginalized communities?
Your response: Data collection practices can disproportionately affect marginalized communities by reinforcing biases, leading to inaccur
ate representation and exclusion from decision-making processes.

 - What steps can be taken to ensure inclusivity and fairness in data analysis?
Your response: To ensure inclusivity and fairness, diverse datasets should be used, biases in data should be identified and mitigated, an
d inclusive decision-making processes should be implemented.

 - Are there cases where biased data has caused harm? If so, how could it have been prevented?
Your response: Yes, biased data has led to unfair outcomes, such as discrimination in hiring or lending, which could have been prevented
by ensuring diverse, representative datasets and regularly auditing for biases.

--------------------------------------------------
--------------------------------------------------

Topic: Role of Data Ethics in Technology Development

 - What ethical principles should guide the development of data-driven technologies?
Your response: Data-driven technologies should be guided by principles of fairness, transparency, accountability, and respect for privacy
.

 - How can organizations ensure accountability in the use of AI and machine learning?
Your response: Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular au
dits, and transparency in decision-making processes.

 - What are the consequences of neglecting data ethics in technology development?
Your response: Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, a
nd loss of public trust.

--------------------------------------------------


Summary of Ethical Considerations:

Topic: Privacy vs Security
 - How should organizations balance individual privacy with the need for security?
   * Organizations should balance individual privacy with security by implementing measures that protect sensitive data while ensuring tr
ansparency and accountability. Privacy should be respected through data minimization, encryption, and consent, while security measures, s
uch as strong authentication and threat detection, safeguard against potential breaches without infringing on personal rights.
 - What measures can be implemented to protect privacy without compromising security?
   *
 - What are examples of when privacy and security have conflicted?
   * Organizations can protect privacy by encrypting sensitive data, using anonymization techniques, and ensuring access controls are in
place. Simultaneously, security can be maintained through robust threat detection systems, secure authentication, and regular audits.

Topic: Impact on Marginalized Communities
 - How can data collection practices disproportionately affect marginalized communities?
   * Data collection practices can disproportionately affect marginalized communities by reinforcing biases, leading to inaccurate repres
entation and exclusion from decision-making processes.
```

```
  - What steps can be taken to ensure inclusivity and fairness in data analysis?
    * To ensure inclusivity and fairness, diverse datasets should be used, biases in data should be identified and mitigated, and inclus
 e decision-making processes should be implemented.
  - Are there cases where biased data has caused harm? If so, how could it have been prevented?
    * Yes, biased data has led to unfair outcomes, such as discrimination in hiring or lending, which could have been prevented by ensur
 g diverse, representative datasets and regularly auditing for biases.

 Topic: Role of Data Ethics in Technology Development
  - What ethical principles should guide the development of data-driven technologies?
    * Data-driven technologies should be guided by principles of fairness, transparency, accountability, and respect for privacy.
  - How can organizations ensure accountability in the use of AI and machine learning?
    * Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular audits, an
 transparency in decision-making processes.
  - What are the consequences of neglecting data ethics in technology development?
    * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and loss
 ansparency and accountability. Privacy should be respected through data minimization, encryption, and consent, while security measures,
 uch as strong authentication and threat detection, safeguard against potential breaches without infringing on personal rights.
  - What measures can be implemented to protect privacy without compromising security?
    *
  - What are examples of when privacy and security have conflicted?
    * Organizations can protect privacy by encrypting sensitive data, using anonymization techniques, and ensuring access controls are i
 place. Simultaneously, security can be maintained through robust threat detection systems, secure authentication, and regular audits.

 Topic: Impact on Marginalized Communities
  - How can data collection practices disproportionately affect marginalized communities?
    * Data collection practices can disproportionately affect marginalized communities by reinforcing biases, leading to inaccurate repr
 entation and exclusion from decision-making processes.
  - What steps can be taken to ensure inclusivity and fairness in data analysis?
    * To ensure inclusivity and fairness, diverse datasets should be used, biases in data should be identified and mitigated, and inclus
 e decision-making processes should be implemented.
  - Are there cases where biased data has caused harm? If so, how could it have been prevented?
    * Yes, biased data has led to unfair outcomes, such as discrimination in hiring or lending, which could have been prevented by ensur
 g diverse, representative datasets and regularly auditing for biases.

 Topic: Role of Data Ethics in Technology Development
  - What ethical principles should guide the development of data-driven technologies?
    * Data-driven technologies should be guided by principles of fairness, transparency, accountability, and respect for privacy.
  - How can organizations ensure accountability in the use of AI and machine learning?
    * Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular audits, an
 transparency in decision-making processes.
```

```
    * Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular audits, and
transparency in decision-making processes.
 - What are the consequences of neglecting data ethics in technology development?
    * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and loss of
 - What measures can be implemented to protect privacy without compromising security?
    *
 - What are examples of when privacy and security have conflicted?
    * Organizations can protect privacy by encrypting sensitive data, using anonymization techniques, and ensuring access controls are in
place. Simultaneously, security can be maintained through robust threat detection systems, secure authentication, and regular audits.

Topic: Impact on Marginalized Communities
 - How can data collection practices disproportionately affect marginalized communities?
    * Data collection practices can disproportionately affect marginalized communities by reinforcing biases, leading to inaccurate repres
entation and exclusion from decision-making processes.
 - What steps can be taken to ensure inclusivity and fairness in data analysis?
    * To ensure inclusivity and fairness, diverse datasets should be used, biases in data should be identified and mitigated, and inclusiv
e decision-making processes should be implemented.
 - Are there cases where biased data has caused harm? If so, how could it have been prevented?
    * Yes, biased data has led to unfair outcomes, such as discrimination in hiring or lending, which could have been prevented by ensurin
g diverse, representative datasets and regularly auditing for biases.

Topic: Role of Data Ethics in Technology Development
 - What ethical principles should guide the development of data-driven technologies?
    * Data-driven technologies should be guided by principles of fairness, transparency, accountability, and respect for privacy.
 - How can organizations ensure accountability in the use of AI and machine learning?
    * Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular audits, and
transparency in decision-making processes.
 - What are the consequences of neglecting data ethics in technology development?
    * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and loss of
 public trust.
    * Data collection practices can disproportionately affect marginalized communities by reinforcing biases, leading to inaccurate repres
entation and exclusion from decision-making processes.
 - What steps can be taken to ensure inclusivity and fairness in data analysis?
    * To ensure inclusivity and fairness, diverse datasets should be used, biases in data should be identified and mitigated, and inclusiv
e decision-making processes should be implemented.
 - Are there cases where biased data has caused harm? If so, how could it have been prevented?
    * Yes, biased data has led to unfair outcomes, such as discrimination in hiring or lending, which could have been prevented by ensurin
g diverse, representative datasets and regularly auditing for biases.
```

```
Topic: Role of Data Ethics in Technology Development
 - What ethical principles should guide the development of data-driven technologies?
   * Data-driven technologies should be guided by principles of fairness, transparency, accountability, and respect for privacy.
 - How can organizations ensure accountability in the use of AI and machine learning?
   * Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular audits,
transparency in decision-making processes.
 - What are the consequences of neglecting data ethics in technology development?
   * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and los
 public trust.
Topic: Role of Data Ethics in Technology Development
 - What ethical principles should guide the development of data-driven technologies?
   * Data-driven technologies should be guided by principles of fairness, transparency, accountability, and respect for privacy.
 - How can organizations ensure accountability in the use of AI and machine learning?
   * Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular audits,
transparency in decision-making processes.
 - What are the consequences of neglecting data ethics in technology development?
   * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and los
 public trust.
   * Organizations can ensure accountability in AI and machine learning by implementing clear governance frameworks, regular audits,
transparency in decision-making processes.
 - What are the consequences of neglecting data ethics in technology development?
   * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and los
 public trust.
 - What are the consequences of neglecting data ethics in technology development?
   * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and los
 public trust.
   * Neglecting data ethics in technology development can lead to biased algorithms, privacy violations, legal repercussions, and los
 public trust.
 public trust.

--------------------------------------------------
--------------------------------------------------


Summary report generated: ethical_considerations_summary_2024-11-26_23-12-16.json
PS C:\Users\RAVIKANT7229\OneDrive\Desktop\Assign_Privacy_Ravikant Maurya_20221433>
```