Module 7

Created by	Karan Maurya
Status	Not started

Topics

Python Cheatsheet: Module 7

(Advance + Intermediate Friendly – Structured and Practical)

Module 7: Introduction to Data Science and Visualization

7.1 Storing and Reading Data with JSON

What is JSON?

- JSON (**JavaScript Object Notation**) is a lightweight data interchange format.
- Used for **storing and exchanging data** between programs.

Working with JSON in Python

Saving (Writing) Data to JSON File:

```
import json

data = {'name': 'John', 'age': 30, 'city': 'New York'}

with open('data.json', 'w') as f:
    json.dump(data, f)
```

Reading Data from JSON File:

```
import json

with open('data.json', 'r') as f:
    loaded_data = json.load(f)
print(loaded_data)
```

Other Useful JSON Methods

Method	Purpose
`json.dump(obj, file)`	Write Python object to file
`json.dumps(obj)`	Serialize Python object to JSON-formatted string
`json.load(file)`	Read JSON object from file
`json.loads(string)`	Deserialize JSON-formatted string to Python object

Example using `dumps()`:

```
json_string = json.dumps(data)
print(json_string)
```

> Tip: Use indent=4 in json.dump/json.dumps to pretty-print JSON.

7.2 User-Generated Data

- Programs can **accept user input**, **process it**, and **store results**.
- Useful when creating **dynamic datasets**.

Example:

```
import json

username = input("Enter your name: ")

with open('user.json', 'w') as f:
    json.dump(username, f)
```

Loading back:

```
with open('user.json') as f:
    username = json.load(f)
print(f"Welcome back, {username}!")
```

7.3 Data Science Libraries

NumPy – Numerical Computing

Key Concepts:

- Fast array operations
- Linear algebra, statistics, random number generation

Creating Arrays:

```
import numpy as np

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

Useful NumPy Methods:

Function	Purpose
np.zeros(shape)	Create array filled with 0s
np.ones(shape)	Create array filled with 1s
np.arange(start, stop, step)	Create range of numbers
np.linspace(start, end, num)	Create evenly spaced numbers
np.reshape(arr, newshape)	Reshape array
np.mean(arr)	Mean
np.median(arr)	Median
np.std(arr)	Standard deviation

Example:

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

Matplotlib – Data Visualization

Plotting Graphs:

Basic Line Plot:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
```

```
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.grid(True)
plt.show()
```

Other Plot Types:

Plot	Function
Line Plot	`plt.plot()`
Scatter Plot	`plt.scatter(x, y)`
Bar Chart	`plt.bar(x, y)`
Histogram	`plt.hist(data)`
Pie Chart	`plt.pie(sizes, labels=labels)`

Example of a Scatter Plot:

```
plt.scatter([1,2,3,4], [10,20,25,30])
plt.show()
```

Pandas – Data Manipulation and Analysis

Basics:

Creating a DataFrame:

```
import pandas as pd

data = {
    'Name': ['John', 'Anna', 'Xiao'],
    'Age': [28, 24, 22]
}
df = pd.DataFrame(data)
print(df)
```

Useful Pandas Methods:

Function	Purpose
`df.head()`	First 5 rows
`df.tail()`	Last 5 rows
`df.info()`	Summary of DataFrame
`df.describe()`	Statistical summary
`df.shape`	Dimensions (rows, cols)
`df.columns`	Column names
`df.isnull()`	Detect missing values
`df.fillna(value)`	Fill missing values
`df.dropna()`	Remove rows with missing values
`df.sort_values(by)`	Sort DataFrame

Example:

```
print(df.describe())
df_sorted = df.sort_values(by='Age')
print(df_sorted)
```

Summary for Module 7

Topic	Focus	Special Notes
JSON Handling	Save/load structured data	Use `indent=4` for readability
User Data	Accept, store, and reuse user info	Combine with JSON or databases
NumPy	Fast math operations	Vectorization improves performance
Matplotlib	Data visualization	Always label your axes!
Pandas	Data analysis	Master DataFrames for real-world tasks

Bonus Tips for Advanced Users

- Combine **Pandas** + **Matplotlib** to plot graphs directly from DataFrames.
- Always perform **null checks** in Pandas before starting any serious data analysis.
- Use **NumPy vectorized operations** instead of loops for faster computations.
- Always **customize plots** with grid, colors, markers for better presentation in Matplotlib.
