NumPy Introduction

What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

**Data Science:** is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Which Language is NumPy written in?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Where is the NumPy Codebase?

The source code for NumPy is located at this github repository <https://github.com/numpy/numpy>

**github:** enables many people to work on the same codebase.

## Installation of NumPy

If you have [Python](https://www.w3schools.com/python/default.asp) and [PIP](https://www.w3schools.com/python/python_pip.asp) already installed on a system, then installation of NumPy is very easy.

Install it using this command:

C:\Users\Your Name>pip install numpy

If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

## Import NumPy

Once NumPy is installed, import it in your applications by adding the import keyword:

import numpy

Now NumPy is imported and ready to use.

### Example

import numpy  
  
arr = numpy.array([1, 2, 3, 4, 5])  
  
print(arr)

## NumPy as np

NumPy is usually imported under the np alias.

**alias:** In Python alias are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

import numpy as np

Now the NumPy package can be referred to as np instead of numpy.

### Example

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5])  
  
print(arr)

## Checking NumPy Version

The version string is stored under \_\_version\_\_ attribute.

### Example

import numpy as np  
  
print(np.\_\_version\_\_)

## Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

### Example

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5])  
  
print(arr)  
  
print(type(arr))

**type():** This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

### Example

Use a tuple to create a NumPy array:

import numpy as np  
  
arr = np.array((1, 2, 3, 4, 5))  
  
print(arr)

## Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

**nested array:** are arrays that have arrays as their elements.

## 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

### Example

Create a 0-D array with value 42

import numpy as np  
  
arr = np.array(42)  
  
print(arr)

## 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

### Example

Create a 1-D array containing the values 1,2,3,4,5:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5])  
  
print(arr)

## 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

### Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

import numpy as np  
  
arr = np.array([[1, 2, 3], [4, 5, 6]])  
  
print(arr)

## 3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

### Example

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

import numpy as np  
  
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])  
  
print(arr)

## Check Number of Dimensions?

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

### Example

Check how many dimensions the arrays have:

import numpy as np  
  
a = np.array(42)  
b = np.array([1, 2, 3, 4, 5])  
c = np.array([[1, 2, 3], [4, 5, 6]])  
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])  
  
print(a.ndim)  
print(b.ndim)  
print(c.ndim)  
print(d.ndim)

## Higher Dimensional Arrays

An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the ndmin argument.

### Example

Create an array with 5 dimensions and verify that it has 5 dimensions:

import numpy as np  
  
arr = np.array([1, 2, 3, 4], ndmin=5)  
  
print(arr)  
print('number of dimensions :', arr.ndim)

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

# NumPy Array Indexing

## Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

### Example

Get the first element from the following array:

import numpy as np  
  
arr = np.array([1, 2, 3, 4])  
  
print(arr[0])

### Example

Get the second element from the following array.

import numpy as np  
  
arr = np.array([1, 2, 3, 4])  
  
print(arr[1])

### Example

Get third and fourth elements from the following array and add them.

import numpy as np  
  
arr = np.array([1, 2, 3, 4])  
  
print(arr[2] + arr[3])

## Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

### Example

Access the 2nd element on 1st dim:

import numpy as np  
  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
  
print('2nd element on 1st dim: ', arr[0, 1])

### Example

Access the 5th element on 2nd dim:

import numpy as np  
  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
  
print('5th element on 2nd dim: ', arr[1, 4])

## Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

### Example

Access the third element of the second array of the first array:

import numpy as np  
  
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])  
  
print(arr[0, 1, 2])

### Example Explained

arr[0, 1, 2] prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:  
[[1, 2, 3], [4, 5, 6]]  
and:  
[[7, 8, 9], [10, 11, 12]]  
Since we selected 0, we are left with the first array:  
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:  
[1, 2, 3]  
and:  
[4, 5, 6]  
Since we selected 1, we are left with the second array:  
[4, 5, 6]

The third number represents the third dimension, which contains three values:  
4  
5  
6  
Since we selected 2, we end up with the third value:  
6

## Negative Indexing

Use negative indexing to access an array from the end.

### Example

Print the last element from the 2nd dim:

import numpy as np  
  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
  
print('Last element from 2nd dim: ', arr[1, -1])

# Pandas Introduction

## What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

## Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

:}

**Data Science:** is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

## What Can Pandas Do?

Pandas gives you answers about the data. Like:

* Is there a correlation between two or more columns?
* What is average value?
* Max value?
* Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

## Where is the Pandas Codebase?

The source code for Pandas is located at this github repository <https://github.com/pandas-dev/pandas>

## Installation of Pandas

If you have [Python](https://www.w3schools.com/python/default.asp) and [PIP](https://www.w3schools.com/python/pandas/python_pip.asp) already installed on a system, then installation of Pandas is very easy.

Install it using this command:

C:\Users\Your Name>pip install pandas

If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

## Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

import pandas

Now Pandas is imported and ready to use.

### Example

import pandas  
  
mydataset = {  
  'cars': ["BMW", "Volvo", "Ford"],  
  'passings': [3, 7, 2]  
}  
  
myvar = pandas.DataFrame(mydataset)  
  
print(myvar)

## Pandas as pd

Pandas is usually imported under the pd alias.

**alias:** In Python alias are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas.

### Example

import pandas as pd  
  
mydataset = {  
  'cars': ["BMW", "Volvo", "Ford"],  
  'passings': [3, 7, 2]  
}  
  
myvar = pd.DataFrame(mydataset)  
  
print(myvar)

## Checking Pandas Version

The version string is stored under \_\_version\_\_ attribute.

### Example

import pandas as pd  
  
print(pd.\_\_version\_\_)

# Pandas Series

## What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

### Example

Create a simple Pandas Series from a list:

import pandas as pd  
  
a = [1, 7, 2]  
  
myvar = pd.Series(a)  
  
print(myvar)

## Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

### Example

Return the first value of the Series:

print(myvar[0])

## Create Labels

With the index argument, you can name your own labels.

### Example

Create you own labels:

import pandas as pd  
  
a = [1, 7, 2]  
  
myvar = pd.Series(a, index = ["x", "y", "z"])  
  
print(myvar)

When you have created labels, you can access an item by referring to the label.

### Example

Return the value of "y":

print(myvar["y"])

## Key/Value Objects as Series

You can also use a key/value object, like a dictionary, when creating a Series.

### Example

Create a simple Pandas Series from a dictionary:

import pandas as pd  
  
calories = {"day1": 420, "day2": 380, "day3": 390}  
  
myvar = pd.Series(calories)  
  
print(myvar)

**Note:** The keys of the dictionary become the labels.

To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

### Example

Create a Series using only data from "day1" and "day2":

import pandas as pd  
  
calories = {"day1": 420, "day2": 380, "day3": 390}  
  
myvar = pd.Series(calories, index = ["day1", "day2"])  
  
print(myvar)

## DataFrames

Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

### Example

Create a DataFrame from two Series:

import pandas as pd  
  
data = {  
  "calories": [420, 380, 390],  
  "duration": [50, 40, 45]  
}  
  
myvar = pd.DataFrame(data)  
  
print(myvar)

## What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

### Example

Create a simple Pandas DataFrame:

import pandas as pd  
  
data = {  
  "calories": [420, 380, 390],  
  "duration": [50, 40, 45]  
}  
  
#load data into a DataFrame object:  
df = pd.DataFrame(data)  
  
print(df)

## Result

calories duration

0 420 50

1 380 40

2 390 45

## Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

### Example

Return row 0:

#refer to the row index:  
print(df.loc[0])

## Result

calories 420

duration 50

Name: 0, dtype: int64

**Note:** This example returns a Pandas **Series**.

### Example

Return row 0 and 1:

#use a list of indexes:  
print(df.loc[[0, 1]])

## Result

calories duration

0 420 50

1 380 40

**Note:** When using [], the result is a Pandas **DataFrame**.

## Named Indexes

With the index argument, you can name your own indexes.

### Example

Add a list of names to give each row a name:

import pandas as pd  
  
data = {  
  "calories": [420, 380, 390],  
  "duration": [50, 40, 45]  
}  
  
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])  
  
print(df)

## Result

calories duration

day1 420 50

day2 380 40

day3 390 45

## Locate Named Indexes

Use the named index in the loc attribute to return the specified row(s).

### Example

Return "day2":

#refer to the named index:  
print(df.loc["day2"])

## Result

calories 380

duration 40

Name: 0, dtype: int64

## Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame.

### Example

Load a comma separated file (CSV file) into a DataFrame:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df)

You will learn more about importing files in the next chapters.

## Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.

[Download data.csv](https://www.w3schools.com/python/pandas/data.csv). or [Open data.csv](https://www.w3schools.com/python/pandas/data.csv.txt)

### Example

Load the CSV into a DataFrame:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df.to\_string())

**Tip:** use to\_string() to print the entire DataFrame.

By default, when you print a DataFrame, you will only get the first 5 rows, and the last 5 rows:

### Example

Print a reduced sample:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df)

# Pandas - Analyzing DataFrames

## Viewing the Data

One of the most used method for getting a quick overview of the DataFrame, is the head() method.

The head() method returns the headers and a specified number of rows, starting from the top.

### Example

Get a quick overview by printing the first 10 rows of the DataFrame:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df.head(10))

In our examples we will be using a CSV file called 'data.csv'.

Download [data.csv](https://www.w3schools.com/python/pandas/data.csv), or open [data.csv](https://www.w3schools.com/python/pandas/data.csv.txt) in your browser.

**Note:** if the number of rows is not specified, the head() method will return the top 5 rows.

### Example

Print the first 5 rows of the DataFrame:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df.head())

There is also a tail() method for viewing the last rows of the DataFrame.

The tail() method returns the headers and a specified number of rows, starting from the bottom.

### Example

Print the last 5 rows of the DataFrame:

print(df.tail())

## Info About the Data

The DataFrames object has a method called info(), that gives you more information about the data set.

### Example

Print information about the data:

print(df.info())

## Result

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 169 entries, 0 to 168

Data columns (total 4 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Duration 169 non-null int64

1 Pulse 169 non-null int64

2 Maxpulse 169 non-null int64

3 Calories 164 non-null float64

dtypes: float64(1), int64(3)

memory usage: 5.4 KB

None

## Result Explained

The result tells us there are 169 rows and 4 columns:

RangeIndex: 169 entries, 0 to 168

Data columns (total 4 columns):

And the name of each column, with the data type:

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Duration 169 non-null int64

1 Pulse 169 non-null int64

2 Maxpulse 169 non-null int64

3 Calories 164 non-null float64

## Null Values

The info() method also tells us how many Non-Null values there are present in each column, and in our data set it seems like there are 164 of 169 Non-Null values in the "Calories" column.

Which means that there are 5 rows with no value at all, in the "Calories" column, for whatever reason.

Empty values, or Null values, can be bad when analyzing data, and you should consider removing rows with empty values. This is a step towards what is called cleaning data, and you will learn more about that in the next chapters.

Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

* Empty cells
* Data in wrong format
* Wrong data
* Duplicates

In this tutorial you will learn how to deal with all of them.

Our Data Set

In the next chapters we will use this data set:

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaN 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 2020/12/26 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).

The data set contains wrong format ("Date" in row 26).

The data set contains wrong data ("Duration" in row 7).

The data set contains duplicates (row 11 and 12).

# Pandas - Cleaning Empty Cells

## Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

## Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

### Example

Return a new Data Frame with no empty cells:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
new\_df = df.dropna()  
  
print(new\_df.to\_string())

In our cleaning examples we will be using a CSV file called 'dirtydata.csv'.

[Download dirtydata.csv](https://www.w3schools.com/python/pandas/dirtydata.csv). or [Open dirtydata.csv](https://www.w3schools.com/python/pandas/dirtydata.csv.txt)

**Note:** By default, the dropna() method returns a new DataFrame, and will not change the original.

If you want to change the original DataFrame, use the inplace = True argument:

### Example

Remove all rows with NULL values:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
df.dropna(inplace = True)  
  
print(df.to\_string())

**Note:** Now, the dropna(inplace = True) will NOT return a new DataFrame, but it will remove all rows containg NULL values from the original DataFrame.

## Replace Empty Values

Another way of dealing with empty cells is to insert a new value instead.

This way you do not have to delete entire rows just because of some empty cells.

The fillna() method allows us to replace empty cells with a value:

### Example

Replace NULL values with the number 130:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
df.fillna(130, inplace = True)

### Replace Only For a Specified Columns

The example above replaces all empty cells in the whole Data Frame.

To only replace empty values for one column, specify the column name for the DataFrame:

### Example

Replace NULL values in the "Calories" columns with the number 130:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
df["Calories"].fillna(130, inplace = True)

## Replace Using Mean, Median, or Mode

A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

Pandas uses the mean() median() and mode() methods to calculate the respective values for a specified column:

### Example

Calculate the MEAN, and replace any empty values with it:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
x = df["Calories"].mean()  
  
df["Calories"].fillna(x, inplace = True)

**Mean** = the average value (the sum of all values divided by number of values).

### Example

Calculate the MEDIAN, and replace any empty values with it:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
x = df["Calories"].median()  
  
df["Calories"].fillna(x, inplace = True)

**Median** = the value in the middle, after you have sorted all values ascending.

### Example

Calculate the MODE, and replace any empty values with it:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
x = df["Calories"].mode()[0]  
  
df["Calories"].fillna(x, inplace = True)

**Mode** = the value that appears most frequently.

# Pandas - Cleaning Data of Wrong Format

## Data of Wrong Format

Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

## Convert Into a Correct Format

In our Data Frame, we have two cells with the wrong format. Check out row 22 and 26, the 'Date' column should be a string that represents a date:

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaN 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 20201226 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

Let's try to convert all cells in the 'Date' column into dates.

Pandas has a to\_datetime() method for this:

### Example

Convert to date:

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
df['Date'] = pd.to\_datetime(df['Date'])  
  
print(df.to\_string())

Result:

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaT 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 '2020/12/26' 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

As you can see from the result, the date in row 26 where fixed, but the empty date in row 22 got a NaT (Not a Time) value, in other words an empty value. One way to deal with empty values is simply removing the entire row.

## Removing Rows

The result from the converting in the example above gave us a NaT value, which can be handled as a NULL value, and we can remove the row by using the dropna() method.

### Example

Remove rows with a NULL value in the "Date" column:

df.dropna(subset=['Date'], inplace = True)

# Pandas - Fixing Wrong Data

## Wrong Data

"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaN 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 20201226 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

How can we fix wrong values, like the one for "Duration" in row 7?

## Replacing Values

One way to fix wrong values is to replace them with something else.

In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:

### Example

Set "Duration" = 45 in row 7:

df.loc[7, 'Duration'] = 45

For small data sets you might be able to replace the wrong data one by one, but not for big data sets.

To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

### Example

Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

for x in df.index:  
  if df.loc[x, "Duration"] > 120:  
    df.loc[x, "Duration"] = 120

## Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.

### Example

Delete rows where "Duration" is higher than 120:

for x in df.index:  
  if df.loc[x, "Duration"] > 120:  
    df.drop(x, inplace = True)

# Pandas - Removing Duplicates

## Discovering Duplicates

Duplicate rows are rows that have been registered more than one time.

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaN 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 20201226 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

By taking a look at our test data set, we can assume that row 11 and 12 are duplicates.

To discover duplicates, we can use the duplicated() method.

The duplicated() method returns a Boolean values for each row:

### Example

Returns True for every row that is a duplicate, othwerwise False:

print(df.duplicated())

## Removing Duplicates

To remove duplicates, use the drop\_duplicates() method.

### Example

Remove all duplicates:

df.drop\_duplicates(inplace = True)

Python | Pandas Dataframe.rank()

* Last Updated : 17 Sep, 2018

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.rank()** method returns a rank of every respective index of a series passed. The rank is returned on the basis of position after sorting.

***Syntax:*** *DataFrame.rank(axis=0, method=’average’, numeric\_only=None, na\_option=’keep’, ascending=True, pct=False)*

***Parameters:******axis:****0 or ‘index’ for rows and 1 or ‘columns’ for Column.****method:****Takes a string input(‘average’, ‘min’, ‘max’, ‘first’, ‘dense’) which tells pandas what to do with same values. Default is average which means assign average of ranks to the similar values.****numeric\_only:****Takes a boolean value and the rank function works on non-numeric value only if it’s False.****na\_option:****Takes 3 string input(‘keep’, ‘top’, ‘bottom’) to set position of Null values if any in the passed Series.****ascending:****Boolean value which ranks in ascending order if True.****pct:****Boolean value which ranks percentage wise if True.*

***Return type:****Series with Rank of every index of caller series.*

For link to CSV file Used in Code, click [here.](https://media.geeksforgeeks.org/wp-content/uploads/nba.csv)

**Example #1:** Ranking Column with Unique values

In the following example, a new rank column is created which ranks the Name of every Player. All the values in Name column are unique and hence there is no need to describe a method.

# importing pandas package

importpandas as pd

# making data frame from csv file

data =pd.read\_csv("nba.csv")

# creating a rank column and passing the returned rank series

data["Rank"] =data["Name"].rank()

# display

data

# sorting w.r.t name column

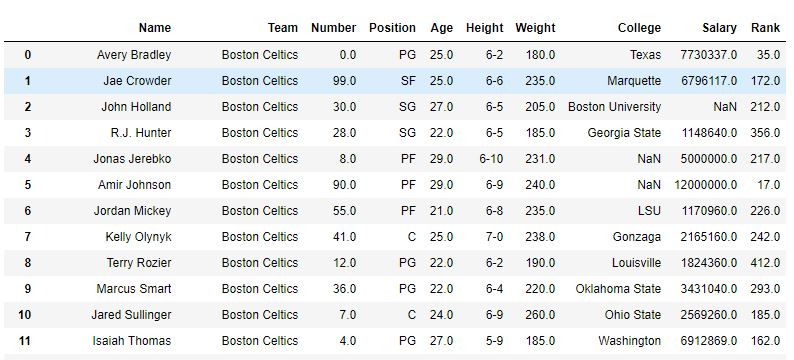
data.sort\_values("Name", inplace =True)

# display after sorting w.r.t Name column

data

**Output:**  
As shown in the image, a column rank was created with rank of every Name. After the sort\_value function sorted the data frame with respect to name, it can be seen that the rank was also sorted since those were ranking of Names only.

**Before Sorting –**



**After Sorting –**



**Example #2:** Sorting Column with some similar values

In the following example, data frame is first sorted with respect to *team name* and first the method is default (i.e. average) and hence the rank of same Team players is average. After that min method is also used to see the output.

# importing pandas package

importpandas as pd

# making data frame from csv file

data =pd.read\_csv("nba.csv")

# sorting w.r.t team name

data.sort\_values("Team", inplace =True)

# creating a rank column and passing the returned rank series

# change method to 'min' to rank by minimum

data["Rank"] =data["Team"].rank(method ='average')

# display

data

**Output:**

***With method=’average’***



***With method=’min’***



Align columns to Left in Pandas – Python

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 03 Jan, 2021

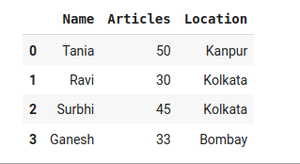
Pandas library is useful for performing exploratory data analysis in Python. A pandas dataframe represents data in a tabular format. We can perform operations on the data and display it. In this article, we are going to align columns to the Left in Pandas. When we display the dataframe, we can align the data in the columns as left, right, or center.

**The default is right alignment as we can see in the below example.**

## Python3

|  |
| --- |
| # Python code demonstrate creating  # DataFrame from dict and left aligning  importpandas as pd    # intialise data of lists.  data ={'Name': ['Tania', 'Ravi',                    'Surbhi', 'Ganesh'],            'Articles': [50, 30, 45, 33],            'Location': ['Kanpur', 'Kolkata',                        'Kolkata', 'Bombay']}    # Create DataFrame  df =pd.DataFrame(data)  display(df) |

**Output:**



In order to align columns to left in pandas dataframe, we use the **dataframe.style.set\_properties()**function.

***Syntax:****Styler.set\_properties(subset=None, \*\*kwargs)*

***Parameters:***

* ***subsetIndexSlice:****A valid slice for data to limit the style application to.*
* ***\*\*kwargsdict:****A dictionary of property, value pairs to be set for each cell.*

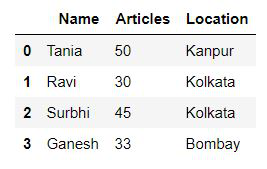
***Returns:****selfStyler*

**Example 1:**

**Python3**

|  |
| --- |
| # Python code demonstrate creating  # DataFrame from dict and left aligning  importpandas as pd    # intialise data of lists.  data ={'Name': ['Tania', 'Ravi',                    'Surbhi', 'Ganesh'],            'Articles': [50, 30, 45, 33],            'Location': ['Kanpur', 'Kolkata',                        'Kolkata', 'Bombay']}    # Create DataFrame  df =pd.DataFrame(data)    left\_aligned\_df =df.style.set\_properties(\*\*{'text-align': 'left'})  display(left\_aligned\_df) |

**Output:**



How to Sort Pandas DataFrame?

* Last Updated : 28 Jul, 2020

In this article, we will discuss how to sort Pandas Dataframe. Let’s create a dataframe.

**Example :**

* Python3

|  |
| --- |
| # importing pandas library  importpandas as pd    # creating and initializing a nested list  age\_list =[['Afghanistan', 1952, 8425333, 'Asia'],              ['Australia', 1957, 9712569, 'Oceania'],              ['Brazil', 1962, 76039390, 'Americas'],              ['China', 1957, 637408000, 'Asia'],              ['France', 1957, 44310863, 'Europe'],              ['India', 1952, 3.72e+08, 'Asia'],              ['United States', 1957, 171984000, 'Americas']]    # creating a pandas dataframe  df =pd.DataFrame(age\_list, columns=['Country', 'Year',                                       'Population', 'Continent'])    df |

**Output :**

## ****Sorting Pandas Data Frame****

In order to sort the data frame in pandas, function [sort\_values()](https://www.geeksforgeeks.org/python-pandas-dataframe-sort_values-set-1/) is used. Pandas sort\_values() can sort the data frame in Ascending or Descending order.

**Example 1:**Sorting the Data frame in Ascending order

* Python3

# importing pandas library

importpandas as pd

# creating and initializing a nested list

age\_list =[['Afghanistan',1952,8425333,'Asia'],

            ['Australia',1957,9712569,'Oceania'],

            ['Brazil',1962,76039390,'Americas'],

            ['China',1957,637408000,'Asia'],

            ['France',1957,44310863,'Europe'],

            ['India',1952,3.72e+08,'Asia'],

            ['United States',1957,171984000,'Americas']]

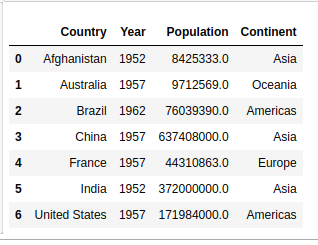
# creating a pandas dataframe

df =pd.DataFrame(age\_list,columns=['Country','Year',

                                    'Population','Continent'])

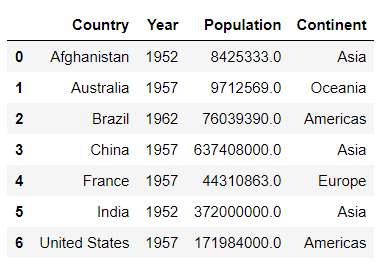
# Sorting by column 'Country'

df.sort\_values(by=['Country'])



**Example 2:**Sorting the Data frame in Descending order

* Python3
* # Sorting Pandas Dataframe in Descending Order
* # importing pandas library
* importpandas as pd
* # Initializing the nested list with Data set
* age\_list =[['Afghanistan', 1952, 8425333, 'Asia'],
* ['Australia', 1957, 9712569, 'Oceania'],
* ['Brazil', 1962, 76039390, 'Americas'],
* ['China', 1957, 637408000, 'Asia'],
* ['France', 1957, 44310863, 'Europe'],
* ['India', 1952, 3.72e+08, 'Asia'],
* ['United States', 1957, 171984000, 'Americas']]
* # creating a pandas dataframe
* df =pd.DataFrame(age\_list, columns=['Country', 'Year',
* 'Population', 'Continent'])
* # Sorting by column "Population"
* df.sort\_values(by=['Population'], ascending=False)



## What is Matplotlib?

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

## Where is the Matplotlib Codebase?

The source code for Matplotlib is located at this github repository <https://github.com/matplotlib/matplotlib>

## Installation of Matplotlib

If you have [Python](https://www.w3schools.com/python/default.asp) and [PIP](https://www.w3schools.com/python/python_pip.asp) already installed on a system, then installation of Matplotlib is very easy.

Install it using this command:

C:\Users\Your Name>pip install matplotlib

If this command fails, then use a python distribution that already has Matplotlib installed,  like Anaconda, Spyder etc.

## Import Matplotlib

Once Matplotlib is installed, import it in your applications by adding the import *module* statement:

import matplotlib

Now Matplotlib is imported and ready to use:

## Checking Matplotlib Version

The version string is stored under \_\_version\_\_ attribute.

### Example

import matplotlib  
  
print(matplotlib.\_\_version\_\_)

**Note:** two underscore characters are used in \_\_version\_\_.

## Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

import matplotlib.pyplot as plt

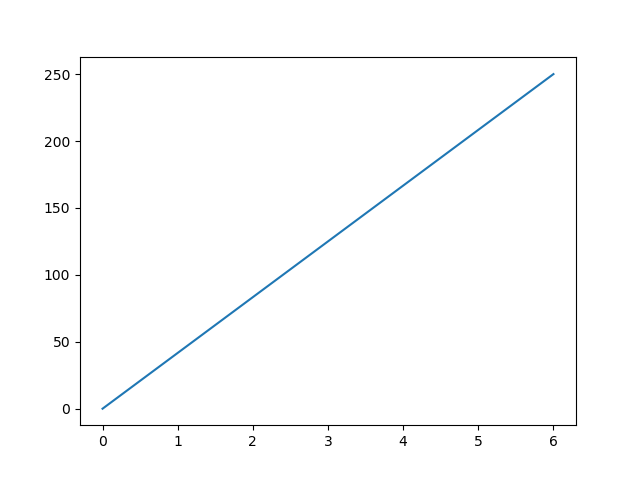
Now the Pyplot package can be referred to as plt.

### Example

Draw a line in a diagram from position (0,0) to position (6,250):

import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([0, 6])  
ypoints = np.array([0, 250])  
  
plt.plot(xpoints, ypoints)  
plt.show()

### Result:



# Matplotlib Plotting

## Plotting x and y points

The plot() function is used to draw points (markers) in a diagram.

By default, the plot() function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the **x-axis**.

Parameter 2 is an array containing the points on the **y-axis**.

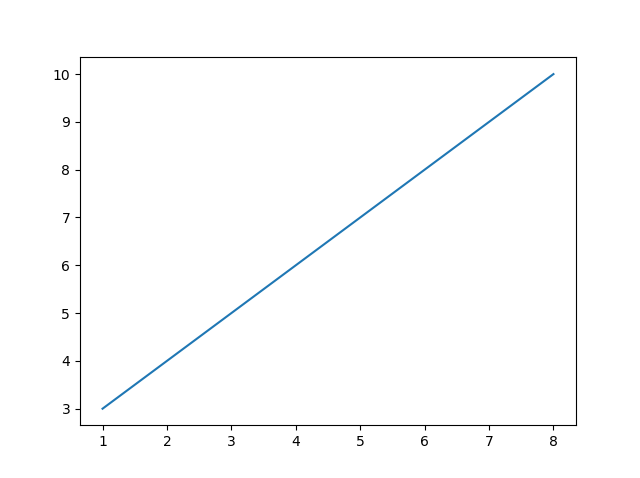
If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

### Example

Draw a line in a diagram from position (1, 3) to position (8, 10):

import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([1, 8])  
ypoints = np.array([3, 10])  
  
plt.plot(xpoints, ypoints)  
plt.show()

### Result:



The **x-axis** is the horizontal axis.

The **y-axis** is the vertical axis.

## Plotting Without Line

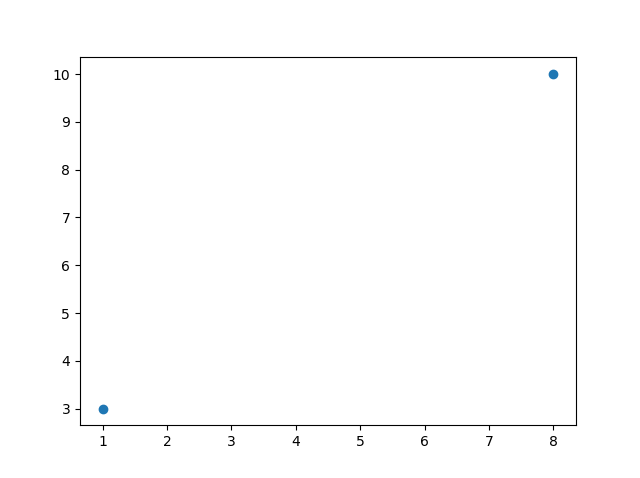
To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'.

### Example

Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([1, 8])  
ypoints = np.array([3, 10])  
  
plt.plot(xpoints, ypoints, 'o')  
plt.show()

### Result:



## Multiple Points

You can plot as many points as you like, just make sure you have the same number of points in both axis.

### Example

Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([1, 2, 6, 8])  
ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(xpoints, ypoints)  
plt.show()

### Result:

