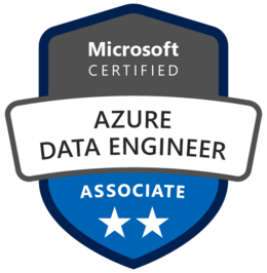
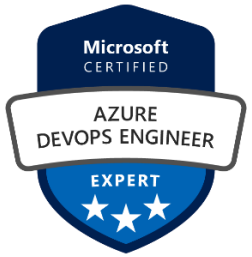
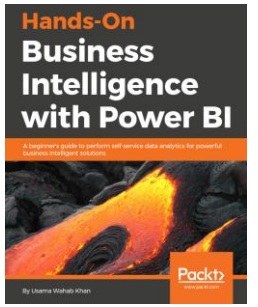
Introduction to Data Science & Python Basics - Dash courses



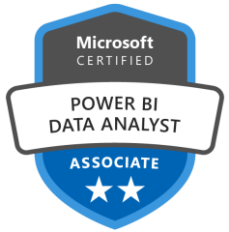




Usama Wahab Khan

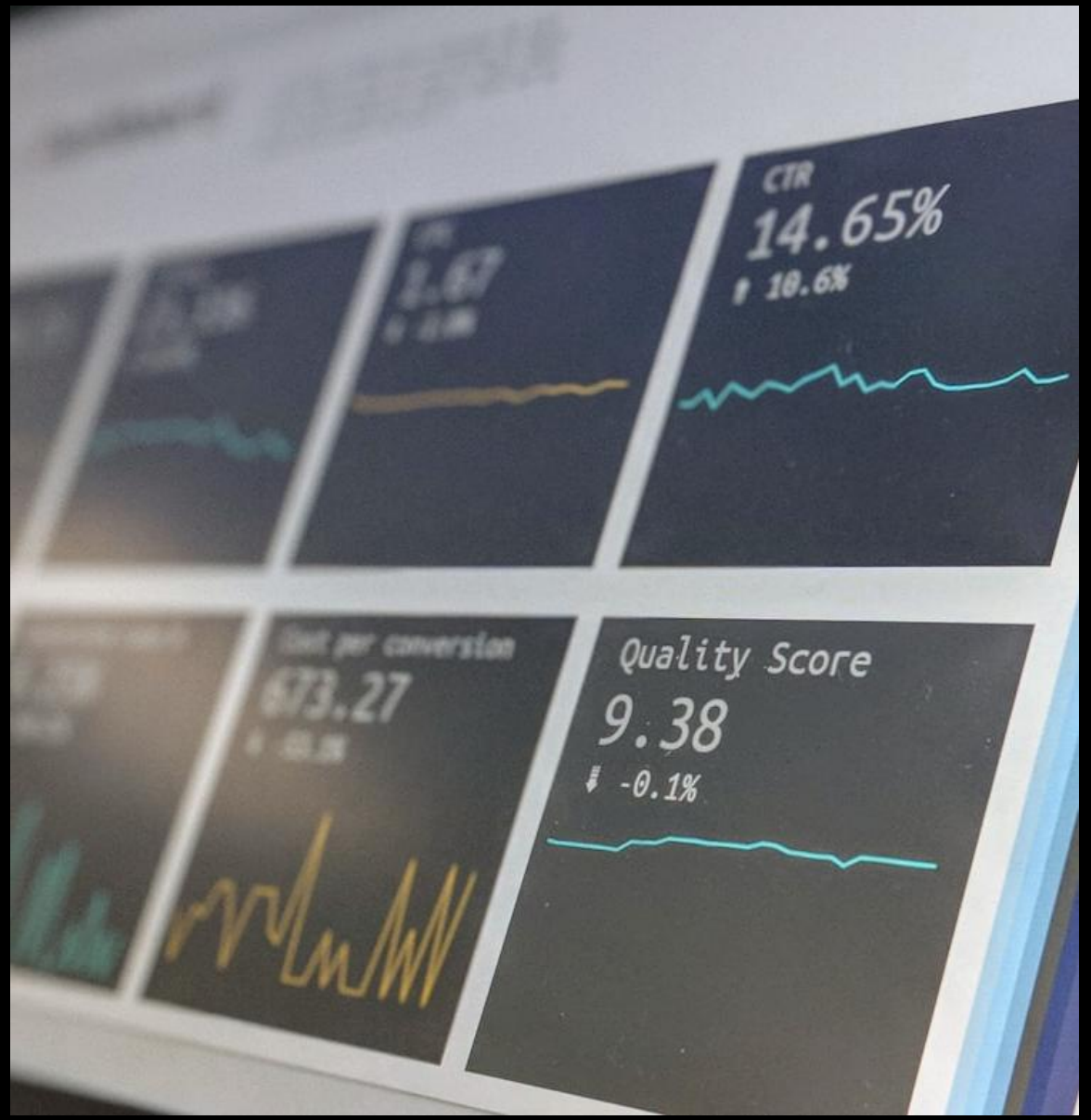
- Father, data Scientist, Developer/Nerd, Traveler

Twitter : @usamawahabkhan
LinkedIn : Usamawahabkhan



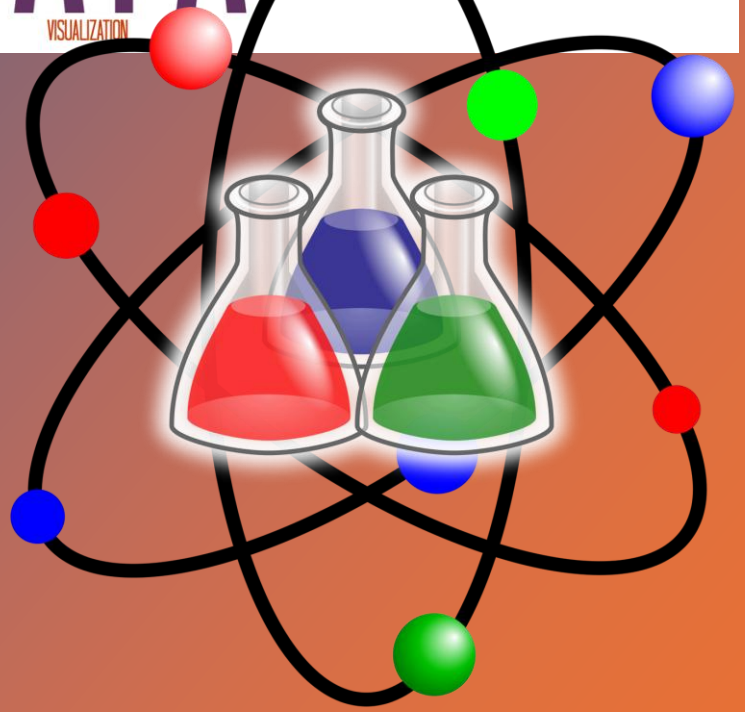
WHAT IS DATA SCIENCE?

Data Science is an interdisciplinary field that combines statistics, computer science, and domain expertise to extract meaningful insights from structured and unstructured data.



Not to be basic but...

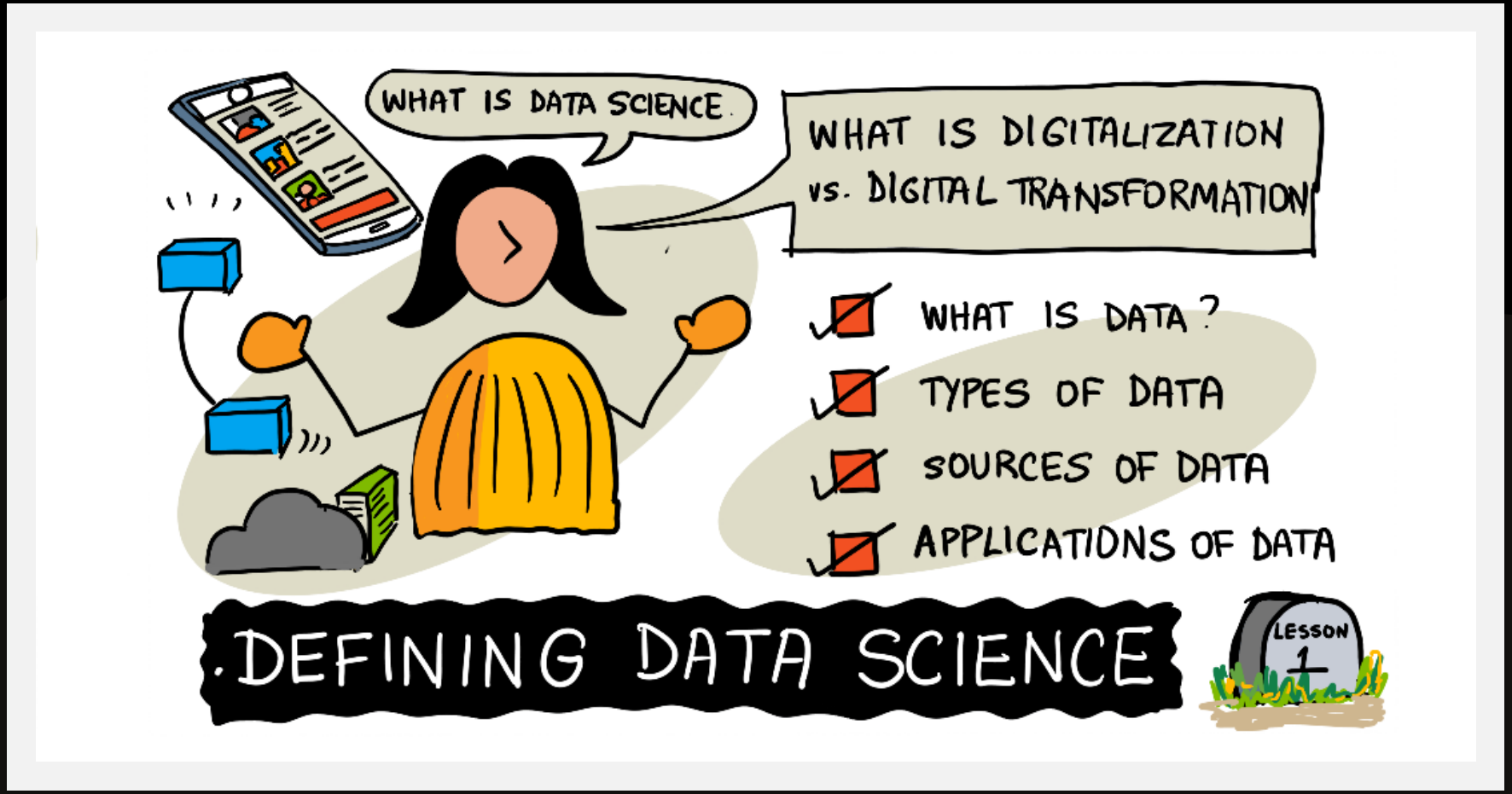
What is Data Science?



What is Data Science?

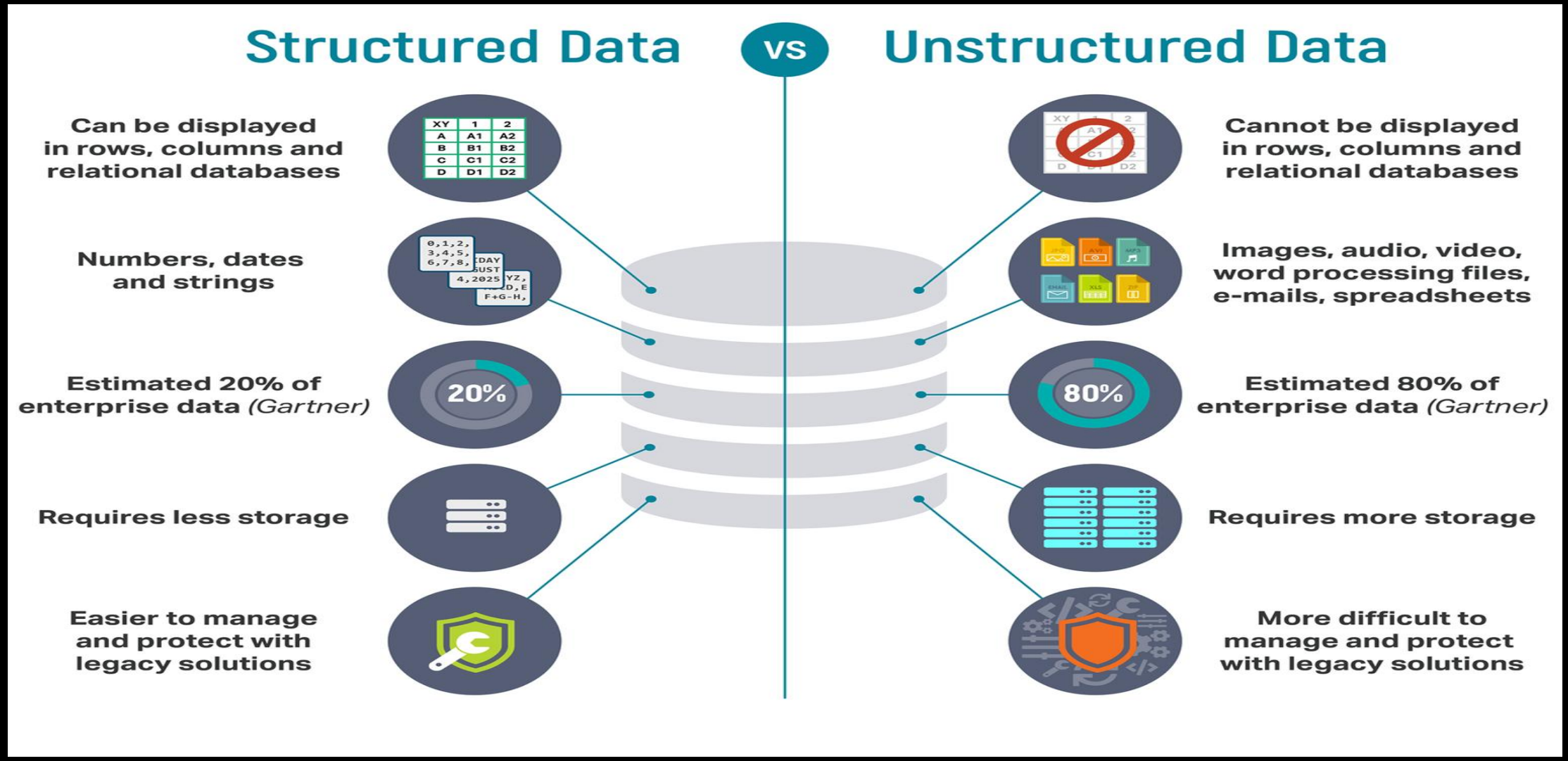
- **Extraction of knowledge from large volumes of data** that are structured or unstructured.

- It is a continuation of the fields **data mining** and **predictive analytics**



Defining Data Science

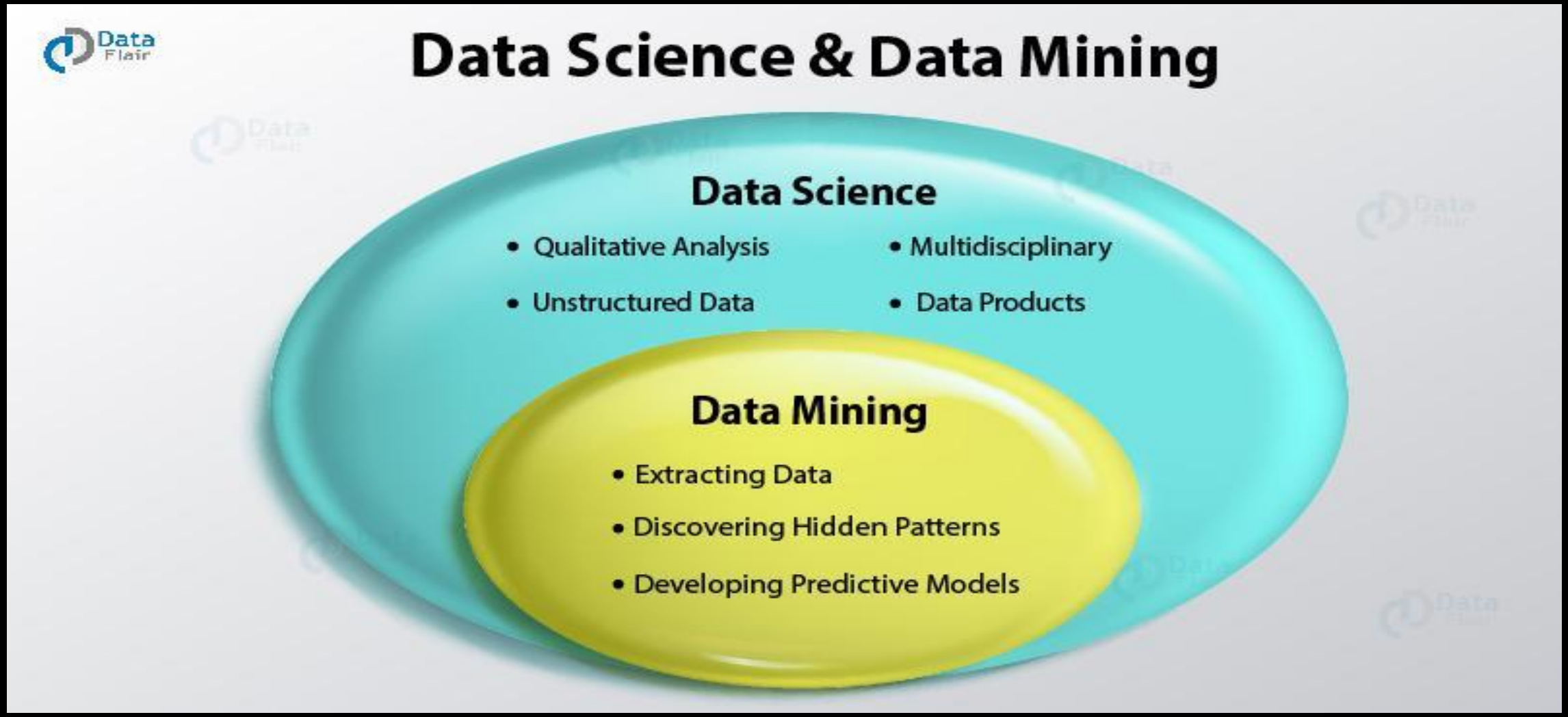
Difference Between Structured and Unstructured Data?



Structured vs. Semi-Structured vs Unstructured Data

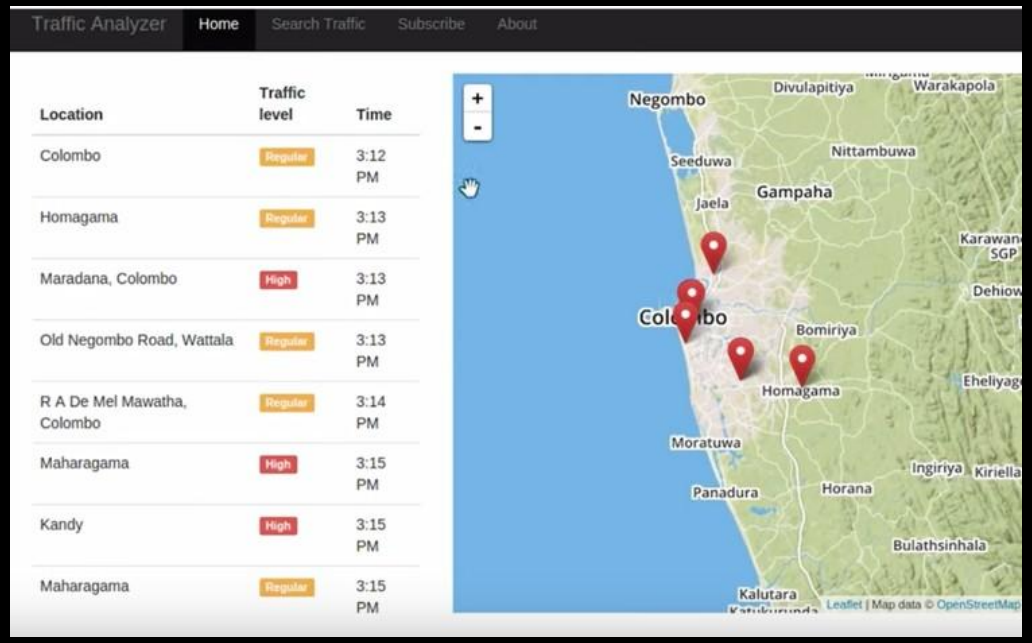
Type	Definition	Examples	Storage Format	Ease of Analysis
Structured Data	Organized in rows and columns with a fixed schema	SQL databases, Excel sheets, sensor logs	Relational databases (RDBMS)	Easy
Semi-Structured Data	Has some organizational properties but no strict schema	JSON, XML, NoSQL databases, HTML	NoSQL, XML/JSON files	Moderate
Unstructured Data	No predefined format or structure	Emails, social media, images, videos	File systems, data lakes	Complex

Essence of Data Mining



Some examples where Data Science might be used

1. Data as tweets
2. Extract time, location, and traffic level using NLP
3. Explore data
4. Model based on time, and it is a holiday
5. Predict traffic given a time and location.



Core Components

- Data Collection & Cleaning
- Exploratory Data Analysis (EDA)
- Statistical Modeling & Machine Learning
- Data Visualization
- Deployment & Decision Making

🧰 Key Goals of EDA

- **Understand data distribution** (e.g., mean, median, variance)
- **Identify patterns and trends** (e.g., seasonal effects, correlations)
- **Detect anomalies or outliers**
- **Check assumptions for modeling**
- **Spot missing or inconsistent data**

📈 Statistical Modeling & Machine Learning

- **Definition**:
  Statistical modeling uses mathematical equations to represent relationships between variables in data. It helps in understanding patterns and making inferences.
- **Examples**:
- **Linear Regression**: Predicting a continuous value (e.g., house prices)
- **Logistic Regression**: Predicting binary outcomes (e.g., spam or not spam)

📈 Statistical Modeling & Machine Learning

- **Definition**:
  Machine Learning (ML) is a subset of AI that enables systems to learn from data and make predictions or decisions without being explicitly programmed.
- **Types of ML**:
- **Supervised Learning**: Learns from labeled data (e.g., classification, regression)
- **Unsupervised Learning**: Finds patterns in unlabeled data (e.g., clustering)
- **Reinforcement Learning**: Learns through trial and error (e.g., game AI)

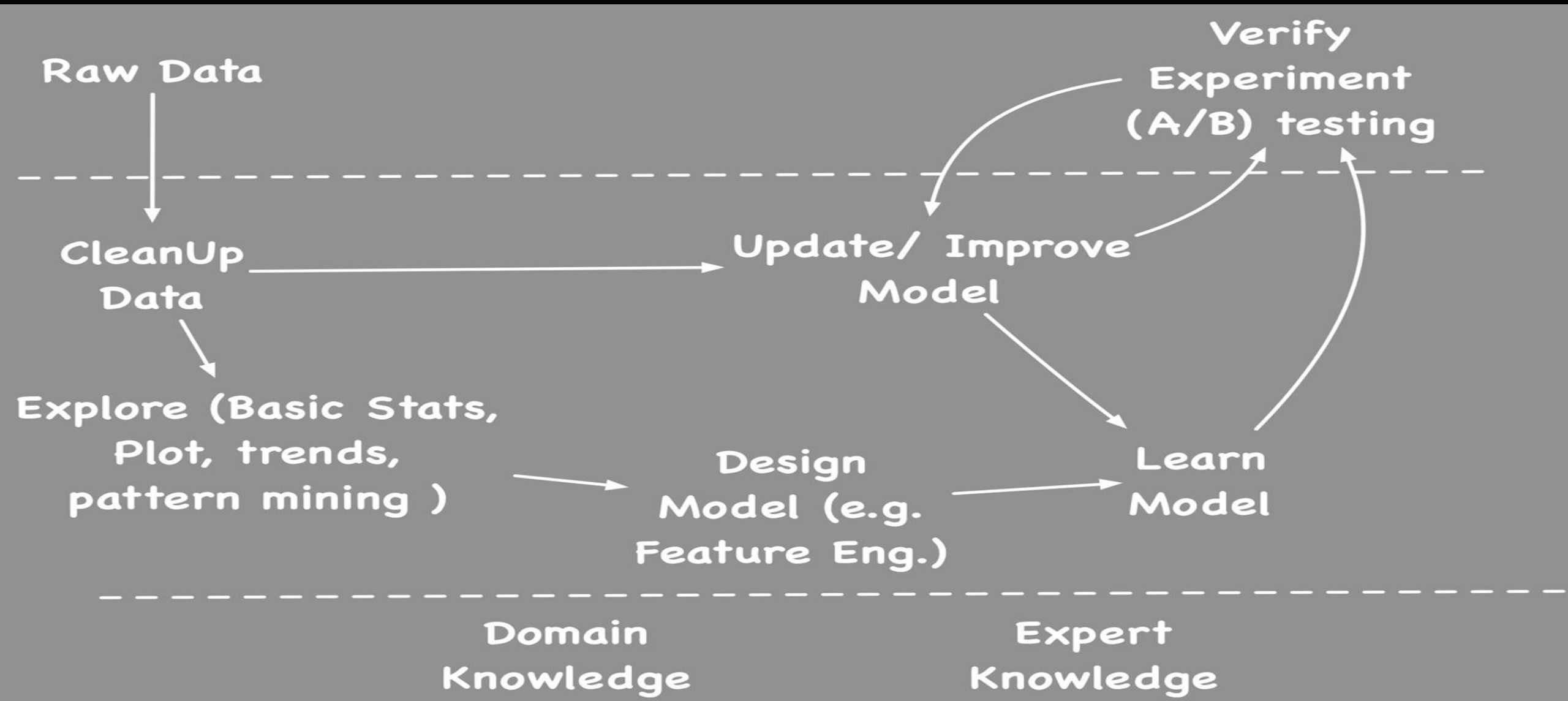
Key Aspects of Data Science

- **Data Collection**: Gathering raw data from various sources.
- **Data Cleaning & Preparation**: Making data usable by handling missing values, errors, and inconsistencies.
- **Exploratory Data Analysis (EDA)**: Understanding patterns, trends, and anomalies in the data.
- **Modeling & Algorithms**: Applying statistical models and machine learning to make predictions or classifications.
- **Interpretation & Communication**: Visualizing and explaining results to guide decision-making.

Core Components

- Data Collection & Cleaning
- Exploratory Data Analysis (EDA)
- Statistical Modeling & Machine Learning
- Data Visualization
- Deployment & Decision Making

Data Science Pipeline



Data Science Pipeline

1. **Problem Definition**
    1. Understand the business or research problem.
    2. Define goals and success metrics.
2. **Data Collection**
    1. Gather data from various sources (databases, APIs, web scraping, sensors, etc.).
3. **Data Cleaning & Preprocessing**
    1. Handle missing values, duplicates, and errors.
    2. Normalize, encode, or transform data as needed.
4. **Exploratory Data Analysis (EDA)**
    1. Use statistics and visualizations to understand patterns, trends, and anomalies.
5. **Feature Engineering**
    1. Create new features or select important ones to improve model performance.
1. **Modeling**
    1. Apply machine learning or statistical models.
    2. Train, validate, and test models.
2. **Evaluation**
    1. Assess model performance using metrics like accuracy, precision, recall, RMSE, etc.
3. **Deployment**
    1. Integrate the model into a production environment (e.g., web app, API).
4. **Monitoring & Maintenance**
    1. Track model performance over time.
    2. Update the model as data or business needs chang

statistical analysis performed using Python, along with visualizations:

1.Descriptive Statistics

1. Summarizes data using measures like mean, median, mode, standard deviation.
2. Example: Average test score of students.

🧠 How They Work Together

- **Descriptive** gives you the *what*.
- **Diagnostic** gives you the *why*.
- **Predictive** gives you the *what next*.
- **Prescriptive** gives you the *what now*.
- **Cognitive** gives you the *how to think*.

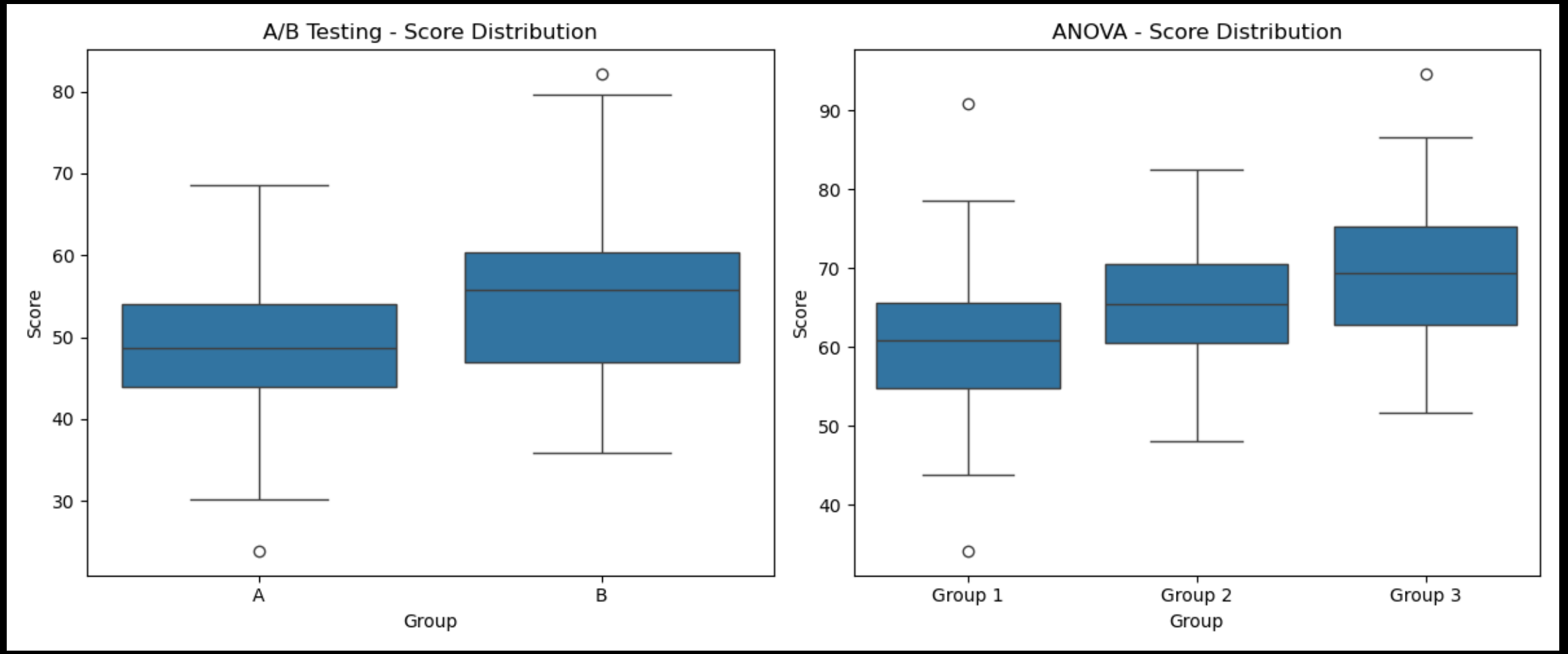
statistical analysis performed using Python, along with visualizations:

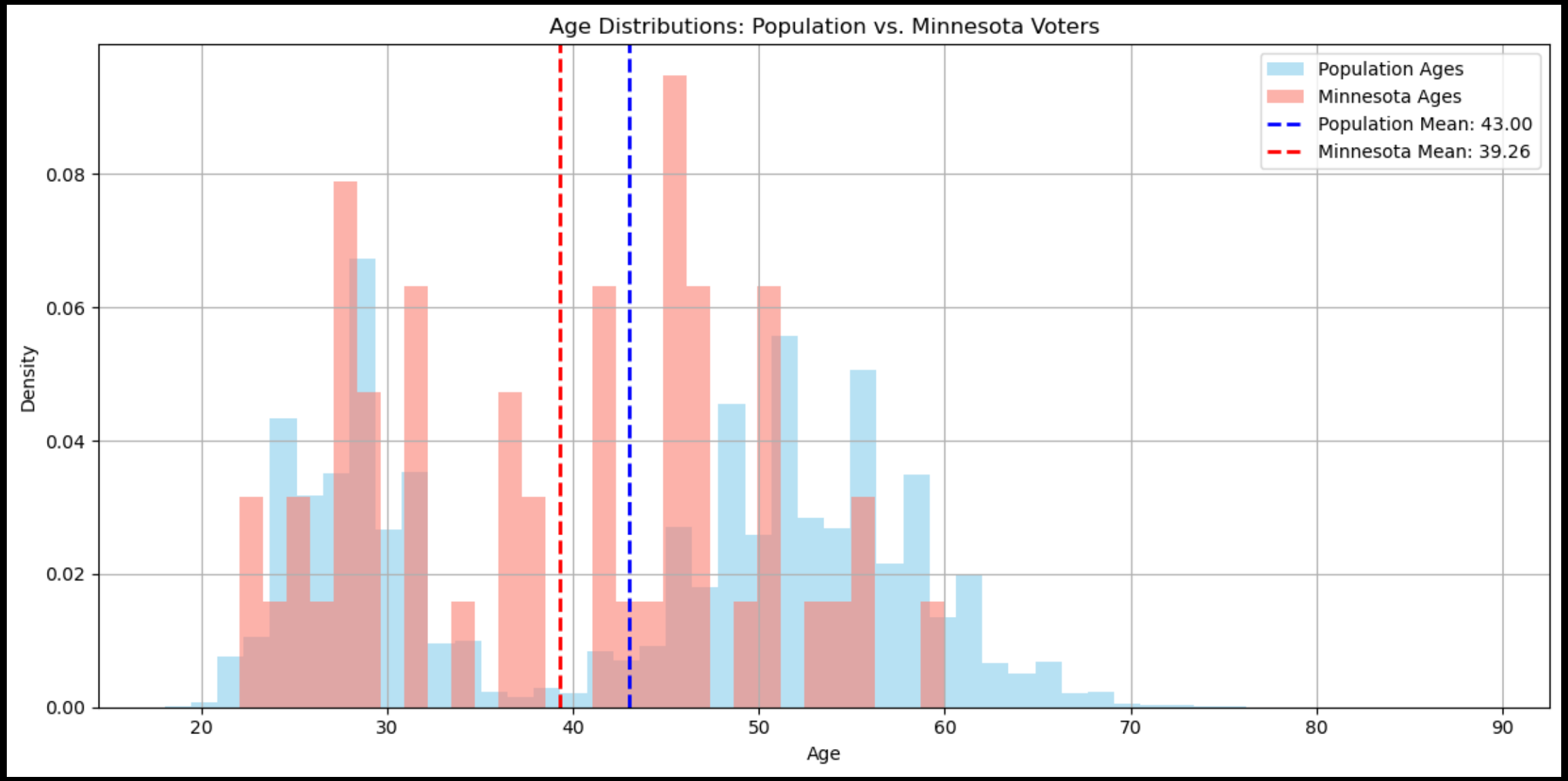
1.Descriptive Statistics

1. Summarizes data using measures like mean, median, mode, standard deviation.
2. Example: Average test score of students.

Hypothesis Testing (t-test)

- **Purpose**: Compare the means of two independent groups (Group A vs. Group B).
- **Result**:
  - **t-statistic** = -4.7547
  - **p-value** ≈ 0.0000
- **Interpretation**: Since the p-value is very small, we reject the null hypothesis—there is a significant difference between the two groups.


A/B Testing - Score Distribution
Score
80
70
60
50
40
30
A
B
Group
ANOVA - Score Distribution
Score
90
80
70
60
50
40
Group 1
Group 2
Group 3
Group


Age Distributions: Population vs. Minnesota Voters
Population Ages
Minnesota Ages
Population Mean: 43.00
Minnesota Mean: 39.26
Density
0.00
0.02
0.04
0.06
0.08
20
30
40
50
60
70
80
90
Age

🔍 What is Machine Learning?

Machine Learning (ML) is a subset of artificial intelligence (AI) that enables systems to **learn from data** and make predictions or decisions **without being explicitly programmed**.

🧠 Types of Machine Learning

Type	Description	Examples
Supervised Learning	Learns from labeled data (input → output)	Spam detection, house price prediction
Unsupervised Learning	Finds patterns in unlabeled data	Customer segmentation, topic modeling
Reinforcement Learning	Learns by trial and error through rewards	Game AI, robotics, self-driving cars

Common Algorithms

- Linear Regression
- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- K-Means Clustering
- Neural Networks

Applications

- Fraud detection
- Recommendation systems (Netflix, Amazon)
- Image and speech recognition
- Predictive maintenance
- Personalized marketing

🎯 Supervised Learning

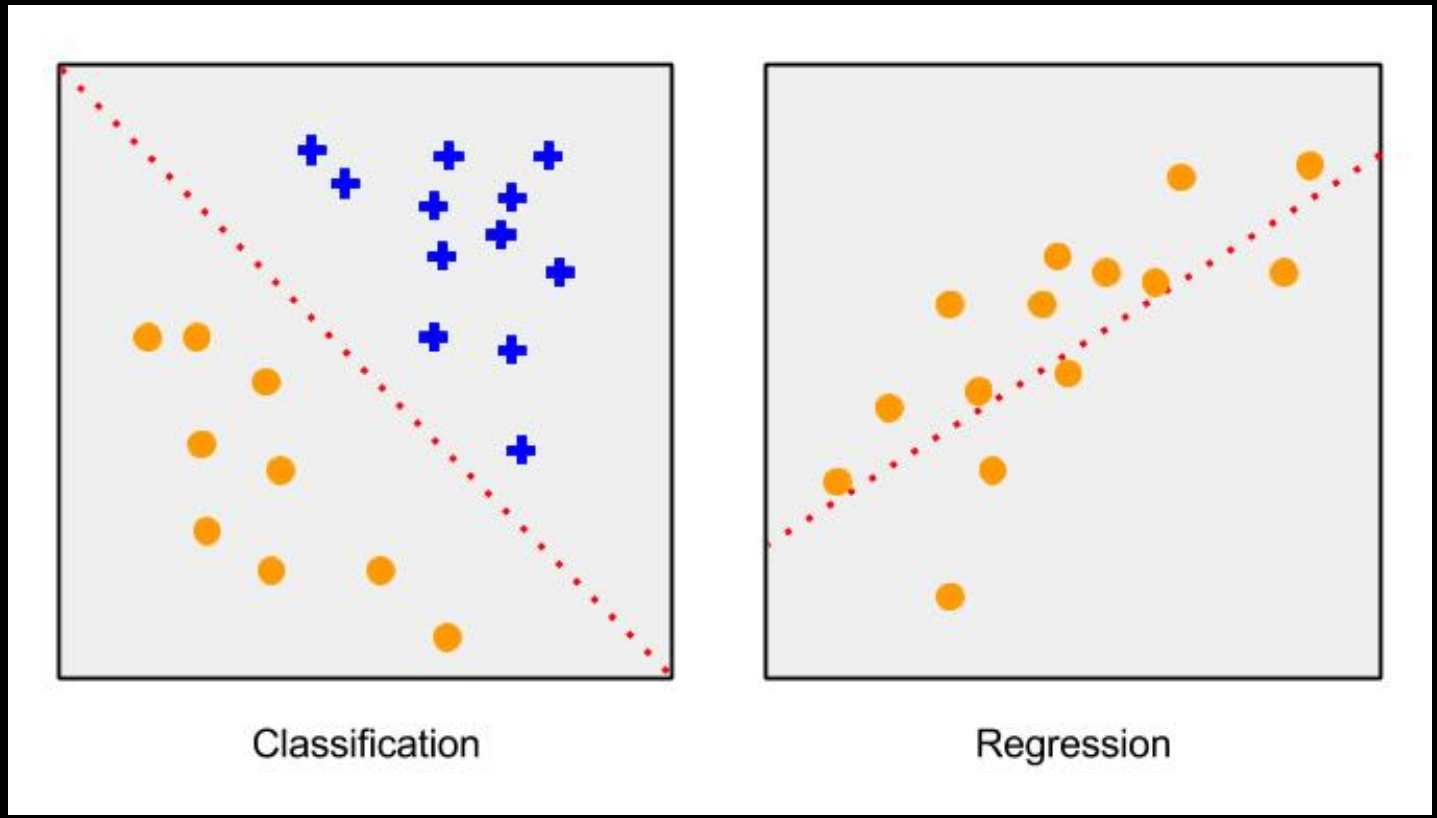
🔍 Definition

Supervised learning is a type of machine learning where the model is trained on a **labeled dataset**—that is, each training example includes both the **input data** and the **correct output** (label). The goal is for the model to learn a mapping from inputs to outputs so it can predict the label for new, unseen data.

🧠 How It Works

1. **Input**: Features (e.g., age, income, hours studied)
2. **Output**: Labels (e.g., pass/fail, price, category)
3. **Training**: The model learns patterns from the labeled data.
4. **Prediction**: The trained model predicts labels for new data.

This is broken down into two categories regression and classification



📘 Common Algorithms

- **Linear Regression** (for regression)
- **Logistic Regression** (for classification)
- **Decision Trees**
- **Random Forest**
- **Support Vector Machines (SVM)**
- **K-Nearest Neighbors (KNN)**
- **Neural Networks**

🧠 Unsupervised Learning

🔮 Definition

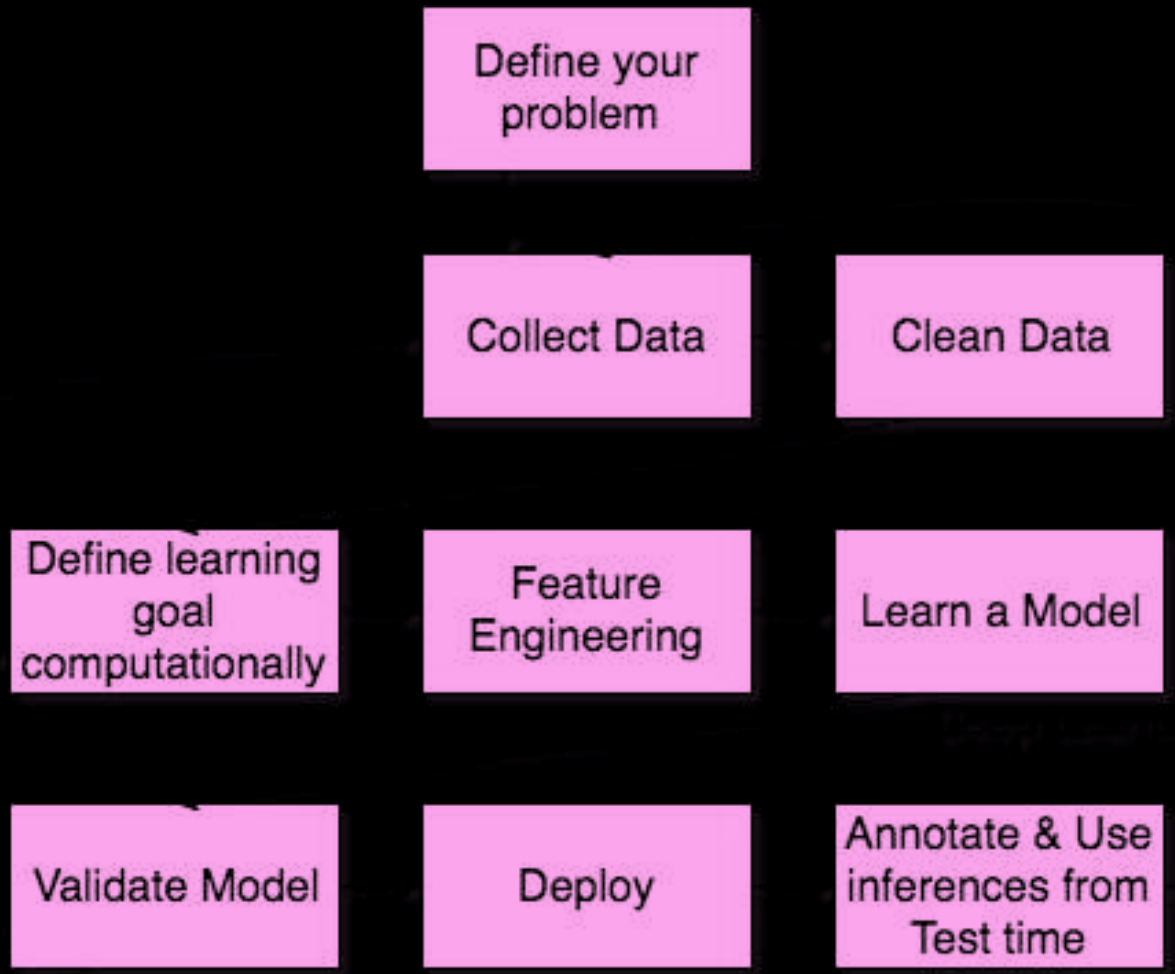
Unsupervised learning is a type of machine learning where the model is trained on **unlabeled data**. The goal is to **discover hidden patterns or structures** in the data without any predefined labels or outcomes

Type	Goal	Example
Clustering	Group similar data points together	Customer segmentation, document grouping
Dimensionality Reduction	Reduce the number of variables while preserving structure	Data visualization, noise reduction
Association Rules	Discover relationships between variables	Market basket analysis (e.g., “people who buy X also buy Y”)

Popular Algorithms

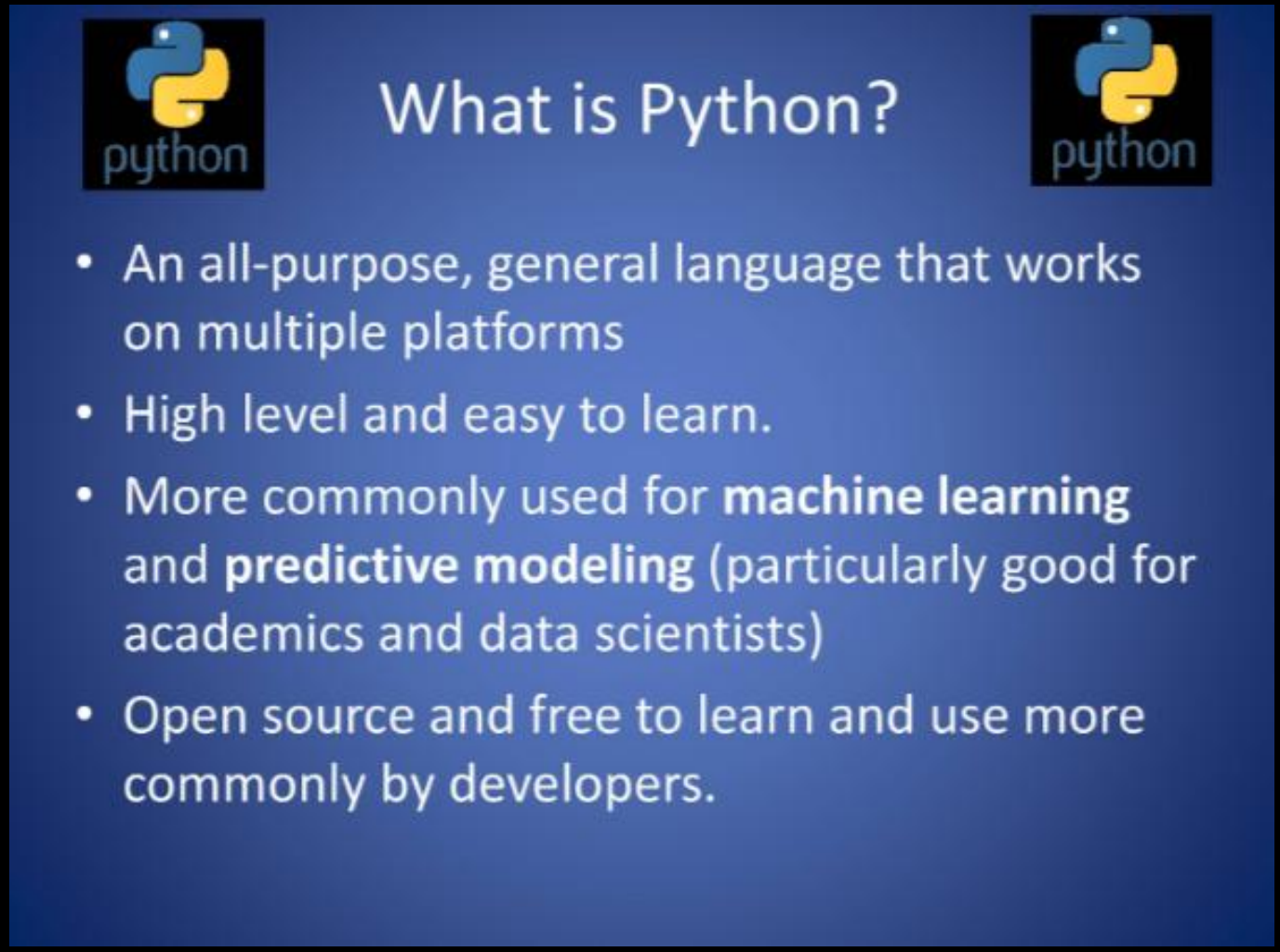
- K-Means Clustering
- Hierarchical Clustering
- DBSCAN
- Principal Component Analysis (PCA)
- t-SNE (for visualization)

Typical Machine Learning Pipeline

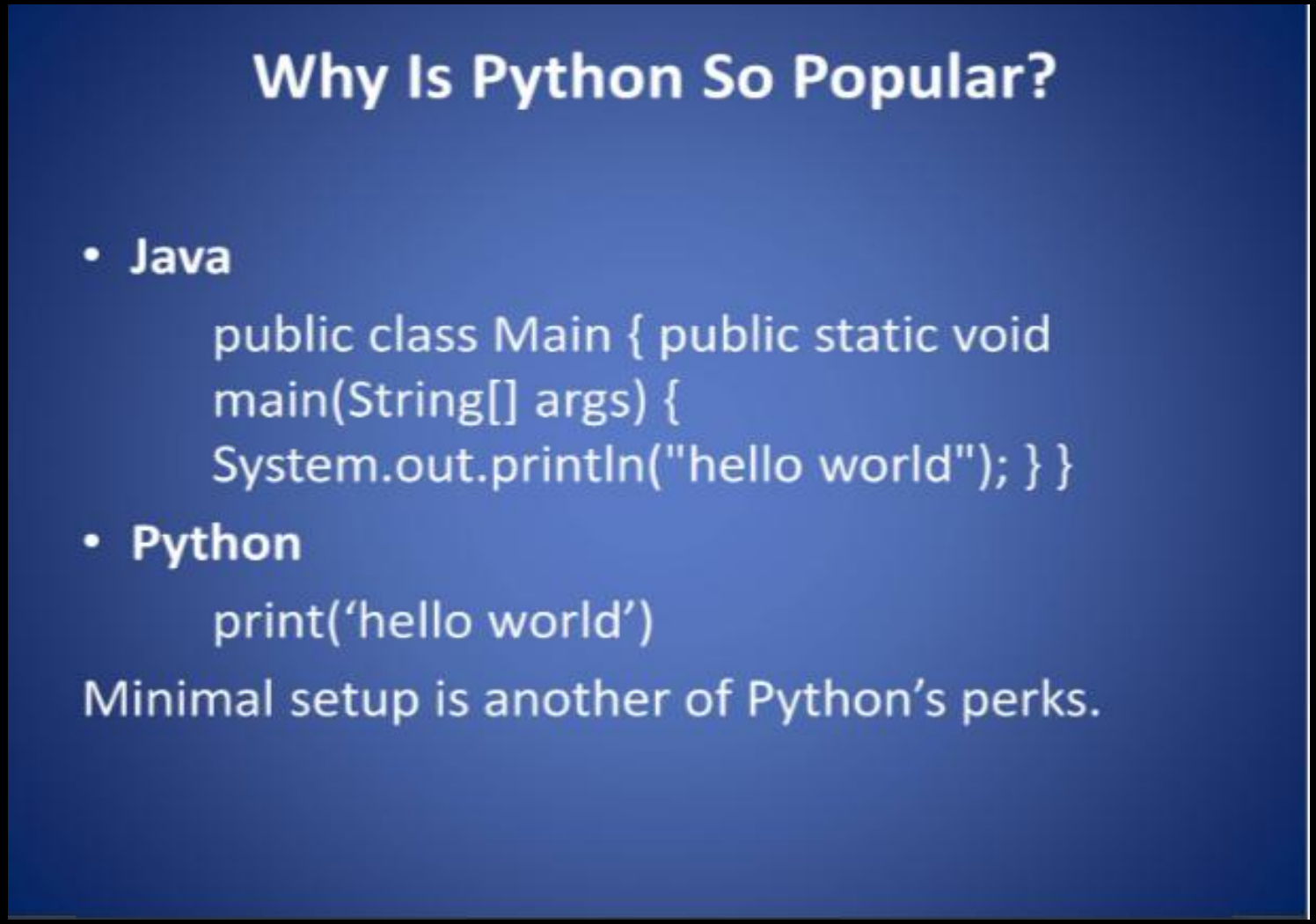


What is the need to use Python?

About Python



About Python



About Python

Why Is Python So Popular?

- The language continued to rank highly on various lists of the world's most popular programming languages.
- Many programmers view Python as a language with a clean syntax and an expansive library.
- Python's massive user base has created something of a positive feedback loop
- In Python's case, it's Google, which uses the programming language in a number of applications (a corporate sponsor).

Overview of data analysis

- Data analysis involves inspecting, cleansing, and modelling data to discover useful information, inform conclusions, and support decision-making. Its purpose is to enhance understanding and improve processes.

TYPES OF DATA ANALYSIS

DESCRIPTIVE ANALYSIS

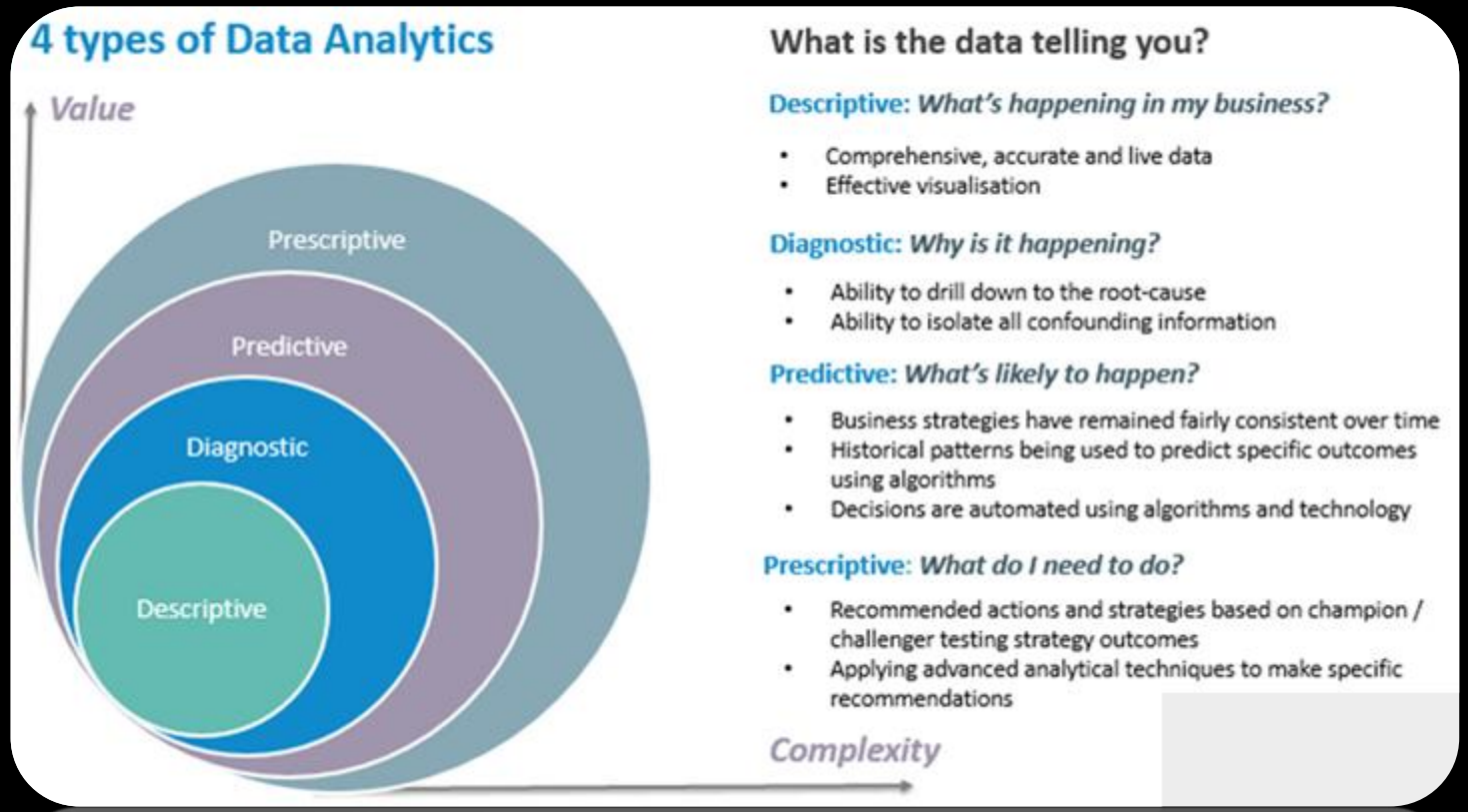
Summarizes past data to identify trends and patterns.

DIAGNOSTIC ANALYSIS

Determines causes of past outcomes through analysis.

PREDICTIVE ANALYSIS

Forecasts future trends based on historical data.



Microsoft

Evolution Technologies

Importance of data analysis in modern software development

Importance of data analysis in modern software development

- Data analysis has become **absolutely critical and deeply intertwined** with modern software development. It's no longer a separate, post-development activity but an integral part of the entire software lifecycle.

Here's a breakdown of its importance:

1. Understanding User Behavior and Improving User Experience (UX)
2. Identifying and Resolving Bugs and Performance Issues
3. Data-Driven Decision Making in Development
4. Building Intelligent and Adaptive Software
5. Understanding Business Value and ROI

Data Types

In data analysis, data is often categorized into three main types based on its structure and the level of organization:

- Structured Data
- Semi-structured Data
- Unstructured Data

Structured Data

Structured data is highly organized data that conforms to a predefined data model (schema).

This makes it easy to store, manage, access, and analyze. It typically resides in relational databases and

spreadsheets, with clear rows and columns defining attributes and records.

Characteristics:

Predefined Schema: Has a fixed format and structure known before the data is stored.

Organized: Arranged in tables with rows (representing records or instances) and columns (representing

attributes or features).

Easily Searchable: Can be efficiently queried and manipulated using structured query languages (like SQL).

Quantitative & Qualitative: Can include both numerical (e.g., sales figures) and categorical (e.g., customer

names, product categories) information, as long as they fit the defined structure.

Lower Flexibility: Changes to the data model can be complex and require modifications across the

entire dataset.

Examples:

•Relational databases (SQL databases)

•Spreadsheets (e.g., Excel, Google Sheets)

•CSV (Comma Separated Values) files

•Tabular data

•Data from point-of-sale systems

•Web form results

•Sensor data with consistent formats

Semi-structured Data

Semi-structured data doesn't conform to a rigid, predefined schema like structured data, but it does have some organizational properties that make it easier to analyze than unstructured data. It contains tags or markers to separate semantic elements and enforce hierarchies within the data.

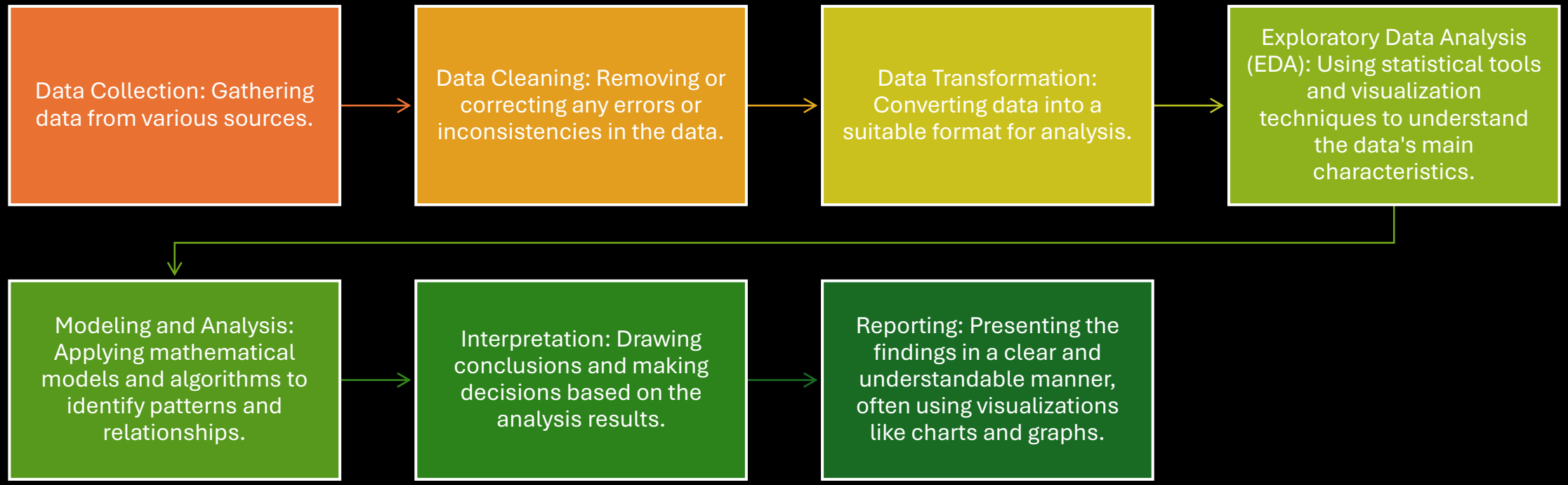
Characteristics:

- Self-Describing to Some Extent: Contains tags or other markers that provide context about the data.
- No Fixed Schema: The structure is not as strict as in relational databases, and attributes can vary between entries.
- Flexibility: Easier to change and evolve compared to structured data.
- Can be Hierarchical: Often organized in nested structures.
- Requires Parsing: Needs to be parsed to extract information for analysis, but the tags provide guidance for this process.
- Examples:
- JSON (JavaScript Object Notation): Uses key-value pairs and nested objects.
- XML (Extensible Markup Language): Uses tags to define elements and attributes.
- HTML (Hypertext Markup Language): Tags define the structure of web pages.CSV files with metadata:
- For example, a header row describing the columns.Log files: Contain timestamps and event information, often with some consistent formatting but varying messages.

Unstructured Data

- Unstructured data does not have a predefined format or organization. It's often text-heavy but can also include non-textual data types. Analyzing unstructured data is more challenging as it requires specialized tools and techniques to extract meaningful information.
- Characteristics:
- No Predefined Schema: Lacks a specific structure, making it difficult to fit into traditional databases.
- Complex and Varied: Can include different data types and formats within the same dataset.
- Difficult to Search and Analyze Directly: Requires techniques like Natural Language Processing (NLP), text mining, and image/video analysis.
- Large Volume: Often constitutes the majority of data in modern systems.
- Rich in Information: Can contain valuable insights into opinions, sentiments, and complex relationships.
- Examples:Text documents (e.g., Word documents, PDFs)
- Emails (the body content)
- Social media posts
- Audio files
- Video files

Data analysis is the process

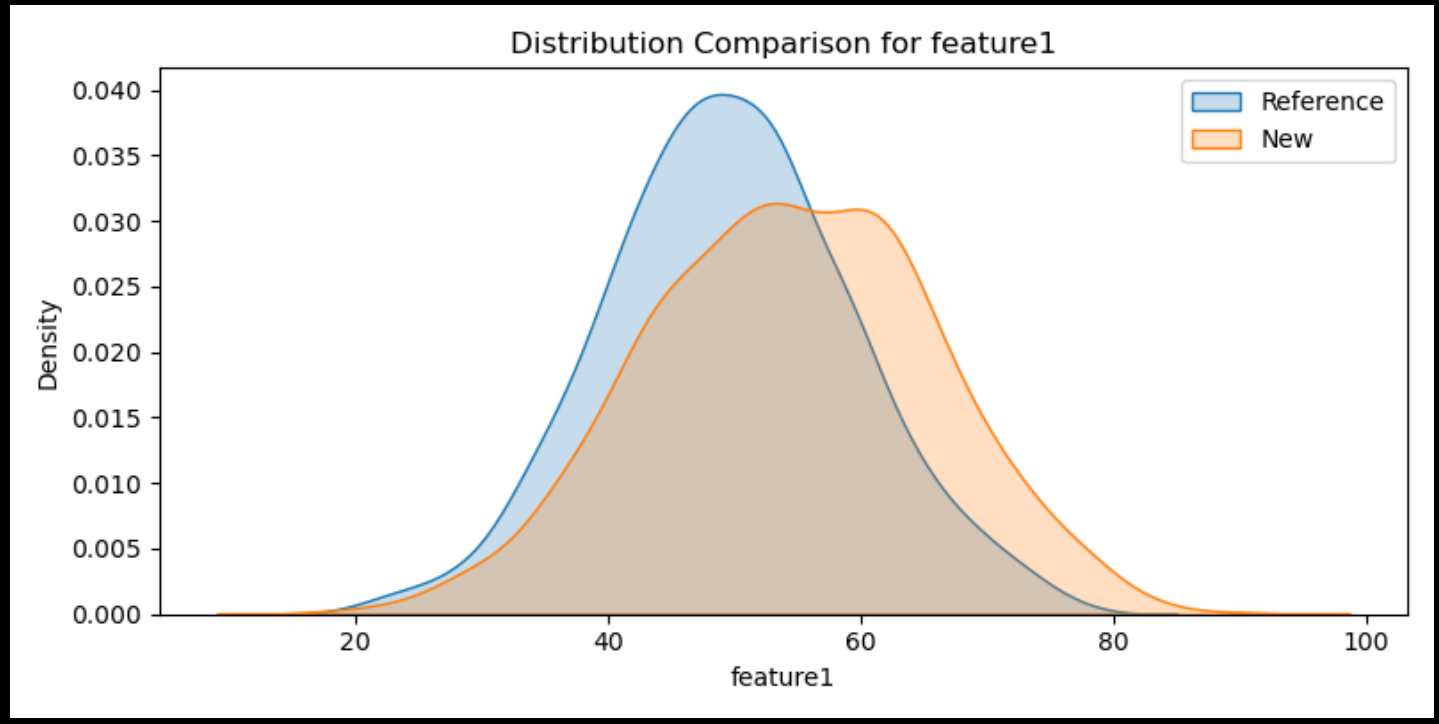


Microsoft

Evolution Technologies

Activity

Visual Comparison of Distributions



Microsoft

Evolution Technologies

Data Preprocessing

Data Preprocessing

- Data Preprocessing in data analysis is a crucial and often time-consuming stage that involves transforming raw data into a clean, well-organized, and suitable format for analysis. It's the set of steps performed to make data more effective and efficient for the subsequent analytical processes, including data mining, machine learning, statistical modeling, and visualization.
- Think of it as preparing your ingredients before you start cooking. If your ingredients are dirty, rotten, or not properly cut, your final dish won't be good, no matter how great your recipe is. Similarly, if your data is messy and inconsistent, the insights you derive from it will likely be flawed or misleading.

Goals of data preprocessing

- Improve Data Quality: By handling issues like missing values, noise, and inconsistencies.
- Make Data Suitable for Analysis: By transforming data into a format that algorithms and analytical tools can effectively process.
- Enhance the Accuracy and Efficiency of Analysis: Clean and well-prepared data leads to more reliable results and faster processing times.

Steps in Data Preprocessing:

Data Cleaning

Data Transformation

Data Reduction

Data Integration

Data Formatting

Microsoft

Evolution Technologies

Pandas

Pandas First Steps: install and import

- Pandas is an easy package to install. Open up your terminal program (shell or cmd) and install it using either of the following commands:

```
$ conda install pandas
        OR
$ pip  install  pandas
```

- For `jupyter notebook` users, you can run this cell

```
!pip install pandas
```

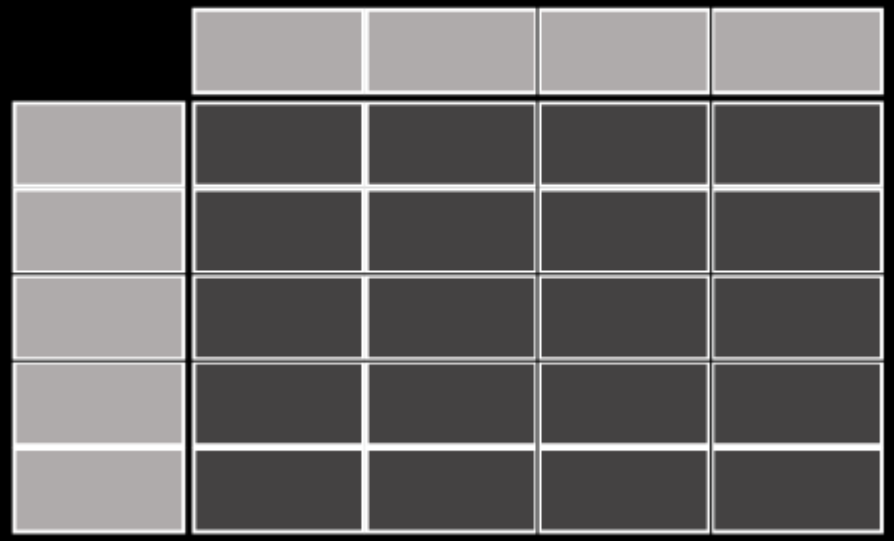
  - The `!` at the beginning runs cells as if they were in a terminal.

- To import pandas we usually import it with a shorter name since it's used so much:

```
import pandas as pd
```

Installation: https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html

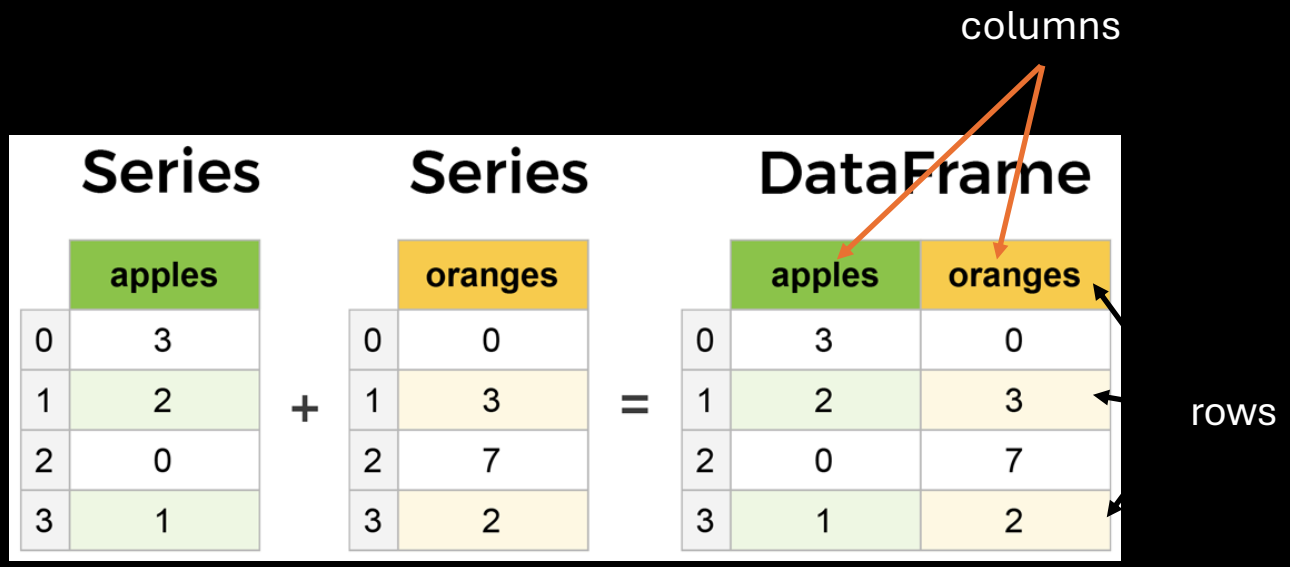
pandas: Data Table Representation



Core components of pandas: Series & DataFrames

- The primary two components of pandas are the Series and DataFrame.
  - Series is essentially a column, and
  - DataFrame is a multi-dimensional table made up of a collection of Series.
- DataFrames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.
  - A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

- Features of DataFrame
  - Potentially columns are of different types
  - Size – Mutable
  - Labeled axes (*rows* and *columns*)
  - Can Perform Arithmetic operations on rows and columns



Types of Data Structure in Pandas

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array with immutable size
Data Frames	2	General 2D labeled, size mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size mutable array.

- **Series & DataFrame**
  - Series is a one-dimensional array (1D Array) like structure with homogeneous data.
  - DataFrame is a two-dimensional array (2D Array) with heterogeneous data.
- **Panel**
  - Panel is a three-dimensional data structure (3D Array) with heterogeneous data.
  - It is hard to represent the panel in graphical representation.
  - But a panel can be illustrated as a container of DataFrame

pandas.DataFrame

```
pandas.DataFrame(data, index , columns , dtype , copy )
```

- data: data takes various forms like *ndarray*, *series*, *map*, *lists*, *dict*, constants and also another *DataFrame*.
- index: For the **row labels**, that are to be used for the resulting frame, Optional, Default is *np.arrange(n)* if no index is passed.
- columns: For **column labels**, the optional default syntax is - *np.arrange(n)*. This is only true if no index is passed.
- dtype: Data type of each column.
- copy: This command (or whatever it is) is used for copying of data, if the default is False.
- **Create DataFrame**
  - A pandas DataFrame can be created using various inputs like –
    - Lists
    - dict
    - Series
    - Numpy ndarrays
    - Another DataFrame

Creating a DataFrame from scratch

Creating a DataFrame from scratch

- There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dict. But first you must import pandas.

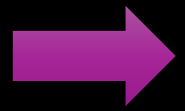
```
import pandas as pd
```

- Let's say we have a fruit stand that sells apples and oranges. We want to have a column for each fruit and a row for each customer purchase. To organize this as a dictionary for pandas we could do something like:

```
data = { 'apples':[3, 2, 0, 1] , 'oranges':[0, 3, 7, 2] }
```

- And then pass it to the pandas DataFrame constructor:

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



	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

How did that work?

- Each (key, value) item in data corresponds to a column in the resulting DataFrame.
- The Index of this DataFrame was given to us on creation as the numbers `0-3`, but we could also create our own when we initialize the DataFrame.
- E.g. if you want to have customer names as the index:

```
df = pd.DataFrame(data, index=['Ahmad', 'Ali', 'Rashed', 'Hamza'])
```

	apples	oranges
Ahmad	3	0
Ali	2	3
Rashed	0	7
Hamza	1	2

- So now we could locate a customer's order by using their names:

```
df.loc['Ali']
```

```
apples      2
oranges     3
Name: Ali, dtype: int64
```

pandas.DataFrame.from_dict

```
pandas.DataFrame.from_dict(data, orient='columns', dtype=None, columns=None)
```

- **data** : dict
  - Of the form **{field:array-like}** or **{field:dict}**.

- **orient** : **{'columns', 'index'}**, default '**columns**'
  - The "orientation" of the data.
  - If the keys of the passed dict should be the columns of the resulting DataFrame, pass 'columns' (default).
  - Otherwise if the keys should be rows, pass 'index'.

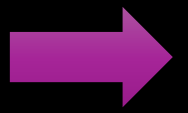
- **dtype** : **dtype**, default **None**
  - Data type to force, otherwise infer.

- **columns** : **list**, default **None**
  - Column labels to use when **orient='index'**. Raises a **ValueError** if used with **orient='columns'**.

https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.from_dict.html

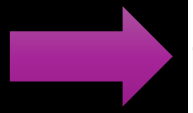
pandas' `orient` keyword

```
data = {'col_1':[3, 2, 1, 0], 'col_2':['a','b','c','d']}
pd.DataFrame.from_dict(data)
```



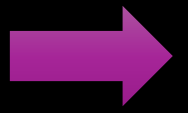
	col_1	col_2
0	3	a
1	2	b
2	1	c
3	0	d

```
data = {'row_1':[3, 2, 1, 0], 'row_2':['a','b','c','d']}
pd.DataFrame.from_dict(data,
                       orient='index')
```



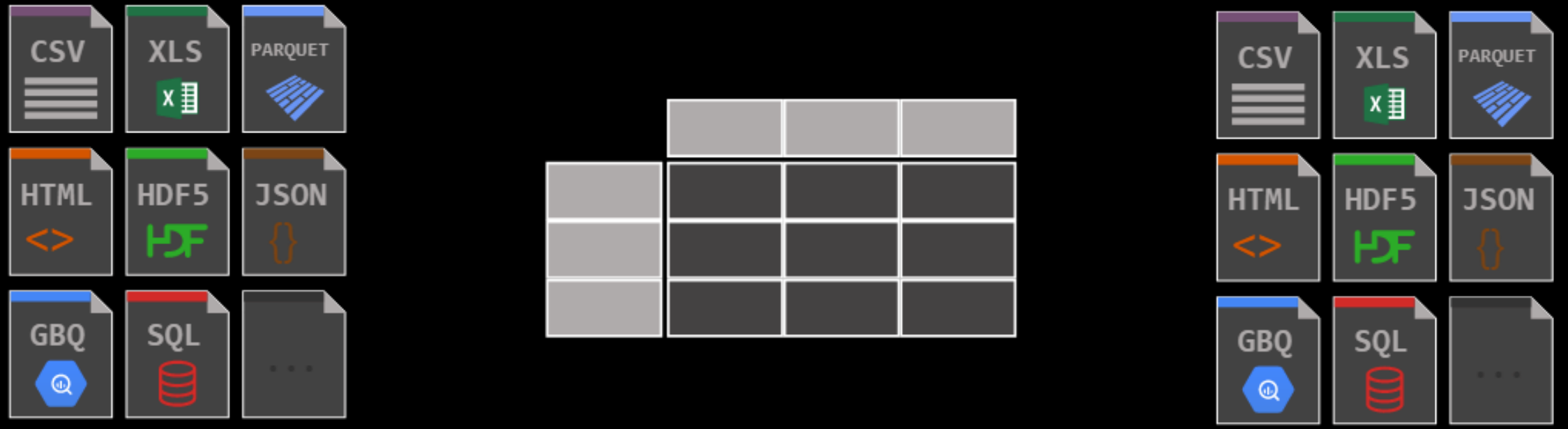
	0	1	2	3
row_1	3	2	1	0
row_2	a	b	c	d

```
data = {'row_1':[3, 2, 1, 0], 'row_2':['a','b','c','d']}
pd.DataFrame.from_dict(data,
                       orient  = 'index',
                       columns = ['A','B','C','D'])
```

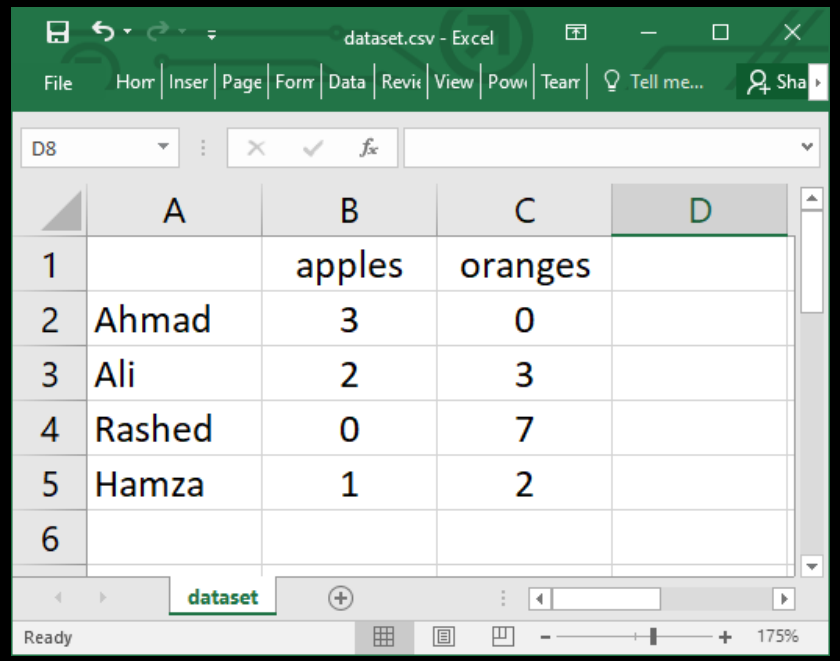


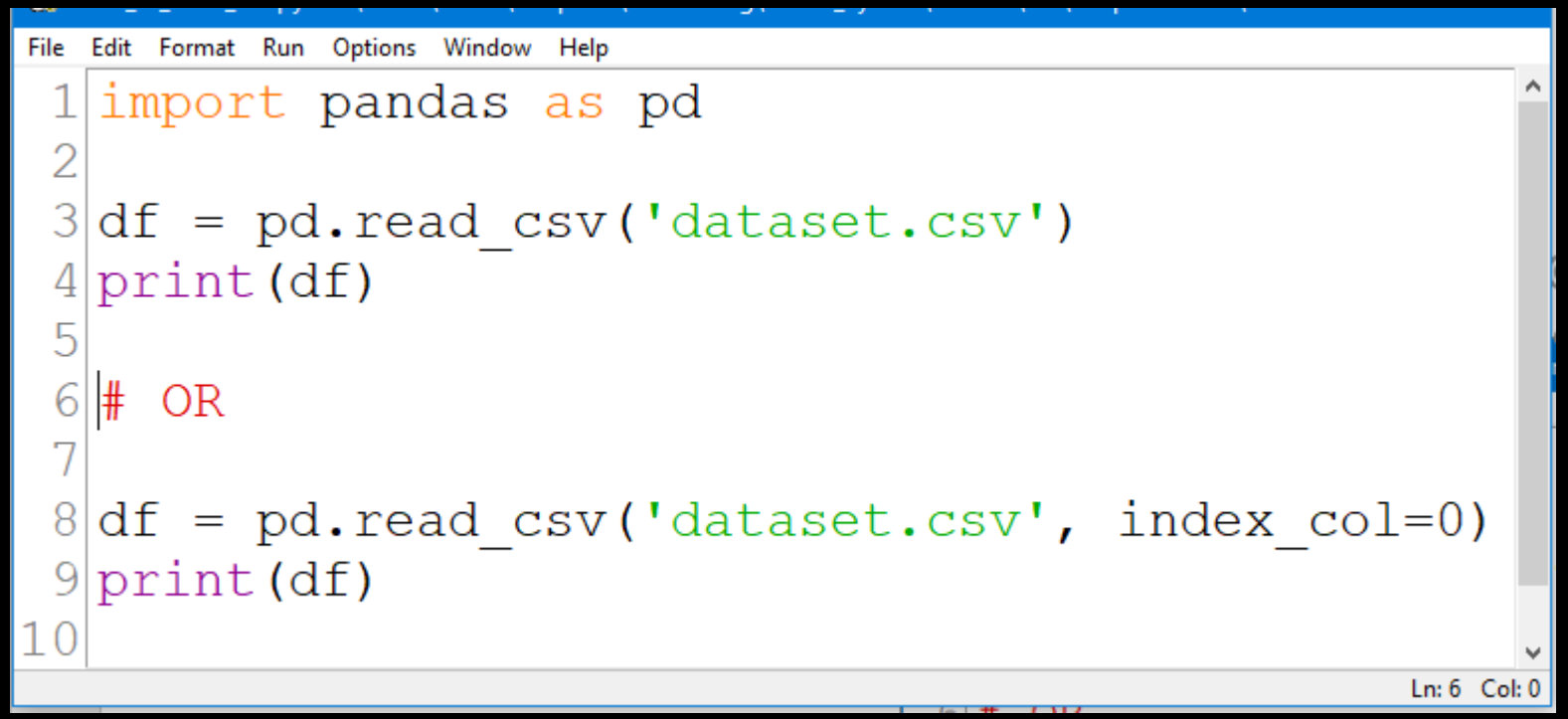
	A	B	C	D
row_1	3	2	1	0
row_2	a	b	c	d

Loading a DataFrame from files



Reading data from a CSV file





Reading data from CSVs

- With CSV files, all you need is a single line to load in the data:

```
df = pd.read_csv('dataset.csv')
```

	Unnamed: 0	apples	oranges
0	Ahmad	3	0
1	Ali	2	3
2	Rashed	0	7
3	Hamza	1	2

- CSVs don't have indexes like our DataFrames, so all we need to do is just designate the **`index_col`** when reading:

```
df = pd.read_csv('dataset.csv', index_col=0)
```

	apples	oranges
Ahmad	3	0
Ali	2	3
Rashed	0	7
Hamza	1	2

  - *Note: here we're setting the index to be column zero.*

Reading data from JSON

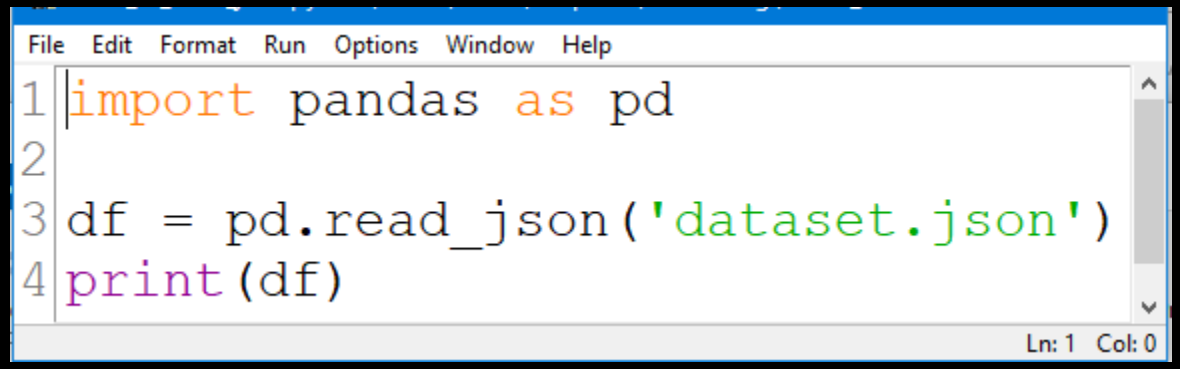
- If you have a JSON file — which is essentially a stored Python dict — pandas can read this just as easily:

```
df = pd.read_json('dataset.json')
```

- Notice this time our index came with us correctly since using JSON allowed indexes to work through nesting.
- Pandas will try to figure out how to create a DataFrame by analyzing structure of your JSON, and sometimes it doesn't get it right.
- Often you'll need to set the `orient` keyword argument depending on the structure

Example #1:Reading data from JSON

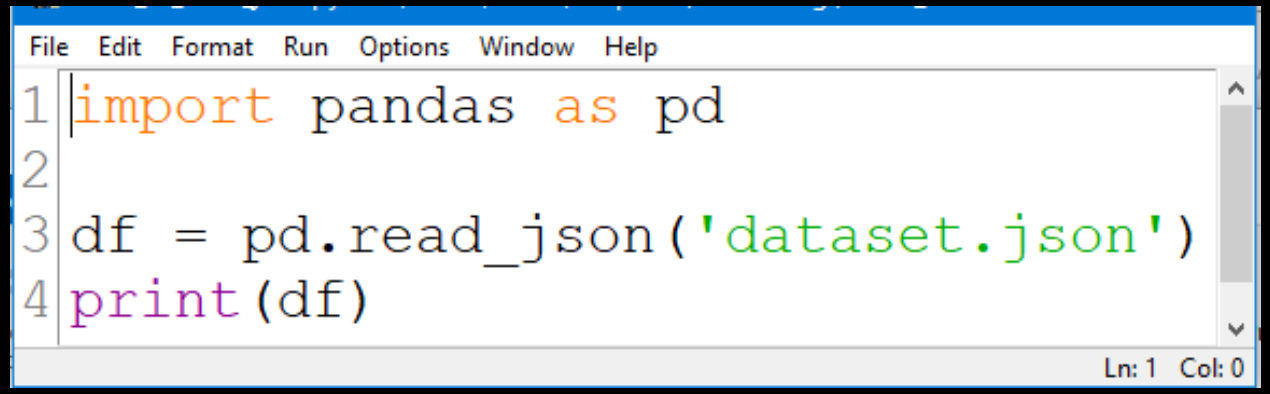
```
{
   "apples" :{"Ahmad":3,"Ali":2,"Rashed":0, "Hamza":1},
   "oranges":{"Ahmad":0,"Ali":3,"Rashed":7, "Hamza":2}
}
```



	apples	oranges
Ahmad	3	0
Ali	2	3
Rashed	0	7
Hamza	1	2

Example #2: Reading data from JSON

```json
{
   "Ahmad"   : {"apples":3,"oranges":0},
   "Ali"     : {"apples":2,"oranges":3},
   "Rashed"  : {"apples":0,"oranges":7},
   "Hamza"   : {"apples":1,"oranges":2}
}
```



	Ahmad	Ali	Rashed	Hamza
apples	3	2	0	1
oranges	0	3	7	2

Example #3: Reading data from JSON

```
{
   "Ahmad"   : {"apples":3,"oranges":0},
   "Ali"     : {"apples":2,"oranges":3},
   "Rashed"  : {"apples":0,"oranges":7},
   "Hamza"   : {"apples":1,"oranges":2}
}
```

```
File  Edit  Format  Run  Options  Window  Help
1 import pandas as pd
2
3 df = pd.read_json('dataset.json',
4                   orient='column')
5 print(df)
                                   Ln: 6  Col: 0
```

	Ahmad	Ali	Rashed	Hamza
apples	3	2	0	1
oranges	0	3	7	2

```
File  Edit  Format  Run  Options  Window  Help
1 import pandas as pd
2
3 df = pd.read_json('dataset.json',
4                   orient='index')
5 print(df)
                                   Ln: 6  Col: 0
```

	apples	oranges
Ahmad	3	0
Ali	2	3
Rashed	0	7
Hamza	1	2

Converting back to a CSV or JSON

- So after extensive work on cleaning your data, you're now ready to save it as a file of your choice. Similar to the ways we read in data, pandas provides intuitive commands to save it:

```
df.to_csv('new_dataset.csv')
df.to_json('new_dataset.json')
```

- When we save JSON and CSV files, all we have to input into those functions is our desired filename with the appropriate file extension.

Reference: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_json.html

Most important DataFrame operations

- DataFrames possess hundreds of methods and other operations that are crucial to any analysis.
- As a beginner, you should know the operations that:
  - that perform simple transformations of your data and those
  - that provide fundamental statistical analysis on your data.

Loading dataset

- We're loading this dataset from a CSV and designating the movie titles to be our index.

```
movies_df = pd.read_csv("movies.csv", index_col="title")
```

https://grouplens.org/datasets/movielens/

Viewing your data

- The first thing to do when opening a new dataset is print out a few rows to keep as a visual reference. We accomplish this with `.head()`:

```
movies_df.head()
```

- .head() outputs the first five rows of your DataFrame by default, but we could also pass a number as well: `movies_df.head(10)` would output the top ten rows, for example.

- To see the last five rows use `.tail()` that also accepts a number, and in this case we printing the bottom two rows.:

```
movies_df.tail(2)
```

Getting info about your data

- `.info()` should be one of the very first commands you run after loading your data
- `.info()` provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.

```
movies_df.info()
```

OUT:

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
Rank                    1000 non-null int64
Genre                   1000 non-null object
Description             1000 non-null object
Director                1000 non-null object
Actors                  1000 non-null object
Year                    1000 non-null int64
Runtime (Minutes)       1000 non-null int64
Rating                  1000 non-null float64
Votes                   1000 non-null int64
Revenue (Millions)      872 non-null float64
Metascore               936 non-null float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

```
movies_df.shape
```

OUT:

```
(1000, 11)
```

Handling duplicates

- This dataset does not have duplicate rows, but it is always important to verify you aren't aggregating duplicate rows.
- To demonstrate, let's simply just double up our movies DataFrame by appending it to itself:
- Using `append()` will return a copy without affecting the original DataFrame. We are capturing this copy in **temp** so we aren't working with the real data.
- Notice call `.shape` quickly proves our DataFrame rows have doubled.

```
temp_df = movies_df.append(movies_df)
temp_df.shape
```

```
OUT:

(2000, 11)
```

Now we can try dropping duplicates:

```
temp_df = temp_df.drop_duplicates()
temp_df.shape
```

```
OUT:

(1000, 11)
```

Handling duplicates

- Just like **`append()`**, the **`drop_duplicates()`** method will also return a copy of your DataFrame, but this time with duplicates removed. Calling **`.shape`** confirms we're back to the 1000 rows of our original dataset.
- It's a little verbose to keep assigning DataFrames to the same variable like in this example. For this reason, pandas has the inplace keyword argument on many of its methods. Using **`inplace=True`** will modify the DataFrame object in place:

```
temp_df.drop_duplicates(inplace=True)
```

- Another important argument for **`drop_duplicates()`** is keep, which has three possible options:
  - **first**: (default) Drop duplicates except for the first occurrence.
  - **last**: Drop duplicates except for the last occurrence.
  - **False**: Drop all duplicates.

https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/

Understanding your variables

- Using `.describe()` on an entire DataFrame we can get a summary of the distribution of continuous variables:

```
movies_df.describe()
```

OUT:

	rank	year	runtime	rating	
count	1000.000000	1000.000000	1000.000000	1000.000000	1.0
mean	500.500000	2012.783000	113.172000	6.723200	1.6
std	288.819436	3.205962	18.810908	0.945429	1.8
min	1.000000	2006.000000	66.000000	1.900000	6.1
25%	250.750000	2010.000000	100.000000	6.200000	3.6
50%	500.500000	2014.000000	111.000000	6.800000	1.1
75%	750.250000	2016.000000	123.000000	7.400000	2.3
max	1000.000000	2016.000000	191.000000	9.000000	1.7

- `.describe()` can also be used on a categorical variable to get the count of rows, unique count of categories, top category, and freq of top category:

```
movies_df['genre'].describe()
```

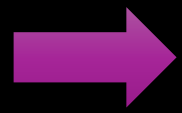
OUT:

```
count                          1000
unique                          207
top       Action,Adventure,Sci-Fi
freq                             50
Name: genre, dtype: object
```

- This tells us that the genre column has 207 unique values, the top value is Action/Adventure/Sci-Fi, which shows up 50 times (freq).

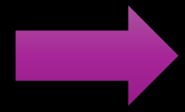
More Examples

```
import pandas as pd
data = [1,2,3,10,20,30]
df = pd.DataFrame(data)
print(df)
```



	0
0	1
1	2
2	3
3	10
4	20
5	30

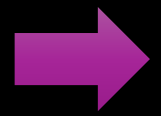
```
import pandas as pd
data = {'Name' : ['AA', 'BB'], 'Age': [30,45]}
df = pd.DataFrame(data)
print(df)
```



	Name	Age
0	AA	30
1	BB	45

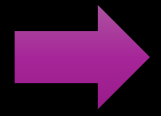
More Examples

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print(df)
```



```
   a   b     c
0  1   2   NaN
1  5  10  20.0
```

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print(df)
```



```
        a   b     c
first   1   2   NaN
second  5  10  20.0
```

More Examples

E.g. This shows how to create a DataFrame with a list of dictionaries, row indices, and column indices.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

#With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data,index=['first','second'],columns=['a','b'])

#With two column indices with one index with other name
df2 = pd.DataFrame(data,index=['first','second'],columns=['a','b1'])

print(df1)
print('............')
print(df2)
```

```
          a   b
first     1   2
second    5  10
............
          a  b1
first     1 NaN
second    5 NaN
```

More Examples:
Create a DataFrame from Dict of Series

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3]   , index=['a', 'b', 'c']),
     'two' : pd.Series([1,2, 3, 4], index=['a', 'b', 'c', 'd'])
    }
df = pd.DataFrame(d)
print(df)
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

More Examples: Column Addition

```
import pandas as pd
d = {'one':pd.Series([1,2,3],   index=['a','b','c']),
     'two':pd.Series([1,2,3,4], index=['a','b','c','d'])
    }
df = pd.DataFrame(d)
# Adding a new column to an existing DataFrame object
# with column label by passing new series

print("Adding a new column by passing as Series:")
df['three'] = pd.Series([10,20,30],index=['a','b','c'])
print(df)

print("Adding a column using an existing columns in
DataFrame:")
df['four'] = df['one']+df['three']
print(df)
```

```
Adding a column using Series:
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN

Adding a column using columns:
   one  two  three  four
a  1.0    1   10.0  11.0
b  2.0    2   20.0  22.0
c  3.0    3   30.0  33.0
d  NaN    4    NaN   NaN
```

More Examples: Column Deletion

```
# Using the previous DataFrame, we will delete a column
# using del function
import pandas as pd
d = {'one'   : pd.Series([1, 2, 3],    index=['a', 'b', 'c']),
     'two'   : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
     'three' : pd.Series([10,20,30],   index=['a','b','c'])
    }
df = pd.DataFrame(d)
print ("Our dataframe is:")
print(df)

# using del function
print("Deleting the first column using DEL function:")
del df['one']
print(df)

# using pop function
print("Deleting another column using POP function:")
df.pop('two')
print(df)
```

```
Our dataframe is:
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN

Deleting the first column:
    two  three
a     1   10.0
b     2   20.0
c     3   30.0
d     4    NaN

Deleting another column:
a   10.0
b   20.0
c   30.0
d    NaN
```

More Examples: Slicing in DataFrames

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3],    index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c','d'])
    }
df = pd.DataFrame(d)
print(df[2:4])
```

```
   one    two
c  3.0    3
d  NaN    4
```

More Examples: Addition of rows

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3],    index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c','d'])
    }
df = pd.DataFrame(d)
print(df)

df2 = pd.DataFrame([[5,6], [7,8]], columns = ['a', 'b'])
df = df.append(df2 )
print(df)
```

```
   one   two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

```
   one  two    a    b
a  1.0  1.0  NaN  NaN
b  2.0  2.0  NaN  NaN
c  3.0  3.0  NaN  NaN
d  NaN  4.0  NaN  NaN
0  NaN  NaN  5.0  6.0
1  NaN  NaN  7.0  8.0
```

More Examples: Deletion of rows

```
import pandas as pd
d = {'one':pd.Series([1, 2, 3],    index=['a','b','c']),
     'two':pd.Series([1, 2, 3, 4], index=['a','b','c','d'])
    }
df = pd.DataFrame(d)
print(df)

df2 = pd.DataFrame([[5,6], [7,8]], columns = ['a', 'b'])
df = df.append(df2 )
print(df)

df = df.drop(0)
print(df)
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4

   one  two    a    b
a  1.0  1.0  NaN  NaN
b  2.0  2.0  NaN  NaN
c  3.0  3.0  NaN  NaN
d  NaN  4.0  NaN  NaN
0  NaN  NaN  5.0  6.0
1  NaN  NaN  7.0  8.0

   one  two    a    b
a  1.0  1.0  NaN  NaN
b  2.0  2.0  NaN  NaN
c  3.0  3.0  NaN  NaN
d  NaN  4.0  NaN  NaN
1  NaN  NaN  7.0  8.0
```

More Examples: Reindexing

```
import pandas as pd
# Creating the first dataframe
df1 = pd.DataFrame({"A":[1, 5, 3, 4, 2],
                    "B":[3, 2, 4, 3, 4],
                    "C":[2, 2, 7, 3, 4],
                    "D":[4, 3, 6, 12, 7]},
                    index =["A1", "A2", "A3", "A4", "A5"])

# Creating the second dataframe
df2 = pd.DataFrame({"A":[10, 11, 7, 8, 5],
                    "B":[21, 5, 32, 4, 6],
                    "C":[11, 21, 23, 7, 9],
                    "D":[1, 5, 3, 8, 6]},
                    index =["A1", "A3", "A4", "A7", "A8"])

# Print the first dataframe
print(df1)
print(df2)
# find matching indexes
df1.reindex_like(df2)
```

- Pandas **`dataframe.reindex_like()`** function return an object with matching indices to myself.
- Any non-matching indexes are filled with NaN values.

Out[72]:

	A	B	C	D
A1	1.0	3.0	2.0	4.0
A3	3.0	4.0	7.0	6.0
A4	4.0	3.0	3.0	12.0
A7	NaN	NaN	NaN	NaN
A8	NaN	NaN	NaN	NaN

More Examples:
Concatenating Objects (Data Frames)

```
import pandas as pd
df1 = pd.DataFrame({'Name':['A','B'], 'SSN':[10,20], 'marks':[90, 95] })
df2 = pd.DataFrame({'Name':['B','C'], 'SSN':[25,30], 'marks':[80, 97] })
df3 = pd.concat([df1, df2])
df3
```

References

- pandas documentation
  - https://pandas.pydata.org/pandas-docs/stable/index.html
- pandas: Input/output
  - https://pandas.pydata.org/pandas-docs/stable/reference/io.html
- pandas: DataFrame
  - https://pandas.pydata.org/pandas-docs/stable/reference/frame.html
- pandas: Series
  - https://pandas.pydata.org/pandas-docs/stable/reference/series.html
- pandas: Plotting
  - https://pandas.pydata.org/pandas-docs/stable/reference/plotting.html

Statistical tools for summarizing data

- In data science, summarizing data is a crucial first step in understanding and preparing data for analysis. Here are some of the most commonly used **statistical tools and techniques** for summarizing data:

📊 Descriptive Statistics

- These provide simple summaries about the sample and the measures.
- **Measures of Central Tendency**
  - **Mean**: Average value.
  - **Median**: Middle value when data is sorted.
  - **Mode**: Most frequently occurring value.
- **Measures of Dispersion**
  - **Range**: Difference between max and min.
  - **Variance**: Average of squared differences from the mean.
  - **Standard Deviation**: Square root of variance.
  - **Interquartile Range (IQR)**: Difference between the 75th and 25th percentiles.
- **Shape of Distribution**
  - **Skewness**: Measure of asymmetry.
  - **Kurtosis**: Measure of tail heaviness.

Data Visualization Tools

- These help in visually summarizing data:
- **Histograms**: Show frequency distribution.
- **Box Plots**: Show median, quartiles, and outliers.
- **Bar Charts**: Compare categorical data.
- **Pie Charts**: Show proportions.
- **Scatter Plots**: Show relationships between two variables.
- **Heatmaps**: Show correlation matrices.

🧮 Statistical Software and Libraries

- Python: pandas, numpy, matplotlib, seaborn, scipy
- R: summary(), dplyr, ggplot2
- Excel: Descriptive statistics toolpak
- SPSS / SAS / Stata: Widely used in academia and industry

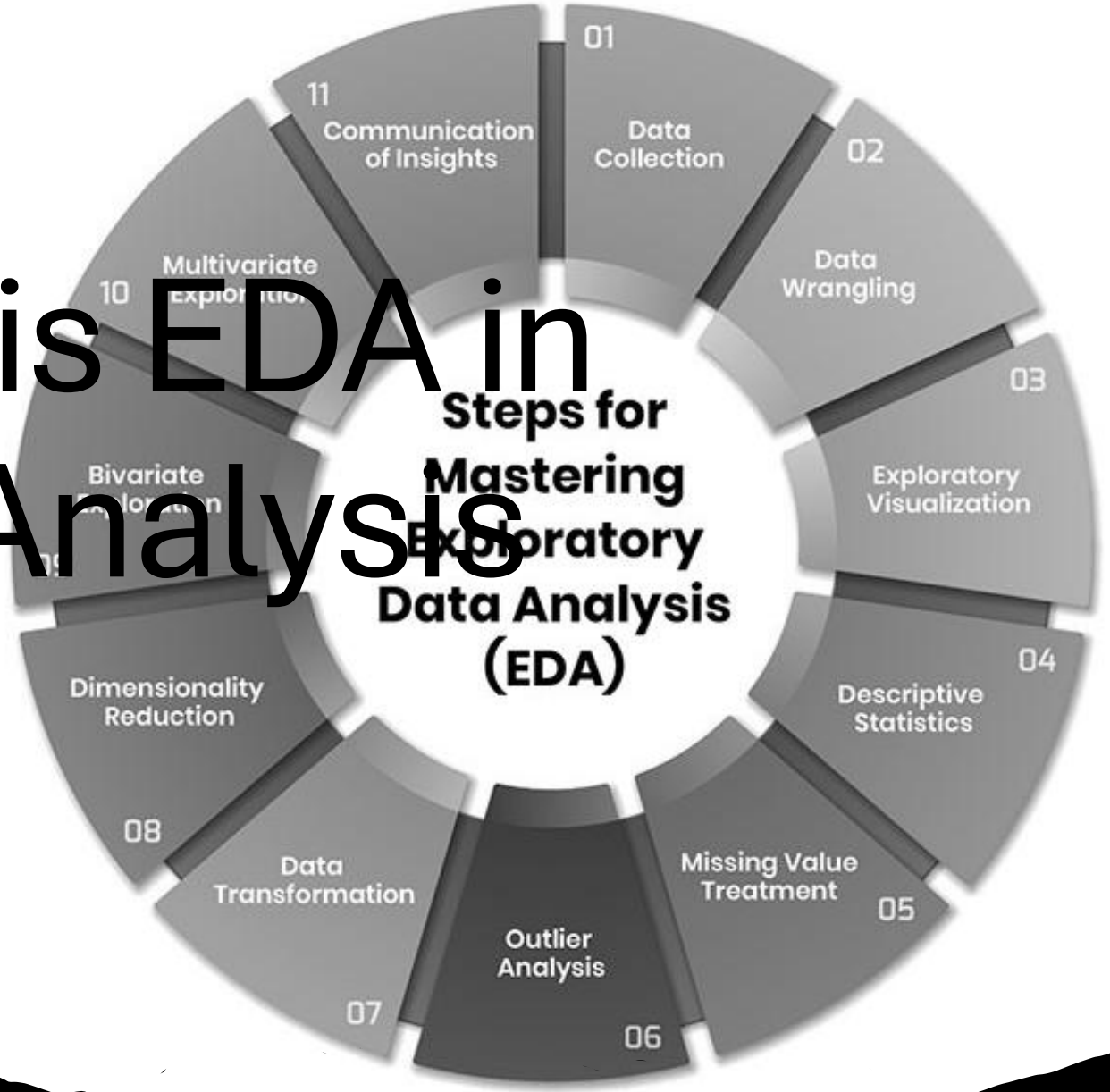
📊 Key Concepts of Distribution

- **Frequency**: How often each value occurs.
- **Shape**: The form of the data spread—common shapes include:
  - **Normal (bell-shaped)**: Most values cluster around the mean.
  - **Skewed**: Data leans to one side (left or right).
  - **Uniform**: All values occur with equal frequency.
  - **Bimodal**: Two peaks in the data.
- **Center**: Where the data tends to cluster (mean, median, mode).
- **Spread**: How much the data varies (range, variance, standard deviation).
- **Outliers**: Values that are far from the rest of the data.

📈 Example:

- Imagine you have test scores for a class. A distribution would show:
- How many students scored in each range (e.g., 60–70, 70–80, etc.).
- Whether most students scored around the average or if scores were widely spread.

What is EDA in Data Analysis



Definition of EDA

- EDA refers to exploratory data analysis, which involves summarizing data sets to foster greater understanding of the data's structure, patterns, and insights before formal modeling.

  - Understanding EDA

Importance of EDA

Guiding Discovery

EDA guides data scientists in uncovering hidden patterns and insights by visualizing data, which drives hypothesis generation.

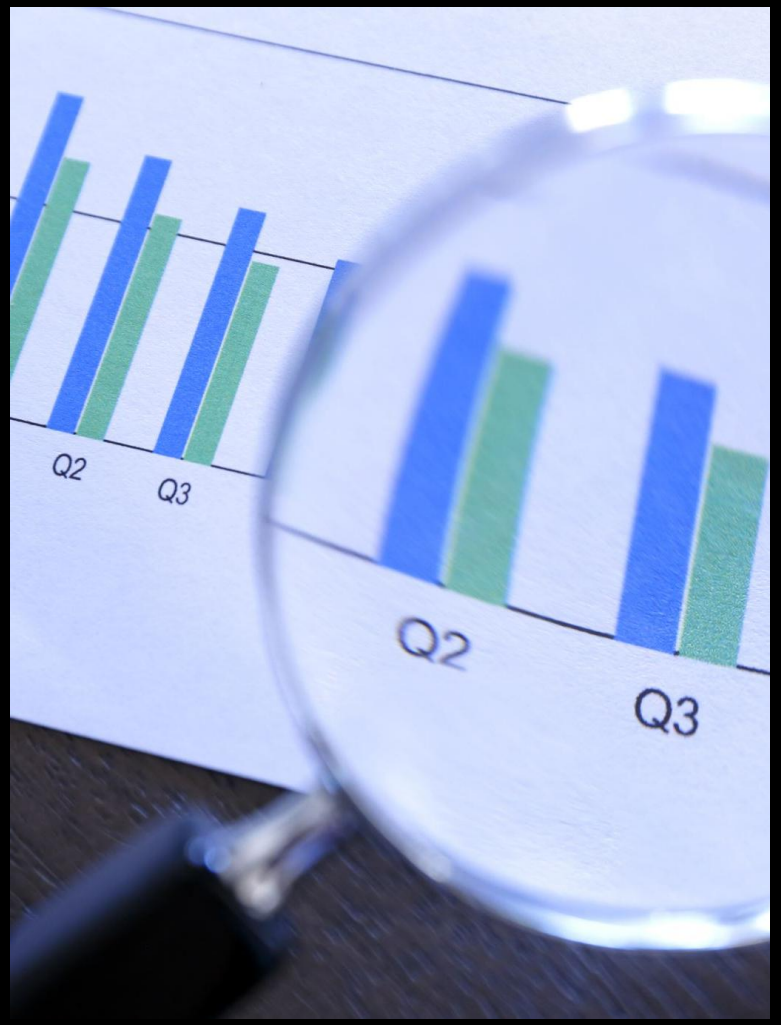
Informed Decisions

Understanding data through EDA helps stakeholders make informed decisions backed by evidence from analyzed trends.

Exploratory Data Analysis (EDA)

Steps in eda

1. Reading Dataset
2. Analyzing the Data
3. Data Reduction
4. Feature Engineering
5. Creating Features
6. Data Cleaning/Wrangling
7. EDA Exploratory Data Analysis
8. Statistics Summary
9. EDA Univariate Analysis
10. Data Transformation
11. EDA Bivariate Analysis



Step 1: 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.
- In this session
- the data to predict **Used car price** is being used as an example. In this dataset, we are trying to analyze the used car's price and how EDA focuses on identifying the factors influencing the car price. We have stored the data in the DataFrame **data.**

```
data = pd.read_csv("used_cars.csv")
```

Analyzing the Data

```
data.head()
```

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

Step 2: 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.

```
# 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   float64
 8   Engine             7207 non-null   float64
 9   Power              7078 non-null   float64
 10  Seats              7200 non-null   float64
 11  New_price          1006 non-null   float64
 12  Price              6019 non-null   float64
dtypes: float64(6), int64(2), object(5)
memory usage: 736.8+ KB
```

We start our Feature Engineering as we need to add some columns required for analysis.

Step 3: Feature Engineering

- **Feature engineering** refers to the process of using domain knowledge to select and transform the most relevant variables from raw data when creating a predictive model using machine learning or statistical modeling. The main goal of Feature engineering is to create meaningful data from raw data.

Step 4: Creating Features

- We will play around with the variables Year and Name in our dataset. If we see the sample data, the column “Year” shows the manufacturing year of the car.
- It would be difficult to find the car’s age if it is in year format as the Age of the car is a contributing factor to Car Price.
- Introducing a new column, “Car_Age” to know the age of the car

Step 4: Creating Features

```
from datetime import date
date.today().year
data['Car_Age']=date.today().year-data['Year']
data.head()
```

Out[32]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_price	Price	Car_Age
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.60	998.0	58.16	5.0	NaN	1.75	12
1	Hyundai Creta 1.6 CRDI SX Option	Pune	2015	41000	Diesel	Manual	First	19.67	1582.0	126.20	5.0	NaN	12.50	7
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.20	1199.0	88.70	5.0	8.61	4.50	11
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77	1248.0	88.76	7.0	NaN	6.00	10
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.20	1968.0	140.80	5.0	NaN	17.74	9

Since car names will not be great predictors of the price in our current data. But we can process this column to extract important information using brand and Model names. **Let's split the name and introduce new variables "Brand" and "Model"**

```
data['Brand'] = data.Name.str.split().str.get(0)
```

```
data['Model'] = data.Name.str.split().str.get(1) + data.Name.str.split().str.get(2)
```

```
data[['Name','Brand','Model']]
```

Out[41]:

	Name	Brand	Model
0	Maruti Wagon R LXI CNG	Maruti	WagonR
1	Hyundai Creta 1.6 CRDI SX Option	Hyundai	Creta1.6
2	Honda Jazz V	Honda	JazzV
3	Maruti Ertiga VDI	Maruti	ErtigaVDI
4	Audi A4 New 2.0 TDI Multitronic	Audi	A4New
...	...	...	...
7248	Volkswagen Vento Diesel Trendline	Volkswagen	VentoDiesel
7249	Volkswagen Polo GT TSI	Volkswagen	PoloGT
7250	Nissan Micra Diesel XV	Nissan	MicraDiesel
7251	Volkswagen Polo GT TSI	Volkswagen	PoloGT
7252	Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan...	Mercedes-Benz	E-Class2009-2013

7253 rows × 3 columns

```
print(data.Brand.unique())
print(data.Brand.nunique())
```

Data Cleaning/Wrangling

Some names of the variables are not relevant and not easy to understand. Some data may have data entry errors, and some variables may need data type conversion. We need to fix this issue in the data.

- In the example, **The brand name ‘Isuzu’ ‘ISUZU’ and ‘Mini’ and ‘Land’ looks incorrect. This needs to be corrected**

Data Cleaning/Wrangling

```
['Maruti' 'Hyundai' 'Honda' 'Audi' 'Nissan' 'Toyota' 'Volkswagen' 'Tata'
 'Land' 'Mitsubishi' 'Renault' 'Mercedes-Benz' 'BMW' 'Mahindra' 'Ford'
 'Porsche' 'Datsun' 'Jaguar' 'Volvo' 'Chevrolet' 'Skoda' 'Mini' 'Fiat'
 'Jeep' 'Smart' 'Ambassador' 'Isuzu' 'ISUZU' 'Force' 'Bentley'
 'Lamborghini' 'Hindustan' 'OpelCorsa']

33
```

Data Cleaning/Wrangling

```
searchfor = ['Isuzu' ,'ISUZU','Mini','Land']
data[data.Brand.str.contains('|'.join(searchfor))].head(5)
```

Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_price	Price	Car_Age	Brand	Model
Delhi	2014	72000	Diesel	Automatic	First	12.70	2179.0	187.70	5.0	NaN	27.00	8	Land	RoverRange
Pune	2012	85000	Diesel	Automatic	Second	0.00	2179.0	115.00	5.0	NaN	17.50	10	Land	RoverFreelander
Jaipur	2017	8525	Diesel	Automatic	Second	16.60	1998.0	112.00	5.0	NaN	23.00	5	Mini	CountrymanCooper
oimbatore	2018	36091	Diesel	Automatic	First	12.70	2179.0	187.70	5.0	NaN	55.76	4	Land	RoverRange
Kochi	2017	26327	Petrol	Automatic	First	16.82	1998.0	189.08	4.0	44.28	35.67	5	Mini	CooperConvertible

```
data["Brand"].replace({"ISUZU": "Isuzu", "Mini": "Mini Cooper","Land":"Land Rover"}, inplace=True)
```

EDA Exploratory Data Analysis

Exploratory Data Analysis refers to the crucial process of performing initial investigations on data to discover patterns to check assumptions with the help of summary statistics and graphical representations.

- EDA can be leveraged to check for outliers, patterns, and trends in the given data.
- EDA helps to find meaningful patterns in data.
- EDA provides in-depth insights into the data sets to solve our business problems.
- EDA gives a clue to impute missing values in the dataset

Introduction to Exploratory Data Analysis (EDA) (EDA)

Definition and Purpose

EDA refers to the analysis of data sets to summarize their key characteristics, often with visual methods.

Importance in Data Science

It ensures that the data used is suitable suitable for analysis, identifying trends trends and relationships.

Key Concepts

Includes central tendencies, variability, variability, and distribution shapes like like normal and skewed.

Exploring EDA Techniques

1 **Descriptive Statistics**

Provide summary through mean, median, mode, and range.

2 **Data Visualization**

Techniques like histograms, box plots, and scatter plots plots make data interpretation intuitive.

3 **Univariate Analysis**

Analyzes a single variable to understand its distribution and distribution and characteristics.

4 **Multivariate Analysis**

Investigates relationships among multiple variables to to identify correlations.

5 **Outlier Detection**

Identifies anomalies in data using techniques like IQR or Z-score.

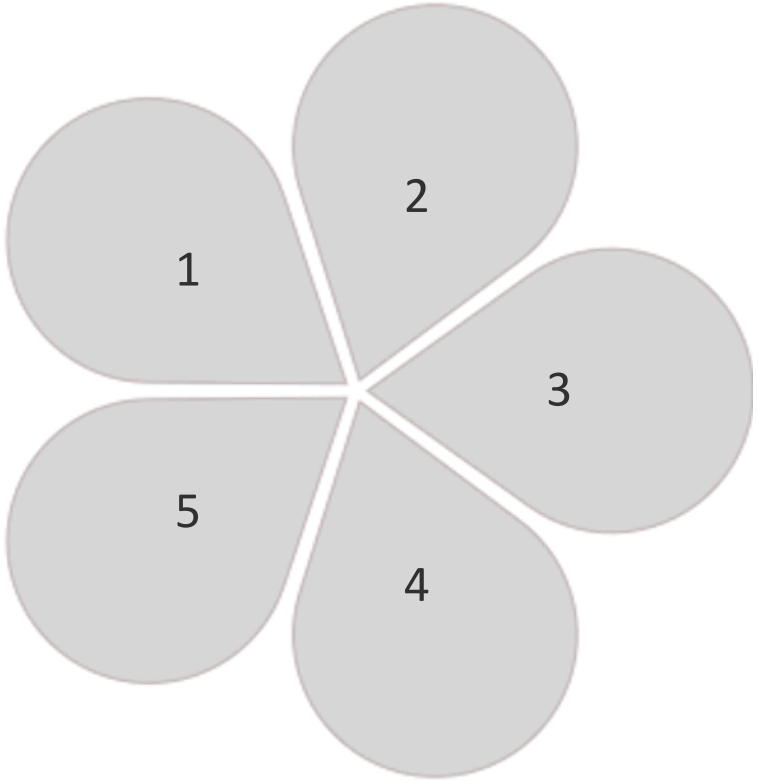
Tools for Exploratory Data Analysis

Python Libraries

Essential for data manipulation and exploratory analysis using Pandas and NumPy.

Tableau and Power BI

User-friendly tools for visualization and EDA without extensive coding.



Data Visualization Libraries

Matplotlib and Seaborn offer flexible flexible plotting options and enhanced enhanced aesthetics.

R Programming

Utilizes packages like ggplot2 for comprehensive EDA workflows.

Jupyter Notebooks

Ideal for interactive EDA combining code, text, and visualizations.

Case Study: EDA in Practice

Dataset Overview

Retail dataset featuring sales figures, figures, transaction times, product details, and customer demographics. demographics.

Initial Review

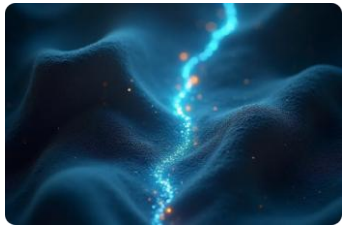
Importing, cleaning, and generating generating descriptive statistics to identify trends and patterns.

Data Visualization

Using bar plots and scatter plots to to uncover correlations and trends in in sales.

Key Insights

Revealing customer segments, seasonal seasonal buying patterns, and insights



Model Development Precursor

Utilizing EDA knowledge to enhance

Key Takeaways:

PATTERN RECOGNITION: UNDERSTAND HOW DATA IS CENTERED AND SPREAD.

OUTLIER DETECTION: SPOT UNUSUAL VALUES THAT MAY SKEW RESULTS.

STATISTICAL TESTING: CHOOSE THE RIGHT TESTS BASED ON DISTRIBUTION.

MODEL PERFORMANCE: SOME MODELS ASSUME NORMALITY FOR BEST RESULTS.

DECISION-MAKING: TAILOR STRATEGIES BASED ON DATA TRENDS.

Conclusion

Exploratory Data Analysis (EDA) plays a crucial role in data science by enhancing the validity of analyses and models. Its integration with advanced technologies promises a more efficient data exploration process.

Summary of EDA Significance

EDA is paramount in data analysis analysis and machine learning, providing essential insights.

Future Trends

The future of EDA lies in automation automation and integration with with machine learning techniques. techniques.

Data Visualization Techniques

- This presentation explores data visualization using Matplotlib and Seaborn, two powerful Python libraries. It covers their functionalities, key features, and practical applications for creating captivating visual representations of data.

Introduction to Matplotlib

Overview of Matplotlib

Matplotlib is a versatile library for creating static, animated, and interactive visualizations in Python.

Basic Components

Matplotlib primarily includes figures, axes, and plots essential for effective visualizations.

Installation and Setup

Installing Matplotlib is straightforward using pip and can be easily imported into Python scripts.

Visualization with Matplotlib

1 Line and Scatter Plots

These fundamental types depict trends and relationships between variables, essential for exploratory data analysis.

2 Bar and Histogram Charts Charts

Bar charts are perfect for categorical data, while histograms visualize frequency distributions, aiding in data comparisons.

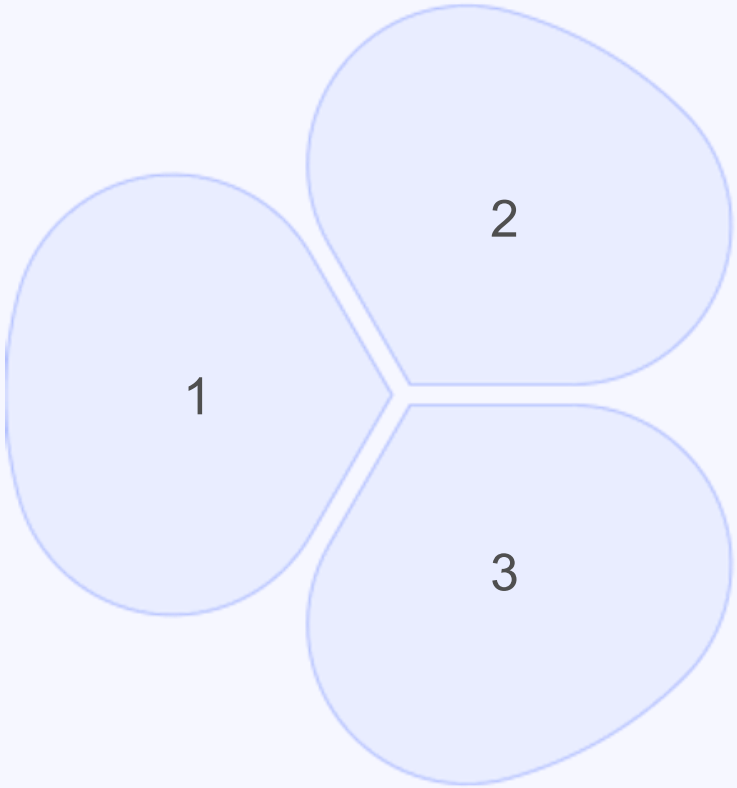
3 Customization Options

Matplotlib's extensive customization includes colors, labels, and legends, enhancing visual clarity and aesthetics.

Introduction to Seaborn

Overview of Seaborn

Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive statistical graphics.



Key Features

Seaborn includes built-in themes and support for complex visualizations such as violin plots and heat maps.

Installation and Usage

Seaborn can be installed using pip and integrates easily into existing Matplotlib visualizations.

Visualization with Seaborn

Statistical Plots

Seaborn's functions handle statistical computations, creating various plots for insights.

Heatmaps and Categorical Categorical Plots

Utilize heatmaps for correlation and categorical plots for data comparisons.

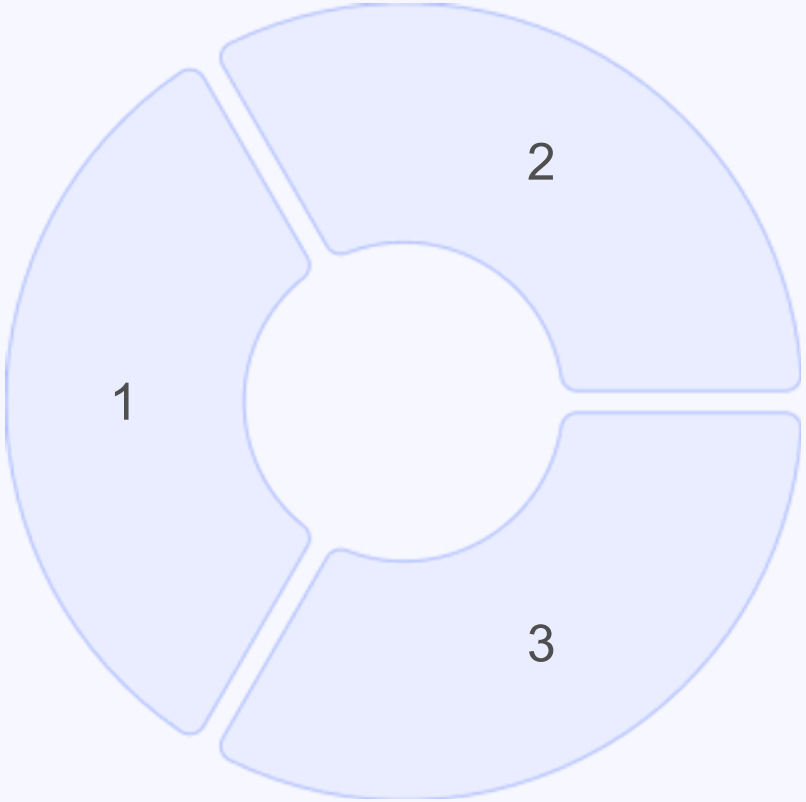
Beautiful Defaults

Seaborn provides aesthetically pleasing charts with minimal effort required.

Combining Matplotlib and Seaborn

Creating Composite Visualizations

Use Seaborn to enhance Matplotlib plots with better aesthetics.



Advanced Customization

Combine Matplotlib's features with Seaborn's convenience for complex visualizations.

Real-world Applications

Apply combined techniques in sectors like finance, healthcare, and marketing.

Practical Applications of Data Visualization

Business Intelligence

Data visualization plays a crucial role in business intelligence, where stakeholders need to quickly grasp insights from complex data sets to inform strategies and decisions.

Scientific Research

Visualizations are pivotal in scientific research for clearly conveying results and trends, facilitating communication among researchers and the broader audience.

Education and Training

Effective data visualization aids in teaching complex concepts, making learning more engaging and accessible for students in various disciplines.

Conclusion on Data Visualization Tools

Data Visualization Importance

Utilizing Matplotlib and Seaborn empowers effective data exploration and presentation.

Enhanced Analytical Skills

Mastering these tools significantly boosts analytical capabilities.

Clearer Insight Communication Communication

Facilitates clearer communication of insights across various fields.

Statistical Tools for Data Analysis

This presentation examines key statistical tools necessary for data summarization and analysis, helping to make informed decisions based on data insights.

Descriptive Statistics Overview

Mean, Median, Mode

These measures summarize data by indicating the average, middle, and most frequent values.

Range and Variance

Range indicates the spread of the data, data, while variance quantifies the degree of variation.

Standard Deviation

This statistic expresses how much data data deviates from the mean, offering offering insights into data consistency. consistency.

Data Visualization Techniques

1. **Bar and Pie Charts**

   Ideal for representing categorical data visually.

2. **Histograms**

   Graphical representation of frequency distribution of numerical numerical data.

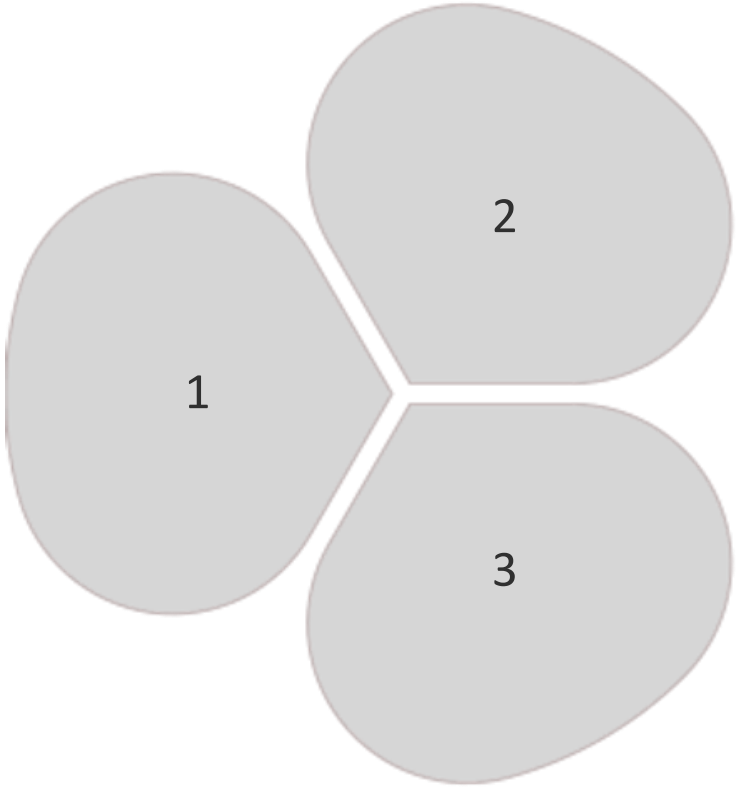
3. **Box Plots**

   Showcases distribution based on on summary statistics, highlighting highlighting median and outliers. outliers.

Inferential Statistics Techniques

Hypothesis Testing

Uses sample data to make conclusions about a population.



Confidence Intervals

Provide a range of values to estimate estimate population parameters.

Regression Analysis

Analyzes relationships between variables for predictions.

Correlation Analysis

Pearson Correlation Coefficient

Assesses linear relationship strength strength and direction.

Spearman's Rank Correlation

Measures association between ranked variables.

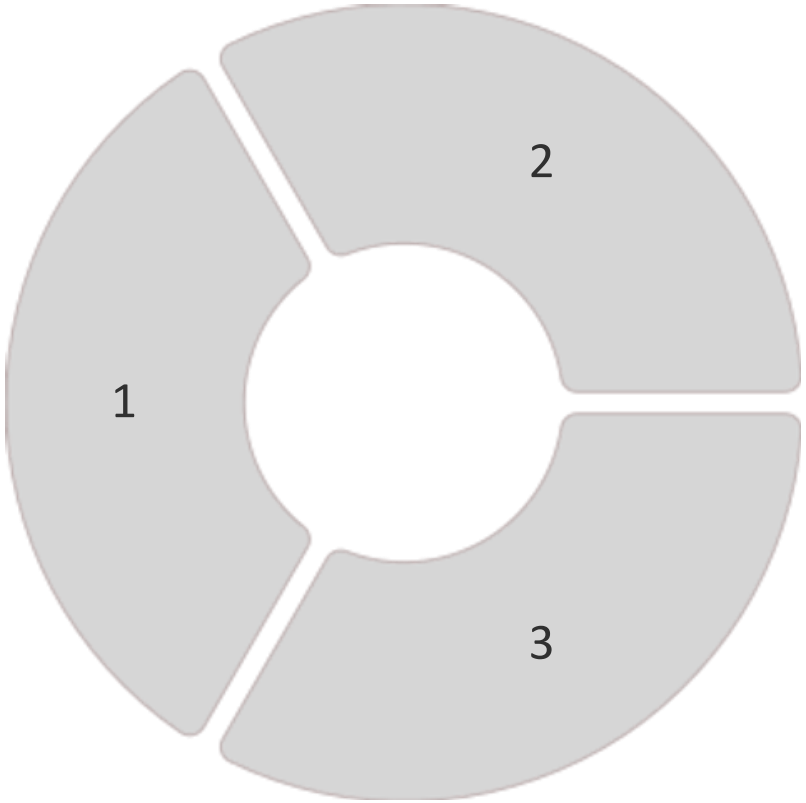
Scatter Plots

Visualizes relationships and identifies correlation trends.

Data Cleaning Techniques

Handling Missing Data

Addressing missing values through through methods like deletion or or imputation ensures data integrity, integrity, allowing for more accurate accurate analysis and conclusions. conclusions.



Outlier Detection

Identifying and managing outliers is crucial, as they can skew results and mislead interpretations; techniques include using z-scores or IQR methods.

Data Transformation

Normalizing or standardizing data prepares it for analysis, enhancing models' performance and ensuring that that results are valid and interpretable. interpretable.

Software Tools Overview

Excel

A versatile tool used for basic statistical analysis and visualization.

R and Python

Powerful programming languages for statistical analysis and advanced modeling.

Statistical Software

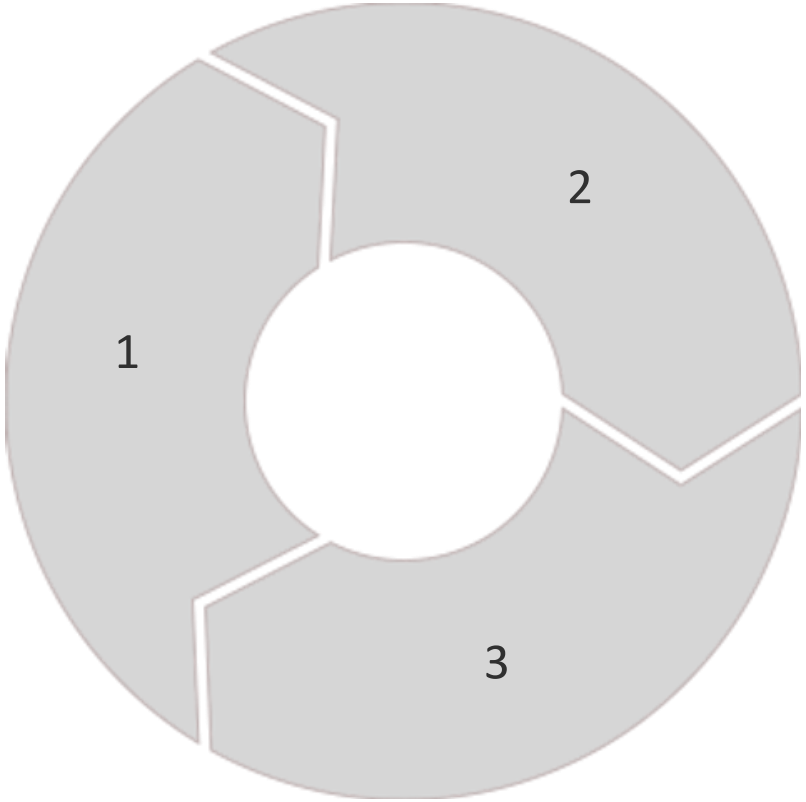
Tools like SPSS and SAS facilitate complex analysis through user-friendly friendly interfaces.

Conclusion and Recommendations

Mastering statistical tools is key to extracting meaningful insights and supporting data-driven decisions. Continuous education in the latest techniques is vital, as statistical tools find broad applications across business, healthcare, and many other fields.

Importance of Statistical Tools

Mastering statistical tools is essential for effective data analysis. analysis.



Continuous Learning

Ongoing education in statistical tools is crucial for professionals.

Application in Various Fields

Statistical tools are widely applicable in applicable in multiple domains.

Data Visualization with Matplotlib & & Seaborn

Data visualization is essential for interpreting complex datasets. Matplotlib and Seaborn are powerful Python libraries that make it easier to create informative and visually appealing graphics to enhance data analysis.

Introduction to Matplotlib

What is Matplotlib?

Matplotlib is a widely-used Python library for creating static, static, animated, and interactive visualizations.

Basic Plotting

Users can create basic plots like line charts, bar graphs, and and scatter plots.

Customization Options

Matplotlib offers extensive customization options for fonts, fonts, colors, and styles.

Integration with Other Libraries

It integrates seamlessly with NumPy and Pandas for easy data visualization.

Introduction to Seaborn

1 What is Seaborn?

Seaborn enhances Matplotlib with attractive statistical graphics.

2 Data Exploration

Easily explore variable relationships through various visualizations.

3 Built-in Themes

Offers themes and palettes for visually appealing graphics.

4 Support for Pandas

Ideal for creating visualizations with Pandas DataFrames.

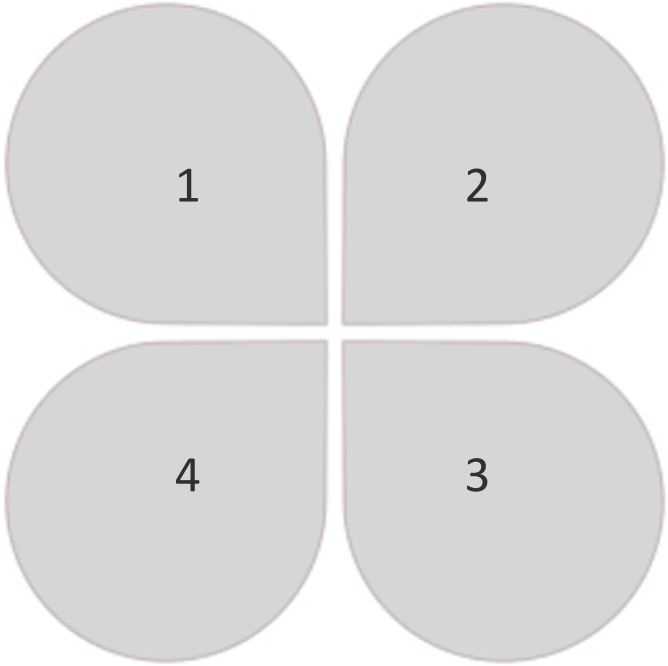
Comparison of Matplotlib and Seaborn

Ease of Use

Seaborn simplifies the syntax for creating creating complex visualizations compared compared to Matplotlib, making it more more accessible for beginners. This can can lead to quicker iteration during data data analysis.

Interactivity

Although both libraries can generate interactive plots, Matplotlib has more options to create interactive visualizations while Seaborn focuses on creating aesthetically pleasing static plots.



Plot Types

While Matplotlib offers a wide range of range of plot types, Seaborn specializes in specializes in statistical plots, such as as regression plots and categorical plots, plots, making it ideal for statistical data. data.

Aesthetics

Seaborn provides better default aesthetics and style options, which reduces the need for extensive customization in Matplotlib. This enhances the visual quality of outputs without much effort.

Practical Examples with Matplotlib

Creating Line Graphs

Easily plot time series data with axis labeling and titles for titles for clarity.

Bar Charts and Histograms

Visualize categorical and frequency distributions to understand data behavior.

Scatter Plots

Visualize relationships between two numerical variables to variables to identify correlations.

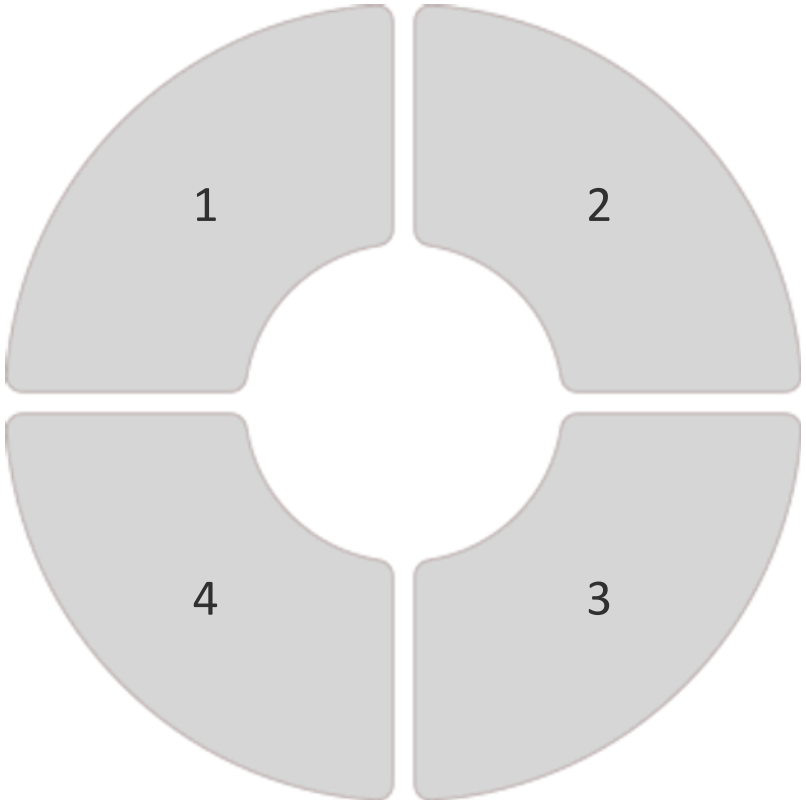
Subplots and Multi-Graphs

Create subplots for comparing multiple datasets in a single figure.

Practical Examples with Seaborn

Heatmaps

Visualize correlation matrices for quick quick relationships identification.



Pair Plots

Display distributions and relationships between variable pairs in a grid format.

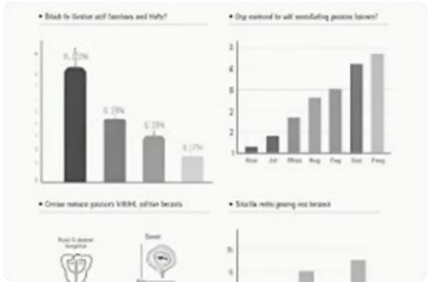
Facet Grids

Visualize data across multiple subplots based on categorical variables.

Box and Violin Plots

Illustrate dataset distributions, highlighting median, quartiles, and outliers.

Conclusion of Data Visualization Insights



Importance of Data Visualization

Crucial for extracting insights and effectively communicating findings.

Choosing the Right Tool

Understanding strengths of Matplotlib and Seaborn for better library selection.

Continuous Learning

Experimenting and learning enhances proficiency for better visualizations.

Resources for Further Learning

Online tutorials, documentation, and community forums to expand knowledge.

Creating Plots with Matplotlib

This presentation explores how to create static, animated, and interactive plots using Matplotlib, a powerful plotting library in Python, library in Python, highlighting its features and practical applications.

Introduction to Matplotlib

Overview of Matplotlib

Matplotlib is a widely used Python library for creating static, animated, and interactive visualizations in a variety of formats.

Installation and Setup

Installing Matplotlib is straightforward straightforward via pip, and it integrates integrates seamlessly with Jupyter Notebooks for interactive use.

Basic Plot Types

Matplotlib supports a range of basic basic plot types including line plots, bar plots, bar charts, and scatter plots, allowing for diverse data visualization. visualization.

Creating Static Plots

1 **Line Plots**

Line plots are created using the plot() function, great for visualizing visualizing trends over time.

2 **Customizing Plots**

Customization options like titles, labels, and legends enhance clarity.

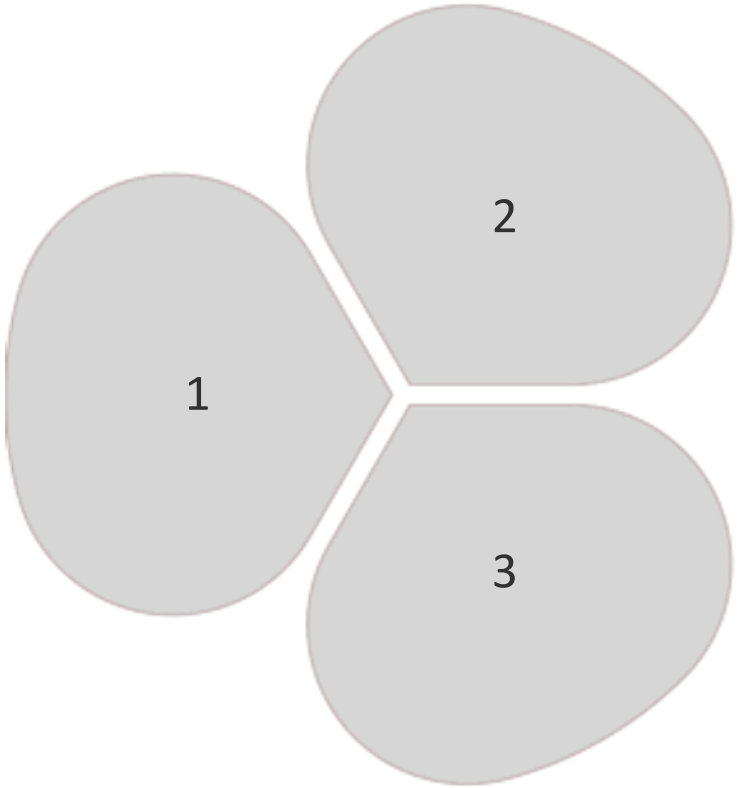
3 **Saving Figures**

Plots can be saved in formats like PNG, PDF, and SVG using savefig().

Animated Plots

Introduction to Animation Animation

Matplotlib's animation module allows for the creation of dynamic dynamic plots.



Creating Simple Animations

Users can update plot elements in a loop using the FuncAnimation class.

Customization in Animations

Animations can be customized with features like frame intervals and looping. looping.

Creating Interactive Plots with Matplotlib

Using Widgets

Matplotlib can integrate with Jupyter Widgets, allowing users to create interactive plots that respond to user inputs.

Event Handling

Event handling capabilities enable enable custom responses to mouse mouse and keyboard events for interactivity.

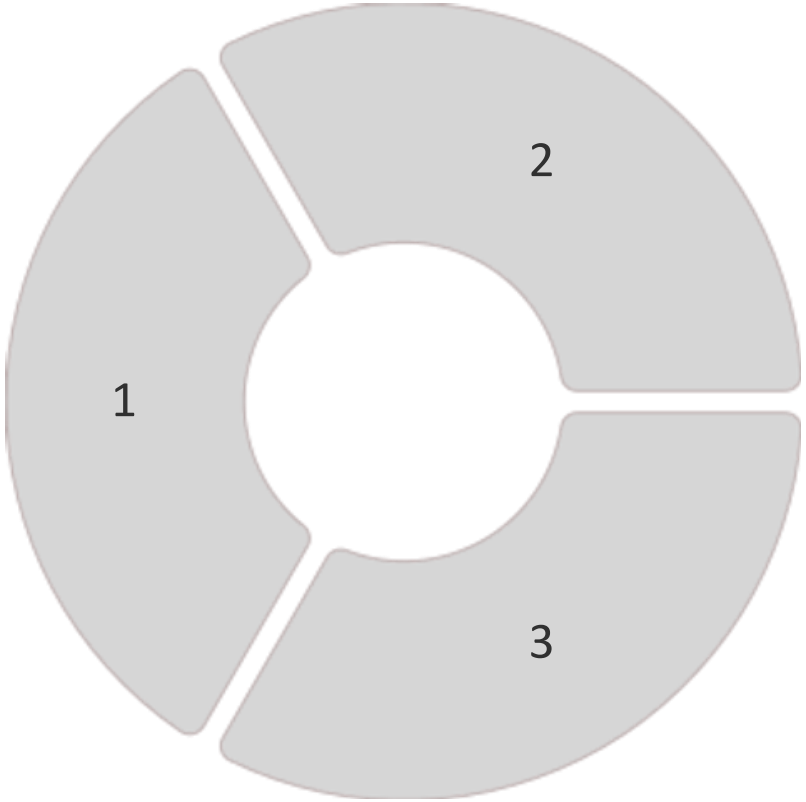
Integration with Other Libraries

Combining Matplotlib with libraries like Plotly or Bokeh can enhance interactivity and user engagement.

Case Study: Data Visualization

Dataset Selection

Choose a relevant dataset that showcases trends or patterns to visualize, maximizing the impact of impact of the plot.



Combining Plot Types

Demonstrate how to combine different plot types within a single figure to convey complex information effectively.

Interpreting Results

Discuss how to interpret the visualizations generated, emphasizing emphasizing insights drawn from the the data and how they can inform decision-making.

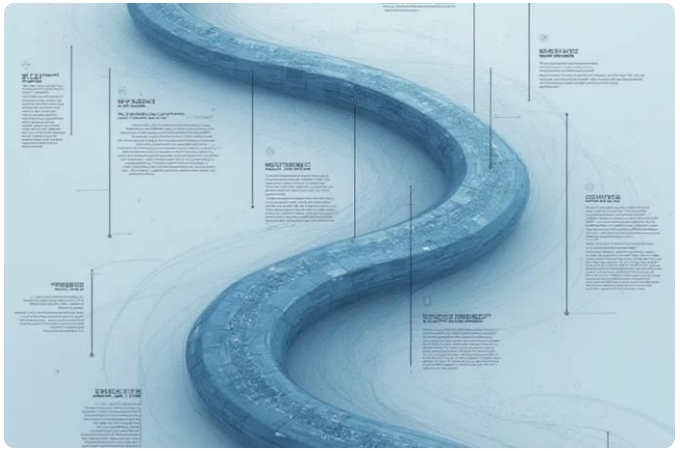
Best Practices

Simplicity and Clarity

Maintain simplicity in design while ensuring that plots are clear and comprehensible, avoiding clutter that can confuse viewers.

Color Usage

Using color wisely can significantly enhance the effectiveness of a plot, plot, with appropriate color schemes schemes aiding in distinguishing data data elements.



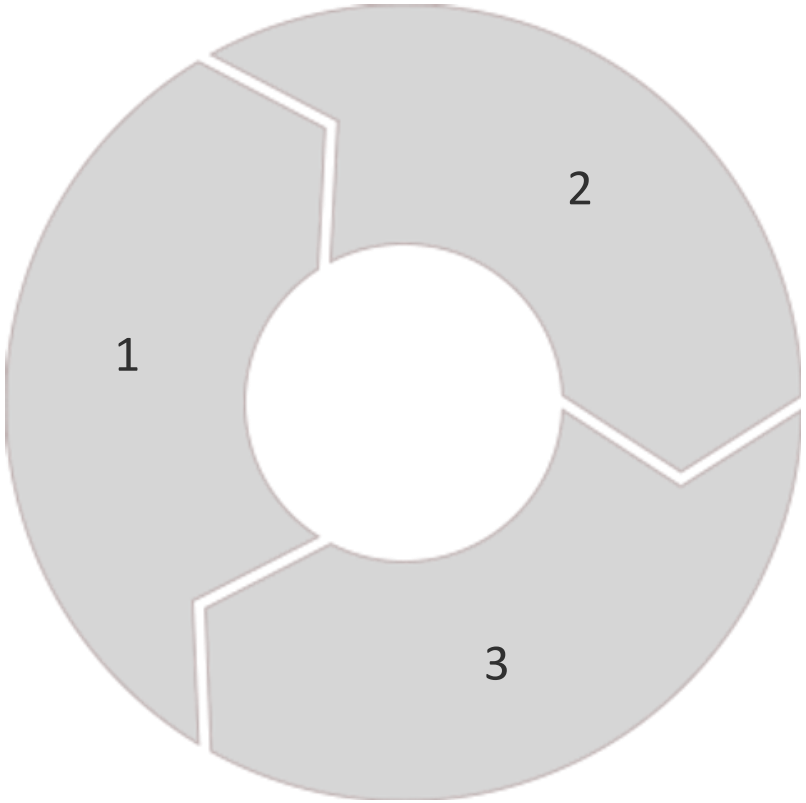
Annotations and Labels

Proper use of annotations and labels labels can provide additional context for context for the data presented, guiding guiding the audience's understanding. understanding.

Conclusion and Future Work

Recap of Key Points

Summarize the steps in creating static, animated, and interactive plots.



Further Exploration

Encourage viewers to delve deeper into Matplotlib's documentation.

Future Learning Opportunities Opportunities

Highlight the value of mastering Matplotlib and other tools.

Predictive Analytics Basics

Predictive analytics involves using data analysis techniques alongside machine learning to forecast future events and behaviors. This presentation will explore the fundamental concepts, processes, and tools in predictive analytics using Python.

Introduction to Predictive Analytics

Definition and Purpose

Predictive analytics uses statistical algorithms and machine learning techniques to identify the likelihood of future outcomes based on historical data.

Applications

Common applications include fraud fraud detection, customer segmentation, risk management, and and demand forecasting across various various industries.

Benefits

Employing predictive analytics helps helps businesses enhance decision-decision-making, optimize operations, operations, and improve customer engagements by anticipating trends. trends.

Data Collection and Preparation

1 Data Sources

Gathering data from various sources is crucial; these may include databases, APIs, and web scraping, each providing valuable insights.

2 Data Cleaning

Cleaning data ensures accuracy by removing duplicates, correcting errors, and handling missing values, which is essential for reliable analysis.

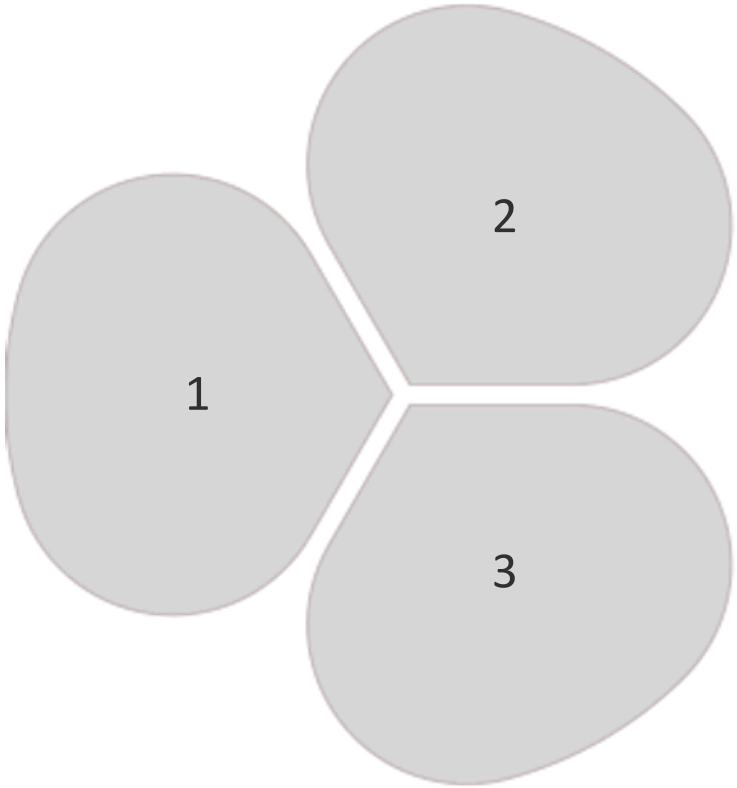
3 Feature Engineering

Creating new features from raw data can enhance model performance. This involves transforming existing variables or creating new ones to provide better predictive power.

Machine Learning Fundamentals

Supervised Learning

Involves training models on labeled labeled data, where the outcome is outcome is known, using algorithms algorithms such as linear regression, regression, decision trees, and support vector machines.



Unsupervised Learning

This approach uses unlabeled data to to discover patterns or groupings, with with techniques like clustering and principal component analysis (PCA).

Evaluation Metrics

Assessing model performance is vital. Common metrics include accuracy, precision, recall, and F1-score, helping to refine predictive models.

Implementing Predictive Models in Python

Libraries and Tools

Python libraries such as Scikit-learn, learn, Pandas, and Matplotlib are are widely used for data manipulation, model building, and and visualization.

Model Training

This involves selecting algorithms, fitting the model to the training dataset, and tuning hyperparameters for optimal performance.

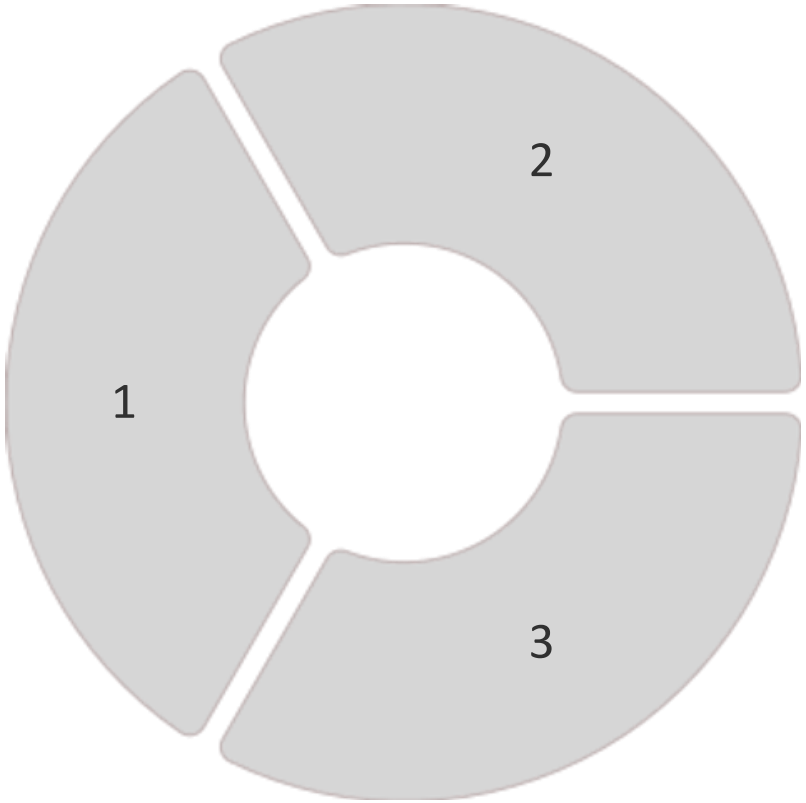
Prediction and Interpretation

After training, models can predict future outcomes, and interpreting results helps understand which factors most influence predictions.

Challenges in Predictive Analytics

Data Quality Issues

Poor data quality can lead to inaccurate predictions. It's essential to address issues related to data integrity and consistency.



Overfitting

This occurs when a model learns noise in noise in the training data rather than the than the underlying patterns, resulting in resulting in poor performance on new new data.

Ethical Considerations

Using predictive analytics raises ethical issues, including biases in data and decisions. Ensuring fair and responsible use is crucial.

Future Trends in Predictive Analytics

AI Integration

The integration of artificial intelligence intelligence will enable more sophisticated predictive capabilities, capabilities, providing deeper insights insights and automation.

Real-time Analytics

Businesses are increasingly leveraging leveraging real-time data for immediate immediate decision-making, enhancing enhancing agility and responsiveness. responsiveness.

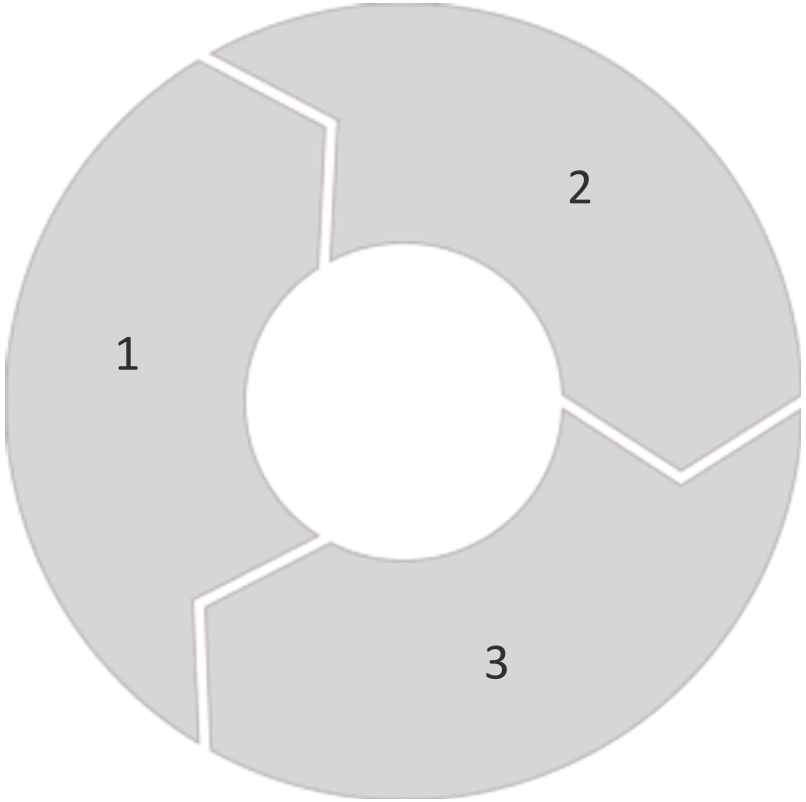
Personalization

As predictive analytics evolves, the ability to personalize customer experiences will improve, driving engagement and loyalty.

Harnessing Predictive Analytics

Understanding Basics

Learn foundational concepts of predictive analytics.



Utilizing Methods

Apply various methods to extract insights insights from data.

Overcoming Challenges

Identify and address common obstacles in implementation.

Introduction to Machine Learning Algorithms

Machine learning algorithms are vital tools in data analysis and artificial intelligence. They enable systems to learn from data, identify patterns, and make decisions with minimal human intervention.

Types of Machine Learning

Supervised Learning

Models are trained on labeled data to map inputs to correct outputs.

Unsupervised Learning

Analyzes unlabeled data to identify patterns and groupings.

Reinforcement Learning

Agents make decisions to maximize cumulative reward in an environment.

Popular Algorithms Overview

1 Linear Regression

A fundamental algorithm for predicting continuous outcomes. It assumes a linear relationship between input variables and a single output, making it easy to interpret.

2 Decision Trees

These models use a tree-like structure to make decisions, decisions, allowing for both regression and classification classification tasks. They split data into branches based on based on feature values.

3 Support Vector Machines

SVMs are powerful for classification tasks. They find the hyperplane that best separates data points of different classes in high-dimensional space.

4 Neural Networks

Inspired by the human brain, these algorithms consist of interconnected layers of nodes (neurons) and are effective for complex data patterns.

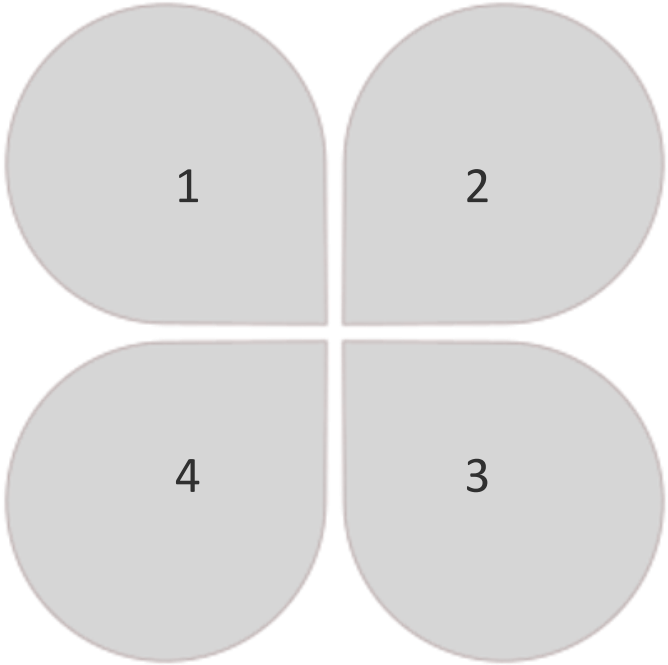
Evaluation Metrics

Accuracy

The ratio of correctly predicted instances to the total instances, potentially misleading for imbalanced data.

Confusion Matrix

A table that visualizes classification model performance, detailing true positives, false positives, true negatives, and false negatives.



Precision and Recall

Precision reflects the accuracy of positive predictions, while recall assesses the model's ability to identify all relevant instances.

F1 Score

The harmonic mean of precision and and recall, providing a balanced measure measure for classification models, essential for imbalanced datasets.

Applications of Machine Learning

Healthcare

Machine learning is employed for predictive analytics in analytics in patient diagnosis and treatment recommendations, improving clinical decision-making making processes.

Finance

Algorithms are critical in fraud detection systems, risk risk assessment, algorithmic trading, and personalized personalized financial recommendations for customers. customers.

Marketing

Businesses use machine learning for customer segmentation, segmentation, predicting consumer behavior, and optimizing optimizing marketing campaigns through targeted strategies. strategies.

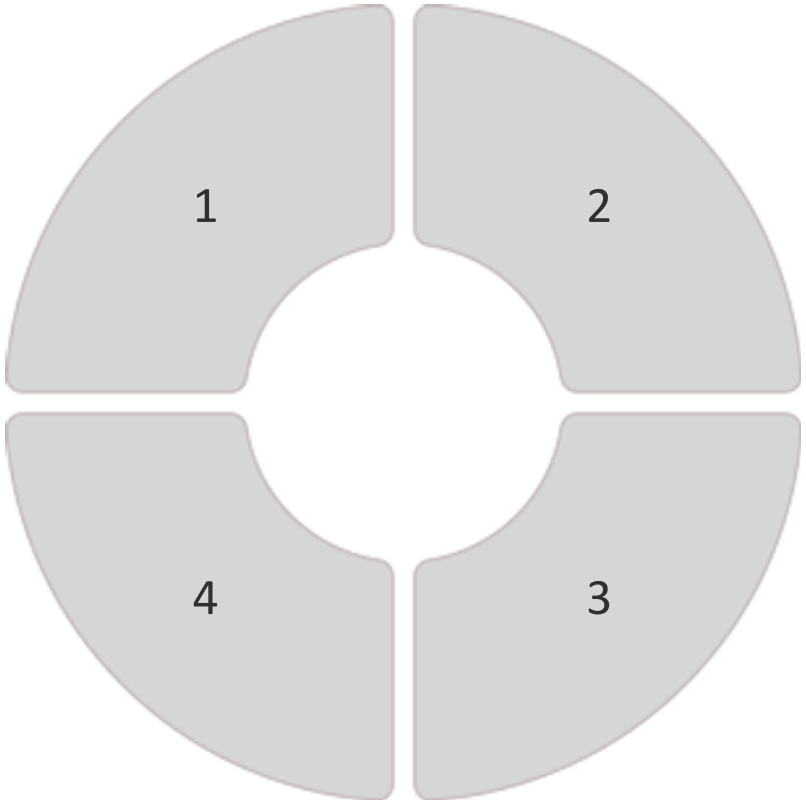
Autonomous Vehicles

Machine learning plays a key role in enabling vehicles to perceive their environment and make real-time decisions, enhancing safety and efficiency in transportation.

Challenges in Machine Learning

Data Quality

Poor quality data can lead to misleading misleading insights and poor algorithm algorithm performance. Ensuring accurate, complete, and relevant data is data is essential for successful machine machine learning applications.



Overfitting

This occurs when models perform well on training data but poorly on unseen data due to being too complex. Techniques like cross-validation are crucial to mitigate this risk.

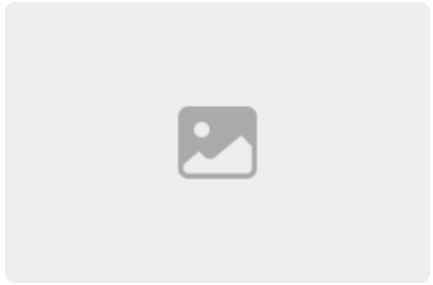
Ethical Concerns

Issues like algorithmic bias, privacy concerns, and the potential for automation to displace jobs are emerging concerns that require careful consideration.

Interpretability

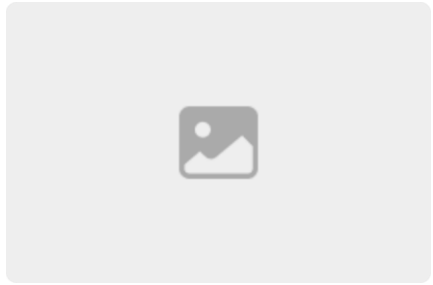
Many machine learning models can be black boxes, making it difficult to interpret their decisions. This is particularly problematic in regulated industries like healthcare.

Future Trends in Machine Learning



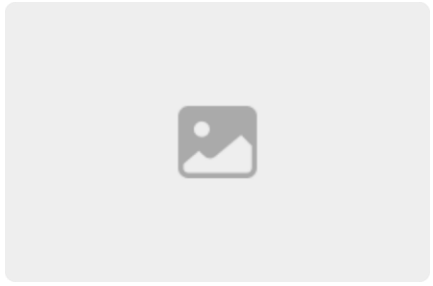
Explainable AI

A push towards interpretable models for transparency in AI systems.



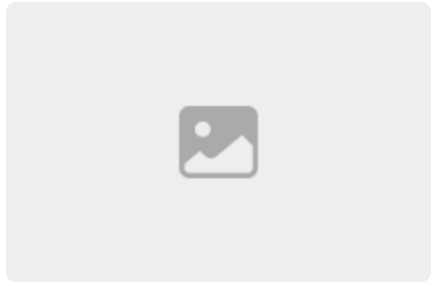
Automated Machine Learning

Automating the entire machine learning process for non-experts.



Federated Learning

Training models across decentralized data sources for enhanced privacy.



Integration with IoT

ML systems combined with IoT for data-driven insights across industries.

Overview of Key Machine Learning Techniques

This presentation provides an overview of essential machine learning techniques, discussing their applications, advantages, and key advantages, and key concepts, aimed at understanding the vast field of machine learning.

Understanding Supervised Learning

Definition and Purpose

Trains models on labeled data to predict outcomes.

Common Algorithms

Includes linear regression, logistic regression, SVM, and decision trees. trees.

Applications

Used in spam detection, sentiment analysis, and credit scoring.

Understanding Unsupervised Learning

1 Understanding Unsupervised Learning

This technique deals with unlabeled data, identifying patterns without specific output guidance.

2 Clustering Techniques

Algorithms like K-means and DBSCAN help group similar data points, identifying intrinsic relationships.

3 Dimensionality Reduction Reduction

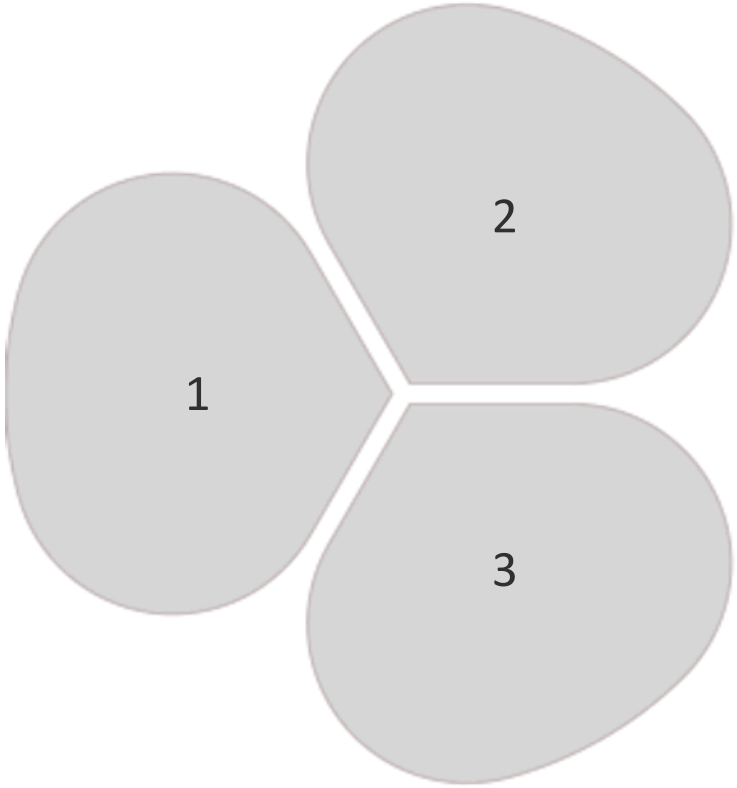
Techniques such as PCA and t-SNE simplify data analysis while retaining essential information.

Reinforcement Learning

Reinforcement learning involves agents that use trial and error to make decisions by receiving feedback in the form of rewards or penalties, optimizing their strategies over time.

Concept Overview

Reinforcement learning is based on based on agents who learn to make make decisions through trial and and error, receiving rewards or penalties in an environment.



Key Components

Essential elements include the agent, agent, environment, actions, rewards, rewards, and policy, with the agent adapting its strategy to maximize rewards.

Applications

Applied in robotics, game playing like AlphaGo, and personalized recommendations, demonstrating versatility and effectiveness.

Understanding Neural Networks

Introduction to Neural Networks

Neural networks are computing systems inspired by the human brain's interconnected neuron structure.

Types of Neural Networks

Various architectures exist like feedforward networks, CNNs for image recognition, and RNNs for for sequential data analysis.

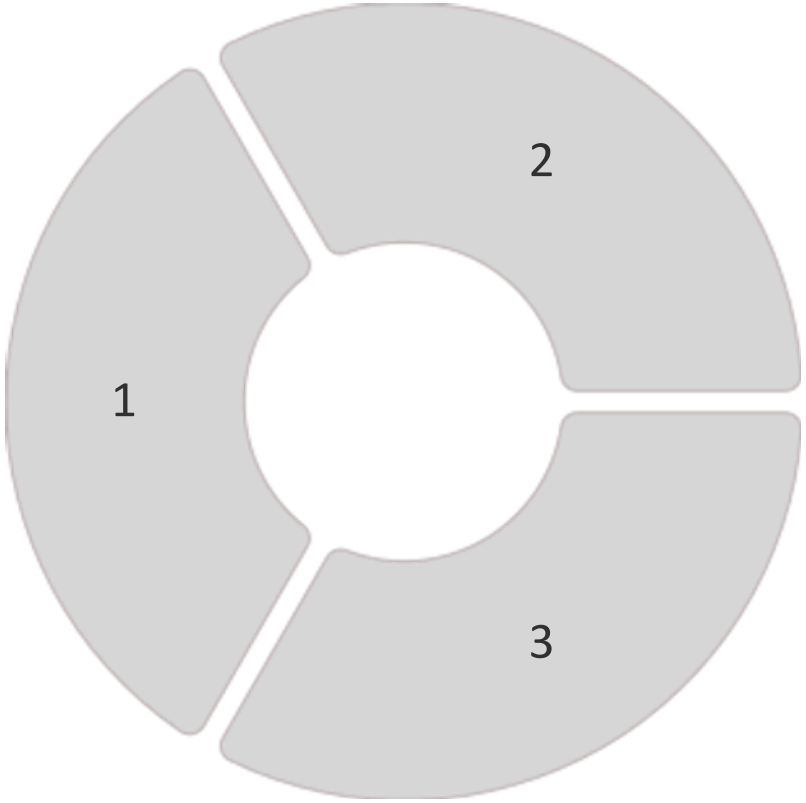
Deep Learning

Deep learning utilizes multiple layers to extract high-level abstractions, driving advances in NLP and computer vision.

Decision Trees

Understanding Decision Trees

Decision trees are flowchart-like structures that make decisions based on answering a series of questions derived from input features, leading to a final output decision.



Advantages

They are easy to interpret, require minimal data preprocessing, and can handle both numerical and categorical data. Their transparency is useful in decision-making processes.

Limitations

Decision trees can be prone to overfitting, especially with complex datasets. Techniques such as pruning or ensemble methods like random forests help mitigate this issue.

Ensemble Learning Overview

Overview of Ensemble Learning

Combines multiple algorithms to improve model performance.

Common Methods

Includes bagging and boosting techniques to enhance accuracy.

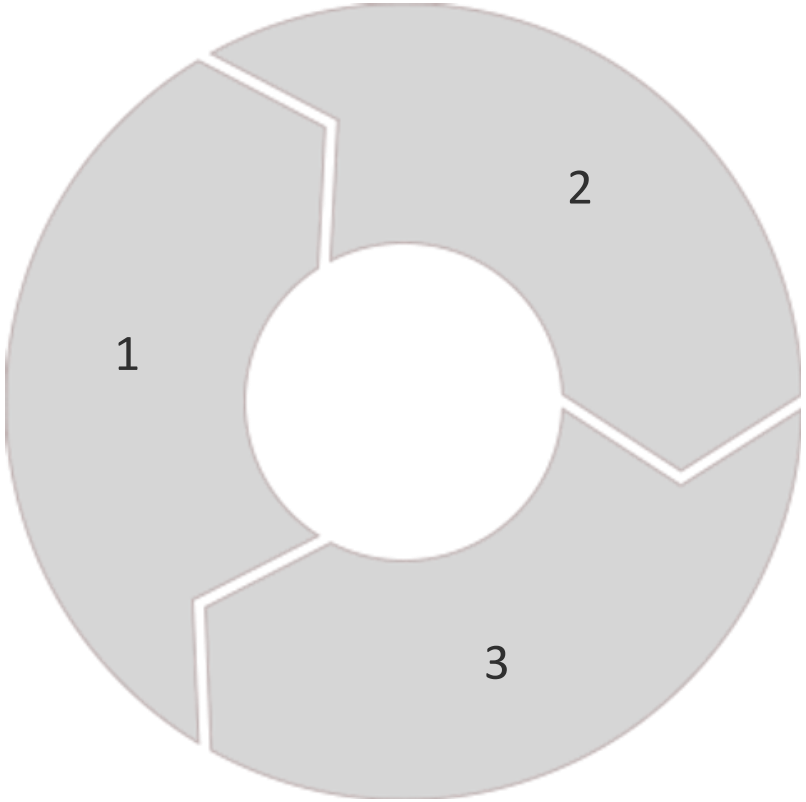
Applications

Used in finance, healthcare, and marketing for predictive modeling.

Model Evaluation and Validation

Evaluation Techniques

Assessing models involves metrics like accuracy, precision, recall, and F1-score, providing insights into a model's performance and its suitability for a given task.



Validation Methods

Cross-validation techniques such as k-fold k-fold help ensure that models generalize generalize well to unseen data, reducing reducing the risk of overfitting and enhancing predictive reliability.

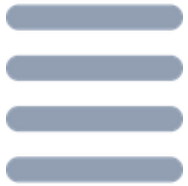
Importance of Data Splitting

Properly splitting data into training, validation, and test sets is crucial to provide unbiased estimates of model performance, guiding further tuning and improvement efforts.

Future Trends in Machine Learning

Emergence of AutoML

Automated Machine Learning (AutoML) tools are gaining prominence, allowing non-experts to build effective machine learning models by automating key processes.



Ethics and Fairness

Addressing ethical concerns and biases in algorithms is vital to ensure fair and responsible use of AI technologies in society.

Integration with Other Technologies

The convergence of machine learning learning with IoT, blockchain, and edge edge computing leads to innovative innovative applications across industries.

Concept of Training and and Testing Models

This presentation outlines the core principles of training and testing models in models in machine learning. It explains their significance, methods, and best best practices to ensure optimal model performance.

Introduction to Models

Definition of Models

A mathematical representation that that learns patterns from data.

Importance of Models

Essential for classification, regression, regression, and clustering tasks.

Types of Models

Includes linear regression, decision trees, SVM, and neural networks.

Training Phase

1

Training Data

A model learns from input-output pairs, adjusting parameters to minimize errors.

2

Overfitting vs. Underfitting

Overfitting captures noise; underfitting fails to capture patterns. patterns.

3

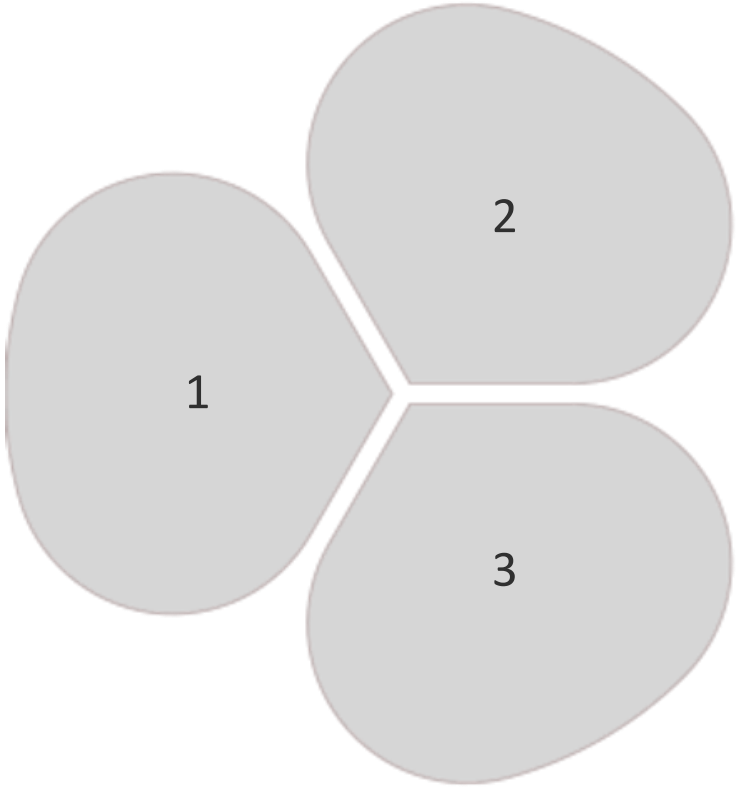
Model Evaluation

Techniques like cross-validation and metrics like accuracy assess a model's generalization.

Testing Phase

Testing Data

After training, the model is evaluated against a separate testing dataset that it hasn't seen.



Significance of Testing

Testing ensures that models don’t just memorize the training data.

Performance Metrics

Different metrics like precision, recall, recall, and AUC help determine a model's model's strengths.

Best Practices for Model Training

Data Preprocessing

Cleaning and preprocessing data is vital because the quality of data directly impacts model performance. Techniques include normalization, handling missing values, and feature engineering.

Hyperparameter Tuning

Adjusting hyperparameters, like learning rate and batch size, can significantly enhance model performance. Techniques like grid grid search and random search are are commonly used for this purpose. purpose.

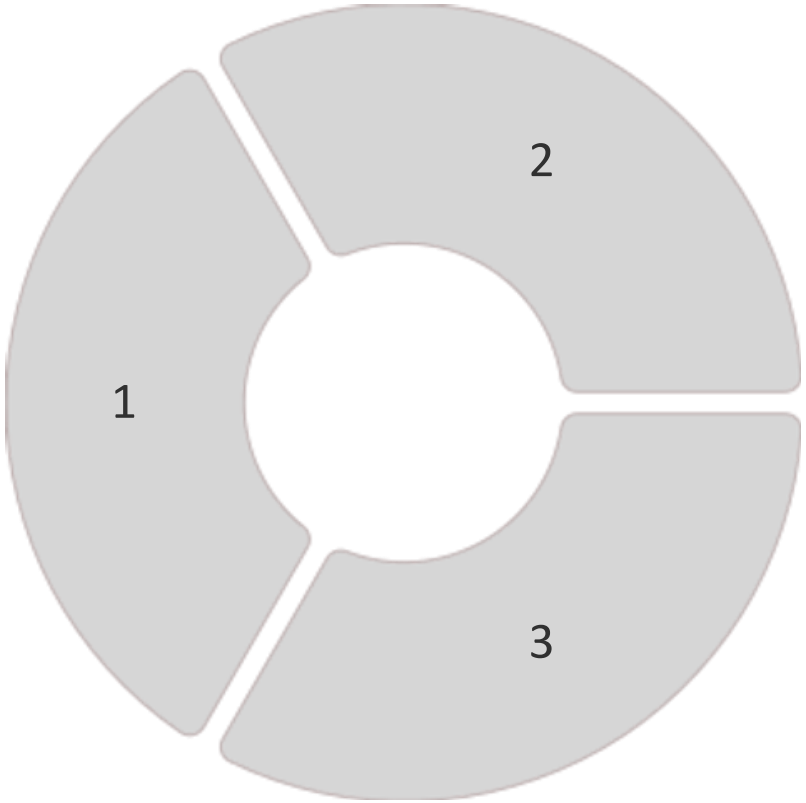
Continuous Learning

Models should be regularly updated with new data to maintain accuracy. Implementing mechanisms for continuous learning can help adapt models to changing environments.

Challenges in Model Training and Testing

Data Imbalance

Techniques such as resampling or using weighted loss functions can help mitigate this issue.



Computational Resources

Access to GPUs or cloud computing can can alleviate these resource constraints. constraints.

Interpretability

Using explainable AI techniques can help help stakeholders understand model model predictions.

Future Trends in Model Development

Automated Machine Learning

Automating model selection and hyperparameter tuning increases accessibility.

Federated Learning

Training on decentralized data sources while preserving privacy.

Explainable AI

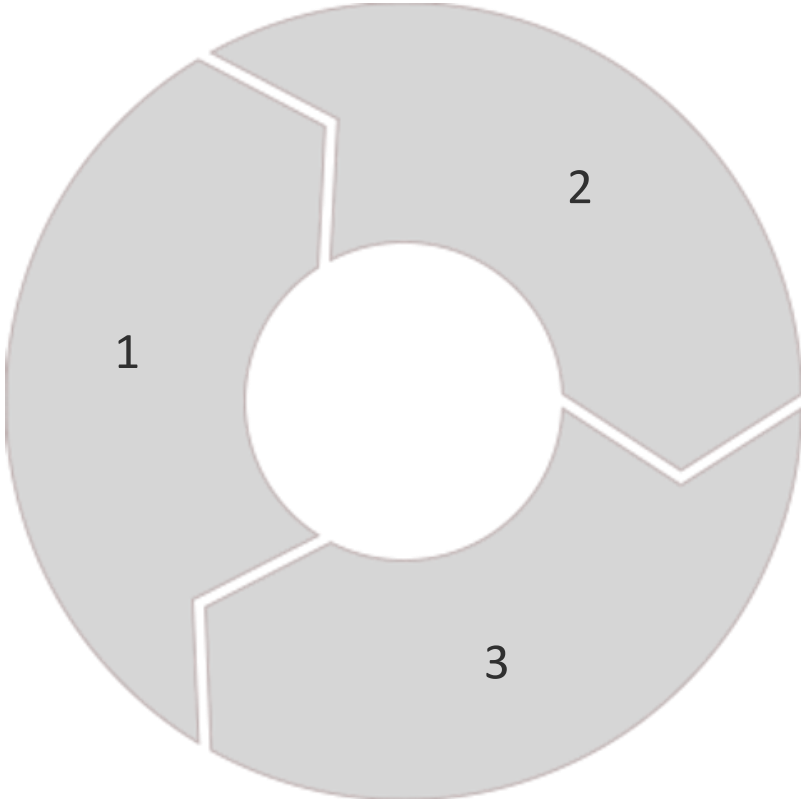
The growing need for transparency and transparency and insights in complex complex models.

Importance of Training and Testing Models

Understanding the concepts of training and testing models is crucial for creating effective machine learning solutions. By following best practices and staying informed about new trends, practitioners can better navigate the challenges of model development and deployment.

Model Understanding

Essential for effective machine learning solutions.



Best Practices

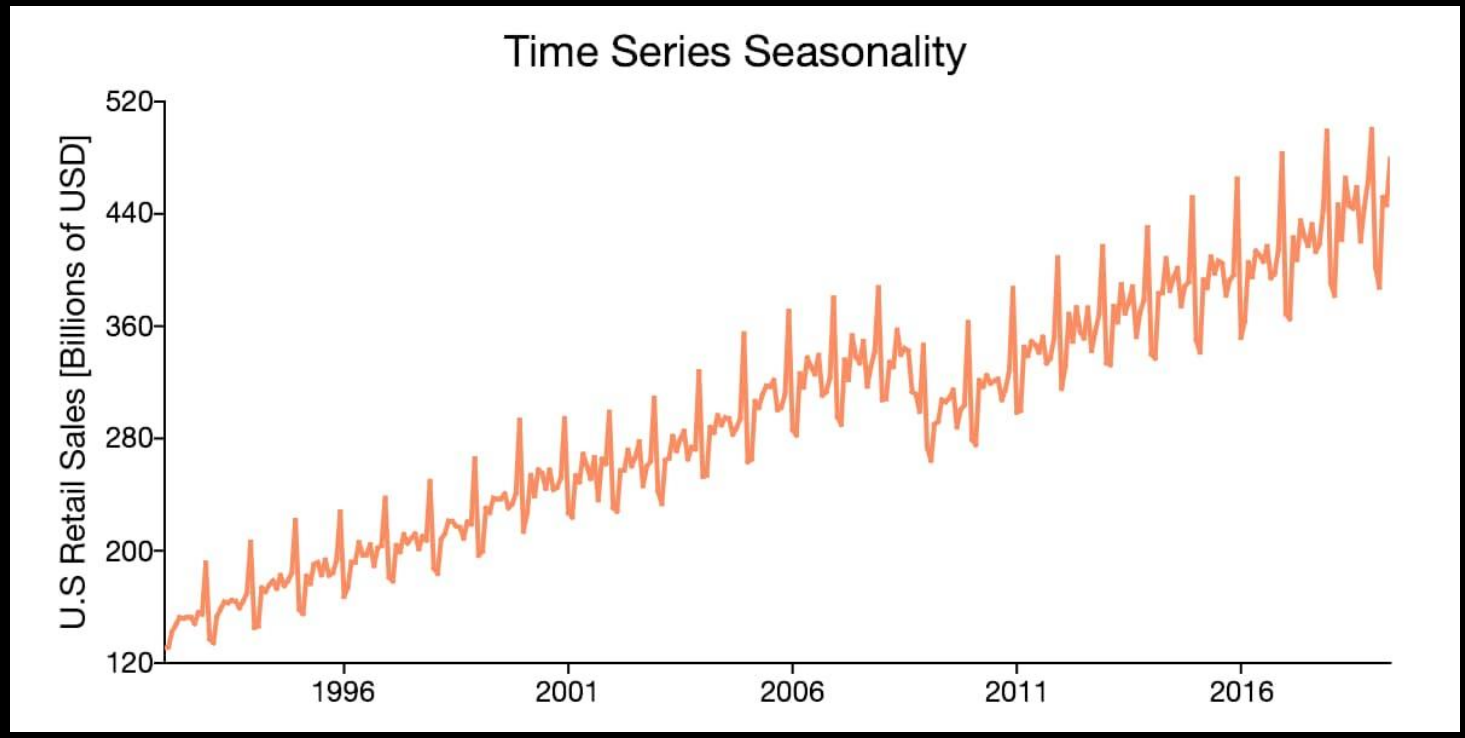
Following guidelines leads to better model outcomes.

Staying Informed

Awareness of trends helps in the deployment process.

What is a Time Series?

- A time series is a sequence of data points collected, recorded, or measured at successive, evenly-spaced time intervals.

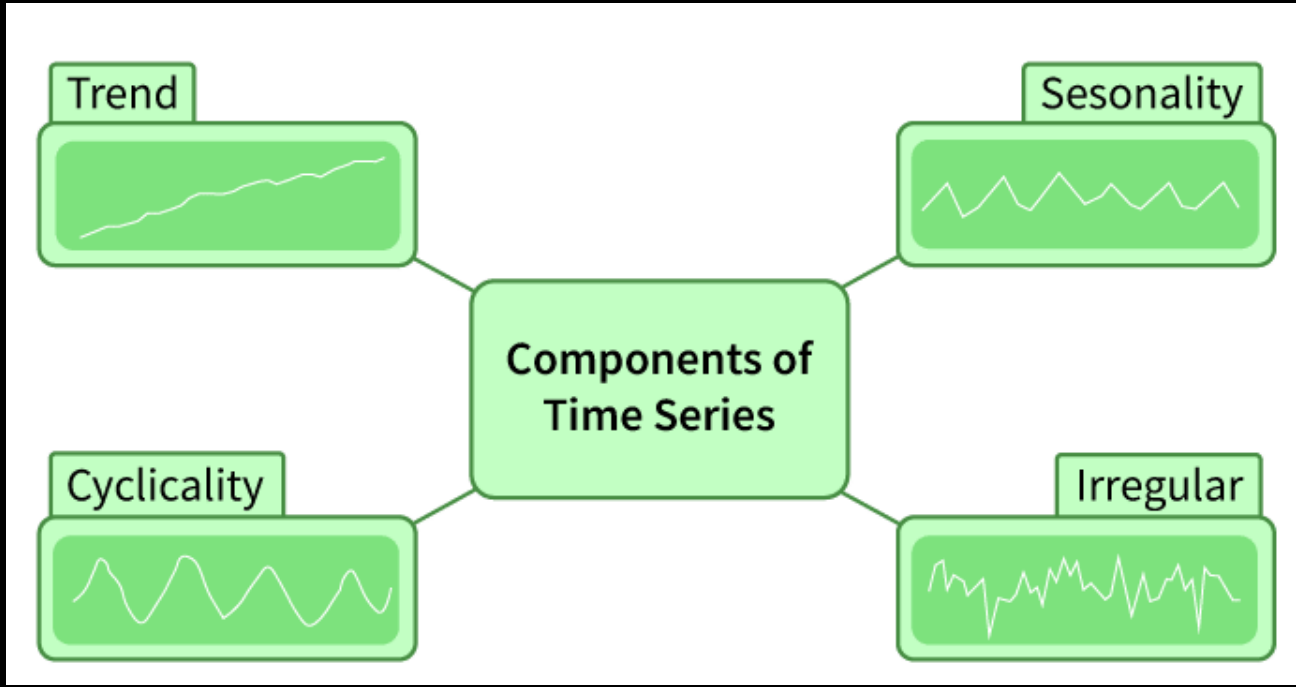


Understanding Time-Series Data

- Definition: Time-series data is a sequence of data points collected or recorded at specific time intervals.
- Examples: Stock prices, weather data, sales figures, etc.
- Importance: Helps in analyzing trends, making predictions, and understanding underlying patterns over time.

Time series data is often represented graphically as a line plot, with time depicted on the horizontal x-axis and the variable's values displayed on the vertical y-axis. This graphical representation facilitates the visualization of trends, patterns, and fluctuations in the variable over time, aiding in the analysis and interpretation of the data.

Components of Time Series Data

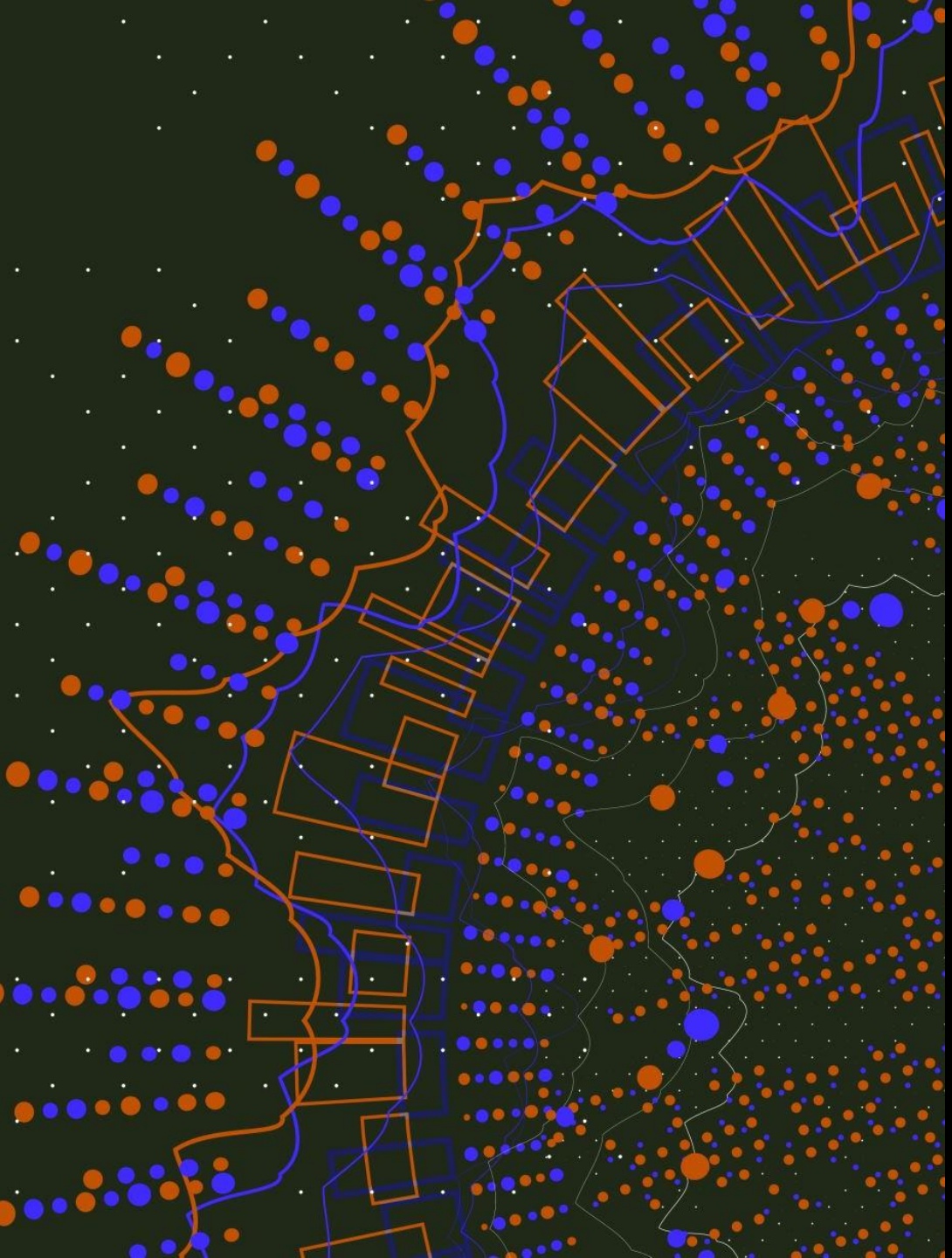


Trend: Trend indicates the long-term direction of data, showing whether it increases, decreases, or remains stable, and can be either linear or nonlinear.

Seasonality: Seasonality involves regular fluctuations in the data that occur at consistent intervals, often influenced by seasonal changes or holidays.

Cyclic variations: These are longer-term fluctuations that occur over multiple years, reflecting economic cycles without a fixed period.

Irregularity (or Noise): This refers to unpredictable data fluctuations not linked to trend, seasonality, or cyclic variations, often caused by random events or errors, making pattern identification difficult.



What is Time Series Forecasting?

- Time Series Forecasting is a statistical technique used to predict future values of a time series based on past observations. In simpler terms, it's like looking into the future of data points plotted over time. By analyzing patterns and trends in historical data, Time Series Forecasting helps make informed predictions about what may happen next, assisting in decision-making and planning for the future.

Applying Forecasting Techniques

- Methods:
  - Moving Average: Smooths out short-term fluctuations and highlights longer-term trends.
  - Exponential Smoothing: Gives more weight to recent observations.
  - ARIMA (AutoRegressive Integrated Moving Average): Combines autoregression, differencing, and moving average.
  - Machine Learning Models: Use algorithms like LSTM (Long Short-Term Memory) for complex forecasting.
- Steps
  - Data Preparation: Clean and preprocess data.
  - Model Selection: Choose appropriate forecasting model.
  - Validation: Test model accuracy using historical data.
  - Implementation: Apply model to make future predictions.

Different Time series visualization graphs

1. **Line Plots:** Line plots display data points over time, allowing easy observation of trends, cycles, and fluctuations.

2. **Seasonal Plots:** These plots break down time series data into seasonal components, helping to visualize patterns within specific time periods.

3. **Histograms and Density Plots:** Shows the distribution of data values over time, providing insights into data characteristics such as skewness and kurtosis.

4. **Autocorrelation and Partial Autocorrelation Plots:** These plots visualize correlation between a time series and its lagged values, helping to identify seasonality and lagged relationships.

5. **Spectral Analysis:** Spectral analysis techniques, such as periodograms and spectrograms, visualize frequency components within time series data, useful for identifying periodicity and cyclical patterns.

6. **Decomposition Plots:** Decomposition plots break down a time series into its trend, seasonal, and residual components, aiding in understanding the underlying patterns.

Q&A

Usama Wahab Khan
Twitter : @usamawahabkhan
LinkedIn : Usamawahabkhan

Evolution Technologies