Day 17 of Training at Ansh Info Tech

Topics Covered

Getting Started with OpenCV

- **Reading and Displaying an Image**: Load an image from disk and display it using OpenCV.
- **Converting an Image to Grayscale**: Convert a color image to grayscale.
- **Save an Image**: Save processed images back to disk.
- **Resize an Image**: Adjust the dimensions of an image.
- **Blur an Image**: Apply blurring techniques to reduce noise.
- **Edge Detection**: Detect edges in images using algorithms like Canny edge detector.
- **Draw a Line**: Draw lines on images for annotations.
- **Draw a Rectangle**: Add rectangles to images for marking regions of interest.
- **Draw a Circle**: Overlay circles on images, useful for highlighting specific points.
- **Write Text on an Image**: Insert text onto images for labeling purposes.
- **Split and Merge Channels**: Manipulate image color channels separately or combine them.
- **Convert an Image to HSV**: Transform color spaces, such as RGB to HSV.
- **Thresholding**: Segment images based on threshold values.
- **Practice Questions on OpenCV and CNN**: Exercises to reinforce understanding of OpenCV concepts and their application in computer vision tasks.
- **Exploratory Data Analysis (EDA)**
    - **Import Libraries**
    - **Reading Dataset**
    - **Data Reduction**
    - **Univariate Analysis**
    - **Bivariate Analysis**
    - **Multivariate Analysis**
    - **Feature Engineering**
    - **Reporting**

Summary

Getting Started with OpenCV

OpenCV (Open Source Computer Vision Library) is widely used for real-time computer vision applications. This session covers fundamental operations to manipulate and process images using OpenCV's rich set of functions and algorithms.

Exploratory Data Analysis in Python Exploratory data analysis (EDA) is a critical initial step in the data science workflow. It involves using Python libraries to inspect, summarize, and visualize data to uncover trends, patterns, and relationships. Here's a breakdown of the key steps in performing EDA with Python:

Importing Libraries: pandas (pd): For data manipulation and analysis. NumPy (np): For numerical computations. Matplotlib.pyplot (plt): For basic plotting functionalities. Seaborn (sns): A built-on top of Matplotlib, providing high-level visualization.

Loading the Data: Use pd.read_csv() for CSV files, similar functions exist for other data formats (e.g., .xlsx, .json).

Initial Inspection: Get an overview of the data using df.head(), .tail(), and .info(). Check data types with df.dtypes.

Data Cleaning: Identify and handle missing values using methods like df.isnull().sum(). Find and address duplicates with df.duplicated().sum().

Univariate Analysis: Analyze single variables at a time. Use descriptive statistics with df.describe() for numerical data. Create histograms, box plots, and density plots to visualize distributions.

Bivariate Analysis: Explore relationships between two variables. Create scatter plots to identify trends and potential correlations.

Visualization: Effective visualizations are crucial for understanding data. Use various plots like bar charts, pie charts, and heatmaps to represent categorical data.

```
Start coding or generate with AI.
```

Getting Started with OpenCV

```
#First, you need to install OpenCV. You can do this using pip:
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages
```

1. Reading and Displaying an Image

```
import cv2
from google.colab.patches import cv2_imshow

# Read the image from file
image = cv2.imread('/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507

# Display the image
cv2_imshow(image)
```

2. Convert an Image to Grayscale

```
import cv2
from google.colab.patches import cv2_imshow

# Read the image from file
image = cv2.imread('/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
cv2_imshow(gray_image)
```

3. Save an Image

```
import cv2
from google.colab.patches import cv2_imshow

# Read the image from file
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we
image = cv2.imread(image_path)

# Check if the image was successfully read
if image is None:
    print(f"Error: Unable to read image from {image_path}")
else:
    # Convert to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Display the grayscale image
    cv2_imshow(gray_image)

    # Save the grayscale image
    cv2.imwrite('/content/gray_image.jpg', gray_image)
```

4. Resize an Image

```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Check if the image was successfully read
if image is None:
    print(f"Error: Unable to read image from {image_path}")
else:
    # Resize the image
    resized_image = cv2.resize(image, (200, 200))

    # Display the resized image
    cv2_imshow(resized_image)

    # Optionally, save the resized image
    cv2.imwrite('/content/resized_image.jpg', resized_image)
```

5. Blur an Image

(15, 15): This is the size of the Gaussian kernel. The kernel size must be a positive odd number (e.g., 3, 5, 7, 9, etc.). In this case, a 15x15 kernel is used, which means the blur effect is calculated based on a 15x15 pixel area around each pixel in the image. A larger kernel size results in a stronger blur effect.

0: This is the standard deviation in the X direction (sigmaX). By setting it to 0, OpenCV will automatically calculate the appropriate value based on the kernel size. You can also specify a value for finer control over the blur effect.

```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Check if the image was successfully read
if image is None:
    print(f"Error: Unable to read image from {image_path}")
else:
    # Apply Gaussian blur
    blurred_image = cv2.GaussianBlur(image, (15, 15), 0)

    # Display the blurred image
    cv2_imshow(blurred_image)
```

6. Edge Detection

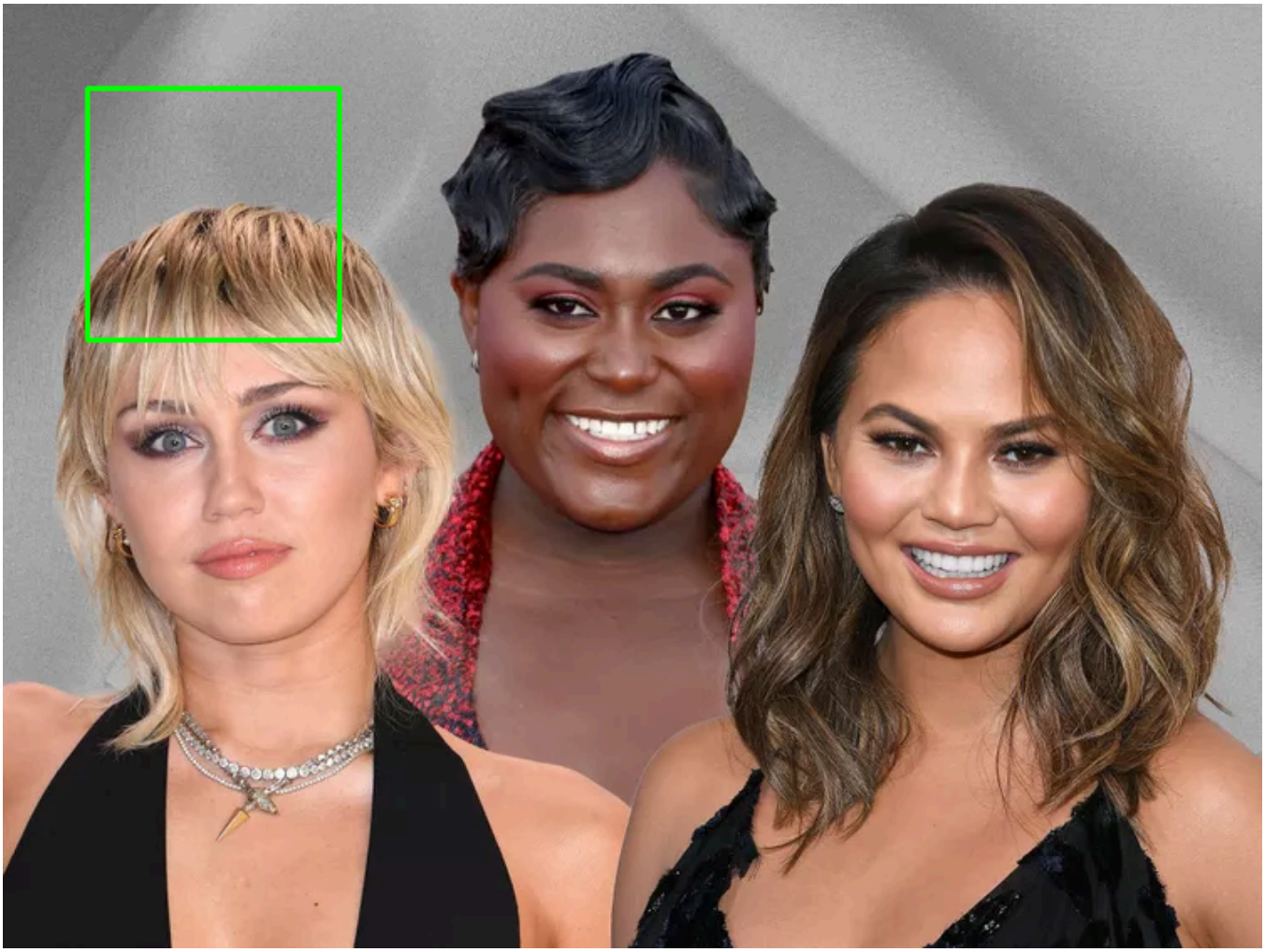
```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply Canny edge detection
edges = cv2.Canny(gray_image, 100, 200)

# Display the edges
cv2_imshow(edges)
```

7. Draw a Line

```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Draw a line on the image
start_point = (50, 50)
end_point = (200, 200)
color = (255, 0, 0)  # BGR format
thickness = 2
cv2.line(image, start_point, end_point, color, thickness)

# Display the image with the line
cv2_imshow(image)
```

8. Draw a Rectangle

```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Draw a rectangle on the image
start_point = (50, 50)
end_point = (200, 200)
color = (0, 255, 0)  # BGR format
thickness = 2
cv2.rectangle(image, start_point, end_point, color, thickness)

# Display the image with the rectangle
cv2_imshow(image)
```



9. Draw a Circle

```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Draw a circle on the image
center_coordinates = (150, 150)
radius = 50
color = (0, 0, 255)  # BGR format
thickness = 2
cv2.circle(image, center_coordinates, radius, color, thickness)

# Display the image with the circle
cv2_imshow(image)
```

12. Write Text on an Image

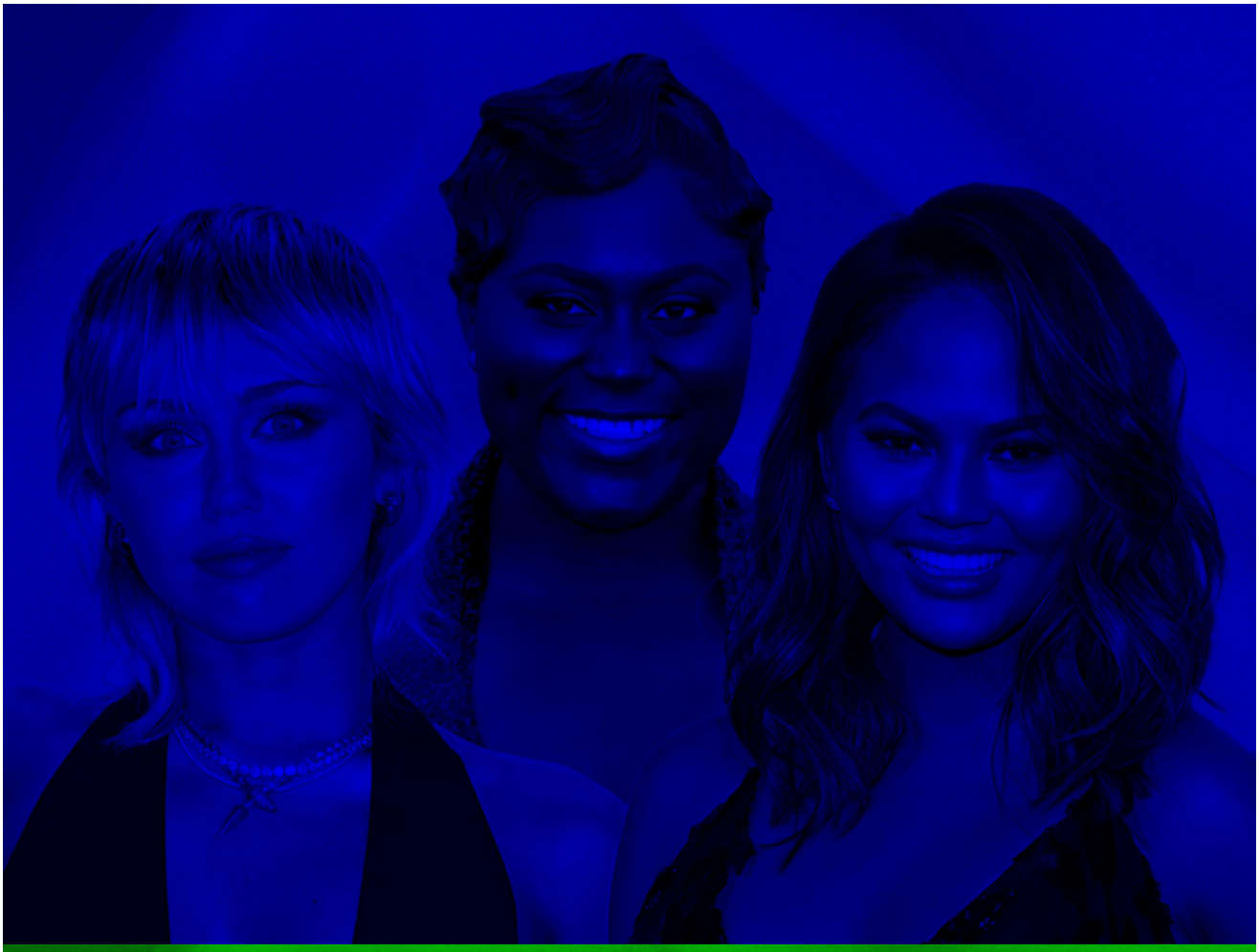
```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Write text on the image
text = 'Hello, OpenCV!'
org = (50, 50)  # Bottom-left corner of the text
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1
color = (255, 255, 255)  # White color in BGR format
thickness = 2
cv2.putText(image, text, org, font, font_scale, color, thickness)

# Display the image with the text
cv2_imshow(image)
```

13. Split and Merge Channels

The line blank = np.zeros_like(b) creates a blank (black) image that has the same dimensions as the blue channel (b). Here's a breakdown of what it does:

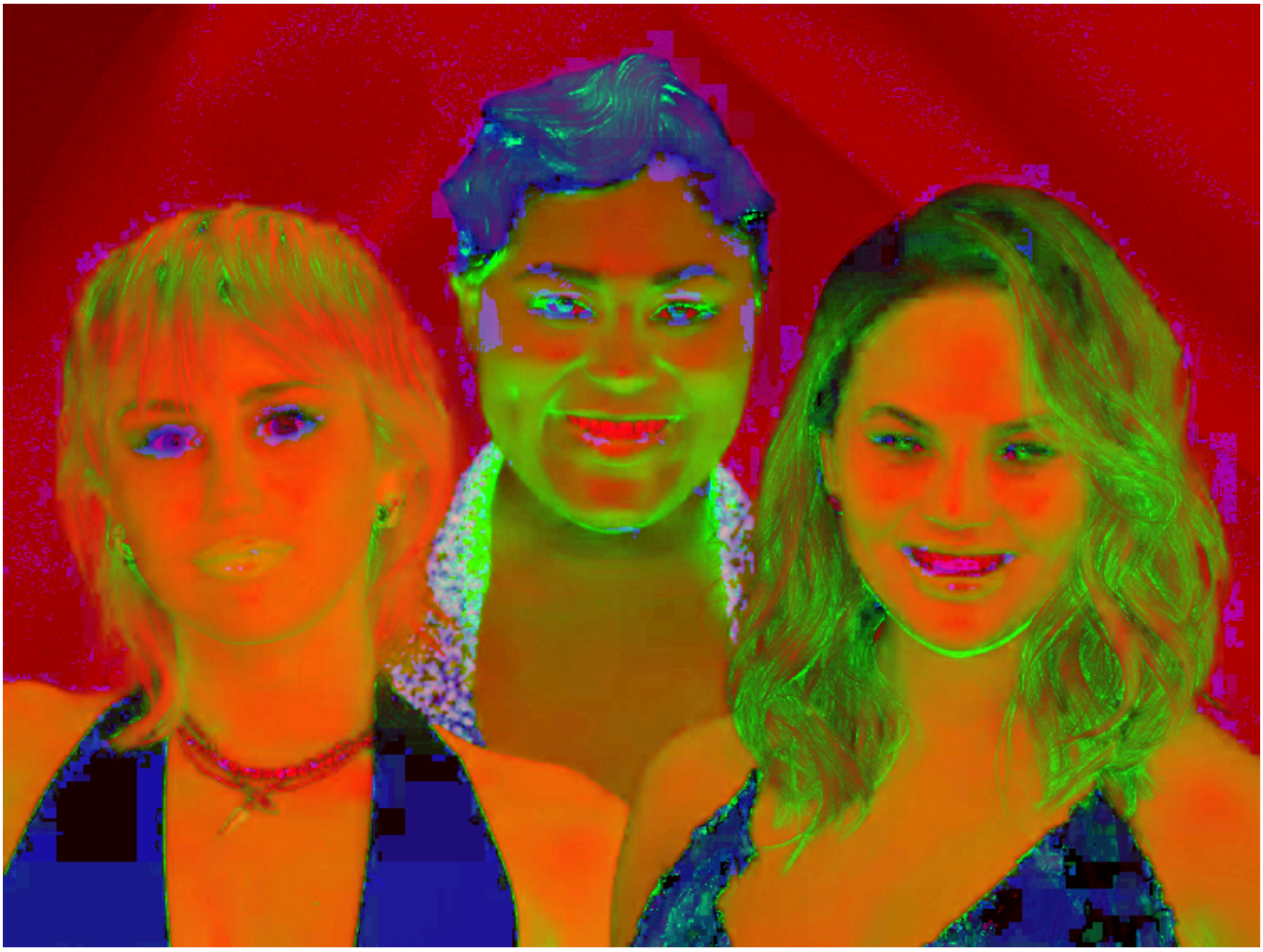
np.zeros_like(b): This function call creates a new array of zeros with the same shape and type as the given array b. b is a 2D array representing one color channel (blue) of the image, so blank will be a 2D array of the same size filled with zeros.

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Split the image into channels
b, g, r = cv2.split(image)

# Create blank channels
blank = np.zeros_like(b)

# Merge the channels with blank channels to visualize them in color
blue_channel = cv2.merge([b, blank, blank])
green_channel = cv2.merge([blank, g, blank])
red_channel = cv2.merge([blank, blank, r])

# Display each channel in its respective color
cv2_imshow(blue_channel)  # Blue channel
cv2_imshow(green_channel)  # Green channel
cv2_imshow(red_channel)  # Red channel
```



14. Convert an Image to HSV

```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Convert BGR image to HSV
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Display the HSV image
cv2_imshow(hsv_image)
```



15. Thresholding

```
import cv2
from google.colab.patches import cv2_imshow

# Define the image path
image_path = '/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.we

# Read the image
image = cv2.imread(image_path)

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding
ret, thresh = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)

# Display the thresholded image
cv2_imshow(thresh)
```

If you have a grayscale image (gray_image) and you apply the thresholding operation with cv2.THRESH_BINARY mode and a threshold value of 127:

Pixels in gray_image that have intensity values greater than 127 will become white (255 in 8-bit images). Pixels with intensity values less than or equal to 127 will become black (0 in 8-bit images).

```
Start coding or generate with AI.
```

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) is a method of analyzing datasets to understand their main characteristics. It involves summarizing data features, detecting patterns, and uncovering relationships through visual and statistical techniques. EDA helps in gaining insights and formulating hypotheses for further analysis.

Step 1: Import Python Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#to ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Step 2: Reading Dataset

The Pandas library offers a wide range of possibilities for loading data into the pandas DataFrame from files like JSON, .csv, .xlsx, .sql, .pickle, .html, .txt, images etc.

Most of the data are available in a tabular format of CSV files. It is trendy and easy to access. Using the read_csv() function, data can be converted to a pandas DataFrame.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#to ignore warnings
import warnings
warnings.filterwarnings('ignore')

data = pd.read_csv("/content/used_cars_data(1).csv")
```

Analyzing the Data

Before we make any inferences, we listen to our data by examining all variables in the data.

The main goal of data understanding is to gain general insights about the data, which covers the number of rows and columns, values in the data, datatypes, and Missing values in the dataset.

shape – shape will display the number of observations(rows) and features(columns) in the dataset

There are 7253 observations and 14 variables in our dataset

head() will display the top 5 observations of the dataset

```
data.head()
```

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	

tail() will display the last 5 observations of the dataset

```
data.tail()
```

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Ow
7248	7248	Volkswagen Vento Diesel Trendline	Hyderabad	2011	89411	Diesel	Manual	
7249	7249	Volkswagen Polo GT TSI	Mumbai	2015	59000	Petrol	Automatic	
7250	7250	Nissan Micra Diesel XV	Kolkata	2012	28000	Diesel	Manual	
		Volkswagen						

info() helps to understand the data type and information about data, including the number of records in each column, data having null or not null, Data type, the memory usage of the dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   S.No.              7253 non-null   int64  
 1   Name               7253 non-null   object 
 2   Location           7253 non-null   object 
 3   Year               7253 non-null   int64  
 4   Kilometers_Driven  7253 non-null   int64  
 5   Fuel_Type          7253 non-null   object 
 6   Transmission       7253 non-null   object 
 7   Owner_Type         7253 non-null   object 
 8   Mileage            7251 non-null   object 
 9   Engine             7207 non-null   object 
 10  Power              7207 non-null   object 
 11  Seats              7200 non-null   float64
 12  New_Price          1006 non-null   object 
 13  Price              6019 non-null   float64
```

```
dtypes: float64(2), int64(3), object(9)
memory usage: 793.4+ KB
```

Check for Duplication

nunique() based on several unique values in each column and the data description, we can identify the continuous and categorical columns in the data. Duplicated data can be handled or removed based on further analysis

```
data.nunique()
```

```
S.No.               7253
Name                2041
Location              11
Year                  23
Kilometers_Driven   3660
Fuel_Type              5
Transmission           2
Owner_Type             4
Mileage              450
Engine               150
Power                386
Seats                  9
New_Price            625
Price               1373
dtype: int64
```

Missing Values Calculation

isnull() is widely been in all pre-processing steps to identify null values in the data

In our example, data.isnull().sum() is used to get the number of missing records in each column

```
data.isnull().sum()
```

```
S.No.                 0
Name                  0
Location              0
Year                  0
Kilometers_Driven     0
Fuel_Type             0
Transmission          0
Owner_Type            0
Mileage               2
```

```
Engine                46
Power                 46
Seats                 53
New_Price           6247
Price               1234
dtype: int64
```

```
#it calculate the percentage of missing values in each column
(data.isnull().sum()/(len(data)))*100
```

```
S.No.                0.000000
Name                 0.000000
Location             0.000000
Year                 0.000000
Kilometers_Driven    0.000000
Fuel_Type            0.000000
Transmission         0.000000
Owner_Type           0.000000
Mileage              0.027575
Engine               0.634220
Power                0.634220
Seats                0.730732
New_Price           86.129877
Price               17.013650
dtype: float64
```

The percentage of missing values for the columns New_Price and Price is ~86% and ~17%, respectively.

Step 3: Data Reduction

Some columns or variables can be dropped if they do not add value to our analysis.

In our dataset, the column S.No have only ID values, assuming they don't have any predictive power to predict the dependent variable.

axis=0: This indicates that the operation should be performed on rows. It is the default value.

axis=1: This indicates that the operation should be performed on columns.

```
# Remove S.No. column from data
data = data.drop(['S.No.'], axis = 1)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```

```
 0   Name               7253 non-null   object
 1   Location           7253 non-null   object
 2   Year               7253 non-null   int64
 3   Kilometers_Driven  7253 non-null   int64
 4   Fuel_Type          7253 non-null   object
 5   Transmission       7253 non-null   object
 6   Owner_Type         7253 non-null   object
 7   Mileage            7251 non-null   object
 8   Engine             7207 non-null   object
 9   Power              7207 non-null   object
 10  Seats              7200 non-null   float64
 11  New_Price          1006 non-null   object
 12  Price              6019 non-null   float64
dtypes: float64(2), int64(2), object(9)
memory usage: 736.8+ KB
```

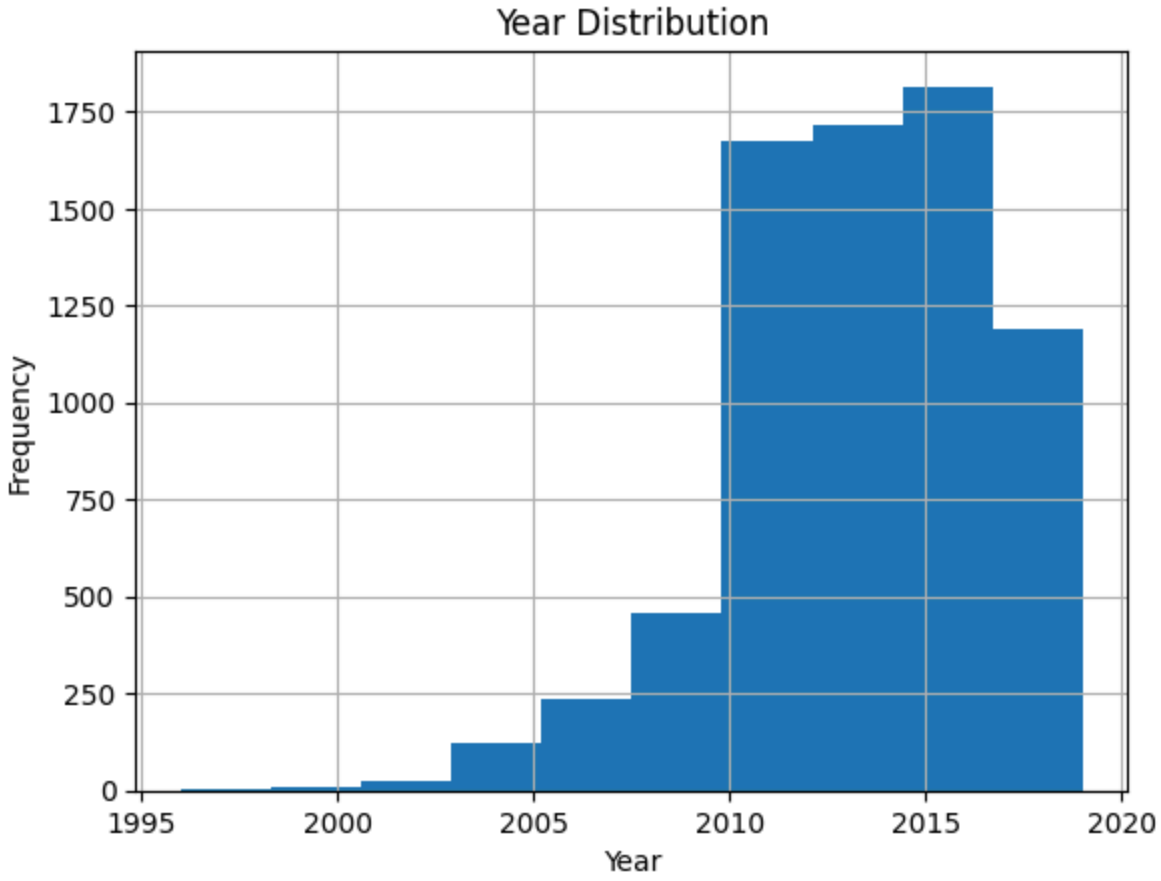
Step 4: Univariate Analysis

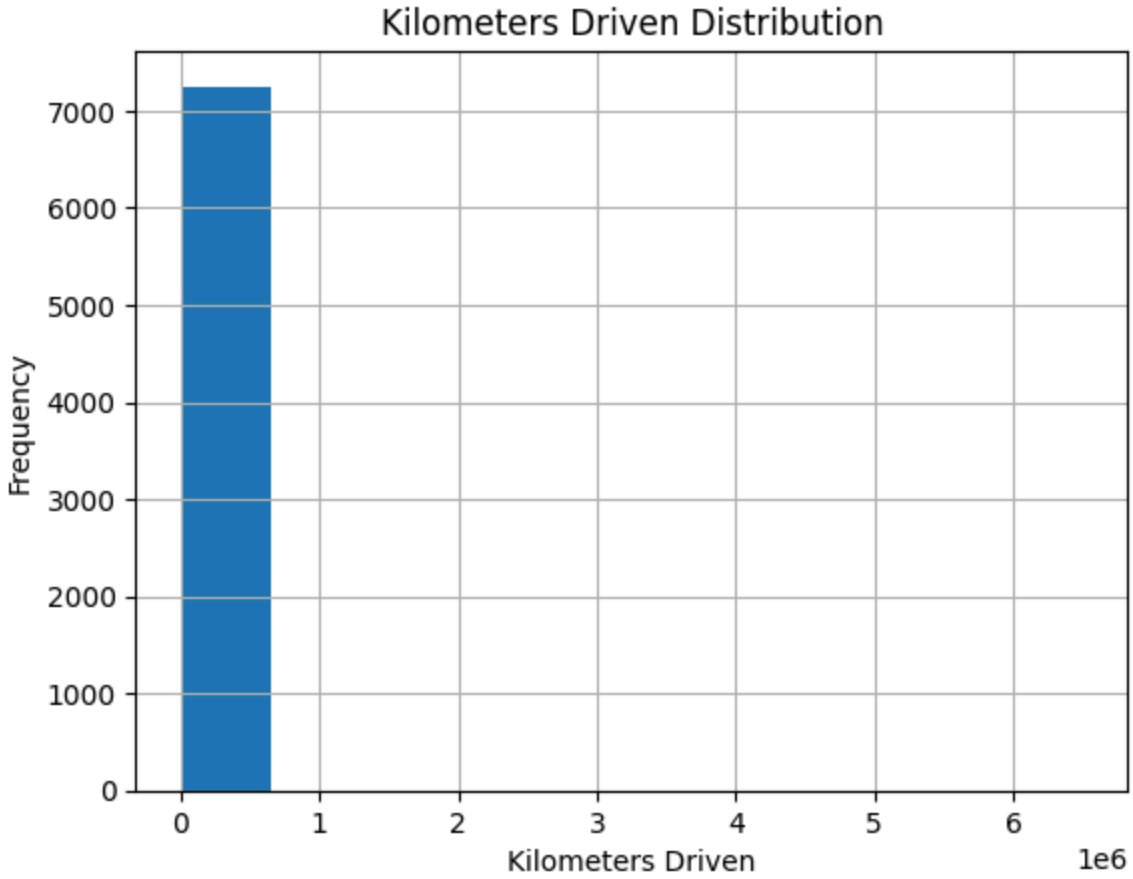
Univariate analysis focuses on a single variable at a time. Its primary goal is to describe the distribution of data and understand its basic properties. This involves examining measures of central tendency (like mean, median, mode) to determine the typical or average value of the variable. Measures of dispersion (such as range, variance, standard deviation) help assess the spread or variability of data points around the central value. Histograms, box plots, and frequency distributions are graphical tools used in univariate analysis to visualize data distributions.

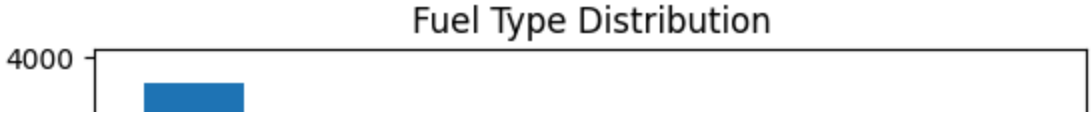
For example, in a study of exam scores, univariate analysis would involve summarizing the scores using measures like the average score, range of scores, and graphical representations to understand how scores are distributed among students.

Examine each variable individually.

```
import matplotlib.pyplot as plt

# Distribution of 'Year'
data['Year'].hist()
plt.xlabel('Year')
plt.ylabel('Frequency')
plt.title('Year Distribution')
plt.show()

# Distribution of 'Kilometers_Driven'
data['Kilometers_Driven'].hist()
plt.xlabel('Kilometers Driven')
plt.ylabel('Frequency')
plt.title('Kilometers Driven Distribution')
plt.show()

# Distribution of 'Fuel_Type'
data['Fuel_Type'].value_counts().plot(kind='bar')
plt.xlabel('Fuel Type')
plt.ylabel('Count')
plt.title('Fuel Type Distribution')
plt.show()

# Distribution of 'Transmission'
data['Transmission'].value_counts().plot(kind='bar')
plt.xlabel('Transmission')
plt.ylabel('Count')
plt.title('Transmission Distribution')
plt.show()

# Distribution of 'Price'
data['Price'].hist()
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Price Distribution')
plt.show()
```


Year Distribution
Frequency
Year
0
250
500
750
1000
1250
1500
1750
1995
2000
2005
2010
2015
2020


Kilometers Driven Distribution
Frequency
Kilometers Driven
1e6
0
1000
2000
3000
4000
5000
6000
7000
0
1
2
3
4
5
6


Fuel Type Distribution
4000

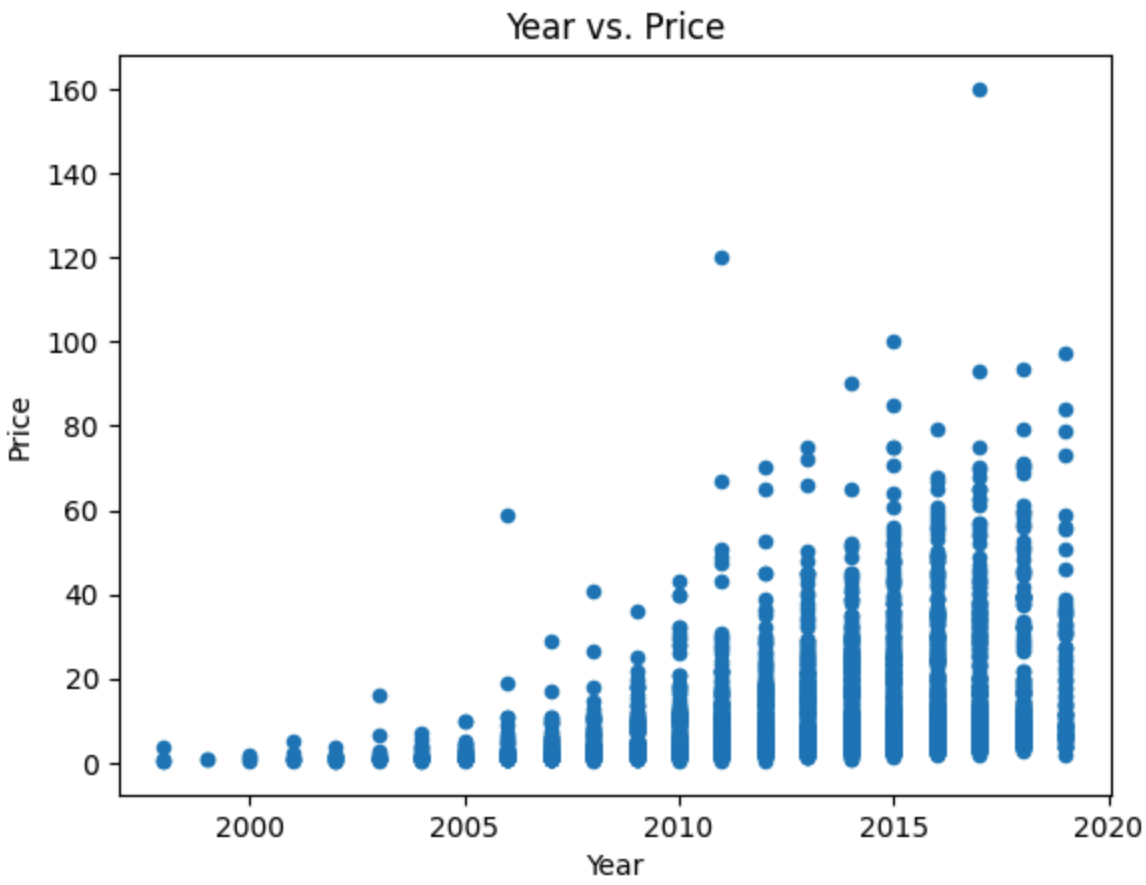
Step 5: Bivariate Analysis

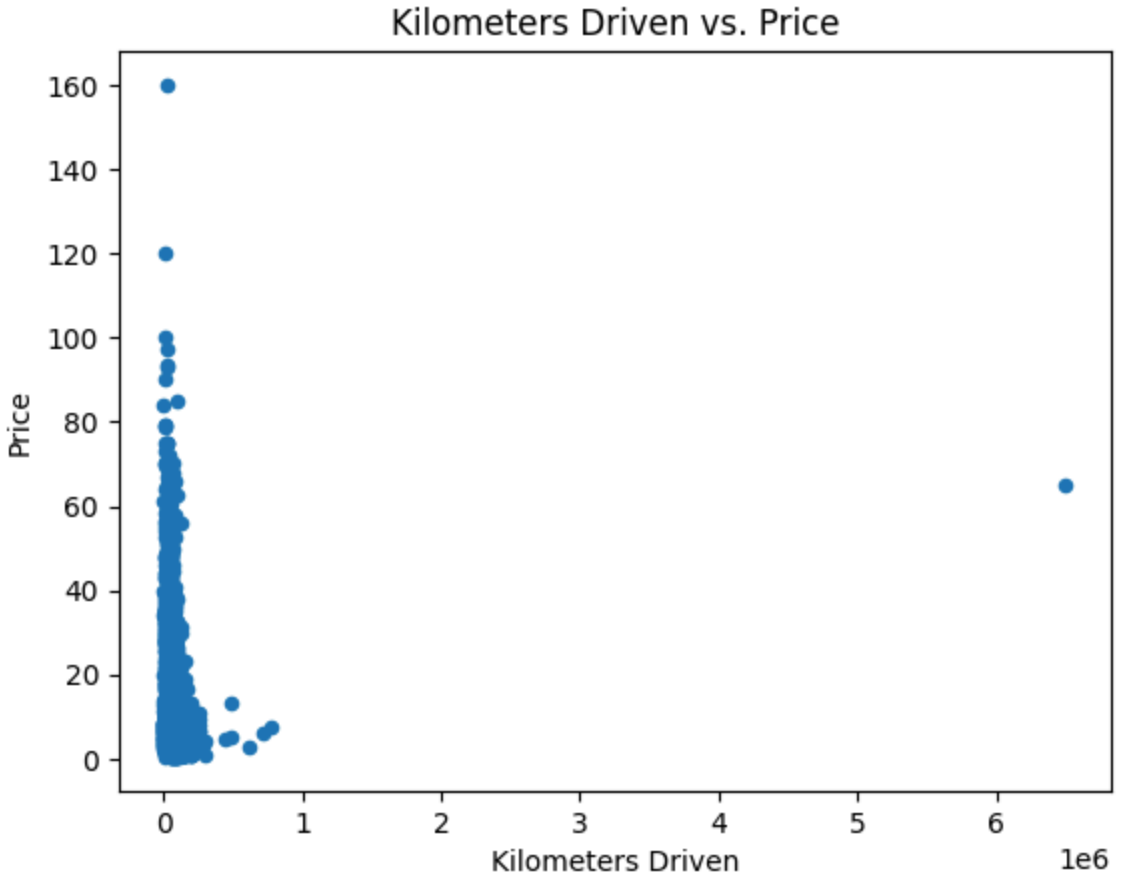
Bivariate analysis examines the relationship between two variables simultaneously. It seeks to determine whether there is a statistical association or correlation between the two variables. Scatter plots are commonly used in bivariate analysis to visualize how one variable changes with respect to another. Correlation coefficients (such as Pearson correlation coefficient) quantify the strength and direction of the relationship between variables.

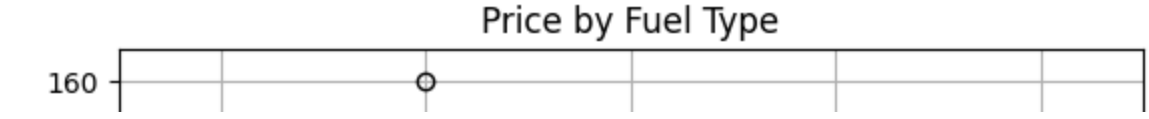
For instance, in a study correlating rainfall amounts with crop yields, bivariate analysis would help establish whether higher rainfall amounts are associated with increased crop yields.

Examine the relationships between pairs of variables.

```
# Scatter plot of 'Year' vs 'Price'
data.plot.scatter(x='Year', y='Price')
plt.xlabel('Year')
plt.ylabel('Price')
plt.title('Year vs. Price')
plt.show()

# Scatter plot of 'Kilometers_Driven' vs 'Price'
data.plot.scatter(x='Kilometers_Driven', y='Price')
plt.xlabel('Kilometers Driven')
plt.ylabel('Price')
plt.title('Kilometers Driven vs. Price')
plt.show()

# Box plot of 'Price' by 'Fuel_Type'
data.boxplot(column='Price', by='Fuel_Type')
plt.xlabel('Fuel Type')
plt.ylabel('Price')
plt.title('Price by Fuel Type')
plt.suptitle('')
plt.show()
```


Year vs. Price
Price
Year
0
20
40
60
80
100
120
140
160
2000
2005
2010
2015
2020


Kilometers Driven vs. Price
Price
Kilometers Driven
1e6
0
20
40
60
80
100
120
140
160
0
1
2
3
4
5
6


Price by Fuel Type
160

Step 6: Multivariate Analysis

Multivariate analysis involves the simultaneous analysis of three or more variables. Its goal is to understand complex relationships among multiple variables and identify patterns that may not be evident in univariate or bivariate analyses alone. Multivariate techniques include regression analysis, principal component analysis (PCA), factor analysis, cluster analysis, and discriminant analysis.

Regression analysis, for example, examines how one or more independent variables predict a dependent variable. PCA reduces the dimensionality of data by transforming variables into a smaller set of orthogonal components, which helps in identifying patterns and relationships among variables. Cluster analysis groups similar objects or individuals into clusters based on their attributes.