**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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

*# 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.

Python

**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.

Python

*# 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.

Python

*# 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()](https://www.geeksforgeeks.org/how-to-drop-one-or-multiple-columns-in-pandas-dataframe) 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\_index()](https://www.geeksforgeeks.org/python-pandas-dataframe-set_index) method, 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(](https://www.geeksforgeeks.org/python-pandas-dataframe-reset_index)**[)](https://www.geeksforgeeks.org/python-pandas-dataframe-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.corr()](https://www.geeksforgeeks.org/python-pandas-dataframe-corr) 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](https://www.geeksforgeeks.org/how-to-iterate-over-rows-in-pandas-dataframe) 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()

Python

**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

Python

**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](https://www.geeksforgeeks.org/python-pandas-categorical) or [pandas.factorize(](https://www.geeksforgeeks.org/python-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()**](https://www.geeksforgeeks.org/python-pandas-series-isnull)**:** It returns True for NaN values or null values and False for present values
* [**notnull()**](https://www.geeksforgeeks.org/python-pandas-dataframe-notnull): It returns False for NaN values and True for present values
* [**dropna()**](https://www.geeksforgeeks.org/python-pandas-dataframe-dropna)**:** It analyzes and drops Rows/Columns with Null values
* [**fillna()**](https://www.geeksforgeeks.org/python-pandas-series-fillna)**:** It let the user replace NaN values with some value of their own
* [**replace()**](https://www.geeksforgeeks.org/python-pandas-dataframe-replace)**:** It is used to replace a string, regex, list, dictionary, series, number, etc.
* [**interpolate():**](https://www.geeksforgeeks.org/python-pandas-dataframe-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.

Python

**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.

Stay tuned for more such resources and keep learning!

https://www.geeksforgeeks.org/pandas-interview-questions/