NumPy and pandas

Refresher

If you had a list of numbers representing temperatures for every day of the year, how would you find the average temperature using only Python?

10 minutes

1. **Download pittsburgh_weather_2024.csv**
2. **Open using fileIO**
3. **Create a list of daily high temperatures**
4. **Calculate average daily high temperature for the year**

Intro

- Pure Python is fine, but not optimal for large-scale data or analysis
- NumPy makes solving this problem faster and easier

Key Terms: Module

- Module
  - A module is a single Python file (.py extension) containing a collection of related functions, classes, and variables.
  - Modules promote code organization within a single file and allow code reuse by importing them into other scripts or modules.
  - Examples:
    - math
    - datetime

Key Terms: Package

- Package
  - A package is a way to organize related modules into a directory hierarchy.
  - A package directory must contain a special file named __init__.py to be recognized as such by Python.
  - Examples:
    - NumPy
    - pandas

Key Terms: Library

- Library
    - A library is a broader term, representing a collection of packages and modules offering a wide range of functionality for solving various problems.
    - Examples:
        - The Python standard library: bundled with every Python installation, contains numerous modules and packages for common tasks.
        - Third-party libraries are available for download from PyPI (Python Package Index).
            - Matplotlib for data visualization or
            - Scikit-learn for machine learning

Key Terms: pip and conda

- pip: the standard package installer for Python.
  - Pip install: installs Python packages from the Python Package Index (PyPI) or other sources, making them available for use in your Python projects.
  - Dependency resolution
  - Version management
- conda: open-source package and environment management system that is language-agnostic
  - Manages packages and also the Python interpreter itself
  - Can handle system-level dependencies and libraries that Python packages might rely on
  - Excels at creating and managing isolated environments that can contain different versions of Python and diverse sets of packages

Setup

Set up a Jupyter notebook in VS code or similar IDE

In a terminal, run:

```
    pip list
    pip install numpy pandas
    pip list
```

NumPy

- Python package for numerical computing that provides support for large arrays and matrices
- Can create arrays from lists
- Can easier conduct mathematical operations for multidimensional arrays

Documentation: https://numpy.org/doc/2.3/reference/index.html

NumPy Arrays vs. Python lists

- Python List
  - A flexible built-in structure that can hold items of any type (e.g., numbers, strings). Easy to use but slow for math and not optimized for large data.
  - Example: `[1, 'apple', 3.14]`

- NumPy Array
  - A fast, memory-efficient structure for numeric data. Only holds one data type and supports vectorized operations like `array + 2` or `array.mean()`.
  - Can use `np.array` for strings, but the arrays are optimized for numeric data and it behaves more like a Python list at that point, just stricter in type
  - Example: `np.array([1, 2, 3])`

Direct Comparison

Feature	Python List	NumPy Array (np.array)
Data Type	Can store **mixed types**	Stores **one data type only**
Performance	Slower for math operations	Much **faster**, optimized in C
Operations	Must use loops manually	Supports **vectorized operations**
Memory Efficiency	Less efficient	More **memory-efficient**
Functionality	Basic	Rich set of **math, stats, reshape**

2D Arrays

- A 2D array is like a grid or table — rows and columns of numbers where each row is a 1D array
- Commonly used for matrices, images, tables of data
- Helpful for Linear Algebra
- Example:

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

Output:

```
[[1 2 3]
 [4 5 6]]
```

- If NumPy sees mixed types (integers and strings), it automatically **upcasts** the integers to strings so all elements share a common type

Examples

Creating Arrays

- `np.array()`
    - Creates a NumPy array from a Python list
    - Example: `arr = np.array([1, 2, 3])`
- `np.arange(start, stop, step)`
    - Create a range of numbers (like range() but returns an array)
    - Example: `np.arange(0, 10, 2)`
- `np.linspace(start, stop, num)`
    - Evenly spaced numbers over an interval
    - Example: `np.linspace(0, 1, 5)`
- `np.zeros(shape)`
    - Array of all 0s
    - Example: `np.zeros((3,2))`
- `np.ones(shape)`
    - Array of all 1s
    - Example: `np.ones(4)`
- `np.random.rand(shape)`
    - Array of random numbers between 0 and 1
    - Example: `np.random.rand(2,2)`

- **Examples**

Array Operations

- Element-wise math: +, -, *, /, **
  - NumPy lets you do math on entire arrays — no loops needed
- Broadcasting
  - NumPy automatically stretches smaller arrays to match shapes when possible.
- Aggregation:
  - `np.mean()`
  - `np.sum()`
  - `np.min()`
  - `np.max()`

Examples

Array Indexing and Slicing

- 1D slicing: `arr[1:3]`
- 2D slicing: `arr[0, 0:2]`
- Boolean indexing: `arr[arr > 5]`
- Shape and Reshaping:
    - `.shape:` returns array dimensions
    - `.reshape():` change shape (must match total elements)
    - `.flatten():` convert 2D to 1D

Exercise – NumPy

Solve our problem from earlier, but with NumPy:

If you had a NumPy array of numbers representing temperatures for every day of the year, how would you find the average temperature?

Steps:

- Use the same list of daily high temperatures
- Convert it to a NumPy array
- Use one of NumPy's built-in aggregation functions to solve the question

pandas

- Python package built on top of NumPy and provides functions designed to work with structured data
- Series, DataFrames
- Can import data from various data types (JSON, CSV, EXCEL)
- Data cleaning, manipulation, aggregation, and transformation

Documentation: https://pandas.pydata.org/docs/reference/index.html

Creating Data

- From lists: `pd.Series()`
  - Create a single-column labeled array (like a NumPy array with labels)
  - Great for one-dimensional data like single column or time series

- From dictionaries: `pd.DataFrame()`
  - Create a table (rows and columns)
  - Keys become column names

- Reading CSVs: `pd.read_csv('file.csv')`
  - Loads spreadsheet-style data into a DataFrame
  - Automatically detects headers and values

- Reading JSON: `pd.read_json()`
  - Works with structured JSON, like nested dictionaries

- Reading Excel: `pd.read_excel()`
  - Requires other packages to be installed (`openpyxl` or `xlrd`)

Examples

Data Exploration

- Basic Inspection
  - `.head()`: first 5 rows
  - `.tail()`: last 5 rows
  - `.info()`: summary of columns, types, and nulls
  - `.describe()`: stats for numeric columns (mean, std, etc.)
- Checking nulls:
  - `.isnull().sum()`: shows how many missing (`NaN`) values are in each column

You should use these types of functions early to catch data quality issue before analysis or modeling

Examples

Selecting and Filtering Data

- Selecting columns: `df['column']`
  - Returns a single column as a Series
  - Use `df[[‘col1’,’col2’]]` to select multiple columns
- Filtering rows: `df[df['col'] > value]`
  - Returns only rows where the condition is true
- `loc[]` vs `iloc[]`
  - `loc[]:` when you know the row/column labels
  - `iloc[]:` when you want to access by position

Method	Access by	Example
`loc[]`	**Label-based**	`df.loc[2, 'Name']`
`iloc[]`	**Index-based**	`df.iloc[2, 0]`

Exercise – Pandas

Load all_olympic_medalists.csv and answer the following questions/ do the following tasks:

1. Show the first 5 rows and the last 5 rows.

2. How many rows are in the dataset?

3. Are there any missing values in any of the columns?

4. How many total medals are recorded in the dataset?

5. How many medals were awarded in the first Olympic year (1896)? How would you do this if you didn't know the first year?

6. How many medals has the US won?

7. What years were medals awarded for Rugby?

8. What was the first year women were included?

Common Cleaning Tasks

- Handling missing values:
  - `.dropna()`: remove rows with missing values
  - `.fillna()`: replace missing values

- Renaming columns: `df.rename(columns={'old_name': 'new_name'})`
  - Pass a dictionary to rename one or more columns.

- Changing types: `.astype()`
  - Useful for converting types (e.g., float → int, object → datetime)
  - Ex: `df['Age'] = df['Age'].astype(int)`

Data Transformation

- Creating new columns: `df['new'] = …`
  - You can add new columns based on existing ones
  - Example: `df['TotalPrice'] = df['Quantity'] * df['UnitPrice']`
- Applying functions (`.apply()`
  - Use `.apply()` with custom functions or `lambda` to modify data.
  - Example: `df['UpperName'] = df['Name'].apply(my_function)`
- Grouping: `df.groupby()`
  - Aggregate data by categories (sum, mean, count, etc.)
  - Example: `df.groupby('Department')['Salary'].mean()`
- Sorting: `.sort_values()`
  - Sort by one or more columns
  - Example: `df.sort_values('Score', ascending=False)`

Examples

Matplotlib

- A powerful Python library for creating static, animated, and interactive plots
- Ideal for data visualization, especially with NumPy or Pandas
- Easy to create line graphs, bar charts, scatter plots, histograms, and more
- Highly customizable: colors, labels, legends, annotations, styles
- Works well with pandas DataFrames: df.plot() uses Matplotlib

Final Exercise

Download top_20_womens_tours.csv
Answer the following questions/do the following tasks:

1. Drop the column 'Ref.'
2. Rename 'Adjusted gross (in 2022 dollars)' to 'Adjusted gross'.
3. List the artists and the number of times they appear in the rankings.
4. What Taylor Swift tours made the rankings?
5. Clean the actual gross and adjusted gross columns and convert them to float.
6. What is the total adjusted gross from all 20 concerts?
7. Calculate average gross/show for each rank using the adjusted gross.
8. How much has Taylor Swift earned from the concerts on this list?