**PRACTICAL JOURNAL**

**in**

**Data Science Implementation**

**Submitted by**

**KSMSCIT007 – DHANRAJ CHINTA**

**For the award of the degree of**

**MASTER OF SCIENCE (INFORMATION TECHNOLOGY)**

**PART – ⅠI**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**KISHINCHAND CHELLARAM COLLEGE**

**(Affiliated to University of HSNCU)**

**MUMBAI – 400020**

**MAHARASHTRA**

**2024-25**

**Subject Code : BIT605D**

**Data Science Implementation**



**KISHINCHAND CHELLARAM COLLEGE**

CHURCHGATE, MUMBAI – 400 020.

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**M.SC.I.T PART- II**

**CERTIFICATE**

This is to certify that the Practical conducted by

Mr. **CHINTA DHANRAJ NAGESH** for M.Sc. (IT) Part- II Semester- III,

Seat No: **KSMSCIT007** at Kishinchand Chellaram College in partial fulfillment for the MASTERS OF SCIENCE (INFORMATION TECHNOLOGY). Degree Examination for semester ⅡI has been periodically examined and signed, and the course of term work has been satisfactorily carried out for the year 2024 - 2025. This Practical journal had not been submitted for any other examination and does not form part of any other course undergone by the candidate.

**Signature Signature Signature**

**Lecturer-In-Charge External Examiner Course Coordination**

**Guided By Examined By Certified By**

**College Stamp**

**✳️Index✳️**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Practical No.** | **Title** | **Date** | **Page No.** | **Signature** |
| 1 | [Exploring Color Maps in Matplotlib: Visualizing Random Data with Different Color Schemes](#Practical1) | 16-06-24 | 1-2 |  |
| 2 | [Geospatial Visualization with GeoPandas](#Practical2) | 30-06-24 | 3-4 |  |
| 3 | [Interactive Geospatial Visualization with Folium: Mapping Major Cities of India](#Practical3) | 07-07-24 | 5-6 |  |
| 4 | [Calculating and Visualizing the Current Position of the Moon Using Astropy](#Practical4) | 28-07-24 | 7-8 |  |
| 5 | [Visualizing New COVID-19 Cases Using Plotly Express:](#Practical5)   * [Daily COVID-19 Case Trends: A Line Plot Visualization](#Practical5) * [Monthly COVID-19 Case Trends: A Bar Plot Visualization](#Practical5) | 28-07-24 | 9-10 |  |
| 6 | [Linear Regression Analysis of Diabetes Data: Predicting Age from BMI](#Practical6) | 11-08-24 | 11-12 |  |
| 7 | [Creating a Word Cloud](#Practical7) | 18-08-24 | 13 |  |
| 8 | [Introduction to Cassandra:](#Practical8)   * [Basic Commands](#Practical8) * [KeySpace Creation](#Practical8) | 18-08-24 | 14-17 |  |
| 9 | [Using OpenCV and File Management Libraries:](#Practical9)   * [Extracting Frames from Video: Converting MP4 to JPEG Images](#Practical9) * [Reconstructing a Video from Image Frames](#Practical9) | 25-08-24 | 18-21 |  |
| 10 | [Working with MongoDB:](#Practical10)   * [Python](#Practical10) * [R](#Practical10) | 15-09-24 | 22-23 |  |
| 11 | [Horus](#Practical11)   * [Audio to CSV File](#Practical11) * [Image to CSV File](#Practical11) | 29-09-24 | 24-26 |  |
| 12 | Data Analysis and Visualization   * Binning * Location Coordinates Average * Random Coordinates and Plottig Using Basemap | 29-09-24 | 27-30 |  |

# Practical 1 – Color Map

**What is a Color Map?**

A color map is a range of colors that can be applied to data to encode information visually. It is essentially a mapping from numerical data to colors. Each color map contains a range of colors, from the lowest to the highest value in the data.

**Matplotlib provides a variety of built-in color maps that fall into several categories:**

1. **Sequential**: Used for data that has a natural progression from low to high (e.g., temperature). These color maps typically have one continuous range of hues.
   * Examples: viridis, plasma, inferno, magma.
2. **Diverging**: Used for data that has a central point and then diverges to higher and lower values (e.g., anomalies like temperature differences).
   * Examples: PiYG, PRGn, bwr, seismic.
3. **Qualitative**: Used for categorical data where there is no specific ordering (e.g., categories like red, blue, green).
   * Examples: Set1, Set2, Paired.

**Importance of Color Maps:**

1. **Data** **Interpretation**: Color maps help the human brain quickly grasp differences in data. For instance, in a heatmap, dark reds may indicate higher values, while lighter shades indicate lower values.
2. **Accessibility**: Color maps like cividis are designed for colorblind-friendly visualizations, which is crucial when working with a diverse audience.
3. **Aesthetics**: Proper use of color maps can make data visualizations more appealing, enhancing readability and clarity, especially in presentations and reports.

**Code**

import matplotlib.pyplot as plt

import numpy as np

for i in plt.colormaps():

  sTitle='Color Map:' + i

  fig=plt.figure(figsize=(10, 10))

  plt.title(sTitle)

  imgplot = plt.imshow(np.random.rand(10,10))

  if i != "cividis":

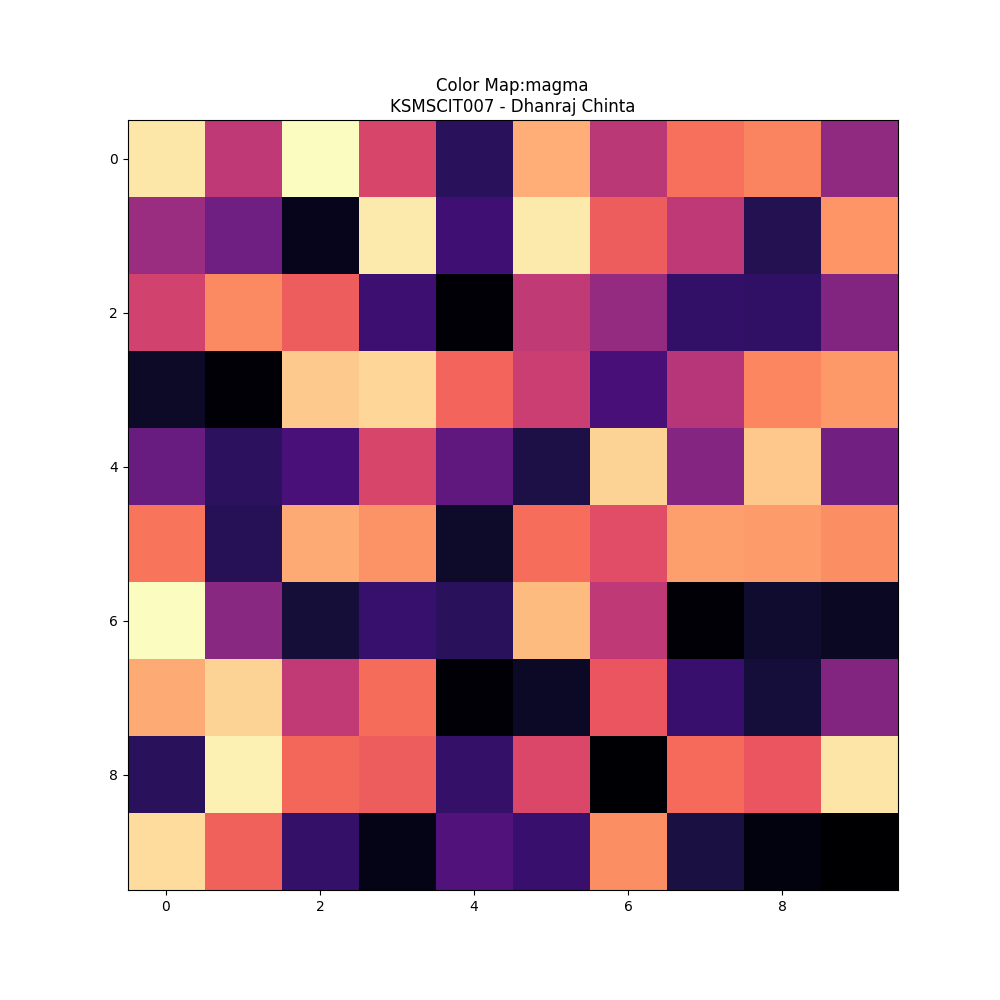
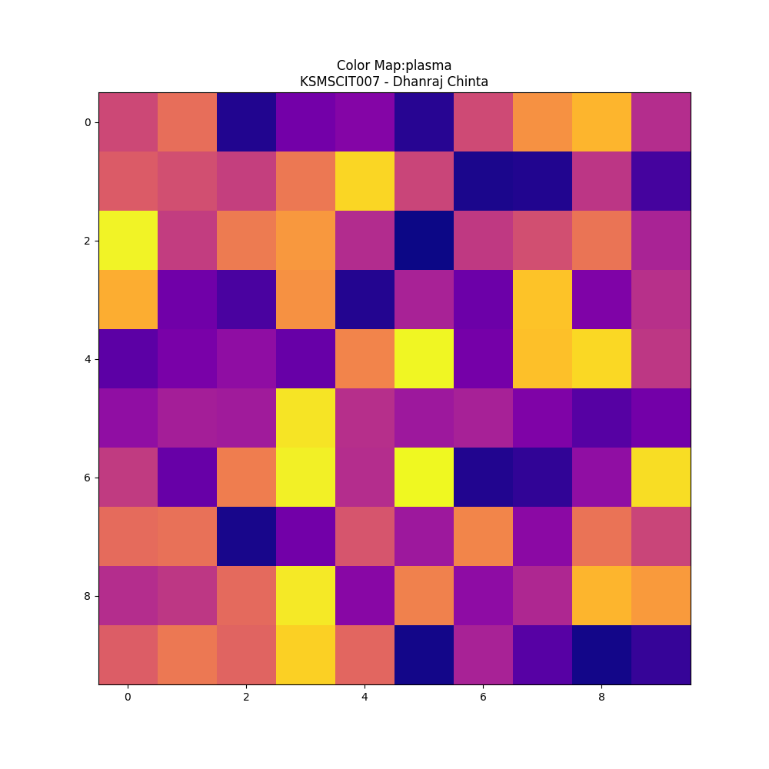
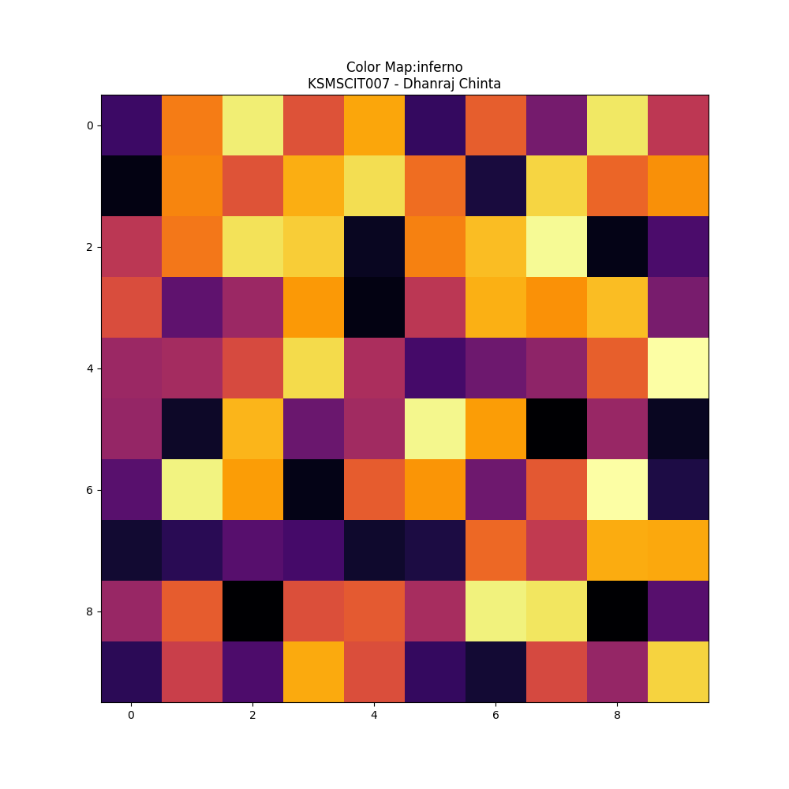
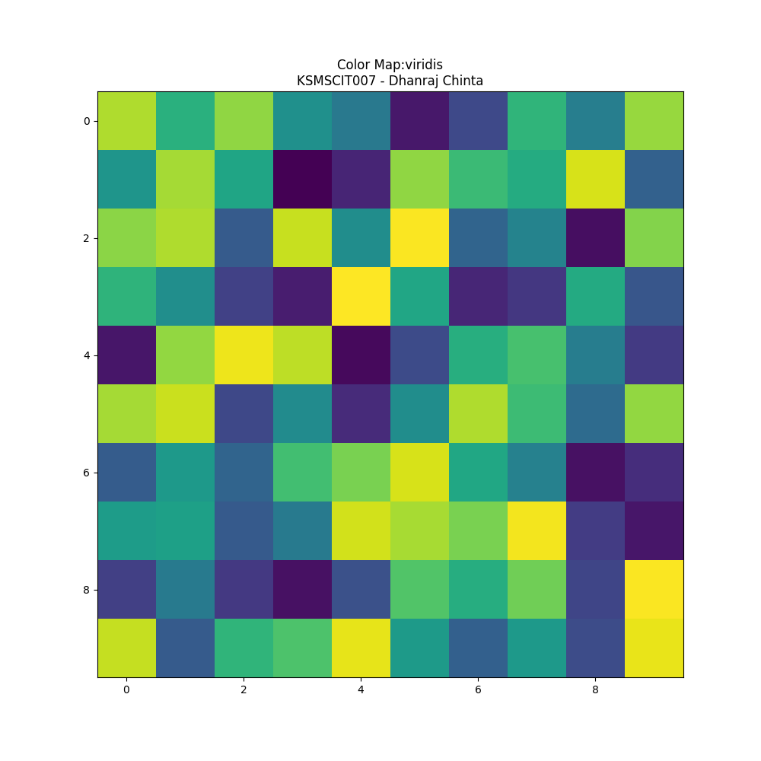
    imgplot.set\_cmap(i)

    plt.show()

  else:

    break

**Output**

**  **

# Practical 2 – GeoPandas

**GeoPandas: A Library for Geospatial Data**

**GeoPandas** is an open-source library that simplifies working with geospatial data in Python. It extends the functionality of Pandas (a popular data manipulation library) to handle geographic data by integrating the capabilities of shapely (for geometric operations), Fiona (for file access), and Pyproj (for coordinate transformations).

**Key Features:**

* Handles Spatial Data: GeoPandas can read, write, and manipulate geospatial data formats like shapefiles, GeoJSON, and more.
* Geometric Objects: GeoPandas introduces a special object called a GeoDataFrame, which includes geometry columns with geometric shapes like points, lines, and polygons.
* Spatial Operations: It allows for spatial joins, buffering, intersection, union, and other geometric manipulations directly on geospatial data.

**Main Components of GeoPandas:**

1. **GeoDataFrame:**
   * An extended version of Pandas DataFrame that contains a geometry column, which holds spatial information (Points, Polygons, etc.).
   * It allows spatial operations like calculating the area, length, or distance between shapes.
2. **Geometry Types:**
   * Points: Represents a single point (e.g., coordinates of a location).
   * Polygons: Represents areas (e.g., a city boundary or a building footprint).
   * LineStrings: Represents lines (e.g., roads or rivers).

import geopandas as gpd

import matplotlib.pyplot as plt

import fiona

from shapely.geometry import Point # Import the Point object

fiona.drvsupport.supported\_drivers['ESRI Shapefile'] = 'rw'

with fiona.Env(SHAPE\_RESTORE\_SHX='YES'):

  india\_gdf = gpd.read\_file("/content/india\_2000-2014\_state.shp")

#Mumbai City in Bharat

mumbai=gpd.GeoDataFrame([{'City':'Mumbai','geometry':Point(72.877,19.0760)}],crs="EPSG:4326")

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

india\_gdf.plot(ax=ax,color='white',edgecolor="black")

mumbai.plot(ax=ax,color='red')

for x,y,label in zip(mumbai.geometry.x,mumbai.geometry.y,mumbai['City']):

  ax.text(x,y,label)

plt.title("Bharat")

plt.show()

****

# Practical 3 - Folium

**Folium**

Folium is a Python library designed for creating interactive and aesthetically pleasing geospatial visualizations. It leverages the Leaflet.js library to render maps and geographical data efficiently. Folium is particularly useful for visualizing location-based data, making it a powerful tool for geospatial analysis and presentation.

**Key Features:**

1. **Map Creation:**
   * Folium allows users to create interactive maps with customizable center points and zoom levels. This provides a flexible foundation for visualizing spatial data.
2. **Markers:**
   * Users can add markers to maps to denote specific locations. Markers can include popups with detailed information and tooltips for quick reference.
3. **GeoJSON Integration:**
   * Folium supports the overlay of GeoJSON data to represent geographical features such as boundaries and regions. This enables the visualization of complex spatial data structures.
4. **Interactive Features:**

* Folium maps support interactive elements, including clickable markers, dynamic popups, and responsive tooltips.

**Code**

import folium

import requests

# Create a map centered around India

m = folium.Map(location=[20.5937, 78.9629], zoom\_start=5)

# GeoJSON data for India's borders

geojson\_data = requests.get("https://raw.githubusercontent.com/python-visualization/folium/master/examples/data/world-countries.json").json()

# List of cities with their names, locations, and populations

cities = [

    {"name": "New Delhi", "location": [28.6139, 77.2090], "population": "21.75 million"},

    {"name": "Mumbai", "location": [19.0760, 72.8777], "population": "20.18 million"},

    {"name": "Bengaluru", "location": [12.9716, 77.5946], "population": "8.42 million"},

    {"name": "Chennai", "location": [13.0827, 80.2707], "population": "10.97 million"},

    {"name": "Kolkata", "location": [22.5726, 88.3639], "population": "14.85 million"}

]

# Highlight India's boundary and add markers

folium.GeoJson(geojson\_data, name="India", highlight\_function=lambda x: {'fillColor': 'green'}).add\_to(m)

# Add markers for each city

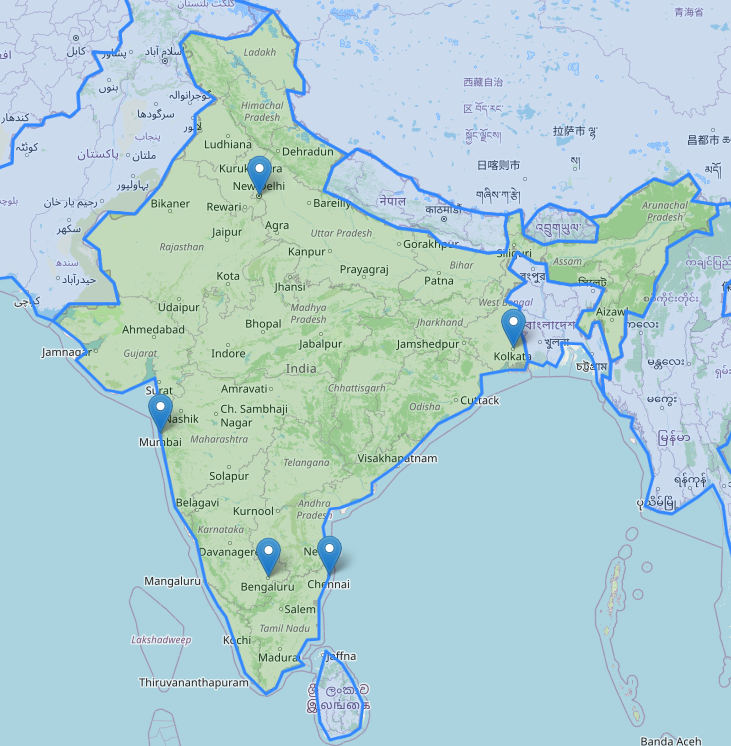
for city in cities:

    folium.Marker(city['location'], popup=f"<b>{city['name']}</b><br>Population: {city['population']}", tooltip=city['name']).add\_to(m)

# Save and display the map

m.save("india\_cities\_map.html")

m



# Practical 4 - Astropy

Astropy plays a crucial role in astronomical computations. This powerful library enables researchers to accurately determine the Moon's position. Consequently, astronomers can conduct observations and experiments effectively.

**Understanding the Moon's Position**

The Moon's position in the sky changes constantly. Various factors contribute to its movement, including:

* Orbital Mechanics: The Moon orbits Earth in an elliptical path
* Gravitational Pull: The gravitational influence from Earth affects its position
* Lunar Phases: The changing phases impact visibility and position

Astropy allows users to compute these positional changes seamlessly.

**Why Use Astropy for Moon Position Calculations?**

Astropy is not just another library; it offers unique advantages. Here are several reasons to consider:

* Accuracy: Astropy relies on precise algorithms
* User-Friendly: The interface is straightforward for both beginners and experts
* Well-Documented: Comprehensive documentation supports users at all levels

Utilizing Astropy enhances the reliability of Moon position calculations.

**Key Features of Astropy**

Astropy comes equipped with several features that simplify computations:

* Time and Location Handling: Users can input geographic coordinates and timestamps
* Ephemeris Data: Access to data helps determine the Moon's position efficiently
  + Users can leverage JPL (Jet Propulsion Laboratory) data
  + The library provides accurate predictions based on scientific models

**Code**

from astropy.coordinates import solar\_system\_ephemeris, get\_moon, AltAz, EarthLocation

import astropy.units as u

import time

from astropy.time import Time

import matplotlib.pyplot as plt

solar\_system\_ephemeris.set('builtin')

time\_utc = Time.now()

print(time\_utc)

moon = get\_moon(time\_utc)

location = EarthLocation.of\_site('Kitt Peak')

moon\_altaz = moon.transform\_to(AltAz(obstime=time\_utc, location=location))

print(f'Moon coordinates (RA, Dec): {moon.ra}, {moon.dec}')

print(f'Moon Altitude: {moon\_altaz.alt}')

print(f'Moon Azimuth: {moon\_altaz.az}')

plt.figure(figsize=(10, 8))

plt.subplot(111, projection='polar')

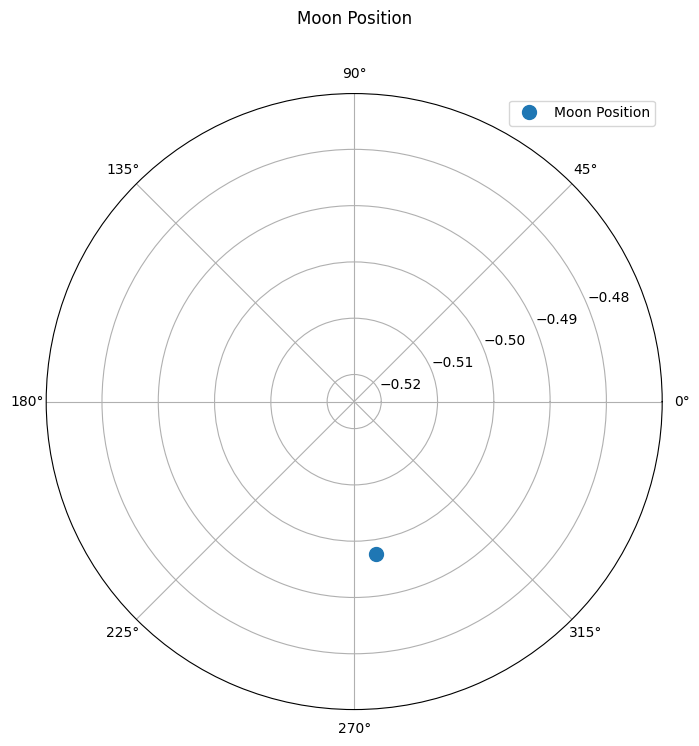
plt.title('Moon Position', y=1.1)

plt.polar(moon.ra.radian, moon.dec.radian, 'o', markersize=10, label='Moon Position')

plt.grid(True)

plt.legend()

plt.show()



# Practical 5 – Plotly Express

**Overview**

This practical demonstrates how to visualize daily and monthly trends of COVID-19 cases using Plotly Express and Pandas. The visualizations include:

1. Daily COVID-19 Case Trends – A line plot to track new cases reported each day.
2. Monthly COVID-19 Case Trends – A bar plot to show the total new cases reported each month.

The dataset used in this practical is sourced from Johns Hopkins University's COVID-19 data repository, which is publicly available.

**What is Plotly?**

plotly.py, colloquially referred to as [Plotly](https://plotly.com/), is an interactive, open-source, and browser-based graphing library. It offers Python-based charting, powered by plotly.js. The library ships with over 30 chart types, including scientific charts, 3D graphs, statistical charts, SVG maps, financial charts, and more.

**Key Concepts**

* Pandas: A versatile data manipulation library used to handle, clean, and aggregate data for analysis.
* Date and Time Handling: Converting columns to datetime or period format is essential for proper time-based analysis and visualization.

**Code**

Daily COVID-19 Case Trends – A line plot to track new cases reported each day.

import pandas as pd

import plotly.express as px

# Load dataset from the internet (Johns Hopkins COVID-19 dataset)

url = "https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/jhu/total\_cases.csv"

df = pd.read\_csv(url)

# Convert 'date' column to datetime

df['Date'] = pd.to\_datetime(df['date'])

# Calculate new cases (daily difference)

df['New Cases'] = df['World'].diff()

# Filter out rows with missing new case values

df = df.dropna(subset=['New Cases'])

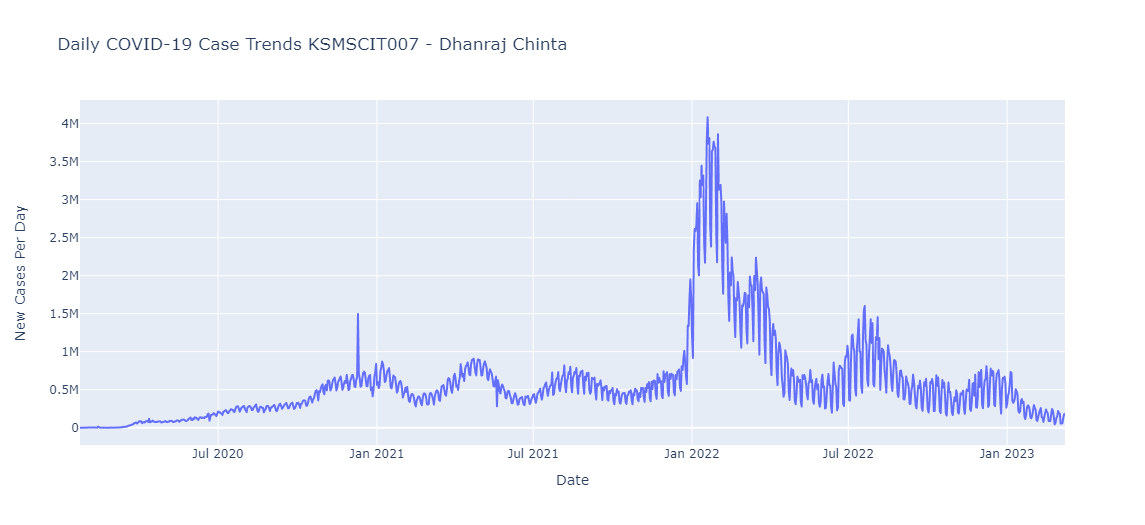
# Daily COVID-19 Case Trends: Line Plot

fig\_daily = px.line(df, x='Date', y='New Cases',

                    title="Daily COVID-19 Case Trends KSMSCIT007 - Dhanraj Chinta",

                    labels={'New Cases': 'New Cases Per Day', 'Date': 'Date'})

fig\_daily.show()

****

Monthly COVID-19 Case Trends – A bar plot to show the total new cases reported each month.

# Create a 'Month' column from 'Date'

df['Month'] = df['Date'].dt.to\_period('M').astype(str)  # Convert Period to string

# Group by 'Month' and sum 'New Cases'

monthly\_data = df.groupby('Month')['New Cases'].sum().reset\_index()

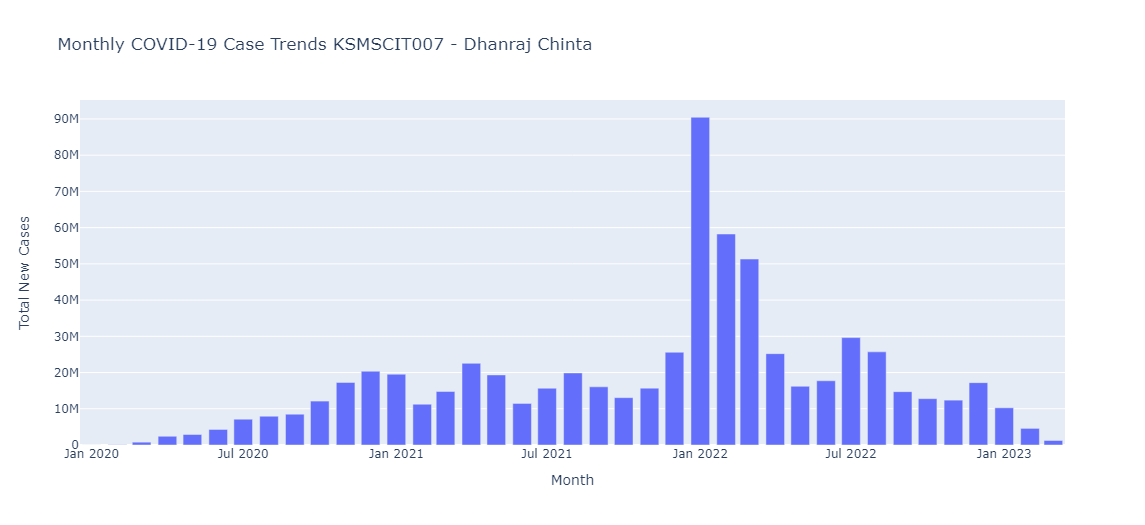
# Monthly COVID-19 Case Trends: Bar Plot

fig\_monthly = px.bar(monthly\_data, x='Month', y='New Cases',

                     title='Monthly COVID-19 Case Trends' +'\n' +'KSMSCIT007 - Dhanraj Chinta' ,

                     labels={'New Cases': 'Total New Cases', 'Month': 'Month'})

fig\_monthly.show()

****

# Practical 6 – Linear Regression

Linear regression is one of the simplest and most widely used machine learning algorithms, particularly for tasks involving prediction of continuous variables. This article explains the concepts used in linear regression by utilizing the **Diabetes dataset** from the sklearn.datasets module. We'll build a linear regression model to predict a target variable (disease progression) using one feature (BMI).

**Dataset Overview: The Diabetes Dataset**

The **Diabetes dataset** is a classic dataset from the **scikit-learn** library, often used for regression tasks. It contains medical information about diabetes progression. There are 10 baseline variables such as age, sex, BMI (Body Mass Index), blood pressure, and others. The target is a quantitative measure of disease progression one year after baseline.

However, in this example, we are focusing on one feature — the **BMI** — to predict the target, which is the measure of disease progression.

**Code**

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the diabetes dataset

diabetes = datasets.load\_diabetes()

# Use only one feature

diabetes\_X = diabetes.data[:, np.newaxis, 2]

# Split the data into training/testing sets

diabetes\_X\_train = diabetes\_X[:-30]

diabetes\_X\_test = diabetes\_X[-50:]

# Split the targets into training/testing sets

diabetes\_y\_train = diabetes.target[:-30]

diabetes\_y\_test = diabetes.target[-50:]

# Create linear regression object

regr = linear\_model.LinearRegression()

# Train the model using the training sets

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

# Make predictions using the testing set

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

# The coefficients

print('Coefficients: \n', regr.coef\_)

# The mean squared error

print("Mean squared error: %.2f"

% mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % r2\_score(diabetes\_y\_test, diabetes\_y\_pred))

print("Sample BMI VALUES")

for i in range(len(diabetes\_X\_test)):

    print(diabetes\_X\_test[i])

# Plot outputs

plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color='blue')

plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color='red', linewidth=2)

plt.xticks(())

plt.yticks(())

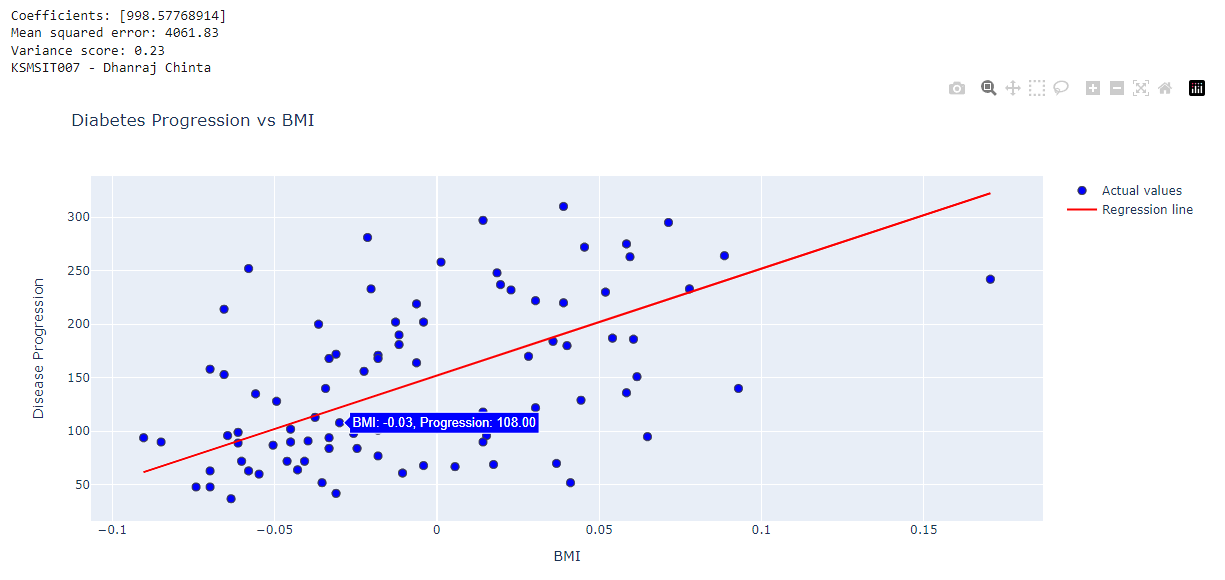
plt.axis('tight')

plt.title("Diabetes")

plt.xlabel("BMI")

plt.ylabel("Age")

plt.show()



# Practical 7 – Word Cloud

**What is a Word Cloud?**

A **word cloud** (or **tag cloud**) is a visual representation of text data, where the size of each word is proportional to its frequency or importance in the text. Word clouds are often used in data visualization to highlight the most common terms in a body of text, helping to quickly grasp the key themes or topics.

For example, in a word cloud generated from an article, the most frequent words would appear larger and bolder, while less frequent words would appear smaller. It's an intuitive way to summarize and visualize textual data.

**Word Cloud in Python**

In Python, word clouds can be generated using the **wordcloud** library, which is a simple and flexible tool for this purpose.

**Code**

import matplotlib.pyplot as plt

text ="“Dhanraj\_Chinta. Grit. Power. Relentless. Edge. Fearless. Unstoppable. Fury. Strength. Chaos. Vibe. Roar. Apex. Bold. Thrill."

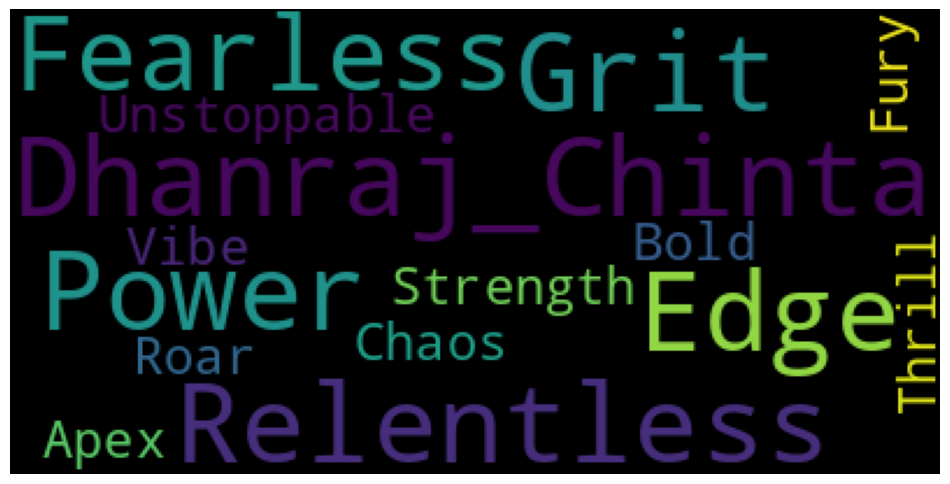
wordcloud=WordCloud().generate(text)

plt.figure(figsize=(12,12))

plt.imshow(wordcloud)

plt.axis("off")

plt.show()



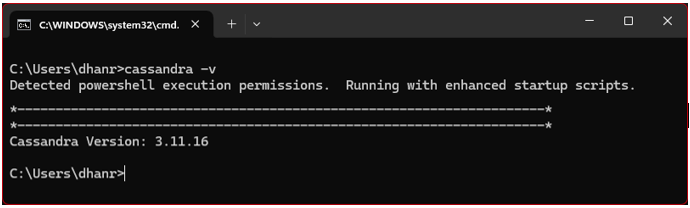
# Practical 8 – Cassandra

CQL offers a model similar to SQL. The data is stored in **tables** containing **rows** of **columns**. For that reason, when used in this document, these terms (tables, rows and columns) have the same definition that they have in SQL.

**Basic Queries**

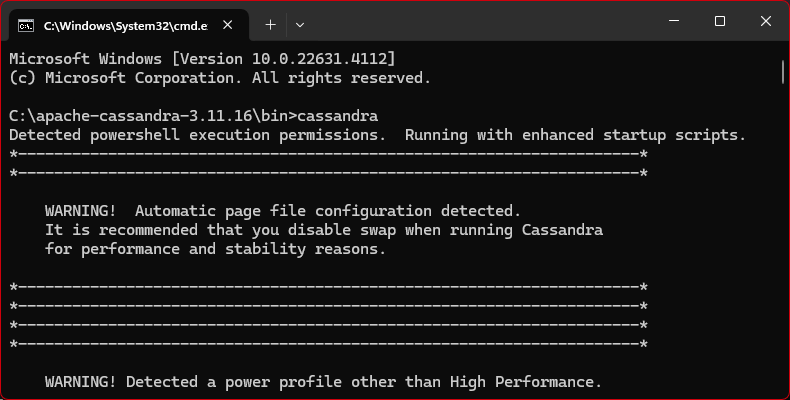
Check Cassandra Installation

**cassandra -v**

****

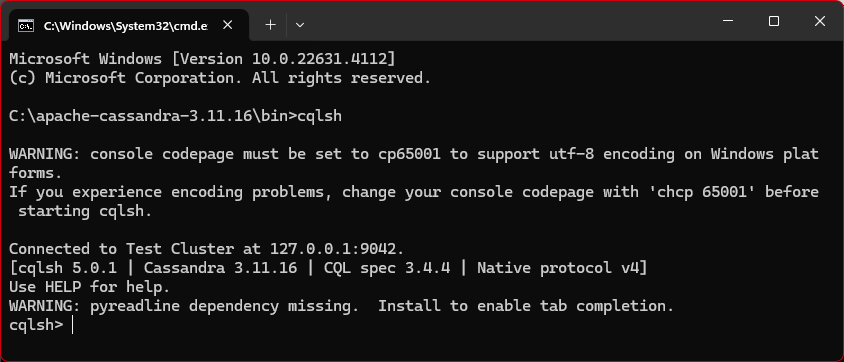
Start Cassandra Service

**cassandra**

****

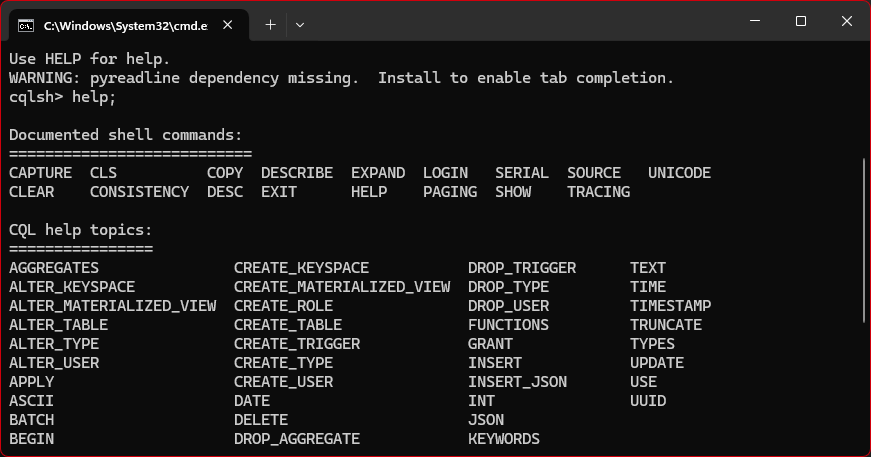
Launch Cassandra Query Language Shell (CQLSH)

**cqlsh**

****

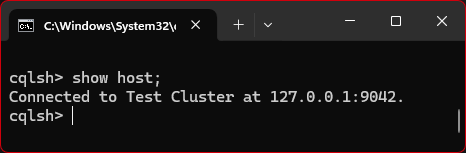
Provides help and lists available commands in the Cassandra shell.

**help;**

****

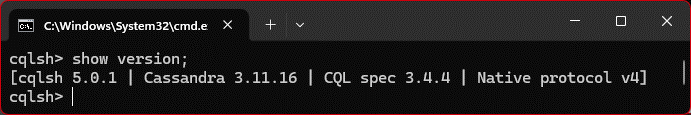
Displays the connected host.

**show host;**

****

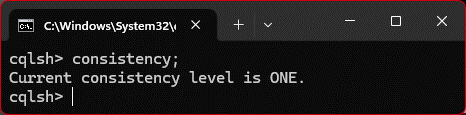
Displays the version of Cassandra.

**show version;**

****

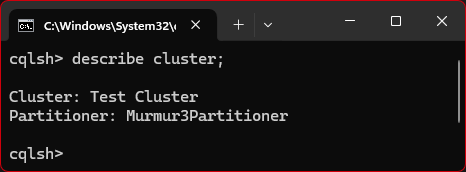
Shows the current consistency level for reads and writes.

**consistency;**

****

Provides details about the current cluster configuration.

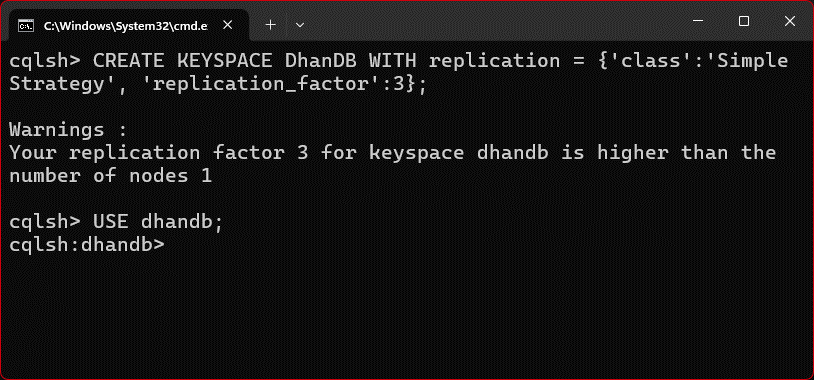
**describe cluster;**

****

**Creating a KEYSPACE**

Create a new Keyspaces

**CREATE KEYSPACE DhanDB WITH replication = {'class':'SimpleStrategy', 'replication\_factor':3};**

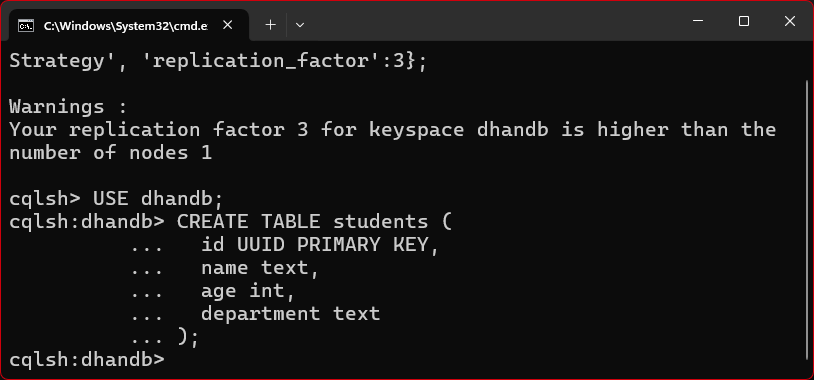
****

Use Keyspace: After creating a keyspace, you need to select it to run queries within it:

**USE dhandb**

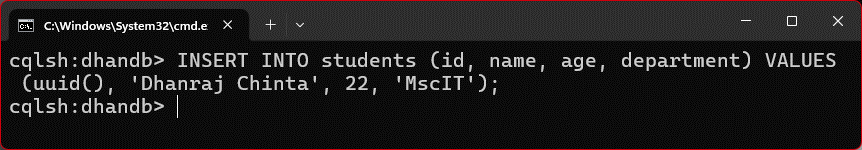
Create Table: You can create tables within your keyspace, for example:

**CREATE TABLE students{...}**

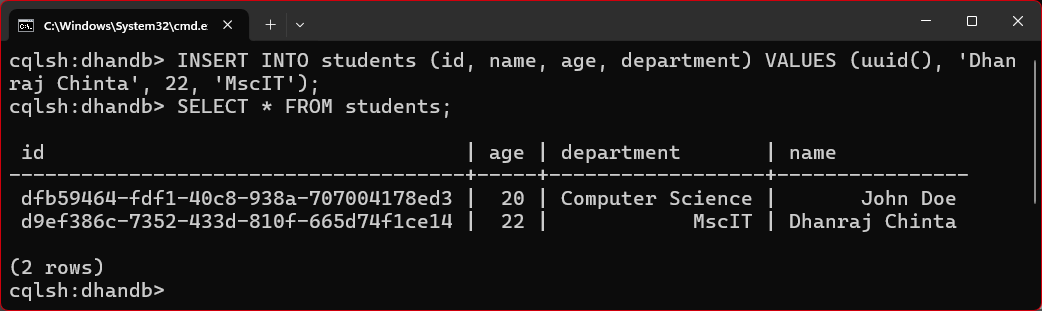
****

Insert Data: To insert data into your table:

INSERT INTO students (id, name, age, department) VALUES (uuid(), 'Dhanraj Chinta', 22, 'MscIT');

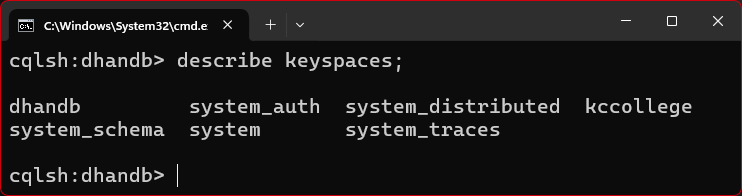
****

**Select Data**: Fetch data from the table:

SELECT \* FROM students;****

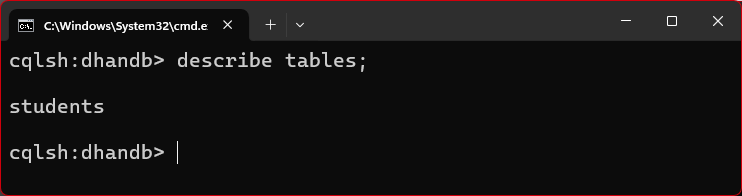
Shows all the keyspaces in the database.

**describe keyspaces;**

****

Lists the tables in the current keyspace (ensure you have a keyspace selected).

**describe tables;**

****

# Practical 9 – Opencv

**Introduction to OpenCV**

OpenCV (Open Source Computer Vision Library) is a powerful, open-source library designed for computer vision and machine learning. Its primary goal is to facilitate the development of real-time computer vision applications. One of the most frequently used modules within OpenCV is cv2, which provides a comprehensive set of tools and functions for image and video processing.

**Advanced Features**

Beyond the basics, cv2 supports more advanced computer vision tasks, such as:

**Object Detection:** Using pre-trained models and techniques like Haar cascades or deep learning-based detectors.

**Feature Matching:** Identifying and matching features across images using algorithms like SIFT, SURF, and ORB.

**Image Segmentation:** Dividing an image into meaningful segments to analyze or process different regions independently.

**Real-Time Applications**

cv2 is used extensively in real-time applications like:

**Facial Recognition:** Identifying or verifying individuals based on facial features.

**Autonomous Vehicles:** Processing camera feeds for object detection, lane detection, and navigation.

**Augmented Reality (AR):** Overlaying digital information on live camera feeds.

**Code**

**Video To Image**

import os

import shutil

import cv2

sinputfilepath = "/content/drive/MyDrive/Test/zoro.gif"

dest = "/content/drive/MyDrive/Test/Images"

if os.path.exists(dest):

  shutil.rmtree(dest)

if not os.path.exists(dest):

  os.makedirs(dest)

vidcap = cv2.VideoCapture(sinputfilepath)

success,image = vidcap.read()

count = 0

lists = []

while success:

    # Construct the filename for the frame

    sFrame = os.path.join(dest, f'pic-frame-{count:04d}.jpg')

    if image is not None:

        # Save the frame

        cv2.imwrite(sFrame, image)

        # Check if the file was created correctly

        if os.path.getsize(sFrame) == 0:

            os.remove(sFrame)

            print(f'Removed: {sFrame}')

        else:

            lists.append(f'Extracted: {sFrame}')

    else:

        print(f'Failed to read frame at index {count}')

    # Read the next frame

    success, image = vidcap.read()

    count += 1

    # Exit conditions

    if count > 70:  # Exit after 70 frames

        break

vidcap.release()

****

**Images to Video**

import os

import matplotlib.pyplot as plt

import matplotlib.animation as animation

from PIL import Image

# Directory containing the frames

frame\_dir = '/content/drive/MyDrive/Test/Images'

# Get list of image files

image\_files = sorted(f for f in os.listdir(frame\_dir) if f.endswith('.jpg'))

# Check if there are images

if not image\_files:

    raise ValueError("No images found in the directory.")

# Read the first image to get the size

first\_frame = Image.open(os.path.join(frame\_dir, image\_files[0]))

width, height = first\_frame.size

# Create a figure and axis

fig, ax = plt.subplots()

# Set axis limits based on the size of the images

ax.set\_xlim(0, width)

ax.set\_ylim(height, 0)

ax.axis('off')

# Function to update the frame in the animation

def update(frame\_number):

    img\_path = os.path.join(frame\_dir, image\_files[frame\_number])

    img = Image.open(img\_path)

    ax.imshow(img)

# Create animation

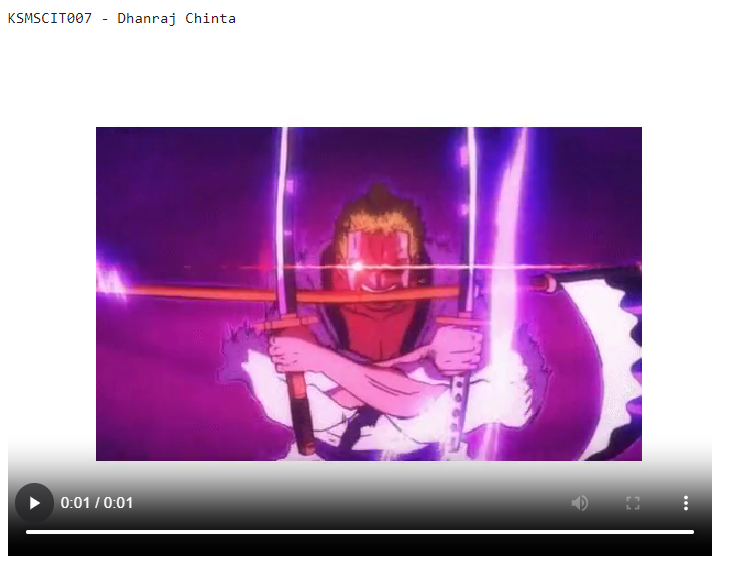
ani = animation.FuncAnimation(fig, update, frames=len(image\_files), repeat=False, interval=50)

print('KSMSCIT007 - Dhanraj Chinta')

# Display the animation

from IPython.display import HTML

HTML(ani.to\_html5\_video())

****

# Practical 10 - Mongo

MongoDB is an open-source, non-relational database management system (DBMS) that stores and processes data as documents instead of tables and rows. MongoDB is known for its flexibility and scalability, and is used by over 47,000 customers across 118 regions.

**Features**

**Document model**

MongoDB's document model is designed to be simple for developers to learn and use. Documents are formatted as Binary JSON (BSON) and can store various types of data.

**Scalability**

MongoDB is a distributed database that's built for high availability, horizontal scaling, and geographic distribution.

**Query API**

MongoDB offers a developer-native query API for working with data.

**Code**

import matplotlib.pyplot as plt

from pymongo import MongoClient

class MongoDBClient:

    def \_\_init\_\_(self, uri, db\_name):

        self.db = MongoClient(uri)[db\_name]

    def retrieve\_documents(self, collection, limit=0):

        return list(self.db[collection].find(limit=limit))

# Setup

client = MongoDBClient("mongodb+srv://hel1801:4PS6UPu452lbK2hE@helcluster1.xp0mw.mongodb.net/HelDB", 'HelDB')

docs = client.retrieve\_documents("Zoro's Arsenal", limit=3)

total\_swords = client.db["Zoro's Arsenal"].count\_documents({})

# Generate message

output = f"Connected to HelDB\nTotal: {total\_swords} Swords\nRetrieved {len(docs)} Swords\n\n" + "\n\n".join(

    [f'{doc.get("name", "N/A")}\nGrade: {doc.get("grade", "N/A")}\nStatus: {doc.get("status", "N/A")}\n'

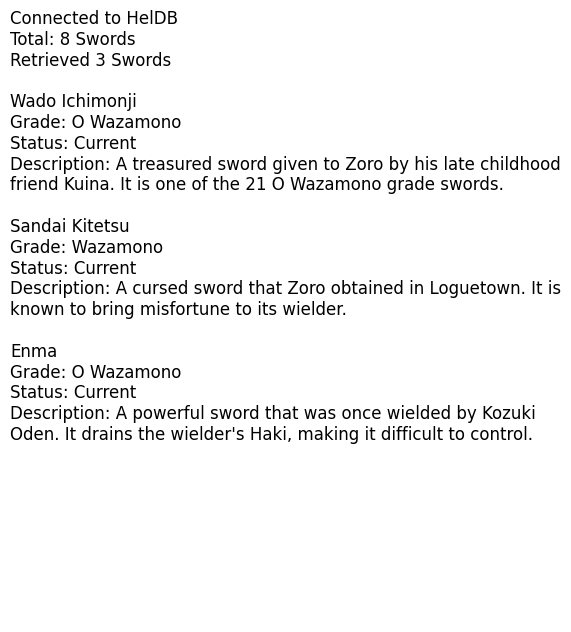
     f'Description: {doc.get("description", "N/A")}' for doc in docs])

# Plot

plt.text(0, 0.5, output, fontsize=12, wrap=True)

plt.axis('off')

plt.show()



**Mongo in R**

library(mongolite)

library(ggplot2)

uri <- "mongodb+srv://hel1801:4PS6UPu452lbK2hE@helcluster1.xp0mw.mongodb.net/HelDB?retryWrites=true&w=majority"

collection\_name <- "Zoro's Arsenal"

mongo\_conn <- tryCatch(mongo(collection = collection\_name, url = uri), error = function(e) NULL)

status <- if (is.null(mongo\_conn)) "Connection to MongoDB failed." else "Connection to MongoDB established successfully."

total\_swords <- mongo\_conn$find()

swords <- mongo\_conn$find(limit = 3)

underline\_text <- function(text) {

  paste0(paste(strsplit(text, NULL)[[1]], collapse = "\u0332"), collapse = "")

  }

output\_message <- if (length(swords) == 0) {

  paste(status, "\nNo documents found.", sep = "\n")

} else {

  swords\_message <- sprintf("Total Swords: %s\nSwords found: %s", nrow(total\_swords), nrow(swords))

  sword\_details <- paste(sprintf(

    "\n%s\nGrade: %s\nStatus: %s\nDescription: %s \n",

    swords$name, swords$grade, swords$status, swords$description

  ), collapse = "")

  paste(status, swords\_message, sword\_details, sep = "\n")

}

ggplot() +

  annotate("text", x = 0.1, y = 0.9, label = output\_message, size = 4, hjust = 0.5, vjust = 0.5) +

  theme\_void() +  # Removes all the plot elements like gridlines

  theme(

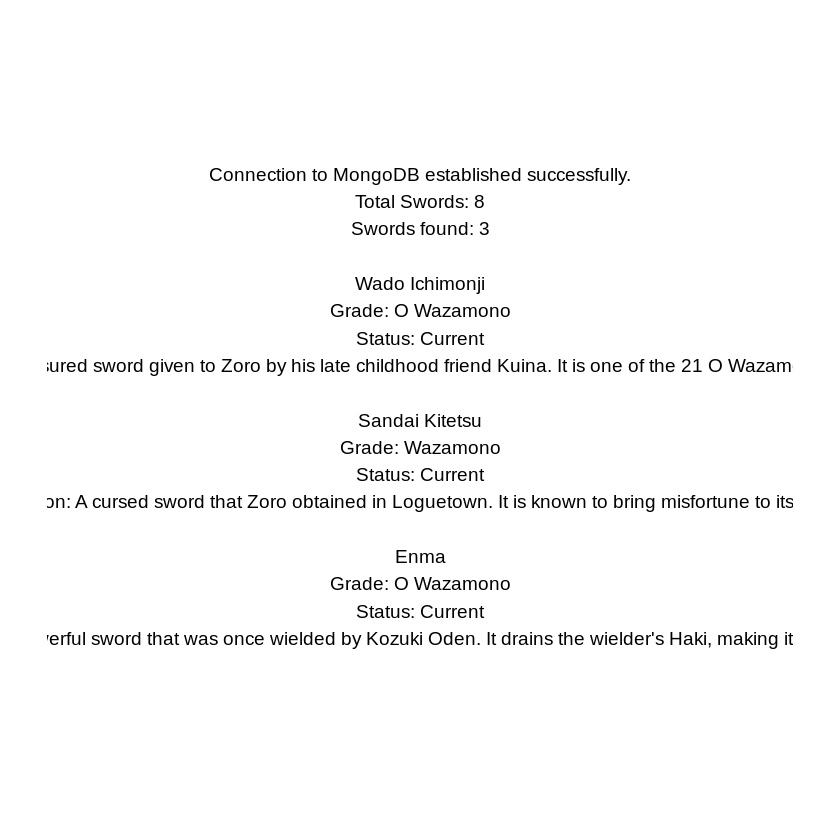
    plot.background = element\_rect(fill = "white", color = NA),  # White background

    panel.background = element\_rect(fill = "white", color = NA),  # White panel background

    plot.margin = unit(c(1, 1, 1, 1), "cm")  # Adjust the margin as needed

  )

**Output**



**Practical 11 – Horus**

Welcome to Horus - your pre-ops buddy in the world of investigations! Formerly known as 'Sentinel', Horus is a comprehensive OSINT (Open-Source Intelligence) and digital forensics tool built in Python.

From leveraging APIs to compiling data, Horus serves as your all-in-one solution for investigative assistance. Whether you're conducting digital forensics or delving into open-source intelligence, Horus provides the tools you need to gather, analyze, and interpret data efficiently.

**Code**

**Audio to Csv**

from scipy.io import wavfile

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

def show\_info(aname, a, r):

    print(f"Audio: {aname}\n")

    print(f"Rate: {r}\n")

    print(f"Shape: {a.shape}")

    plot\_info(aname, a, r)

def plot\_info(aname, a,r):

    sTitle= 'Signal Wave - '+ aname + ' at ' + str(r) + 'hz'

    plt.title(sTitle)

    sLegend=[]

    for c in range(a.shape[1]):

        sLabel = 'Ch' + str(c+1)

        sLegend=sLegend+[str(c+1)]

        plt.plot(a[:,c], label=sLabel)

    plt.legend(sLegend)

    plt.show()

sInputFileName = '/content/drive/MyDrive/Test/4ch-sound.wav'

print('Processing : ', sInputFileName)

InputRate, InputData = wavfile.read(sInputFileName)

show\_info("2 channel", InputData,InputRate)

ProcessData=pd.DataFrame(InputData)

sColumns= ['Ch1','Ch2','ch3','ch4']

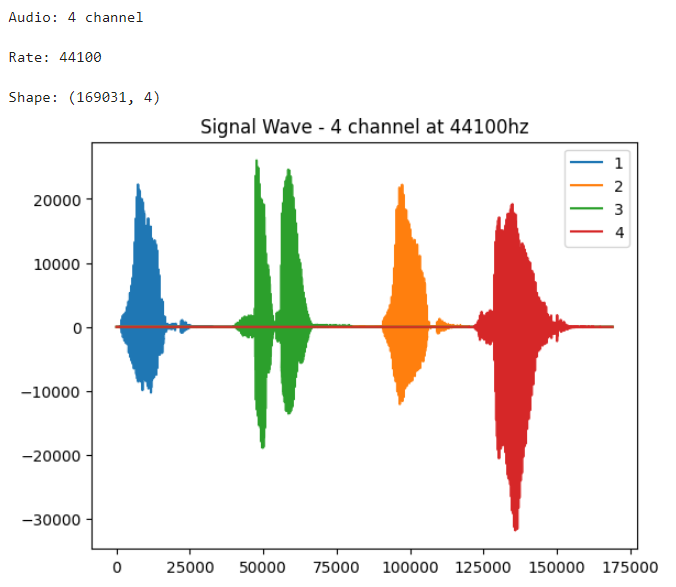
ProcessData.columns=sColumns

OutputData=ProcessData

sOutputFileName='/content/drive/MyDrive/Test/HORUS-ElvisBoiiiAudio-4ch.csv'

OutputData.to\_csv(sOutputFileName, index = False)

**Output**

****

**Image to Csv**

from PIL import Image

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

def image\_to\_csv\_and\_show(image\_path, csv\_output\_path):

    # Load image, convert to RGB, and convert to NumPy array

    img\_array = np.array(Image.open(image\_path).convert('RGB'))

    # Print image details

    print(f"Shape: {img\_array.shape}, Dtype: {img\_array.dtype}, Min: {img\_array.min()}, Max: {img\_array.max()}")

    # Convert to DataFrame and save to CSV

    pd.DataFrame(img\_array.reshape(-1, 3), columns=['R', 'G', 'B']).to\_csv(csv\_output\_path, index=False)

    # Display image

    plt.imshow(img\_array)

    plt.axis('off')

    plt.show()

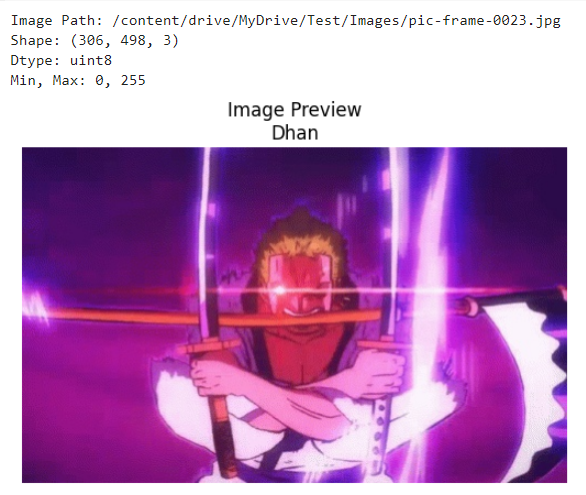
# Example usage

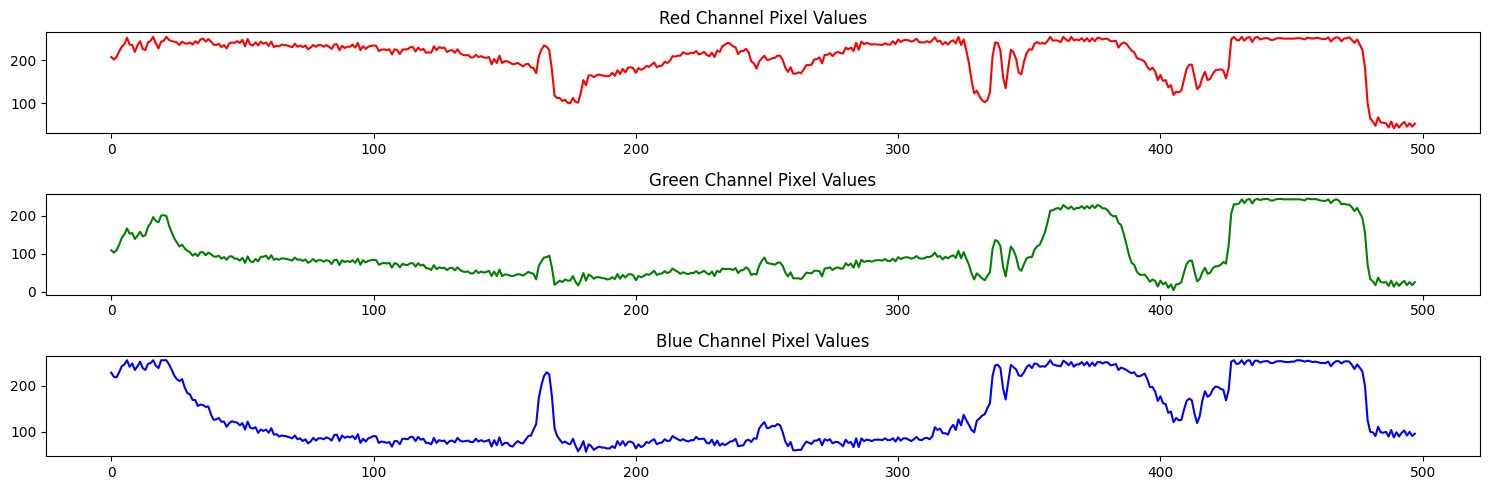
image\_path = 'path/to/image.png'  # Replace with your image path

csv\_output\_path = 'path/to/image\_data.csv'  # Replace with your desired CSV output path

image\_to\_csv\_and\_show(image\_path, csv\_output\_path)

**Output**

****



**Practical 12**

**Binning**

Binning (or bucketing) is a crucial data pre-processing technique used to group data points into discrete intervals called "bins." This process helps to analyze data distribution, visualize patterns, and prepare data for further analysis or modeling. We'll explore two common approaches to binning in Python using NumPy and Pandas libraries.

**Code**

**First Approach**

import numpy as np

# Generate random data and define bins

data = np.random.random(10000)

bins = np.linspace(0, 1, 11)

# Calculate means for each bin using digitize

digitized = np.digitize(data, bins)

bin\_means = [data[digitized == i].mean() for i in range(1, len(bins))]

print("\nDigitized Data Means:")

for b, mean in enumerate(bin\_means, 1):

    print(f'Bin: {b} -> {mean}')

# Calculate means using histogram

hist\_counts, \_ = np.histogram(data, bins)

weighted\_counts, \_ = np.histogram(data, bins, weights=data)

bin\_means2 = weighted\_counts / hist\_counts

print("\nHistogram Means:")

for b, mean in enumerate(bin\_means2, 1):

    print(f'Bin: {b} -> {mean}')

**Second approach**

import pandas as pd

# Define ages and bins

ages = [18, 23, 22, 25, 46, 45, 34, 87]

bins = [0, 18, 40, 60, 75, 100]

bin\_labels = ['Young', 'Adult', 'Senior', 'Existential Crisis', 'Dead']

# Categorize ages and print results

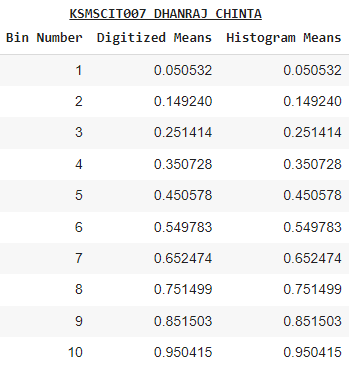
age\_bins = pd.cut(ages, bins, labels=bin\_labels)

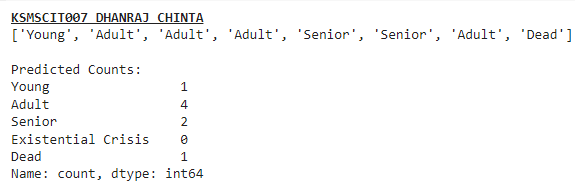
print(age\_bins.tolist())

print("\nPredicted Counts:")

print(age\_bins.value\_counts())

**Output**

****

****

**Location Coordinates Average**

**Code**

import numpy as np

num\_points = 10  # Number of random coordinates

latitudes = np.random.uniform(-90, 90, num\_points)

longitudes = np.random.uniform(-180, 180, num\_points)

# Create a DataFrame

coordinates\_df = pd.DataFrame({

    'Latitude': latitudes,

    'Longitude': longitudes

})

# Calculate the mean latitude and longitude

mean\_latitude = np.mean(latitudes)

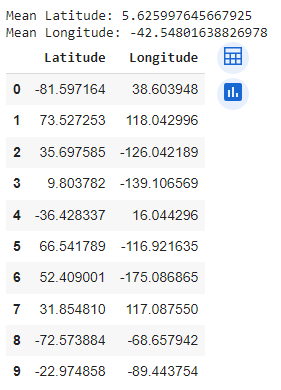
mean\_longitude = np.mean(longitudes)

print(coordinates\_df)

print("Mean Latitude:", mean\_latitude)

print("Mean Longitude:", mean\_longitude)

**Output**

****

**Random Coordinates and Plottig Using Basemap**

**Code**

import matplotlib.pyplot as plt

from mpl\_toolkits.basemap import Basemap

# Initialize the plot and Basemap

plt.figure(figsize=(10, 6))

m = Basemap(projection='cyl', resolution='l', llcrnrlat=-90, urcrnrlat=90, llcrnrlon=-180, urcrnrlon=180)

# Draw map features

m.drawcoastlines()

m.drawcountries()

m.drawmapboundary(fill\_color='aqua')

m.fillcontinents(color='lightgreen', lake\_color='aqua')

# Plot random and mean coordinates

m.scatter(longitudes, latitudes, color='red', marker='o', s=100, label='Random Points')

m.scatter(mean\_longitude, mean\_latitude, color='blue', marker='X', s=200, label='Mean Point')

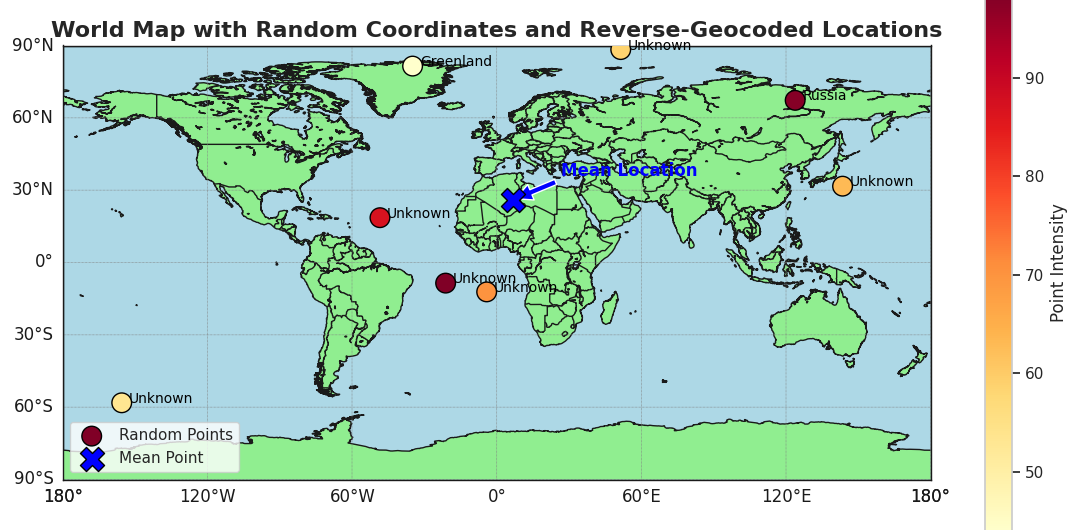
# Finalize plot

plt.legend()

plt.title('Random Coordinates and Mean Point on Base Map')

plt.show()

**Output**

****