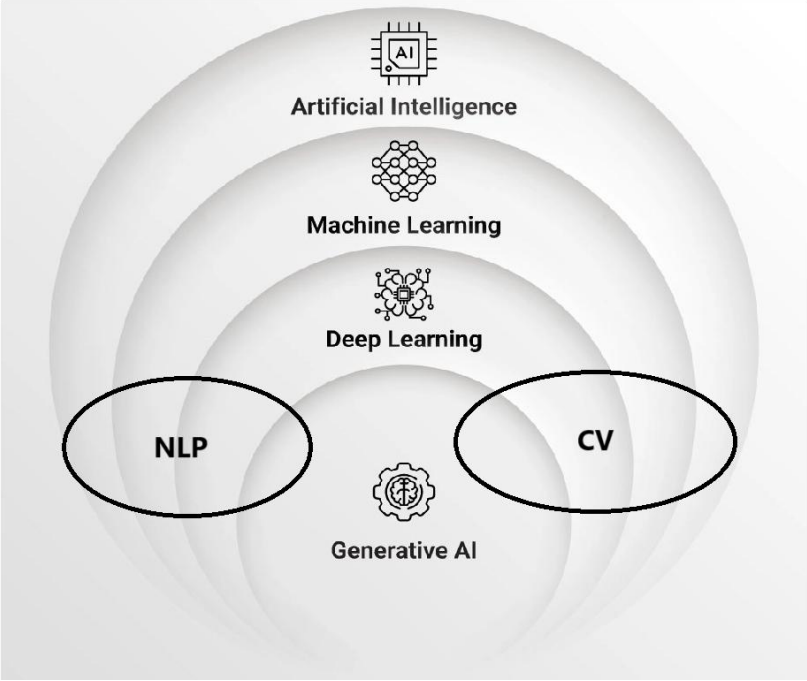
Pandas Notes

Introduction to Data Science

Data science enables organizations to make informed decisions, solve problems, and understand human behavior. The most common languages used for data science are Python and R.

AI vs ML vs DL vs NLP vs CV

- **AI (Artificial Intelligence):** The broader concept of machines performing tasks that require human intelligence.
- **ML (Machine Learning):** A subset of AI where algorithms learn patterns from data.
- **DL (Deep Learning):** A subset of ML using neural networks with multiple layers.
- **NLP (Natural Language Processing):** A branch of AI that helps computers understand and process human language.
- **CV (Computer Vision):** A field of AI that enables machines to interpret visual data.



ML Project Pipeline

1. Data Collection
2. Data Preprocessing
3. Feature Engineering
4. Model Selection
5. Model Training
6. Model Evaluation
7. Deployment

Installing and Importing Pandas

Installation:

Pandas can be installed using pip:

```
!pip install pandas
```

To use Pandas in Python, import it as follows:

```
import pandas as pd
```

Key Data Structures in Pandas

1. Series (1D Array):

A Series is a one-dimensional array-like object containing values and an index.

Ex1:

```
a = [1, 7, 2]

ser = pd.Series(a)

print(ser)
```

Ex2:

```
a = [1, 7, 2]

ser = pd.Series(a, index = ["x", "y", "z"])
```

```
print(ser["z"])
```

Ex3:

```
d = {"day1":100 , "day2":200, "day3":300}
ser = pd.Series(d)
print(ser["day2"])
```

Ex4:

```
data = (10, 20, 30, 40)
ser = pd.Series(data)
print(ser)
```

Ex5:

```
ser = pd.Series(10 , index = ["a", "b", "c"])
print(ser)
```

2. DataFrames (2D Array):

A DataFrame is a two-dimensional, tabular data structure with labeled axes (rows and columns).

Ex1:

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

Ex2:

```
fruits = ["Apple", "Banana", "Orange", "grapes"]
df = pd.DataFrame(fruits)
print(df)
```

Ex3:

```
fruits = [["Apple", 100], ["Banana", 200], ["Orange", 300], ["grapes",400]]
df = pd.DataFrame(fruits, columns = ["Fruit name" , "Price"])
print(df)
```

Ex4:

```
Fruit = {
    "Fruit name" : ["Apple", "Banana", "Orange", "grapes"],
    "Price" : [100, 200, 300, 400]
}
df = pd.DataFrame(Fruit)
print(df)
```

3. Load and Save data from CSV and Excel files:

Loading CSV files:

```
df_csv = pd.read_csv("C:/Users/Prikshit_Ishi/CSV_data.csv")
print(df_csv)
```

Saving CSV files:

```
df_csv.to_csv("abc.csv")
```

Loading Excel files:

```
df = pd.read_excel("Excel_data.xlsx")
```

Saving Excel files:

```
df.to_excel("xyz.xlsx")
```

3. Exploring the DataFrames:

Head (First 5 rows by default):

Used to quickly inspect the first few rows of a DataFrame for analysis.

```
df.head()
```

```
df.head(10)
```

Tail (Last 5 rows by default):

Displays the last few rows of the DataFrame for quick inspection.

```
df.tail()
```

```
df.tail(10)
```

Selecting Data

Selecting a Single Column

Using Bracket Notation

This method is commonly used to access a single column in a DataFrame.

```
df['Name']
```

Using Dot Notation

Another way to access a column, but it doesn't work if the column name has spaces or special characters.

```
df.Name
```

Selecting Multiple Columns

This method allows us to extract multiple columns by passing a list of column names.

```
df[['ID', 'Salary']]
```

Selecting Rows and Columns

Using loc (Label-Based Indexing)

This method selects rows and columns based on labels.

```
df.loc[1:5, 'Name' : 'Profession']
```

Using iloc (Position-Based Indexing)

This method selects rows and columns by numerical index positions.

```
df.iloc[1:5, 1:4]
df.iloc[2:5, [0,3]]
```

Modifying DataFrame Index

Setting a Column as Index

This changes the DataFrame index to a specific column.

```
df.set_index('ID', inplace=True)
```

Resetting the Index

Restores the default integer index.

```
df.reset_index(inplace=True)
```

Checking DataFrame Properties

Shape of DataFrame

Returns the number of rows and columns in the DataFrame.

```
df.shape
```

Data Types of Columns

Displays the data types of all columns.

```
df.dtypes
```

Changing Data Type of a Column

Converts the 'Name' column to string type.

```
df['Name'] = df.Name.astype('string')
```

Summary of DataFrame

Provides a concise summary including data types and non-null values.

```
df.info()
```

Statistical Summary of DataFrame

Returns summary statistics for numerical columns.

```
df.describe()
```

Handling Missing Values

Identifying Missing Values

Returns a DataFrame indicating where NaN values exist.

```
df.isnull()
```

Counting Missing Values in Each Column

Shows the number of missing values in each column.

```
df.isnull().sum()
```

Creating a Duplicate DataFrame

Creates a copy of the DataFrame.

```
df1 = df.copy()
```

Removing Rows with Missing Values

Removes rows that contain NaN values.

```
df1 = df1.dropna()
df1.isnull().sum()
```

Removing Columns with Missing Values

Drops columns where any NaN values are present.

```
df2 = df2.dropna(axis=1, how='any')
```

Replacing Values in a DataFrame

Replacing Specific Values in a Column

Replaces occurrences of "Olivia" with "Puneet".

```
df4["Name"] = df4.Name.replace("Olivia", "Puneet")
```

Handling Duplicate Values

Identifying Duplicate Rows

Checks for duplicate rows.

```
df.duplicated()
```

Counting Duplicate Rows

Returns the number of duplicate rows.

```
df.duplicated().sum()
```

Displaying Duplicate Rows

Shows all duplicate rows.

```
df[df.duplicated()]
```

Checking Duplicates Based on a Specific Column

Finds duplicate rows based on the 'Profession' column.

```
df[df.duplicated(subset=["Profession"])]
```

Removing Duplicate Rows

Drops duplicate rows from the DataFrame.

```
df5 = df5.drop_duplicates()
```

Removing Duplicates Based on a Specific Column

Drops duplicates while keeping the first occurrence.

```
df5 = df5.drop_duplicates(subset=["Profession"])
```

Aggregation Functions

Sum of a Column

Returns the sum of all values in the 'Salary' column.

```
df['Salary'].sum()
```

Mean of a Column

Computes the average value of the 'Salary' column.

```
df['Salary'].mean()
```

Grouping and Summing Values

Groups data by 'Profession' and sums numerical values.

```
df.groupby('Profession').sum()
```

Sorting Data

Sorting Rows by Column Values

Sorts DataFrame based on 'Age' in ascending order.

```
df.sort_values('Age')
```

Sorting Rows by Index

Sorts the DataFrame based on the index.

```
df.sort_index()
```

Ranking Data

Assigning Rank to a Column

Assigns ranks to values in the 'Salary' column.

```
df['rank'] = df['Salary'].rank()
```

Applying Functions to Columns

Applying a Lambda Function to a Column

Applies a 10% salary increment.

```
df['Salary'] = df['Salary'].apply(lambda x : x*1.1)
```

Merging DataFrames

Creating Sample DataFrames

Creates two DataFrames for merging.

```
df1 = pd.DataFrame({"ID": [1,2,3], "Name": ['Sumit', 'Puneet', 'Rahul']})
df2 = pd.DataFrame({"ID": [2,3,4], "Score": [85, 90, 95]})
```

Inner Join on a Common Column

Merges DataFrames on the 'ID' column using an inner join.

```
merge_df = pd.merge(df1, df2, on='ID')
```

Left Join on a Common Column

Keeps all rows from df1 and matches records from df2 where possible.

```
merge_df = pd.merge(df1, df2, on='ID', how='left')
print(merge_df)
```