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

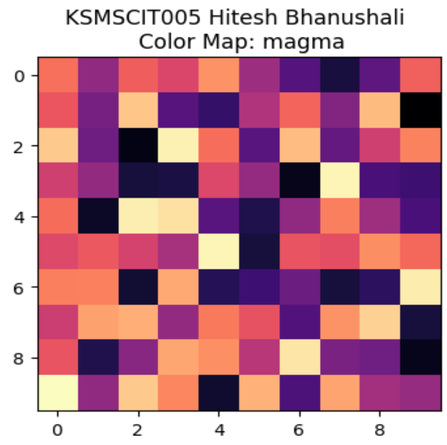
Practical No. 1

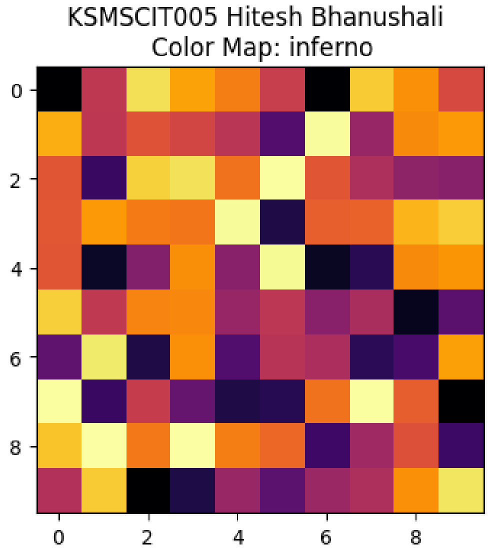
Aim: Exploring Color Maps in Matplotlib: Visualizing Random Data with Different Color Schemes.

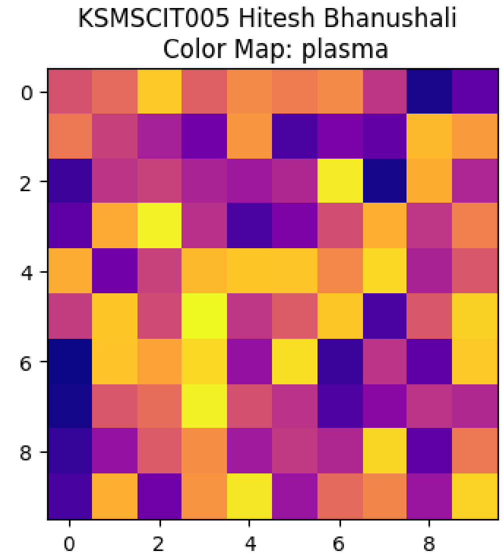
Code:

```
# 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:


KSMSCIT005 Hitesh Bhanushali
Color Map: magma
0
2
4
6
8
0
2
4
6
8


KSMSCIT005 Hitesh Bhanushali
Color Map: inferno
0
2
4
6
8
0
2
4
6
8



Practical No. 2

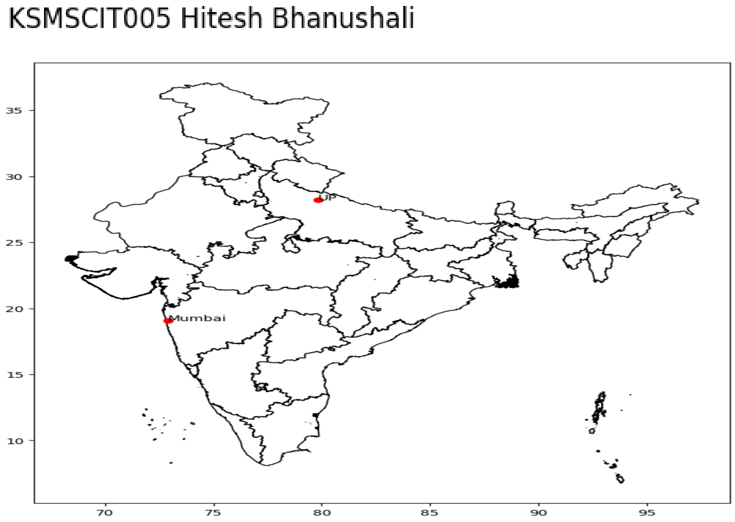
Aim: Geospatial Visualization with GeoPandas

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:



Practical No. 3

Aim: Interactive Geospatial Visualization with Folium:
. Mapping Major Cities of India

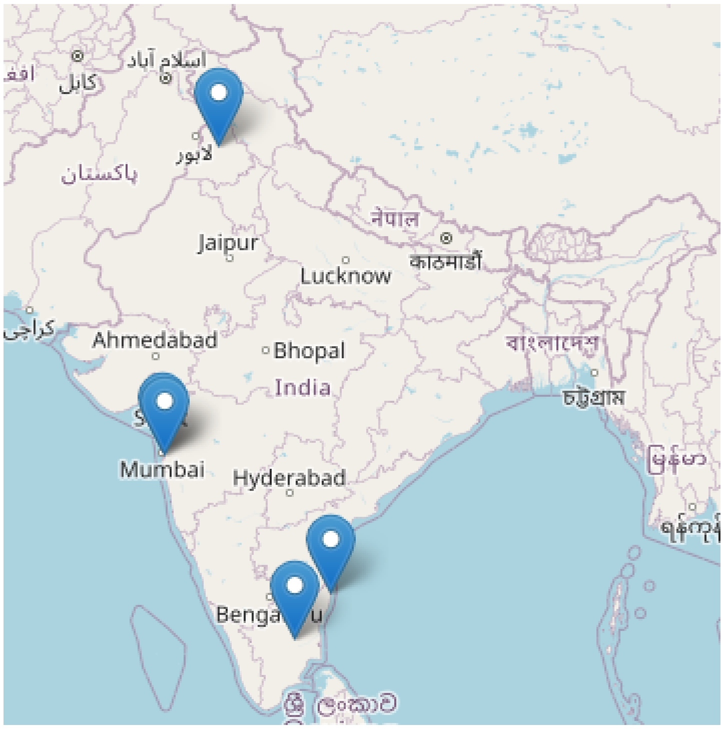
Code:

```
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:

KSMSCIT005 Hitesh Bhanushali



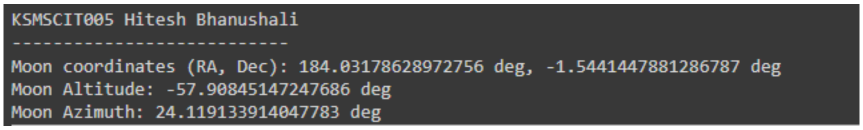
Practical No. 4

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

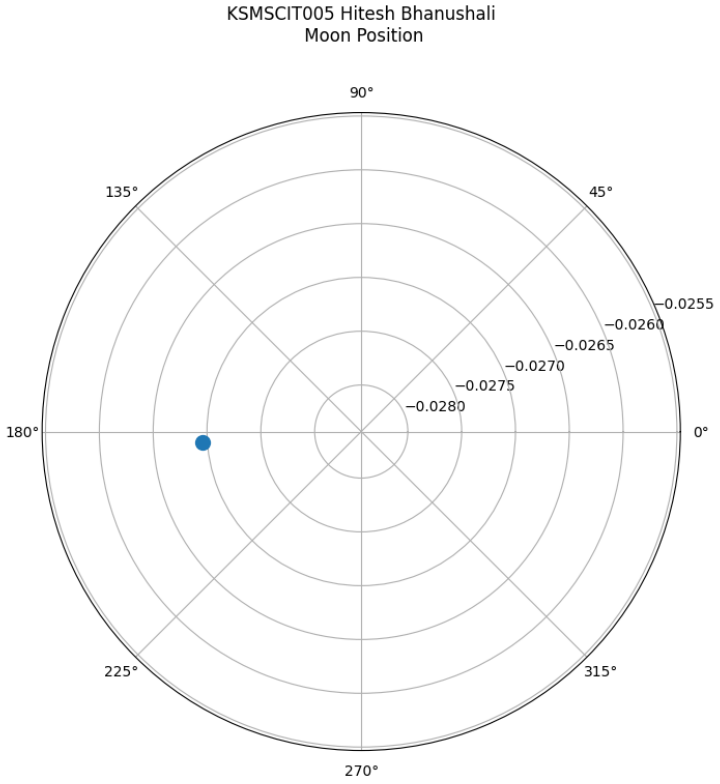
Code:

```
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:



```
KSMSCIT005 Hitesh Bhanushali
---------------------------
Moon coordinates (RA, Dec): 184.03178628972756 deg, -1.5441447881286787 deg
Moon Altitude: -57.90845147247686 deg
Moon Azimuth: 24.119133914047783 deg
```



KSMSCIT005 Hitesh Bhanushali
Moon Position

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()
```

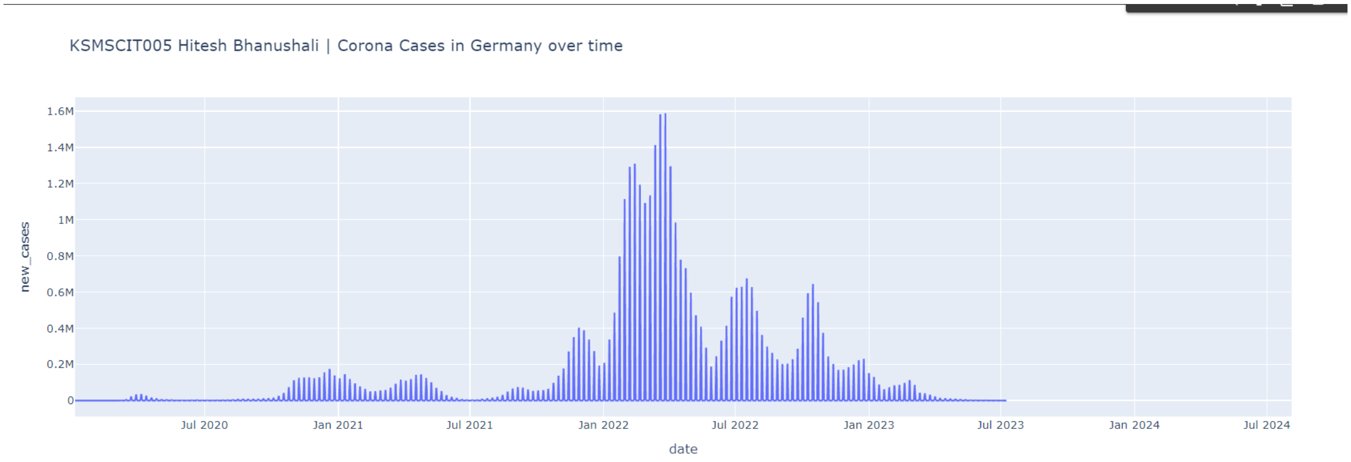
Part B:

```
 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
```

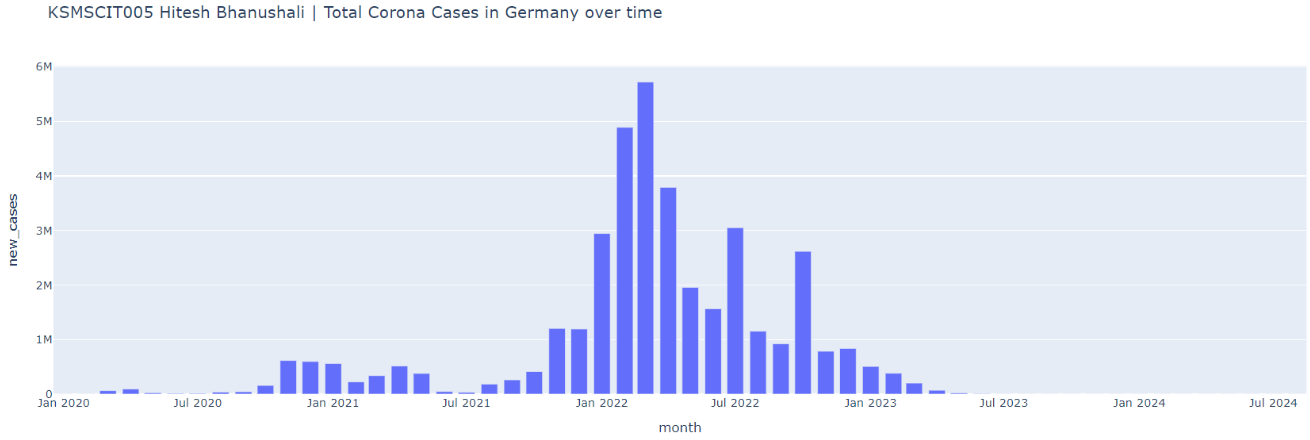
```
df = pd.read_csv(URL)

# 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

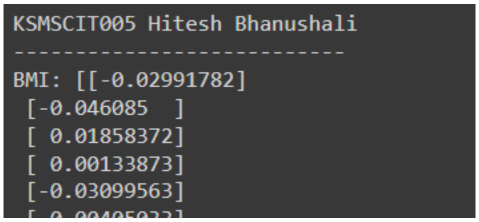
Aim: Linear Regression Analysis of Diabetes Data:
. Predicting Age from BMI

Code:

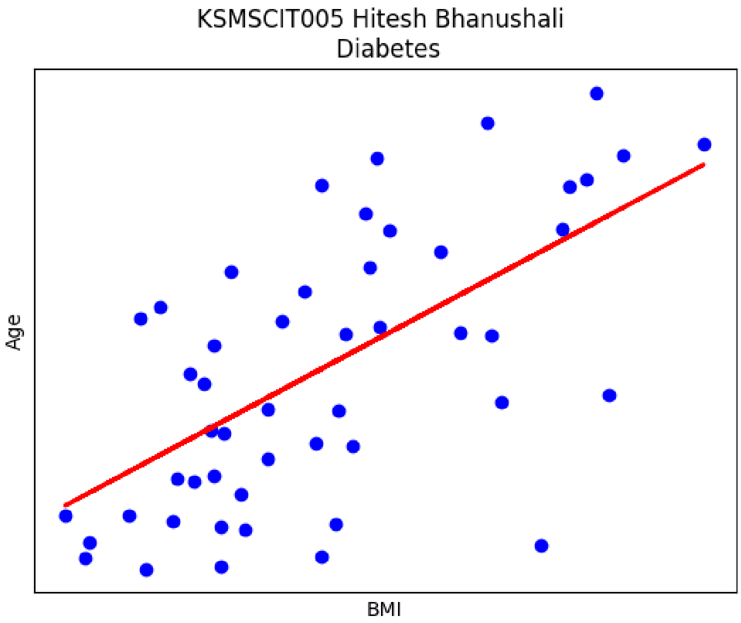
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
print("KSMSCIT005 Hitesh Bhanushali")
print("---------------------------")

# 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:



```
KSMSCIT005 Hitesh Bhanushali
----------------------------
BMI: [[-0.02991782]
 [-0.046085  ]
 [ 0.01858372]
 [ 0.00133873]
 [-0.03099563]
 [ 0.00405033]
```



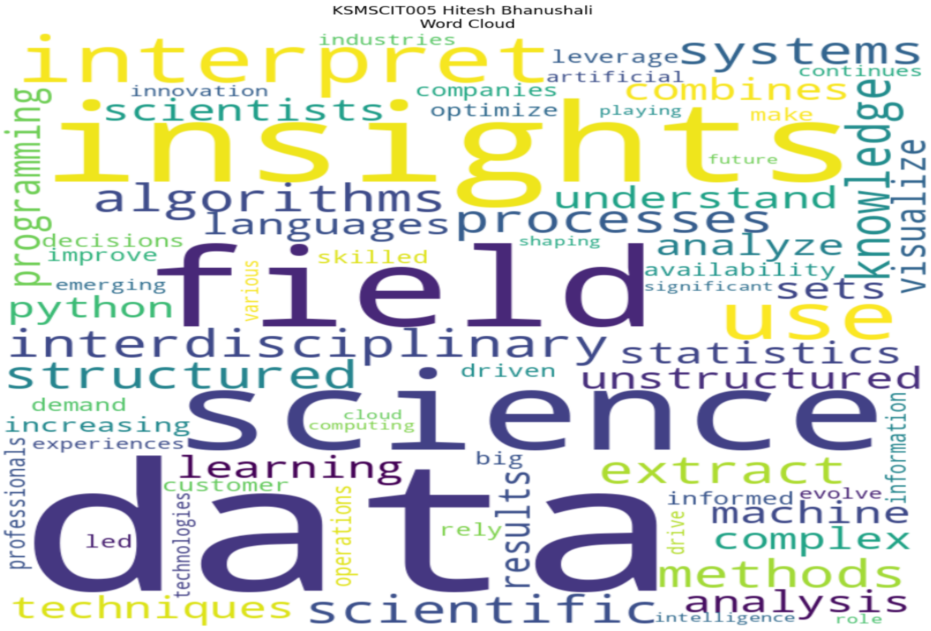
Practical No 7

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()
```

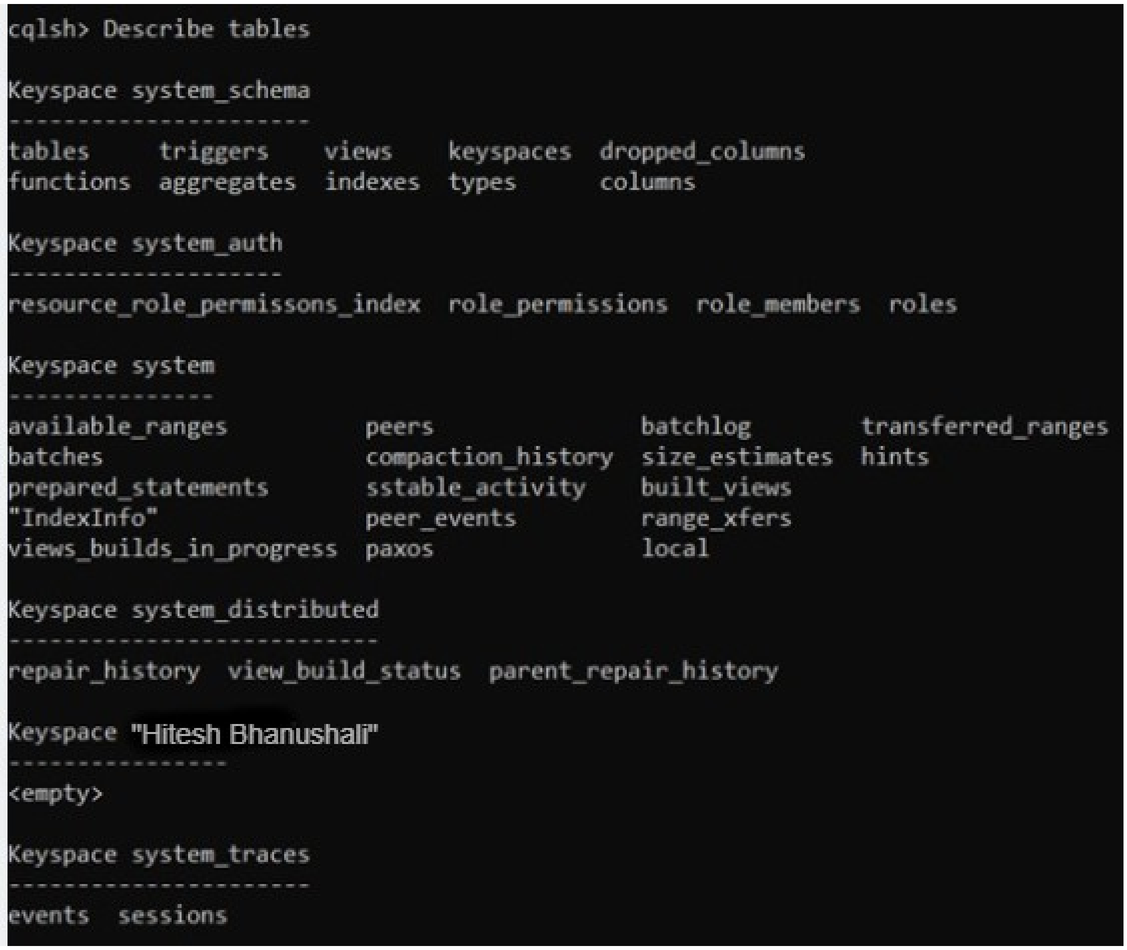
Output:



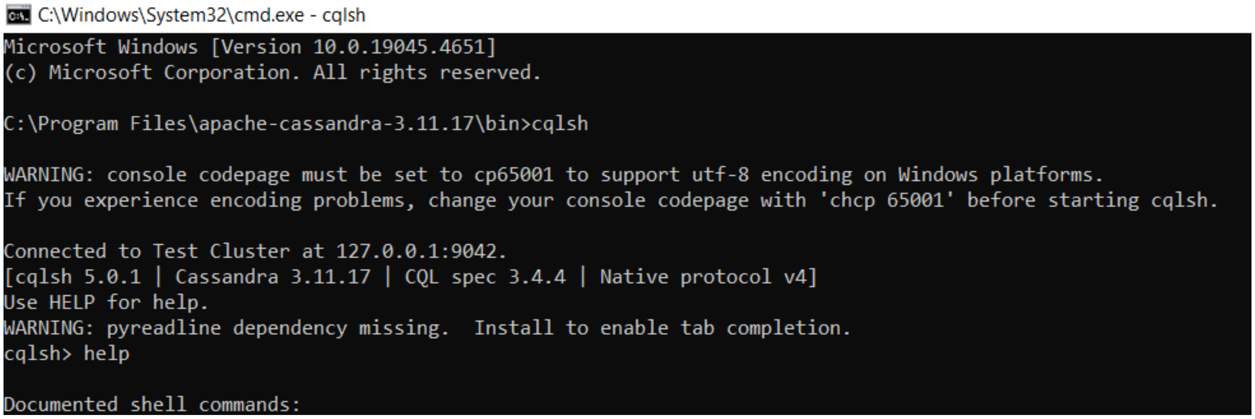
Practical No. 8

Aim: Introduction to Cassandra:
. Basic Commands
. KeySpace Creation

Implementation:

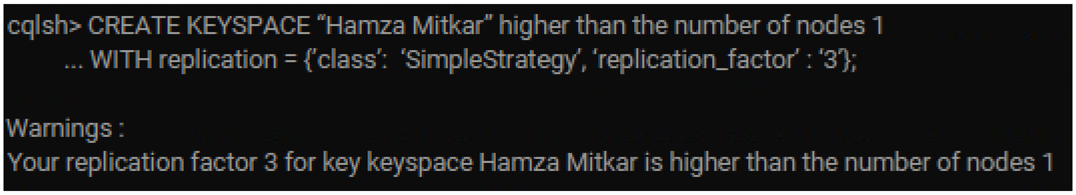


```
cqlsh> Describe tables

Keyspace system_schema
----------------------
tables     triggers    views    keyspaces  dropped_columns
functions  aggregates  indexes  types      columns

Keyspace system_auth
--------------------
resource_role_permissons_index  role_permissions  role_members  roles

Keyspace system
---------------
available_ranges          peers               batchlog        transferred_ranges
batches                   compaction_history  size_estimates  hints
prepared_statements       sstable_activity    built_views
"IndexInfo"               peer_events         range_xfers
views_builds_in_progress  paxos               local

Keyspace system_distributed
---------------------------
repair_history  view_build_status  parent_repair_history

Keyspace "Hitesh Bhanushali"
----------------
<empty>

Keyspace system_traces
----------------------
events  sessions
```

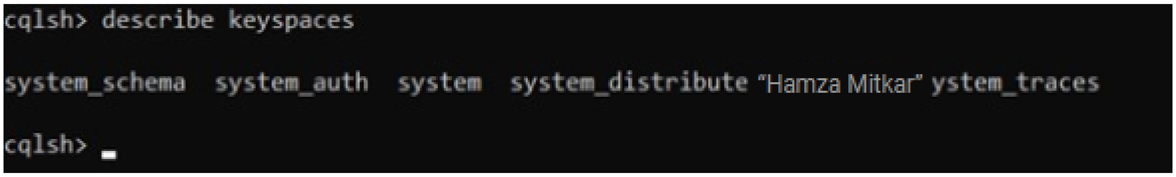


C:\Windows\System32\cmd.exe - cqlsh

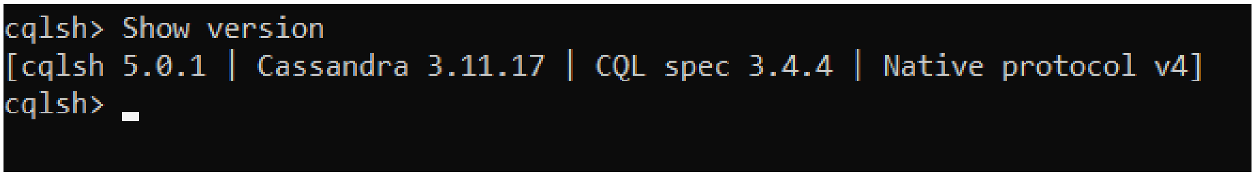
```
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\apache-cassandra-3.11.17\bin>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.17 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh> help

Documented shell commands:
```



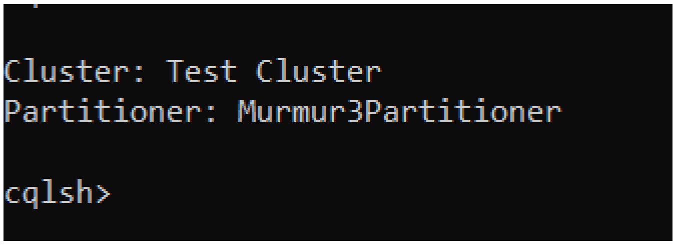
```
cqlsh> CREATE KEYSPACE “Hamza Mitkar” higher than the number of nodes 1
     ... WITH replication = {‘class’:  ‘SimpleStrategy’, ‘replication_factor’ : ‘3’};

Warnings :
Your replication factor 3 for key keyspace Hamza Mitkar is higher than the number of nodes 1
```



```
cqlsh> describe keyspaces

system_schema  system_auth  system  system_distribute “Hamza Mitkar” ystem_traces

cqlsh>
```



```
cqlsh> Show version
[cqlsh 5.0.1 | Cassandra 3.11.17 | CQL spec 3.4.4 | Native protocol v4]
cqlsh>
```



```
Cluster: Test Cluster
Partitioner: Murmur3Partitioner

cqlsh>
```

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:

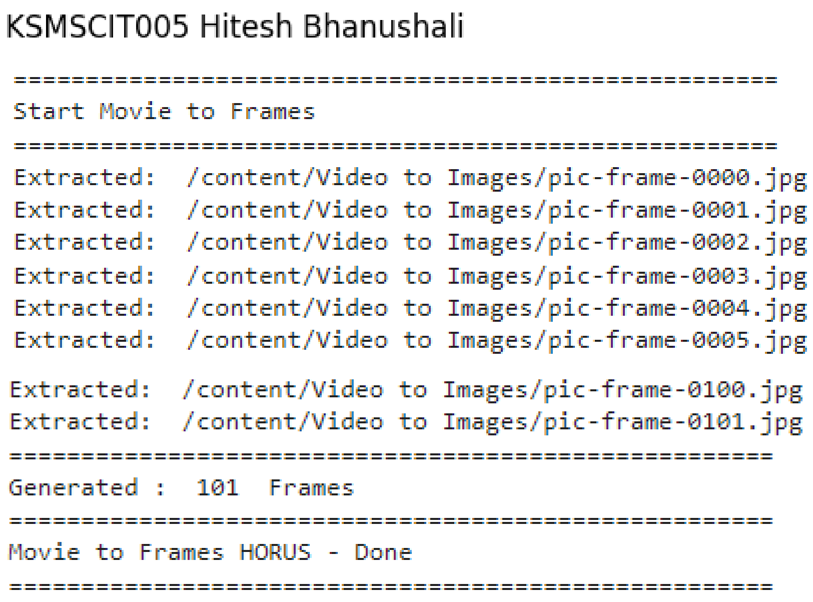
```
#9 - B
import cv2
import os
print("KSMSCIT005 Hitesh Bhanushali")

# 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:

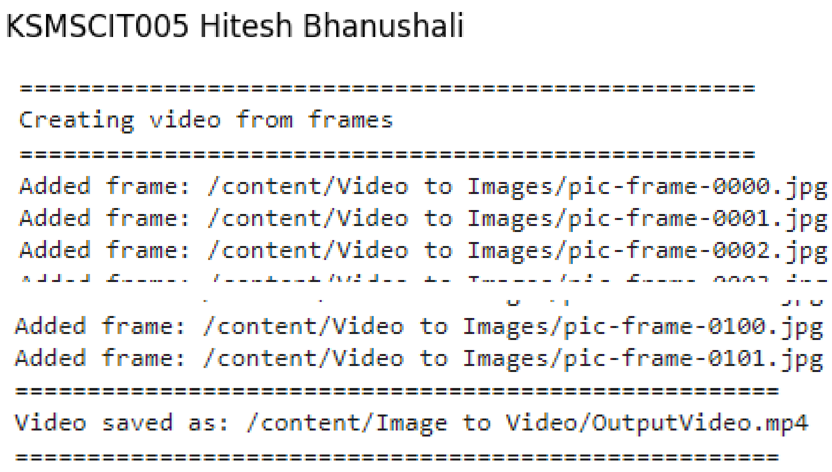
Part A:



KSMSCIT005 Hitesh Bhanushali

```
=====================================================
Start Movie to Frames
=====================================================
Extracted:  /content/Video to Images/pic-frame-0000.jpg
Extracted:  /content/Video to Images/pic-frame-0001.jpg
Extracted:  /content/Video to Images/pic-frame-0002.jpg
Extracted:  /content/Video to Images/pic-frame-0003.jpg
Extracted:  /content/Video to Images/pic-frame-0004.jpg
Extracted:  /content/Video to Images/pic-frame-0005.jpg

Extracted:  /content/Video to Images/pic-frame-0100.jpg
Extracted:  /content/Video to Images/pic-frame-0101.jpg
=====================================================
Generated :  101  Frames
=====================================================
Movie to Frames HORUS - Done
=====================================================
```

Part B:



KSMSCIT005 Hitesh Bhanushali

```
===================================================
Creating video from frames
===================================================
Added frame: /content/Video to Images/pic-frame-0000.jpg
Added frame: /content/Video to Images/pic-frame-0001.jpg
Added frame: /content/Video to Images/pic-frame-0002.jpg
Added frame: /content/Video to Images/pic-frame-0003.jpg
Added frame: /content/Video to Images/pic-frame-0100.jpg
Added frame: /content/Video to Images/pic-frame-0101.jpg
=====================================================
Video saved as: /content/Image to Video/OutputVideo.mp4
=====================================================
```

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))
```

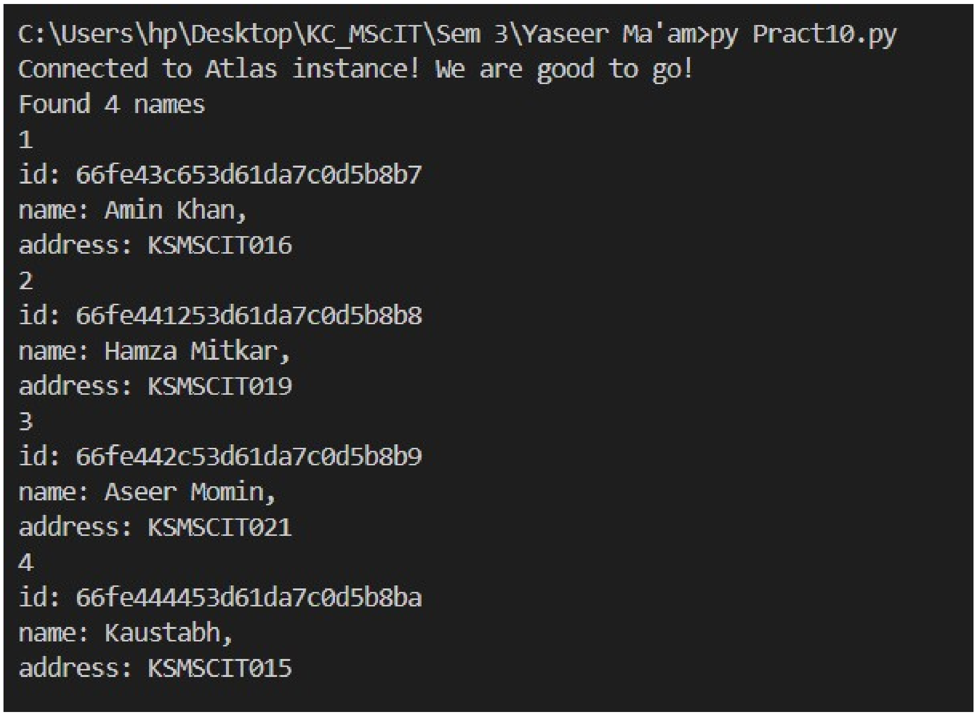
```
        return items

# 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"))
}
```

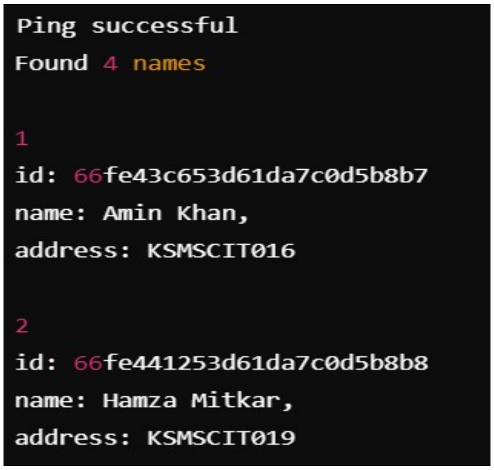
Output:

Part A:

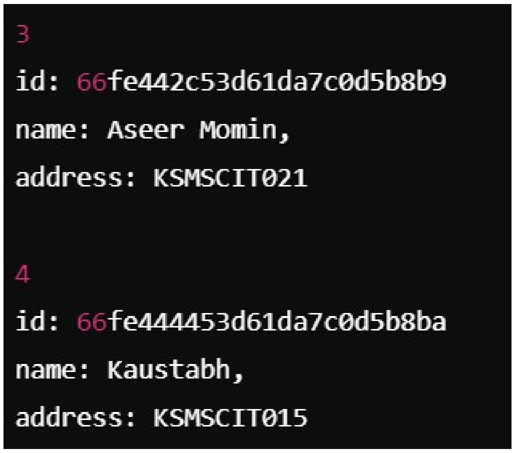


```
C:\Users\hp\Desktop\KC_MScIT\Sem 3\Yaseer Ma'am>py Pract10.py
Connected to Atlas instance! We are good to go!
Found 4 names
1
id: 66fe43c653d61da7c0d5b8b7
name: Amin Khan,
address: KSMSCIT016
2
id: 66fe441253d61da7c0d5b8b8
name: Hamza Mitkar,
address: KSMSCIT019
3
id: 66fe442c53d61da7c0d5b8b9
name: Aseer Momin,
address: KSMSCIT021
4
id: 66fe444453d61da7c0d5b8ba
name: Kaustabh,
address: KSMSCIT015
```

Part B:



```
Ping successful
Found 4 names

1
id: 66fe43c653d61da7c0d5b8b7
name: Amin Khan,
address: KSMSCIT016

2
id: 66fe441253d61da7c0d5b8b8
name: Hamza Mitkar,
address: KSMSCIT019
```



```
3
id: 66fe442c53d61da7c0d5b8b9
name: Aseer Momin,
address: KSMSCIT021


4
id: 66fe444453d61da7c0d5b8ba
name: Kaustabh,
address: KSMSCIT015
```

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']
```

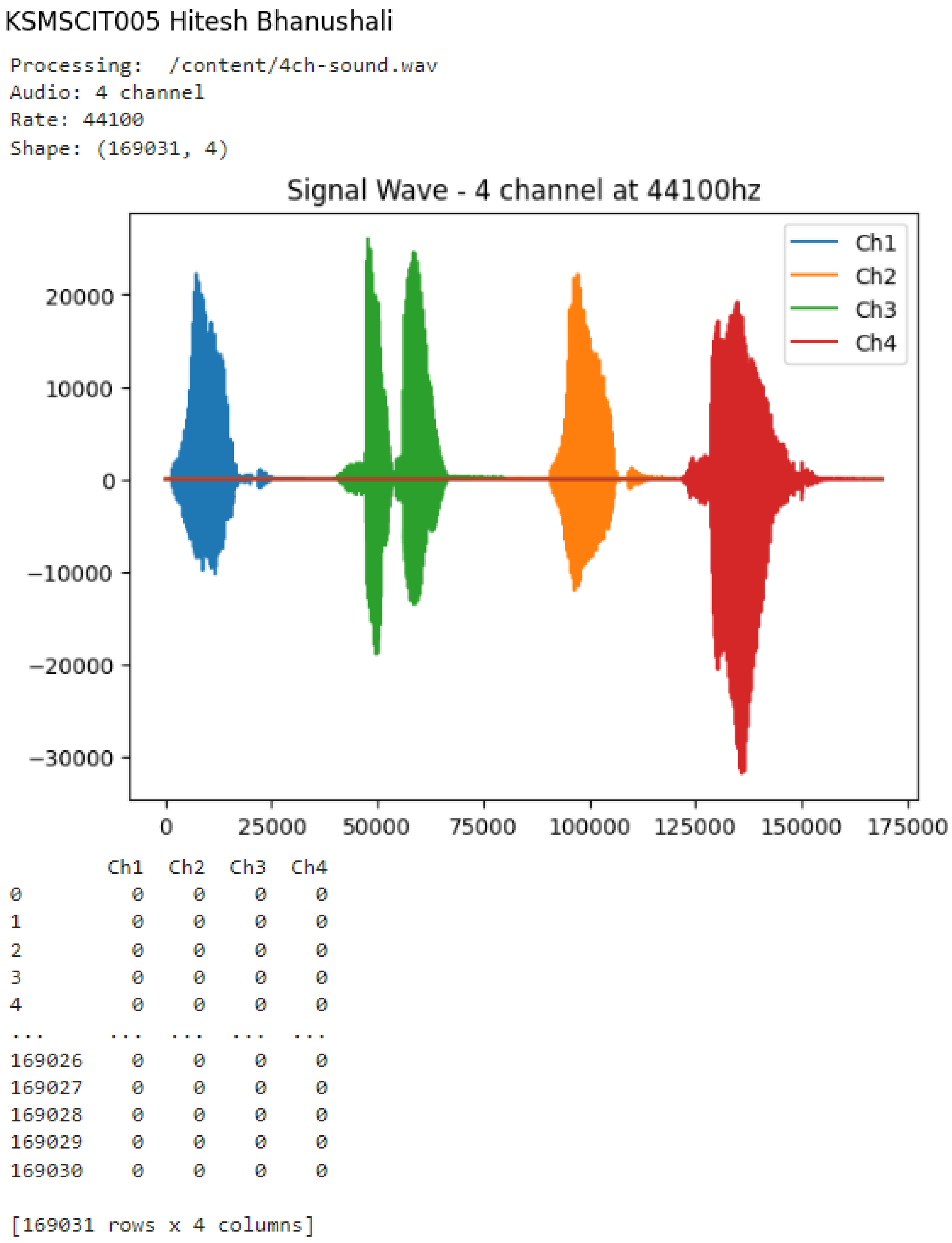
```
# Save DataFrame to CSV
sOutputFileName = '/content/Output/AudioToCSV.csv'
ProcessData.to_csv(sOutputFileName, index=False)

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:

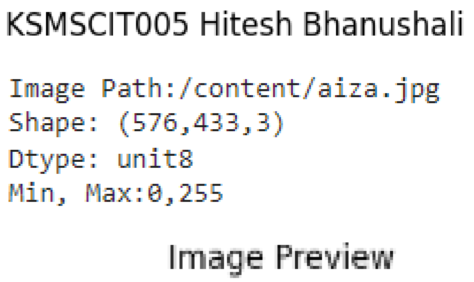
Part A:


KSMSCIT005 Hitesh Bhanushali

```
Processing:  /content/4ch-sound.wav
Audio: 4 channel
Rate: 44100
Shape: (169031, 4)
```

```
        Ch1  Ch2  Ch3  Ch4
0         0    0    0    0
1         0    0    0    0
2         0    0    0    0
3         0    0    0    0
4         0    0    0    0
...     ...  ...  ...  ...
169026    0    0    0    0
169027    0    0    0    0
169028    0    0    0    0
169029    0    0    0    0
169030    0    0    0    0

[169031 rows x 4 columns]
```

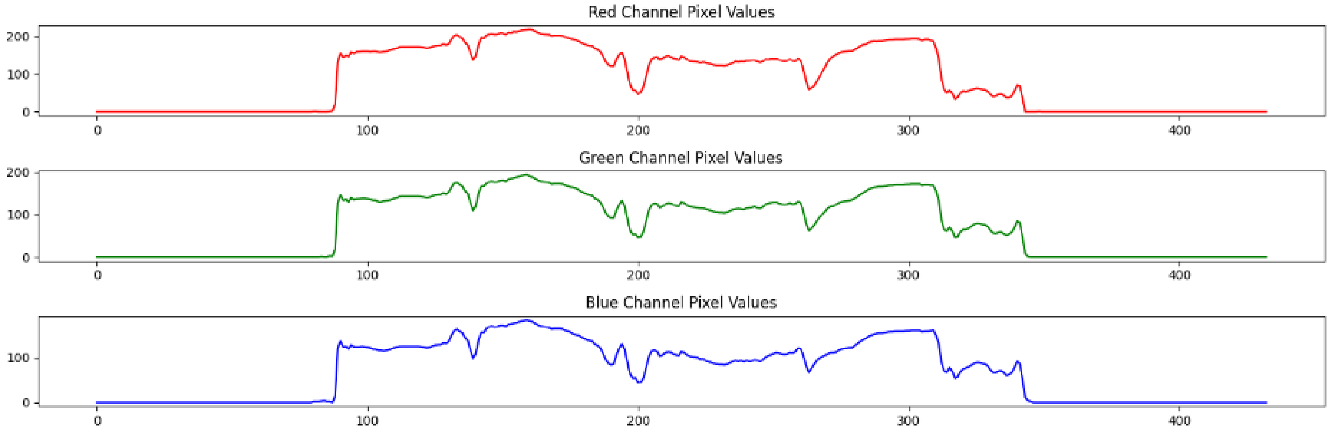
Part B:



KSMSCIT005 Hitesh Bhanushali

```
Image Path:/content/aiza.jpg
Shape: (576,433,3)
Dtype: unit8
Min, Max:0,255
```

Image Preview



Practical No. 12

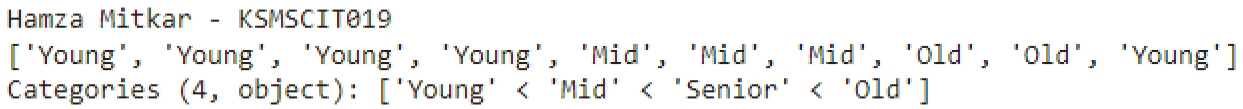
Aim : Data analysis and Visualization

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:



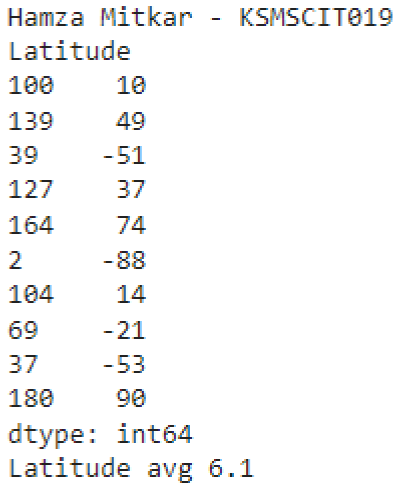
```
Hamza Mitkar - KSMSCIT019
['Young', 'Young', 'Young', 'Young', 'Mid', 'Mid', 'Mid', 'Old', 'Old', 'Young']
Categories (4, object): ['Young' < 'Mid' < 'Senior' < 'Old']
```

Part B

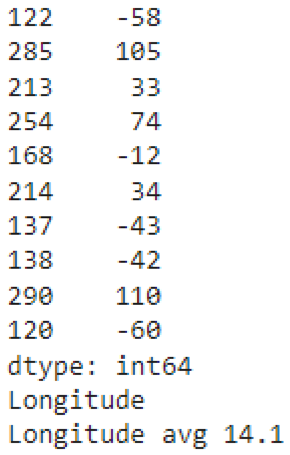
Code:

```
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:



```
Hamza Mitkar - KSMSCIT019
Latitude
100    10
139    49
39    -51
127    37
164    74
2     -88
104    14
69    -21
37    -53
180    90
dtype: int64
Latitude avg 6.1
```



```
122    -58
285    105
213     33
254     74
168    -12
214     34
137    -43
138    -42
290    110
120    -60
dtype: int64
Longitude
Longitude avg 14.1
```

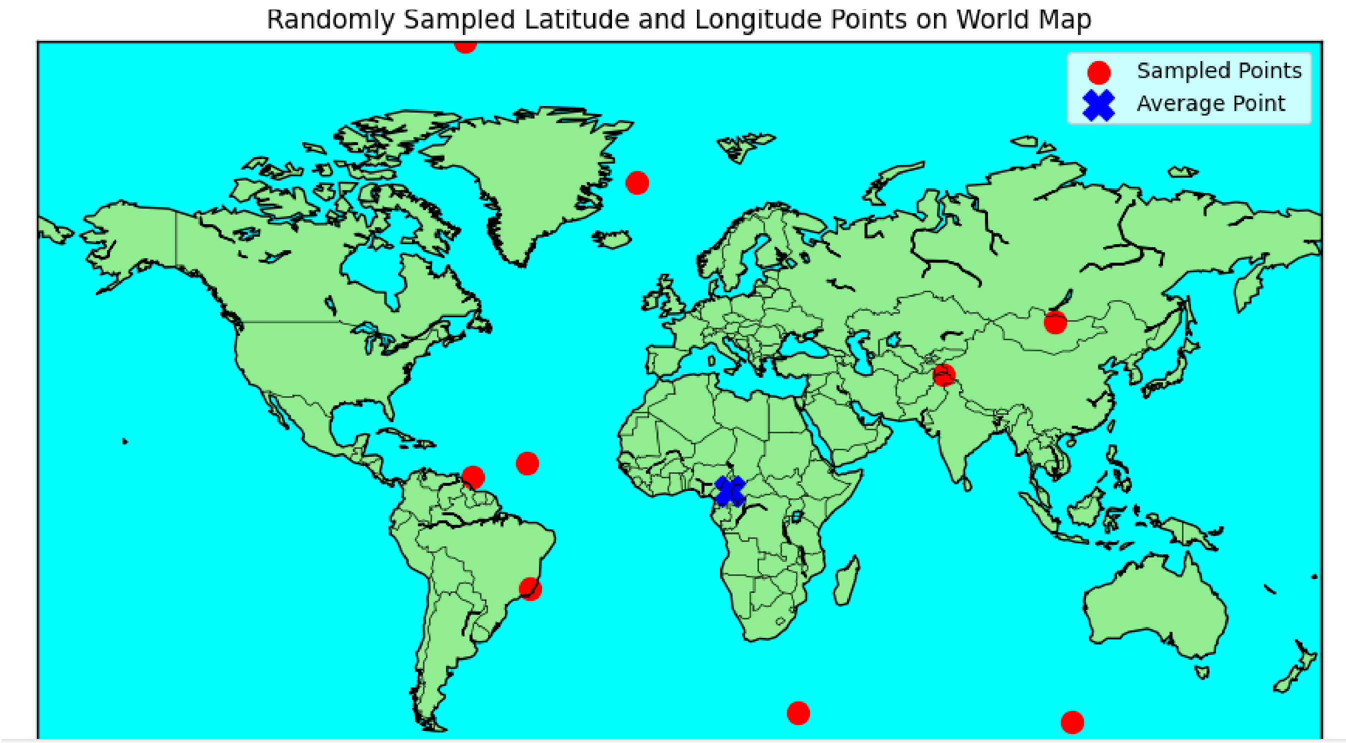
Part C

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:


Randomly Sampled Latitude and Longitude Points on World Map
● Sampled Points
✖ Average Point