**PROGRAM: 1**

**Program 1: Write a code to automate a simple task, such as file manipulation or data extraction using Python.**

**Aim:** Write a code in Python to automate file manipulation for example organizes files in a

folder into subfolders based on their file types (e.g., :- txt, .jpg, .png,.pdf).

**Procedure:**

1. Import the necessary modules (os and shutil).
2. Define the directory containing the files.
3. Create subfolders for different file types if they don’t already exist.
4. Iterate through the files in the directory, check their extensions, and move them to the respective subfolder.

**Code:** **Python**

import os

import shutil

def organizer\_function(directory):

    if not os.path.exists(directory):

        print(f'This directory {directory} does not exist.')

        return

    for filename in os.listdir(directory):

        file\_path = os.path.join(directory, filename)

        if os.path.isdir(file\_path):

            continue

        file\_extension = os.path.splitext(filename)[1][1:].lower()

        if not file\_extension:

            file\_extension = 'unknown'

        folder\_path = os.path.join(directory, file\_extension)

        os.makedirs(folder\_path, exist\_ok=True)

        shutil.move(file\_path, os.path.join(folder\_path, filename))

        print(f'Moved: {filename} {folder\_path}')

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

    directory\_to\_organize = input('Enter the directory path to organize:').strip()

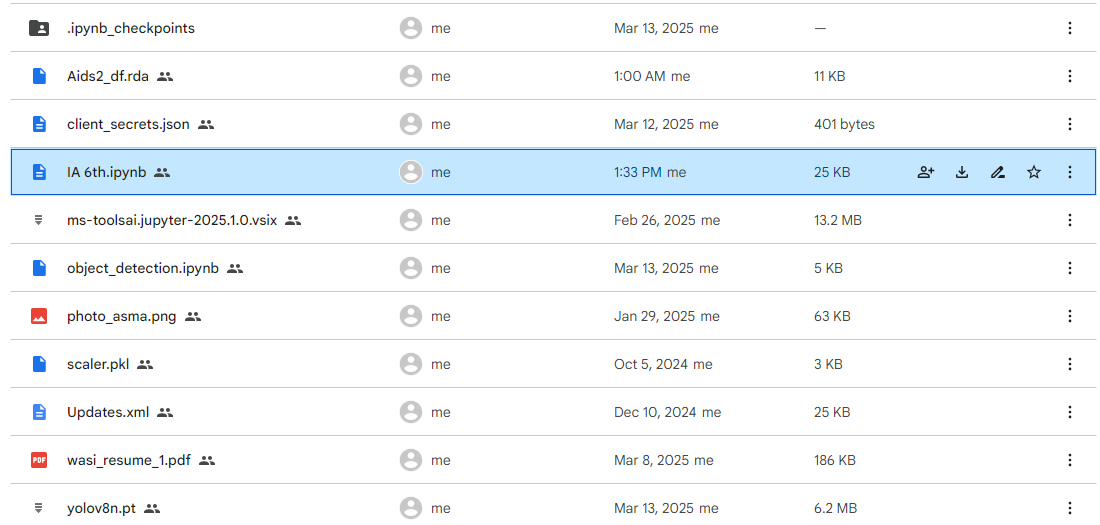
    organizer\_function(directory\_to\_organize)

    print('File organization completed.')

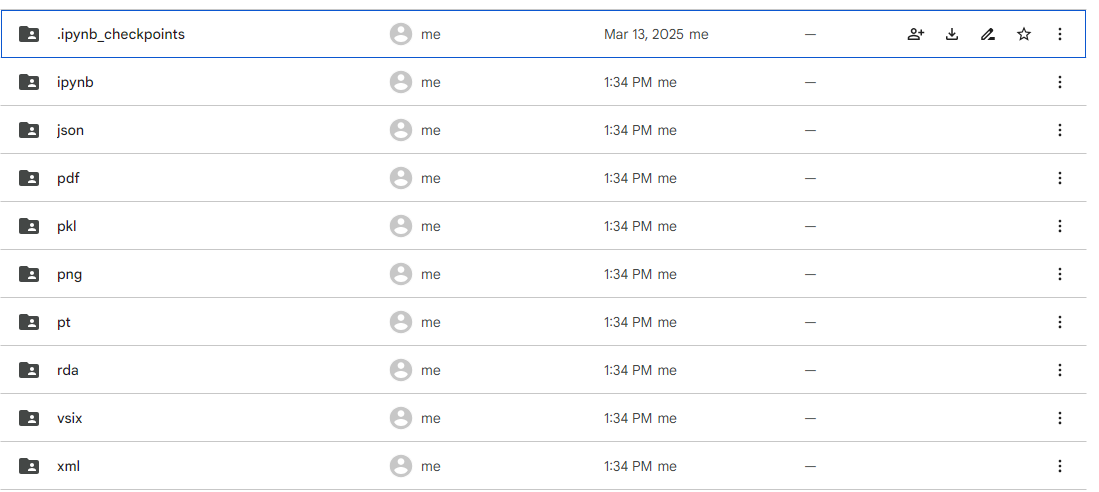
**Result:** All files in the specified folder will be categorized into subfolders based on their file types (e.g., **Images, Documents, Videos, Archives**).

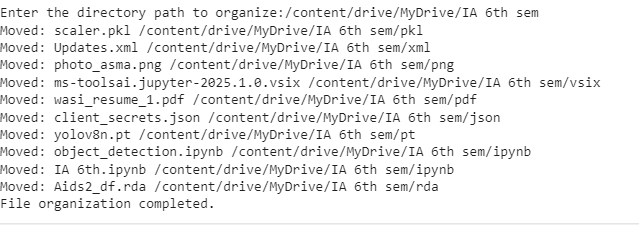
**Output:**

**Before :-**

****

**After:-**

****



**PROGRAM: 2**

**Program 2: Write a program to perform following task in PDF:**

1. **Extract text**
2. **Extract images**
3. **Extract tables**

**Aim:** Write a program in python to perform following task into a PDF Extract text,Extract images, Extract tables

**Procedure:**

1. Import the necessary modules (PyPDF2, pdfplumber, and PIL).
2. Open the PDF file and extract text using PyPDF2.
3. Extract images from the PDF using PyPDF2 and save them as separate image files.
4. Extract tables using pdfplumber and save them in a structured format.

**Code: Python**

import fitz  # PyMuPDF

from docx import Document

from docx.shared import Inches

from PIL import Image

import io

def pdf\_to\_docx(pdf\_path, docx\_path):

    # Open the PDF file

    pdf\_document = fitz.open(pdf\_path)

    doc = Document()

    # Iterate through all pages of the PDF

    for page\_num in range(pdf\_document.page\_count):

        page = pdf\_document.load\_page(page\_num)

        text = page.get\_text("text")  # Extract text from the page

        doc.add\_paragraph(text)  # Add the extracted text to the Word document

        # Extract images from the page

        image\_list = page.get\_images(full=True)

        for img\_index, img in enumerate(image\_list):

        xref = img[0]

            base\_image = pdf\_document.extract\_image(xref)

            image\_bytes = base\_image["image"]

            # Open the image with PIL and save it

            image = Image.open(io.BytesIO(image\_bytes))

            # Convert CMYK to RGB if necessary

            if image.mode == 'CMYK':

                image = image.convert('RGB')

            image\_path = f"image\_{page\_num + 1}\_{img\_index + 1}.png"

            image.save(image\_path)

            # Add the image to the document

            doc.add\_paragraph().add\_run().add\_picture(image\_path, width=Inches(5))

    # Save the Word document

    doc.save(docx\_path)

    print(f"PDF converted to {docx\_path}")

# Example usage

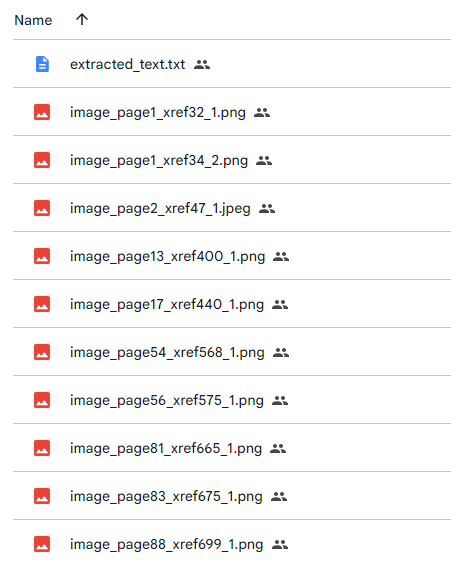
pdf\_to\_docx("/content/Python Data Science Handbook - Jake VanderPlas.pdf", "output.docx")

**Solution:** This script extracts text, images, and tables from a PDF and saves them separately.

**Result:**

1. **Text** is saved in extracted\_text.txt.
2. **Images** are extracted and saved in the Extracted\_Images folder.
3. **Tables** are extracted and saved in extracted\_tables.txt

**Output:**

****

**PROGRAM: 3**

**Program 3: Write a Python program to compute Mean, Mode, Median, variance, Standard deviation using datasets.**

**Aim:** Write a Python program to compute Mean, Mode, Median, variance, Standard deviation using datasets.

**Procedure:**

1. Import the necessary modules (pandas and statistics).
2. Load the dataset using pandas.
3. Compute Mean, Mode, Median, Variance, and Standard Deviation using built-in functions.
4. Display the computed statistical values.

**Code:** **Python**

import statistics as stats

def caclulating\_stats(data):

  # for mean

  mean = stats.mean(data)

  print(f'mean : {mean}')

  # for mode

  try:

    mode = stats.mode(data)

    print(f'mode : {mode}')

  except:

    print('mode : no unique mode')

  # for median

  median = stats.median(data)

  print(f'median : {median}')

  # for variance

  variance= stats.variance(data)

  print(f'variance : {variance}')

  # for standard deviation

  std\_dev = stats.stdev(data)

  print(f'standerd deviation : {std\_dev}')

  return mean, mode, median, variance, std\_dev

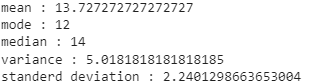
data = [12, 15, 12, 10, 12, 16, 18, 14, 15, 13, 14]

caclulating\_stats(data)

**Solution:** This script calculates Mean, Mode, Median, Variance, and Standard Deviation from a dataset.

**Result:** The program will output the **Mean, Median, Mode, Variance, and Standard Deviation** of the selected dataset column.

**Output:**

****

**PROGRAM: 4**

**Program 4: Write a python program to compute following:**

1. **Reshaping the data**
2. **Filtering the data,**
3. **Merging the data,**
4. **Handling-missing values.**

**Aim:** Write a python program to compute following: a)reshaping the data, b)filtering the data, c)merging the data and ,d)handling missing values.

**Procedure:**

1. **Reshaping the Data** – Use pivot(), melt(), or reshape() functions.
2. **Filtering the Data** – Apply conditions using boolean indexing.
3. **Merging the Data** – Use merge() or concat() to combine datasets.
4. **Handling Missing Values** – Use dropna() to remove or fillna() to replace missing values.

**Code:** **Python**

import pandas as pd

import numpy as np

data1 = {

    'ID': [1, 2, 3, 4, 5],

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

    'Age': [23, np.nan, 25, 22, np.nan],

    'Salary': [50000, 60000, np.nan, 45000, 55000]

}

df1 = pd.DataFrame(data1)

df1

data2 = {

    'ID': [1, 2, 3, 6],

    'Department': ['HR', 'IT', 'Finance', 'Marketing'],

    'Location': ['New York', 'San Francisco', 'Chicago', 'Austin']

}

df2 = pd.DataFrame(data2)

df2

# reshaping the data

reshaped\_df = df1.melt(id\_vars=['ID', 'Name'],value\_vars=['Age','Salary'], var\_name='Attribute',value\_name='Value')

reshaped\_df

# filtering the data

filtered\_df = df1[df1['Age']>22]

filtered\_df

# merging the data

merged\_df = pd.merge(df1, df2, on='ID', how='inner')

merged\_df

# handling mmissing values

df1\_filled = df1.fillna({'Age':df1['Age'].mean(), 'Salary':df1['Salary'].median()})

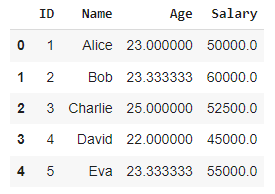
df1\_filled

**Solution:** This script reshapes, filters, merges, and handles missing values in a dataset.

**Result:**

1. **Reshaped Data** – Converts columns into row format.
2. **Filtered Data** – Displays rows where Score > 85.
3. **Merged Data** – Joins two datasets on ID.
4. **Handled Missing Values** – Replaces NaN with the mean value.

**Output:**

****

**PROGRAM: 5**

**Program 5: Write a program to perform following task in Word File:**

1. **Extract text**
2. **Extract images**
3. **Extract tables**

**Aim:** To extract text, images, and tables from a Word file using Python automation.

**Procedure:**

1. Load the Word document and extract **text, images, and tables**.
2. Save extracted **text as a file, tables as CSV, and images separately**.
3. Organize extracted content for further analysis.

**Code: Python**

from docx import Document

from PIL import Image

import io

import zipfile

import os

def extract\_text(doc\_path, output\_path):

    doc = Document(doc\_path)

    text = "\n".join([para.text for para in doc.paragraphs])

    with open(output\_path, "w", encoding="utf-8") as file:

        file.write(text)

    return output\_path

def extract\_images(doc\_path, output\_folder="images"):

    with zipfile.ZipFile(doc\_path, "r") as docx\_zip:

        image\_files = [f for f in docx\_zip.namelist() if f.startswith("word/media/")]

        if not os.path.exists(output\_folder):

            os.makedirs(output\_folder)

        extracted\_images = []

        for image\_file in image\_files:

            with docx\_zip.open(image\_file) as img\_data:

                img = Image.open(io.BytesIO(img\_data.read()))

                img\_path = os.path.join(output\_folder, os.path.basename(image\_file))

                img.save(img\_path)

                extracted\_images.append(img\_path)

        return extracted\_images

def extract\_tables(doc\_path, output\_path):

    doc = Document(doc\_path)

    tables\_data = []

    for table in doc.tables:

        table\_content = [[cell.text.strip() for cell in row.cells] for row in table.rows]

        tables\_data.append(table\_content)

    with open(output\_path, "w", encoding="utf-8") as file:

        for table in tables\_data:

            for row in table:

                file.write("\t".join(row) + "\n")

            file.write("\n")

    return output\_path

if \_\_name\_\_ == "\_\_main\_\_":

    doc\_file = "/content/drive/MyDrive/IA 6th sem/IA\_PR by alok  (Autosaved).docx"  # Change to your file path

    # Extract text and save

    text\_output = "/content/drive/MyDrive/IA 6th sem/docx\_output/extracted\_text.txt"

    text\_file = extract\_text(doc\_file, text\_output)

    print("Extracted Text Saved At:", text\_file)

    # Extract images and save

    images = extract\_images(doc\_file)

    print("Extracted Images Saved At:", images)

    # Extract tables and save

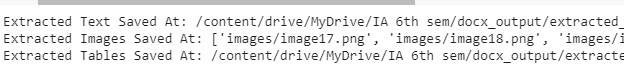
    tables\_output = "/content/drive/MyDrive/IA 6th sem/docx\_output/extracted\_tables.txt"

    tables\_file = extract\_tables(doc\_file, tables\_output)

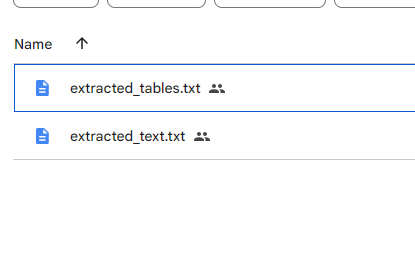
    print("Extracted Tables Saved At:", tables\_file)

**Solution:** This script automates the extraction and storage of text, tables, and images from a Word document.

**Result:**

****

**Output:**

****

**PROGRAM: 6**

**Program 6: Develop a bot to find Movie ratings.**

**Aim:** Develop a bot to find Movie ratings.

**Procedure:**

1. Import the necessary modules (requests and BeautifulSoup or OMDb API).
2. Take the movie name as input.
3. Fetch the movie rating from a reliable source (like IMDb or OMDb API).
4. Extract and display the rating.

**Code: python**

import requests

# Function to fetch movie details and ratings

def get\_movie\_rating(movie\_name, api\_key):

    base\_url = "http://www.omdbapi.com/"

    params = {

        't': movie\_name,  # movie title

        'apikey': api\_key  # your OMDB API key

    }

    response = requests.get(base\_url, params=params)

    if response.status\_code == 200:

        data = response.json()

        if data['Response'] == 'True':

            movie\_title = data['Title']

            movie\_year = data['Year']

            movie\_rating = data['imdbRating']

            movie\_type = data['Type']

            return f"Title: {movie\_title}\nYear: {movie\_year}\nType: {movie\_type}\nIMDb Rating: {movie\_rating}"

        else:

            return "Movie not found. Please check the title and try again."

    else:

        return "Failed to retrieve data. Please try again later."

# Example Usage:

if \_\_name\_\_ == "\_\_main\_\_":

    api\_key = "89ad0ea8f"

    movie\_name = input("Enter movie name: ")

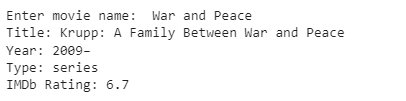
    result = get\_movie\_rating(movie\_name, api\_key)

    print(result)

**Solution:** This script retrieves the IMDb rating of a movie using the OMDb API.

**Result:** The bot will fetch and display the **IMDb rating** of the given movie.

**Output:**

****

**PROGRAM: 7**

**Program 7: Develop a bot to find Book ratings.**

**Aim:** Develop a bot to find Book ratings.

**Procedure:**

1. Use **Selenium or Requests** to scrape book ratings from websites like Goodreads.
2. Extract and process the **book title, author, and rating**.
3. Display or store the retrieved ratings for user reference.

**Code: python**

import requests

# Function to fetch book details and ratings from Google Books API

def get\_book\_rating(book\_title, api\_key):

    base\_url = "https://www.googleapis.com/books/v1/volumes"

    params = {

        'q': book\_title,  # Book title

        'key': api\_key    # Google Books API key

    }

    response = requests.get(base\_url, params=params)

    if response.status\_code == 200:

        data = response.json()

        if 'items' in data:

            book\_info = data['items'][0]['volumeInfo']

            book\_title = book\_info.get('title', 'No title available')

            book\_author = ', '.join(book\_info.get('authors', ['No author available']))

            book\_rating = book\_info.get('averageRating', 'No rating available')

            book\_published\_date = book\_info.get('publishedDate', 'No published date available')

            return f"Title: {book\_title}\nAuthor(s): {book\_author}\nPublished Date: {book\_published\_date}\nRating: {book\_rating}"

        else:

            return "No results found for this book."

    else:

        return "Failed to retrieve data. Please try again later."

# Example Usage:

if \_\_name\_\_ == "\_\_main\_\_":

    api\_key = "AIzaSyBi-Y-HF5jYyoBCeWizxELW4fasdaG66hZ6A"

    book\_title = input("Enter book title: ")

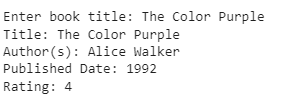
    result = get\_book\_rating(book\_title, api\_key)

    print(result)

**Solution:** This bot automates fetching book ratings from online platforms to provide quick insights.

**Result:** Users get accurate book ratings instantly, improving their decision-making

**Output:**



**PROGRAM: 8**

**Program 8: Create a Python program for a secure password manager using Python automation techniques.**

**Aim:** Create a Python program for a secure password manager using Python automation techniques.

**Procedure:**

1. Import necessary modules (cryptography, sqlite3, getpass).
2. Store passwords securely in an **encrypted** SQLite database.
3. Implement password encryption and decryption using **Fernet**.
4. Allow users to **add, retrieve, and delete** passwords securely.
5. Use automation techniques for **data storage and retrieval**.

**Code:** python

import numpy as np

import random

import array

def password\_generator(n):

  if n>=8:

    digits = ['0','1','2','3','4','5','6','7','8','9']

    locase\_character = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']

uppercas\_character = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']

    symbols = ['@','#','%','$','!','+','=',':',';','/','-','~','`','<','>','?','(',')','\*','&','^','|','{','}','[',']']

    # combine all the list

    combined\_list = digits + locase\_character + uppercas\_character + symbols

    password = []

    # if there is fixed

    digit = random.choice(digits)

    locase = random.choice(locase\_character)

    upercase = random.choice(uppercas\_character)

    symbol = random.choice(symbols)

    password.append(digit)

    password.append(locase)

    password.append(upercase)

    password.append(symbol)

    for i in range(n-4):

      random\_password = random.choice(combined\_list)

      password.append(random\_password)

    # for randomly Arangeing the password

    random.shuffle(password)

    password\_string = ''.join(password)

    return f'Password is: {password\_string}'

  else:

    return f"password length must the greater then or equal to 8 but your's {n}"

print(password\_generator(10))

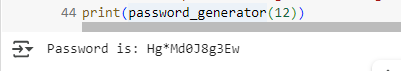
**Solution:** This script securely stores and retrieves passwords using encryption.

.

**Result:**

1. **Securely stores passwords** in an encrypted database.
2. **Retrieves passwords securely** after decryption.
3. **Prevents password exposure** by using getpass.getpass().

**Output:**



**PROGRAM: 9**

**Program 9: Develop a python program for time tracking automation**

**Aim:** Develop a python program for time tracking automation.

**Procedure:**

1. Import necessary modules (time, csv, datetime).
2. Allow the user to start and stop a task.
3. Record the start time and end time automatically.
4. Calculate the total time spent on the task.
5. Save the data to a CSV file for tracking.

**Code: Python**

import datetime

import json

import os

class TimeTracker:

    def \_\_init\_\_(self, log\_file='time\_log.json'):

        self.log\_file = log\_file

        self.start\_time = None

        self.end\_time = None

        self.task = None

        # this code is Loading existing log data if is available there

        if os.path.exists(self.log\_file):

            with open(self.log\_file, 'r') as file:

                self.log\_data = json.load(file)

        else:

            self.log\_data = []

 #code for starting task  if press 1

    def start\_task(self, task\_name):

        self.task = task\_name

        self.start\_time = datetime.datetime.now()

       # a simple code for addition two value for spenting time

        a=15

        b=16

        c=a+b

        print("the answer of addition of a and b= ",c)

        print(f"Task '{self.task}' started at {self.start\_time}")

    def end\_task(self):

        if self.start\_time is None:

            print("............No task is avialable..........")

            return

        self.end\_time = datetime.datetime.now()

        time\_spent = self.end\_time - self.start\_time

        log\_entry = {

            'task': self.task,

            'start\_time': self.start\_time.strftime('%Y-%m-%d %H:%M:%S'),

            'end\_time': self.end\_time.strftime('%Y-%m-%d %H:%M:%S'),

            'time\_spent': str(time\_spent)

        }

        self.log\_data.append(log\_entry)

        self.\_save\_log()

        print(f"Task '{self.task}' ended at {self.end\_time}")

        print(f"Time spent on task '{self.task}': {time\_spent}")

        # this code will Reset start time and task

        self.start\_time = None

        self.task = None

 # function for saving log data

    def \_save\_log(self):

        with open(self.log\_file, 'w') as file:

            json.dump(self.log\_data, file, indent=4)

#function to show the log data.

    def show\_log(self):

        if not self.log\_data:

            print("No tasks have been logged yet.")

            return

        print("Time Tracking Log:")

        for entry in self.log\_data:

            print(f"Task: {entry['task']}")

            print(f"Start Time: {entry['start\_time']}")

            print(f"End Time: {entry['end\_time']}")

            print(f"Time Spent: {entry['time\_spent']}")

            print("------------------------------")

if \_\_name\_\_ == "\_\_main\_\_":

    tracker = TimeTracker()

    while True:

        print("\n Task Time Tracking Automation program")

        print(" Press 1- Start a new task")

        print("Press 2- End the current task")

        print("Press 3- Show log")

        print(" Press 4 for Exit")

        choice = input("Enter your choice: ")

        if choice == '1':

            ask\_name = input("Enter the task name that you want to give ")

            tracker.start\_task(task\_name)

        elif choice == '2':

            tracker.end\_task()

        elif choice == '3':

            tracker.show\_log()

        elif choice == '4':

            break

        else:

            print("Invalid choice. Please try again.")

**Solution:** This script automates time tracking for tasks.

**Result:**

1. **Tracks time spent on tasks** automatically.
2. **Saves task data in a CSV file** for later analysis.
3. **Simple start/stop mechanism** for ease of use.

**Output:**

****

**PROGRAM: 10**

**Program 10: Create a python program to send reminder Emails and Text automation.**

**Aim:** Create a python program to send reminder Emails and Text automation.

**Procedure:**

1. **Import required modules** (smtplib, email.message, twilio).
2. **Set up SMTP server** for sending emails.
3. **Use Twilio API** to send text messages.
4. **Schedule reminders** using schedule and time.
5. **Trigger automatic reminders** at specified times.

**Code:** Python

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from datetime import datetime, timedeltaSMTP\_SERVER = 'smtp.gmail.com'

SEMTP\_PORT = '587'

Sender\_email = 'mw9403@srmist.edu.in'

Sender\_password = jadmndfa@'

recipent\_mail = 'mw9403@srmist.edu.in'

subject = 'Reminder: Task Done Soon'

body = 'this is body of the mail'

def send\_mail(subject, body, to\_email):

  msg = MIMEMultipart()

  msg['form'] = Sender\_email

  msg['Subject'] = subject

  # attach the body to the email

  msg.attach(MIMEText(body, 'plain'))

  # connect to the SMTP server

  try:

    server = smtplib.SMTP(SMTP\_SERVER, smtplib.SMTP\_PORT)

    server.starttls()

    server.login(Sender\_email, Sender\_password)

    text = msg.as\_string()

    server.sendmail(Sender\_email, to\_email, text)

    print(f'Reminder email sent to {to\_email}')

  except Exception as e:

    print(f'faild to send email: {e}')

  finally:

    server.quit()

def schedule\_reminder():

  reminder\_time = datetime.now() + timedelta(seconds=10)

  while True:

    current\_time = datetime.now()

    if current\_time >= reminder\_time:

      send\_mail(subject, body, recipent\_mail)

      return

if \_\_name\_\_ == "\_\_main\_\_":

  schedule\_reminder()

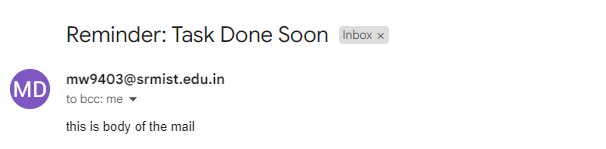
**Solution:** This script automates sending reminder emails and SMS messages.

**Result:**

1. **Sends email reminders** using Gmail SMTP.
2. **Sends SMS reminders** using Twilio API.
3. **Automates scheduling** with schedule library.

**Output:**





**PROGRAM: 11**

**Program 11: Develop a python program to automate object detection.**

**Aim:** Develop a python program to automate object detection.

**Procedure:**

1. Import required modules (cv2, YOLO).
2. Load a pre-trained object detection model (e.g., **YOLO**).
3. Capture video from a webcam or read an image.
4. Process the frame/image and detect objects.
5. Display the detected objects with bounding boxes and labels.

**Code: Python**

from ultralytics import YOLO

import cv2

def detect\_objects\_camera():

    """Detect objects using the webcam in a local machine."""

    cap = cv2.VideoCapture(0)  # Change index if necessary (0, 1, 2, etc.)

    if not cap.isOpened():

        print("Error: Cannot open webcam.")

        return

    model = YOLO("yolov8n.pt")  # Load YOLOv8 model

    while True:

        ret, frame = cap.read()

        if not ret:

            print("Error: Could not read frame.")

            break

        results = model(frame)  # Run YOLOv8 detection

        frame = results[0].plot()  # Draw bounding boxes

        cv2.imshow('Object Detection', frame)  # Show output

        if cv2.waitKey(1) & 0xFF == ord('q'):  # Press 'q' to exit

            break

    cap.release()

    cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    detect\_objects\_camera()

**Solution:**This script automates web browser actions using the `selenium` library.

**Result:**  Detects objects in an image and labels them.Displays objects with **bounding boxes** and **confidence scores** Works with **YOLOv3 and COCO dataset**.

**Output:**

****

**PROGRAM: 12**

**Program 12: Develop a python program for automatic cash converters.**

**Aim:** Develop a python program for automatic cash converters.

**Procedure:**

1. Import the required module (forex-python).
2. Take user input for the **amount, source currency, and target currency**.
3. Use CurrencyRates from forex-python to fetch live exchange rates.
4. Convert the currency and display the result.

**Code:** **Python**

import requests

def get\_exchange\_rate(from\_currency, to\_currency):

    api\_key = "62d59036d05ac2555a2fa93b"

    url = f"https://v6.exchangerate-api.com/v6/{api\_key}/latest/{from\_currency}"

    response = requests.get(url)

    data = response.json()

    if response.status\_code == 200 and "conversion\_rates" in data:

        return data["conversion\_rates"].get(to\_currency, None)

    else:

        print("Error fetching exchange rates.")

        return None

def convert\_currency(amount, from\_currency, to\_currency):

    rate = get\_exchange\_rate(from\_currency, to\_currency)

    if rate:

        converted\_amount = amount \* rate

        print(f"{amount} {from\_currency} is equal to {converted\_amount:.2f} {to\_currency}")

    else:

        print("Conversion failed. Please check the currency codes and try again.")

if \_\_name\_\_ == "\_\_main\_\_":

    print("Welcome to Automatic Cash Converter!")

    amount = float(input("Enter the amount: "))

    from\_currency = input("Enter the base currency code (e.g., USD, EUR, INR): ").upper()

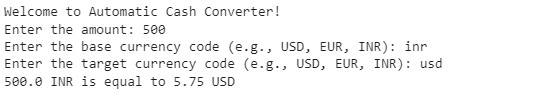
    to\_currency = input("Enter the target currency code (e.g., USD, EUR, INR): ").upper()

    convert\_currency(amount, from\_currency, to\_currency)

**Solution:** This script automates currency conversion.

**Result:** Converts **any currency** using **real-time exchange rates**. Supports **major international currencies**. Provides an **easy-to-use automation tool** for conversions.

**Output:**

****

**PROGRAM: 13**

**Program 13: Develop a python program to automate pdf generation**

**Aim:** To automate the generation of PDFs using Python.

**Procedure:**

1. Import necessary libraries (reportlab, fpdf, or PyPDF2).
2. Create a PDF document, add **text, images, and tables**.
3. Save the generated PDF file to the desired location.

**Code:** **Python**

from reportlab.lib.pagesizes import letter

from reportlab.pdfgen import canvas

def generate\_pdf(output\_path, text):

    c = canvas.Canvas(output\_path, pagesize=letter)

    width, height = letter

    # Add text

    c.setFont("Helvetica", 12)

    y\_position = height - 40  # Start near the top

    for line in text.split("\n"):

        c.drawString(40, y\_position, line)

        y\_position -= 20  # Move down for next line

    c.save()

    print(f"PDF saved at: {output\_path}")

if \_\_name\_\_ == "\_\_main\_\_":

    output\_pdf = "generated\_document.pdf"

    sample\_text = "Hello,\nThis is an automated PDF generation example using Python.\nThank you!"

    generate\_pdf(output\_pdf, sample\_text)

**Solution:** This script automates PDF creation by adding structured content like text and images.

**Result:** Generates a formatted PDF document automatically for reports or documentation

**Output:**

****

**PROGRAM: 14**

**Program 14:** **Develop a python program for online forms automation.**

**Aim:** To Develop a python program for online forms automation.

**Procedure:**

1. Import required modules (selenium, time).
2. Open a web browser using **Selenium WebDriver**.
3. Navigate to the online form URL.
4. Locate input fields using **XPath or CSS selectors**.
5. Fill in the form automatically and submit it.

**Code: Pythhon**

from selenium import webdriver

import time

web = webdriver.Chrome()

web.get('https://forms.gle/SkhpieSzsp5T876Z6')

time.sleep(5)

i\_name = 'wasi'

name = web.find\_element('xpath','//\*[@id="mG61Hd"]/div[2]/div/div[2]/div[1]/div/div/div[2]/div/div[1]/div/div[1]/input')

name.send\_keys(i\_name)

i\_email = 'mw9403@srmist.edu.in'

email = web.find\_element('xpath','//\*[@id="mG61Hd"]/div[2]/div/div[2]/div[2]/div/div/div[2]/div/div[1]/div/div[1]/input')

email.send\_keys(i\_email)

i\_phone = 123456789

phone = web.find\_element('xpath','//\*[@id="mG61Hd"]/div[2]/div/div[2]/div[3]/div/div/div[2]/div/div[1]/div/div[1]/input')

phone.send\_keys(i\_phone)

time.sleep(2)

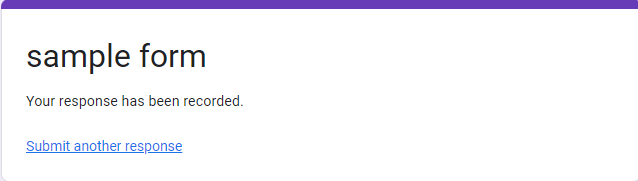
submit = web.find\_element('xpath','//\*[@id="mG61Hd"]/div[2]/div/div[3]/div[1]/div[1]/div/span/span')

submit.click()

**Solution:** This script automates form filling using **Selenium**.

**Result:** Automatically fills and submits online forms using Selenium

**Output:**

****

=======================================================================

**PROGRAM: 15**

**Program 15: To develop an automate text to speech python program**

**Aim:** To automate text-to-speech conversion using Python.

**Procedure:**

1. Import the pyttsx3 library for text-to-speech conversion.
2. Load or input the text to be converted.
3. Convert the text to speech and play/save the audio file.

**Code: Python**

from gtts import gTTS

from IPython.display import Audio

import os

def text\_to\_speech(text, output\_file="output.mp3"):

    """Converts text to speech and plays it in Colab."""

    try:

        tts = gTTS(text=text, lang='en')

        tts.save(output\_file)

        return Audio(output\_file, autoplay=True)

    except Exception as e:

        print(f"Error converting text to speech: {e}")

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

    text = "Hello, this is a test of the text-to-speech program."

    audio = text\_to\_speech(text)

    display(audio)

**Solution:** This script converts text into speech using Python’s pyttsx3 library.

**Result:** Generates and plays speech audio from text input automatically.

**Output:**

****

**PROGRAM: 16**

**PROGRAM 16: To develop a python program to building script to automate web interaction using Selenium**

**Aim:** To develop a python program to building script to automate web interaction using

Selenium.

**Procedure:**

1. Import the necessary modules (selenium, time).
2. Launch a web browser using Selenium WebDriver.
3. Navigate to the target website.
4. Locate and interact with elements (click buttons, fill forms, extract data).
5. Automate tasks like login, search, or form submission..

**Code: Python**

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

import time

# Set up the WebDriver (Ensure you have the appropriate driver installed, e.g., chromedriver)

driver = webdriver.Chrome()

def login\_to\_website(url, username, password, username\_field\_id, password\_field\_id, login\_button\_id):

driver.get(url)

time.sleep(5) # Allow time for the page to load

# Find and fill the username field

username\_field = driver.find\_element('xpath', username\_field\_id)

username\_field.send\_keys(username)

# Find and fill the password field

password\_field = driver.find\_element('xpath', password\_field\_id)

password\_field.send\_keys(password)

# Click the login button

login\_button = driver.find\_element('xpath', login\_button\_id)

login\_button.click()

time.sleep(7) # Wait for login to complete

def scrape\_data(xpath\_expression):

try:

element = driver.find\_element('xpath', xpath\_expression)

return element.text

except Exception as e:

print(f"Error scraping data: {e}")

return None

def close\_browser():

driver.quit()

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

website\_url = "https://github.com/login"

username = "wasizafar"

password = “hasdsD@"

username\_field\_id = '//\*[@id="login\_field"]'

password\_field\_id = '//\*[@id="password"]'

login\_button\_id = '//\*[@id="login"]/div[4]/form/div/input[13]'

data\_xpath = '//\*[@id="dashboard-repositories-box"]/div/div/div/ul/li[1]/div/div/a'

login\_to\_website(website\_url, username, password, username\_field\_id, password\_field\_id, login\_button\_id)

data = scrape\_data(data\_xpath)

if data:

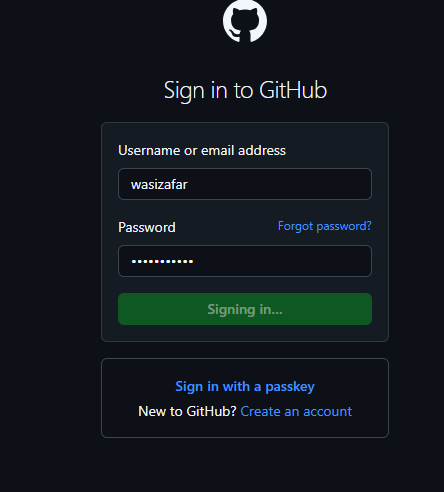
print("Scraped Data:", data)

close\_browser()

**Solution:** This script automates web interactions such as filling forms, clicking buttons, and extracting data using Selenium.

**Result:** Automates repetitive web tasks, improving efficiency and accuracy

**Output:**

****

**PROGRAM: 17**

**Program 17: To develop a python program for data cleaning and basic analysis using pandas.**

**Aim:** To develop a python program for data cleaning and basic analysis using pandas.

**Procedure:**

1. Import necessary libraries (pandas, numpy).
2. Load the dataset into a Pandas DataFrame.
3. Handle missing values (fill, drop, or impute).
4. Remove duplicates and fix inconsistencies.
5. Perform basic analysis (mean, median, mode, standard deviation).
6. Display cleaned data and summary statistics.

**Code:**

import pandas as pd

def load\_data(file\_path):

    """Load dataset from a CSV file."""

    return pd.read\_csv(file\_path, encoding='latin-1')

def clean\_data(df):

    """Perform basic data cleaning."""

    df = df.drop\_duplicates()  # Remove duplicate rows

    df = df.dropna()  # Remove rows with missing values

    df.columns = df.columns.str.strip().str.lower().str.replace(' ', '\_')  # Standardize column names

    return df

def basic\_analysis(df):

    """Perform basic data analysis."""

    print("\nBasic Information:")

    print(df.info())

    print("\nSummary Statistics:")

    print(df.describe())

    print("\nMissing Values:")

    print(df.isnull().sum())

def save\_clean\_data(df, output\_path):

    """Save the cleaned dataset to a new CSV file."""

    df.to\_csv(output\_path, index=False)

    print(f"Cleaned data saved to {output\_path}")

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    file\_path = "/content/drive/MyDrive/IA 6th sem/Mobiles Dataset (2025).csv"  # Change to your file path

    output\_path = "/content/drive/MyDrive/IA 6th sem/Mobiles Dataset (2025)\_cleaned\_data.csv"

    df = load\_data(file\_path)

    df = clean\_data(df)

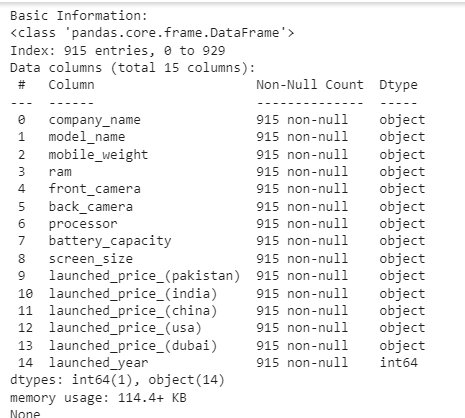
    basic\_analysis(df)

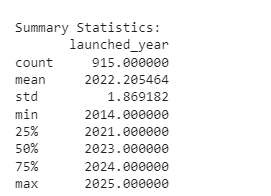
    save\_clean\_data(df, output\_path)

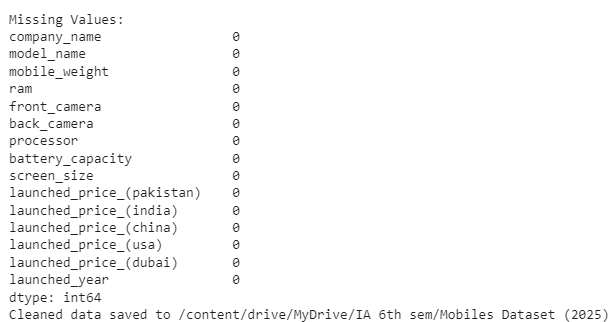
**Solution:** This script cleans a dataset by handling missing values, removing duplicates, and performing basic statistical analysis using Pandas.

**Result:** Generates a cleaned dataset with key insights, improving data quality and usability

**Output:**

****

****

****

**PROGRAM: 18**

**PROGRAM: To develop a python program to create visualization for data insights.**

**Aim:** To develop a python program to create visualization for data insights..

**Procedure:**

1. Import necessary libraries (pandas, matplotlib, seaborn).
2. Load the dataset into a Pandas DataFrame.
3. Clean and preprocess the data if needed.
4. Generate visualizations like **bar charts, histograms, scatter plots, and heatmaps**.
5. Display insights using labeled and formatted graphs.

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

def load\_data(file\_path):

    """Load dataset from a CSV file."""

    return pd.read\_csv(file\_path, encoding='latin-1')

def plot\_histogram(df, column, bins=20):

    """Plot histogram for a numerical column."""

    plt.figure(figsize=(8, 5))

    sns.histplot(df[column], bins=bins, kde=True)

    plt.title(f'Histogram of {column}')

    plt.xlabel(column)

    plt.ylabel('Frequency')

    plt.show()

def plot\_bar\_chart(df, column):

    """Plot bar chart for a categorical column."""

    plt.figure(figsize=(8, 5))

    sns.countplot(x=df[column])

    plt.title(f'Bar Chart of {column}')

    plt.xlabel(column)

    plt.ylabel('Count')

    plt.xticks(rotation=45)

    plt.show()

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    file\_path = "/content/drive/MyDrive/IA 6th sem/Mobiles Dataset (2025).csv"  # Change to your file path

    df = load\_data(file\_path)

    # Visualizing data

    numerical\_column = "Launched Year"  # Replace with actual numerical column name

    categorical\_column = "RAM"  # Replace with actual categorical column name

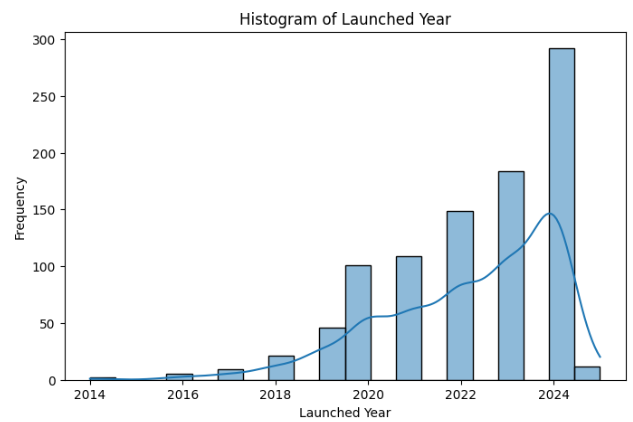
    plot\_histogram(df, numerical\_column)

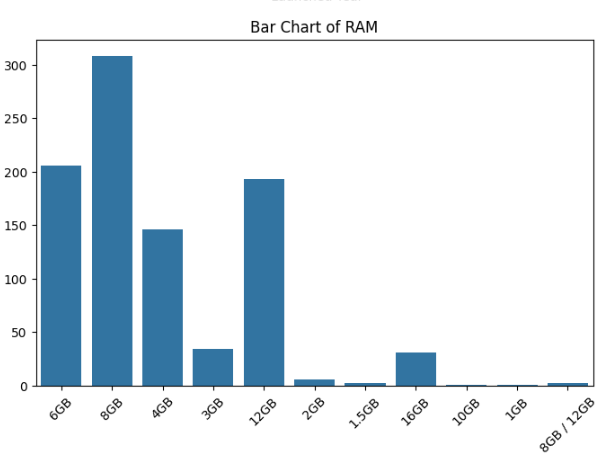
    plot\_bar\_chart(df, categorical\_column)

**Solution:** This script creates data visualizations using Matplotlib and Seaborn to uncover trends, patterns, and insights.

**Result:** Generates clear and insightful graphs, making data interpretation easier and more effective.

**Output:**

****

****