Python | Pandas Interview Questions And Answers

1. What is Pandas?

Ans: Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

---

2. What is Python pandas used for?

Ans: Pandas is a software library written for the **Python** programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. **pandas** is free software released under the three-clause BSD license.

---

3. What is a Series in Pandas?

Ans: Pandas Series is a one-dimensional labelled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index. **Pandas Series** is nothing but a column in an excel sheet.

---

4. Mention the different Types of Data structures in pandas??

Ans: There are two data structures supported by pandas library, Series and DataFrames. Both of the data structures are built on top of Numpy. **Series** is a one-dimensional data structure in pandas and **DataFrame** is the two-dimensional data structure in pandas. There is one more axis label known as **Panel** which is a three-dimensional data structure and it includes items, major_axis, and minor_axis.

---

5. Explain Reindexing in pandas?

Ans: Re-indexing means to conform DataFrame to a new index with optional filling logic, placing NA/NaN in locations having no value in the previous index. It changes the row labels and column labels of a DataFrame.

---

6. What are the key features of pandas library ?

Ans: There are various features in pandas library and some of them are mentioned below

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

---

7. What is pandas Used For ?

Ans: This library is written for the Python programming language for performing operations like data manipulation, data analysis, etc. The library provides various operations as well as data structures to manipulate time series and numerical tables.

---

8. How can we create copy of series in Pandas?

Ans: pandas.Series.copy

Series.copy(*deep=True*)

pandas.Series.copy. Make a deep **copy**, including a **copy** of the data and the indices. With deep=False neither the indices or the data are **copied**. Note that when deep=True data is **copied**, actual **python** objects will not be **copied** recursively, only the reference to the object.

---

9. What is Time Series in pandas?

Ans: A time series is an ordered sequence of data which basically represents how some quantity changes over time. pandas contains extensive capabilities and features for working with time series data for all domains.

pandas supports:

Parsing time series information from various sources and formats

Generate sequences of fixed-frequency dates and time spans

Manipulating and converting date time with timezone information

Resampling or converting a time series to a particular frequency

Performing date and time arithmetic with absolute or relative time increments

---

10. Explain Categorical Data in Pandas?

Ans: Categorical are a pandas data type corresponding to categorical variables in statistics. A categorical variable takes on a limited and usually fixed, number of possible values (categories; levels in R). Examples are gender, social class, blood type, country affiliation, observation time or rating via Likert scales. All values of categorical data are either in categories or np.nan.

The categorical data type is useful in the following cases:

A string variable consisting of only a few different values. Converting such a string variable to a categorical variable will save some memory,

The lexical order of a variable is not the same as the logical order (“one”, “two”, “three”). By converting to a categorical and specifying an order on the categories, sorting and min/max will use the logical order instead of the lexical order,

As a signal to other Python libraries that this column should be treated as a categorical variable (e.g. to use suitable statistical methods or plot types).

---

11. How will you create a series from dict in Python?

Ans: A **Series** is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). It has to be remembered that unlike Python lists, a Series will always contain data of the

same type.

Let's see how to create a Pandas Series from Dictionary.

Using Series() method without index parameter.

---

12. What are operations on Series in pandas?

Ans: Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called *index*. Pandas Series is nothing but a column in an excel sheet.

Creating a Pandas Series-

In the real world, a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas Series can be created from the lists, dictionary, and from a scalar value etc. Series can be created in different ways, here are some ways by which we create a series:

Creating a series from array: In order to create a series from array, we have to import a numpy module and have to use array() function.

```
# import pandas as pd

import pandas as pd


# import numpy as np

import numpy as np


# simple array

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


ser = pd.Series(data)

print(ser)
```

Output :

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

---

13. What is a DataFrame in pandas?

Ans: Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.

Creating a Pandas DataFrame-

n the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

Creating a dataframe using List: DataFrame can be created using a single list or a list of lists.

```
# import pandas as pd
```

```
import pandas as pd


# list of strings

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

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


# Calling DataFrame constructor on list

df = pd.DataFrame(lst)

print(df)
```

Output:

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

---

14. What are the different ways in which a DataFrame can be created in Pandas?

Ans: Pandas DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. It is generally the most commonly used pandas object.

Pandas DataFrame can be created in multiple ways. Let's discuss different ways to create a DataFrame one by one.

Creating Pandas DataFrame from lists of lists.

```
Import pandas library

import pandas as pd


# initialize list of lists

data = [['tom', 10], ['nick', 15], ['juli', 14]]


# Create the pandas DataFrame

df = pd.DataFrame(data, columns = ['Name', 'Age'])


# print dataframe.

df
```

Output:

	Name	Age
0	tom	10
1	nick	15
2	juli	14

---

15. How will you create an empty DataFrame in pandas?

Ans: To create a completely empty Pandas dataframe, we use do the following:

import pandas as pd

MyEmptydf = pd.DataFrame()

This will create an empty dataframe with no columns or rows.

To create an empty dataframe with three empty column (columns X, Y and Z), we do:

df = pd.DataFrame(columns=['X', 'Y', 'Z'])

---

16. How will you add a column to a pandas DataFrame?

Ans: Adding new column to existing DataFrame in Pandas

```
Import pandas package

import pandas as pd


# Define a dictionary containing Students data

data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],

'Height': [5.1, 6.2, 5.1, 5.2],

'Qualification': ['Msc', 'MA', 'Msc', 'Msc']}


# Convert the dictionary into DataFrame

df = pd.DataFrame(data)


# Declare a list that is to be converted into a column

address = ['Delhi', 'Bangalore', 'Chennai', 'Patna']


# Using 'Address' as the column name

# and equating it to the list

df['Address'] = address


# Observe the result

df
```

Output:

	Name	Height	Qualification	Address
0	Jai	5.1	Msc	Delhi
1	Princi	6.2	MA	Bangalore
2	Gaurav	5.1	Msc	Chennai
3	Anuj	5.2	Msc	Patna

---

17. How will you retrieve a single column from pandas DataFrame?

Ans: To start a project in Django, use the command $django-admin.py and then use the following command:

Project

init.py

manage.py

settings.py

urls.py

---

18. range () vs and xrange () functions in Python?

Ans:

In **Python 2** we have the following two functions to produce a list of numbers within a given range.

range()

xrange()

in Python 3, **xrange() is deprecated**, i.e. xrange() is removed from python 3.x.

Now In Python 3, we have only one function to produce the numbers within a given range i.e. range() function.

But, range() function of python 3 works same as xrange() of python 2 (i.e. internal implementation of range() function of python 3 is same as xrange() of Python 2).

So The **difference** between **range()** and **xrange()** functions **becomes relevant only when you are using python 2.**

range() and xrange() function values

a). range() creates a list i.e., range returns a Python list object, for example, range (1,500,1) will create a python list of 499 integers in memory. Remember, **range() generates all numbers at once**.

b).xrange() functions returns an xrange object that evaluates lazily. That means xrange only stores the range arguments and generates the numbers on demand. It doesn't generate all numbers at once like range(). Furthermore, this object only supports indexing, iteration, and the len() function.

On the other hand **xrange() generates the numbers on demand.** That means it produces number one by one as for loop moves to the next number. In every iteration of for loop, it generates the next number and assigns it to the iterator variable of for loop.

Printing return type of range() function:

range_numbers = range(2,10,2)

```
print (“The return type of range() is : “)

print (type(range_numbers ))
```

Output:

```
The return type of range() is :

<type ‘list’>
```

Printing return type of xrange() function:

```
xrange_numbers = xrange(1,10,1)

print “The return type of xrange() is : ”

print type(xrange_numbers )
```

Output:

```
The return type of xrange() is :

<type ‘xrange’>
```

---

19. What is the name of pandas library tools used to create a scatter plot matrix?

Ans: Scatter_matrix

---

20. What is pylab?

Ans: **PyLab** is a package that contains NumPy, SciPy, and Matplotlib into a single namespace.

- Machine Learning Interview Questions And Answers

Our Blogs

Cognitive RPA And The Need For The Change
Non_IT Person UiPath RPA Journey
Future of Manufacturing Using AI
The Difference Between Artificial Intelligence, Machine learning And Deep Learning
Top 10 Key Features to Grow your Career with RPA
How Robotics Industry Overcome the Awareness Gap?
RPA : Is it the fresh IT job killer?
THE HISTORY OF DATA SCIENCE
Fundamentals of Robotic Process Automation (RPA)

21. Define the different ways a DataFrame can be created in pandas?

Ans: We can create a DataFrame using following ways:

- Lists
- Dict of ndarrays

Example-1: Create a DataFrame using List:

```
1. importpandas as pd
2. # a list of strings
3. a = ['Python', 'Pandas']
4. # Calling DataFrame constructor on list
5. info = pd.DataFrame(a)
6. print(info)
```

Output:

```
0

0   Python

1   Pandas
```

Example-2: Create a DataFrame from dict of ndarrays:

```
1. importpandas as pd
2. info = {'ID':[101, 102, 103],'Department' :['B.Sc','B.Tech','M.Tech',]}
3. info = pd.DataFrame(info)
4. print (info)
```

Output:

```
ID     Department

0     101       B.Sc

1     102       B.Tech

2     103       M.Tech
```

---

22. Explain Categorical data in Pandas?

Ans: A Categorical data is defined as a Pandas data type that corresponds to a categorical variable in statistics. A categorical variable is generally used to take a limited and usually fixed number of possible values. Examples: gender, country affiliation, blood type, social class, observation time, or rating via Likert scales. All values of categorical data are either in categories or np.nan.

This data type is useful in the following cases:

- It is useful for a string variable that consists of only a few different values. If we want to save some memory, we can convert a string variable to a categorical variable.
- It is useful for the lexical order of a variable that is not the same as the logical order (?one?, ?two?, ?three?) By converting into a categorical and specify an order on the categories, sorting and min/max is responsible for using the logical order instead of the lexical order.
- It is useful as a signal to other Python libraries because this column should be treated as a categorical variable.

Parameters-

val : [list-like] The values of categorical. **categories** : [index like] Unique categorization of the cate

Code :

```
# Python code explaining
# numpy.pandas.Categorical()

# importing libraries
import numpy as np
import pandas as pd

# Categorical using dtype
c = pd.Series(["a", "b", "d", "a", "d"], dtype ="category")
print ("\nCategorical without pandas.Categorical() : \n", c)

c1 = pd.Categorical([1, 2, 3, 1, 2, 3])
print ("\n\nc1 : ", c1)

c2 = pd.Categorical(['e', 'm', 'f', 'i',
'f', 'e', 'h', 'm' ])
print ("\nc2 : ", c2)
```

OutPut:

```
Categorical without pandas.Categorical() :
 0    a
1    b
2    d
3    a
4    d
dtype: category
Categories (3, object): [a, b, d]


c1 :  [1, 2, 3, 1, 2, 3]
Categories (3, int64): [1, 2, 3]

c2 :  [e, m, f, i, f, e, h, m]
Categories (5, object): [e, f, h, i, m]
```

---

23. How will you create a series from dict in Pandas?

Ans: A Series is defined as a one-dimensional array that is capable of storing various data types.

We can create a Pandas Series from Dictionary:

Create a Series from dict:

We can also create a Series from dict. If the dictionary object is being passed as an input and the index is not specified, then the dictionary keys are taken in a sorted order to construct the index.

If index is passed, then values correspond to a particular label in the index will be extracted from the dictionary.

```
1. import pandas as pd
2. import numpy as np
3. info = {'x': 0., 'y' : 1., 'z' : 2.}
4. a = pd.Series(info)
5. print (a)
```

Output:

x 0.0

y 1.0

z 2.0

dtype: float64

---

24. How can we create a copy of the series in Pandas?

Ans: We can create the copy of series by using the following syntax:

pandas.Series.copy
Series.copy(deep=True)

The above statements make a deep copy that includes a copy of the data and the indices. If we set the value of deep to **False**, it will neither copy the indices nor the data.

25.) How will you create an empty DataFrame in Pandas?

Ans: A DataFrame is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns) It is defined as a standard way to store data and has two different indexes, i.e., row index and column index.

Create an empty DataFrame:

The below code shows how to create an empty DataFrame in Pandas:

```
1. # importing the pandas library
2. importpandas as pd
3. info = pd.DataFrame()
4. print (info)
```

Output:

Empty DataFrame

Columns: [ ]

Index: [ ]

---

26. How will you add a column to a pandas DataFrame?

Ans: We can add any new column to an existing DataFrame. The below code demonstrates how to add any new column to an existing DataFrame:

```
1. # importing the pandas library
2. import pandas as pd
3. info = {'one': pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
4. 'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}
5.
6. info = pd.DataFrame(info)
7.
8. # Add a new column to an existing DataFrame object
9.
10. print ("Add new column by passing series")
11. info['three']=pd.Series([20,40,60],index=['a','b','c'])
12. print (info)
13. print ("Add new column using existing DataFrame columns")
14. info['four']=info['one']+info['three']
```

15. print (info)

Output:

Add new column by passing series

one two three

a 1.0 1 20.0

b 2.0 2 40.0

c 3.0 3 60.0

d 4.0 4 NaN

e 5.0 5 NaN

f NaN 6 NaN

Add new column using existing DataFrame columns

one two three four

a 1.0 1 20.0 21.0

b 2.0 2 40.0 42.0

c 3.0 3 60.0 63.0

d 4.0 4 NaN NaN

e 5.0 5 NaN NaN

f NaN 6 NaN NaN

---

27. How to add an Index, row, or column to a Pandas DataFrame?

Ans: Adding an Index to a DataFrame

Pandas allow adding the inputs to the **index** argument if you create a DataFrame. It will make sure that you have the desired index. If you don?t specify inputs, the DataFrame contains, by default, a numerically valued index that starts with 0 and ends on the last row of the DataFrame.

Adding Rows to a DataFrame

We can use **.loc**, **iloc**, and **ix** to insert the rows in the DataFrame.

- The **loc** basically works for the labels of our index. It can be understood as if we insert in loc[4], which means we are looking for that values of DataFrame that have an index labeled 4.
- The **iloc** basically works for the positions in the index. It can be understood as if we insert in iloc[4], which means we are looking for the values of DataFrame that are present at index '4`.
- The **ix** is a complex case because if the index is integer-based, we pass a label to **ix**. The **ix[4]** means that we are looking in the DataFrame for those values that have an index labeled 4. However, if the index is not only integer-based, ix will deal with the positions as **iloc**.

Adding Columns to a DataFrame

If we want to add the column to the DataFrame, we can easily follow the same procedure as adding an index to the DataFrame by using loc or iloc.

Add row with specific index name:

```
import pandas as pd
employees = pd.DataFrame(
    data={'Name': ['John Doe', 'William Spark'],
          'Occupation': ['Chemist', 'Statistician'],
          'Date Of Join': ['2018-01-25', '2018-01-26'],
          'Age': [23, 24]},
```

```
            index=['Emp001', 'Emp002'],
            columns=['Name', 'Occupation', 'Date Of Join', 'Age'])
print("\n------------ BEFORE ----------------\n")
print(employees)
employees.loc['Emp003'] = ['Sunny', 'Programmer', '2018-01-25', 45]
print("\n------------ AFTER ----------------\n")
print(employees)
```

OUTPUT :

```
C:\pandas>python example22.py

------------ BEFORE ----------------

                Name    Occupation Date Of Join  Age
Emp001       John Doe       Chemist   2018-01-25   23
Emp002  William Spark  Statistician   2018-01-26   24

------------ AFTER ----------------

                Name    Occupation Date Of Join  Age
Emp001       John Doe       Chemist   2018-01-25   23
Emp002  William Spark  Statistician   2018-01-26   24
Emp003          Sunny    Programmer   2018-01-25   45
```

---

28. How to Delete Indices, Rows or Columns From a Pandas Data Frame?

Ans: Deleting an Index from Your DataFrame

If you want to remove the index from the DataFrame, you should have to do the following:

Reset the index of DataFrame.

Executing **del df.index.name** to remove the index name.

Remove duplicate index values by resetting the index and drop the duplicate values from the index column.

Remove an index with a row.

Deleting a Column from Your DataFrame

You can use the **drop()** method for deleting a column from the DataFrame.

The axis argument that is passed to the **drop()** method is either 0 if it indicates the rows and 1 if it drops the columns.

You can pass the argument **inplace** and set it to True to delete the column without reassign the DataFrame.

You can also delete the duplicate values from the column by using the drop_duplicates() method.

Removing a Row from Your DataFrame

By using **df.drop_duplicates()**, we can remove duplicate rows from the DataFrame.

You can use the **drop()** method to specify the index of the rows that we want to remove from the DataFrame.

---

29. How to Rename the Index or Columns of a Pandas DataFrame?

Ans: You can use the **.rename** method to give different values to the columns or the index values of DataFrame.

There are the following ways to change index / columns names (labels) of `pandas.DataFrame`.

- Use `pandas.DataFrame.rename()`
  - Change any index / columns names individually with `dict`
  - Change all index / columns names with a function
- Use `pandas.DataFrame.add_prefix()`,`pandas.DataFrame.add_suffix()`

- Add prefix and suffix to columns name

- Update the `index` / `columns` attributes of `pandas.DataFrame`
  - Replace all index / columns names

`set_index()` method that sets an existing column as an index is also provided. See the following post for detail.

Specify the original name and the new name in `dict` like `{original name: new name}` to `index` / `columns` of `rename()`.

`index` is for index name and `columns` is for the columns name. If you want to change either, you need only specify one of `index` or `columns`.

A new `DataFrame` is returned, the original `DataFrame` is not changed.

```
df_new = df.rename(columns={'A': 'a'}, index={'ONE': 'one'})
print(df_new)
#         a   B   C
# one    11  12  13
# TWO    21  22  23
# THREE  31  32  33

print(df)
#         A   B   C
# ONE    11  12  13
# TWO    21  22  23
# THREE  31  32  33
```

---

30. How to iterate over a Pandas DataFrame?

Ans. You can iterate over the rows of the DataFrame by using for loop in combination with an iterrows() call on the DataFrame.

```
import pandas as pd
 import numpy as np

df = pd.DataFrame([{'c1':10, 'c2':100}, {'c1':11,'c2':110}, {'c1':12,'c2':120}])

for index, row in df.iterrows():
    print(row['c1'], row['c2'])

Output:
   10 100
   11 110
   12 120
```

---

31. How to get the items of series A not present in series B?

Ans: We can remove items present in **p2** from **p1** using isin() method.

1. **import** pandas as pd
2. p1 = pd.Series([2, 4, 6, 8, 10])
3. p2 = pd.Series([8, 10, 12, 14, 16])
4. p1[~p1.isin(p2)]

Solution

0 2

1 4

2 6

dtype: int64

---

32. How to get the items not common to both series A and series B?

Ans: We get all the items of p1 and p2 not common to both using below example:

```
1. import pandas as pd
2. import numpy as np
3. p1 = pd.Series([2, 4, 6, 8, 10])
4. p2 = pd.Series([8, 10, 12, 14, 16])
5. p1[~p1.isin(p2)]
6. p_u = pd.Series(np.union1d(p1, p2))  # union
7. p_i = pd.Series(np.intersect1d(p1, p2))  # intersect
8. p_u[~p_u.isin(p_i)]
```

Output:

```
0  2
1  4
2  6
5  12
6  14
7  16
dtype: int64
```

---

33. How to get the minimum, 25th percentile, median, 75th, and max of a numeric series?

Ans: We can compute the minimum, 25th percentile, median, 75th, and maximum of **p** as below example:

```
1. import pandas as pd
2. import numpy as np
3. p = pd.Series(np.random.normal(14, 6, 22))
4. state = np.random.RandomState(120)
5. p = pd.Series(state.normal(14, 6, 22))
6. percentile(p, q=[0, 25, 50, 75, 100])
```

Output:

```
array([ 4.61498692, 12.15572753, 14.67780756, 17.58054104, 33.24975515])
```

---

34. How to get frequency counts of unique items of a series?

Ans: We can calculate the frequency counts of each unique value p as below example:

```
1. import pandas as pd
2. import numpy as np
3. p= pd.Series(np.take(list('pqrstu'), np.random.randint(6, size=17)))
4. p = pd.Series(np.take(list('pqrstu'), np.random.randint(6, size=17)))
5. value_counts()
```

Output:

```
s  4
r  4
q  3
p  3
u  3
```

---

35. How to convert a numpy array to a dataframe of given shape?

Ans. We can reshape the series **p** into a dataframe with 6 rows and 2 columns as below example:

```
1. import pandas as pd
2. import numpy as np
3. p = pd.Series(np.random.randint(1, 7, 35))
4. # Input
5. p = pd.Series(np.random.randint(1, 7, 35))
6. info = pd.DataFrame(p.values.reshape(7,5))
7. print(info)
```

Output:

```
0 1 2 3 4

0 3 2 5 5 1

1 3 2 5 5 5

2 1 3 1 2 6

3 1 1 1 2 2

4 3 5 3 3 3

5 2 5 3 6 4

6 3 6 6 6 5
```

---

36. How can we convert a Series to DataFrame?

Ans: The Pandas **Series.to_frame()** function is used to convert the series object to the DataFrame.

```
1. to_frame(name=None)
```

name: Refers to the object. Its Default value is None. If it has one value, the passed name will be substituted for the series name.

```
1. s = pd.Series(["a", "b", "c"],
2. name="vals")
3. to_frame()
```

Output:

```
vals

0       a

1       b

2       c
```

---

37. How can we sort the DataFrame?

Ans: We can efficiently perform sorting in the DataFrame through different kinds:

- By label
- By Actual value

1). By label

The DataFrame can be sorted by using the sort_index() method. It can be done by passing the axis arguments and the order of sorting. The sorting is done on row labels in ascending order by default.

Using the **sort_index()** method, by passing the axis arguments and the order of sorting, DataFrame can be sorted. By default, sorting is done on row labels in ascending order.

```
import pandas as pd
import numpy as np
```

```
unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,0,7],colu
   mns = ['col2','col1'])

sorted_df=unsorted_df.sort_index()
print sorted_df
```

Its **output** is as follows –

```
        col2       col1
0   0.208464   0.627037
1   0.641004   0.331352
2  -0.038067  -0.464730
3  -0.638456  -0.021466
4   0.014646  -0.737438
5  -0.290761  -1.669827
6  -0.797303  -0.018737
7   0.525753   1.628921
8  -0.567031   0.775951
9   0.060724  -0.322425
```

Order of Sorting

By passing the Boolean value to ascending parameter, the order of the sorting can be controlled. Let us consider the following example to understand the same.

```
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,0,7],colu
   mns = ['col2','col1'])

sorted_df = unsorted_df.sort_index(ascending=False)
print sorted_df
```

Its **output** is as follows –

```
         col2        col1
9    0.825697    0.374463
8   -1.699509    0.510373
7   -0.581378    0.622958
6   -0.202951    0.954300
5   -1.289321   -1.551250
4    1.302561    0.851385
3   -0.157915   -0.388659
2   -1.222295    0.166609
1    0.584890   -0.291048
0    0.668444   -0.061294
```

Sort the Columns

By passing the axis argument with a value 0 or 1, the sorting can be done on the column labels. By default, axis=0, sort by row. Let us consider the following example to understand the same.

```
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,0,7],colu
   mns = ['col2','col1'])

sorted_df=unsorted_df.sort_index(axis=1)

print sorted_df
```

Its **output** is as follows –

```
        col1      col2
1   -0.291048  0.584890
4    0.851385  1.302561
6    0.954300 -0.202951
2    0.166609 -1.222295
3   -0.388659 -0.157915
5   -1.551250 -1.289321
9    0.374463  0.825697
8    0.510373 -1.699509
0   -0.061294  0.668444
7    0.622958 -0.581378
```

2). By Actual Value

It is another kind through which sorting can be performed in the DataFrame. Like index sorting, **sort_values()** is a method for sorting the values.

It also provides a feature in which we can specify the column name of the DataFrame with which values are to be sorted. It is done by passing the '**by**' argument.

```
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame({'col1':[2,1,1,1],'col2':[1,3,2,4]})
   sorted_df = unsorted_df.sort_values(by='col1')

print sorted_df
```

Its **output** is as follows –

```
   col1  col2
1    1    3
2    1    2
3    1    4
0    2    1
```

Observe, col1 values are sorted and the respective col2 value and row index will alter along with col1. Thus, they look unsorted.

'by' argument takes a list of column values.

```
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame({'col1':[2,1,1,1],'col2':[1,3,2,4]})
   sorted_df = unsorted_df.sort_values(by=['col1','col2'])

print sorted_df
```

Its **output** is as follows –

```
  col1 col2
2   1   2
1   1   3
3   1   4
0   2   1
```

Sorting Algorithm

sort_values() provides a provision to choose the algorithm from mergesort, heapsort and quicksort. Mergesort is the only stable algorithm.

```
import pandas as pd
import numpy as np

unsorted_df = pd.DataFrame({'col1':[2,1,1,1],'col2':[1,3,2,4]})
sorted_df = unsorted_df.sort_values(by='col1' ,kind='mergesort')
```

```
print sorted_df
```

Its **output** is as follows –

```
  col1 col2
1    1    3
2    1    2
3    1    4
0    2    1
```

---

38. How to convert String to date?

Ans: The below code demonstrates how to convert the string to date:

```
1. From datetime import datetime
2.
3. # Define dates as the strings
4. dmy_str1 = 'Wednesday, July 14, 2018'
5. dmy_str2 = '14/7/17'
6. dmy_str3 = '14-07-2017'
7.
8. # Define dates as the datetime objects
9. dmy_dt1 = datetime.strptime(date_str1, '%A, %B %d, %Y')
10. dmy_dt2 = datetime.strptime(date_str2, '%m/%d/%y')
11. dmy_dt3 = datetime.strptime(date_str3, '%m-%d-%Y')
12.
13. #Print the converted dates
14. print(dmy_dt1)
15. print(dmy_dt2)
16. print(dmy_dt3)
```

Output:

2017-07-14 00:00:00

2017-07-14 00:00:00

2018-07-14 00:00:00

39.What is Data Aggregation?

Ans: The main task of Data Aggregation is to apply some aggregation to one or more columns. It uses the following:

- **sum:** It is used to return the sum of the values for the requested axis.
- **min:** It is used to return a minimum of the values for the requested axis.
- **max:** It is used to return a maximum values for the requested axis.

 Screenshot of the pandas aggregations

Examples

```
import pandas as pd
import numpy as np

df = pd.DataFrame([[1, 2, 3],
                   [4, 5, 6],
                    [7, 8, 9],
                   [np.nan, np.nan, np.nan]],
columns=['A', 'B', 'C'])

print(df)
```

```
# Aggregate these functions over the rows.

print(df.agg(['sum', 'min']))

# Different aggregations per column.

print(df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']}))

# Aggregate over the columns.

print(df.agg("mean", axis="columns"))

# Aggregate over the rows.

print(df.agg("mean", axis="rows"))
```

OUTPUT :

```
A  B    C
0  1.0  2  3.0
1  4.0  5  6.0
2  7.0  8  9.0
3  NaN  NaN  NaN

        A   B     C
sum  12.0  15.0  18.0
min   1.0  2.0   3.0

        A    B
max   NaN  8.0
min   1.0  2.0
sum  12.0  NaN

0    2.0
1    5.0
2    8.0
3    NaN
dtype: float64

A    4.0
B    5.0
C    6.0
dtype: float64
```

---

40. What is Pandas Index?

Ans:

Indexing in Pandas :
Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame. Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns. Indexing can also be known as **Subset Selection**.

Pandas Indexing using `[ ]`, `.loc[]`, `.iloc[ ]`, `.ix[ ]`

There are a lot of ways to pull the elements, rows, and columns from a DataFrame. There are some indexing method in Pandas which help in getting an element from a DataFrame. These indexing methods appear very similar but behave very differently. Pandas support four types of Multi-axes indexing they are:

1. **Dataframe.[ ] ;** This function also known as indexing operator
2. **Dataframe.loc[ ] :** This function is used for labels.
3. **Dataframe.iloc[ ] :** This function is used for positions or integer based

4. **Dataframe.ix[] :** This function is used for both label and integer based

Collectively, they are called the **indexers**. These are by far the most common ways to index data. These are four function which help in getting the elements, rows, and columns from a DataFrame.

1) Indexing a Dataframe using indexing operator `[]` :

Indexing operator is used to refer to the square brackets following an object. The `.loc` and `.iloc` indexers also use the indexing operator to make selections. In this indexing operator to refer to df[].

In order to select a single column, we simply put the name of the column in-between the brackets

filter_nonebrightness_4

```
# importing pandas package
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col ="Name")
# retrieving columns by indexing operator
first =data["Age"]
print(first)
```

Output:

```
Name
Avery Bradley              25.0
Jae Crowder                25.0
John Holland               27.0
R.J. Hunter                22.0
Jonas Jerebko              29.0
Amir Johnson               29.0
Jordan Mickey              21.0
Kelly Olynyk               25.0
Terry Rozier               22.0
Marcus Smart               22.0
Jared Sullinger            24.0
Isaiah Thomas              27.0
 .                          .
 .                          .
 .                          .
Joe Ingles                 28.0
Chris Johnson              26.0
Trey Lyles                 20.0
Shelvin Mack               26.0
Raul Neto                  24.0
Tibor Pleiss               26.0
Jeff Withey                26.0
NaN                         NaN
Name: Age, Length: 458, dtype: float64
```

2. Indexing a DataFrame using `.loc[ ]` :

This function selects data by the **label** of the rows and columns. The `df.loc` indexer selects data in a different way than just the indexing operator. It can select subsets of rows or columns. It can also simultaneously select subsets of rows and columns.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving row by loc method
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]


print(first, "\n\n\n", second)
```

Output:

As shown in the output image, two series were returned since there was only one parameter both of the times.

```
Team        Boston Celtics
Number                   0
Position                PG
Age                     25
Height                 6-2
Weight                 180
College              Texas
Salary        7.73034e+06
Name: Avery Bradley, dtype: object
```

```
 Team        Boston Celtics
Number                   28
Position                 SG
Age                      22
Height                  6-5
Weight                  185
College       Georgia State
Salary          1.14864e+06
Name: R.J. Hunter, dtype: object
```

3. Indexing a DataFrame using `.iloc[ ]` :

This function allows us to retrieve rows and columns by position. In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well. The `df.iloc` indexer is very similar to `df.loc` but only uses integer locations to make its selections.

In order to select a single row using `.iloc[]`, we can pass a single integer to `.iloc[]` function.

```
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col ="Name")
# retrieving rows by iloc method
row2 =data.iloc[3]
print(row2)
```

Output :

```
Team        Boston Celtics
Number                  28
Position                SG
Age                     22
Height                 6-5
Weight                 185
College       Georgia State
Salary          1.14864e+06
Name: R.J. Hunter, dtype: object
```

---

4. Indexing a using Dataframe.ix[ ] :

Early in the development of pandas, there existed another indexer, `ix`. This indexer was capable of selecting both by label and by integer location. While it was versatile, it caused lots of confusion because it's not explicit. Sometimes integers can also be labels for rows or columns. Thus there were instances where it was ambiguous. Generally, `ix` is label based and acts just as the **.loc** indexer. However, `.ix` also supports integer type selections (as in .iloc) where passed an integer. This only works where the index of the DataFrame is not integer based `.ix` will accept any of the inputs of `.loc` and `.iloc`.

Note: The **.ix** indexer has been deprecated in recent versions of Pandas.

Selecting a single row using `.ix[]` as `.loc[]`

In order to select a single row, we put a single row label in a `.ix` function. This function act similar as .loc[ ] if we pass a row label as a argument of a function.

```
# importing pandas package
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col ="Name")
# retrieving row by ix method
first =data.ix["Avery Bradley"]
print(first)
```

Output :

```
Team        Boston Celtics
Number                   0
Position                PG
Age                     25
Height                 6-2
Weight                 180
College              Texas
Salary        7.73034e+06
Name: Avery Bradley, dtype: object
```

---

41. Define ReIndexing?

Ans: Reindexing changes the row labels and column labels of a DataFrame. To *reindex* means to conform the data to match a given set of labels along a particular axis.

Multiple operations can be accomplished through indexing like –

- Reorder the existing data to match a new set of labels.
- Insert missing value (NA) markers in label locations where no data for the label existed.

Example

```
import pandas as pd
import numpy as np

N=20

df = pd.DataFrame({
   'A': pd.date_range(start='2016-01-01',periods=N,freq='D'),
   'x': np.linspace(0,stop=N-1,num=N),
   'y': np.random.rand(N),
   'C': np.random.choice(['Low','Medium','High'],N).tolist(),
   'D': np.random.normal(100, 10, size=(N)).tolist()
})

#reindex the DataFrame
df_reindexed = df.reindex(index=[0,2,5], columns=['A', 'C', 'B'])

print df_reindexed
```

Its **output** is as follows –

```
           A    C     B
0  2016-01-01  Low   NaN
2  2016-01-03  High  NaN
5  2016-01-06  Low   NaN
```

42. Define Multiple Indexing?

Ans: Multiple indexing is defined as essential indexing because it deals with data analysis and manipulation, especially for working with higher dimensional data. It also enables us to store and manipulate data with the arbitrary number of dimensions in lower-dimensional data structures like Series and DataFrame.

Multiple index Column

In this example, two columns will be made as index column. Drop parameter is used to Drop the column and append parameter is used to append passed columns to the already existing index column.

filter_nonebrightness_4

```
# importing pandas package
import pandas as pd

# making data frame from csv file
```

```
data = pd.read_csv("employees.csv")

# setting first name as index column
data.set_index(["First Name", "Gender"], inplace = True,
                               append = True, drop = False)

# display
data.head()
```

Output:
As shown in the output Image, the data is having 3 index columns.

			First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
	First Name	Gender								
0	Douglas	Male	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

43. How to Set the index?

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. ***Pandas*** is one of those packages and makes importing and analyzing data much easier.

Pandas `set_index()` is a method to set a List, Series or Data frame as index of a Data Frame. Index column can be set while making a data frame too. But sometimes a data frame is made out of two or more data frames and hence later index can be changed using this method.

Syntax:

> **DataFrame.set_index(keys, drop=True, append=False, inplace=False, verify_integrity=False**

Parameters:

> **keys:** Column name or list of column name.
> **drop:** Boolean value which drops the column used for index if True.
> **append:** Appends the column to existing index column if True.
> **inplace:** Makes the changes in the dataframe if True.
> **verify_integrity:** Checks the new index column for duplicates if True.

Changing Index column
In this example, First Name column has been made the index column of Data Frame.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# setting first name as index column
data.set_index("First Name", inplace = True)

# display
data.head()
```

Output:
As shown in the output images, earlier the index column was a series of number but later it has been replaced with First name.
Before Operation –

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

After Operation

First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

44. How to Reset the index?

Ans: Pandas series is a One-dimensional ndarray with axis labels. The labels need not be unique but must be a hashable type. The object supports both integer- and label-based indexing and provides a host of methods for performing operations involving the index.

Pandas **`Series.reset_index()`** function generate a new DataFrame or Series with the index reset. This comes handy when index is need to be used as a column.

Syntax: Series.reset_index(level=None, drop=False, name=None, inplace=False)

Parameter :

level : For a Series with a MultiIndex

drop : Just reset the index, without inserting it as a column in the new DataFrame.

name : The name to use for the column containing the original Series values.

inplace : Modify the Series in place

Returns : result : Series

Example #1: Use `Series.reset_index()` function to reset the index of the given Series object.

```
# importing pandas as pd
import pandas as pd

# Creating the Series
sr = pd.Series([10, 25, 3, 11, 24, 6])

# Create the Index
index_ = ['Coca Cola', 'Sprite', 'Coke', 'Fanta', 'Dew', 'ThumbsUp']

# set the index
sr.index = index_

# Print the series
print(sr)
```

Output:

```
Coca Cola    10
Sprite       25
Coke          3
Fanta        11
Dew          24
ThumbsUp      6
dtype: int64
```

Now we will use `Series.reset_index()` function to reset the index of the given series object.

```
# reset the index
result = sr.reset_index()
```

```
# Print the result
print(result)
```

Output :

	index	0
0	Coca Cola	10
1	Sprite	25
2	Coke	3
3	Fanta	11
4	Dew	24
5	ThumbsUp	6

As we can see in the output, the `Series.reset_index()` function has reset the index of the given Series object to default. It has preserved the index and it has converted it to a column.

---

45. Describe Data Operations in Pandas?

Ans: In Pandas, there are different useful data operations for DataFrame, which are as follows:

- **Row and column selection**

We can select any row and column of the DataFrame by passing the name of the rows and columns. When you select it from the DataFrame, it becomes one-dimensional and considered as Series.

- **Filter Data**

We can filter the data by providing some of the boolean expressions in DataFrame.

- **Null values**

A Null value occurs when no data is provided to the items. The various columns may contain no values, which are usually represented as NaN.

---

46. Define GroupBy in Pandas?

Ans: Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. ***Pandas*** is one of those packages and makes importing and analyzing data much easier.

Pandas **`dataframe.groupby()`** function is used to split the data into groups based on some criteria. pandas objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names.

Syntax: DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)

Parameters :

by : mapping, function, str, or iterable
axis : int, default 0
level : If the axis is a MultiIndex (hierarchical), group by a particular level or levels
as_index : For aggregated output, return object with group labels as the index. Only relevant for DataFrame input. as_index=False is effectively "SQL-style" grouped output
sort : Sort group keys. Get better performance by turning this off. Note this does not influence the order of observations within each group. groupby preserves the order of rows within each group.
group_keys : When calling apply, add group keys to index to identify pieces
squeeze : Reduce the dimensionality of the return type if possible, otherwise return a consistent type

Returns : GroupBy object

---

47. How will you append new rows to a pandas DataFrame?

Ans: Pandas `dataframe.append()` function is used to append rows of other dataframe to the end of the given dataframe, returning a new dataframe object. Columns not in the original dataframes are added as new columns and the new cells are populated with `NaN` value.

> **Syntax:** DataFrame.append(other, ignore_index=False, verify_integrity=False, sort=None)
> **Parameters :**
> **other :** DataFrame or Series/dict-like object, or list of these
> **ignore_index :** If True, do not use the index labels.
> **verify_integrity :** If True, raise ValueError on creating index with duplicates.
> **sort :** Sort columns if the columns of self and other are not aligned. The default sorting is deprecated and will change to not-sorting in a future version of pandas. Explicitly pass sort=True to silence the warning and sort. Explicitly pass sort=False to silence the warning and not sort.
> **Returns:** appended : DataFrame

Example #1: Create two data frames and append the second to the first one.

```
# Importing pandas as pd
importpandas as pd
# Creating the first Dataframe using dictionary
df1 =df =pd.DataFrame({"a":[1, 2, 3, 4],
"b":[5, 6, 7, 8]})
# Creating the Second Dataframe using dictionary
df2 =pd.DataFrame({"a":[1, 2, 3],
"b":[5, 6, 7]})
# Print df1
print(df1, "\n")
# Print df2
df2
```

	a	b
0	1	5
1	2	6
2	3	7
3	4	8

	a	b
0	1	5
1	2	6
2	3	7

Now append df2 at the end of df1.

```
# to append df2 at the end of df1 dataframe
df1.append(df2)
```

	a	b
0	1	5
1	2	6
2	3	7
3	4	8
0	1	5
1	2	6
2	3	7

Notice the index value of second data frame is maintained in the appended data frame. If we do not want it to happen then we can set ignore_index=True.

```
# A continuous index value will be maintained
# across the rows in the new appended data frame.
df.append(df2, ignore_index = True)
```

Output :

	a	b
0	1	5
1	2	6
2	3	7
3	4	8
4	1	5
5	2	6
6	3	7

48. How will you delete rows from a pandas DataFrame?

Ans:

Import modules

```
import pandas as pd
```

Create a dataframe

```
data = {'name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'year': [2012, 2012, 2013, 2014, 2014],
        'reports': [4, 24, 31, 2, 3]}
df = pd.DataFrame(data, index = ['Cochice', 'Pima', 'Santa Cruz', 'Maricopa', 'Yuma'])
df
```

	name	reports	year
Cochice	Jason	4	2012
Pima	Molly	24	2012
Santa Cruz	Tina	31	2013
Maricopa	Jake	2	2014
Yuma	Amy	3	2014

Delete a row

```
df.drop(['Cochice', 'Pima'])
```

```
Output :
```

	name	reports	year
Santa Cruz	Tina	31	2013
Maricopa	Jake	2	2014
Yuma	Amy	3	2014

49. How will you get the number of rows and columns of a DataFrame in pandas?

Ans:

```
import pandas as pd
import numpy as np
```

```
raw_data = {'name': ['Willard Morris', 'Al Jennings', 'Omar Mullins', 'Spencer McDaniel'],
'age': [20, 19, 22, 21],
'favorite_color': ['blue', 'red', 'yellow', "green"],
'grade': [88, 92, 95, 70]}
df = pd.DataFrame(raw_data, columns = ['name', 'age', 'favorite_color', 'grade'])
df
```

Output:

	name	age	favorite_color	grade
0	Willard Morris	20	blue	88
1	Al Jennings	19	red	92
2	Omar Mullins	22	yellow	95
3	Spencer McDaniel	21	green	70

get the row and column count of the df

```
df.shape
```

```
(4, 4)
```

---

50. What is Pandas ml?

Ans: pandas_ml is a package which integrates pandas, scikit-learn, xgboost into one package for easy handling of data and creation of machine learning models

Installation

$ pip install pandas_ml

Example

```
>>> import pandas_ml as pdml
>>> import sklearn.datasets as datasets
# create ModelFrame instance from sklearn.datasets
>>> df = pdml.ModelFrame(datasets.load_digits())
>>> type(df)
<class 'pandas_ml.core.frame.ModelFrame'>
# binarize data (features), not touching target
>>> df.data = df.data.preprocessing.binarize()
>>> df.head()
.target  0  1  2  3  4  5  6  7  8 ...  54  55  56  57  58  59  60  61  62  63
0        0  0  0  1  1  1  1  0  0  0 ...   0   0   0   0   1   1   1   0   0   0
1        1  0  0  0  1  1  1  0  0  0 ...   0   0   0   0   0   1   1   1   0   0
2        2  0  0  0  1  1  1  0  0  0 ...   1   0   0   0   0   1   1   1   1   0
3        3  0  0  1  1  1  1  0  0  0 ...   1   0   0   0   1   1   1   1   0   0
4        4  0  0  0  1  1  0  0  0  0 ...   0   0   0   0   0   1   1   1   0   0
[5 rows x 65 columns]
# split to training and test data
>>> train_df, test_df = df.model_selection.train_test_split()
# create estimator (accessor is mapped to sklearn namespace)
>>> estimator = df.svm.LinearSVC()
# fit to training data
```

```
>>> train_df.fit(estimator)
# predict test data
>>> test_df.predict(estimator)
0     4
1     2
2     7
...
448    5
449    8
Length: 450, dtype: int64
# Evaluate the result
>>> test_df.metrics.confusion_matrix()
Predicted   0   1   2   3   4   5   6   7   8   9
Target
0          52   0   0   0   0   0   0   0   0   0
1           0  37   1   0   0   1   0   0   3   3
2           0   2  48   1   0   0   0   1   1   0
3           1   1   0  44   0   1   0   0   3   1
4           1   0   0   0  43   0   1   0   0   0
5           0   1   0   0   0  39   0   0   0   0
6           0   1   0   0   1   0  35   0   0   0
7           0   0   0   0   2   0   0  42   1   0
8           0   2   1   0   1   0   0   0  33   1
9           0   2   1   2   0   0   0   0   1  38
```

---

51. What is Pandas Charm?

Ans: pandas-charm is a small Python package for getting character matrices (alignments) into and out of pandas. Use this library to make pandas interoperable with BioPython and DendroPy.

Convert between the following objects:

- BioPython Multiple Seq Alignment <-> pandas DataFrame
- DendroPy Character Matrix <-> pandas DataFrame
- “Sequence dictionary” <-> pandas DataFrame

The code has been tested with Python 2.7, 3.5 and 3.6.

Installation :

```
$ pip install pandas-charm
```

You may consider installing pandas-charm and its required Python packages within a virtual environment in order to avoid cluttering your system’s Python path. See for example the environment management system conda or the package virtualenv.

Running the tests

Testing is carried out with pytest:

```
$ pytest -v test_pandascharm.py
```

Test coverage can be calculated with Coverage.py using the following commands:

```
$ coverage run -m pytest
$ coverage report -m pandascharm.py
```

The code follow style conventions in PEP8, which can be checked with pycodestyle:

```
$ pycodestyle pandascharm.py test_pandascharm.py setup.py
```

Usage

The following examples show how to use pandas-charm. The examples are written with Python 3 code, but pandas-charm should work also with Python 2.7+. You need to install BioPython and/or DendroPy manually before you start:

```
$ pip install biopython
$ pip install dendropy
```

DendroPy CharacterMatrix to pandas DataFrame

```
>>> import pandas as pd
>>> import pandascharm as pc
>>> import dendropy
>>> dna_string = '3 5\nt1  TCCAA\nt2  TGCAA\nt3  TG-AA\n'
>>> print(dna_string)
3 5
t1  TCCAA
t2  TGCAA
t3  TG-AA
>>> matrix = dendropy.DnaCharacterMatrix.get(
...     data=dna_string, schema='phylip')
>>> df = pc.from_charmatrix(matrix)
>>> df
  t1 t2 t3
0  T  T  T
1  C  G  G
2  C  C  -
3  A  A  A
4  A  A  A
```

By default, characters are stored as rows and sequences as columns in the DataFrame. If you want rows to hold sequences, just transpose the matrix in pandas:

```
>>> df.transpose()
    0  1  2  3  4
t1  T  C  C  A  A
t2  T  G  C  A  A
t3  T  G  -  A  A
```

---

52. How will you add a scalar column with same value for all rows to a pandas DataFrame?

Ans:

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric Python packages. Pandas is one of those packages and makes importing and analyzing data much easier.

`Dataframe.add()` method is used for addition of dataframe and other, element-wise (binary operator add). Equivalent to dataframe + other, but with support to substitute a fill_value for missing data in one of the inputs.

Syntax: DataFrame.add(other, axis='columns', level=None, fill_value=None)

Parameters:

other :Series, DataFrame, or constant
axis :{0, 1, 'index', 'columns'} For Series input, axis to match Series index on
fill_value : [None or float value, default None] Fill missing (NaN) values with this value. If both DataFrame locations are missing, the result will be missing.
level : [int or name] Broadcast across a level, matching Index values on the passed MultiIndex level

Returns: result DataFrame

```
# Importing Pandas as pd
import pandas as pd

# Importing numpy as np
import numpy as np
```

```
# Creating a dataframe
# Setting the seed value to re-generate the result.
np.random.seed(25)

df = pd.DataFrame(np.random.rand(10, 3), columns =['A', 'B', 'C'])

# np.random.rand(10, 3) has generated a
# random 2-Dimensional array of shape 10 * 3
# which is then converted to a dataframe

df
```

Output :

Out[39]:

	A	B	C
0	0.870124	0.582277	0.278839
1	0.185911	0.411100	0.117376
2	0.684969	0.437611	0.556229
3	0.367080	0.402366	0.113041
4	0.447031	0.585445	0.161985
5	0.520719	0.326051	0.699186
6	0.366395	0.836375	0.481343
7	0.516502	0.383048	0.997541
8	0.514244	0.559053	0.034450
9	0.719930	0.421004	0.436935

---

53. How can we select a column in pandas DataFrame?

Ans:

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric Python packages. ***Pandas*** is one of those packages and makes importing and analyzing data much easier.

Let's discuss all different ways of selecting multiple columns in a pandas DataFrame.

Method #1: Basic Method

Given a dictionary which contains Employee entity as keys and list of those entity as values.

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

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

# select two columns
df[['Name', 'Qualification']]
```

Output:

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

Select Second to fourth column.

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

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

# select all rows
# and second to fourth column
df[df.columns[1:4]]
```

Output :

	Age	Address	Qualification
0	27	Delhi	Msc
1	24	Kanpur	MA
2	22	Allahabad	MCA
3	32	Kannauj	Phd

---

54. How can we retrieve a row in pandas DataFrame ?

Ans: Pandas provide a unique method to retrieve rows from a Data frame. `DataFrame.loc[]` method is a method that takes only index labels and returns row or dataframe if the index label exists in the caller data frame.

Syntax: pandas.DataFrame.loc[ ]

Parameters:
Index label: String or list of string of index label of rows

Return type: Data frame or Series depending on parameters

Example #1 : Extracting single Row

In this example, Name column is made as the index column and then two single rows are extracted one by one in the form of series using index label of rows.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving row by loc method
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]


print(first, "\n\n\n", second)
```

Output:
As shown in the output image, two series were returned since there was only one parameter both of the times.

```
Team        Boston Celtics
Number                   0
Position                PG
Age                     25
Height                 6-2
Weight                 180
College              Texas
Salary         7.73034e+06
Name: Avery Bradley, dtype: object


 Team        Boston Celtics
Number                   28
Position                 SG
Age                      22
Height                  6-5
Weight                  185
College       Georgia State
Salary          1.14864e+06
Name: R.J. Hunter, dtype: object
```

Example #2: Multiple parameters

In this example, Name column is made as the index column and then two single rows are extracted at the same time by passing a list as parameter.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving rows by loc method
rows = data.loc[["Avery Bradley", "R.J. Hunter"]]

# checking data type of rows
print(type(rows))

# display
rows
```

Output:
As shown in the output image, this time the data type of returned value is a data frame. Both of the rows were extracted and displayed like a new data frame.

Name	Team	Number	Position	Age	Height	Weight	College	Salary
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0

55. How will you convert a DataFrame to an array in pandas?

Ans:

For performing some high-level mathematical functions, we can convert Pandas DataFrame to numpy arrays. It uses the DataFrame.to_numpy() function.

The **DataFrame.to_numpy()** function is applied on the DataFrame that returns the numpy ndarray.

Syntax:

DataFrame.to_numpy(dtype=None, copy=False)

Parameters

- **dtype:** It is an optional parameter that pass the dtype to numpy.asarray().
- **copy:** It returns the boolean value that has the default value **False**.

It ensures that the returned value is not a view on another array.

Returns

It returns the **numpy.ndarray** as an output.

Example1:

```
1. import pandas as pd
2. pd.DataFrame({"P": [2, 3], "Q": [4, 5]}).to_numpy()
3. info = pd.DataFrame({"P": [2, 3], "Q": [4.0, 5.8]})
4. info.to_numpy()
5. info['R'] = pd.date_range('2000', periods=2)
6. info.to_numpy()
```

Output :

```
array([[2, 4.0, Timestamp('2000-01-01 00:00:00')],
[3, 5.8, Timestamp('2000-01-02 00:00:00')]], dtype=object)
```

Example 2:

```
1. import pandas as pd
2. #initializing the dataframe
3. info = pd.DataFrame([[17, 62, 35],[25, 36, 54],[42, 20, 15],[48, 62, 76]],
4. columns=['x', 'y', 'z'])
5. print('DataFrame\n———-\n', info)
6. #convert the dataframe to a numpy array
7. arr = info.to_numpy()
8. print('\nNumpy Array\n———-\n', arr)
```

Output:

```
DataFrame
----------
x    y   z
0  17  62  35
1  25  36  54
2  42  20  15
3  48  62  76
Numpy Array
----------
[[17 62 35]
[25 36 54]
[42 20 15]
[48 62 76]]
```

---

56. How can you check if a DataFrame is empty in pandas?

Ans : Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. It can be thought of as a dict-like container for Series objects. This is the primary data structure of the Pandas.

Pandas **`DataFrame.empty`** attribute checks if the dataframe is empty or not. It return `True` if the dataframe is empty else it return `False`.

Syntax: DataFrame.empty

Parameter : None

Returns : bool

Example #1: Use `DataFrame.empty` attribute to check if the given dataframe is empty or not

```
# importing pandas as pd
import pandas as pd

# Creating the DataFrame
df = pd.DataFrame({'Weight':[45, 88, 56, 15, 71],
                   'Name':['Sam', 'Andrea', 'Alex', 'Robin', 'Kia'],
                   'Age':[14, 25, 55, 8, 21]})

# Create the index
index_ = ['Row_1', 'Row_2', 'Row_3', 'Row_4', 'Row_5']

# Set the index
df.index = index_

# Print the DataFrame
print(df)
```

Output :

	Weight	Name	Age
Row_1	45	Sam	14
Row_2	88	Andrea	25
Row_3	56	Alex	55
Row_4	15	Robin	8
Row_5	71	Kia	21

Now we will use `DataFrame.empty` attribute to check if the given dataframe is empty or not.

```
# check if there is any element
# in the given dataframe or not
```

```
result = df.empty
# Print the result
print(result)
```

Output :

```
False
```

As we can see in the output, the `DataFrame.empty` attribute has returned `False` indicating that the given dataframe is not empty.

Example #2: Use `DataFrame.empty` attribute to check if the given dataframe is empty or not.

```
# importing pandas as pd
import pandas as pd
# Creating an empty DataFrame
df = pd.DataFrame(index = ['Row_1', 'Row_2', 'Row_3', 'Row_4', 'Row_5'])
# Print the DataFrame
print(df)
```

Output :

```
Row_1
Row_2
Row_3
Row_4
Row_5
```

Now we will use `DataFrame.empty` attribute to check if the given dataframe is empty or not.

```
# check if there is any element
# in the given dataframe or not
result = df.empty
# Print the result
print(result)
```

Output :

```
True
```

As we can see in the output, the `DataFrame.empty` attribute has returned `True` indicating that the given dataframe is empty.

---

57. How can you get the sum of values of a column in pandas DataFrame?

Ans: Pandas `dataframe.sum()` function return the sum of the values for the requested axis. If the input is index axis then it adds all the values in a column and repeats the same for all the columns and returns a series containing the sum of all the values in each column. It also provides support to skip the missing values in the dataframe while calculating the sum in the dataframe.

Syntax: DataFrame.sum(axis=None, skipna=None, level=None, numeric_only=None, min_count=0, **kwargs)

Parameters :
axis : {index (0), columns (1)}
skipna : Exclude NA/null values when computing the result.
level : If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series
numeric_only : Include only float, int, boolean columns. If None, will attempt to use everything, then use only numeric data. Not implemented for Series.
min_count : The required number of valid values to perform the operation. If fewer than min_count non-NA values are present the result will be NA.

Returns : sum : Series or DataFrame (if level specified)

Example #1: Use `sum()` function to find the sum of all the values over the index axis.

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# Print the dataframe
df
```

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0
14	Tyler Zeller	Boston Celtics	44.0	C	26.0	7-0	253.0	North Carolina	2616975.0
15	Bojan Bogdanovic	Brooklyn Nets	44.0	SG	27.0	6-8	216.0	NaN	3425510.0
16	Markel Brown	Brooklyn Nets	22.0	SG	24.0	6-3	190.0	Oklahoma State	845059.0
17	Wayne Ellington	Brooklyn Nets	21.0	SG	28.0	6-4	200.0	North Carolina	1500000.0

Now find the sum of all values along the index axis. We are going to skip the `NaN` values in the calculation of the sum.

```
# finding sum over index axis
# By default the axis is set to 0
df.sum(axis = 0, skipna = True)
```

Output:

```
Number    8.079000e+03
Age       1.231100e+04
Weight    1.012360e+05
Salary    2.159837e+09
dtype: float64
```

---

58. How will you get the average of values of a column in pandas DataFrame?

Ans:

Pandas **`dataframe.mean()`** function return the mean of the values for the requested axis. If the method is applied on a pandas series object, then the method returns a scalar value which is the mean value of all the observations in the dataframe. If the method is applied on a pandas dataframe object, then the method returns a pandas series object which contains the mean of the values over the specified axis.

Syntax: DataFrame.mean(axis=None, skipna=None, level=None, numeric_only=None, **kwargs)

Parameters :
axis : {index (0), columns (1)}
skipna : Exclude NA/null values when computing the result

level : If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series

numeric_only : Include only float, int, boolean columns. If None, will attempt to use everything, then use only numeric data. Not implemented for Series.

Returns : mean : Series or DataFrame (if level specified)

Example : Use `mean()` function to find the mean of all the observations over the index axis.

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.DataFrame({"A":[12, 4, 5, 44, 1],
                   "B":[5, 2, 54, 3, 2],
                   "C":[20, 16, 7, 3, 8],
                   "D":[14, 3, 17, 2, 6]})

# Print the dataframe
df
```

	A	B	C	D
0	12	5	20	14
1	4	2	16	3
2	5	54	7	17
3	44	3	3	2
4	1	2	8	6

Let's use the `dataframe.mean()` function to find the mean over the index axis.

```
# Even if we do not specify axis = 0,
# the method will return the mean over
# the index axis by default
df.mean(axis = 0)
```

Output:

```
A    13.2
B    13.2
C    10.8
D     8.4
dtype: float64
```

---

59. How will you apply a function to every data element in a DataFrame?

Ans:

One can use `apply()` function in order to apply function to every row in given dataframe. Let's see the ways we can do this task.

Example

```
# Import pandas package
import pandas as pd

# Function to add
def add(a, b, c):
    return a + b + c

def main():
```

```
    # create a dictionary with
    # three fields each
    data = {
            'A':[1, 2, 3],
            'B':[4, 5, 6],
            'C':[7, 8, 9] }

    # Convert the dictionary into DataFrame
    df = pd.DataFrame(data)
    print("Original DataFrame:\n", df)

    df['add'] = df.apply(lambda row : add(row['A'],
                    row['B'], row['C']), axis = 1)

    print('\nAfter Applying Function: ')
    # printing the new dataframe
    print(df)

if __name__ == '__main__':
    main()
```

Output:

```
Original DataFrame:
    A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

After Applying Function:
   A  B  C  add
0  1  4  7   12
1  2  5  8   15
2  3  6  9   18
```

---

60. How will you get the top 2 rows from a DataFrame in pandas?

Select the first 2 rows of the Dataframe

dfObj1 = empDfObj.head(2)
print("First 2 rows of the Dataframe : ")
print(dfObj1)

Output:

First 2 rows of the Dataframe :
Name Age City Experience
a jack 34 Sydney 5
b Riti 31 Delhi 7

---

61. List major features of the Python pandas?

Ans:

Some of the major features of Python Pandas are,

- Fast and efficient in handling the data with its DataFrame object.
- It provides tools for loading data into in-memory data objects from various file formats.
- It has high-performance in merging and joining data.
- It has Time Series functionality.
- It provides functions for Data set merging and joining.
- It has functionalities for label-based slicing, fancy indexing, and subsetting of large data sets.
- It provides functionalities for reshaping and pivoting of data sets.

---

62. Enlist different types of Data Structures available in Pandas?

Ans: Different types of data structures available in Pandas are,

Series – It is immutable in size and homogeneous one-dimensional array data structure.

DataFrame – It is a tabular data structure which comprises of rows and columns. Here, data and size are mutable.

Panel – It is a three-dimensional data structure to store the data heterogeneously.

---

63. How To Write a Pandas DataFrame to a File

When you have done your data munging and manipulation with Pandas, you might want to export the DataFrame to another format. This section will cover two ways of outputting your DataFrame: to a CSV or to an Excel file.

Outputting a DataFrame to CSV
To output a Pandas DataFrame as a CSV file, you can use to_csv().

Writing a DataFrame to Excel
Very similar to what you did to output your DataFrame to CSV, you can use to_excel() to write your table to Excel.

---

64. When, Why And How You Should Reshape Your Pandas DataFrame

Ans: Reshaping your DataFrame is basically transforming it so that the resulting structure makes it more suitable for your data analysis.

In other words, reshaping is not so much concerned with formatting the values that are contained within the DataFrame, but more about transforming the shape of it.

This answers the when and why. Now onto the how of reshaping your DataFrame.

There are three ways of reshaping that frequently raise questions with users: pivoting, stacking and unstacking and melting.

Keep on reading to find out more!

Remember that if you want to see code examples and want to practice your DataFrame skills in our interactive DataCamp environment, go here.

Pivoting Your DataFrame
You can use the pivot() function to create a new derived table out of your original one. When you use the function, you can pass three arguments:

- Values: this argument allows you to specify which values of your original DataFrame you want to see in your pivot table.
- Columns: whatever you pass to this argument will become a column in your resulting table.
- Index: whatever you pass to this argument will become an index in your resulting table.

When you don't specifically fill in what values you expect to be present in your resulting table, you will pivot by multiple columns. Note that your data can not have rows with duplicate values for the columns that you specify. If this is not the case, you will get an error message. If you can't ensure the uniqueness of your data, you will want to use the pivot_table method instead .

Using stack() and unstack() to Reshape Your Pandas DataFrame
You have already seen an example of stacking in the answer to question 5!

Good news, you already know why you would use this and what you need to do to do it.

To repeat, when you stack a DataFrame, you make it taller. You move the innermost column index to become the innermost row index. You return a DataFrame with an index with a new inner-most level of row labels.

Go back to the full walk-through of the answer to question 5 “Splitting Text Into Multiple Columns” if you’re unsure of the workings of `stack().

The inverse of stacking is called unstacking. Much like stack(), you use unstack() to move the innermost row index to become the innermost column index.

Reshaping Your DataFrame With Melt()
Melting is considered to be very useful for when you have a data that has one or more columns that are identifier variables, while all other columns are considered measured variables.

These measured variables are all “unpivoted” to the row axis. That is, while the measured variables that were spread out over the width of the DataFrame, the melt will make sure that they will be placed in the height of it. Or, yet in other words, your DataFrame will now become longer instead of wider.

As a result, you just have two non-identifier columns, namely, ‘variable’ and ‘value’.

---

65. Does Pandas Recognize Dates When Importing Data?

Ans:

Pandas can recognize it, but you need to help it a tiny bit: add the argument parse_dates when you’reading in data from, let’s say, a comma-separated value (CSV) file.

There are, however, always weird date-time formats.

(Honestly, who has never had this?)

In such cases, you can construct your own parser to deal with this. You could, for example, make a lambda function that takes your DateTime and controls it with a format string.

---

66. How To Format The Data in Your Pandas DataFrame?

Ans:

Most of the times, you will also want to be able to do some operations on the actual values that are in your DataFrame.

Keep on reading to find out what the most common Pandas questions are when it comes to formatting your DataFrame’s values!

Replacing All Occurrences of a String in a DataFrame
To replace certain Strings in your DataFrame, you can easily use replace(): pass the values that you would like to change, followed by the values you want to replace them by.

Note that there is also a regex argument that can help you out tremendously when you’re faced with strange string combinations. In short, replace() is mostly what you need to deal with when you want to replace values or strings in your DataFrame by others.

Removing Parts From Strings in the Cells of Your DataFrame
Removing unwanted parts of strings is cumbersome work. Luckily, there is a solution in place! You use map() on the column result to apply the lambda function over each element or element-wise of the column. The function in itself takes the string value and strips the + or — that’s located on the left, and also strips away any of the six aAbBcC on the right.

Splitting Text in a Column into Multiple Rows in a DataFrame
Splitting your text into multiple rows is quite complex. For a complete walkthrough, go here.

Applying A Function to Your Pandas DataFrame’s Columns or Rows
You might want to adjust the data in your DataFrame by applying a function to it. Go to this page for the code chunks that explain how to apply a function to a DataFrame.

---

67. How To Add an Index, Row or Column to a Pandas DataFrame?

Ans: Now that you have learned how to select a value from a DataFrame, it's time to get to the real work and add an index, row or column to it!

Adding an Index to a DataFrame
When you create a DataFrame, you have the option to add input to the 'index' argument to make sure that you have the index that you desire. When you don't specify this, your DataFrame will have, by default, a numerically valued index that starts with 0 and continues until the last row of your DataFrame.

However, even when your index is specified for you automatically, you still have the power to re-use one of your columns and make it your index. You can easily do this by calling set_index() on your DataFrame.

Adding Rows to a DataFrame
Before you can get to the solution, it's first a good idea to grasp the concept of loc and how it differs from other indexing attributes such as .iloc and .ix:

- loc works on labels of your index. This means that if you give in loc[2], you look for the values of your DataFrame that have an index labeled 2.
- iloc works on the positions in your index. This means that if you give in iloc[2], you look for the values of your DataFrame that are at index '2`.
- ix is a more complex case: when the index is integer-based, you pass a label to ix. ix[2] then means that you're looking in your DataFrame for values that have an index labeled 2. This is just like loc! However, if your index is not solely integer-based, ix will work with positions, just like iloc.

Now that the difference between iloc, loc and ix is clear, you are ready to give adding rows to your DataFrame a go!

As a consequence of what has just been explained, you understand that the general recommendation is that you use .loc to insert rows in your DataFrame.

If you would use df.ix[], you might try to reference a numerically valued index with the index value and accidentally overwrite an existing row of your DataFrame.

You better avoid this!

Adding a Column to Your DataFrame
In some cases, you want to make your index part of your DataFrame. You can easily do this by taking a column from your DataFrame or by referring to a column that you haven't made yet and assigning it to the .index property.

However, if you want to append columns to your DataFrame, you could also follow the same approach as adding an index to your DataFrame: you use loc or iloc.

Note that the observation that was made earlier about loc still stays valid also for when you're adding columns to your DataFrame!

Resetting the Index of Your DataFrame
When your index doesn't look entirely the way you want it to, you can opt to reset it. This can easily ben done with .reset_index().

---

68. How To Select an Index or Column From a Pandas DataFrame?

Before you start with adding, deleting and renaming the components of your DataFrame, you first need to know how you can select these elements.

So, how do you do this?

Well, in essence, selecting an index, column or value from your DataFrame isn't that hard. It's really very similar to what you see in other languages that are used for data analysis (and which you might already know!).

Let's take R for example. You use the [,] notation to access the data frame's values. In Pandas DataFrames, this is not too much different: the most important constructions to use are, without a doubt, loc and iloc. The subtle differences between these two will be discussed in the next sections. For now, it suffices to know that you can either access the values by calling them by their label or by their position in the index or column.

---

69. How will you get the average of values of a column in pandas DataFrame?

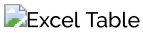
Ans;

Steps to get the Average for each Column and Row in Pandas DataFrame

Step 1: Gather the data

To start, gather the data that needs to be averaged.

For example, I gathered the following data about the commission earned by 3 employees (over the first 6 months of the year):



The goal is to get the average of the commission earned:

- For each employee over the first 6 months (average by column)
- For each month across all employees (average by row)

Step 2: Create the DataFrame

Next, create the DataFrame in order to capture the above data in Python:

```
import pandas as pd
data = {'Month': ['Jan ','Feb ','Mar ','Apr ','May ','Jun '],
'Jon Commission': [7000,5500,6000,4500,8000,6000],
'Maria Commission': [10000,7500,6500,6000,9000,8500],
'Olivia Commission': [3000,6000,4500,4500,4000,5500],
}
df = pd.DataFrame(data,columns=['Month','Jon Commission','Maria Commission','Olivia Commission'])
print (df)
```

Run the code in Python, and you'll get the following DataFrame:

Step 3: Get the Average for each Column and Row in Pandas DataFrame

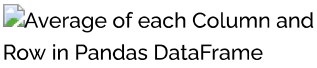
You can then apply the following syntax to get the average for each *column:*

```
df.mean(axis=0)
```

For our example, this is the complete Python code to get the average commission earned for each employee over the 6 first months (average by column):

```
import pandas as pd
data = {'Month': ['Jan ','Feb ','Mar ','Apr ','May ','Jun '],
'Jon Commission': [7000,5500,6000,4500,8000,6000],
'Maria Commission': [10000,7500,6500,6000,9000,8500],
'Olivia Commission': [3000,6000,4500,4500,4000,5500]
}
df = pd.DataFrame(data,columns=['Month','Jon Commission','Maria Commission','Olivia Commission'])
av_column = df.mean(axis=0)
print (av_column)
```

Run the code, and you'll get the average commission per employee:



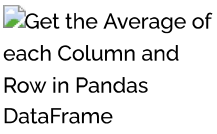
Alternatively, you can get the average for each *row* using the following syntax:

```
df.mean(axis=1)
```

Here is the code that you can use to get the average commission earned for each month across all employees (average by row):

```
import pandas as pd
data = {'Month': ['Jan ','Feb ','Mar ','Apr ','May ','Jun '],
'Jon Commission': [7000,5500,6000,4500,8000,6000],
'Maria Commission': [10000,7500,6500,6000,9000,8500],
'Olivia Commission': [3000,6000,4500,4500,4000,5500],
}
df = pd.DataFrame(data,columns=['Month','Jon Commission','Maria Commission','Olivia Commission'], index
av_row = df.mean(axis=1)
print (av_row)
```

Once you run the code in Python, you'll get the average commission earned per month:



You may also want to check the following source that explains the steps to get the sum for each column and row in pandas DataFrame.

---

70. How to Apply function to every row in a Pandas DataFrame?

Ans: Python is a great language for performing data analysis tasks. It provides with a huge amount of Classes and function which help in analyzing and manipulating data in an easier way.

One can use `apply()` function in order to apply function to every row in given dataframe. Let's see the ways we can do this task.

Example

```
# Import pandas package
import pandas as pd

# Function to add
def add(a, b, c):
    return a + b + c

def main():

    # create a dictionary with
    # three fields each
    data = {
            'A':[1, 2, 3],
            'B':[4, 5, 6],
            'C':[7, 8, 9] }

    # Convert the dictionary into DataFrame
    df = pd.DataFrame(data)
    print("Original DataFrame:\n", df)

    df['add'] = df.apply(lambda row : add(row['A'],
                    row['B'], row['C']), axis = 1)

    print('\nAfter Applying Function: ')
    # printing the new dataframe
    print(df)

if __name__ == '__main__':
    main()
```

Output:

```
Original DataFrame:
    A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

After Applying Function:
   A  B  C  add
0  1  4  7   12
1  2  5  8   15
2  3  6  9   18
```

---

71. What is use of GroupBy objects in Pandas?

Ans:

GroupBy is used to **split the data into groups**. It groups the data based on some criteria. Grouping also provides a mapping of labels to the group names. It has a lot of variations that can be defined with the parameters and makes the task of splitting the data quick and easy.

---

72. What is Pandas NumPy?

Ans: Pandas Numpy is an open-source library developed for Python that is used to work with a large number of datasets. It contains a powerful N-dimensional array object and sophisticated mathematical functions for scientific computing with Python.

Some of the popular functionalities present with Numpy are Fourier transforms, linear algebra, and random number capabilities. It also has tools for integrating with C/C++ and Fortran code.

73. What is Vectorization in Python pandas?

Ans: Vectorization is the process of running operations on the entire array. This is done to reduce the amount of iteration performed by the functions. Pandas have a number of vectorized functions like aggregations, and string functions that are optimized to operate specifically on series and DataFrames. So it is preferred to use the vectorized pandas functions to execute the operations quickly.

74. List some alternatives of Python Pandas?

Ans: Some of the alternatives to the Python Pandas are

- the NumPy,
- R language,
- Anaconda,
- SciPy,
- PySpark,
- Dask,
- Pentaho Data, and Panda.

75. How to convert a DataFrame to an array in Pandas?

Ans: The function to_numpy() is used to **convert the DataFrame** to a **NumPy array**.

```
//syntax
DataFrame.to_numpy(self, dtype=None, copy=False)
```

The dtype parameter defines the data type to pass to the array and the copy ensures the returned value is not a view on another array.

76. List some statistical functions in Python Pandas?

Ans: Some of the statistical functions in Python Pandas are,

sum() – it returns the sum of the values.

mean() – returns the mean that is the average of the values.

std() – returns the standard deviation of the numerical columns.

min() – returns the minimum value.

max() – returns the maximum value.

abs() – returns the absolute value.

prod() – returns the product of the values.

77. Explain Series In pandas. How To Create Copy Of Series In pandas?

Ans: Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index. The basic method to create a Series is to call:

>>> s = pd.Series(data, index=index), where the data can be a Python dict, an ndarray or a scalar value.

To create a copy in pandas, we can call copy() function on a series such that

s2=s1.copy() will create copy of series s1 in a new series s2.

78. What Are The Different Ways A DataFrame Can Be Created In pandas?

Ans:

DataFrame can be created in different ways here are some ways by which we create a DataFrame:

- *Using **List**:*

initialize list of lists

data = [['p', 1], ['q', 2], ['r', 3]]

Create the pandas DataFrame

df = pd.DataFrame(data, columns = ['Letter', 'Number'])

print dataframe.

df

- *Using **dict** of narray/lists:*

To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

- *Using **arrays**:*

DataFrame using arrays.

import pandas as pd

initialise data of lists.

data = {'Name':['Tom', 'Jack', 'nick', 'juli'], 'marks':[99, 98, 95, 90]}

```
# Creates pandas DataFrame.

df = pd.DataFrame(data, index =['rank1', 'rank2', 'rank3', 'rank4'])

# print the data

df
```


Pune:

Bhandarkar Road

Anujit Building, Ground Floor,

Opposite Kamala Nehru Park, Bhandarkar Road,

Erandwane, Pune, Maharashtra

info@kausalvikash.in

+91-735-007-0755

Kharadi

Supreme Arcade, 1st Floor, Office No -5,

Above More Store, Kharadi, Pune, Maharashtra

info@kausalvikash.in

+91-735-007-0755

Mumbai:

103-104, Siddhgiri Building, Next to VIP showroom, Borivali West, Mumbai 91

info@kausalvikash.in

+91-998-733-9060

+91-735-007-0755

Keep In Touch

Noida 4 :

A-93, THE OFFICE PASS(TOP), near 16 metro,

Sector – 4, Noida, Uttar Pradesh – 201301

info.delhi@emergenteck.in

+91-762-082-4981

Noida 65 :

Unboxed Co-working, C 15, C Block Road, C Block,

Sector – 65, Noida, Uttar Pradesh – 201301

info.delhi@emergenteck.in

+91-762-082-4981

Quick Links

- About Us
- RPA Training in Pune | Mumbai | Delhi
- Blue Prism Training Pune | Mumbai
- Blue Prism Training in Delhi | Noida
- Cognitive RPA Training
- RPA & Cognitive for Strategic Management Training
- RPA UiPath Training Course in Noida
- Python Interview Questions and Answers for Freshers
- Python | Django Interview Questions And Answers for Freshers
- Python | Pandas Interview Questions And Answers for Freshers
- Machine Learning Interview Questions And Answers for Freshers