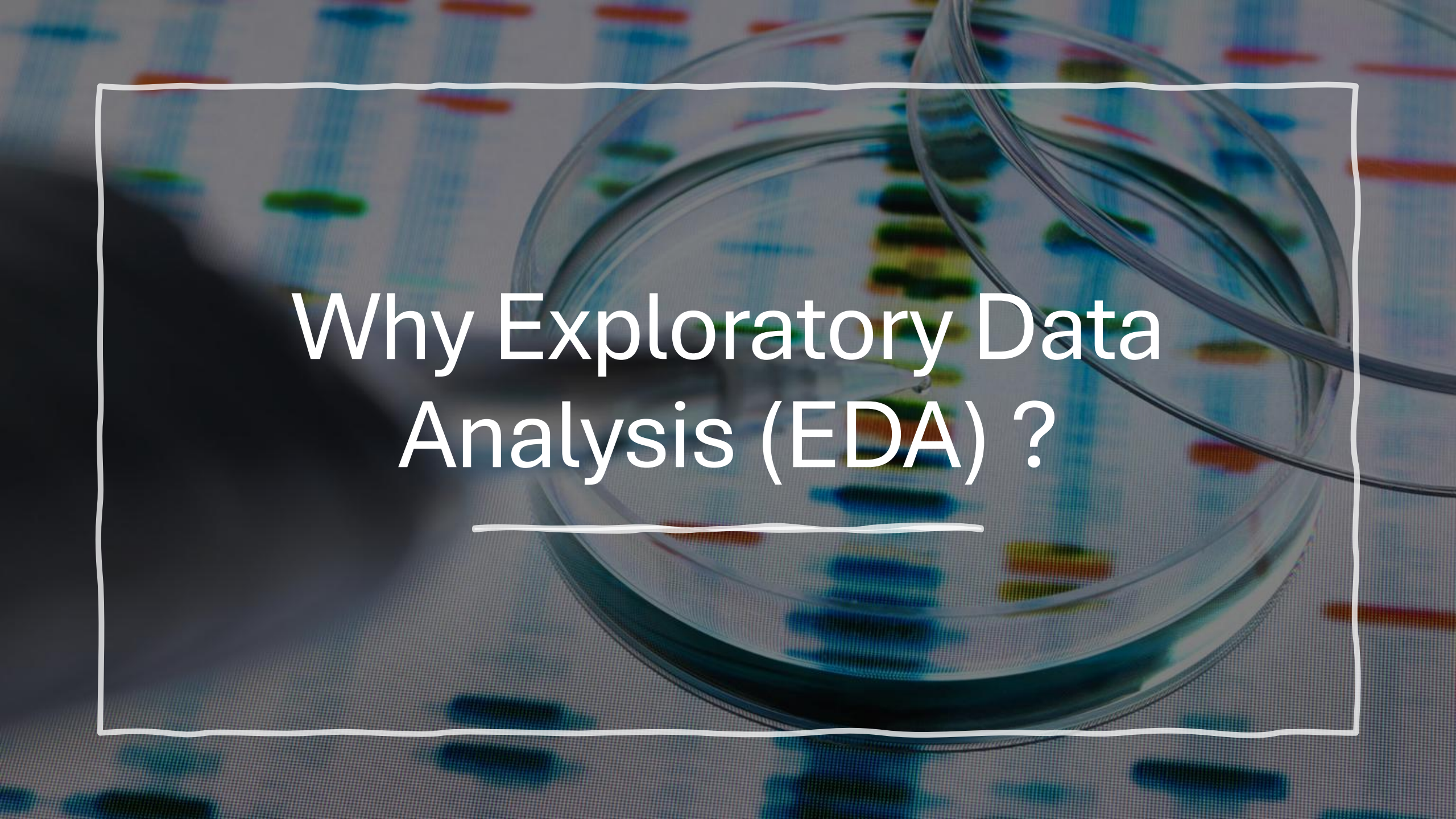
Introduction to EDA & Python Libraries

By Sakthi Arul



Why Exploratory Data Analysis (EDA) ?

Ice Cream Dataset

Flavor	Price (₹)	Sales (Units)	Sugar Content (g)	Customer Rating (1-5)
Chocolate	35	200	25	4.8
Vanilla	30	180	20	4.5
Strawberry	35	150	22	4.2
Mint Chip	40	100	28	4.0
Cookie Dough	45	50	30	5.0
Mango	40	0	18	3.5

Identify Patterns and Relationships

Flavor	Price (₹)	Sales (Units)	Sugar Content (g)	Customer Rating (1-5)
Chocolate	35	200	25	4.8
Vanilla	30	180	20	4.5
Strawberry	35	150	22	4.2
Mint Chip	40	100	28	4.0
Cookie Dough	45	50	30	5.0
Mango	40	0	18	3.5

Spot Anomalies: Catch Outliers or Errors

Flavor	Price (₹)	Sales (Units)	Sugar Content (g)	Customer Rating (1-5)
Chocolate	35	200	25	4.8
Vanilla	30	180	20	4.5
Strawberry	35	150	22	4.2
Mint Chip	40	100	28	4.0
Cookie Dough	45	50	30	5.0
Mango	40	0	18	3.5

Treasure Hunter

Treasure Hunter

Data Scientist

Treasure Hunter

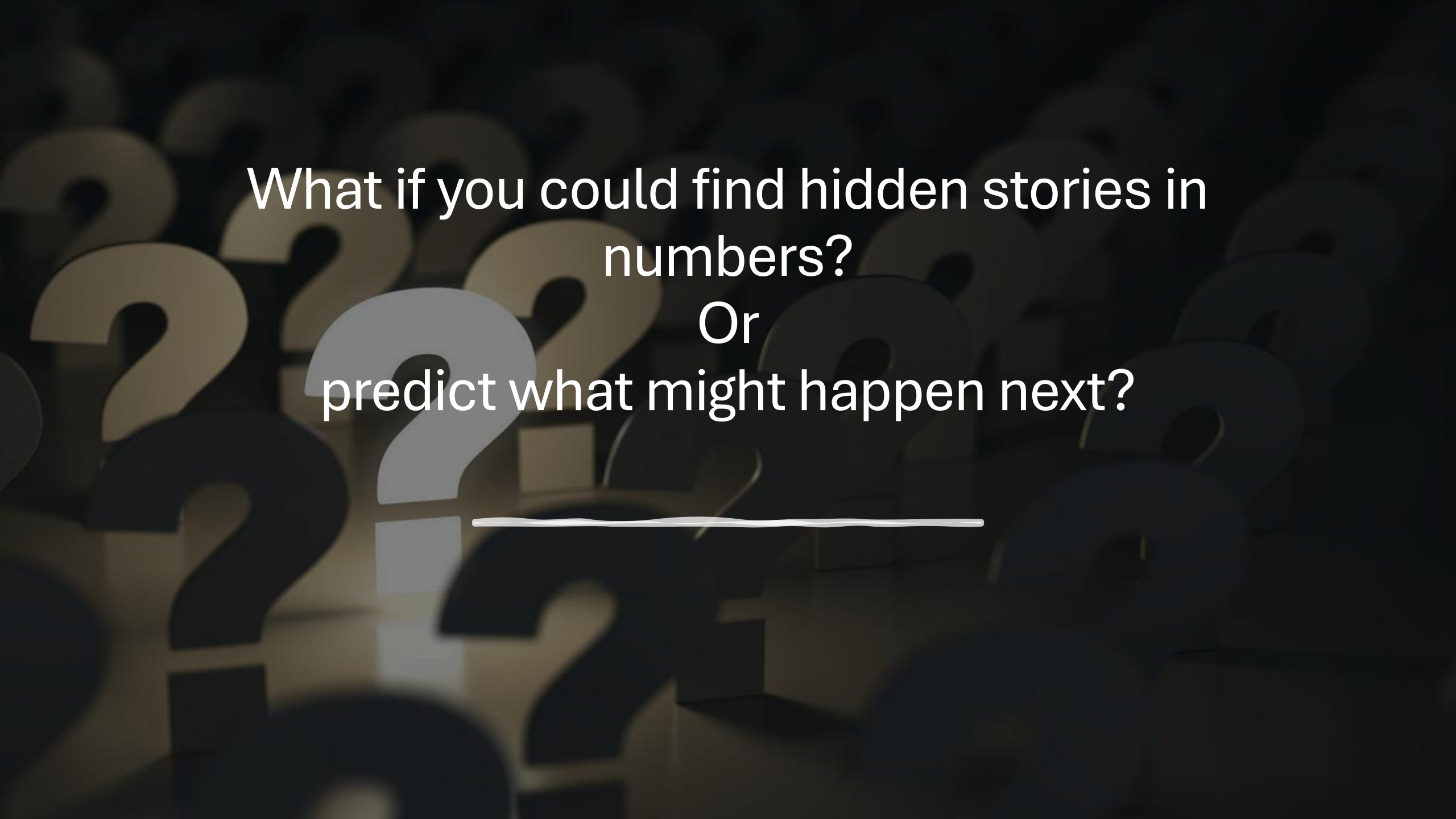
- Treasure Trove Location
- Looking for Clues
- Digging
- Treasure

Data Scientist

- Dataset
- Looking Patterns or Anomalies
- Manipulating the data
- Insights from data

What is EDA?

- Exploratory Data Analysis is the process of investigating datasets to discover patterns, spot anomalies, test hypotheses, and check assumptions using statistical summaries and graphical representations.


What if you could find hidden stories in numbers?
Or
predict what might happen next?

Key Objective

- Understand the structure and characteristics of data
- Identify patterns, trends, and relationships
- Detect outliers and anomalies
- Validate assumptions before modeling
- Generate hypotheses for further investigation

When poll is active respond at **PollEv.com/sakthiarul800**

When you hear "Data Exploration," what words come to mind?

0

Join by Web

PollEv.com/sakthiarul800

Join by QR code

Scan with your camera app

Role of EDA in Data Science

- **Data Quality Assessment** - Identifies missing values, duplicates, and errors
- **Feature Understanding** - Reveals which variables are important

Role of EDA in Data Science

- **Relationship Discovery** - Uncovers correlations and dependencies
- **Model Selection Guidance** - Informs appropriate algorithm choices
- **Storytelling Foundation** - Creates narratives from data insights

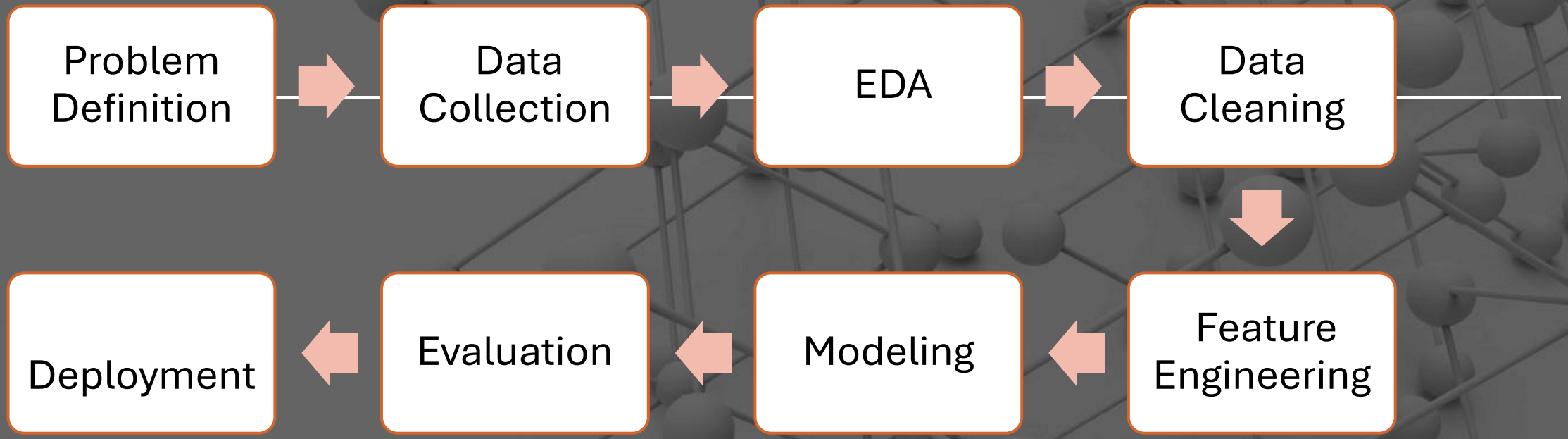
Why EDA Matters ?

- Improves data quality and model performance
- Reveals hidden biases and edge cases
- Guides feature engineering and modeling choices

Why Does Exploratory Data Analysis Matter?

- **Catch unexpected problems early:**
- **Find patterns you wouldn’t see otherwise:**
- **Clarify what’s actually important**
- **Guide better decisions:**

EDA in the Data Science Pipeline:



Python libraries

- NumPy
- Pandas
- Matplotlib & Seaborn

NumPy Operations: Broadcasting and Vectorization

Python Recap: Data Types, Loops, Functions

Installing numpy

- pip install numpy
- %pip install numpy
- conda install numpy

Check the version

- import numpy as np
- print(np.__version__)

Types of Arrays

- 1D Array: np.array([1,2,3])
- 2D Array (Matrix): np.array([[1,2,3], [4,5,6]])
- 3D Array (Tensor-like): np.array([[[1,2], [3,4]], [[5,6], [7,8]]])

Array Properties & Manipulation

- print(arr.shape) # (Rows, Columns)
- print(arr.size)   # Total number of elements
- print(arr.dtype) # Data type

number of dimensions

- The `ndim` attribute in NumPy is used to determine the number of dimensions (axes) of an array.

```
# 1D array
arr_1d = np.array([1, 2, 3])
print(arr_1d.ndim) # Output: 1

# 2D array
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr_2d.ndim) # Output: 2

# 3D array
arr_3d = np.zeros((2, 3, 4))
print(arr_3d.ndim) # Output: 3
```

Flattening an Array

- Arr = np.array([[1,2,3],[4,5,6]])
- a= arr.flatten()
- print(a)

Reshaping Arrays

- arr = np.array([1,2,3,4,5,6])
- reshaped_arr = arr.reshape(2,3)
- print(reshaped_arr)

NumPy Arrays - Basics

Creating NumPy Arrays

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
matrix = np.array([[1, 2, 3], [4, 5, 6]])
```

Quick test

- compares the speed of operations between a normal Python list and a NumPy array:

special functions

- zeros_matrix = np.zeros((3, 4)) # 3x4 matrix of zeros
- ones_matrix = np.ones((2, 3))   # 2x3 matrix of ones
- rand_matrix = np.random.rand(2, 2) # 2x2 matrix of random values

question

- Create a 3×3 NumPy array filled with zeros and a separate 4×2 array filled with ones.
- Create an array with values from 10 to 50 with a step of 5.
- Convert a 1D array of values [1,2,3,4,5,6,7,8,9] into a 3×3 matrix.

Arithmetic Operations

```
g = a - b
 array([[-0.5, 0. , 0. ],
        [-3. , -3. , -3. ]])
np.subtract(a,b)
b + a
array([[ 2.5, 4. , 6. ],
[ 5. , 7. , 9. ]])
np.add(b,a)
a  b
array([[ 0.66666667, 1. , 1. ],
[ 0.25 , 0.4 , 0.5 ]])
```

Arithmetic Operations

- np.divide(a,b)
- a b
- np.multiply(a,b)
- np.exp(b)
- np.sqrt(b)
- np.sin(a)
- np.cos(b)
- np.log(a)

Element-wise Comparisons

- a = np.array([1, 2, 3, 4])
- b = np.array([2, 2, 3, 5])

- print(a == b) # [False True True False]
- print(a != b) # [True False False True]
- print(a > b)   # [False False False False]
- print(a < b)   # [True False False True]

- import numpy as np

- # From a list
- arr1 = np.array([1, 2, 3, 4, 5])

- # Using NumPy functions
- arr2 = np.zeros(5)
- arr3 = np.ones(5)
- arr4 = np.arange(0, 10, 2)
- arr5 = np.linspace(0, 1, 5)

Questions

- you are analysing customer ratings of a product collected from different regions. Store the following ratings in a 2D NumPy array and find the average rating per region.
- ratings = np.array([[4, 5, 3], [5, 5, 4], [3, 4, 2]])

Array Indexing

- 1D Array Indexing:
- arr = np.array([10, 20, 30, 40, 50])
- print(arr[0]) # 10 (first element)
- print(arr[-1]) # 50 (last element)

Array Indexing

- 2D Array Indexing:
- matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

- # Access element at row 1, column 2
- print(matrix[1, 2]) # 6

- # Access entire row
- print(matrix[0])   # [1 2 3]

- # Access entire column
- print(matrix[:, 1]) # [2 5 8]

Array Slicing

- 1D Array Slicing: array[start:stop:step]

Questions

- *Problem:* You have temperature data for 7 days. Extract temperatures from Monday to Wednesday.

Mathematical Operations

- Element-wise addition
- Arr 1 + arr 2
- Element-wise subtraction
- Arr1 – arr2
- # Element-wise v multiplication
- Arr1 * arr2
- # Element-wise division
- Arr1/ arr2

- Arr1**2

- np.sqrt(arr1)

Question

- A factory produces 3 types of products daily. The production per day is given. Find the total weekly production.

- daily_production = np.array([500, 700, 800])

Statistical function

- data = np.array([5, 10, 15, 20, 25])

- print(np.mean(data)) # Average
- print(np.median(data)) # Median
- print(np.std(data)) # Standard Deviation
- print(np.var(data)) # Variance

Problem

- *:* A school wants to analyze the height of students. Find the average and standard deviation

- heights = np.array([160, 165, 170, 155, 180])

Other operation

- Transpose
- a = np.array([[1, 2], [3, 4]])

- print(a.T)
- # [[1 3]
- # [2 4]]

Inverse

- a = np.array([[1, 2], [3, 4]])

- print(np.linalg.inv(a))
- # [[-2. 1.]
- # [1.5 -0.5]]

loc

Iloc

Broadcasting

- NumPy's way to perform operations on arrays of different shapes without explicit reshaping— "stretches" smaller arrays to match larger ones.

Broadcasting

Benefits: Saves memory and time; no need for loops or manual replication.

Basic Broadcasting -Code

```
import numpy as np

array1 = np.array([1, 2, 3])
array2 = 10

result = array1 + array2
print(result)
```

Broadcasting with Different Dimensions

```
import numpy as np

array1 = np.array([[1, 2, 3], [4, 5, 6]])
array2 = np.array([10, 20, 30])

result = array1 + array2
print(result)
```

Broadcasting with Multi-Dimensional Arrays

```
import numpy as np

array1 = np.array([[1, 2, 3], [4, 5, 6]])
array2 = np.array([[10], [20]])

result = array1 + array2
print(result)
```

Broadcasting Rules

Rule 1: Start from trailing (rightmost) dimensions.

Rule 2:Two dimensions are compatible when they are equal, or one of them is 1.

Vectorization

- Applying operations to entire arrays at once, using optimized C-backend code—no explicit loops.

- **Benefits:** 10-100x faster than Python loops; less code, fewer errors.

Vectorization– Code Test Run

- import numpy as np
- data = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
- sines = np.sin(data) # Vectorized: sin() on whole array
- print(sines)

Vectorization vs. Loops

- import numpy as np
- import time
- data = np.random.rand(1000000) # 1M random numbers

- # Loop version
- start = time.time()
- sines_loop = [np.sin(x) for x in data]
- loop_time = time.time() - start
- print("Loop time:", loop_time, "seconds")

- # Vectorized version
- start = time.time()
- sines_vec = np.sin(data)
- vec_time = time.time() - start
- print("Vectorized time:", vec_time, "seconds")

Questions

- Create a 3x4 array of random integers (0-10). Add a 1D array [1, 2, 3, 4] to each row. What’s the output shape? (Hint: Use np.random.randint).