

LAB ACTIVITY 15

NAME : Dhevishree GN

REG NO : 22MID0136;

COURSE CODE: CSI3007; LAB : L7+L8;

DATE : 23/09/2025

DIGITAL ASSIGNMENT-4

TASK - 1: The Life Cycle of Matplotlib

```
In [5]: # Importing the necessary libraries
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

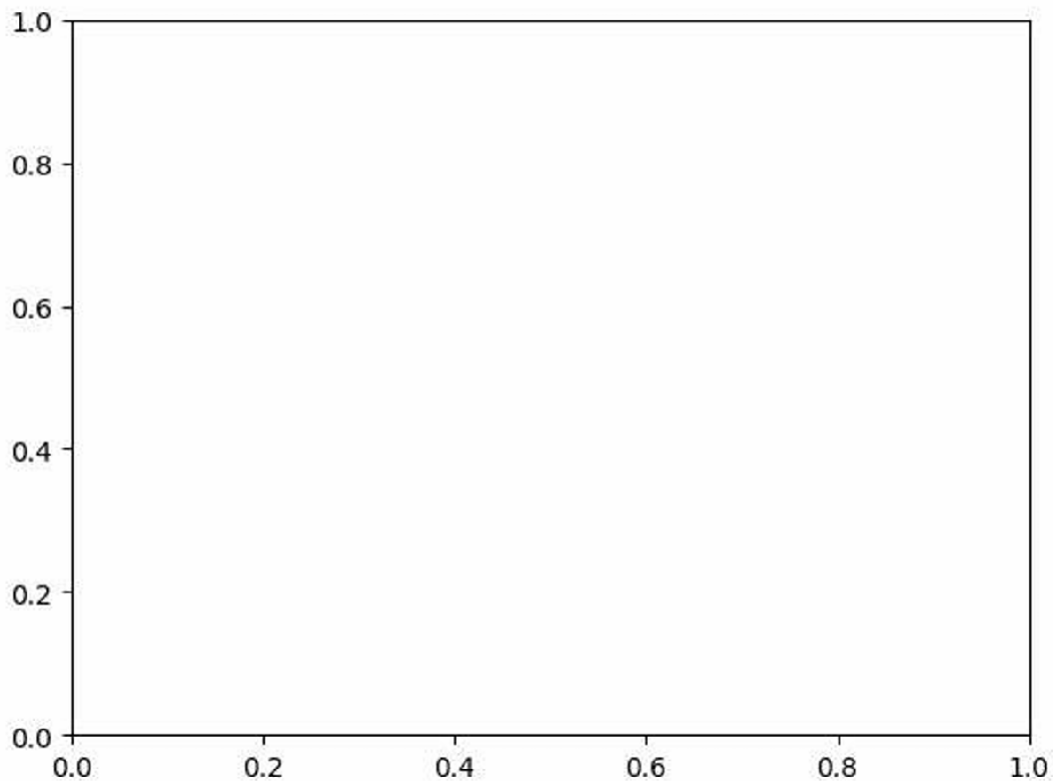
```
In [7]: # Loading the dataset using read_csv() from pandas library
```

```
data = pd.read_csv('Housing.csv')
print(data.head()) # displaying the first 5 rows
```

| | price | area | bedrooms | bathrooms | stories | mainroad | guestroom | basement | \ |
|---|----------|------|----------|-----------|---------|----------|-----------|----------|---|
| 0 | 13300000 | 7420 | 4 | 2 | 3 | yes | no | no | |
| 1 | 12250000 | 8960 | 4 | 4 | 4 | yes | no | no | |
| 2 | 12250000 | 9960 | 3 | 2 | 2 | yes | no | yes | |
| 3 | 12215000 | 7500 | 4 | 2 | 2 | yes | no | yes | |
| 4 | 11410000 | 7420 | 4 | 1 | 2 | yes | yes | yes | |

| | hotwaterheating | airconditioning | parking | prefarea | furnishingstatus |
|---|-----------------|-----------------|---------|----------|------------------|
| 0 | no | yes | 2 | yes | furnished |
| 1 | no | yes | 3 | no | furnished |
| 2 | no | no | 2 | yes | semi-furnished |
| 3 | no | yes | 3 | yes | furnished |
| 4 | no | yes | 2 | no | furnished |

```
In [9]: # Create a figure and axes (Object-Oriented Interface)
fig, ax = plt.subplots()
```



```
: fig, ax = plt.subplots()
```

```
# Plot data on axes
```

```
ax.plot(data["area"], data["price"], color="blue", marker="o", linestyle="-", label="Price vs Area")
```

```
# Customize plot (labels, title, legend, grid)
```

```
ax.set_xlabel("Area (sq ft)")
```

```
ax.set_ylabel("Price (INR)")
```

```
ax.set_title("Housing Price vs Area") ax.legend() ax.grid(True)
```



```
In [15]: # Aggregate data for plotting
```

```
avg_price_by_furnishing = data.groupby('furnishingstatus')['price'].mean()
furnishing_categories = avg_price_by_furnishing.index
average_prices = avg_price_by_furnishing.values
```

```
In [17]: # Create the figure and axes objects
```

```
fig, ax = plt.subplots(figsize=(9, 7))
```

```
# Plot the data on the Axes
```

```
ax.bar(furnishing_categories, average_prices, color=['blue', 'orange', 'green'])
```

```
# Customize the title, labels, and tick marks
```

```
ax.set_title('Average House Price by Furnishing Status', fontsize=18, pad=20)
```

```
ax.set_xlabel('Furnishing Status', fontsize=14)
```

```
ax.set_ylabel('Average Price (INR)', fontsize=14)
```

```
ax.tick_params(axis='x', labelsize=12)
```

```
ax.tick_params(axis='y', labelsize=12)
```

```
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'{x:,.0f}'))
```

```
# Add data labels
```

```
for i, price in enumerate(average_prices):
```

```
    ax.text(i, price + 100000, f'{price:,.0f}', ha='center', va='bottom', fontsize=11, fontweight='bold')
```

```
# Add a grid for better readability
```

```
ax.grid(axis='y', linestyle='--', alpha=0.7)
```

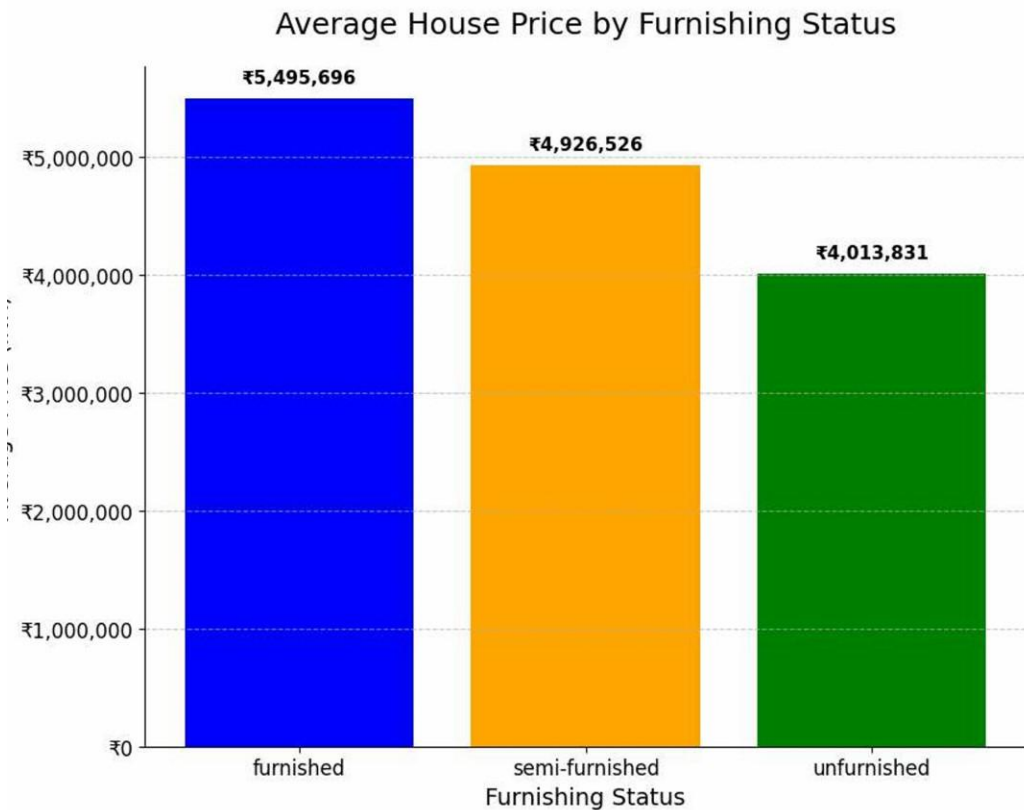
```
ax.spines['top'].set_visible(False)
```

```
ax.spines['right'].set_visible(False)
```

```
# display the figure
```

```
plt.tight_layout()
```

```
plt.show()
```



TASK - 2: Load the Image and change its characteristics (Brightness, hue, saturation)

```
import cv2
```

```
import matplotlib.pyplot as plt
```

```
# Load image (change path as needed)
```

```
image = cv2.imread("sample.jpg") # reads in BGR format
```

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # convert to RGB for matplotlib
```

```
# --- Brightness Adjustment ---
```

```
def adjust_brightness(img, factor=1.2):
```

```
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV).astype("float32")
```

```
    h, s, v = cv2.split(hsv)
```

```

v = v * factor
v = np.clip(v, 0, 255)
hsv = cv2.merge([h, s, v])
return cv2.cvtColor(hsv.astype("uint8"), cv2.COLOR_HSV2RGB)

# --- Saturation Adjustment ---
def adjust_saturation(img, factor=1.5):
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV).astype("float32")
    h, s, v = cv2.split(hsv)
    s = s * factor
    s = np.clip(s, 0, 255)
    hsv = cv2.merge([h, s, v])
    return cv2.cvtColor(hsv.astype("uint8"), cv2.COLOR_HSV2RGB)

# --- Hue Adjustment ---
def adjust_hue(img, shift=30): # shift hue by 30 degrees
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    h, s, v = cv2.split(hsv)
    h = (h.astype(int) + shift) % 180 # Hue range is [0,179]
    hsv = cv2.merge([h.astype("uint8"), s, v])
    return cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)

# Apply transformations
bright_img = adjust_brightness(image, 1.5)
saturated_img = adjust_saturation(image, 1.8)
hue_img = adjust_hue(image, 40)

```

```
# Show Results
```

```
titles = ["Original", "Brightness", "Saturation", "Hue"]
```

```
images = [image, bright_img, saturated_img, hue_img]
```

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

```
for i in range(4):
```

```
    plt.subplot(1,4,i+1)
```

```
    plt.imshow(images[i])
```

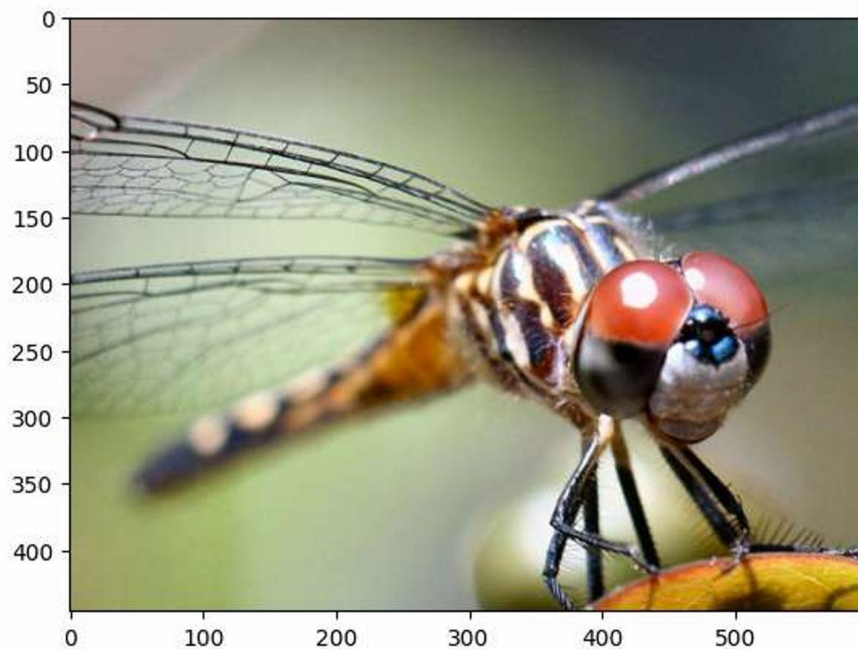
```
    plt.title(titles[i])
```

```
    plt.axis("off")
```

```
plt.show()
```

```
[163, 124, 0]], dtype=uint8)
```

```
In [28]: # Plotting numpy arrays as images  
imgplot = plt.imshow(img)
```



APPLYING PSEUDO CODE:

```
import cv2

import matplotlib.pyplot as plt

# Load image in grayscale
gray_img = cv2.imread("sample.jpg", cv2.IMREAD_GRAYSCALE)

# Apply pseudocolor maps
color1 = cv2.applyColorMap(gray_img, cv2.COLORMAP_JET)    # Jet colormap (rainbow style)
color2 = cv2.applyColorMap(gray_img, cv2.COLORMAP_HOT)     # Hot colormap
color3 = cv2.applyColorMap(gray_img, cv2.COLORMAP_OCEAN)   # Ocean colormap

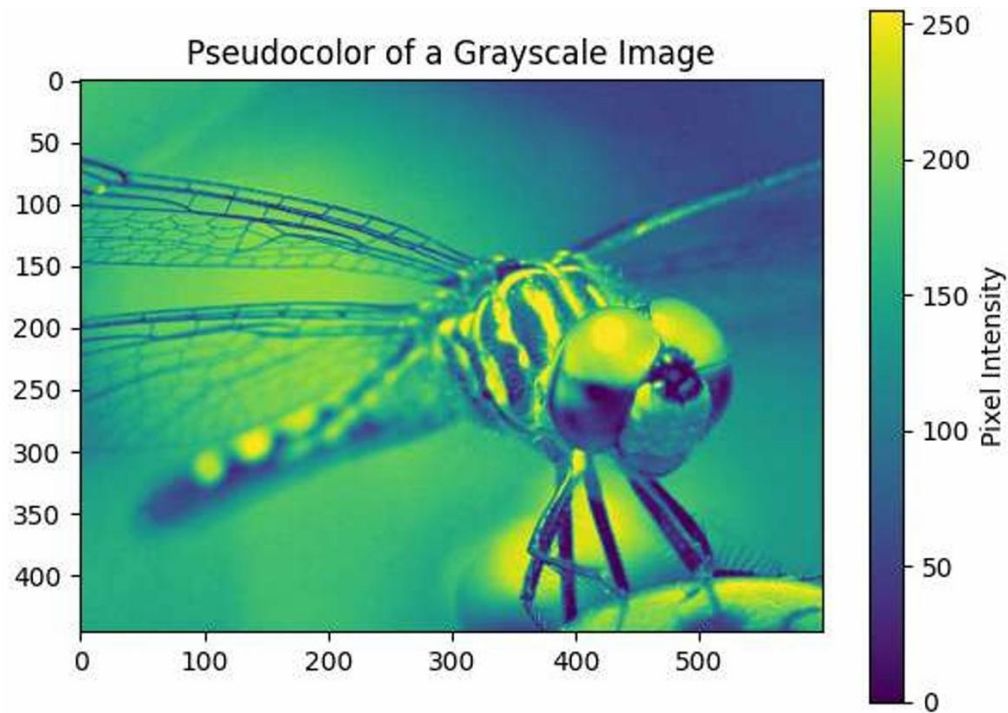
# Convert BGR (OpenCV default) to RGB for matplotlib
color1 = cv2.cvtColor(color1, cv2.COLOR_BGR2RGB)
color2 = cv2.cvtColor(color2, cv2.COLOR_BGR2RGB)
color3 = cv2.cvtColor(color3, cv2.COLOR_BGR2RGB)

# Display results
titles = ["Grayscale", "Pseudocolor: JET", "Pseudocolor: HOT", "Pseudocolor: OCEAN"]
images = [gray_img, color1, color2, color3]

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

for i in range(4):
    plt.subplot(1,4,i+1)
    if i==0:
        plt.imshow(images[i], cmap="gray")
    else:
```

```
plt.imshow(images[i])  
plt.title(titles[i])  
plt.axis("off")  
plt.show()
```



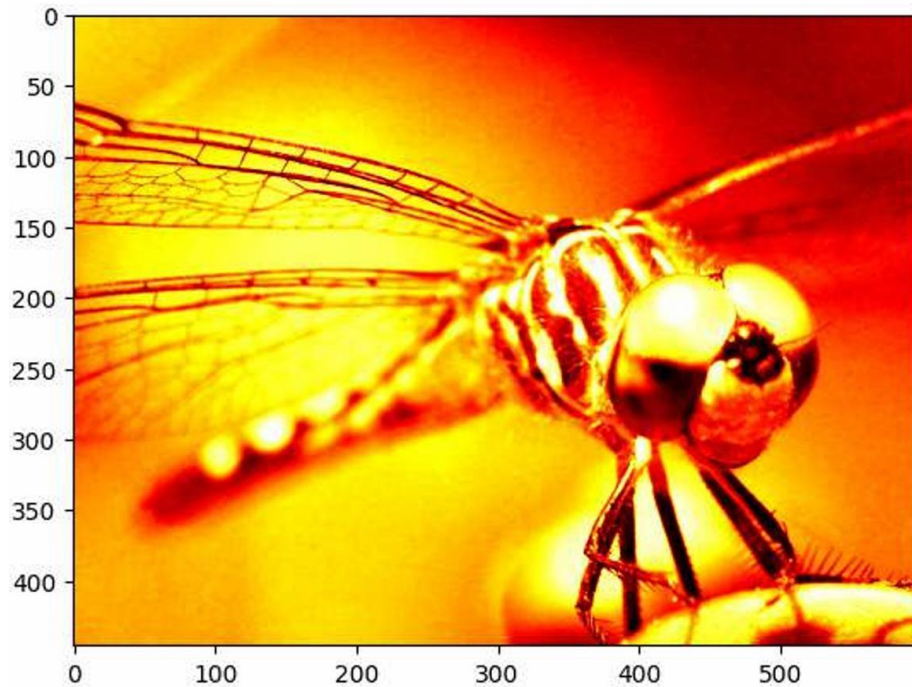
Explanation:

The original image (img) is a 3D NumPy matrix \rightarrow [height, width, 3/4] (RGB).

luminosity (2D, no color) image, the default is called viridis. There are plenty of others (ex: hot)

```
plt.imshow(lum_img, cmap="hot")
```

Out[66]



:

```
import cv2
```

```
import matplotlib.pyplot as plt
```

```
# Load grayscale image
```

```
gray_img = cv2.imread("sample.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# List of all colormaps
```

```
colormaps = [
```

```
    cv2.COLORMAP_AUTUMN, cv2.COLORMAP_BONE, cv2.COLORMAP_JET,
```

```
    cv2.COLORMAP_WINTER, cv2.COLORMAP_RAINBOW, cv2.COLORMAP_OCEAN,
```

```
    cv2.COLORMAP_SUMMER, cv2.COLORMAP_SPRING, cv2.COLORMAP_COOL,
```

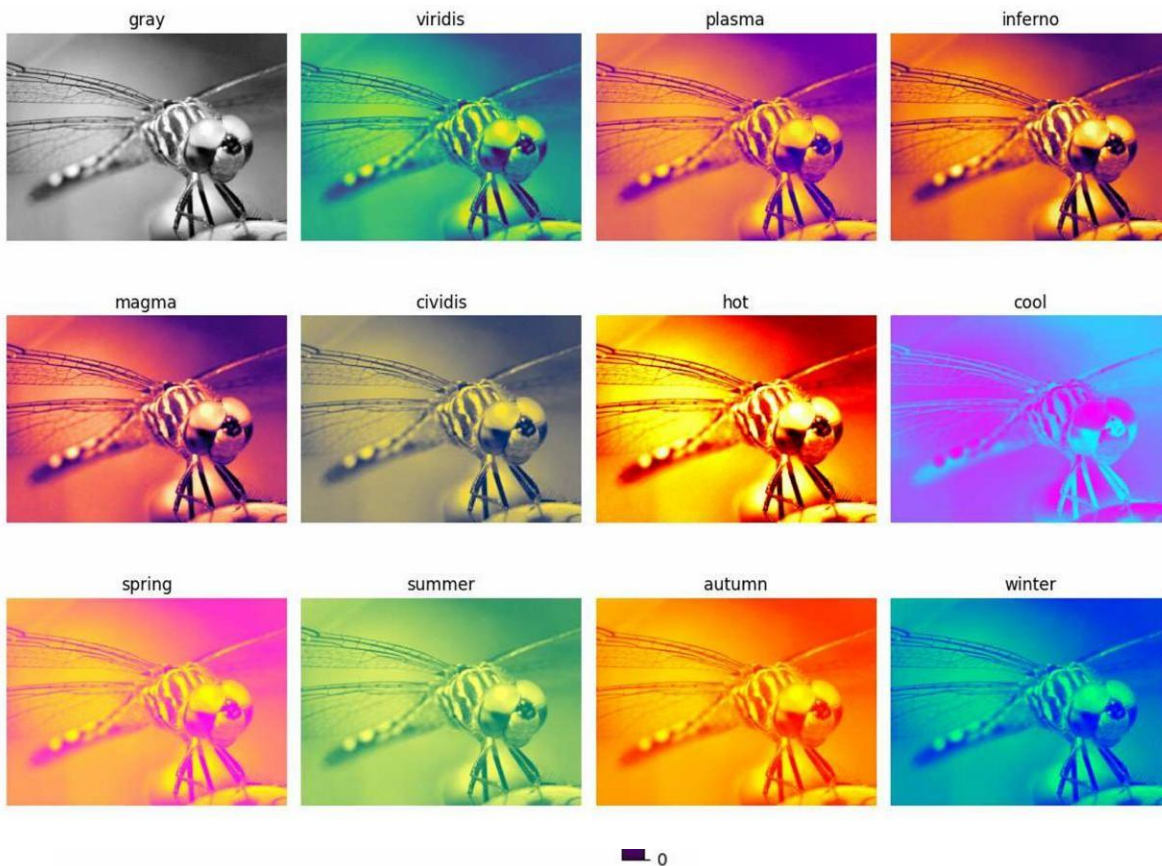
```
    cv2.COLORMAP_HSV, cv2.COLORMAP_PINK, cv2.COLORMAP_HOT,
```

```
    cv2.COLORMAP_PARULA, cv2.COLORMAP_MAGMA, cv2.COLORMAP_INFERNO,
```

```
    cv2.COLORMAP_PLASMA, cv2.COLORMAP_VIRIDIS, cv2.COLORMAP_CIVIDIS,
```

```
cv2.COLORMAP_TWILIGHT, cv2.COLORMAP_TWILIGHT_SHIFTED,  
cv2.COLORMAP_TURBO, cv2.COLORMAP_DEEPPGREEN  
]  
  
names = [  
    "AUTUMN", "BONE", "JET", "WINTER", "RAINBOW", "OCEAN", "SUMMER",  
    "SPRING", "COOL", "HSV", "PINK", "HOT", "PARULA", "MAGMA",  
    "INFERNO", "PLASMA", "VIRIDIS", "CIVIDIS", "TWILIGHT",  
    "TWILIGHT_SHIFTED", "TURBO", "DEEPPGREEN"  
]
```

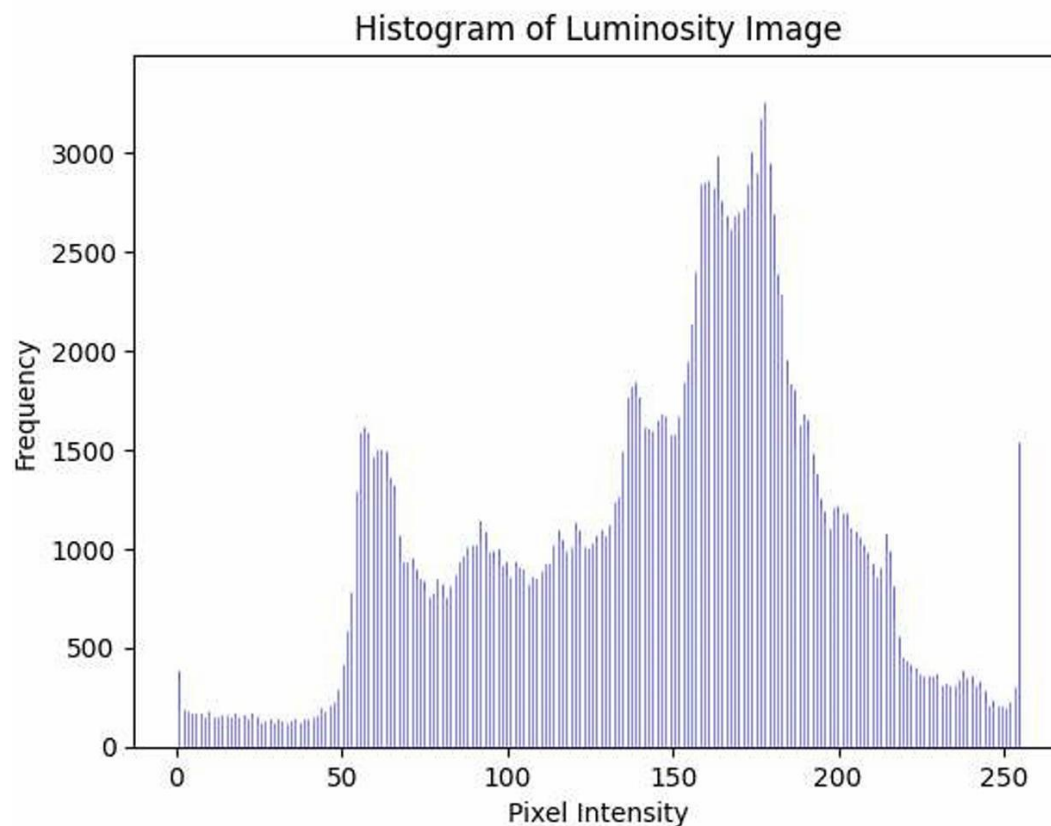
```
plt.figure(figsize=(15,12))  
  
for i, (cmap, name) in enumerate(zip(colormaps, names)):  
    color_img = cv2.applyColorMap(gray_img, cmap)  
    color_img = cv2.cvtColor(color_img, cv2.COLOR_BGR2RGB)  
  
    plt.subplot(5,5,i+1)  
    plt.imshow(color_img)  
    plt.title(name)  
    plt.axis("off")  
  
plt.tight_layout()  
plt.show()
```



In [87]: # Examining a specific data range

```
plt.hist(lum_img.ravel(), bins=range(256), fc='blue', ec='white') # bins Set the bins from 0 to 255 (the full range
                                                                    # fc Set the face color (fc) of the histogram bars to bl
                                                                    # ec Set the edge color (ec) of the histogram bars to wh

plt.title("Histogram of Luminosity Image")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.show()
```



TASK - 3: Finding the inverse of image (In Matrix of pixel values) and finding the negative image (inverted image)

```
In [119... # Load the image
img = Image.open("dragonfly.png")           # load image
img = img.convert("RGB")                   # ensure 3 channels
img_array = np.array(img)                  # convert to NumPy array
```

```
In [128... print("Original Image array: ",img_array)
```

Original Image array: [[[177 148 144]

[179 150 146]

[181 152 148]

...

[66 81 78]

[64 80 77]

[63 79 76]]

[[179 150 146]

[180 151 147]

[182 153 149]

...

[65 80 77]

[63 79 76]

[62 78 75]]

[[180 152 148]

[180 152 148]

[181 153 149]

...

[65 80 77]

[63 79 76]

[62 78 75]]

...

[[150 155 97]

[150 155 97]

[150 155 97]

...

[174 137 7]

[177 140 7]

[180 144 8]]

[[150 155 97]

[150 155 97]

[150 155 97]

...

[170 133 3]

[171 132 1]

[172 134 1]]

```

[ 92 131 255]]]

In [125... # Display original and negative side by side
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].imshow(img_array)
axs[0].set_title("Original Image")
axs[0].axis("off")

axs[1].imshow(negative_img_array)
axs[1].set_title("Negative (Inverted) Image")
axs[1].axis("off")

plt.show()

```

Original Image



Negative (Inverted) Image



TASK - 4: Plotting numerical data, categorical data, text data (Word cloud), ordinal data, audio data and image data

```

In [169... import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Load Housing dataset
data = pd.read_csv("Housing.csv")

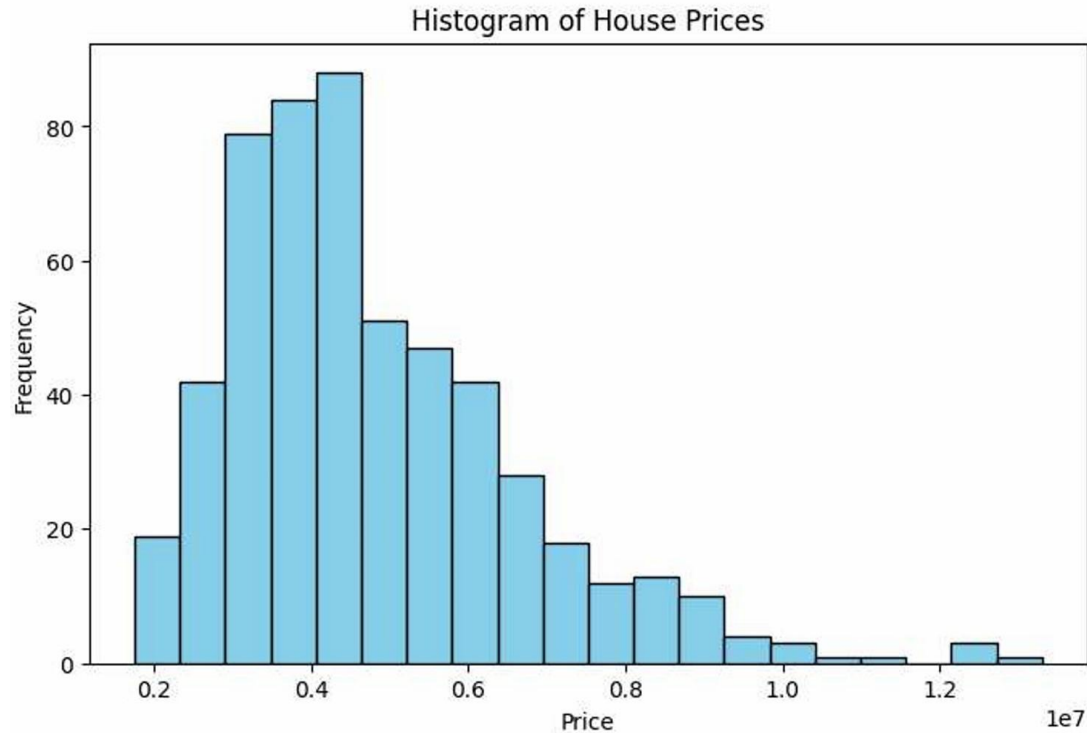
```

1. Numerical Data (price)

```

In [192... plt.figure(figsize=(8,5))
plt.hist(data['price'], bins=20, color='skyblue', edgecolor='black')
plt.title("Histogram of House Prices") # frequency of prices
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.show()

```



LAB ACTIVITY 16

NAME : Dhevishree GN;

REG NO : 22MID0136; COURSE CODE: CSI3007; LAB : L7+L8;

DATE : 30/09/2025

MultIndex creation, indexing, slicing, swapping, alignment, and advanced operations in Pandas.

1. Creating a Series using pandas

```
In [7]: ## TASK-1
import pandas as pd # Importing the pandas library and giving it the alias 'pd'

# Initializing a Series from a list
data = [10, 3.5, 'b', 7, 8] # A mixed list with integer, float, string, and integer values
series_from_list = pd.Series(data) # Creating a Pandas Series from the list
print(series_from_list) # Printing the Series to see its index and values

0    10
1    3.5
2     b
3     7
4     8
dtype: object
```

2. Creating a DataFrame using a dictionary

```
In [9]: ## TASK-2
import pandas as pd # Import pandas library for data manipulation

# Creating a DataFrame using a dictionary
```

```
data = {
    'Name': ['Kamesh', 'Sonali', 'Vedhika', 'Rithika'], # Column 'Name' with string values
    'Age': [26, 21, 23, 27], # Column 'Age' with integer values
    'Score': [88, 91, 80, 95] # Column 'Score' with integer values
}

df = pd.DataFrame(data) # Converting dictionary into a DataFrame
print(df) # Printing the DataFrame to display rows and columns
```

| | Name | Age | Score |
|---|---------|-----|-------|
| 0 | Kamesh | 26 | 88 |
| 1 | Sonali | 21 | 91 |
| 2 | Vedhika | 23 | 80 |
| 3 | Rithika | 27 | 95 |

```
In [10]: ## TASK-3
import pandas as pd # Import pandas library

# Creating first Series with custom indices
s1 = pd.Series([10, 20, 30], index=["x", "y", "z"])

# Creating second Series with overlapping and non-overlapping indices
s2 = pd.Series([40, 50, 60], index=["y", "z", "w"])

# Multiplying the two Series
print(s1 * s2)
```

| | |
|---|--------|
| w | NaN |
| x | NaN |
| y | 800.0 |
| z | 1500.0 |

dtype: float64

```
In [11]: ## TASK-4
import pandas as pd # Import pandas library

# Creating first Series with default index [0, 1, 2]
series_a = pd.Series([10, 20, 30])

# Creating second Series with default index [0, 1, 2]
series_b = pd.Series([40, 50, 60])
```

```
# Adding the two Series element-wise (index-wise)
sum_series = series_a + series_b

# Printing the resulting Series
print(sum_series)
```

```
0    50
1    70
2    90
dtype: int64
```

3. Creating a MultiIndex Series in Different Ways

```
## TASK-5
import pandas as pd # Import pandas library

# Defining two arrays that will act as hierarchical index levels
arrays = [
    ['X', 'X', 'Y', 'Y'], # First level of index
    ['History', 'Geography', 'History', 'Geography'] # Second level of index
]

# Creating a MultiIndex object from the arrays with custom names
index = pd.MultiIndex.from_arrays(arrays, names=('Letter', 'Subject'))

# Creating a Series with the MultiIndex
multi_s = pd.Series([75, 82, 89, 95], index=index)

# Printing the MultiIndex Series
print(multi_s)
```

```
Letter Subject
X    History    75
     Geography  82
Y    History    89
     Geography  95
dtype: int64
```

```
## TASK-6
import pandas as pd # Import pandas library
```

```

# Defining list of tuples, each tuple represents one index pair
tuples = [
    ('C', 'History'),
    ('C', 'Geography'),
    ('D', 'History'),
    ('D', 'Geography')
]

# Creating MultiIndex from tuples with level names
index = pd.MultiIndex.from_tuples(tuples, names=('Letter', 'Subject'))

# Creating a Series with values mapped to the MultiIndex
multi_s = pd.Series([76, 82, 89, 94], index=index)

# Printing the MultiIndex Series
print(multi_s)

```

```

Letter  Subject
C       History    76
        Geography   82
D       History    89
        Geography   94
dtype: int64

```

```

## TASK-7
# Creating MultiIndex from the Cartesian product of two lists
index = pd.MultiIndex.from_product(
    [['C', 'D'], ['History', 'Geography']], # First list x Second list
    names=('Letter', 'Subject')             # Naming index levels
)

# Creating a Series with values mapped to the MultiIndex
multi_s = pd.Series([70, 75, 88, 93], index=index)

# Printing the MultiIndex Series
print(multi_s)

```

| Letter | Subject | |
|--------|-----------|----|
| C | History | 70 |
| | Geography | 75 |
| D | History | 88 |
| | Geography | 93 |

dtype: int64

```
36... ## TASK-8
import numpy as np

# From product (Cartesian product of iterables)
iterables = [{"C", "D"}, {"History", "Geography"}]
index = pd.MultiIndex.from_product(iterables, names=("Letter", "Subject"))
multi_s = pd.Series(np.random.randint(60, 100, size=4), index=index) # Creating a Series with MultiIndex
print("MultiIndex Series from product:\n", multi_s, "\n")
```

MultiIndex Series from product:

| Letter | Subject | |
|--------|-----------|----|
| C | History | 82 |
| | Geography | 99 |
| D | History | 98 |
| | Geography | 98 |

dtype: int32

```
38... ## TASK-9
# Creating a DataFrame that will be used to generate MultiIndex
df = pd.DataFrame({
    'Letter': ['C', 'C', 'D', 'D'], # First column for first-level index
    'Subject': ['History', 'Geography', 'History', 'Geography'] # Second column for second-level index
})

# Creating MultiIndex from the DataFrame
index = pd.MultiIndex.from_frame(df, names=('Letter', 'Subject'))

# Creating a Series with MultiIndex
multi_s = pd.Series([72, 81, 89, 96], index=index)

# Printing the MultiIndex Series
print(multi_s)
```

```
Letter Subject
C      History    72
      Geography   81
D      History    89
      Geography   96
dtype: int64
```

4. Accessing and Indexing

```
.40... ## TASK-10
print("Access all subjects for 'C':\n", multi_s.loc["C"], "\n")
print("Access specific element (D, Geography):\n", multi_s.loc[("D", "Geography")], "\n")

Access all subjects for 'C':
Subject
History    72
Geography   81
dtype: int64

Access specific element (D, Geography):
96
```

5. Slicing in MultiIndex

```
.42... ## TASK-11
print("Slicing from C to D:\n", multi_s.loc["C":"D"], "\n")
print("Partial slice for all History:\n", multi_s.loc[:, "History"], "\n")
```

```
Slicing from C to D:
  Letter  Subject
C        History    72
         Geography   81
D        History    89
         Geography   96
dtype: int64
```

```
Partial slice for all History:
  Letter
C      72
D      89
dtype: int64
```

6. Swapping and Reordering Levels

```
134... ## TASK-12
print("Swapping levels:\n", multi_s.swaplevel(), "\n")
print("Reordering levels:\n", multi_s.reorder_levels(["Subject", "Let
```

```
Swapping levels:
  Subject  Letter
History   C      72
Geography C      81
History   D      89
Geography D      96
dtype: int64
```

```
Reordering levels:
  Subject  Letter
History   C      72
Geography C      81
History   D      89
Geography D      96
dtype: int64
```

7. Passing List of Arrays directly to Series / DataFrame

LAB ACTIVITY 17

NAME :Dhevishree GN;

REG NO : 22MID0136; COURSE CODE: CSI3007; LAB : L7+L8;

DATE : 03/10/2025

**Essential Data Analysis Techniques in Python: DataFrames,
indexing, grouping, merging, and pivot tables in Pandas.**

1.Creating DataFrames (different ways)


```
import pandas as pd
```

```
# Create a DataFrame from dictionary
```

```
data = {  
    "Name": ["Alice", "Bob", "Charlie", "David"],  
    "Age": [24, 27, 22, 32],  
    "Department": ["HR", "IT", "Finance", "IT"],  
    "Salary": [45000, 54000, 50000, 60000]  
}
```

```
df = pd.DataFrame(data)
```

```
print(df)
```

 Original DataFrame

| | Name | Age | Department | Salary |
|---|---------|-----|------------|--------|
| 0 | Alice | 24 | HR | 45000 |
| 1 | Bob | 27 | IT | 54000 |
| 2 | Charlie | 22 | Finance | 50000 |
| 3 | David | 32 | IT | 60000 |

```
Mean salary by department (Series):
Finance    70000.0
IT         72500.0
dtype: float64
```

```
[82]: # Grouping by department (no aggregation yet)

# groupby returns a GroupBy object; actual aggregation not performed yet
grouped_series = salary.groupby(department)
print("Grouped Series (just groups, no aggregation):\n", grouped_series, "\n")
```

```
Grouped Series (just groups, no aggregation):
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000001E2BB3B6270>
```

```
[84]: # Accessing a specific group

# get_group retrieves all entries belonging to a specific group
print("Finance group in Series:\n", grouped_series.get_group("Finance"), "\n")
```

```
Finance group in Series:
Sonali    72000
Mahi      68000
dtype: int64
```

```
[86]: #Aggregation Example

# Aggregation on grouped Series
# Compute mean salary per department
aggregated_series = grouped_series.mean()
print("Aggregated Series (mean salary per department):\n", aggregated_series, "\n")

# Other aggregations
print("Sum per department:\n", grouped_series.sum(), "\n")
print("Max per department:\n", grouped_series.max(), "\n")
```

10. Generating Summary Tables

Summary table

- `df.pivot_table(values='NumericColumn', index='RowCategory', columns='ColumnCategory', aggfunc='mean')` values → numeric column to summarize
- index → row labels
- columns → column labels (optional)
- `aggfunc` → aggregation function (mean, sum, count, etc.)

```
] : ## TASK-18
# Import pandas library
import pandas as pd

# Create a sample DataFrame with Department, Team, and Salary

df = pd.DataFrame({
    "Department": ["Finance", "Finance", "IT", "IT"], # Two departments
```

```

    "Team": ["A", "B", "A", "B"],          # Two teams within each department
    "Salary": [70000, 72000, 65000, 67000] # Salary values for each team
})

# Create a pivot table to summarize salaries
# values="Salary" → we are summarizing the Salary column
# index="Department" → Departments become row index
# columns="Team" → Teams become column headers
# aggfunc="mean" → average salary is calculated per group

summary = df.pivot_table(values="Salary", index="Department", columns="Team", aggfunc="mean")
# Print the summary table
print("Pivot Table Summary:\n", summary)

```

```

Pivot Table Summary:
      Team      A      B
Department
Finance   70000.0  72000.0
IT         65000.0  67000.0

```

```

: ## TASK-19

# Import pandas library
import pandas as pd

# Create a DataFrame with Department, Team, and Salary
df = pd.DataFrame({
    "Department": ["Finance", "Finance", "IT", "IT", "Finance", "IT"], # Departments
    "Team": ["X", "Y", "X", "Y", "X", "Y"], # Teams inside each department
    "Salary": [70000, 72000, 65000, 67000, 71000, 68000] # Salary values
})

# Create a pivot table to summarize the average Salary
# values="Salary" → we want to summarize the Salary column
# index="Department" → rows will be Departments
# columns="Team" → columns will be Teams (X, Y)
# aggfunc="mean" → average salary is calculated if multiple entries exist

summary_table = df.pivot_table(values="Salary", index="Department", columns="Team", aggfunc="mean")
# Print the summary pivot table
print("Summary Table (Average Salary):\n", summary_table)

```

Summary Table (Average Salary):

| Team | X | Y |
|------------|---------|---------|
| Department | | |
| Finance | 70500.0 | 72000.0 |
| IT | 65000.0 | 67500.0 |

```
] : ## TASK-20
# Pivot Table with Multiple Aggregation Functions

# applying more than one aggregation function at once using a list:
summary_table_multi = df.pivot_table(
    values="Salary",
    index="Department",
    columns="Team",
    aggfunc=["mean", "sum", "max"]
)
print("Pivot Table with Multiple Aggregations:\n", summary_table_multi)
```

Pivot Table with Multiple Aggregations:

| Team | mean | | sum | | max | |
|------------|---------|---------|--------|--------|-------|-------|
| | X | Y | X | Y | X | Y |
| Department | | | | | | |
| Finance | 70500.0 | 72000.0 | 141000 | 72000 | 71000 | 72000 |
| IT | 65000.0 | 67500.0 | 65000 | 135000 | 65000 | 68000 |

```
] : ## TASK-21

# Handling Missing Data
# Pivot tables automatically fill missing combinations with NaN.

# Replace missing values with fill value
summary_table_fill = df.pivot_table(
    values="Salary",
    index="Department",
    columns="Team",
    aggfunc="mean",
    fill_value=0
)
print("Pivot Table with Missing Values Filled:\n", summary_table_fill)
```

Pivot Table with Missing Values Filled:

| Team | X | Y |
|------------|---------|---------|
| Department | | |
| Finance | 70500.0 | 72000.0 |
| IT | 65000.0 | 67500.0 |

```
] : ## TASK-22
```

```
# Grouping vs Pivot Tables  
# Pivot tables are essentially groupby + aggregation + reshape in one step.  
# They are easier to read when summarizing across two categorical variables.  
  
# Equivalent using groupby  
grouped = df.groupby(["Department", "Team"])["Salary"].mean().unstack()  
print("Equivalent using groupby + unstack:\n", grouped)
```

Equivalent using groupby + unstack:

| Team | X | Y |
|------------|---------|---------|
| Department | | |
| Finance | 70500.0 | 72000.0 |
| IT | 65000.0 | 67500.0 |

```
] :
```