Getting familiar with pandas

This section is aimed at introducing the foundational concepts and functionalities of the Pandas library, which is an essential tool for data manipulation in Python. You'll start by understanding the two primary data structures in Pandas: Series and DataFrames. Series is a one-dimensional array, while DataFrames are two-dimensional tables that can store heterogeneous data. This section will also guide you on how to create these structures from various data sources, such as lists, dictionaries, and CSV files. You'll practice basic operations like selecting, filtering, and modifying data, which are crucial for working with data in Pandas.

install pandas

In [17]:
```
#install pandas package
import pandas as pd
```

creation of dataframes and series using pandas:

You'll learn how to generate these structures from basic Python data types such as lists and dictionaries. Additionally, you'll practice loading data from external sources like CSV files into DataFrames, which is a common task in data science.

from a list

In [18]:
```
#creation of series
series = pd.Series([2,4,6,8,10])
series
```

Out[18]:
```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

In [20]:
```
#creation of dataframe
df = pd.DataFrame(series,columns=['Numbers'])
df
```

Out[20]:

	Numbers
0	2
1	4
2	6
3	8
4	10

from a dictionary

In [22]:
```
df1 = pd.DataFrame({'name':['jack','joy','john'],
                    'course':['csd','csm','mech']})
df1
```

Out[22]:

	name	course
0	jack	csd
1	joy	csm
2	john	mech

from a CSV file

In [28]:
```
# creation of df using a CSV file
df2 = pd.read_csv('notepad.csv')
df2
```

Out[28]:

	name	age	grade	marks
0	a	15.0	A	23.0
1	b	17.0	A	24.0
2	c	19.0	A	22.0
3	d	NaN	B	NaN
4	e	24.0	NaN	27.0
5	f	10.0	NaN	NaN
6	g	16.0	B	NaN

some common operations

it covers essential operations you can perform on DataFrames and Series, such as selecting specific data, filtering rows based on conditions, and modifying the contents of a DataFrame. These operations form the backbone of data manipulation tasks and are necessary for effective data analysis.

selecting data

In [30]:
```
# selecting a single column
df2['name']
```

Out[30]:
```
0    a
1    b
2    c
3    d
4    e
5    f
6    g
Name: name, dtype: object
```

In [31]:
```
#selecting multiple columns
df2[['name','age']]
```

Out[31]:

	name	age
0	a	15.0
1	b	17.0
2	c	19.0
3	d	NaN
4	e	24.0
5	f	10.0
6	g	16.0

In [32]:
```
#selecting specific rows
df2.iloc[1]
```

Out[32]:
```
name        b
age      17.0
grade       A
marks    24.0
Name: 1, dtype: object
```

In [34]:
```
df2.iloc[0:2]
```

Out[34]:

	name	age	grade	marks
0	a	15.0	A	23.0
1	b	17.0	A	24.0

filtering rows:

In [35]:
```
df2[df2['age']>16]
```

Out[35]:

	name	age	grade	marks
1	b	17.0	A	24.0
2	c	19.0	A	22.0
4	e	24.0	NaN	27.0

modifying data

In [37]:
```
df2['age'] = df2['age']+5
df2
```

Out[37]:

	name	age	grade	marks
0	a	25.0	A	23.0
1	b	27.0	A	24.0
2	c	29.0	A	22.0
3	d	NaN	B	NaN
4	e	34.0	NaN	27.0
5	f	20.0	NaN	NaN
6	g	26.0	B	NaN

data handling with pandas:

In this section, the focus shifts to practical data handling techniques using Pandas. You will learn how to read data from different file formats (like CSV), handle missing data, and perform essential data transformations. The ability to clean and preprocess data is vital in data analysis, and this section will help you gain expertise in tasks such as filling or dropping missing values, removing duplicates, and converting data types. By the end of this section, you should be comfortable with the basic steps required to prepare a dataset for analysis.

reading data from files

it includes how to import data from various file formats into Pandas DataFrames. This skill is crucial as most data analysis tasks begin with loading data from external sources, such as CSV files. You will practice using functions like pd.read_csv() to read data efficiently.

In [39]:
```
# reading data from a csv file
df2 = pd.read_csv('notepad.csv')
df2
```

Out[39]:

	name	age	grade	marks
0	a	15.0	A	23.0
1	b	17.0	A	24.0
2	c	19.0	A	22.0
3	d	NaN	B	NaN
4	e	24.0	NaN	27.0
5	f	10.0	NaN	NaN
6	g	16.0	B	NaN

handling missing data

it focuses on strategies for dealing with missing or incomplete data, a common issue in real-world datasets. You'll learn methods to fill missing values with appropriate substitutes, drop rows or columns with missing data, and assess the impact of missing data on your analysis.

In [42]:
```
# filling missing values with a specific value
df2.fillna(0,inplace=True)
df2
```

Out[42]:

	name	age	grade	marks
0	a	15.0	A	23.0
1	b	17.0	A	24.0
2	c	19.0	A	22.0
3	d	0.0	B	0.0
4	e	24.0	0	27.0
5	f	10.0	0	0.0
6	g	16.0	B	0.0

In [43]:

```
#droping rows with missing data
df2.dropna(inplace=True)
df2
```

Out[43]:

	name	age	grade	marks
0	a	15.0	A	23.0
1	b	17.0	A	24.0
2	c	19.0	A	22.0
3	d	0.0	B	0.0
4	e	24.0	0	27.0
5	f	10.0	0	0.0
6	g	16.0	B	0.0

transforming data

Transforming data involves cleaning and preparing it for analysis. In this section, you'll explore techniques to remove duplicate entries, convert data types, and make other adjustments to ensure your data is in the correct format. These preprocessing steps are vital for accurate and meaningful analysis.

In [47]:

```
# Removing duplicates
df2.drop_duplicates(inplace=True)
df2
```

Out[47]:

	name	age	grade	marks
0	a	15.0	A	23.0
1	b	17.0	A	24.0
2	c	19.0	A	22.0
3	d	0.0	B	0.0
4	e	24.0	0	27.0
5	f	10.0	0	0.0
6	g	16.0	B	0.0

In [50]:

```
#converting datatypes
df2['age']=df2['age'].astype(float)
df2
```

Out[50]:

	name	age	grade	marks
0	a	15.0	A	23.0
1	b	17.0	A	24.0
2	c	19.0	A	22.0
3	d	0.0	B	0.0
4	e	24.0	0	27.0
5	f	10.0	0	0.0
6	g	16.0	B	0.0

Data analysis with pandas

This part of the program covers the data analysis capabilities of Pandas. You will learn how to generate summary statistics, group data by specific columns, and apply aggregate functions to gain insights from your data. Additionally, you will explore advanced data manipulation techniques such as merging, joining, and concatenating DataFrames, which are essential for combining and restructuring datasets. These skills are key for conducting detailed analyses and extracting meaningful information from data.

Generating Summary Statistics

This introduces methods for summarizing your data to understand its main characteristics quickly. You'll learn how to generate descriptive statistics, such as mean, median, and standard deviation, using Pandas functions. These summaries are essential for gaining initial insights into your dataset.

In [53]:
```
# summary statistics
summary = df2.describe()
summary
```

Out[53]:

	age	marks
count	7.000000	7.000000
mean	14.428571	13.714286
std	7.634508	12.919163
min	0.000000	0.000000
25%	12.500000	0.000000
50%	16.000000	22.000000
75%	18.000000	23.500000
max	24.000000	27.000000

Grouping Data and Applying Aggregate Functions

it involves how to group data by specific categories and apply aggregate functions like mean(), sum(), and count() to these groups. Grouping data is a powerful way to analyze subsets of your data and uncover patterns or trends that may not be visible in the overall dataset.

In [60]:
```
grouped_data = df2.groupby('grade')['age'].mean()
grouped_data
```

Out[60]:
```
grade
0    17.0
A    17.0
B     8.0
Name: age, dtype: float64
```

Advanced Data Manipulation Techniques

these techniques dive into more complex data manipulation tasks, such as merging, joining, and concatenating multiple DataFrames. These techniques are crucial when working with large datasets that need to be combined or restructured, enabling you to perform more comprehensive analyses.

In [62]:
```
#merging dataframe
df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})
df2 = pd.DataFrame({'ID': [1, 2, 4], 'Age': [25, 30, 40]})
merged_df = pd.merge(df1, df2, on='ID', how='inner')
merged_df
```

Out[62]:

	ID	Name	Age
0	1	Alice	25
1	2	Bob	30

In [67]:
```
#Concatenating DataFrames
concatenated_df = pd.concat([df1, df2], axis=0)
concatenated_df
```

Out[67]:

	ID	Name	Age
0	1	Alice	NaN
1	2	Bob	NaN
2	3	Charlie	NaN
0	1	NaN	25.0
1	2	NaN	30.0
2	4	NaN	40.0

Application in Data Science

Advantages of Pandas:

Efficiency: Pandas are optimized for fast data manipulation and analysis, unlike traditional Python data structures such as lists and dictionaries.

Functionality: It provides powerful tools for data handling, cleaning, and analysis, making it an essential library for data science professionals.

Ease of Use: Pandas have a user-friendly API that simplifies complex data operations.

Real-World Examples:

Data Cleaning: Removing duplicates, handling missing values, and converting data types are essential steps in preparing data for analysis.

Exploratory Data Analysis (EDA): Generating summary statistics, visualizing distributions, and understanding relationships between variables are common EDA tasks where Pandas play a crucial role.

In [ ]:

```
```