**pandas  
1. Why is Python Preferred for Data Analysis?**

**Key Reasons:**

* **Ease of Learning** – Simple syntax makes it beginner-friendly.
* **Versatile Libraries** – Libraries like **Pandas, NumPy, Matplotlib, and Seaborn** streamline data analysis.
* **Community Support** – A large Python community ensures access to resources and troubleshooting.
* **Integration & Automation** – Python integrates easily with databases and supports automation for repetitive tasks.
* **Industry Demand** – Python is widely used, making it a **valuable job skill**.

**Tip:** Be prepared to **explain why you chose Python over other languages** like R or SQL in an interview.

**2. Difference Between List, Tuple, and Dictionary**

* **Lists** – Mutable, ordered collections. Example: my\_list = ["pen", "notebook", "eraser"]
* **Tuples** – Immutable, ordered collections. Example: my\_tuple = ("pen", "notebook", "eraser")
* **Dictionaries** – Key-value pairs for quick lookups. Example: my\_dict = {"pen": "blue", "book": "math"}

**Tip:** Use **tuples** when data should remain constant and **dictionaries** when quick lookups are required.

**3. Handling Missing Values in Pandas**

* **Remove Missing Values**: df.dropna()
* **Fill Missing Values**: df.fillna(value) (e.g., using mean, median, or zero)

**Tip:** Be prepared to **justify why you are dropping or imputing values**—losing too much data can affect analysis.

**4. Lambda Functions in Python**

Lambda functions are **anonymous functions** used for **short-term operations**:

python

CopyEdit

square = lambda x: x \* x

print(square(5)) # Output: 25

Common Use Cases:

* **Sorting**: sorted(data, key=lambda x: x[1])
* **Quick calculations** without defining full functions

**Tip:** Use lambda functions for **simple, single-use operations** to make code cleaner.

**5. List Comprehension**

**Traditional Approach:**

python

CopyEdit

squares = []

for i in range(11):

if i % 2 == 0:

squares.append(i \*\* 2)

**List Comprehension Equivalent:**

python

CopyEdit

squares = [i \*\* 2 for i in range(11) if i % 2 == 0]

**Tip:** Use **list comprehensions** for cleaner, **more readable** code.

**6. Deep vs. Shallow Copy**

* **Shallow Copy (copy.copy())** – Creates a new object but references the same nested data.
* **Deep Copy (copy.deepcopy())** – Creates a completely independent clone of the object.

**Tip:** If modifying nested structures, use **deep copy** to avoid unintended changes.

**7. PEP 8 (Python Coding Standards)**

PEP 8 is **Python's official style guide**, ensuring:

* **Readability** – Indentation (4 spaces), naming conventions (snake\_case for variables).
* **Collaboration** – Consistent formatting across teams.
* **Maintainability** – Clean, structured code is easier to update.

**Tip:** Mention **PEP 8** in an interview to show you follow coding best practices.

**8. Merging DataFrames in Pandas**

python

CopyEdit

merged\_df = pd.merge(df1, df2, on="common\_column", how="inner")

Types of Joins:

* **Inner Join** – Only common data is kept.
* **Outer Join** – All data is kept (fills missing values with NaN).
* **Left/Right Join** – Keeps all data from one dataset while merging the second dataset.

**Tip:** Explain **why you choose a specific join type** based on the dataset.

**9. NumPy vs. Regular Python Lists**

* **NumPy Arrays** are **faster and more memory-efficient** than Python lists.
* **Supports Vectorized Operations** (e.g., numpy\_array \* 2 applies to all elements).

**Tip:** Use **NumPy** for numerical computations instead of regular lists.

**10. Improving Python Performance**

* **Use Efficient Data Structures** (e.g., deque over lists for fast appends).
* **Vectorized Operations** (Use NumPy/Pandas instead of loops).
* **Avoid Global Variables** (Accessing them is slow).
* **Use Built-in Functions** (They are optimized for performance).

**Tip:** If your script is slow, **profile the bottlenecks** using cProfile before optimizing.

**11. GroupBy in Pandas**

Grouping and aggregating data:

python

CopyEdit

df.groupby("Category")["Sales"].sum()

Use cases:

* **Summing Sales by Category**
* **Counting Occurrences by Group**

**Tip:** Use groupby() for **data summarization** and avoid manual loops.

**12. Data Visualization with Matplotlib & Seaborn**

* **Matplotlib** – Low-level, customizable visualization library.
* **Seaborn** – High-level statistical plots built on Matplotlib.

Example:

python

CopyEdit

import seaborn as sns

import matplotlib.pyplot as plt

sns.barplot(x="category", y="sales", data=df)

plt.show()

**Tip:** Use **Seaborn** for visually appealing plots with minimal code.

**13. Preventing Overfitting in Machine Learning**

* **More Data** – Helps models generalize better.
* **Simpler Models** – Reduce complexity (e.g., fewer features).
* **Cross-validation** – Rotates data through training/testing sets.
* **Regularization** – Adds penalty to over-complex models.

**Tip:** Explain overfitting as **memorization instead of learning patterns**.

**14. Data Cleaning Steps in Pandas**

1. **Remove Duplicates**: df.drop\_duplicates()
2. **Handle Missing Values**: df.fillna(value) or df.dropna()
3. **Correct Data Types**: df.astype()
4. **Remove Outliers**: Use **z-score filtering or IQR method**

**Tip:** Discuss **data validation techniques** used in your projects.

**15. Splitting Data into Training & Test Sets**

python

CopyEdit

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Tip:** Mention **stratified sampling** for imbalanced datasets.

**16. SQL in Python (Pandas & SQLite)**

Fetching data from SQL into Pandas:

python

CopyEdit

import sqlite3

import pandas as pd

conn = sqlite3.connect("database.db")

df = pd.read\_sql\_query("SELECT \* FROM sales\_data", conn)

**Tip:** Be ready to discuss **JOINs and aggregate functions** in SQL.

**17. Error Handling in Python (try-except-finally)**

python

CopyEdit

try:

x = 10 / 0 # Will cause an error

except ZeroDivisionError:

print("Cannot divide by zero")

finally:

print("Execution finished")

**Tip:** Always **handle specific exceptions** rather than generic except:.

**Final Takeaway**

This video provides **a solid Python interview prep guide**, covering:  
✅ Core Python concepts  
✅ Pandas for data manipulation  
✅ NumPy for fast computations  
✅ Data visualization techniques  
✅ Machine learning essentials

**Tip:** **Practice coding these topics** and be prepared to **explain your thought process** clearly.  
  
Pandas is a popular **Python software** toolkit for performing high-level data analysis and manipulating the data. Pandas provide data structures and other advanced tools to run complicated data applications, allowing analysts and data engineers to alter time series characteristics, tables, and other factors. The Pandas interview questions revolve around the tool's features, data structures, and functions in Python interviews.

Pandas is a popular Python data munging tool. This data analysis package can handle a wide range of data types. We've compiled a list of the most important **Panda Interview Questions and Answers** in this article.

Panda Interview Questions and Answers2024 (Updated) weblog had been created into the following stages; they are:

* [**Basic**](https://mindmajix.com/pandas-interview-questions#basic)
* [**Advanced**](https://mindmajix.com/pandas-interview-questions#advanced)
* [**FAQ's**](https://mindmajix.com/pandas-interview-questions#faqs)

**Commonly Asked Pandas Interview Questions**

* [**What is the basic use of pandas?**](https://mindmajix.com/pandas-interview-questions#basic-use)
* [**Is Panda a module or library?**](https://mindmajix.com/pandas-interview-questions#module-or-library)
* [**What is the full form of pandas?**](https://mindmajix.com/pandas-interview-questions#full-form)
* [**What type of inputs are accepted by pandas?**](https://mindmajix.com/pandas-interview-questions#inputs-type)
* [**Which are the data structures available with Pandas?**](https://mindmajix.com/pandas-interview-questions#data-structures)

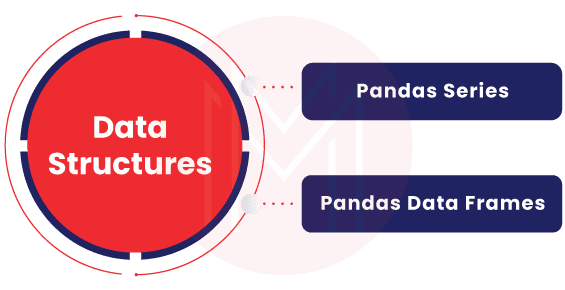
**Basic Pandas Interview Questions and Answers**

**1. Define Python Pandas?**

**Ans:**Pandas refer to a data analysis and manipulation software library built specifically for Python. Wes McKinney designed Pandas, an open-source, cross-platform library. It was first released in 2008, and it included data structures and procedures for manipulating numerical and time-series data. Pandas can be installed with the pip package manager or the Anaconda distribution. Pandas make doing machine learning algorithms on tabular data a breeze.

**2. Mention different types of Data Structures in Panda?**

**Ans: Series** and **DataFrames**are the two types of data structures that the Pandas library supports. Numpy serves as the foundation for both data structures. A DataFrame is a two-dimensional data structure in Pandas, while a Series is a one-dimensional data structure. A panel, a three-dimensional data structure that includes items, a major axis, and a minor axis, is another axis label.



**3. What is Python Panda used for?**

**Ans:** Pandas is a data manipulation and analysis software library for the Python programming language. It includes data structures and methods for manipulating numerical tables and time series, in particular. Pandas is open-source software licensed under the BSD three-clause license.

|  |
| --- |
|  |

**4. List out the key features of Panda Library?**

**Ans:** The pandas library has a number of features, some of which are shown here.

* Memory Efficient
* Time Series
* Reshaping
* Data Alignment
* Merge and join

**5. Give a brief description about time series in Panda?**

**Ans:**A time series is an organized collection of data that depicts the evolution of a quantity through time. Pandas have a wide range of capabilities and tools for working with time-series data in all fields.

**Supported by pandas:**

* Analyzing time-series data from a variety of sources and formats
* Create time and date sequences with preset frequencies.
* Date and time manipulation and conversion with timezone information
* A time series is resampled or converted to a specific frequency.
* Calculating dates and times using absolute or relative time increments is one way to

**6. Explain how to create and copy a series in Pandas?**

**Ans:** To copy the series in pandas:  
pandas. series.copy

series.copy (deep=True)

pandas. series. copy. Make a significant copy of everything, including the data and indices. Deep=False copies of neither the indices nor the data. When deep = True, data is transferred, only the connection to the object is emulated recursively, not the actual Python objects.

**7. Characterize the Data Frames in Pandas?**

**Ans:**A DataFrame is a panda-specific lewis structure that functions with a two-dimensional display with tomahawks (rows and columns). A DataFrame is a typical way of storing data that has two separate indices, namely a row index and a column index. It includes the following characteristics:

Columns such as int and bool are heterogeneous.

It's commonly thought of as a term reference for a series structure that includes both rows and columns. If there are columns, it is denoted as "columns," and if there are lines, it is denoted as "index."

**Syntax:** import pandas as pd  
df=pd.Dataframe()

**8. Explain how to create a series from dict in Pandas?**

**Ans:** A Series is a one-dimensional designated array that can hold any form of data (python objects, strings, integers, floating-point numbers, etc.). It's important to understand that, unlike Python lists, a series always contains the same type of data.

Let's look at how to make a Panda Series using the Dictionary.

The Series () method is used without the index parameter.

**9. Explain about the operation on Series in Pandas?**

**Ans:**The Pandas Series is a one-dimensional classified array that may hold any type of data (python objects, strings, integers, floating-point numbers, etc.). The axis identifiers are referred to as an index. The Pandas Series is merely a column in an excel spreadsheet.

**Putting Together a Pandas Series-**

A Pandas Series is built in the real world by loading datasets from existing storage, which can be a SQL database, a CSV file, or an Excel file. Pandas Series can be made from lists, dictionaries, and other things. A series can be developed in a number of ways; here are a few examples: cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval cheval

Creating a series from an array: To construct a series from an array, we must first load a NumPy module and then use its array() functions.

# import pandas as pd  
import pandas as pd  
   
# import numpy as np  
import numpy as np  
   
# simple array  
data = np.array([‘M’,’I’,’N’,’D’,’M’,’A’,’J’,’I’,’X’])  
   
ser = pd.Series(data)  
print(ser)  
   
**Output: MINDMAJIX**

|  |
| --- |
|  |

**10. Explain different ways of creating Data Frames in Panda?**

**Ans:** A data frame can be created in 3 different ways:  
   
**By making use of lists:**  
   
d = [[‘a’, 2], [‘b’, 3], [‘c’, 4]]  
   
**Creating the Pandas Dataframe:**

df = pd.DataFrame (d, columns = [‘Strings’, ‘Integer’])  
print(df)  
   
**By making use of a dictionary of lists:**

All of the arrays in a data frame made from a list's dictionaries must be the same length. If the list is passed, the running time of the list will match the running time of the shows. If no document is specified, the items will be a range (n), where n is the array length, as is conventional.  
   
**By using arrays:**

import pandas as pd  
d = {‘Name’:[‘XYZ, ‘ABC’, ‘DEFC’, ‘ASWE’], ‘marks’:[85, 80, 75, 70]}  
df = pd.DataFrame(d, index =[‘first’, ‘second’, ‘third’, ‘fourth’])  
print(df)

**11. Explain how to create empty DataFrames in Panda?**

**Ans:** To make a Pandas data frame that is fully empty, perform the following:  
   
import pandas as pd   
MyEmptydf ()= pd.DataFrame  
   
This will result in a data frame that has no columns or rows.  
   
We do the following to construct an empty dataframe with three empty columns (columns A, B, and C):  
   
 df= pd.DataFrame(columns=['A', 'B', 'C'])

**12. How will you add a column to the Existing Data Frames in Panda?**

**Ans:** Import pandas as a package, import pandas as pd  
   
# Define a dictionary containing employee data.  
Employee ={ ‘Emp\_name’:{‘Name’: [‘Ravi’, ‘Roshan', ‘Vinod’, ‘Sailu’],  
                    ‘ Emp\_id’: [123, 234, 145, 125],  
                     ‘Emp\_qualification’= [‘Msc’, ‘BA’, ‘MBA’, ‘Msc’]}  
   
# Convert the dictionary into DataFrame  
df = pd.DataFrame(Employee)  
   
# Declare a list that is to be converted into a column  
Emp\_address = [‘Hyderabad’, ‘Delhi’, ‘Lucknow’, ‘Vijayawada’]  
   
# Using ‘Address’ as the column name  
# and equating it to the list  
df[‘Address’] = Emp\_address  
   
# Observe the result  
df  
   
**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Emp\_name** | **Emp\_id** | **Emp\_qualification** | **Emp\_address** |
| 0 | Ravi | 123 | MSC | Hyderabad |
| 1 | Roshan | 234 | BA | Delhi |
| 2 | Vinod | 145 | MBA | Lucknow |
| 3 | Sailu | 125 | MSC | Vijayawada |

**13. Tell us now how to retrieve a single column from a Panda Dataframe?**

Ans: Use the query $django-admin.py to start a Django project, and then use the following queries:

Project  
\_init\_.py  
manage.py  
settings.py  
urls.py

**14. Explain about Categorical Data in Pandas?**

**Ans:**Categorical is a data type in Pandas that corresponds to categorical variables in statistics. A categorical variable has a limited and usually fixed, set of possible values (categories; levels in R). Gender, social class, blood type, national affiliation, observation time, or rating using Likert scales are some examples. All categorical data values are either in categories or np. nan.

**In the following situations, categorical data is useful:**  
   
A string parameter with a small number of distinct values. Transforming a string parameter to a category variable can help you save memory.  
   
A variable's lexical order differs from its analytical order ("one," "two," and "three"). Indexing and min/max will utilize the analytical order rather than the lexical order after transforming to a categorical and providing order on the categories.  
   
To indicate to other Python libraries that this column is a categorical variable (so that appropriate statistical technique or plot types can be used).

**15. Explain about Multi Indexing in pandas?**

**Ans:**Multiple indexing is classified as fundamental indexing because it simplifies information inspection and control, especially when dealing with higher-dimensional data. It also allows us to store and handle data in lower-dimensional data structures like series and dataframes with an unlimited number of measurements.

**Advanced Pandas Interview Questions and Answers**

**16. Explain about Pandas index?**

**Ans:** Indexing in Pandas is the process of extracting specific rows and columns of data from a DataFrame. Indexing could simply be selecting all of the rows and some of the columns, or part of the rows and all of the columns, along with some of each row and column. Indexing is often referred to as subset selection.

Pandas Indexing with [],.loc[],.iloc[], and.ix []  
There are numerous methods for obtaining the objects, elements, items, rows, and columns from a data frame. In Pandas, some indexing methods can be used to retrieve an object/element/item from a data frame. These indexing systems look to be extremely similar. However, they perform significantly differently. The Pandas support four methods of multi-axis indexing, which are as follows:

* **Dataframe. []:** This method is also known as the indexing operator.
* **Dataframe. loc []:**This method is used for labels.
* **Dataframe.iloc[] :** This method is utilized for integer or position based
* **Dataframe. ix[]:**This function is utilized for both integer and label based  
  They are referred to collectively as indexers. All of those are, by far, the most popular methods of indexing data. These four functions assist in retrieving the object/elements/items, rows, and columns from a DataFrame.

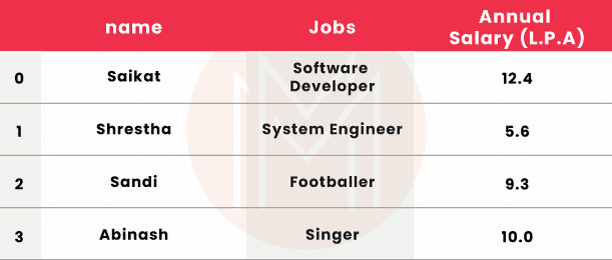
|  |
| --- |
| *Also Read:*[***Python Partial Function Using Functools***](https://mindmajix.com/python-partial-function) |

**17. Explain about Reindexing in Pandas?**

**Ans:** The DataFrame is reindexed to adhere to a new index with configurable filling logic. It inserts NA/NaN in the areas where the elements are missing from the previous index. Unless the new index is constructed as equivalent to the present one, in which case the copy value becomes false. It is used to modify the index of the dataframe's rows and columns.

**18. Can you explain multi-indexing columns in Pandas?**

**Ans:** Because it involves data manipulation and analysis, multiple indexing is characterized as vital indexing. This is certainly relevant when operating with hyperdimensional data.  It also allows us to store and modify data in lower-dimensional data structures like  DataFrame and series with an indefinite number of dimensions.  
   
**Multiple Index Columns**  
Two columns will be used as index columns in this case. The drop option is used to remove a column, whereas the append attribute is used to append given columns to an index column that already exists.  
   
**Example:**  
   
# importing pandas library from  
# python  
import pandas as pd  
   
# Creating data  
Information = {'name': ["Saikat", "Shrestha", "Sandi", "Abinash"],  
                  
            'Jobs': ["Software Developer", "System Engineer",  
                        "Footballer", "Singer"],  
                  
            'Annual Salary(L.P.A)': [12.4, 5.6, 9.3, 10]}  
   
# Data Framing the whole data  
df = pd.DataFrame(dict)  
   
# Showing the above data  
print(df)  
   
**Output:**

****

**19. What is meant by set the index in Pandas?**

**Ans:**Python is an excellent language for analyzing data, particularly with its vast ecological community of data-driven Python packages. Pandas is another of those packages, and it makes data import and analysis considerably easier.

Pandas set\_index () is a function for modifiying the index of a data frame from a data frame, series, or list. The index column can also be set while creating a data frame. However, because a data frame might be made up of two or more data frames, the index can be altered later using this method.

**Syntax:**

DataFrame.set\_index(keys, drop=True, append=False,

inplace=False, verify\_integrity=False

**Parameters:**

keys: The name of the column or a list of column names.  
If True, **drop**is a Boolean value that removes the index column.  
If True, the column is **appended**to the existing index column.  
**Inplace**, If True, the changes are made in the data frame.  
If True, **verify\_integrity** will check the new index column for duplicates.

**Example:**

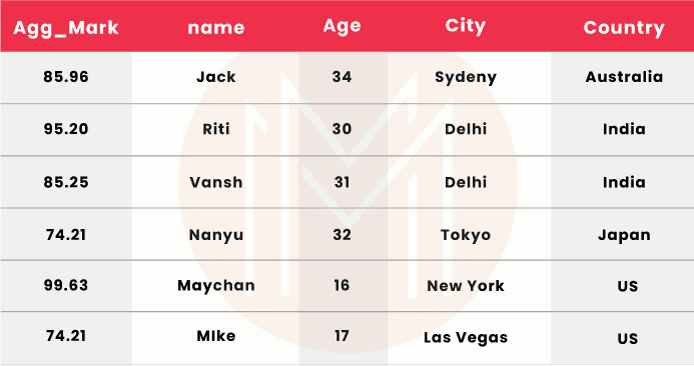
# importing pandas library  
import pandas as pd

# creating and initializing a nested list  
students = [['jack', 34, 'Sydeny', 'Australia',85.96],  
            ['Riti', 30, 'Delhi', 'India',95.20],  
            ['Vansh', 31, 'Delhi', 'India',85.25],  
            ['Nanyu', 32, 'Tokyo', 'Japan',74.21],  
            ['Maychan', 16, 'New York', 'US',99.63],  
            ['Mike', 17, 'las vegas', 'US',47.28]]

# Create a DataFrame object  
df = pd.DataFrame(students,  
                    columns=['Name', 'Age', 'City', 'Country','Agg\_Marks'],  
                        index=['a', 'b', 'c', 'd', 'e', 'f'])

# here we set Float column 'Agg\_Marks' as index of data frame  
# using dataframe.set\_index() function  
df = df.set\_index('Agg\_Marks')

# Displaying the Data frame  
df



**20. Explain how to reset the index in pandas?**

**Ans:**Pandas is a one-dimensional ndarray with identifiers on the axes. The identifiers do not have to be distinct, but they must be of the hashable type. The entity allows both label-based and integer indexing, as well as a set of techniques for handling the index.

The pandas function series.reset\_index () creates a reinvigorated series or data frame with the index reset. This is useful when an index must be utilized as a column.

**Syntax:**

reset\_index(level=None, drop=False, inplace=False, col\_level=0, col\_fill='')

**Parameters:**

level: int, str, tuple, or list None(default)  
Only the specified levels should be removed from the index. By default, all levels are removed.

drop: default False, bool  
Inserting indexes into data frame columns is not recommended. This returns the index to its original integer value.

inplace: False by default bool  
Modify the existing DataFrame (do not create a new object).

col\_level: default 0 for int and str  
This determines the level the labels are inserted into if the columns have several levels. It is inserted into the first level by default.

col\_fill: default object  
Evaluate how the other levels are named if the columns have different levels. If there is no value, the index name is replicated.

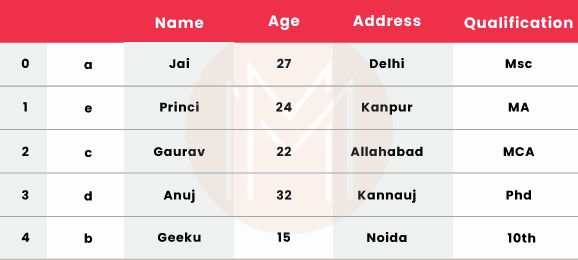
# Import pandas package  
import pandas as pd  
      
# Define a dictionary containing employee data  
data = {'Name':['Jai', 'Kanth', 'Vinod, 'Seeraj', 'Kokila'],  
        'Age':[27, 26, 23, 30, 25],  
        'Address':['Delhi', 'Gujart', 'Hyderabad', 'Vizag', 'Noida'],  
        'Qualification':['MCA', 'Ms', ‘BA’, 'Phd', 'MS'] }

index = {'a', 'b', 'c', 'd', 'e'}

# Convert the dictionary into DataFrame  
df = pd.DataFrame(data, index)

# Make Own Index as index  
# In this case default index is exist  
df.reset\_index(inplace = True)

df



**21. Explain about Data operations in Pandas?**

**Ans:** There are several useful data operations for DataFrame in Pandas, which are as follows:

**-> Row and column selection:**

We can retrieve any row and column of the DataFrame by specifying the names of the rows and columns. It is one-dimensional and is regarded as a Series when you select it from the DataFrame.

**-> Filter Data:**

By using some of the boolean logic in DataFrame, you may filter the data.

**-> Null values:**

When no data is being sent to the items, a Null value can appear. There may be no values in the respective columns, which are commonly represented as NaN. Pandas provide several useful functions for identifying, deleting, and changing null values in Data Frames. The following are the functions:

* **isnull():** isnull 's job is to return true if either of the rows have null values.
* **notnull():** It is the inverse of the isnull() function, returning true values for non-null values.
* **dropna():** This function evaluates and removes null values from rows and columns.
* **fillna():** It enables users to substitute other values for the NaN values.
* **replace():**It's a powerful function that can take the role of a regex, dictionary, string,  series,  and more.
* **interpolate():**It's a useful function for filling null values in a series or data frame.

**-> String Operation:**

Pandas provide a set of string functions for working with string data while ignoring missing/NaN values. The .str. option can be used to conduct various string operations. The following are the functions:

* **lower():** Any strings in the  index or series are converted to lowercase letters.
* **upper():** Any strings in the  index or series are converted to uppercase letters.
* **strip():**This method eliminates spacing from every string in the Series/index, along with a new line.
* **split(' '):**It's a method that separates a string according to a pattern.
* **cat(sep=' '):**With a defined separator, it concatenates series/index items.
* **contains(pattern):** If a substring is available in the current element, it returns True; otherwise, it returns False.
* **replace(a,b):** It substitutes the value b for the value a.
* **repeat(value):** Each element is repeated a defined multiple times.
* **count(pattern):**It returns the number of times a pattern appears in each element.
* **startswith(pattern):** If all of the components in the series begin with a pattern, it returns True.
* **endswith(pattern):** If all of the components in the series terminate in a pattern, it returns True.
* **find(pattern):** It can be used to return the pattern's first occurrence.
* **findall(pattern):**It gives you a list of all the times the pattern appears.
* **swapcase:** It is used to switch the lower/upper case.
* **islower():** If all of the characters in the Series/Index string are lowercase, it returns True. Otherwise, False is returned.
* **isupper():** If all of the characters in the Series/Index string are uppercase, it returns True. Otherwise, False is returned.
* **isnumeric():**If all of the characters in the Series/Index string are numeric, it returns True. Otherwise, False is returned.

**-> Count Values:**

Using the 'value counts()' option, this process is used to count the overall possible combinations.

**22. How Can A Dataframe Be Converted To An Excel File?**

**Ans:**Using the to excel () function, we can export the data frame to an excel file. We must mention the target file name to write a single object to an excel file. If we wish to write to many sheets, we must build an ExcelWriter object with the target filename and the sheet in the file that we want to write to.

***Visit here to learn***[***Python Training in Bangalore***](https://mindmajix.com/python-training-bangalore)

**23. How To Format The Data in Your Pandas DataFrame?**

**Ans:**Almost all of the time, you'll want to be ready to execute operations on the absolute measurements in your data frame.

**Replacing All String Occurrences in a DataFrame:**

The Replace() method can be used to easily replace specific strings in your data frame. Simply pass the values you are trying to enhance, accompanied by the values you would like to substitute them with.  
It's worth noting that there's a regex argument that can come in handy when dealing with unusual string combinations. In a nutshell, replace() method is used when you wish to substitute values or strings in your DataFrame with those from elsewhere.

**Removing Parts From Strings in the Cells of Your DataFrame:**

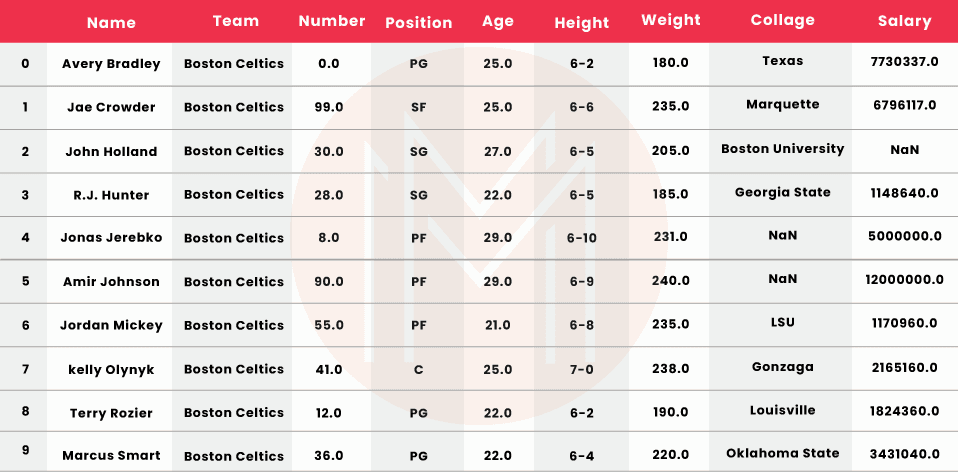
Removing unnecessary strings is a time-consuming task. Fortunately, there is a remedy! You apply the lambda function to each element or element-by-element of the column using map() on the column result. The function takes the string value and removes the + or — on the left, as well as any of the six aAbBcC on the right.

**Splitting Text in a Column into Multiple Rows in a DataFrame:**

It's difficult to divide your text into many rows.

**Applying A Function to Your Pandas DataFrame’s Columns or Rows:**

You might want to use a function to alter the information in the DataFrame. The code pieces illustrate how to implement a method to a DataFrame.



**24. Explain about Data Aggregation in Pandas?**

**Ans:**To implement any aggregation method across one or more columns, use the Dataframe. aggregate() method. Use strings, callables, dictionaries, or a collection of strings to aggregate. The following are the most common aggregations:

* **Sum:**This method returns the sum of the values for the requested column.
* **Min:**This method returns the minimum value for the requested column.
* **Max:**This method returns the maximum value for the requested column.

**Syntax:** DataFrame.aggregate(func, axis=0, \*args, \*\*kwargs)

**function:**string, callable, list, or dictionary of callables. Use this function to aggregate the data. If a function is handed a data frame, it must either work or be allowed to pass to the data frame. apply. If the variables are DataFrame column names, you can give a dict to a DataFrame.  
the axis (default 0) 1 or 'columns', 0 or 'index' Apply the method to each column with a 0 or index. 1 or 'columns': for each row, apply the function.

Let us see an example for data aggregation:

# importing pandas package  
import pandas as pd

# making data frame from csv file  
df = pd.read\_csv("nba.csv")

# printing the first 10 rows of the dataframe  
df[:10]

|  |
| --- |
| *Explore*[***Python Sample Resumes***](https://mindmajix.com/python-sample-resumes)*! Download & Edit, Get Noticed by Top Employers!* |

**25. What is the use of GroupBy Pandas?**

**Ans:**The data is divided into groups using GroupBy. It organizes the data according to certain parameters. Labels are mapped to group names when using grouping. It has a lot of different versions that can be made using the parameters, and it makes separating data a breeze.

**Syntax:**DataFrame.groupby(by=None, axis=0, level=None, as\_index=True, sort=True, group\_keys=True, squeeze=False, \*\*kwargs)

**26. What is Pandas Numpy?**

**Ans:**Pandas Numpy is an open-source Python package that would be used to work with a huge number of datasets. It includes a robust N-dimensional array object as well as complicated mathematical algorithms for data processing with Python.

Fourier transformations, linear programming, and pseudo-random capabilities are among the prominent features provided by Numpy. It also includes integrated tools for C/C++ and Fortran programming.

**27. What is Vectorization in Python pandas?**

**Ans:**The procedure of executing operations on the full array is known as vectorization. This is intended to limit the number of iterations that the methods do. Pandas have a series of vectorized methods, such as string functions and aggregations, that are optimized for use with series and dataframes. As a result, it is preferable to use vectorized pandas methods to perform the tasks quickly.

**28. How will you combine different Data Frames in Panda?**

**Ans:**Following are the ways to combine different Data Frames in panda:

**-> append() method:**This is used to horizontally stack the dataframes.

**Syntax:**df1.append(df2)

**-> concat() method:** This is used to sequentially stack data frames. This works best because the data frames have the same fields and columns.

**Syntax:**pd.concat([df1, df2])

**-> join() method:**This is used to extract data from different dataframes that have one or more common columns.

**Syntax:**df1.join(df2)

**29. How can you iterate over the Data frame in Pandas?**

**Ans:** Iterating over a DataFrame in pandas for loop can be merged with an iterrows () call.

|  |
| --- |
| *Read Related Article:*[***Python for loop***](https://mindmajix.com/python-for-loop) |

**30. What Method Will You Use To Rename The Index Or Columns Of Pandas Dataframe?**

**Ans:**The .rename method is used to rename DataFrame index values or columns.

**Frequently Asked Pandas Interview Questions**

**1. What is the basic use of pandas?**

**Ans:**Pandas is a popular open-source Python library used for analyzing the data, Machine learning and data science.

**2. Is Panda a module or library?**

**Ans:** Pandas is a programming interface for Python. It offers ready-to-use high-performance data analysis tools and data structures. Pandas is a Python package for analyzing the data and data science that runs on top of NumPy.

**3. What is the full form of pandas?**

**Ans:** The acronym for "Python Data Analysis Library" is "Python Data Analysis Library." The phrase comes from the multiple linear regression term "panel data," which applies to dimensional discrete classes "Pandas," according to the Wikipedia article. However, I feel it is a catchy moniker for a fantastic Python package!

**4. What type of inputs are accepted by pandas?**

**Ans:** Like Series, DataFrame accepts many different kinds of input:

* Dict of 1D ndarrays, lists, dicts, or Series.
* 2-D numpy. ndarray.
* Structured or recorded ndarray.
* A Series.
* Another DataFrame.

**5. Which are the data structures available with Pandas?**

**Ans:** Series and Data Frames are the two basic types of data structures supported by Pandas. Series is a one-dimensional data structure, whereas DataFrames are two-dimensional data structures.

**Conclusion**

Since we've gone over all of the most significant**Panda Interview Questions and Answers**, it's crucial to remember that we should constantly remember these concepts when coding.

**Pandas for Data Analysis Interviews**

**pandas** interview questions commonly asked in data analysis interviews. The instructor walks through various **real-world coding exercises** using a sample dataset containing **countries, capitals, populations, areas, continents, and GDP values**. Below is a structured summary of the key topics covered:

**1. Setting Up the Environment**

* Uses **VS Code**, but mentions alternatives like Jupyter Notebook, PyCharm, Google Colab.
* Ensures compatibility by installing pandas==2.0.3 to match the tutorial version.

**2. Sample Dataset**

* The dataset includes **Country, Capital, Population, Area, Continent, and GDP** in billion dollars.
* The instructor creates a pandas **DataFrame** from this dataset.

**3. Key Pandas Interview Questions & Solutions**

**Q1: Which country has the highest population density?**

* **Formula:** Population Density = Population / Area
* **Solution:**

python

CopyEdit

df['Population Density'] = df['Population'] / df['Area']

highest\_density\_country = df.loc[df['Population Density'].idxmax(), 'Country']

print(highest\_density\_country)

* **Answer:** **USA** (as per sample data).

**Q2: Total GDP of all countries in each continent**

* **Solution:**

python

CopyEdit

gdp\_by\_continent = df.groupby('Continent')['GDP'].sum().sort\_values(ascending=False)

print(gdp\_by\_continent)

* **Answer:** **North America** has the highest total GDP.

**Q3: Countries with GDP greater than the average GDP**

* **Solution:**

python

CopyEdit

avg\_gdp = df['GDP'].mean()

high\_gdp\_countries = df[df['GDP'] > avg\_gdp]['Country']

print(high\_gdp\_countries)

* **Answer:** **Only the USA** (due to a significantly high GDP value in the dataset).

**Q4: Ratio of total area of Europe to Asia**

* **Solution:**

python

CopyEdit

area\_by\_continent = df.groupby('Continent')['Area'].sum()

europe\_to\_asia\_ratio = area\_by\_continent['Europe'] / area\_by\_continent['Asia']

print(round(europe\_to\_asia\_ratio, 2))

* **Answer:** **2.65**

**Q5: If GDP increases by 10%, what is the new global GDP?**

* **Solution:**

python

CopyEdit

df['New GDP'] = df['GDP'] \* 1.1

total\_new\_gdp = df['New GDP'].sum()

print(total\_new\_gdp)

* **Answer:** **The total new GDP is calculated in billion dollars.**

**Q6: Which country has the lowest GDP per capita?**

* **Formula:** GDP per Capita = GDP / Population
* **Solution:**

python

CopyEdit

df['GDP per Capita'] = (df['GDP'] \* 1e9) / df['Population']

lowest\_gdp\_country = df.loc[df['GDP per Capita'].idxmin(), 'Country']

print(lowest\_gdp\_country)

* **Answer:** **Japan** (as per sample data).

**Q7: Percentage of the world’s population represented by each country**

* **Solution:**

python

CopyEdit

total\_population = df['Population'].sum()

df['World Population %'] = (df['Population'] / total\_population) \* 100

print(df[['Country', 'World Population %']])

* **Output:** Percentage distribution of each country’s population.

**Q8: Average area of countries in each continent**

* **Solution:**

python

CopyEdit

avg\_area\_by\_continent = df.groupby('Continent')['Area'].mean()

print(avg\_area\_by\_continent)

* **Output:** The average area of countries for each continent.

**Q9: Countries with population above the 75th percentile**

* **Solution:**

python

CopyEdit

population\_75th\_percentile = df['Population'].quantile(0.75)

high\_population\_countries = df[df['Population'] > population\_75th\_percentile]['Country'].tolist()

print(high\_population\_countries)

* **Answer:** **USA and Japan** (as per the dataset).

**Q10: Which continent has the highest average GDP per capita?**

* **Solution:**

python

CopyEdit

avg\_gdp\_per\_capita\_by\_continent = df.groupby('Continent')['GDP per Capita'].mean()

highest\_gdp\_continent = avg\_gdp\_per\_capita\_by\_continent.idxmax()

print(highest\_gdp\_continent)

* **Answer:** **Oceania** (as per sample data).

**Q11: Correlation between GDP and Population**

* **Solution:**

python

CopyEdit

correlation = df['GDP'].corr(df['Population'])

print(correlation)

* **Answer:** **0.98** (Strong Positive Correlation)

**Key Takeaways from the Video**

* Focused on **pandas functions** relevant to **data analysis interviews**.
* Covered **sorting, grouping, aggregating, filtering, and statistical calculations**.
* **Real-world problem-solving** approach to interpreting data.
* Demonstrated **Python code efficiency** using pandas built-in methods.

This video provides an **excellent practice set for mastering pandas for data analysis interviews** with real coding exercises! 🚀  
###

**Pandas Interview Questions**

Panda is a **FOSS (Free and Open Source Software)** Python library which provides high-performance data manipulation, in Python. It is used in various areas like data science and machine learning.

Pandas is not just a library, it's an essential skill for professionals in various domains, including finance, healthcare, and marketing. This library streamlines data manipulation tasks, offering robust features for data loading, cleaning, transforming, and much more. As a result, understanding Pandas is a key requirement in many data-centric job roles.

This **Panda interview question** for [data science](https://www.geeksforgeeks.org/what-is-data-science) covers **basic and advanced topics**to help you succeed with confidence in your upcoming interviews. We do not just cover theoretical questions, we also provide practical coding questions to test your hands-on skills. This is particularly beneficial for aspiring [Data Scientists](https://www.geeksforgeeks.org/data-scientist-salary-skills-and-job-roles) and [ML](https://www.geeksforgeeks.org/machine-learning)professionals who wish to demonstrate their proficiency in real-world problem-solving.

So, whether you are starting your journey in Python programming or looking to brush up on your skills, "**This Panda Interview Questions**" is your essential resource for acing those technical interviews.

**Pandas Interview Questions for Freshers**

**Q1. What are Pandas?**

[Pandas](https://www.geeksforgeeks.org/pandas-tutorial) is an open-source Python library that is built on top of the NumPy library. It is made for working with relational or labelled data. It provides various data structures for manipulating, cleaning and analyzing numerical data. It can easily handle missing data as well. Pandas are fast and have high performance and productivity.

**Q2. What are the Different Types of Data Structures in Pandas?**

The two data structures that are supported by Pandas are **Series** and **DataFrames**.

* **Pandas** [Series](https://www.geeksforgeeks.org/python-pandas-series) is a one-dimensional labelled array that can hold data of any type. It is mostly used to represent a single column or row of data.
* **Pandas**[DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe) is a two-dimensional heterogeneous data structure. It stores data in a tabular form. Its three main components are **data, rows,** and **columns**.

**Q3. List Key Features of Pandas.**

Pandas are used for efficient data analysis. The key features of Pandas are as follows:

* Fast and efficient data manipulation and analysis
* Provides time-series functionality
* Easy missing data handling
* Faster data merging and joining
* Flexible reshaping and pivoting of data sets
* Powerful group by functionality
* Data from different file objects can be loaded
* Integrates with NumPy

**Q4. What is Series in Pandas?**

Ans: A Series in Pandas is a one-dimensional labelled array. Its columns are like an Excel sheet that can hold any type of data, which can be, an integer, string, or Python objects, etc. Its axis labels are known as the **index**. Series contains homogeneous data and its values can be changed but the size of the series is immutable. A series can be created from a Python tuple, list and dictionary. The syntax for creating a series is as follows:

import pandas as pd  
series = pd.Series(data)

**Q5. What are the Different Ways to Create a Series?**

Ans: In Pandas, a series can be created in many ways. They are as follows:

**Creating an Empty Series**

An empty series can be created by just calling the **pandas.Series()** constructor.

*# import pandas as pd*

**import** **pandas** **as** **pd**

*# Creating empty series*

print(pd.Series())

**Output:**

Series([], dtype: float64)

**Creating a Series from an Array**

In order to create a series from the NumPy array, we have to import the NumPy module and have to use the **array()** function.

*# import pandas and numpy*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*# simple array*

data = np.array(['g', 'e', 'e', 'k', 's'])

*# convert array to Series*

print(pd.Series(data))

**Output:**

0 g  
1 e  
2 e  
3 k  
4 s  
dtype: object

**Creating a Series from an Array with a custom Index**

In order to create a series by explicitly proving the index instead of the default, we have to provide a list of elements to the index parameter with the same number of elements as it is an array.

*# import pandas and numpy*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*# simple array*

data = np.array(['g', 'e', 'e', 'k', 's'])

*# providing an index*

ser = pd.Series(data, index=[10, 11, 12, 13, 14])

print(ser)

**Output:**

10 g  
11 e  
12 e  
13 k  
14 s  
dtype: object

**Creating a Series from a List**

We can create a series using a Python list and pass it to the Series() constructor.

*# import pandas*

**import** **pandas** **as** **pd**

*# a simple list*

list = ['g', 'e', 'e', 'k', 's']

*# create series form a list*

print(pd.Series(list))

**Output:**

0 g  
1 e  
2 e  
3 k  
4 s  
dtype: object

**Creating a Series from Dictionary**

A Series can also be created from a Python dictionary. The keys of the dictionary as used to construct indexes of the series.

*# import pandas*

**import** **pandas** **as** **pd**

*# a simple dictionary*

dict = {'Geeks': 10,

'for': 20,

'geeks': 30}

*# create series from dictionary*

print(pd.Series(dict))

**Output:**

Geeks 10  
for 20  
geeks 30  
dtype: int64

**Creating a Series from Scalar Value**

To create a series from a Scalar value, we must provide an index. The Series constructor will take two arguments, one will be the scalar value and the other will be a list of indexes. The value will repeat until all the index values are filled.

*# import pandas and numpy*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*# giving a scalar value with index*

ser = pd.Series(10, index=[0, 1, 2, 3, 4, 5])

print(ser)

**Output:**

0 10  
1 10  
2 10  
3 10  
4 10  
5 10  
dtype: int64

**Creating a Series using NumPy Functions**

The Numpy module's functions, such as **numpy.linspace()**, and **numpy.random.randn()** can also be used to create a Pandas series.

*# import pandas and numpy*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*# series with numpy linspace()*

ser1 = pd.Series(np.linspace(3, 33, 3))

print(ser1)

*# series with numpy linspace()*

ser2 = pd.Series(np.random.randn(3))

print("**\n**", ser2)

**Output:**

0 3.0  
1 18.0  
2 33.0  
dtype: float64  
 0 0.694519  
1 0.782243  
2 0.082820  
dtype: float64

**Creating a Series using the Range Function**

We can also create a series in Python by using the range function.

*# import pandas*

**import** **pandas** **as** **pd**

print(pd.Series(range(5)))

**Output:**

0 0  
1 1  
2 2  
3 3  
4 4  
dtype: int64

**Creating a Series using List Comprehension**

Here, we will use the Python list comprehension technique to create a series in Pandas. We will use the range function to define the values and a for loop for indexes.

*# import pandas*

**import** **pandas** **as** **pd**

ser = pd.Series(range(1, 20, 3),

index=[x **for** x **in** 'abcdefg'])

print(ser)

**Output:**

a 1  
b 4  
c 7  
d 10  
e 13  
f 16  
g 19  
dtype: int64

**Q6. How can we Create a Copy of the Series?**

Ans: In Pandas, there are two ways to create a copy of the Series. They are as follows:

**Shallow Copy**is a copy of the series object where the indices and the data of the original object are not copied. It only copies the references to the indices and data. This means any changes made to a series will be reflected in the other. A shallow copy of the series can be created by writing the following syntax:

ser.copy(deep=False)

**Deep Copy**is a copy of the series object where it has its own indices and data. This means nay changes made to a copy of the object will not be reflected tot he original series object. A deep copy of the series can be created by writing the following syntax:

ser.copy(deep=True)

The default value of the deep parameter of the copy() function is set to True.

**Q7. What is a DataFrame in Pandas?**

Ans: A DataFrame in Panda is a data structure used to store the data in tabular form, that is in the form of rows and columns. It is two-dimensional, size-mutable, and heterogeneous in nature. The main components of a dataframe are data, rows, and columns. A dataframe can be created by loading the dataset from existing storage, such as SQL database, CSV file, Excel file, etc. The syntax for creating a dataframe is as follows:

import pandas as pd  
dataframe = pd.DataFrame(data)

**Q8. What are the Different ways to Create a DataFrame in Pandas?**

Ans: In Pandas, a dataframe can be created in many ways. They are as follows:

**Creating an Empty DataFrame**

An empty dataframe can be created by just calling the **pandas.DataFrame()** constructor.

*# import pandas as pd*

**import** **pandas** **as** **pd**

*# Calling DataFrame constructor*

print(pd.DataFrame())

**Output:**

Empty DataFrame  
Columns: []  
Index: []

**Creating a DataFrame using a List**

In order to create a DataFrame from a Python list, just pass the list to the DataFrame() constructor.

*# import pandas as pd*

**import** **pandas** **as** **pd**

*# list of strings*

lst = ['Geeks', 'For', 'Geeks', 'is',

'portal', 'for', 'Geeks']

*# Calling DataFrame constructor on list*

print(pd.DataFrame(lst))

**Output:**

0  
0 Geeks  
1 For  
2 Geeks  
3 is  
4 portal  
5 for  
6 Geeks

**Creating a DataFrame using a List of Lists**

A DataFrame can be created from a Python list of lists and passed the main list to the DataFrame() constructor along with the column names.

*# import pandas as pd*

**import** **pandas** **as** **pd**

*# list of strings*

lst = [[1, 'Geeks'], [2, 'For'], [3, 'Geeks']]

*# Calling DataFrame constructor*

*# on list with column names*

print(pd.DataFrame(lst, columns=['Id', 'Data']))

**Output:**

Id Data  
0 1 Geeks  
1 2 For  
2 3 Geeks

**Creating a DataFrame using a Dictionary**

A DataFrame can be created from a Python dictionary and passed to the DataFrame() constructor. The Keys of the dictionary will be the column names and the values of the dictionary are the data of the DataFrame.

**import** **pandas** **as** **pd**

*# initialise data of lists.*

data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}

*# Print the dataframe created*

print(pd.DataFrame(data))

**Output:**

Name Age  
0 Tom 20  
1 nick 21  
2 krish 19  
3 jack 18

**Creating a DataFrame using a List of Dictionaries**

Another way to create a DataFrame is by using Python list of dictionaries. The list is passed to the DataFrame() constructor. The Keys of each dictionary element will be the column names.

*# import pandas as pd*

**import** **pandas** **as** **pd**

*# list of strings*

lst = [{1: 'Geeks', 2: 'For', 3: 'Geeks'},

{1: 'Portal', 2: 'for', 3: 'Geeks'}]

*# Calling DataFrame constructor on list*

print(pd.DataFrame(lst))

**Output:**

1 2 3  
0 Geeks For Geeks  
1 Portal for Geeks

**Creating a DataFrame from Pandas Series**

A DataFrame in Pandas can also be created by using the Pandas series.

*# import pandas as pd*

**import** **pandas** **as** **pd**

*# list of strings*

lst = pd.Series(['Geeks', 'For', 'Geeks'])

*# Calling DataFrame constructor on list*

print(pd.DataFrame(lst))

**Output:**

0  
0 Geeks  
1 For  
2 Geeks

**Q9. How to Read Data into a DataFrame from a CSV file?**

Ans: We can create a data frame from a CSV file - "Comma Separated Values". This can be done by using the **[read\_csv()](https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv)** method which takes the csv file as the parameter.

pandas.read\_csv(file\_name)

Another way to do this is by using the **[read\_table()](https://www.geeksforgeeks.org/pandas-read_table-function)** method which takes the CSV file and a delimiter value as the parameter.

pandas.read\_table(file\_name, deliniter)

**Q10. How to access the first few rows of a dataframe?**

Ans: The first few records of a dataframe can be accessed by using the pandas **head()** method. It takes one optional argument **n**, which is the number of rows. By default, it returns the first 5 rows of the dataframe. The head() method has the following syntax:

df.head(n)

Another way to do it is by using **iloc()** method. It is similar to the Python list-slicing technique. It has the following syntax:

df.iloc[:n]

**Q11. What is Reindexing in Pandas?**

Ans: Reindexing in Pandas as the name suggests means changing the index of the rows and columns of a dataframe. It can be done by using the Pandas **reindex()** method. In case of missing values or new values that are not present in the dataframe, the reindex() method assigns it as NaN.

df.reindex(new\_index)

**Q12. How to Select a Single Column of a DataFrame?**

Ans: There are many ways to [Select a single column](https://www.geeksforgeeks.org/how-to-select-single-column-of-a-pandas-dataframe) of a dataframe. They are as follows:

By using the **Dot operator**, we can access any column of a dataframe.

Dataframe.column\_name

Another way to select a column is by using the **square brackets** [].

DataFrame[column\_name]

**Q13. How to Rename a Column in a DataFrame?**

Ans: A column of the dataframe can be renamed by using the [**rename()**](https://www.geeksforgeeks.org/python-pandas-dataframe-rename) function. We can rename a single as well as multiple columns at the same time using this method.

DataFrame.rename(columns={'column1': 'COLUMN\_1', 'column2':'COLUMN\_2'}, inplace=True)

Another way is by using the **[set\_axis()](https://www.geeksforgeeks.org/python-pandas-series-set_axis)** function which takes the new column name and axis to be replaced with the new name.

DataFrame.set\_axis(labels=['COLUMN\_1','COLUMN\_2'], axis=1, inplace=True)

In case we want to add a prefix or suffix to the column names, we can use the **[add\_prefix()](https://www.geeksforgeeks.org/python-pandas-dataframe-add_prefix)** or **[add\_suffix()](https://www.geeksforgeeks.org/python-pandas-dataframe-add_suffix)** methods.

DataFrame.add\_prefix(prefix='PREFIX\_')  
DataFrame.add\_suffix(suffix='\_suffix')

**Q14. How to add an Index, Row, or Column to an Existing Dataframe?**

**Ans: Adding Index**

We can add an index to an existing dataframe by using the Pandas **set\_index()** method which is used to set a list, series, or dataframe as the index of a dataframe. The set\_index() method has the following syntax:

df.set\_index(keys, drop=True, append=False, inplace=False, verify\_integrity=False)

**Adding Rows**

The **df.loc[]**is used to access a group of rows or columns and can be used to add a row to a dataframe.

DataFrame.loc[Row\_Index]=new\_row

We can also add multiple rows in a dataframe by using **pandas.concat()** function which takes a list of dataframes to be added together.

pandas.concat([Dataframe1,Dataframe2])

**Adding Columns**

We can add a column to an existing dataframe by just declaring the column name and the list or dictionary of values.

DataFrame[data] = list\_of\_values

Another way to add a column is by using **[df.insert()](https://www.geeksforgeeks.org/python-pandas-dataframe-insert)** method which take a value where the column should be added, column name and the value of the column as parameters.

DataFrameName.insert(col\_index, col\_name, value)

We can also add a column to a dataframe by using **[df.assign()](https://www.geeksforgeeks.org/python-pandas-dataframe-assign)** function

DataFrame.assign(\*\*kwargs)

**Q15. How to Delete an Index, Row, or Column from an Existing DataFrame?**

Ans: We can delete a row or a column from a dataframe by using  df.drop()method. and provide the row or column name as the parameter.

**To delete a column**

DataFrame.drop(['Column\_Name'], axis=1)

**To delete a row**

DataFrame.drop([Row\_Index\_Number], axis=0)

**Q16. How to set the Index in a Panda dataFrame?**

Ans: We can set the index to a Pandas dataframe by using the set.indexmethod, which is used to set a list, series, or dataframe as the index of a dataframe.

DataFrame.set\_index('Column\_Name')

**Q17. How to Reset the Index of a DataFrame?**

Ans: The index of Pandas dataframes can be reset by using the  reset index method. It can be used to simply reset the index to the default integer index beginning at 0.

DataFrame.reset\_index(inplace = True)

**Q18. How to Find the Correlation Using Pandas?**

Ans: Pandas  dataframe()method is used to find the correlation of all the columns of a dataframe. It automatically ignores any missing or non-numerical values.

DataFrame.corr()

**Q19. How to Iterate over Dataframe in Pandas?**

Ans: There are various ways to  iterate the rows and columns of a dataframe.

**Iteration over Rows**

In order to iterate over rows, we apply a **iterrows()** function this function returns each index value along with a series containing the data in each row. Another way to iterate over rows is by using **iteritems()** method, which iterates over each column as key-value pairs. We can also use **itertuples()** function which returns a tuple for each row in the DataFrame.The first element of the tuple will be the row’s corresponding index value, while the remaining values are the row values.

**Iteration over Columns**

To iterate columns of a dataframe, we just need to create a list of dataframe columns by using the list constructor and passing the dataframe to it.

**Q20. What are the Important Conditions to keep in mind before Iterating?**

Ans: Iterating is not the best option when it comes to Pandas Dataframe. Pandas provides a lot of functions using which we can perform certain operations instead of iterating through the dataframe. While iterating a dataframe, we need to keep in mind the following things:

* While printing the data frame, instead of iterating, we can use DataFrame.to\_string() methods which will display the data in tabular form.
* If we are concerned about time performance, iteration is not a good option. Instead, we should choose vectorization as pandas have a number of highly optimized and efficient built-in methods.
* We should use the apply() method instead of iteration when there is an operation to be applied to a few rows and not the whole database.

**Pandas Interview Questions for Experienced**

**Q21. What is Categorical Data and How it is represented in Pandas?**

Ans: Categorical data is a set of predefined data values under some categories. It usually has a limited and fixed range of possible values and can be either numerical or textual in nature. A few examples of categorical data are gender, educational qualifications, blood type, country affiliation, observation time, etc. In Pandas categorical data is often represented by Object datatype.

**Q22. How can a DataFrame be Converted to an Excel File?**

Ans: A Pandas dataframe can be converted to an Excel file by using the [to\_excel()](https://www.geeksforgeeks.org/dataframe-to_excel-method-in-pandas) function which takes the file name as the parameter. We can also specify the sheet name in this function.

DataFrame.to\_excel(file\_name)

**Q23. What is Multi-Indexing in Pandas?**

Ans: Multi-indexing refers to selecting two or more rows or columns in the index. It is a multi-level or hierarchical object for pandas object and deals with data analysis and works with higher dimensional data. Multi-indexing in Pandas can be achieved by using a number of functions, such as **MultiIndex.from\_arrays,** **MultiIndex.from\_tuples,** **MultiIndex.from\_product,** **MultiIndex.from\_frame**,etc which helps us to create multiple indexes from arrays, tuples, dataframes, etc.

**Q24. How to select Specific Data-types to Include or Exclude in the DataFrame?**

Ans: The Pandas [select\_dtypes()](https://www.geeksforgeeks.org/select-columns-with-specific-data-types-in-pandas-dataframe) method is used to include or exclude a specific type of data in the dataframe. The datatypes to include or exclude are specified to it as a list or parameters to the function. It has the following syntax:

DataFrame.select\_dtypes(include=['object','float'], exclude =['int'])

**Q25. How to Convert a DataFrame into a Numpy Array?**

Ans: Pandas Numpy is an inbuilt Python package that is used to perform large numerical computations. It is used for processing multidimensional array elements to perform complicated mathematical operations.

The pandas dataframe can be converted to a NumPy array by using the [to\_numpy()](https://www.geeksforgeeks.org/pandas-dataframe-to_numpy-convert-dataframe-to-numpy-array) method. We can also provide the datatype as an optional argument.

Dataframe.to\_numpy()

We can also use .values to convert dataframe values to NumPy array

df.values

**Q26. How to Split a DataFrame according to a Boolean Criterion?**

Ans: Boolean masking is a technique that can be used in Pandas to split a DataFrame depending on a boolean criterion. You may divide different regions of the DataFrame and filter rows depending on a certain criterion using boolean masking.

# Define the condition  
condition = DataFrame['col\_name'] < VALUE   
# DataFrame with rows where the condition is True  
DataFrame1 = DataFrame[condition]   
# DataFrame with rows where the condition is False  
DataFrame1 = DataFrame[~condition]

**Q27. What is Time Series in Pandas?**

Ans: Time series is a collection of data points with timestamps. It depicts the evolution of quantity over time. Pandas provide various functions to handle time series data efficiently. It is used to work with data timestamps, resampling time series for different time periods, working with missing data, slicing the data using timestamps, etc.

| **Pandas Built-in Function** | **Operation** |
| --- | --- |
| pandas.to\_datetime(DataFrame['Date']) | Convert 'Date' column of DataFrame to datetime dtype |
| DataFrame.set\_index('Date', inplace=True) | Set 'Date' as the index |
| DataFrame.resample('H').sum() | Resample time series to a different frequency (e.g., Hourly, daily, weekly, monthly etc) |
| DataFrame.interpolate() | Fill missing values using linear interpolation |
| DataFrame.loc[start\_date:end\_date] | Slice the data based on timestamps |

**Q28. What is Time Delta in Pandas?**

Ans: The time delta is the difference in dates and time. Similar to the [timedelta()](https://www.geeksforgeeks.org/python-datetime-timedelta-function) object in the datetime module, a Timedelta in Pandas indicates the duration or difference in time. For addressing time durations or time variations in a DataFrame or Series, Pandas has a dedicated data type.

The time delta object can be created by using the **timedelta**() method and providing the number of weeks, days, seconds, milliseconds, etc as the parameter.

Duration = pandas.Timedelta(days=7, hours=4, minutes=30, seconds=23)

With the help of the Timedelta data type, you can easily perform arithmetic operations, comparisons, and other time-related manipulations. In terms of different units, such as days, hours, minutes, seconds, milliseconds, and microseconds, it can give durations.

Duration + pandas.Timedelta('2 days 6 hours')

**Q29. What is Data Aggregation in Pandas?**

Ans: In Pandas, data aggregation refers to the act of summarizing or decreasing data in order to produce a consolidated view or summary statistics of one or more columns in a dataset. In order to calculate statistical measures like sum, mean, minimum, maximum, count, etc., aggregation functions must be applied to groups or subsets of data.

The **agg()** function in Pandas is frequently used to aggregate data. Applying one or more aggregation functions to one or more columns in a DataFrame or Series is possible using this approach. Pandas' built-in functions or specially created user-defined functions can be used as aggregation functions.

DataFrame.agg({'Col\_name1': ['sum', 'min', 'max'], 'Col\_name2': 'count'})

**Q30. Difference between merge() and concat()**

Ans: The following table shows the difference between merge() and concat():

| **.merge()** | **concat()** |
| --- | --- |
| It is used to join exactly 2 dataframes based on a common column or index | It is used to join 2 or more dataframes along a particular axis i.e rows or columns |
| Perform different types of joins such as inner join, outer join, left join, and right join. | Performs concatenation by appending the dataframes one below the other (along the rows) or side by side (along the columns). |
| Join types and column names have to be specified. | By default, performs row-wise concatenation (i.e. axis=0). To perform column-wise concatenation (i.e. axis=1) |
| Multiple columns can be merged if needed | Does not perform any sort of matching or joining based on column values |
| Used when we want to combine data based on a shared column or index. | Commonly used when you want to combine dataframes vertically or horizontally without any matching criteria. |

**Q31. Difference between map(), applymap(), and apply()**

Ans: The map(), applymap(), and apply() methods are used in pandas for applying functions or transformations to elements in a DataFrame or Series. The following table shows the difference between map(), applymap() and apply():

| **map()** | **applymap()** | **apply()** |
| --- | --- | --- |
| Defined only in Series | Defined only in Dataframe | Defined in both Series and DataFrame |
| Used to apply a function or a dictionary to each element of the Series. | Used to apply a function to each element of the DataFrame. | Used to apply a function along a specific axis of the DataFrame or Series. |
| Series.map() works element-wise and can be used to perform element-wise transformations or mappings. | DataFrame.applymap() works element-wise, applying the provided function to each element in the DataFrame. | DataFrame.apply() works on either entire rows or columns element-wise of a Dataframe or Series |
| Used when we want to apply a simple transformation or mapping operation to each element of a series | Used when we want to apply a function to each individual element of a Dataframe | Used when we want to apply a function that aggregates or transforms data across rows or columns. |

**Q32. Difference between pivot\_table() and groupby()**

Ans: Both pivot\_table() and groupby() are powerful methods in pandas used for aggregating and summarizing data. The following table shows the difference between pivot\_table() and groupby():

| **pivot\_table()** | **groupby()** |
| --- | --- |
| It summarizes and aggregates data in a tabular format | It performs aggregation on grouped data of one or more columns |
| Used to transform data by reshaping it based on column values. | Used to group data based on categorical variables then we can apply various aggregation functions to the grouped data. |
| It can handle multiple levels of grouping and aggregation, providing flexibility in summarizing data. | It performs grouping based on column values and creates a GroupBy object then aggregation functions, such as sum, mean, count, etc., can be applied to the grouped data. |
| It is used when we want to compare the data across multiple dimensions | It is used to summarize data within groups |

**Q33. How can we use Pivot and Melt Data in Pandas?**

Ans: We can pivot the dataframe in Pandas by using th**e** [pivot\_table()](https://www.geeksforgeeks.org/python-pandas-pivot_table) method. To unpivot the dataframe to its original form we can melt the dataframe by using the [melt()](https://www.geeksforgeeks.org/python-pandas-melt) method.

**Q34. How to convert a String to Datetime in Pandas?**

Ans: A Python string can be converted to a DateTime object by using the [to\_datetime()](https://www.geeksforgeeks.org/python-pandas-to_datetime) function or **[strptime()](https://www.geeksforgeeks.org/python-datetime-strptime-function)** method of datetime. It returns a DateTime object corresponding to date\_string, parsed according to the format string given by the user.

Using Pandas.to\_datetime()

**import** **pandas** **as** **pd**

*# Convert a string to a datetime object*

date\_string = '2023-07-17'

dateTime = pd.to\_datetime(date\_string)

print(dateTime)

**Output**:

2023-07-17 00:00:00

Using datetime.strptime

**from** **datetime** **import** datetime

*# Convert a string to a datetime object*

date\_string = '2023-07-17'

dateTime = datetime.strptime(date\_string, '%Y-%m-**%d**')

print(dateTime)

**Output**:

2023-07-17 00:00:00

**Q35. What is the Significance of Pandas Described Command?**

Ans: Pandas [describe()](https://www.geeksforgeeks.org/python-pandas-dataframe-describe-method) is used to view some basic statistical details of a data frame or a series of numeric values. It can give a different output when it is applied to a series of strings. It can get details like percentile, mean, standard deviation, etc.

DataFrame.describe()

**Q36. How to Compute Mean, Median, Mode, Variance, Standard Deviation, and Various Quantile Ranges in Pandas?**

Ans: The mean, median, mode, Variance, Standard Deviation, and Quantile range can be computed using the following commands in Python.

* DataFrame.mean(): To calculate the mean
* DataFrame.median(): To calculate median
* DataFrame.mode(): To calculate the mode
* DataFrame.var(): To calculate variance
* DataFrame.std(): To calculate the standard deviation
* DataFrame.quantile(): To calculate quantile range, with range value as a parameter

**Q37. How to make Label Encoding using Pandas?**

Ans: Label encoding is used to convert categorical data into numerical data so that a machine-learning model can fit it. To apply label encoding using pandas we can use the  pandas .Categorical().codes or  pandas.factorize() method to replace the categorical values with numerical values.

**Q38. How to make Onehot Encoding using Pandas?**

Ans: One-hot encoding is a technique for representing categorical data as numerical values in a machine-learning model. It works by creating a separate binary variable for each category in the data. The value of the binary variable is 1 if the observation belongs to that category and 0 otherwise. It can improve the performance of the model. To apply one hot encoding, we greater a dummy column for our dataframe by using [get\_dummies()](https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python) method.

**Q39. How to make a Boxplot using Pandas?**

Ans: A Boxplot is a visual representation of grouped data. It is used for detecting outliers in the data set. We can create a boxplot using the Pandas dataframe by using the [boxplot()](https://www.geeksforgeeks.org/box-plot-visualization-with-pandas-and-seaborn) method and providing the parameter based on which we want the boxplot to be created.

DataFrame.boxplot(column='Col\_Name', grid=False)

**Q40. How to make a Distribution Plot using Pandas?**

Ans: A distribution plot is a graphical representation of the distribution of data. It is a type of histogram that shows the frequency of each value in a dataset. To create a distribution plot using Pandas, you can use the plot.hist() method. This method takes a DataFrame as input and creates a histogram for each column in the DataFrame.

DataFrame['Numerical\_Col\_Name'].plot.hist()

**Pandas Interview Questions for Data Scientists**

**Q41. How to Sort a Dataframe?**

Ans: A dataframe in pandas can be sorted in ascending or descending order according to a particular column. We can do so by using the **[sort\_values()](https://www.geeksforgeeks.org/python-pandas-dataframe-sort_values-set-1)** method. and providing the column name according to which we want to sort the dataframe. we can also sort it by multiple columns. To sort it in descending order, we pass an additional parameter 'ascending' and set it to False.

DataFrame.sort\_values(by='Age',ascending=True)

**Q42. How to Check and Remove Duplicate Values in Pandas.**

Ans: In pandas, duplicate values can be checked by using the [duplicated()](https://www.geeksforgeeks.org/python-pandas-dataframe-duplicated) method.

DataFrame.duplicated()

To remove the duplicated values we can use the [drop\_duplicates()](https://www.geeksforgeeks.org/python-pandas-dataframe-drop_duplicates) method.

DataFrame.drop\_duplicates()

**Q43. How to Create a New Column Based on Existing Columns?**

Ans: We can create a column from an existing column in a DataFrame by using the [df.apply()](https://www.geeksforgeeks.org/python-pandas-apply) and [df.map()](https://www.geeksforgeeks.org/python-pandas-map) functions

**Q44. How to Handle Missing Data in Pandas?**

Ans: Generally dataset has some missing values, and it can happen for a variety of reasons, such as data collection issues, data entry errors, or data not being available for certain observations. This can cause a big problem. To handle these missing values Pandas provides various functions. These functions are used for detecting, removing, and replacing null values in Pandas DataFrame:

* isnull()**:** It returns True for NaN values or null values and False for present values
* notnull(): It returns False for NaN values and True for present values
* dropna()**:** It analyzes and drops Rows/Columns with Null values
* fillna()**:** It let the user replace NaN values with some value of their own
* replacena()**:** It is used to replace a string, regex, list, dictionary, series, number, etc.
* interpolate: It fills NA values in the dataframe or series.

**Q45. What is groupby() Function in Pandas?**

Ans: The [groupby()](https://www.geeksforgeeks.org/python-pandas-dataframe-groupby) function is used to group or aggregate the data according to a category. It makes the task of splitting the Dataframe over some criteria really easy and efficient. It has the following syntax:

DataFrame.groupby(by=['Col\_name'])

**Q46. What are loc and iloc methods in Pandas?**

Ans: Pandas Subset Selection is also known as Pandas Indexing. It means selecting a particular row or column from a dataframe. We can also select a number of rows or columns as well. Pandas support the following types of indexing:

* **Dataframe.[ ]:** This function is also known as the indexing operator
* **Dataframe.loc[ ]:** This function is used for label-based indexing.
* **Dataframe.iloc[ ]:** This function is used for positions or integer-based indexing.

**Q47. How to Merge Two DataFrames?**

Ans: In pandas, we can combine two dataframes using the [pandas.merge()](https://www.geeksforgeeks.org/how-to-join-pandas-dataframes-using-merge) method which takes 2 dataframes as the parameters.

**import** **pandas** **as** **pd**

*# Create two DataFrames*

df1 = pd.DataFrame({'A': [1, 2, 3],

'B': [4, 5, 6]},

index=[10, 20, 30])

df2 = pd.DataFrame({'C': [7, 8, 9],

'D': [10, 11, 12]},

index=[20, 30, 40])

*# Merge both dataframe*

result = pd.merge(df1, df2, left\_index=**True**, right\_index=**True**)

print(result)

**Output**:

A B C D  
20 2 5 7 10  
30 3 6 8 11

**Q48. Difference between iloc() and loc()**

Ans: The iloc() and loc() functions of pandas are used for accessing data from a DataFrame.The following table shows the difference between iloc() and loc():

| **iloc()** | **loc()** |
| --- | --- |
| It is an indexed-based selection method | It is labelled based selection method |
| It allows you to access rows and columns of a DataFrame by their integer positions | It allows you to access rows and columns of a DataFrame using their labels or names. |
| The indexing starts from 0 for both rows and columns. | The indexing can be based on row labels, column labels, or a combination of both. |
| Used for integer-based slicing, which can be single integers, lists or arrays of integers for specific rows or columns. | Used for label-based slicing, the labels can be single labels, lists or arrays of labels for specific rows or columns |
| **Syntax:**  DataFrame.iloc[row\_index, column\_index] | **Syntax**:  DataFrame.loc[row\_label, column\_label] |

**Q49. Difference between join() and merge()**

Ans: Both join() and merge() functions in pandas are used to combine data from multiple DataFrames. The following table shows the difference between join and merge():

| **join()** | **merge()** |
| --- | --- |
| Combines dataframes on their indexes | Combines dataframes by specifying the columns as a merge key |
| Joining is performed on the DataFrame's index and not on any specified columns. | Joining is performed based on the values in the specified columns or indexes. |
| Does not support merging based on column values or multiple columns. | Supports merging based on one or more columns or indexes, allowing for more flexibility in combining DataFrames. |

**Q50. Difference between the interpolate() and fillna()**

Ans: The interpolate() and fillna() methods in pandas are used to handle missing or NaN (Not a Number) values in a DataFrame or Series. The following table shows the difference between interpolate() and fillna():

| **interpolate()** | **fillna()** |
| --- | --- |
| Fill in the missing values based on the interpolation or estimate values based on the existing data. | Fill missing values with specified values that can be based on some strategies. |
| Performs interpolation based on various methods such as linear interpolation, polynomial interpolation, and time-based interpolation. | Replaces NaN values with a constant like zero, mean, median, mode, or any other custom value computed from the existing data. |
| Applied to both numerical and DateTime data when dealing with time series data or when there is a logical relationship between the missing values and the existing data. | Can be applied to both numerical and categorical data. |

**Conclusion**

In conclusion, our *Pandas Interview Questions and answers* article serves as a comprehensive guide for anyone aspiring to make a mark in the Data Science and ML profession. With a wide range of questions from basic to advanced, including practical coding questions, we've covered all the bases to ensure you're well-prepared for your interviews.

Remember, the key to acing an interview is not just knowing the answers, but understanding the concepts behind them. We hope this article has been helpful in your preparation and wish you all the best in your journey.

**Pandas Interview Questions - FAQs**

**1. Which three 3 main objects does pandas have?**

*The Three fundamental objects around which the whole pandas function revolves around are Series, DataFrame , and Index.*

**2. Why does everyone use pandas?**

*Pandas allow wide range of data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features. Apart from that, Pandas is very compatible with file-handling operation such as importing data from various file formats such as comma-separated values, JSON, Parquet, SQL database tables or queries, and Microsoft Excel.*

**3. What is all () in pandas?**

*DataFrame.all() method checks whether all elements are True, potentially over an axis. It returns True if all elements within a series or along a Dataframe axis are non-zero, not-empty or not-False.*

**Python for machine learning - pandas intrview tips and triks**

To prepare for a Python machine learning interview focusing on pandas, it's essential to understand the types of questions you may encounter and the key concepts related to data manipulation and analysis. Here are some tips and tricks to help you excel in your interview:

Key Topics to Focus On

1. **Pandas Basics**:
   * Understand the fundamental data structures: **Series** and **DataFrame**.
   * Be able to explain how to create these structures and their differences.
2. **Data Manipulation Techniques**:
   * Familiarize yourself with common operations such as filtering, grouping, merging, and aggregating data.
   * Practice using methods like groupby(), merge(), and concat() to manipulate datasets effectively.
3. **Handling Missing Data**:
   * Know how to identify and handle missing values using fillna(), dropna(), and interpolate() methods.
   * Be prepared to discuss strategies like using the Interquartile Range (IQR) for outlier detection and removal[2](https://www.projectpro.io/article/pandas-interview-questions-and-answers/985)[3](https://www.interviewbit.com/pandas-interview-questions/).
4. **Data Preprocessing**:
   * Understand normalization and standardization techniques, particularly using MinMaxScaler and StandardScaler from sklearn.preprocessing[2](https://www.projectpro.io/article/pandas-interview-questions-and-answers/985).
   * Be ready to explain how to clean data, such as removing duplicates or irrelevant features.
5. **Advanced Pandas Features**:
   * Learn about multi-indexing for hierarchical data representation.
   * Familiarize yourself with time series operations, such as resampling and time-based indexing[3](https://www.interviewbit.com/pandas-interview-questions/)[4](https://www.upgrad.com/blog/pandas-interview-questions-answers-for-freshers-experienced/).

Common Interview Questions

* **What is Pandas?**
  + Explain that pandas is a powerful library for data manipulation and analysis in Python, widely used in data science and machine learning projects.
* **How do you filter rows in a DataFrame based on a condition?**
  + You can use boolean indexing, e.g., df[df['column\_name'] > value][2](https://www.projectpro.io/article/pandas-interview-questions-and-answers/985).
* **What are the differences between interpolate() and fillna()?**
  + interpolate() estimates missing values based on surrounding data, while fillna() replaces missing values with a specified constant or method[2](https://www.projectpro.io/article/pandas-interview-questions-and-answers/985).
* **How would you handle outliers in a dataset?**
  + Discuss methods like IQR or z-score for identifying outliers and strategies for either removing or adjusting them[2](https://www.projectpro.io/article/pandas-interview-questions-and-answers/985).

Coding Practice

Be prepared to write code during your interview. Here are some example tasks you might be asked to perform:

* **Filling Missing Values**:

python

df['column\_name'].fillna(value=0, inplace=True)

* **Calculating Median**:

python

median\_value = df['column\_name'].median()

* **Iterating Over Rows**:

python

**for** index, row **in** df.iterrows():

**print**(row['column\_name'])

Additional Tips

* **Practice Coding**: Use platforms like LeetCode or HackerRank to practice coding challenges specifically focused on pandas.
* **Real-World Applications**: Be ready to discuss how you've used pandas in past projects or how it integrates with machine learning workflows.
* **Stay Updated**: Keep abreast of the latest features in pandas by reviewing the official documentation regularly.

**Machine Learning With Python Questions**

This set of Machine Learning interview questions deal with Python-related Machine Learning questions.

**Q1. Name a few libraries in Python used for Data Analysis and Scientific Computations.**

Here is a list of Python libraries mainly used for Data Analysis:

* NumPy
* SciPy
* Pandas
* SciKit
* Matplotlib
* Seaborn
* Bokeh

**Q2. Which library would you prefer for plotting in Python language: Seaborn or Matplotlib or Bokeh?**

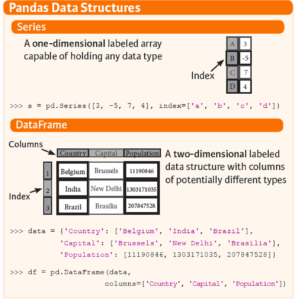
It depends on the visualization you’re trying to achieve. Each of these libraries is used for a specific purpose:

* **Matplotlib:** Used for basic plotting like bars, pies, lines, scatter plots, etc
* **Seaborn:** Is built on top of Matplotlib and Pandas to ease data plotting. It is used for statistical visualizations like creating heatmaps or showing the distribution of your data
* **Bokeh:** Used for interactive visualization. In case your data is too complex and you haven’t found any “message” in the data, then use Bokeh to create interactive visualizations that will allow your viewers to explore the data themselves

**Q3. How are NumPy and SciPy related?**

* NumPy is part of SciPy.
* NumPy defines arrays along with some basic numerical functions like indexing, sorting, reshaping, etc.
* SciPy implements computations such as numerical integration, optimization and machine learning using NumPy’s functionality.

**Q4. What is the main difference between a Pandas series and a single-column DataFrame in Python?**



**Q5. How can you handle duplicate values in a dataset for a variable in Python?**

Consider the following Python code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | bill\_data=pd.read\_csv("datasetsTelecom Data AnalysisBill.csv")  bill\_data.shape  #Identify duplicates records in the data  Dupes = bill\_data.duplicated()  sum(dupes)  #Removing Duplicates  bill\_data\_uniq = bill\_data.drop\_duplicates() |

**Q6. Write a basic Machine Learning program to check the accuracy of a model,  by importing any dataset using any****classifier?**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #importing dataset  import sklearn  from sklearn import datasets  iris = datasets.load\_iris()  X = iris.data  Y = iris.target    #splitting the dataset  from sklearn.cross\_validation import train\_test\_split  X\_train, Y\_train, X\_test, Y\_test = train\_test\_split(X,Y, test\_size = 0.5)    #Selecting Classifier  my\_classifier = tree.DecisionTreeClassifier()  My\_classifier.fit(X\_train, Y\_train)  predictions = my\_classifier(X\_test)  #check accuracy  From sklear.metrics import accuracy\_score  print accuracy\_score(y\_test, predictions) |

**Pandas interview questions** that are commonly asked in **data science and machine learning** interviews. Below is a structured summary of the **key topics and solutions** discussed.

**1. Creating and Manipulating DataFrames in Pandas**

**Question:** How do you create a Pandas DataFrame?

* Use pd.DataFrame() with lists or dictionaries.

python

CopyEdit

import pandas as pd

df = pd.DataFrame({'Platform': ['Windows', 'Mac', 'Linux'], 'Users': [1000, 500, 200]})

**Question:** How do you create an empty DataFrame?

* df = pd.DataFrame() (Creates a memory reference with no records).
* df = pd.DataFrame(columns=['Column1', 'Column2']) (Creates an empty DataFrame with specific columns).

**2. Data Type Conversion in Pandas**

**Question:** How do you change the data type of a column?

* Use .astype() to convert column types.

python

CopyEdit

df['Users'] = df['Users'].astype(float)

💡 **Tip:** Use copy=False inside .astype() to avoid returning a new object.

**3. Handling Missing Values**

**Question:** How do you handle missing values in Pandas?

* **Drop missing values**:

python

CopyEdit

df.dropna(inplace=True)

* **Fill missing values with a default value**:

python

CopyEdit

df.fillna(0, inplace=True)

💡 **Tip:** Be prepared to **justify why** you choose dropna() vs. fillna(), especially in real-world data analysis.

**4. Extracting Date and Finding Leap Year**

**Question:** How do you extract the current date and find the next leap year using Pandas?

* Use pd.Timestamp.now() and loop to find the next leap year.

python

CopyEdit

import pandas as pd

def next\_leap\_year():

today = pd.Timestamp.now()

year = today.year

while not pd.Timestamp(year, 2, 29, 0).is\_leap\_year:

year += 1

return today.date(), year

print(next\_leap\_year())

💡 **Tip:** Pandas internally uses Python’s datetime module for date manipulations.

**5. Grouping and Sorting Data in Pandas**

**Question:** How do you find the **top 3 rows** based on a column after groupby()?

* Use .groupby() followed by .nlargest().

python

CopyEdit

df.groupby('4G')['int\_memory'].nlargest(3)

💡 **Tip:** This method is commonly used in **telecom and e-commerce analytics**.

**Question:** How do you check if **all 4G phones have 3G and Wi-Fi enabled**?

* Use groupby() and check distributions.

python

CopyEdit

df.groupby(['4G', '3G'])['WiFi'].count()

* If 1,0 (4G phone without 3G) **doesn’t exist**, then all 4G phones have 3G.

**6. Finding the Row with Maximum RAM for Each Price Range**

**Question:** How do you find the row containing the **highest RAM** for each price range?

* Use .groupby() with .idxmax().

python

CopyEdit

df.loc[df.groupby('price\_range')['ram'].idxmax()]

💡 **Tip:** idxmax() returns the index of the highest value, which can be used to retrieve full row data.

**7. Working with Multi-Index DataFrames**

**Question:** How do you sum values at level 0 in a Multi-Index DataFrame?

* Use .xs() and .sum().

python

CopyEdit

df.xs(key=0, level=0).sum()

💡 **Tip:** Multi-indexing is useful in **financial and stock market analysis**.

**8. Exploding and Splitting List Columns in DataFrames**

**Question:** How do you **convert a list column into multiple rows**?

* Use .explode().

python

CopyEdit

df.explode('subjects')

**Question:** How do you **convert a list column into multiple columns**?

* Use pd.DataFrame().

python

CopyEdit

pd.DataFrame(df['subjects'].tolist())

💡 **Tip:** **Exploding vs. Splitting:** Exploding keeps values in rows, while splitting distributes them into columns.

**9. Difference Between set\_index() and reindex()**

**Question:** What is the difference between set\_index() and reindex()?

* **set\_index()**: Converts a column into an index.
* **reindex()**: Matches new indices with existing ones, filling missing values with NaN.

python

CopyEdit

df.set\_index('name', inplace=True)

df.reindex(['Rahul', 'Sita', 'John'])

💡 **Tip:** set\_index() requires **unique** values, while reindex() can introduce NaN.

**10. Renaming Columns and Indices in Pandas**

**Question:** How do you rename columns in Pandas?

* Use .rename().

python

CopyEdit

df.rename(columns={'old\_column': 'new\_column'}, inplace=True)

**Question:** How do you rename index labels?

* Use .index.names.

python

CopyEdit

df.index.names = ['New\_Index']

💡 **Tip:** This is useful for **restructuring datasets** for reporting.

**11. Sorting Data in Pandas**

**Question:** How do you sort a DataFrame by **ascending age** and **descending subject**?

* Use .sort\_values().

python

CopyEdit

df.sort\_values(by=['age', 'subjects'], ascending=[True, False])

💡 **Tip:** Sorting is **essential for ranking and leaderboard data analysis**.

**12. Finding Rank of a Value in Pandas**

**Question:** How do you find the **rank of a person** based on a column?

* Use .rank().

python

CopyEdit

df['rank'] = df['age'].rank()

💡 **Tip:** This is commonly used in **sports, university rankings, and employee performance evaluations**.

**13. Time Series Operations in Pandas**

**Question:** How do you **increment the current date by P days**?

* Use pd.to\_timedelta().

python

CopyEdit

df['new\_date'] = df['current\_date'] + pd.to\_timedelta(5, unit='D')

💡 **Tip:** **Time series manipulations** are common in **forecasting and trend analysis**.

**Final Takeaways**

✅ **Pandas is crucial for data manipulation**—master **grouping, merging, filtering, and sorting**.  
✅ **Learn Pandas for both structured and unstructured data processing**.  
✅ **Understand when to use Pandas vs. SQL for data handling in interviews**.  
✅ **Practice live coding in Pandas to improve speed and accuracy** in technical rounds.