NumPy & Pandas

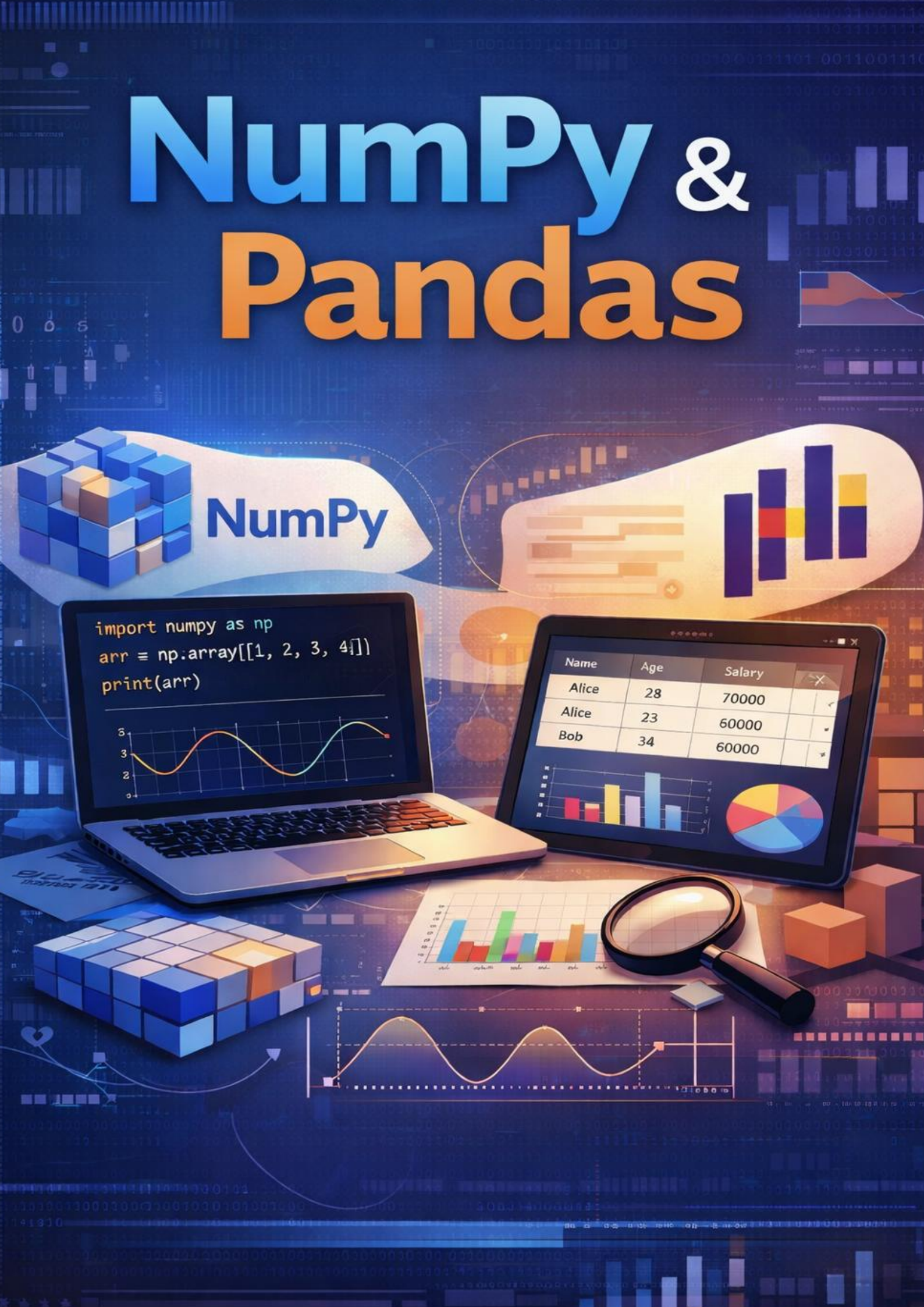

Table of Contents

Introduction

Purpose

The purpose of this document is to provide a detailed guide to two of the most essential Python libraries for data science: **NumPy** and **Pandas**. It aims to equip learners and professionals with the knowledge required to handle data efficiently and perform advanced analysis.

Scope

This document covers NumPy arrays, random number generation, array manipulations, mathematical operations, indexing, slicing, and linear algebra, as well as Pandas Series, DataFrames, data selection, merging, grouping, reshaping, time handling, and utility functions.

Procedure

The material was collected and refined through practical coding, courses, and structured note-taking over two years. Code examples, tables, and outputs are included to illustrate each concept clearly.

Problem

Data manipulation and analysis can be challenging without efficient tools. Beginners often rely on Python's basic lists and dictionaries, which are limited for large-scale operations. This document resolves that gap by teaching NumPy and Pandas systematically, enabling users to manage, clean, and analyze data effectively.

Numpy

- **NumPy = Numerical Python, the core library for numerical computing in Python.**
- **Provides N-dimensional arrays (ndarray), efficient operations, mathematical/statistical functions, linear algebra, random number generation, etc.**
- **Much faster than lists because arrays are contiguous in memory and operations are vectorized.**

Importing Library

```
import numpy as np
```

Array Creation

Array

- **Create an array from a Python list, tuple, or iterable.**

Syntax

```
np.array(object, dtype=None, copy=True, ndmin=0)
```

- **object → list, tuple, or nested list**
- **dtype → specify type (int32, float64, etc.)**
- **copy → force copy or not (default: True)**
- **ndmin → minimum number of dimensions**

Examples

```
np.array([1, 2, 3])            # [1 2 3]
np.array((1, 2, 3), dtype=float)  # [1. 2. 3.]
np.array([1, 2, 3], ndmin=2)      # [[1 2 3]]
```

Arange

- **Creates evenly spaced values in a range.**

Syntax

```
np.arange(start, stop, step, dtype=None)
```

- **start → starting value (default = 0)**
- **stop → end (exclusive)**
- **step → difference between numbers (default = 1)**
- **dtype → optional data type**

Examples

```
np.arange(5)        # [0 1 2 3 4]
np.arange(1, 10, 2)   # [1 3 5 7 9]
np.arange(0, 1, 0.2)  # [0.  0.2  0.4  0.6  0.8]
```

Linespace

- **Creates evenly spaced numbers between two points.**

Syntax

```
np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
```

- **start → start value**
- **stop → end value**
- **num → number of samples (default = 50)**
- **endpoint → include stop (default = True)**
- **retstep → if True, return step size as well**
- **dtype → data type**

Examples

```
np.linspace(0, 1, 5)            # [0.   0.25 0.5  0.75 1.  ]

np.linspace(0, 10, 4, endpoint=False)  # [0. 2.5 5. 7.5]

vals, step = np.linspace(0, 5, 6, retstep=True)
print(vals)  # [0. 1. 2. 3. 4. 5.]
print(step)  # 1.0
```

Zeros

- **Creates an array filled with zeros.**

Syntax

```
np.zeros(shape, dtype=float)
```

- **shape → int or tuple of dimensions**
- **dtype → data type (default float)**

Examples

```
np.zeros(3)            # [0. 0. 0.]
np.zeros((2, 3), int)    # [[0 0 0]
                         #  [0 0 0]]
```

Ones

- **Creates an array filled with ones.**

Syntax

```
np.ones(shape, dtype=float)
```

- **shape → int or tuple of dimensions**
- **dtype → data type (default float)**

Examples

```
np.ones(4)            # [1. 1. 1. 1.]
np.ones((2, 2), int)    # [[1 1]
                        #  [1 1]]
```

Full

- **Creates an array filled with a specified value.**

Syntax

```
np.full(shape, fill_value, dtype=None)
```

Examples

```
np.full((2, 3), 7)        # [[7 7 7]
                          #  [7 7 7]]
np.full((2, 2), 3.14)     # [[3.14 3.14]
                          #  [3.14 3.14]]
```

Eye & Identity

- **Create an identity matrix (diagonal ones).**

Syntax

```
np.eye(N, M=None, k=0, dtype=float)
np.identity(n, dtype=None)
```

- **N → rows**
- **M → columns (default = N)**
- **k → diagonal index (0 = main diagonal)**

Examples

```
np.eye(3)
# [[1. 0. 0.]
#  [0. 1. 0.]
#  [0. 0. 1.]]

np.eye(3, 4, k=1)
# [[0. 1. 0. 0.]
#  [0. 0. 1. 0.]
#  [0. 0. 0. 1.]]
```

Random Module

Random.rand

- **Returns random floats in the half-open interval [0, 1).**

Syntax

```
np.random.rand(d0, d1, ..., dn)
```

- **d0, d1, ... → dimensions of the output**

Examples

```
np.random.rand()       # 0.7383 (single float)
np.random.rand(3)      # [0.23 0.61 0.95]
np.random.rand(2, 3)   # 2x3 array of floats
```

Random.randIn

- **Returns samples from the standard normal distribution (mean = 0, std = 1).**

Syntax

```
np.random.randn(d0, d1, ..., dn)
```

- **d0, d1, ... → dimensions of the output**

Examples

```
np.random.randn()      # e.g. -0.564
np.random.randn(4)     # [ 0.12 -1.53  0.38  1.45]
np.random.randn(2, 2)  # 2x2 normal distribution
```

Random.randint

- **Returns random integers from low (inclusive) to high (exclusive).**

Syntax

```
np.random.randint(low, high=None, size=None, dtype=int)
```

- **low → lower bound (inclusive)**
- **high → upper bound (exclusive). If None, range is [0, low).**
- **size → shape of output**
- **dtype → integer type**

Examples

```
np.random.randint(10)          # single int 0–9
np.random.randint(5, 15)        # single int 5–14
np.random.randint(1, 10, size=5)  # array of 5 ints
np.random.randint(1, 10, (2, 3))  # 2x3 array of ints
```

Random.choice

- **Generates a random sample from a 1-D array.**

Syntax

```
np.random.choice(a, size=None, replace=True, p=None)
```

- **a → if int, values from [0, a); if array, sample from it**
- **size → number of samples**
- **replace → with or without replacement**
- **p → probability distribution (must sum to 1)**

Examples

```
np.random.choice(5, 3)        # e.g. [3 1 4]
np.random.choice([10, 20, 30], 2)  # sample from list
np.random.choice(5, 3, replace=False) # no repeats
np.random.choice(5, 5, p=[0.1,0.2,0.3,0.2,0.2]) # biased probabilities
```

Random.uniform

- **Draws samples from a uniform distribution over [low, high).**

Syntax

```
np.random.uniform(low=0.0, high=1.0, size=None)
```

Examples

```
np.random.uniform()            # single float in [0,1)
np.random.uniform(5, 10, 3)      # 3 values between 5–10
np.random.uniform(-1, 1, (2,2))   # 2x2 array
```

Random.normal

- **Draws random samples from a normal (Gaussian) distribution.**

Syntax

```
np.random.normal(loc=0.0, scale=1.0, size=None)
```

- **loc → mean**
- **scale → standard deviation**
- **size → shape**

Examples

```
np.random.normal(0, 1, 5)    # 5 samples from N(0,1)
np.random.normal(10, 2, (2,3))  # 2x3 array, mean=10, std=2
```

Random.seed

- **Sets the seed for reproducible random results.**

Syntax

```
np.random.seed(42)
print(np.random.randint(0, 10, 3))  # always [6 3 7]
```

Array Inspection and Attributes

Shape

- **Returns the dimensions of the array as a tuple (rows, cols, ...).**

Examples

```
arr = np.array([[1,2,3],[4,5,6]])
arr.shape   # (2, 3)
```

Ndim

- **Returns the number of dimensions (axes) of the array.**

Examples

```
np.array(5).ndim       # 0D
np.array([1,2,3]).ndim   # 1D
np.array([[1],[2]]).ndim # 2D
```

Size

- **Returns the total number of elements in the array.**

Examples

```
arr = np.array([[1,2,3],[4,5,6]])
arr.size   # 6
```

Dtype

- **Returns the data type of array elements.**

Examples

```
np.array([1,2,3]).dtype     # int64
np.array([1.0,2.0]).dtype   # float64
np.array(['a','b']).dtype   # <U1 (unicode string)
```

T

- **Returns the transpose of the array (swap rows ↔ columns).**

Examples

```
arr = np.array([[1,2,3],[4,5,6]])
arr.T
# [[1 4]
#  [2 5]
#  [3 6]]
```

Flat

- **Returns an iterator to access array elements as if it were 1D.**

Examples

```
arr = np.array([[1,2],[3,4]])
for x in arr.flat:
   print(x, end=" ")  # 1 2 3 4
```

Real and Imag

- **Returns the real and imaginary parts of complex numbers.**

Examples

```
arr = np.array([1+2j, 3+4j])
arr.real   # [1. 3.]
arr.imag   # [2. 4.]
```

Array Manipulation

Reshape

- **Changes the shape of an array without changing its data.**
- **Must match total number of elements.**

Examples

```
arr = np.arange(6)      # [0 1 2 3 4 5]
arr.reshape(2, 3)             # [[0 1 2]
                              #  [3 4 5]]

arr.reshape(-1, 2)            # Infer rows(auto): [[0 1]
                              #                    [2 3]
                              #                    [4 5]]
```

Ravel

- **Returns a flattened 1D array (view when possible).**

Examples

```
arr = np.array([[1,2],[3,4]])
arr.ravel()   # [1 2 3 4]
```

Flatten

- **Returns a copy of the array as 1D (unlike ravel, always copy).**

Examples

```
arr.flatten()  # [1 2 3 4]
```

Resize

- **Resizes the array in-place. Can repeat or truncate data.**

Examples

```
arr = np.array([1,2,3])
arr.resize((2,4))
# [[1 2 3 1]
#  [2 3 1 2]]
```

Stack

- **Joins arrays along a new axis.**

Examples

```
a = np.array([1,2])
b = np.array([3,4])
np.stack((a,b))         # [[1 2]
                        #  [3 4]]
np.stack((a,b), axis=1)  # [[1 3]
                         #  [2 4]]
```

Hstack & Vstack

- **Stack arrays horizontally or vertically.**

```
a = np.array([1,2])
b = np.array([3,4])
np.hstack((a,b))   # [1 2 3 4]
np.vstack((a,b))     # [[1 2]
                     #  [3 4]]
```

Concatenate

- **Joins arrays along an existing axis.**

Examples

```
a = np.array([[1,2]])
b = np.array([[3,4]])
np.concatenate((a,b), axis=0)  # [[1 2]
                               #  [3 4]]

np.concatenate((a,b), axis=1)   [1 2 3 4]
```

Split

- **Splits array into multiple sub-arrays.**

Examples

```
arr = np.arange(6)
np.split(arr, 3)   # [array([0,1]), array([2,3]), array([4,5])]
np.split(arr, [2,4])  # [array([0,1]), array([2,3]), array([4,5])],          2,4 are the indices at which split occurs
```

Hsplit & Vsplit

- **Splits array horizontally or vertically.**

Examples

```
arr = np.arange(16).reshape(4,4)
np.hsplit(arr, 2)   # splits into 2 arrays column-wise
np.vsplit(arr, 2)   # splits into 2 arrays row-wise
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]


[array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]]),

 array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15]])]


[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]),

array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])]
```

Repeat

- **Repeats elements of an array.**

Examples

```
arr = np.array([1,2,3])
arr.repeat(2)     # [1 1 2 2 3 3]
arr.repeat([1,2,3]) # [1 2 2 3 3 3], 1 is repeated once, 2 repeated twice, 3 repeated 3 times based on array
```

Tile

- **Constructs an array by repeating an array in a grid pattern.**

Examples

```
arr = np.array([1,2])
np.tile(arr, 3)    # [1 2 1 2 1 2]
np.tile(arr, (2,2)) # [[1 2 1 2]
                    #  [1 2 1 2]]
```

Mathematical and Statistical Functions

Function	Description	Example
np.sum(arr)	Sum of all elements (or along axis).	np.sum([1,2,3]) → 6 np.sum([[1,2],[3,4]], axis=0) → [4 6]
np.prod(arr)	Product of all elements.	np.prod([1,2,3]) → 6
np.mean(arr)	Arithmetic mean.	np.mean([1,2,3]) → 2.0
np.median(arr)	Median value.	np.median([1,3,5,7]) → 4.0
np.average(arr, weights=...)	Weighted average.	np.average([1,2,3], weights=[1,1,2]) → 2.25
np.std(arr)	Standard deviation.	np.std([1,2,3]) → 0.816
np.var(arr)	Variance.	np.var([1,2,3]) → 0.667
np.min(arr) / np.max(arr)	Minimum / maximum value.	np.min([2,5,1]) → 1np.max([2,5,1]) → 5
np.argmin(arr) / np.argmax(arr)	Index of min / max value.	np.argmax([2,8,5]) → 1
np.ptp(arr)	Peak-to-peak (max - min).	np.ptp([1,5,9]) → 8
np.cumsum(arr)	Cumulative sum.	np.cumsum([1,2,3]) → [1 3 6]
np.cumprod(arr)	Cumulative product.	np.cumprod([1,2,3]) → [1 2 6]
np.cummax(arr)	Cumulative maximum (NumPy ≥1.20).	np.cummax([2,1,5,3]) → [2 2 5 5]
np.cummin(arr)	Cumulative minimum (NumPy ≥1.20).	np.cummin([2,1,5,3]) → [2 1 1 1]
np.dot(a, b) / @	Dot product of two arrays.	np.dot([1,2],[3,4]) → 11
np.matmul(a, b) / @	Matrix multiplication.	np.array([[1,2],[3,4]]) @ np.array([1,0]) → [1 3]
np.cross(a, b)	Cross product (for 3D vectors).	np.cross([1,0,0],[0,1,0]) → [0 0 1]
np.corrcoef(x,y)	Correlation coefficient matrix.	np.corrcoef([1,2,3],[1,5,7])
np.cov(x,y)	Covariance matrix.	np.cov([1,2,3],[1,5,7])

Operators and Comparisons

Array Arithmetic Operators

Operator	Description	Example
+	Element-wise addition	np.array([1,2]) + np.array([3,4]) → [4 6]
-	Element-wise subtraction	np.array([5,6]) - np.array([2,3]) → [3 3]
*	Element-wise multiplication	np.array([2,3]) * np.array([4,5]) → [8 15]
/	Element-wise division	np.array([6,8]) / np.array([2,4]) → [3. 2.]
//	Floor division	np.array([7,8]) // np.array([2,3]) → [3 2]
%	Modulus	np.array([7,8]) % np.array([2,3]) → [1 2]
**	Power (exponentiation)	np.array([2,3]) ** 2 → [4 9]

Boolean Comparison

Operator	Description	Example
>	Greater than	np.array([1,2,3]) > 2 → [False False True]
<	Less than	np.array([1,2,3]) < 2 → [True False False]
>=	Greater than or equal	np.array([2,3]) >= 2 → [True True]
<=	Less than or equal	np.array([2,3]) <= 2 → [True False]
==	Equality	np.array([1,2,3]) == 2 → [False True False]
!=	Inequality	np.array([1,2,3]) != 2 → [True False True]

Boolean Array Functions

Function	Description	Example
np.any(arr)	Checks if any element is True	np.any([False,True,False]) → True
np.all(arr)	Checks if all elements are True	np.all([True,True]) → True
np.where(cond, x, y)	Return elements chosen from x or y based on condition	np.where([True,False],[1,2],[3,4]) → [1 4]
np.nonzero(arr)	Returns indices of non-zero (or True) elements	np.nonzero([0,2,0,3]) → (array([1,3]),)
np.isin(arr, values)	Checks membership	np.isin([1,2,3], [2,4]) → [False True False]

Linear Algebra

Function	Description	Example
np.dot(a, b)	Dot product	np.dot([1,2],[3,4]) → 11
np.matmul(a, b) / @	Matrix multiplication	np.array([[1,2],[3,4]]) @ np.array([[1],[0]]) → [[1],[3]]
np.inner(a, b)	Inner product	np.inner([1,2,3],[0,1,0]) → 2
np.outer(a, b)	Outer product	np.outer([1,2],[3,4]) → [[3 4],[6 8]]
np.linalg.inv(A)	Matrix inverse	np.linalg.inv([[1,2],[3,4]])
np.linalg.det(A)	Determinant	np.linalg.det([[1,2],[3,4]]) → -2.0
np.linalg.eig(A)	Eigenvalues & eigenvectors	vals, vecs = np.linalg.eig([[1,2],[2,1]])
np.linalg.svd(A)	Singular Value Decomposition	U, S, V = np.linalg.svd([[1,2],[3,4]])
np.linalg.norm(A)	Vector/matrix norm (length, magnitude)	np.linalg.norm([3,4]) → 5.0
np.trace(A)	Sum of diagonal elements	np.trace([[1,2],[3,4]]) → 5
np.linalg.solve(A, b)	Solve linear system Ax=b	np.linalg.solve([[3,1],[1,2]], [9,8]) → [2. 3.]

Indexing and Slicing

Slicing

```
arr = np.array([10,20,30,40,50])
arr[1:4]      # [20 30 40]
arr[:3]       # [10 20 30]
arr[::2]      # [10 30 50]  (step = 2)

arr2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
arr2d[0, 1]    # 2  (row 0, col 1)
arr2d[1:, :2]  # [[4 5] [7 8]] (rows 1→end, first 2 cols)
```

Indexing

```
arr = np.array([1,2,3,4,5])
arr[-1]    # 5
arr[-3:]   # [3 4 5]

arr = np.array([10,20,30,40,50])
arr[[0,2,4]]      # [10 30 50]

arr2d = np.array([[10,20],[30,40],[50,60]])
arr2d[[0,2],[1,0]]  # [20 50] (positions (0,1) and (2,0))

arr = np.array([1,2,3,4,5])
mask = arr > 3
arr[mask]   # [4 5]   Boolean Masking

arr[arr % 2 == 0]   # [2 4] (all even numbers)

arr = np.array([10,20,30,40])
np.where(arr > 25)     # (array([2, 3]),)
arr[np.where(arr > 25)]  # [30 40]

arr = np.arange(20).reshape(4,5)
arr[1:3, [0,2,4]]  # rows 1–2, cols 0,2,4
# [[ 5  7  9]
#  [10 12 14]]
```

Miscellaneous

Function / Concept	Description	Example
0D array (scalar)	A single value.	np.array(42) → 0D
1D array (vector)	A list of values.	np.array([1,2,3]) → 1D
2D array (matrix)	Rows and columns.	np.array([[1,2,3],[4,5,6]]) → 2D
3D array	Multiple matrices stacked.	np.array([[[1,2],[3,4]], [[5,6],[7,8]]]) → 3D
nD array (tensor)	General n-dimensional array.	np.array(range(16)).reshape(2,2,2,2) → 4D tensor
np.copy(arr)	Creates a deep copy (not linked to original).	b = np.copy(a)
arr.view()	Creates a shallow copy (linked view, changes reflect).	b = a.view()
arr.astype(dtype)	Converts array to a different data type.	arr.astype(float)
np.empty(shape)	Creates an uninitialized array (random values in memory).	np.empty((2,3))
np.append(arr, values, axis=None)	Appends values to end of array.	np.append([1,2,3], [4,5]) → [1 2 3 4 5]
np.insert(arr, index, values, axis=None)	Inserts values at given index.	np.insert([1,2,3], 1, 99) → [1 99 2 3]
np.delete(arr, index, axis=None)	Deletes elements by index.	np.delete([1,2,3], 1) → [1 3]
np.unique(arr)	Returns sorted unique elements.	np.unique([1,2,2,3]) → [1 2 3]
np.sort(arr)	Sorts array (returns sorted copy).	np.sort([3,1,2]) → [1 2 3]
np.flip(arr, axis)	Reverses elements.	np.flip([1,2,3]) → [3 2 1]
np.roll(arr, shift)	Rolls elements along axis.	np.roll([1,2,3,4], 2) → [3 4 1 2]
np.random.shuffle(arr)	Shuffles array **in place**.	np.random.shuffle(arr)
np.random.permutation(arr)	Returns shuffled copy.	np.random.permutation([1,2,3])

Function / Concept	Description	Example
np.round(arr, decimals=0)	Rounds to nearest integer (or decimals).	np.round([1.23, 1.78],1) → [1.2 1.8]
np.floor(arr)	Rounds down (toward -∞).	np.floor([1.9, -1.1]) → [1. -2.]
np.ceil(arr)	Rounds up (toward +∞).	np.ceil([1.1, -1.9]) → [2. -1.]
np.trunc(arr)	Truncates fractional part.	np.trunc([1.9, -1.9]) → [1. -1.]
np.clip(arr, min, max)	Limits values to given range.	np.clip([1,5,10], 2, 8) → [2 5 8]
np.isnan(arr)	Detects NaN values.	np.isnan([1, np.nan, 2]) → [False True False]
np.isfinite(arr)	Detects finite numbers.	np.isfinite([1, np.nan, np.inf]) → [True False False]
np.save('file.npy', arr)	Saves array to a binary .npy file (efficient storage of a single array).	np.save('myarray.npy', arr)
np.load('file.npy')	Loads array from .npy file.	arr = np.load('myarray.npy')
np.savez('file.npz', a=arr1, b=arr2)	Saves multiple arrays into a compressed .npz file (dict-like storage).	np.savez('arrays.npz', x=arr1, y=arr2)
np.savez_compressed('file.npz', a=arr1, b=arr2)	Same as above but compressed to save disk space.	np.savez_compressed('arrays.npz', x=arr1, y=arr2)
np.load('file.npz')	Loads arrays from .npz file into a dict-like object.	data = np.load('arrays.npz'); data['x']
np.savetxt('file.txt', arr, delimiter=',')	Saves array as text/CSV file.	np.savetxt('array.csv', arr, delimiter=',')
np.loadtxt('file.txt', delimiter=',')	Loads array from text/CSV file.	arr = np.loadtxt('array.csv', delimiter=',')

- **Axis =1 -> row**
- **Axis =0 -> column**
- **np.array([]) -> Empty numpy array**

Pandas

Pandas is a Python library built on top of NumPy for **data manipulation and analysis**.
It provides two main data structures:

- **Series** → 1D labeled array (like a column in Excel).
- **DataFrame** → 2D labeled data structure (like a table in Excel/SQL).

Pandas is widely used in data science, machine learning, and analytics for tasks like data cleaning, transformation, and visualization.

Importing Pandas

```
import pandas as pd
```

Pandas Core Data Structures

Object	Description	Example
Series	One-dimensional labeled array that can hold data of any type (int, float, string, etc.).	pd.Series([10,20,30])
DataFrame	Two-dimensional table with labeled axes (rows & columns).	pd.DataFrame({'Name':['Ali','Sara'],'Age':[23,25]})
Index	Immutable sequence that labels the axes of Series/DataFrame.	df.index

Creating and Saving Data in Pandas

Method	Description	Example
pd.Series(list)	Create a Series from list/array.	pd.Series([1,2,3])
pd.Series(dict)	Create a Series from dict.	pd.Series({'a':10,'b':20})
pd.DataFrame(dict)	Create a DataFrame from dict of lists.	pd.DataFrame({'A':[1,2], 'B':[3,4]})
pd.DataFrame(list of dicts)	Create DataFrame from list of dictionaries.	pd.DataFrame([{'a':1,'b':2},{'a':3,'b':4}])
pd.read_csv('file.csv')	Load data from CSV file.	pd.read_csv('data.csv')
pd.read_excel('file.xlsx')	Load data from Excel file.	pd.read_excel('data.xlsx')
pd.read_sql(query, conn)	Load data from SQL database.	pd.read_sql('SELECT * FROM table', conn)
df.to_csv('file.csv')	Save DataFrame to CSV file.	df.to_csv('data.csv', index=False)
df.to_excel('file.xlsx')	Save DataFrame to Excel file.	df.to_excel('data.xlsx', index=False)
df.to_sql('table', conn)	Save DataFrame to SQL table (requires SQLAlchemy).	df.to_sql('mytable', conn, if_exists='replace')

- **We can include index parameter in saving data to save row numbers (index=True)**

Creating Series

```
# From a list
s1 = pd.Series([10, 20, 30])
print(s1)
# 0   10
# 1   20
# 2   30

# From list with custom index
s2 = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(s2)
# a   10
# b   20
# c   30

# From dictionary (keys become index)
s3 = pd.Series({'a': 10, 'b': 20, 'c': 30})
print(s3)
# a   10
# b   20
# c   30

# From NumPy array
s4 = pd.Series(np.array([1, 2, 3]))
print(s4)
# 0   1
# 1   2
# 2   3

# From scalar (broadcasted to all index values)
s5 = pd.Series(5, index=['x', 'y', 'z'])
print(s5)
# x   5
# y   5
# z   5

# With explicit dtype
s6 = pd.Series([1, 2, 3], dtype="float32")
print(s6)
# 0   1.0
# 1   2.0
# 2   3.0
# dtype: float32
```

Creating DataFrame

```
# From dictionary of lists
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print(df1)
#    A  B
# 0  1  3
# 1  2  4

# From list of dictionaries
df2 = pd.DataFrame([{'A': 1, 'B': 2}, {'A': 3, 'B': 4}])
print(df2)
#    A  B
# 0  1  2
# 1  3  4

# From list of lists with column names
df3 = pd.DataFrame([[1, 2, 3], [4, 5, 6]], columns=['A', 'B', 'C'])
print(df3)
#    A  B  C
# 0  1  2  3
# 1  4  5  6

# From NumPy array with row & column labels
df4 = pd.DataFrame(
   np.array([[1, 2], [3, 4]]),
   columns=['A', 'B'],
   index=['row1', 'row2']
)
print(df4)
#         A  B
# row1  1  2
# row2  3  4

# From random NumPy array
df5 = pd.DataFrame(np.random.randn(3, 3), columns=list('ABC'))
print(df5)
#          A           B           C
# 0 -0.123456  0.987654 -1.234567
# 1  0.345678 -0.456789  0.567890
# 2  1.234567  0.345678 -0.987654

# Mixing Series and lists
df6 = pd.DataFrame({
   'A': pd.Series([1, 2, 3], index=['x', 'y', 'z']),
   'B': [4, 5, 6]
})
print(df6)
#     A   B
# x  1.0  4
```

```
# y  2.0  5
# z  3.0  6

df7 = pd.concat((df1,df2),ignore_index=True) #  Concatenating two dataframes together, ignore_index
#fixes index issue
```

MultiIndex

```
# Datetime index (useful for time series)
dates = pd.date_range('2025-01-01', periods=3, freq='D')
df8 = pd.DataFrame({'val': [10, 20, 30]}, index=dates)
print(df8)
#             val
# 2025-01-01   10
# 2025-01-02   20
# 2025-01-03   30

# MultiIndex from tuples
multi_idx = pd.MultiIndex.from_tuples([('A', 1), ('A', 2), ('B', 1)])
df9 = pd.DataFrame({'val': [100, 200, 300]}, index=multi_idx)
print(df9)
#          val
# A 1   100
# A 2   200
# B 1   300

# MultiIndex from product
multi_idx2 = pd.MultiIndex.from_product([['A', 'B'], [1, 2]])
df10 = pd.DataFrame(np.random.randn(4, 2), index=multi_idx2, columns=['val1', 'val2'])
print(df10)
#            val1      val2
# A 1 -0.12345  0.23456
# A 2  0.34567 -1.23456
# B 1  1.23456  0.98765
# B 2 -0.45678  0.12345
```

DataFrame Manipulation and Reshaping

Selection and Slicing

- **Used to access rows, columns, or subsets of the DataFrame.**

```
df = pd.DataFrame({
   'A': range(1, 6),
   'B': list('abcde'),
   'C': np.random.randint(10, 20, 5)
}, index=list('VWXYZ'))
print(df)
#     A  B   C
# V  1  a  11
# W 2  b  16
# X  3  c  14
# Y  4  d  19
# Z  5  e  12

# Column selection
print(df['A'])     # Single column
print(df[['A', 'C']])  # Multiple columns

# Row selection by label
print(df.loc['X'])
# A    3
# B    c
# C   14

# Row selection by integer location
print(df.iloc[2])   # Same as above

# Row/Column slice
print(df.loc[['W':'Y'],['A']])   # Inclusive of end
print(df.iloc[1:4])     # Exclusive of end
print(df.iloc[2:4,3:5])     # 3rd and 4th row with 4th and 5th column

# Conditional selection
print(df[df['A'] > 2])
print(df[(df['A'] > 2) & (df['C'] < 18)])
```

Inserting, Popping and Dropping

- **Modify DataFrame by adding or removing rows/columns.**

```
# Insert a new column
df.insert(1, 'NewCol', [100, 200, 300, 400, 500])
print(df)

# Pop removes and returns a column
val = df.pop('NewCol')
print(val)

# Drop column(s)
df2 = df.drop(columns=['B'])
print(df2)

# Drop row(s)
df3 = df.drop(index=['X', 'Y'])
print(df3)
```

```
   A  NewCol  B   C
V  1     100  a  19
W  2     200  b  14
X  3     300  c  11
Y  4     400  d  18
Z  5     500  e  10


V    100
W    200
X    300
Y    400
Z    500


   A   C
V  1  19
W  2  14
X  3  11
Y  4  18
Z  5  10


   A  B  C
V  1  a  19
W  2  b  14
Z  5  e  10
```

Sorting and Renaming

- **Change order of rows/columns or rename them.**

```
# Sort by column values
print(df.sort_values('C', ascending=False))
df.sort_values(['A','B'], ascending=False) # Sort by A then B

# Sort by index
print(df.sort_index(ascending=False))

# Rename columns / index
print(df.rename(columns={'A': 'Alpha'}, index={'V': 'Row1'}))

# Reset index (from start till length of df -1)
print(df.reset_index(drop=True)) # drop=True -> drops multi-indices
```

```
   A  B   C
W  2  b  19
V  1  a  17
X  3  c  16
Y  4  d  16
Z  5  e  10


   A  B   C
Z  5  e  10
Y  4  d  16
X  3  c  16
W  2  b  19
V  1  a  17


      Alpha  B   C
Row1      1  a  17
W         2  b  19
X         3  c  16
Y         4  d  16
Z         5  e  10


   A  B   C
0  1  a  17
1  2  b  19
2  3  c  16
3  4  d  16
4  5  e  10
```

Sampling and Filtering

- **Select random or conditional subsets.**

```
# Random row sample
print(df.sample(2)) # 2 is no of rows

# Random fraction sample
print(df.sample(frac=0.4)) # 40% of df

# Filter with query
print(df.query("A > 3 and C < 18"))

# Filter with regex
print(df.filter(regex='regex pattern'))
```

```
   A  B   C
W  2  b  18
V  1  a  10

   A  B   C
V  1  a  10
Y  4  d  14

   A  B   C
Y  4  d  14
Z  5  e  17
```

Grouping and Aggregation

- **Group rows by key(s) and compute summary statistics.**

```
dfg = pd.DataFrame({
   'Team': ['A', 'A', 'B', 'B', 'C'],
   'Score': [10, 20, 30, 40, 50],
   'Wins': [1, 1, 2, 3, 5]
})
print(dfg)

# Group by single column
print(dfg.groupby('Team')['Score'].mean())

# Multiple aggregations
print(dfg.groupby('Team').agg({'Score': ['mean', 'sum'], 'Wins': 'max'}))

# Using lambda
print(dfg.groupby('Team')['Score'].agg(lambda x: x.max() - x.min()))
```

```
   Team  Score  Wins
0     A     10     1
1     A     20     1
2     B     30     2
3     B     40     3
4     C     50     5


Team
A    15.0
B    35.0
C    50.0


     Score      Wins
      mean sum  max
Team
A     15.0  30    1
B     35.0  70    3
C     50.0  50    5


Team
A    10
B    10
C     0
```

Merging and Joining

- **Combine multiple DataFrames (like SQL joins).**

```
left = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})
right = pd.DataFrame({'ID': [2, 3, 4], 'Score': [90, 80, 70]})

# Merge (like SQL join)
print(pd.merge(left, right, on='ID', how='inner'))
print(pd.merge(left, right, on='ID', how='outer'))
print(pd.merge(left, right, on='ID', how='left'))
print(pd.merge(left, right, on='ID', how='right'))

# Join by index
df1 = pd.DataFrame({'A': [1, 2]}, index=['x', 'y'])
df2 = pd.DataFrame({'B': [3, 4]}, index=['x', 'z'])
print(df1.join(df2, how='outer'))
```

```
   ID      Name  Score
0   2       Bob     90
1   3   Charlie     80


   ID      Name  Score
0   1     Alice    NaN
1   2       Bob   90.0
2   3   Charlie   80.0
3   4       NaN   70.0


   ID      Name  Score
0   1     Alice    NaN
1   2       Bob   90.0
2   3   Charlie   80.0


   ID      Name  Score
0   2       Bob     90
1   3   Charlie     80
2   4       NaN     70


     A    B
x  1.0  3.0
y  2.0  NaN
z  NaN  4.0
```

Melting and Pivoting

- **Pivoting reshapes *long* → *wide* (turns unique values into columns).**
- **Melting reshapes *wide* → *long* (gathers columns into rows).**

```
dfp = pd.DataFrame({
   'Date': ['2025-01','2025-01','2025-02','2025-02'],
   'City': ['NY','LA','NY','LA'],
   'Sales': [200,300,250,400]
})

# Pivot (rows → columns)
dfp.pivot(index='Date', columns='City', values='Sales')

# Melt (columns → rows)
dfm = pd.DataFrame({'ID':[1,2], 'Math':[90,80], 'Sci':[85,95]})
dfm.melt(id_vars='ID', var_name='Subject', value_name='Score')
```

```
     Date   City  Sales
0  2025-01   NY     200
1  2025-01   LA     300
2  2025-02   NY     250
3  2025-02   LA     400
```

```
City       LA   NY
Date
2025-01   300  200
2025-02   400  250
```

```
    ID  Math  Sci

0   1    90   85

1   2    80   95
```

```
   ID Subject  Score
0   1    Math     90
1   2    Math     80
2   1     Sci     85
3   2     Sci     95
```

Crosstab

- **Creates a frequency table (like pivot, but for counts).**

```
dfp = pd.DataFrame({
   'Date': ['2025-01','2025-01','2025-02','2025-02'],
   'City': ['NY','LA','NY','LA'],
   'Sales': [200,300,250,400]
})

pd.crosstab(dfp['Date'], dfp['City'])
```

```
      Date City  Sales
0  2025-01   NY    200
1  2025-01   LA    300
2  2025-02   NY    250
3  2025-02   LA    400


City      LA  NY
Date
2025-01    1   1
2025-02    1   1
```

Date and Time

- **Pandas' date and time handling is one of its most powerful features, especially for time-series analysis.**

Creating Datetime Objects

- **Convert strings, lists, or numbers into datetime format.**

```
# From string
pd.to_datetime("2025-09-13")
# Timestamp('2025-09-13 00:00:00')

# From list of strings
pd.to_datetime(["2025-01-01", "2025-02-01", "2025-03-01"])
# DatetimeIndex(['2025-01-01', '2025-02-01', '2025-03-01'], dtype='datetime64[ns]', freq=None)

# From integers (YYYYMMDD)
pd.to_datetime([20250101, 20250201], format="%Y%m%d")

# With errors='coerce' (invalid → NaT)
pd.to_datetime(["2025-01-01", "not-a-date"], errors='coerce')

# DataFrame Date Column with a certain format
df['Date'] = pd.to_datetime(df['Date'],format='%Y-%m-%d')
```

Data Ranges

- **Generate sequences of datetime values.**

```
# Daily range
pd.date_range("2025-01-01", periods=5, freq="D")
# DatetimeIndex(['2025-01-01', ..., '2025-01-05'], dtype='datetime64[ns]', freq='D')

# Monthly range
pd.date_range("2025-01-01", periods=5, freq="M")
# DatetimeIndex(['2025-01-31', ..., '2025-05-31'], dtype='datetime64[ns]', freq='M')

# Custom frequency (every 2 hours)
pd.date_range("2025-01-01", periods=6, freq="2H")
```

Common frequency codes:

- **"D" = daily**
- **"M" = month end**
- **"MS" = month start**
- **"W" = weekly**
- **"H" = hourly**
- **"T" or "min" = minutes**
- **"S" = seconds**

Extracting Date and Time Components

- **Access year, month, day, weekday, etc.**

```
dt = pd.to_datetime("2025-09-13 15:30:45")

dt.year       # 2025
dt.month      # 9
dt.day        # 13
dt.hour       # 15
dt.minute     # 30
dt.second     # 45
dt.weekday()  # 5 (Saturday, 0=Monday)
dt.day_name()  # 'Saturday'
```

Working with Datetime in DataFrames

- **Store and manipulate datetime columns.**

```
df = pd.DataFrame({
   "Date": pd.date_range("2025-01-01", periods=5, freq="D"),
   "Value": np.random.randint(10, 100, 5)
})
print(df)

# Extract parts
df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
df["Weekday"] = df["Date"].dt.day_name()
print(df)
```

```
        Date  Value
0 2025-01-01     98
1 2025-01-02     99
2 2025-01-03     83
3 2025-01-04     53
4 2025-01-05     29


        Date  Value  Year  Month    Weekday
0 2025-01-01     98  2025      1  Wednesday
1 2025-01-02     99  2025      1   Thursday
2 2025-01-03     83  2025      1     Friday
3 2025-01-04     53  2025      1   Saturday
4 2025-01-05     29  2025      1     Sunday
```

Offsetting

- **Move datetime values forward/backward.**

```
df["Next Day"] = df["Date"] + pd.Timedelta(days=1)
df["Prev Hour"] = df["Date"] - pd.Timedelta(hours=1)
```

Functions

General Information

Function	Description	Example
df.head(n)	Returns the first n rows of the DataFrame (default 5).	df.head(3)
df.tail(n)	Returns the last n rows of the DataFrame.	df.tail(2)
df.info()	Shows concise summary: columns, dtypes, memory usage, total no . of rows and columns, number of non-null values for each column	df.info()
df.shape	Returns tuple (rows, columns).	df.shape → (100, 5)
df.dtypes	Returns data types of each column.	df.dtypes
df.describe()	Statistical summary of numeric columns (count, mean, std, etc.).	df.describe()
df.describe(include='all')	Statistical summary of all columns (count, mean, std, etc.).	df.describe(include='all')
df.columns	Lists column names in df	df.columns
df.index	Lists row labels of df	df.index

Handling missing Data

Function	Description	Example
df.isnull()	Returns boolean DataFrame: True where values are null.	df.isnull().sum()
df.notnull()	Opposite of isnull().	df[df["col"].notnull()]
df.dropna()	Drops rows (default) or columns with missing values.	df.dropna(subset=['Age'])
df.isna()	Returns DataFrame with True for missing values and False otherwise	df.isna().sum()/len(df) * 100
df.fillna(value)	Replaces NaN with given value or strategy.	df.fillna(0)
df.interpolate()	Fills missing values using interpolation.	df.interpolate()

Column and Row Operations

Function	Description	Example
df.drop(labels, axis)	Drops row(s) or column(s).	df.drop("col", axis=1)
df.rename(columns={})	Renames columns or index.	df.rename(columns={"old":"new"})
df.insert(loc, col, value)	Inserts column at given position.	df.insert(1,"new_col",val)
df.pop(col)	Removes and returns a column.	s = df.pop("col")
df.assign(newcol=...)	Adds new column without modifying original.	df.assign(z=df.x+df.y)
df.apply(func)	Applies function to each column/row.	df.apply(np.sqrt)
df.applymap(func)	Applies function to each element.	df.applymap(str.upper)

Selection and Filtering

Function	Description	Example
df["col"]	Selects a column.	df["Age"]
df.loc[row, col]	Label-based selection.	df.loc[0,"Name"]
df.iloc[row, col]	Position-based selection.	df.iloc[2,1]
df.at[row, col]	Fast access for single value (label-based).	df.at[0,"Name"]
df.iat[row, col]	Fast access for single value (position-based).	df.iat[0,1]
df.query(expr)	SQL-like querying.	df.query("Age > 30")
df.filter(items, regex, like)	Selects columns/rows by labels or regex.	df.filter(like="Age")
df.isin(items)	Filters df according to items given	df[df['Name'].isin(['Youssef'])

Math, Statistics and other utilities

Function	Description	Example
df.sum(axis)	Sum of values (rows/cols).	df.sum(axis=1)
df.mean()	Mean of values.	df["Salary"].mean()
df.median()	Median.	df.median()
df.mode()	Mode	df.mode()
df.std()	Standard deviation.	df.std()
df.corr()	Correlation matrix.	df.corr()
df.cov()	Covariance matrix.	df.cov()
df.value_counts()	Frequency count of values.	df["Dept"].value_counts() df["Dept"].value_counts().index df["Dept"].value_counts().values
df.unique()	Unique values of a Series.	df["Dept"].unique()
df.nunique()	Number of unique values.	df["Dept"].nunique()
df.duplicated()	Returns Boolean mask marking duplicate rows.	df.duplicated().sum()
df.drop_duplicates()	Removes duplicate rows (keeps first by default).	df.drop_duplicates(subset=["col"])

Note

- **Axis=1 -> column**
- **Axis=0 -> row**

Conclusion

This document has explored essential libraries such as NumPy and Pandas. Each section included explanations, structured tables, and practical code examples designed to make concepts both understandable and applicable.

By studying this document, learners, developers, and data enthusiasts can enhance their data manipulation and analysis skills with NumPy and Pandas, and confidently apply these tools in real-world problem solving. The aim of this resource is not only to serve as a learning guide but also as a reference for continuous growth and exploration in programming and data science.

Feedback & Contribution

This is a living document. If you have feedback, suggestions, or additional insights that could improve it, I welcome your input. Sharing ideas helps keep this material accurate, relevant, and valuable for everyone.

Copyright & Usage

© 2025 [Youssef Amgad Elkhatib].
This document is intended **for learning and reference purposes only**.
You are free to read, use, and share the content for personal or educational use, but:

- **Do not copy or republish this document as your own.**
- Always provide proper credit when referencing.
- Commercial use or redistribution without permission is not allowed.

You may not reproduce this document in whole or in part without attribution. Suggestions and feedback are welcome, and contributors will be acknowledged, but copyright remains with the author.

Acknowledgments

This document is authored and copyrighted by **[Youssef Amgad Elkhatib]**.
Special thanks to the contributors who provided valuable feedback and suggestions for improvements:

Name	Contribution