Python Viva

Important Questions Unit wise:

Unit I: Introduction to Data Science & Data Analysis

Conceptual Questions:

1. What is Data Science, and how does it differ from traditional programming?
2. Explain the components of Data Science.
3. Describe the Data Science process.
4. What are the different job roles in Data Science?
5. List some tools used in Data Science and their purposes.
6. How is Data Science different from Business Intelligence (BI)?
7. Explain some real-world applications of Data Science.
8. What are some challenges faced in Data Science?

Data Analysis Questions:

1. What is Data Analysis, and why is it important?

2. What are the different types of Data Analysis techniques?
3. What is the Data Analysis process?
4. Name some popular Data Analysis tools.

Python Basics Questions:

1. What are the features of Python that make it useful for Data Science?
2. What is the difference between a **Python script** and the **interactive interpreter**?
3. Explain different Python modes: Interactive vs. Script mode.
4. What are Python's basic data types?
5. What are variables and keywords in Python?
6. What are identifiers in Python?
7. Explain the concept of statements in Python with an example.

Unit II: Python Fundamentals & Control Flow

Expressions and Operators:

1. What are expressions in Python?
2. What is the importance of operator precedence?
3. Explain tuple assignment with an example.
4. What are different types of operators in Python?
5. What is the difference between `==` and `is` operators?

Functions in Python:

1. What is a function in Python?
2. What are the types of functions in Python?
3. Explain the concept of function arguments and parameters.
4. What is the difference between a **fruitful function** and a **void function**?
5. Explain **local** and **global** variables with examples.
6. What is function composition?

7. What are modules in Python? How do you import a module?

Conditional & Iteration Statements:

1. What are different types of conditional statements in Python?
2. What is the difference between **if-else** and **if-elif-else**?
3. What is a nested conditional statement?
4. Explain different loop types in Python (`for`, `while`).
5. What is the purpose of `break`, `continue`, and `pass` statements?
6. What is the difference between a `for` loop and a `while` loop?

Unit III: Data Structures in Python

Strings:

1. What are strings in Python?
2. What is meant by **immutability** of strings?
3. What are some common string methods?
4. How do you access a substring using **slicing**?
5. What is the difference between a string function and a string method?

Lists and Tuples:

1. What is a list in Python?
2. What are list operations? Provide examples.
3. What is **list mutability**, and how does it impact performance?
4. Explain **aliasing** and **cloning** of lists.
5. How do you perform **list slicing**?
6. What are tuples? How are they different from lists?
7. How do you assign values to a tuple?
8. Can a tuple be modified? If not, why?

Dictionaries:

1. What is a dictionary in Python?
2. How do you create a dictionary?
3. How do you access and modify dictionary elements?
4. What are the major differences between **lists, tuples, and dictionaries**?

Advanced List Processing:

1. What is **list comprehension**? Provide an example.
2. What is a **nested list**, and how do you work with it?

Unit IV: Introduction to NumPy

NumPy Arrays & Computations:

1. What is NumPy, and why is it useful?
2. How is a **NumPy array** different from a Python list?
3. How do you create a NumPy array?
4. How do you perform element-wise operations in NumPy?
5. Explain aggregation functions in NumPy (`sum()`, `mean()`, `max()`, etc.).
6. What are **Boolean masks** in NumPy?
7. How does **fancy indexing** work in NumPy?
8. How do you sort a NumPy array?

Structured Data with NumPy:

1. What is structured data in NumPy?
2. How do you store and manipulate structured data in NumPy?

Unit V: Data Manipulation with Pandas

Pandas Basics:

1. What are the main data structures in Pandas?
2. What is a Pandas **Series**? How is it different from a NumPy array?

3. What is a Pandas **DataFrame**? How do you create one?
4. How do you index and select data in Pandas?

Handling Missing Data:

1. What are different methods to handle missing data in Pandas?
2. How do you use **fillna()** to fill missing values?
3. What is **forward fill** and **backward fill** in Pandas?

Hierarchical Indexing:

1. What is **hierarchical indexing** in Pandas?
2. How do you create a MultiIndex DataFrame?

Combining Datasets:

1. What are different ways to combine datasets in Pandas?
2. Explain the difference between **merge()**, **concat()**, and **join()**.
3. What is an **inner join** and an **outer join**?

Aggregation and Grouping:

1. What is the purpose of `groupby()` in Pandas?
2. How do you apply multiple aggregation functions in a groupby operation?
3. Provide an example of **grouping data** and applying an aggregation function.

Solutions of Questions

Unit I: Introduction to Data Science & Data Analysis

Conceptual Questions:

1. **What is Data Science, and how does it differ from traditional programming?**

   Data Science is like being a super-smart explorer who uses numbers and facts (data) to figure things out, like predicting if it'll rain or what games kids like. Traditional programming is more like building tools or games with code, not solving mysteries with data.

2. **Explain the components of Data Science.**

   Data Science has three big parts:

   - **Math**: Like counting and adding to find patterns.
   - **Coding**: Telling computers what to do (like with Python).
   - **Curiosity**: Asking questions like "Why do people buy more ice cream in summer?"

3. **Describe the Data Science process.**

   It's like a treasure hunt:

   - Collect clues (data).
   - Clean up the mess (fix bad data).
   - Look for patterns (analyze).
   - Share your discovery (tell others what you found).

4. **What are the different job roles in Data Science?**

   - **Data Scientist**: The main detective solving data puzzles.
   - **Data Analyst**: Helps find simple answers from data.
   - **Machine Learning Engineer**: Teaches computers to learn from data.
   - **Data Engineer**: Builds the pipes to move data around.

5. **List some tools used in Data Science and their purposes.**

   - **Python**: For coding and analyzing data.
   - **Excel**: For making charts and lists.
   - **R**: Another coding tool for math stuff.
   - **Tableau**: Makes cool pictures from data.

6. **How is Data Science different from Business Intelligence (BI)?**

Data Science is about guessing what might happen (like “Will it snow?”), while BI is about looking at what already happened (like “How much snow fell last week?”).

7. **Explain some real-world applications of Data Science.**
    - Netflix suggesting shows you’ll like.
    - Doctors predicting if someone might get sick.
    - Stores knowing what toys to stock for Christmas.
8. **What are some challenges faced in Data Science?**
    - Messy data (like a puzzle with missing pieces).
    - Too much data to handle.
    - Making sure the answers make sense and aren’t silly.

Data Analysis Questions:

1. **What is Data Analysis, and why is it important?**

    Data Analysis is like sorting your toys to see which ones you play with most. It’s important because it helps us understand things and make smart choices, like knowing what snacks to buy.
2. **What are the different types of Data Analysis techniques?**
    - **Descriptive**: What happened? (e.g., “I ate 5 cookies.”)
    - **Diagnostic**: Why did it happen? (e.g., “I was hungry!”)
    - **Predictive**: What might happen? (e.g., “I’ll eat more tomorrow.”)
    - **Prescriptive**: What should I do? (e.g., “Buy more cookies!”)
3. **What is the Data Analysis process?**
    - Grab the data (like counting your toys).
    - Clean it (throw out broken ones).
    - Look at it (see which toys are favorites).
    - Tell someone (say, “I love my robot best!”).
4. **Name some popular Data Analysis tools.**
    - **Python**: For coding and counting.

- **Excel**: For making tables.
- **Google Sheets**: Like Excel, but online.
- **Power BI**: For fancy charts.

Python Basics Questions:

1. **What are the features of Python that make it useful for Data Science?**
    - Easy to read (like a storybook).
    - Lots of helpers (libraries like Pandas for data).
    - Works on any computer.
    - Great for math and pictures.
2. **What is the difference between a Python script and the interactive interpreter?**
    - **Script**: A saved list of instructions (like a recipe you write down).
    - **Interpreter**: Talking to Python live (like chatting with a friend).
3. **Explain different Python modes: Interactive vs. Script mode.**
    - **Interactive**: Type one thing, get an answer right away (e.g., `2 + 2` → `4`).
    - **Script**: Write a bunch of code in a file, then run it all at once.
4. **What are Python's basic data types?**
    - **Numbers**: Like `5` or `3.14`.
    - **Text (Strings)**: Like `"Hello"`.
    - **Lists**: Like `[1, 2, 3]` (a shopping list).
    - **True/False (Boolean)**: Like `True` or `False`.
5. **What are variables and keywords in Python?**
    - **Variables**: Boxes to hold stuff (e.g., `age = 10`).
    - **Keywords**: Special words Python understands (e.g., `if`, `for`—you can't use them as box names).
6. **What are identifiers in Python?**

    Identifiers are names you make up for things like variables (e.g., `my_toys`). They can't start with numbers or use spaces.

7. **Explain the concept of statements in Python with an example.**

   A statement is like a command you give Python. Here's an example:

   ```
   candies = 5  # This is a statement—it puts 5 in a box called "candies"
   print(candies)  # This statement tells Python to say "5"
   ```

Example Code for Fun:

Here's a little Python code to try:

```
# Counting my toys
toys = 3
print("I have", toys, "toys!")
toys = toys + 2  # I got more toys!
print("Now I have", toys, "toys!")
```

This will say:

`I have 3 toys!`

`Now I have 5 toys!`

Unit II: Python Fundamentals & Control Flow

Expressions and Operators:

1. **What are expressions in Python?**

   An expression is like a math problem or a little sentence Python can figure out, like `2 + 3` (which equals `5`) or `"Hi" + " there"` (which makes `"Hi there"`).

2. **What is the importance of operator precedence?**

   It's like deciding who goes first in a game. In `2 + 3 * 4`, Python does the `*` (multiplication) first, so it's `2 + 12 = 14`, not `5 * 4 = 20`. It's the rules for math in code!

3. **Explain tuple assignment with an example.**

   It's like unpacking a lunchbox with two things at once. Example:

```
lunch = ("apple", "juice")  # A pair (tuple)
fruit, drink = lunch  # Unpacks it!
print(fruit)  # Says "apple"
print(drink)  # Says "juice"
```

4. **What are different types of operators in Python?**
    - **Math**: `+`, , , `/` (add, subtract, multiply, divide).
    - **Comparison**: `==` (equal), `!=` (not equal), `<`, `>`.
    - **Logical**: `and`, `or`, `not` (for true/false stuff).
    - **Assignment**: `=` (puts stuff in a box).
5. **What is the difference between `==` and `is` operators?**
    - `==` checks if two things have the same value (like `5 == 5` is `True`).
    - `is` checks if they’re the exact same thing in memory (like two toys being the same toy, not just looking alike).

Functions in Python:

1. **What is a function in Python?**

    A function is like a little robot you tell what to do once, and it does it whenever you call its name. Like a “clean my room” button!
2. **What are the types of functions in Python?**
    - **Built-in**: Ones Python already knows (like `print()`).
    - **User-defined**: Ones you make up (like your own robot).
3. **Explain the concept of function arguments and parameters.**
    - **Parameters**: The names you give the function to expect stuff (like “toy”).
    - **Arguments**: The actual stuff you give it (like “ball”).

```
def play(toy):  # "toy" is the parameter
    print("I like my", toy)
play("ball")  # "ball" is the argument
```

4. **What is the difference between a fruitful function and a void function?**
   - **Fruitful**: Gives you something back (like `add(2, 3)` gives `5`).
   - **Void**: Just does something, no gift (like `print("Hi")`—it shows “Hi” but doesn’t give you anything to keep).
5. **Explain local and global variables with examples.**
   - **Local**: A secret only the function knows.
   - **Global**: A toy everyone can see.

```
toy = "car"  # Global (outside)
def play():
    toy = "ball"  # Local (inside)
    print(toy)  # Says "ball"
play()
print(toy)  # Says "car" (global stays "car")
```

6. **What is function composition?**

   It’s like telling one robot to use another robot’s work. Example:

```
def add(a, b):
    return a + b
def double(x):
    return add(x, x)  # Uses "add" inside!
print(double(3))  # 3 + 3 = 6
```

7. **What are modules in Python? How do you import a module?**

   A module is like a toolbox with extra tools. You “import” it to use it:

```
import math  # Gets the math toolbox
print(math.sqrt(16))  # Uses "sqrt" to find 4 (square root)
```

Conditional & Iteration Statements:

1. **What are different types of conditional statements in Python?**
   - `if`: Do something if true (like “If it’s sunny, I’ll play”).

- `else`: Do something else if false.
- `elif`: Check another “if” (like “If not sunny, but warm, I’ll swim”).

2. **What is the difference between if-else and if-elif-else?**
    - `if-else`: One choice or another (yes or no).
    - `if-elif-else`: Lots of choices (like picking ice cream, cake, or nothing).

```
weather = "rainy"
if weather == "sunny":
    print("Play outside!")
elif weather == "rainy":
    print("Stay inside!")
else:
    print("Maybe read!")
```

3. **What is a nested conditional statement?**

    It’s an “if” inside another “if,” like a question inside a question:

```
age = 10
if age > 5:
    if age < 12:
        print("You’re a big kid!")  # Only if both are true
```

4. **Explain different loop types in Python (`for`, `while`).**
    - `for`: Counts through a list (like “Eat 1 cookie, 2 cookies, 3 cookies”).
    - `while`: Keeps going until you say stop (like “Keep eating until full”).

```
for i in [1, 2, 3]:  # For loop
    print("Cookie", i)
count = 0
while count < 3:  # While loop
    print("Yum", count)
    count = count + 1
```

5. **What is the purpose of `break`, `continue`, and `pass` statements?**
    - `break`: Stops the loop early (like “I’m full, stop eating!”).

- `continue`: Skips one turn (like “Skip this yucky cookie”).
- `pass`: Does nothing (like “I’ll think about it later”).

```
for i in [1, 2, 3]:
    if i == 2:
        continue  # Skips 2
    print(i)  # Prints 1, 3
```

6. **What is the difference between a `for` loop and a `while` loop?**
    - `for`: Knows exactly how many times to go (like counting 5 toys).
    - `while`: Goes until something changes (like eating until the jar’s empty).

Fun Example Code:

Here’s a little game with what we learned:

```
def guess_number(secret):
    tries = 0
    while tries < 3:  # You get 3 tries
        guess = 4  # Pretend you guessed 4
        if guess == secret:
            print("You win!")
            break  # Stop if you’re right
        else:
            print("Try again!")
        tries = tries + 1
    print("Game over!")

guess_number(4)  # Secret number is 4
```

This will say:

`You win!`

`Game over!`

Unit III: Data Structures in Python

Strings:

1. **What are strings in Python?**

   Strings are like words or sentences in quotes, like `"hello"` or `"I like candy"`. They're just text Python can read!

2. **What is meant by immutability of strings?**

   Immutability means you can't change a string once it's made—it's like writing in permanent marker. You can make a new string, but the old one stays the same.

3. **What are some common string methods?**

   - `.upper()`: Makes it shouty, like `"hi"` → `"HI"`.
   - `.lower()`: Makes it quiet, like `"HI"` → `"hi"`.
   - `.replace("a", "b")`: Swaps letters, like `"cat"` → `"cbt"`.

   ```
   word = "dog"
   print(word.upper())  # Says "DOG"
   ```

4. **How do you access a substring using slicing?**

   Slicing is like cutting a piece of a string. Use `[start:end]`:

   ```
   text = "pizza"
   print(text[0:3])  # Says "piz" (starts at 0, stops before 3)
   ```

5. **What is the difference between a string function and a string method?**

   - **Function**: A helper outside the string, like `len("hi")` (tells you 2).
   - **Method**: A trick the string knows, like `"hi".upper()` (makes `"HI"`).

Lists and Tuples:

1. **What is a list in Python?**

   A list is like a toy box where you can put lots of things, like `[1, "ball", 3]`. You can add or take stuff out!

2. **What are list operations? Provide examples.**

   - Add: `.append()` puts something in.
   - Remove: `.pop()` takes something out.

```
toys = ["car", "doll"]
toys.append("ball")  # Adds "ball"
print(toys)  # Says ["car", "doll", "ball"]
toys.pop()  # Takes "ball" out
print(toys)  # Says ["car", "doll"]
```

3. **What is list mutability, and how does it impact performance?**

   Mutability means you can change the list (add or remove stuff). It's fast for small changes but can slow down if the list gets super big because Python has to rearrange things.

4. **Explain aliasing and cloning of lists.**

   - **Aliasing**: Two names for the same list, like nicknames. If you change one, the other changes too.
   - **Cloning**: Making a copy, so they're separate.

```
list1 = [1, 2]
list2 = list1  # Aliasing (same list)
list2[0] = 5
print(list1)  # Says [5, 2]
list3 = list1[:]  # Cloning (new list)
list3[0] = 9
print(list1)  # Still [5, 2]
```

5. **How do you perform list slicing?**

   It's like cutting a piece of the list:

```
numbers = [10, 20, 30, 40]
print(numbers[1:3])  # Says [20, 30] (from 1 to before 3)
```

6. **What are tuples? How are they different from lists?**

   Tuples are like lists but locked—you can't change them. They use `()` instead of `[]`. Example: `(1, "hi")`.

7. **How do you assign values to a tuple?**

   Just put them in `()` or even without:

```
stuff = (1, "cat")  # Tuple with 2 things
a, b = stuff  # Unpacks: a = 1, b = "cat"
print(a, b)  # Says 1 cat
```

8. **Can a tuple be modified? If not, why?**

   No, it's like a toy sealed in a box—immutable! Python locks it to keep it safe and fast for things that shouldn't change.

Dictionaries:

1. **What is a dictionary in Python?**

   A dictionary is like a magic book where you look up a word (key) and get something back (value), like `{"name": "Max", "age": 10}`.

2. **How do you create a dictionary?**

   Use curly braces `{}` with keys and values:

   ```
   kid = {"name": "Alex", "toys": 3}
   print(kid)  # Says {"name": "Alex", "toys": 3}
   ```

3. **How do you access and modify dictionary elements?**

   - **Access**: Use the key in `[]`.
   - **Modify**: Set a new value with `=`.

   ```
   kid = {"name": "Alex"}
   print(kid["name"])  # Says "Alex"
   kid["name"] = "Sam"  # Changes it
   print(kid["name"])  # Says "Sam"
   ```

4. **What are the major differences between lists, tuples, and dictionaries?**

   - **Lists**: Changeable, ordered, use `[1, 2, 3]`.
   - **Tuples**: Unchangeable, ordered, use `(1, 2, 3)`.
   - **Dictionaries**: Changeable, unordered, use `{"key": "value"}` with keys instead of numbers.

Advanced List Processing:

1. **What is list comprehension? Provide an example.**

   It's a quick way to make a list, like a magic spell:

   ```
   numbers = [x * 2 for x in [1, 2, 3]]  # Doubles each number
   print(numbers)  # Says [2, 4, 6]
   ```

2. **What is a nested list, and how do you work with it?**

   A nested list is a list inside a list, like a box in a box:

   ```
   boxes = [[1, 2], [3, 4]]
   print(boxes[0])  # Says [1, 2] (first box)
   print(boxes[0][1])  # Says 2 (second thing in first box)
   ```

Fun Example Code:

Here's a little toy box adventure:

```
# My toy box
toys = ["car", "doll", "ball"]
print("I have:", toys)

# Add a toy with a dictionary
toy_info = {"name": "robot", "count": 1}
toys.append(toy_info["name"])
print("Now I have:", toys)

# Slice my favorites
faves = toys[1:3]
print("My faves:", faves)
```

This will say:

`I have: ['car', 'doll', 'ball']`

`Now I have: ['car', 'doll', 'ball', 'robot']`

`My faves: ['doll', 'ball']`

Unit IV: Introduction to NumPy

NumPy Arrays & Computations:

1. **What is NumPy, and why is it useful?**

   NumPy is like a magic math toolbox for Python. It helps you work with lots of numbers super fast—like counting all your candies or toys in one go! It's great for science, games, and big data.

2. **How is a NumPy array different from a Python list?**

   - A **Python list** is like a toy box where anything fits (`[1, "hi", 3]`).
   - A **NumPy array** is a neat row of numbers (or one type of thing) that's faster and better for math, like `[1, 2, 3]` .

3. **How do you create a NumPy array?**

   You use the NumPy toolbox (called `numpy`) and give it numbers:

   ```
   import numpy as np  # Nickname "np" for NumPy
   my_array = np.array([1, 2, 3])  # Makes an array
   print(my_array)  # Says [1 2 3]
   ```

4. **How do you perform element-wise operations in NumPy?**

   It's like doing math to every number at once:

   ```
   import numpy as np
   numbers = np.array([1, 2, 3])
   doubled = numbers * 2  # Multiplies each one
   print(doubled)  # Says [2 4 6]
   ```

5. **Explain aggregation functions in NumPy (`sum()` , `mean()` , `max()` , etc.).**

   These are like shortcuts to summarize your numbers:

   - `sum()` : Adds everything up.
   - `mean()` : Finds the average.
   - `max()` : Picks the biggest.

   ```
   import numpy as np
   scores = np.array([10, 20, 30])
   print(scores.sum())  # Says 60
   ```

```
print(scores.mean())  # Says 20.0
print(scores.max())  # Says 30
```

6. **What are Boolean masks in NumPy?**

   A Boolean mask is like a treasure map with `True` or `False` to pick certain numbers:

   ```
   import numpy as np
   ages = np.array([5, 10, 15])
   mask = ages > 8  # Checks which are bigger than 8
   print(mask)  # Says [False True True]
   print(ages[mask])  # Says [10 15] (only the True ones)
   ```

7. **How does fancy indexing work in NumPy?**

   It's like picking your favorite toys by their spots:

   ```
   import numpy as np
   toys = np.array(["car", "doll", "ball"])
   picks = [0, 2]  # I want the 1st and 3rd
   print(toys[picks])  # Says ["car" "ball"]
   ```

8. **How do you sort a NumPy array?**

   Use `.sort()` to line up numbers or words:

   ```
   import numpy as np
   numbers = np.array([3, 1, 2])
   numbers.sort()  # Puts them in order
   print(numbers)  # Says [1 2 3]
   ```

Structured Data with NumPy:

1. **What is structured data in NumPy?**

   Structured data is like a chart where each row has different kinds of info—like a list of kids with names, ages, and toys all in one neat package.

2. **How do you store and manipulate structured data in NumPy?**

   You make a special array with names for each column:

```
import numpy as np
# Define the types (like name is text, age is number)
kids = np.array([("Max", 10, 3), ("Lily", 8, 2)],
              dtype=[("name", "U10"), ("age", "i4"), ("toys", "i4")])
print(kids)  # Shows the whole chart
print(kids["name"])  # Says ["Max" "Lily"] (just names)
kids["toys"][0] = 5  # Max got more toys!
print(kids["toys"])  # Says [5 2]
```

Fun Example Code:

Here's a little NumPy adventure:

```
import numpy as np

# My candy stash
candies = np.array([2, 5, 1, 4])
print("My candies:", candies)

# Double my candies!
doubled = candies * 2
print("Doubled:", doubled)  # Says [4 10 2 8]

# Who has more than 3?
mask = candies > 3
print("More than 3:", candies[mask])  # Says [5 4]

# Total candies
print("Total:", candies.sum())  # Says 12
```

This will show:

`My candies: [2 5 1 4]`

`Doubled: [4 10 2 8]`

`More than 3: [5 4]`

`Total: 12`

Unit V: Data Manipulation with Pandas

Pandas Basics:

1. **What are the main data structures in Pandas?**
    - **Series**: Like a single list of stuff (e.g., your candy counts).
    - **DataFrame**: Like a big chart or table (e.g., a list of kids with their names and candies).
2. **What is a Pandas Series? How is it different from a NumPy array?**

    A Series is like a labeled list—like `[2, 5, 3]` with names for each spot (e.g., "Monday", "Tuesday"). A NumPy array is just numbers without labels, and Series can mix types better.

    ```
    import pandas as pd
    candies = pd.Series([2, 5, 3], index=["Mon", "Tue", "Wed"])
    print(candies)  # Says: Mon 2, Tue 5, Wed 3
    ```
3. **What is a Pandas DataFrame? How do you create one?**

    A DataFrame is like a table with rows and columns—like a school chart of names and grades. You can make one from a dictionary:

    ```
    import pandas as pd
    data = {"Name": ["Max", "Lily"], "Candies": [3, 5]}
    df = pd.DataFrame(data)
    print(df)  # Shows a table: Name | Candies
    ```
4. **How do you index and select data in Pandas?**
    - Use labels with `.loc`: Pick by name.
    - Use numbers with `.iloc`: Pick by spot.

    ```
    import pandas as pd
    df = pd.DataFrame({"Name": ["Max", "Lily"], "Candies": [3, 5]})
    print(df.loc[0, "Name"])  # Says "Max" (row 0, "Name" column)
    print(df.iloc[1, 1])  # Says 5 (row 1, column 1)
    ```

Handling Missing Data:

1. **What are different methods to handle missing data in Pandas?**

- **Drop it**: Throw out rows with missing stuff using `dropna()`.
- **Fill it**: Put in a guess with `fillna()`.
- **Ignore it**: Just work around it if you can.

2. **How do you use fillna() to fill missing values?**

   It's like filling empty candy jars with a number:

```
import pandas as pd
df = pd.DataFrame({"Candies": [3, None, 5]})
df.fillna(0, inplace=True)  # Fills None with 0
print(df)  # Says: 3, 0, 5
```

3. **What is forward fill and backward fill in Pandas?**

   - **Forward fill**: Copy the last number forward (like "If Monday had 3, Tuesday gets 3 too").
   - **Backward fill**: Copy the next number back.

```
import pandas as pd
df = pd.DataFrame({"Candies": [3, None, 5]})
print(df.fillna(method="ffill"))  # Says: 3, 3, 5 (forward)
print(df.fillna(method="bfill"))  # Says: 3, 5, 5 (backward)
```

Hierarchical Indexing:

1. **What is hierarchical indexing in Pandas?**

   It's like having a table with extra layers—like sorting candies by kid AND day. It's called a MultiIndex, and it helps organize big data.

2. **How do you create a MultiIndex DataFrame?**

   Use multiple labels for rows or columns:

```
import pandas as pd
arrays = [["Max", "Max", "Lily", "Lily"], ["Mon", "Tue", "Mon", "Tue"]]
index = pd.MultiIndex.from_arrays(arrays, names=["Kid", "Day"])
df = pd.DataFrame({"Candies": [3, 4, 5, 2]}, index=index)
print(df)  # Shows a table with Kid and Day layers
```

Combining Datasets:

1. **What are different ways to combine datasets in Pandas?**
    - **Concat**: Stack tables up or side by side.
    - **Merge**: Mix tables by matching stuff (like names).
    - **Join**: Stick tables together by their row labels.
2. **Explain the difference between merge(), concat(), and join().**
    - **merge()**: Combines tables by matching columns (like pairing kids with their toys).
    - **concat()**: Glues tables together (stacking or side-by-side, no matching needed).
    - **join()**: Matches by row labels (like zippering two lists).

    ```
    import pandas as pd
    df1 = pd.DataFrame({"Name": ["Max"], "Candies": [3]})
    df2 = pd.DataFrame({"Name": ["Max"], "Toys": [2]})
    merged = pd.merge(df1, df2, on="Name")  # Matches by Name
    print(merged)  # Says: Name Max, Candies 3, Toys 2
    ```
3. **What is an inner join and an outer join?**
    - **Inner join**: Only keeps matches (if Max is in both tables, he stays).
    - **Outer join**: Keeps everything, even non-matches (fills with `NaN` if missing).

    ```
    import pandas as pd
    df1 = pd.DataFrame({"Name": ["Max"], "Candies": [3]})
    df2 = pd.DataFrame({"Name": ["Lily"], "Toys": [2]})
    print(pd.merge(df1, df2, on="Name", how="outer"))  # Says: Max 3 Na
    N, Lily NaN 2
    ```

Aggregation and Grouping:

1. **What is the purpose of `groupby()` in Pandas?**

It’s like sorting your toys into piles—like “How many candies per kid?” It groups data by something (like names) and then sums or averages it.

2. **How do you apply multiple aggregation functions in a groupby operation?**

   Use `.agg()` with a list of tricks (like sum AND average):

   ```
   import pandas as pd
   df = pd.DataFrame({"Kid": ["Max", "Max", "Lily"], "Candies": [3, 4, 5]})
   result = df.groupby("Kid").agg(["sum", "mean"])
   print(result)  # Shows sums and averages per kid
   ```

3. **Provide an example of grouping data and applying an aggregation function.**

   Here’s counting candies per kid:

   ```
   import pandas as pd
   df = pd.DataFrame({"Kid": ["Max", "Max", "Lily"], "Candies": [3, 4, 5]})
   grouped = df.groupby("Kid")["Candies"].sum()
   print(grouped)  # Says: Max 7, Lily 5
   ```

Fun Example Code:

Here’s a little Pandas candy party:

```
import pandas as pd

# My candy table
data = {"Kid": ["Max", "Lily", "Max"], "Candies": [3, None, 4]}
df = pd.DataFrame(data)
print("Before:", df)

# Fill missing candies
df["Candies"] = df["Candies"].fillna(0)
print("After filling:", df)

# How many candies per kid?
```

```
total = df.groupby("Kid")["Candies"].sum()
print("Total per kid:", total)
```

This will show:

```
Before: Kid Candies
0 Max 3.0
1 Lily NaN
2 Max 4.0
After filling: Kid Candies
0 Max 3.0
1 Lily 0.0
2 Max 4.0
Total per kid: Kid
Lily 0.0
Max 7.0
```