A green and gold logo

Description automatically generated

**PRACTICAL JOURNAL**

in

**DATA SCIENCE IMPLEMENTATION**

Submitted by

**KSMSCIT005 HITESH BHANUSHALI**

for the award of the Degree of

**MASTERS 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. **HITESH VERSHI BHANUSHALI** for M.Sc. (IT) Part- II Semester- III, Seat No: **KSMSCIT005** 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**

**KISHINCHAND CHELLARAM COLLEGE, MUMBAI - 20**

**M.Sc (I.T.) Part-1 Semester II**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **TITLE** | **Date** | **Page No.** | **Signature** |
|  | **PRACTICALS** |  |  |  |
| **1** | Exploring Color Maps in Matplotlib: Visualizing Random Data with Different Color Schemes | 16/06/24 | 1-2 |  |
| **2** | Geospatial Visualization with GeoPandas | 30/06/24 | 3 |  |
| **3** | Interactive Geospatial Visualization with Folium: | 07/07/24 | 4-5 |  |
| **4** | Calculating and Visualizing the Current Position of the Moon Using Astropy | 28/07/24 | 6-7 |  |
| **5** | L Visualizing New COVID-19 Cases Using Plotly Express:   1. Daily COVID-19 Case Trends: A Line Plot Visualization 2. Monthly COVID-19 Case Trends: A Bar Plot Visualization | 28/0/24 | 8-10 |  |
| **6** | Linear Regression Analysis of Diabetes Data:   * Predicting Age from BMI | 11/08/24 | 11-12 |  |
| **7** | Creating a Word Cloud | 18/08/24 | 13 |  |
| **8** | Introduction to Cassandra: Basic Commnds , KeySpace | 18/08/24 | 14-15 |  |
| **9** | Using OpenCV and File Management Libraries:   1. Extracting Frames from Video: Converting MP4 to JPEG Images 2. Reconstructing a Video from Image Frames | 25/08/24 | 16-19 |  |
| **10** | Working with MongoDB: Python and R | 15/09/24 | 20-23 |  |
| **11** | Horus: Audio to CSV File Image to CSV File | 29/09/24 | 24-28 |  |
| **12** | Data Analysis: Binning, location, and random | 29/09/24 | 29-31 |  |
|  | **ASSIGNMENTS** |  |  |  |
| **1** | Assignment 1 | 18/07/24 |  |  |
| **2** | Assignment 2 | 25/08/24 |  |  |

**Practical No. 1**

**Aim:** Exploring Color Maps in Matplotlib: Visualizing Random Data with Different Color Schemes.

**Code:**

# Import the necessary libraries

import matplotlib.pyplot as plt # For creating plots

import numpy as np # For numerical operations, especially for generating random numbers

# Loop through the color maps, limiting to the first 5

for index, i in enumerate(plt.colormaps()):

if index >= 3: # Stop the loop after 5 iterations

break

# Set the title for each plot based on the color map name

sTitle = 'KSMSCIT005 Hitesh Bhanushali \n Color Map: ' + i

# Create a figure for the plot with a specific size

fig = plt.figure(figsize=(4, 4))

# Set the title for the plot

plt.title(sTitle)

# Generate a random 10x10 matrix and plot it as an image

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

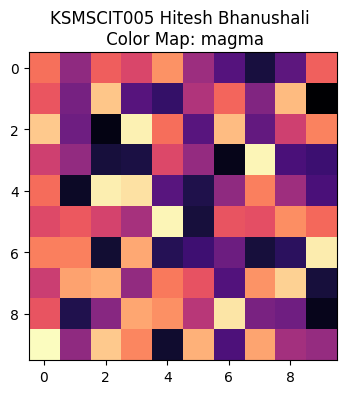
# Apply the current color map to the image plot

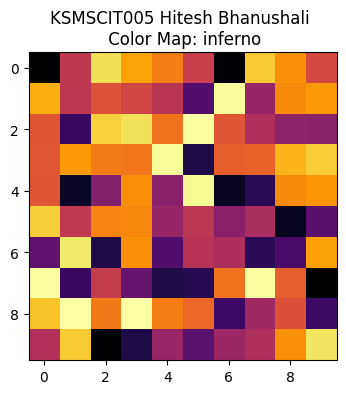
imgplot.set\_cmap(i)

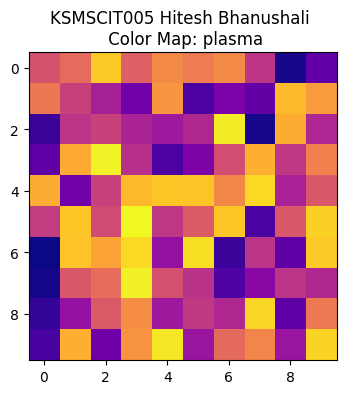
# Display the plot

plt.show()

**Output:**

****

****

****

**Practical No. 2**

**Aim:** Geospatial Visualization with GeoPandas

**Code:**

#Visualising Geospecial data with geopanda

import geopandas as gpd

import matplotlib.pyplot as plt

import fiona

from shapely.geometry import Point

# Set the SHAPE\_RESTORE\_SHX config option to YES

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

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

india\_gdf = gpd.read\_file("/content/sample\_data/indian\_borders\_for\_indian\_viewers.shp")

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

ax.text(x,y,label)

plt.title("KSMSCIT005 Hitesh Bhanushali ")

plt.show()

**Output:**

****

A map of india with red dots

Description automatically generated

**Practical No. 3**

**Aim:** Interactive Geospatial Visualization with Folium:

. Mapping Major Cities of India

**Code:**

import numpy as np

import pandas as pd

import folium

print(‘KSMSCIT005 Hitesh Bhanushali')

# Create a base map centered on India's geographical coordinates with a starting zoom level of 5

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

# List of cities with their name, geographic coordinates, and population

cities = [

{"name": "Tamil Nadu", "location": [11.1271, 78.6569], "population": "21.75 million"},

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

{"name": "Punjab", "location": [31.1471, 75.3412], "population": "8.42 million"},

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

{"name": "Uran", "location": [18.8772, 72.9283], "population": "14.85 million"}

# Loop through each city in the list to add a marker to the map

for city in cities:

    folium.Marker(

        location=city["location"],

        popup=f"<b>{city['name']}</b><br>Population: {city['population']}",

        tooltip=city['name']

    ).add\_to(rm)

# Generate random latitude and longitude points within India's approximate geographical bounds

# Latitude range: 6 to 35 (north to south India), Longitude range: 68 to 97 (west to east India)

latitudes = np.random.uniform(6, 35, 5)

longitudes = np.random.uniform(68, 97, 5)

# List of random village names for the generated points

village\_names = ['Village A', 'Village B', 'Village C', 'Village D', 'Village E']

# Loop through the random latitudes and longitudes to add markers for random villages

for lat, lon, village\_name in zip(latitudes, longitudes, village\_names):

    folium.Marker(

        location=[lat, lon],

        popup=f"<b>{village\_name}</b><br>Randomly Generated Location",

        tooltip=village\_name

    ).add\_to(rm)

# Save the map to an HTML file

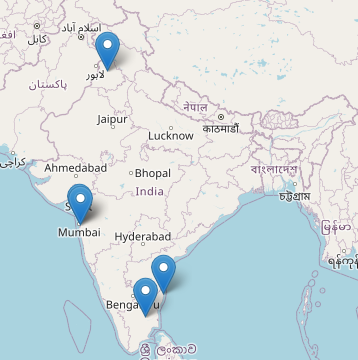
rm.save('india\_map\_with\_random\_villages.html')

# Display the map in a Jupyter notebook or similar environment (optional)

rm

**Output:**

****

****

**Practical No. 4**

**Aim:** Calculating and Visualizing the Current Position of the Moon Using Astropy

**Code:**

import warnings

warnings.filterwarnings('ignore')

from astropy.time import Time

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

import astropy.units as u

import matplotlib.pyplot as plt

print(‘KSMSCIT005 Hitesh Bhanushali')

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

# Set up the ephemeris and get the current time in UTC

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

time\_utc = Time.now()

# Calculate the Moon's current position in celestial coordinates

moon = get\_moon(time\_utc)

# Define the observer's location and transform the Moon's position to AltAz coordinates

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

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

# Print the Moon's Right Ascension (RA), Declination (Dec), Altitude, and Azimuth

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

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

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

# Create a polar plot to visualize the Moon's position in the sky

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

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

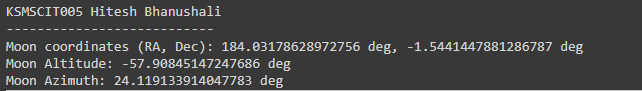
plt.title('\n KSMSCIT005 Hitesh Bhanushali \n Moon Position', y=1.1)

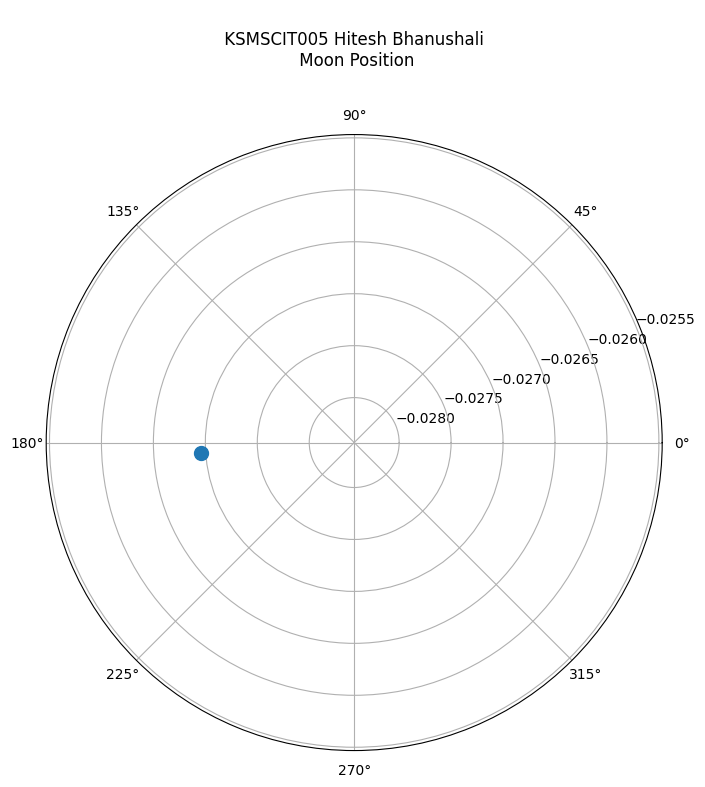
plt.polar(moon.ra.radian, moon.dec.radian, 'o', markersize=10)

plt.grid(True)

plt.show()

**Output:**

****



**Practical No. 5**

**Aim:** Visualizing New COVID-19 Cases Using Plotly Express:

. Daily COVID-19 Case Trends: A Line Plot Visualization

. Monthly COVID-19 Case Trends: A Bar Plot Visualization

**Code:**

**Part A:**

!pip install pandas plotly

import pandas as pd

import plotly.express as px

# URL for the COVID-19 data

URL = "https://covid.ourworldindata.org/data/owid-covid-data.csv"

# Load the dataset into a DataFrame

df = pd.read\_csv(URL)

# Filter the DataFrame for the specific country

country = 'Germany'

df\_country = df[df['location'] == country]

# Select relevant columns for the plot

df\_country = df\_country[['date', 'new\_cases']]

# Create a line plot to visualize COVID-19 new cases over time for the selected country

fig = px.line(df\_country, x='date', y='new\_cases', title=f'KSMSCIT005 Hitesh Bhanushali | Corona Cases in {country} over time')

# Show the plot

fig.show()

**Part B:**

import pandas as pd

import plotly.express as px

# URL for the COVID-19 data

URL = "https://covid.ourworldindata.org/data/owid-covid-data.csv"

# Load the dataset into a DataFrame

df = pd.read\_csv(URL)

# Define the country of interest

country = 'Germany'

# Filter the DataFrame for the specific country

df\_country = df[df['location'] == country]

# Check if the filtered DataFrame is empty and raise an error if so

if df\_country.empty:

    raise ValueError(f'Country {country} not found in the dataset')

# Convert the 'date' column to datetime format

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

# Extract the month from the date and create a new column for it

df\_country['month'] = df\_country['date'].dt.to\_period('M')

# Aggregate the data to get the total number of new cases per month

monthly\_cases = df\_country.groupby('month')['new\_cases'].sum().reset\_index()

# Convert the 'month' period to a string for plotting

monthly\_cases['month'] = monthly\_cases['month'].astype(str)

# Create a bar plot to visualize the total number of new COVID-19 cases per month

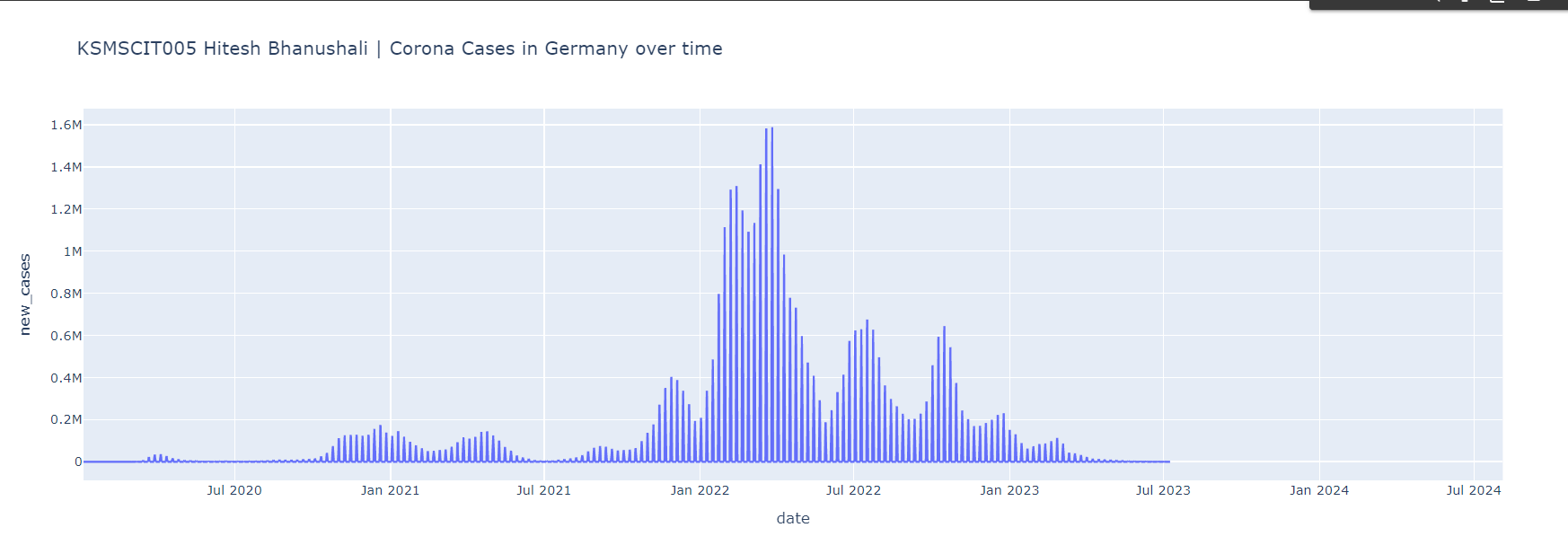
fig = px.bar(monthly\_cases, x='month', y='new\_cases', title=f'KSMSCIT005 Hitesh Bhanushali | Total Corona Cases in {country} over time')

# Show the plot

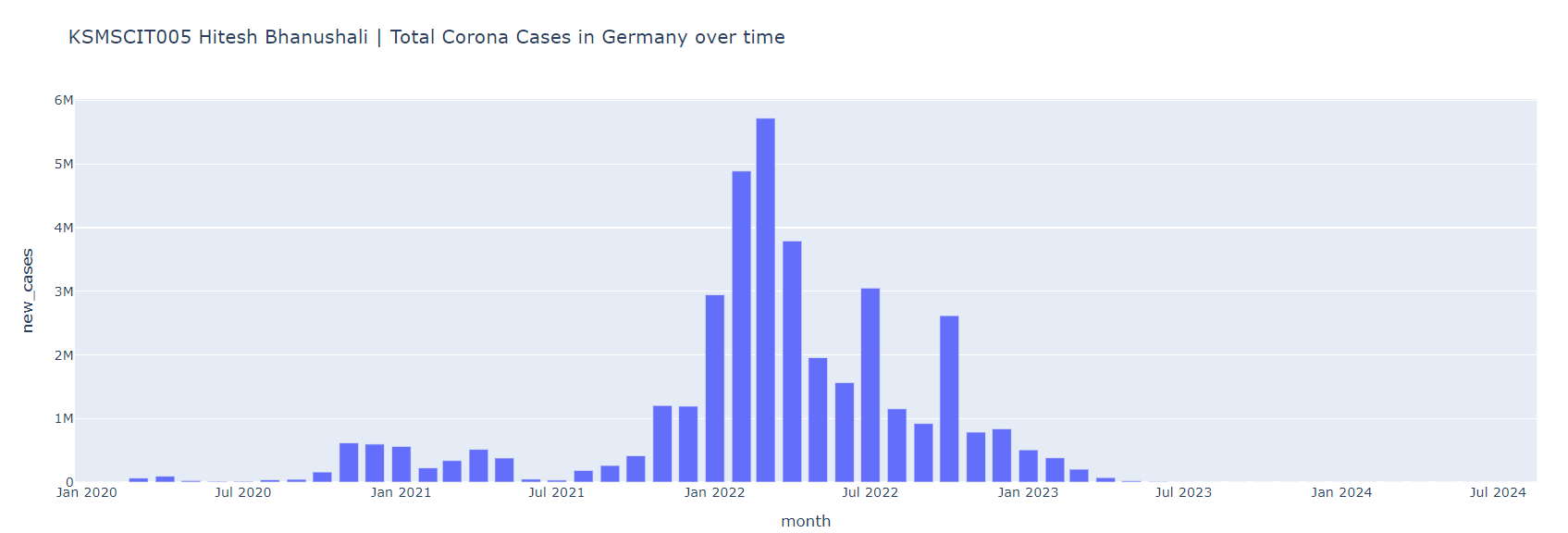
fig.show()

**Output:**

**Part A:**

****

**Part B:**

****

**Practical No. 6**

**Aim:** Linear Regression Analysis of Diabetes Data:

. Predicting Age from BMI

**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

print("KSMSCIT005 Hitesh Bhanushali")

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

# 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:]

print("BMI:",diabetes\_X\_test)

# 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))

# 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')

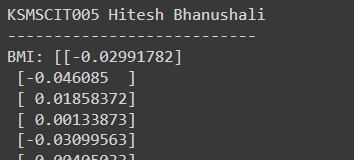
plt.title("KSMSCIT005 Hitesh Bhanushali \n Diabetes")

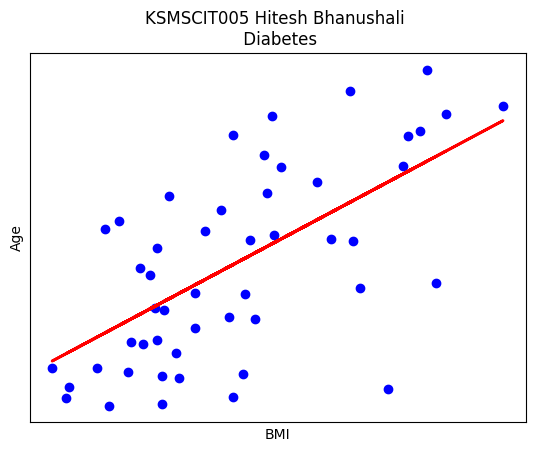
plt.xlabel("BMI")

plt.ylabel("Age")

plt.show()

**Output:**

****



**Practical No 7**

**Aim:** Creating a Word Cloud

**Code:**

from wordcloud import WordCloud

import matplotlib.pyplot as plt

# Sample text related to data science

text = """

data science is an interdisciplinary field that uses scientific methods processes algorithms and systems to extract knowledge….

"""

# Generate a word cloud from the text

wordcloud = WordCloud(width=800, height=800, background\_color='white').generate(text)

# Plot the word cloud

plt.figure(figsize=(5,5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()

**Output:**



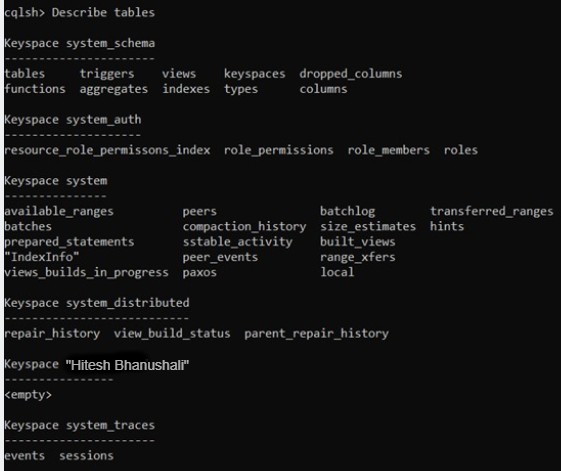
**Practical No. 8**

**Aim:** Introduction to Cassandra:

. Basic Commands

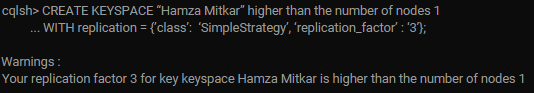
. KeySpace Creation

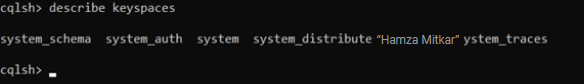
**Implementation:**

****

A screenshot of a computer program

Description automatically generated





A black background with white text

Description automatically generated

A black screen with white text

Description automatically generated

**Practical No. 9**

**Aim:** Using OpenCV and File Management Libraries:

. Extracting Frames from Video: Converting MP4 to JPEG Images

. Reconstructing a Video from Image Frames

**Code:**

**Part A:**

import os

import shutil

import cv2

print("KSMSCIT005 Hitesh Bhanushali")

# Define input file and output directory

sInputFileName = '/content/sample.mp4'

sDataBaseDir = '/content/Video to Images'

# Remove the output directory if it already exists, and create a new one

if os.path.exists(sDataBaseDir):

    shutil.rmtree(sDataBaseDir)

if not os.path.exists(sDataBaseDir):

    os.makedirs(sDataBaseDir)

# Notify the user that the process is starting

print('=====================================================')

print('Start Movie to Frames')

print('=====================================================')

# Open the video file

vidcap = cv2.VideoCapture(sInputFileName)

success, image = vidcap.read()

count = 0

# Read and process frames from the video

while success:

    success, image = vidcap.read()  # Read the next frame

    if not success:

        break

    # Define the filename for the extracted frame

    sFrame = sDataBaseDir + str('/pic-frame-' + str(format(count, '04d')) + '.jpg')

    print('Extracted: ', sFrame)

    # Save the frame as a JPEG file

    cv2.imwrite(sFrame, image)

    # Check if the saved frame is empty, and remove it if so

    if os.path.getsize(sFrame) == 0:

        count -= 1  # Decrement the frame count

        os.remove(sFrame)  # Remove the empty frame

        print('Removed: ', sFrame)

    # Exit if the Escape key is pressed

    if cv2.waitKey(10) == 27:

        break

    # Exit after processing a certain number of frames (e.g., 15)

    if count > 100:

        break

    # Increment the frame count

    count += 1

# Notify the user that the process is complete

print('=====================================================')

print('Generated : ', count, ' Frames')

print('=====================================================')

print('Movie to Frames HORUS - Done')

print('=====================================================')

**Part B:**

#9 - B

import cv2

import os

print("KSMSCIT005 Hitesh Bhanushali")

# Define the directory containing the images and the path for the output video

sDataBaseDir = "/content/Video to Images"

output\_video\_file = "/content/Image to Video/OutputVideo.mp4"

# Ensure the output directory exists

os.makedirs(os.path.dirname(output\_video\_file), exist\_ok=True)

# List all image files in the directory and sort them

frame\_files = [f for f in os.listdir(sDataBaseDir) if f.endswith('.jpg')]

frame\_files.sort()  # Ensure images are processed in the correct order

# Check if there are any images to process

if not frame\_files:

    print("No frames found in the directory.")

    exit()

# Read the first frame to get the dimensions for the video

first\_frame\_path = os.path.join(sDataBaseDir, frame\_files[0])

first\_frame = cv2.imread(first\_frame\_path)

if first\_frame is None:

    print(f"Error reading the first frame: {first\_frame\_path}")

    exit()

# Get the height and width of the frames

height, width, \_ = first\_frame.shape

# Define the codec and create a VideoWriter object

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')  # Codec for MP4 format

fps = 30  # Frames per second

out = cv2.VideoWriter(output\_video\_file, fourcc, fps, (width, height))

print("===================================================")

print("Creating video from frames")

print("===================================================")

# Write each frame to the video file

for frame\_file in frame\_files:

    frame\_path = os.path.join(sDataBaseDir, frame\_file)

    frame = cv2.imread(frame\_path)

    if frame is not None:

        out.write(frame)

        print(f"Added frame: {frame\_path}")

    else:

        print(f"Failed to read frame: {frame\_path}")

# Release the VideoWriter object

out.release()

print("=====================================================")

print(f'Video saved as: {output\_video\_file}')

print("=====================================================")

**Output:**

**Part A:**

****

**A screenshot of a computer

Description automatically generated**

**A text on a white background

Description automatically generated**

**Part B:**

****

**A screenshot of a computer

Description automatically generated**

**A close up of text

Description automatically generated**

**Practical No. 10**

**Aim:** Working with MongoDB:

. Python

. R

**Code:**

**Part A:**

# MongoDB Atlas URI connection string

ATLAS\_URI = "mongodb://localhost:27017"

# Install pymongo library with the srv extra required for MongoDB Atlas

#! pip install pymongo[srv]==4.6.2

# Import MongoClient from the pymongo library

from pymongo import MongoClient

# Define a class to interact with MongoDB Atlas

class AtlasClient:

    # Constructor to initialize the MongoClient and select a specific database

    def \_\_init\_\_(self, atlas\_uri, dbname):

        # Connect to the MongoDB Atlas instance using the URI

        self.mongodb\_client = MongoClient(atlas\_uri)

        # Set the database we will be working with

        self.database = self.mongodb\_client[dbname]

    # Quick method to check if the connection to MongoDB Atlas is successful

    def ping(self):

        # Sends a ping command to the server to ensure it's reachable

        self.mongodb\_client.admin.command('ping')

    # Method to get a specific collection from the database

    def get\_collection(self, collection\_name):

        # Return the collection specified by 'collection\_name'

        collection = self.database[collection\_name]

        return collection

    # Method to find documents in a collection with an optional filter and limit

    def find(self, collection\_name, filter={}, limit=0):

        # Get the specified collection

        collection = self.database[collection\_name]

        # Perform a query on the collection, return as a list

        items = list(collection.find(filter=filter, limit=limit))

        return items

# Database name and collection name we want to interact with

DB\_NAME = 'Practice'

COLLECTION\_NAME = 'UserList'

# Create an instance of AtlasClient using the MongoDB URI and database name

atlas\_client = AtlasClient(ATLAS\_URI, DB\_NAME)

# Ping the MongoDB Atlas instance to test the connection

atlas\_client.ping()

print('Connected to Atlas instance! We are good to go!')

# Retrieve all documents from the 'UserList' collection

names = atlas\_client.find(collection\_name=COLLECTION\_NAME)

print(f"Found {len(names)} names")

# Loop through each document (name) found and print details (id, name, address)

for idx, name in enumerate(names):

    print(f'{idx+1}\nid: {name["\_id"]}\nname: {name["name"]},\naddress: {name["Rollno"]}')

**Part B:**

# Install the mongolite library

# install.packages("mongolite")

# Load the mongolite library

library(mongolite)

# MongoDB Atlas URI connection string

ATLAS\_URI <- "mongodb://localhost:27017"

# Define a function to create an object that interacts with MongoDB Atlas

AtlasClient <- function(atlas\_uri, dbname) {

  client <- list()

  # Connect to the MongoDB Atlas instance using the URI

  client$mongodb\_client <- mongo(url = atlas\_uri)

  # Set the database we will be working with

  client$database <- function(collection\_name) {

    mongo(collection = collection\_name, db = dbname, url = atlas\_uri)

  }

  # Quick method to check if the connection to MongoDB Atlas is successful

  client$ping <- function() {

    tryCatch({

      client$mongodb\_client$run('{"ping": 1}')

      print("Ping successful")

    }, error = function(e) {

      print(paste("Ping failed:", e$message))

    })

  }

  # Method to find documents in a collection with an optional filter and limit

  client$find <- function(collection\_name, filter = '{}', limit = 0) {

    collection <- client$database(collection\_name)

    result <- collection$find(query = filter, limit = limit)

    return(result)

  }

  return(client)

}

# Database name and collection name we want to interact with

DB\_NAME <- 'Practice'

COLLECTION\_NAME <- 'UserList'

# Create an instance of AtlasClient using the MongoDB URI and database name

atlas\_client <- AtlasClient(ATLAS\_URI, DB\_NAME)

# Ping the MongoDB Atlas instance to test the connection

atlas\_client$ping()

# Retrieve all documents from the 'UserList' collection

names <- atlas\_client$find(collection\_name = COLLECTION\_NAME)

# Print how many documents (names) were found

print(paste("Found", nrow(names), "names"))

# Loop through each document (name) found and print details (id, name, address)

for (idx in 1:nrow(names)) {

  name <- names[idx, ]

  cat(paste0(idx, "\nid: ", name$`\_id`, "\nname: ", name$Name, ",\naddress: ", name$Rollno, "\n\n"))

}

**Output:**

**Part A:**

**A screenshot of a computer

Description automatically generated**

**Part B:**

**A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated**

**Practical No. 11**

**Aim:** Horus:

. Audio to CSV File

. Image to CSV File

**Code:**

**Part A:**

from scipy.io import wavfile

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

print("KSMSCIT005 Hitesh Bhanushali")

# Display audio file info and plot the audio signal

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

    print(f"Audio: {aname}\nRate: {r}\nShape: {a.shape}")

    plot\_info(aname, a, r)

# Plot the audio signal for each channel

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

    plt.title(f'Signal Wave - {aname} at {r}hz')

    sLegend = []

    for c in range(a.shape[1]):

        sLabel = 'Ch' + str(c + 1)

        sLegend.append(sLabel)

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

    plt.legend(sLegend)

    plt.show()

sInputFileName = '/content/4ch-sound.wav'

print('Processing: ', sInputFileName)

# Read audio file

InputRate, InputData = wavfile.read(sInputFileName)

show\_info("4 channel", InputData, InputRate)

# Convert audio data to DataFrame

ProcessData = pd.DataFrame(InputData)

ProcessData.columns = ['Ch1', 'Ch2', 'Ch3', 'Ch4']

# Save DataFrame to CSV

sOutputFileName = '/content/Output/AudioToCSV.csv'

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

print(ProcessData)

**Part B:**

from PIL import Image

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

print("KSMSCIT005 Hitesh Bhanushali")

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

    # Open and convert image to RGB

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

    # Convert image to NumPy array

    img\_array = np.array(img)

    # Print image info

    print(f"Image Path: {image\_path}")

    print(f"Shape: {img\_array.shape}")

    print(f"Dtype: {img\_array.dtype}")

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

    # Reshape array to 2D (rows of pixel values) and convert to DataFrame

    df = pd.DataFrame(img\_array.reshape(-1, 3), columns=['R', 'G', 'B'])

    # Save DataFrame to CSV

    df.to\_csv(csv\_output\_path, index=False, header=False)  # Avoid header and index

    # Show the image

    plt.imshow(img\_array)

    plt.title('Image Preview')

    plt.axis('off')  # Hide axis

    plt.show()

def visualize\_csv(csv\_path, image\_shape):

    df = pd.read\_csv(csv\_path, header=None, names=['R', 'G', 'B'])

    # Reshape DataFrame to image shape (height, width, 3)

    img\_array = df.values.reshape(image\_shape)

    # Plot a horizontal strip of the image (e.g., the middle row) for each channel

    mid\_row = img\_array.shape[0] // 2

    plt.figure(figsize=(15, 5))

    # Plot Red channel

    plt.subplot(3, 1, 1)

    plt.plot(img\_array[mid\_row, :, 0], color='red')

    plt.title('Red Channel Pixel Values')

    # Plot Green channel

    plt.subplot(3, 1, 2)

    plt.plot(img\_array[mid\_row, :, 1], color='green')

    plt.title('Green Channel Pixel Values')

    # Plot Blue channel

    plt.subplot(3, 1, 3)

    plt.plot(img\_array[mid\_row, :, 2], color='blue')

    plt.title('Blue Channel Pixel Values')

    plt.tight\_layout()

    plt.show()

# Define file paths

image\_path = '/content/p11#2.jpg'  # Replace with your image path

csv\_output\_path = '/content/Output/ImageToCSV.csv'  # Replace with your desired CSV output path

# Get image array and shape

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

shape = img\_array.shape

# Process image and save as CSV

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

# Visualize data from CSV

visualize\_csv(csv\_output\_path, shape)

**Output:**

**Part A:**

****

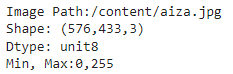
**A screen shot of a graph

Description automatically generated**

**Part B:**

****

**A screenshot of a computer

Description automatically generated** ****

**A screenshot of a computer

Description automatically generated**

**Practical No. 12**

**Aim :** Data analysis and Visualization

**Part A**

**Code:**

print("KSMSCIT005 Hitesh Bhanushali")

import pandas as pd

ages=[18,23,22,25,46,34,45,87,100,6]

bins=[0,25,50,75,100]

bin\_label=["Young","Mid","Senior","Old"]

age\_bin=pd.cut(ages,bins=bins,labels=bin\_label,right=True)

print(age\_bin)

**Output:**



**Part B**

**Code:**

import numpy as np

import pandas as pd

print('Latitude')

print(latitudeset)

print('Latitude avg',latitudeavg)

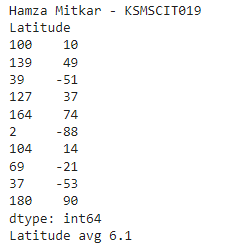
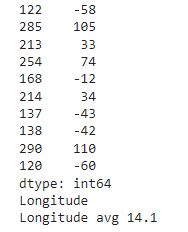
print("==============================")

print(longitudeset)

print('Longitude')

print('Longitude avg',longitudeavg)

**Output:**

**** ****

**Part C**

**Code:**

print("KSMSCIT005 Hitesh Bhanushali")

import matplotlib.pyplot as plt

!pip install basemap

from mpl\_toolkits.basemap import Basemap

# Plotting on a world map

plt.figure(figsize=(12, 6))

map = Basemap(projection='mill', llcrnrlat=-60, urcrnrlat=90,

llcrnrlon=-180, urcrnrlon=180, resolution='c')

map.drawcoastlines()

map.drawcountries()

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

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

# Convert latitude and longitude to map projection coordinates

x, y = map(longitudeset.values, latitudeset.values)

# Plot the sampled points

map.scatter(x, y, marker='o', color='red', s=100, label='Sampled Points')

# Plot the average point

avg\_x, avg\_y = map(longitudeavg, latitudeavg)

map.scatter(avg\_x, avg\_y, marker='X', color='blue', s=200, label='Average Point')

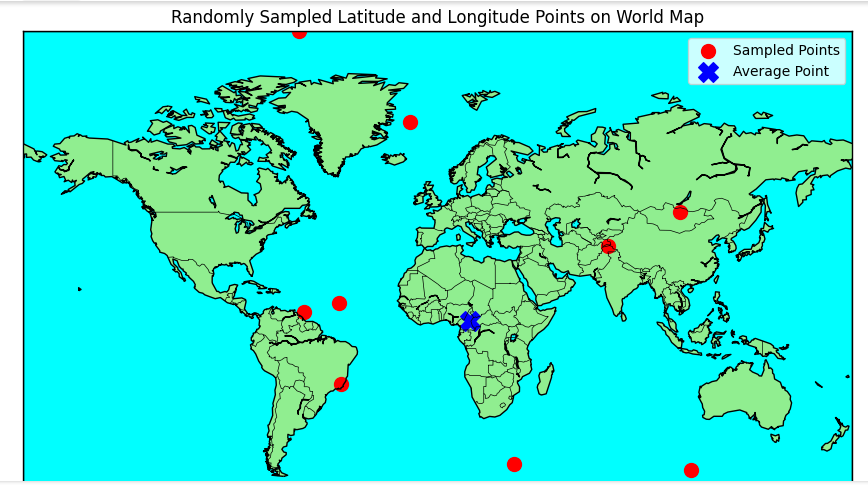
# Add title and legend

plt.title('Randomly Sampled Latitude and Longitude Points on World Map')

plt.legend()

plt.show()

**Output:**

****