**Python, NumPy, and Pandas Interview Cheat Sheet**

**Python**

1. **What are Python's key features?**
   * Interpreted, high-level, dynamically typed, and object-oriented.
   * Rich standard library and extensive third-party modules.
   * Easy to learn, read, and maintain. **Example:** Python’s simplicity allows quick prototyping. For instance, building a web scraper using requests and BeautifulSoup is straightforward.
2. **What is the difference between a list, set, tuple, and dictionary?**
   * **List**: Ordered, mutable, allows duplicates. Example: [1, 2, 3].
   * **Set**: Unordered, unique elements, mutable. Example: {1, 2, 3}.
   * **Tuple**: Ordered, immutable, allows duplicates. Example: (1, 2, 3).
   * **Dictionary**: Key-value pairs, mutable, unique keys. Example: {'a': 1, 'b': 2}. **Real-World Use:**
   * Use lists for ordered data like tasks.
   * Sets for removing duplicates in datasets.
   * Tuples for fixed configurations like geographical coordinates.
   * Dictionaries for mapping relationships like usernames and user IDs.
3. **What is list comprehension, and why use it?**
   * Compact way to create lists.
   * Example: [x\*\*2 for x in range(5)] generates [0, 1, 4, 9, 16]. **Real-World Use:** Extract email domains from a list of email addresses:
   * emails = ['user1@gmail.com', 'user2@yahoo.com']
   * domains = [email.split('@')[1] for email in emails]
4. **What is the difference between deep copy and shallow copy?**
   * **Shallow Copy**: Copies references to objects. Changes in nested objects reflect in the copy. Use copy.copy().
   * **Deep Copy**: Creates an independent copy of objects and nested objects. Use copy.deepcopy(). **Example:**
     1. import copy
     2. original = [[1, 2], [3, 4]]
     3. shallow = copy.copy(original)
     4. deep = copy.deepcopy(original)

**NumPy**

1. **What is NumPy and its main advantage?**
   * A library for numerical computing in Python.
   * Provides efficient operations on large arrays and matrices using vectorization. **Example:**
     1. import numpy as np
     2. arr = np.array([1, 2, 3])
     3. print(arr + 1) # Output: [2 3 4]

**Real-World Use:** Analyse sensor data from IoT devices by performing operations on arrays.

1. **How is NumPy array different from Python list?**
   * NumPy arrays are faster, require less memory, and support element-wise operations.
     1. Example: np.array([1, 2, 3]) + 1 gives [2, 3, 4].
2. **How do you create a Numpy array?**
   * From a list: np.array([1, 2, 3]).
   * Zeros: np.zeros((2, 3)).
   * Random: np.random.rand(3, 2). **Example:**
   * np.arange(0, 10, 2) # Output: array([0, 2, 4, 6, 8])
3. **Explain NumPy's broadcasting.**
   * Automatically aligns array shapes during operations without explicit reshaping. It expands the shape of one array to the other array shape.
   * **Arrays of different shape.**
   * Example: Adding scalar to an array: arr + 5. **Real-World Use:** Apply tax adjustments to income data stored in arrays.

**Drawbacks**: First dimensions are similar in arrays. Number of dimensions are equal

1. **What is vectorization, and why is it important?**
   * Eliminates for-loops by performing operations on entire arrays.
   * **Arrays of Same shape**
   * It leveraging the numpy’s internal c- optimized resources. Hence improved performance.
   * Faster and concise code. Example: arr1 \* arr2 vs. looping.

**Example:**

* + arr1 = np.array([1, 2, 3])
  + arr2 = np.array([4, 5, 6])
  + result = arr1 \* arr2 # Output: [4 10 18]

**Drawbacks**: Vectorised operations often require allocating memory for intermediate arrays, which can lead to high memory usage.

**6**. **np.bincount**: counting occurrences of values.

**7. np.histogram**: Define the bins for range of values. Along with bin edges.

**8**. **vstack vs hstack:**

**Vstack**: stack arrays vertically (row wise)

**Note: All imput arrays have same number of columns otherwise it raises the value error**

**Example**:

**a = np.array([[1, 2], [3, 4]]) # Shape: (2, 2)**

**b = np.array([[5, 6]]) # Shape: (1, 2)**

**result = np.vstack((a, b))**

**Output: [[1 2][3 4][5 6]]**

**Hstack**: stack arrays horizontally (column wise)

Note: All input arrays have same number of rows otherwise it raises the value error.

**Example:**

a = np.array([[1, 2], [3, 4]]) # Shape: (2, 2)

b = np.array([[5], [6]]) # Shape: (2, 1)

result = np.hstack((a, b))

Output: # [[1 2 5] # [3 4 6]]

**Pandas**

1. **What is Pandas?**
   * A Python library for data manipulation and analysis.
   * Works with structured data in **DataFrame** and **Series** formats.
2. **What are Series and DataFrame?**
   * **Series**: 1D labeled array. Example: pd.Series([1, 2, 3], index=['a', 'b', 'c']).
   * **DataFrame**: 2D table with labeled rows and columns. Example: pd.DataFrame({'A': [1, 2], 'B': [3, 4]}). **Real-World Use:**
   * Use Series to hold time-series data like stock prices.
   * Use DataFrames to handle tabular data like sales records.
3. **How to handle missing data in Pandas?**
   * Drop rows/columns: df.dropna().
   * Fill values: df.fillna(value).

**Example:**df['column'].fillna(df['column'].mean(), inplace=True)

1. **Explain groupby in Pandas.**
   * Used to split, apply, and combine data based on a key.
   * Example: df.groupby('column')['target'].sum().

**Real-World Use case :** Group sales data by region to calculate total revenue.

1. **How to merge DataFrames in Pandas?**
   * **merge()**: Combine based on a key. Example: pd.merge(df1, df2, on='key').
   * **concat()**: Stack DataFrames. Example: pd.concat([df1, df2]).
2. **What is the difference between loc and iloc?**
   * **loc**: Access rows/columns by labels.

Example: df.loc[0:2, 'col1'].

* + **iloc**: Access rows/columns by index.

Example: df.iloc[0:2, 0].

Extract rows based on condition:

Example: df.loc[df['age'] > 30]

7.  **Pd.merge: Combines two Data frames based on the common column name**

***Inner Join:Ex: df\_addresses = pd.merge(df\_addresses, df\_cities, on='City')***

***Left join: df\_addresses = pd.merge(df\_addresses, df\_cities, on='City', how = ‘left’)***

***Right join: df\_addresses = pd.merge(df\_addresses, df\_cities, on='City', how = ‘left’)***

**8. Agg and join:** When converting values in a **pandas DataFrame or Series back into a single string**, we typically use **agg** and **join** in combination to concatenate those values

***Ex:***

*df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]})*

*result = df['Name'].agg(', '.join)*

**8. Difference between apply, applymap and map:**

 Map → Series, applies function element-wise.

 **apply map** → DataFrame, applies function element-wise.

 apply → DataFrame or Series, applies function along an axis (rows or columns)

|  |
| --- |
|  |
|  |

| **Scenario** | **Use** |
| --- | --- |
| Transform each element in a **Series (element wise Transformation)** | map |
| Apply a function to each element in a **DataFrame (ele wise trans)** | applymap |
| Perform **row-wise/column-wise operations** or apply a function to a **Series** | apply |

**Examples:**

1. **map**: Transform values in a single column (City → Uppercase).
2. **apply**: Perform operations across columns or rows (calculate Net Cost for a row).
3. **applymap**: Apply transformations element-wise across the entire DataFrame (e.g., scaling numerical values).