Numpy Tutorial Part 2 – Vital Functions for Data Analysis

Numpy is the core package for data analysis and scientific computing in python. This is part 2 of a mega numpy tutorial. In this part, I go into the details of the advanced features of numpy that are essential for data analysis and manipulations.

Numpy Tutorial Part 2: Vital Functions for Data Analysis. Photo by Ana Philipa Neves.

Contents

Introduction

In part 1 of the numpy tutorial [https://www.machinelearningplus.com/numpy-tutorial-part1-array-python-examples/] we got introduced to numpy and why its so important to know numpy if you are to work with datasets in python. In particular, we discussed how to create arrays, explore it, indexing, reshaping, flattening, generating random numbers and many other functions.

In part 2 (this tutorial), I continue from where we left and take it up a notch by dealing with slightly more advanced but essential topics for data analysis.

I will assume that you have some familiarity with python, know basic math and have already read the part 1 of the numpy tutorial.

The best way to approach this post is to read the whole article fairly quick in one go and then come back to the beginning and try out the examples in a jupyter notebook.

Let's begin.

1. How to get index locations that satisfy a given condition using np.where?

Previously you saw how to extract items from an array that satisfy a given condition. Boolean indexing, remember?

But sometimes we want to know the index positions of the items (that satisfy a condition) and do whatever you want with it.

`np.where` locates the positions in the array where a given condition holds true.

```
# Create an array
import numpy as np
arr_rand = np.array([8, 8, 3, 7, 7, 0, 4, 2, 5, 2])
print("Array: ", arr_rand)

# Positions where value > 5
index_gt5 = np.where(arr_rand > 5)
print("Positions where value > 5: ", index_gt5)
```

```
#> Array:  [8 8 3 7 7 0 4 2 5 2]
#> Positions where value > 5:  (array([0, 1, 3, 4]),)
```

Once you have the positions, you can extract them using the array's `take` method.

```
# Take items at given index
arr_rand.take(index_gt5)
```

```
#> array([[8, 8, 7, 7]])
```

Thankfully, `np.where` also accepts 2 more optional arguments x and y. Whenever condition is true, 'x' is yielded else 'y'.

Below, I try to create an array that will have the string 'gt5' whenever the condition is true, else, it will have 'lt5'.

```
# If value > 5, then yield 'gt5' else 'le5'
np.where(arr_rand > 5, 'gt5', 'le5')
```

```
#> array(['gt5', 'gt5', 'le5', 'gt5', 'gt5', 'le5', 'le5', 'le5', 'le5', 'le5'],
      dtype='<U3')
```

Let's find the location of the maximum and minimum values as well.

```
# Location of the max
print('Position of max value: ', np.argmax(arr_rand))

# Location of the min
print('Position of min value: ', np.argmin(arr_rand))
```

```
#> Position of max value:  0
#> Position of min value:  5
```

Good.

2. How to import and export data as a csv file?

A standard way to import datasets is to use the `np.genfromtxt` function. It can import datasets from web URLs, handle missing values, multiple delimiters, handle irregular number of columns etc.

A less versatile version is the `np.loadtxt` which assumes the dataset has no missing values.

As an example, let's try to read a .csv file [http://'https://raw.githubusercontent.com/selva86/datasets/master/Auto.csv'] from the below URL. Since all elements in a numpy array should be of the same data type, the last column which is a text will be imported as a 'nan' by default.

By setting the `filling_values` argument you can replace the missing values with something else.

```
# Turn off scientific notation
np.set_printoptions(suppress=True)

# Import data from csv file url
path = 'https://raw.githubusercontent.com/selva86/datasets/master/Auto.csv'
data = np.genfromtxt(path, delimiter=',', skip_header=1, filling_values=-999, dtype='f
loat')
data[:3]  # see first 3 rows
```

```
#> array([[   18. ,     8. ,   307. ,   130. ,  3504. ,    12. ,    70. ,
#>             1. ,  -999. ],
#>        [   15. ,     8. ,   350. ,   165. ,  3693. ,    11.5,    70. ,
#>             1. ,  -999. ],
#>        [   18. ,     8. ,   318. ,   150. ,  3436. ,    11. ,    70. ,
#>             1. ,  -999. ]])
```

That was neat. But did you notice all the values in last column has the same value '-999'?

That happened because, I had mentioned the. `dtype='float'`. The last column in the file contained text values and since all the values in a numpy array has to be of the same `dtype`, `np.genfromtxt` didn't know how to convert it to a float.

2.1 How to handle datasets that has both numbers and text columns?

In case, you MUST have the text column as it is without replacing it with a placeholder, you can either set the dtype as 'object' or as None.

```
# data2 = np.genfromtxt(path, delimiter=',', skip_header=1, dtype='object')
data2 = np.genfromtxt(path, delimiter=',', skip_header=1, dtype=None)
data2[:3]  # see first 3 rows
```

```
#> array([( 18., 8,  307., 130, 3504,  12. , 70, 1, b'"chevrolet chevelle malibu"'),
#>        ( 15., 8,  350., 165, 3693,  11.5, 70, 1, b'"buick skylark 320"'),
#>        ( 18., 8,  318., 150, 3436,  11. , 70, 1, b'"plymouth satellite"')],
#>       dtype=[('f0', '<f8'), ('f1', '<i8'), ('f2', '<f8'), ('f3', '<i8'), ('f4', '<i
8'), ('f5', '<f8'), ('f6', '<i8'), ('f7', '<i8'), ('f8', 'S38')])
```

Excellent!

Finally, 'np.savetxt' lets you export the array as a csv file.

```
# Save the array as a csv file
np.savetxt("out.csv", data, delimiter=",")
```

3. How to save and load numpy objects?

At some point, we will want to save large transformed numpy arrays to disk and load it back to console directly without having the re-run the data transformations code.

Numpy provides the .npy and the .npz file types for this purpose.

If you want to store a single `ndarray` object, store it as a `.npy` file using `np.save`. This can be loaded back using the `np.load`.

If you want to store more than 1 `ndarray` object in a single file, then save it as a `.npz` file using `np.savez`.

```
# Save single numpy array object as .npy file
np.save('myarray.npy', arr2d)

# Save multile numy arrays as a .npz file
np.savez('array.npz', arr2d_f, arr2d_b)
```

Load back the .npy file.

```
# Load a .npy file
a = np.load('myarray.npy')
print(a)
```

```
#> [[0 1 2]
#>  [3 4 5]
#>  [6 7 8]]
```

Load back the .npz file.

```
# Load a .npz file
b = np.load('array.npz')
print(b.files)
b['arr_0']
```

```
#> ['arr_0', 'arr_1']

#> array([[ 0.,  1.,  2.],
#>        [ 3.,  4.,  5.],
#>        [ 6.,  7.,  8.]])
```

Feedback

4. How to concatenate two numpy arrays columnwise and row wise

There are 3 different ways of concatenating two or more numpy arrays.

- Method 1: `np.concatenate` by changing the axis parameter to 0 and 1
- Method 2: `np.vstack` and `np.hstack`
- Method 3: `np.r_` and `np.c_`

All three methods provide the same output.

One key difference to notice is unlike the other 2 methods, both `np.r_` and `np.c_` use square brackets to stack arrays. But first, let me create the arrays to be concatenated.

```
a = np.zeros([4, 4])
b = np.ones([4, 4])
print(a)
print(b)
```

```
#> [[ 0.  0.  0.  0.]
#>  [ 0.  0.  0.  0.]
#>  [ 0.  0.  0.  0.]
#>  [ 0.  0.  0.  0.]]

#> [[ 1.  1.  1.  1.]
#>  [ 1.  1.  1.  1.]
#>  [ 1.  1.  1.  1.]
#>  [ 1.  1.  1.  1.]]
```

Let's stack the arrays vertically.

```
# Vertical Stack Equivalents (Row wise)
np.concatenate([a, b], axis=0)
np.vstack([a,b])
np.r_[a,b]
```

```
#> array([[ 0.,  0.,  0.,  0.],
#>        [ 0.,  0.,  0.,  0.],
#>        [ 0.,  0.,  0.,  0.],
#>        [ 0.,  0.,  0.,  0.],
#>        [ 1.,  1.,  1.,  1.],
#>        [ 1.,  1.,  1.,  1.],
#>        [ 1.,  1.,  1.,  1.],
#>        [ 1.,  1.,  1.,  1.]])
```

That was what we wanted. Let's do it horizontally (columns wise) as well.

```
# Horizontal Stack Equivalents (Coliumn wise)
np.concatenate([a, b], axis=1)
np.hstack([a,b])
np.c_[a,b]
```

```
#> array([[ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.],
#>        [ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.],
#>        [ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.],
#>        [ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.]])
```

Besides, you can use `np.r_` to create more complex number sequences in 1d arrays.

```
np.r_[[1,2,3], 0, 0, [4,5,6]]
```

```
#> array([1, 2, 3, 0, 0, 4, 5, 6])
```

5. How to sort a numpy array based on one or more columns?

Let's try and sort a 2d array based on the first column.

```
arr = np.random.randint(1,6, size=[8, 4])
arr
```

```
#> array([[3, 3, 2, 1],
#>        [1, 5, 4, 5],
#>        [3, 1, 4, 2],
#>        [3, 4, 5, 5],
#>        [2, 4, 5, 5],
#>        [4, 4, 4, 2],
#>        [2, 4, 1, 3],
#>        [2, 2, 4, 3]])
```

We have a random array of 8 rows and 4 columns.

If you use the `np.sort` function with `axis=0`, all the columns will be sorted in ascending order independent of eachother, effectively compromising the integrity of the row items. In simple terms, the values in each row gets corrupted with values from other rows.

```
# Sort each columns of arr
np.sort(arr, axis=0)
```

```
#> array([[1, 1, 1, 1],
#>        [2, 2, 2, 2],
#>        [2, 3, 4, 2],
#>        [2, 4, 4, 3],
#>        [3, 4, 4, 3],
#>        [3, 4, 4, 5],
#>        [3, 4, 5, 5],
#>        [4, 5, 5, 5]])
```

Since I don't want the content of rows to be disturbed, I resort to an indirect method using `np.argsort`.

5.1 How to sort a numpy array based on 1 column using argsort?

Let's first understand what `np.argsort` does.

`np.argsort` returns the index positions of that would make a given 1d array sorted.

```
# Get the index positions that would sort the array
x = np.array([1, 10, 5, 2, 8, 9])
sort_index = np.argsort(x)
print(sort_index)
```

```
#> [0 3 2 4 5 1]
```

How to interpret this?

In array ‘x’, the 0th item is the smallest, 3rd item is the second smallest and so on.

```
x[sort_index]
```

```
#> array([ 1,  2,  5,  8,  9, 10])
```

Now, in order to sort the original `arr`, I am going to do an argsort on the 1st column and use the resulting index positions to sort `arr`. See the code.

```
# Argsort the first column
sorted_index_1stcol = arr[:, 0].argsort()

# Sort 'arr' by first column without disturbing the integrity of rows
arr[sorted_index_1stcol]
```

```
#> array([[1, 5, 4, 5],
#>        [2, 4, 5, 5],
#>        [2, 4, 1, 3],
#>        [2, 2, 4, 3],
#>        [3, 3, 2, 1],
#>        [3, 1, 4, 2],
#>        [3, 4, 5, 5],
#>        [4, 4, 4, 2]])
```

To sort it in decreasing order, simply reverse the argsorted index.

```
# Descending sort
arr[sorted_index_1stcol[::-1]]
```

```
#> array([[4, 4, 4, 2],
#>        [3, 4, 5, 5],
#>        [3, 1, 4, 2],
#>        [3, 3, 2, 1],
#>        [2, 2, 4, 3],
#>        [2, 4, 1, 3],
#>        [2, 4, 5, 5],
#>        [1, 5, 4, 5]])
```

5.2 How to sort a numpy array based on 2 or more columns?

You can do this using `np.lexsort` by passing a tuple of columns based on which the array should be sorted.

Just remember to place the column to be sorted first at the rightmost side inside the tuple.

```
# Sort by column 0, then by column 1
lexsorted_index = np.lexsort((arr[:, 1], arr[:, 0]))
arr[lexsorted_index]
```

```
#> array([[1, 5, 4, 5],
#>        [2, 2, 4, 3],
#>        [2, 4, 5, 5],
#>        [2, 4, 1, 3],
#>        [3, 1, 4, 2],
#>        [3, 3, 2, 1],
#>        [3, 4, 5, 5],
#>        [4, 4, 4, 2]])
```

6. Working with dates

Numpy implements dates through the `np.datetime64` object which supports a precision till nanoseconds. You can create one using a standard YYYY-MM-DD formatted date strings.

```
# Create a datetime64 object
date64 = np.datetime64('2018-02-04 23:10:10')
date64
```

```
#> numpy.datetime64('2018-02-04T23:10:10')
```

Ofcourse you can pass hours, minutes, seconds till nanoseconds as well.

Let's remove the time component from `date64`.

```
# Drop the time part from the datetime64 object
dt64 = np.datetime64(date64, 'D')
dt64
```

```
#> numpy.datetime64('2018-02-04')
```

By default, if you add a number increases the number of days. But if you need to increase any other time unit like months, hours, seconds etc, then the `timedelta` object is much convenient.

```
# Create the timedeltas (individual units of time)
tenminutes = np.timedelta64(10, 'm')  # 10 minutes
tenseconds = np.timedelta64(10, 's')  # 10 seconds
tennanoseconds = np.timedelta64(10, 'ns')  # 10 nanoseconds

print('Add 10 days: ', dt64 + 10)
print('Add 10 minutes: ', dt64 + tenminutes)
print('Add 10 seconds: ', dt64 + tenseconds)
print('Add 10 nanoseconds: ', dt64 + tennanoseconds)
```

```
#> Add 10 days:  2018-02-14
#> Add 10 minutes:  2018-02-04T00:10
#> Add 10 seconds:  2018-02-04T00:00:10
#> Add 10 nanoseconds:  2018-02-04T00:00:00.000000010
```

Let me convert the `dt64` back to a string.

```
# Convert np.datetime64 back to a string
np.datetime_as_string(dt64)
```

```
#> '2018-02-04'
```

When working with dates, you would often need to filter out the business days from the data. You can know if a given date is a business day or not using the `np.is_busday()`.

```
print('Date: ', dt64)
print("Is it a business day?: ", np.is_busday(dt64))
print("Add 2 business days, rolling forward to nearest biz day: ", np.busday_offset(dt64, 2, roll='forward'))
print("Add 2 business days, rolling backward to nearest biz day: ", np.busday_offset(dt64, 2, roll='backward'))
```

```
#> Date:  2018-02-04
#> Is it a business day?:  False
#> Add 2 business days, rolling forward to nearest biz day:  2018-02-07
#> Add 2 business days, rolling backward to nearest biz day:  2018-02-06
```

6.1 How to create a sequence of dates?

It can simply be done using the `np.arange` itself.

```
# Create date sequence
dates = np.arange(np.datetime64('2018-02-01'), np.datetime64('2018-02-10'))
print(dates)

# Check if its a business day
np.is_busday(dates)
```

```
#> ['2018-02-01' '2018-02-02' '2018-02-03' '2018-02-04' '2018-02-05'
#>  '2018-02-06' '2018-02-07' '2018-02-08' '2018-02-09']

array([ True,  True, False, False,  True,  True,  True,  True,  True], dtype=bool)
```

6.2 How to convert numpy.datetime64 to datetime.datetime object?

```
# Convert np.datetime64 to datetime.datetime
import datetime
dt = dt64.tolist()
dt
```

```
#> datetime.date(2018, 2, 4)
```

Once you convert it to a `datetime.date` object, you have a lot more facilities to extract the day of month, month of year etc.

```
print('Year: ', dt.year)
print('Day of month: ', dt.day)
print('Month of year: ', dt.month)
print('Day of Week: ', dt.weekday())  # Sunday
```

```
#> Year:  2018
#> Day of month:  4
#> Month of year:  2
#> Day of Week:  6
```

7. Advanced numpy functions

7.1 vectorize – Make a scalar function work on vectors

With the help of `vectorize()` you can make a function that is meant to work on individual numbers, to work on arrays.

Let's see a simplified example.

The function `foo` (see code below) accepts a number and squares it if it is 'odd' else it divides it by 2.

When you apply this function on a scalar (individual numbers) it works perfectly, but fails when applied on an array.

With numpy's `vectorize()`, you can magically make it work on arrays as well.

```
# Define a scalar function
def foo(x):
    if x % 2 == 1:
        return x**2
    else:
        return x/2

# On a scalar
print('x = 10 returns ', foo(10))
print('x = 11 returns ', foo(11))

# On a vector, doesn't work
# print('x = [10, 11, 12] returns ', foo([10, 11, 12]))  # Error
```

```
#> x = 10 returns  5.0
#> x = 11 returns  121
```

Let's vectorize `foo()` so it will work on arrays.

```
# Vectorize foo(). Make it work on vectors.
foo_v = np.vectorize(foo, otypes=[float])

print('x = [10, 11, 12] returns ', foo_v([10, 11, 12]))
print('x = [[10, 11, 12], [1, 2, 3]] returns ', foo_v([[10, 11, 12], [1, 2, 3]]))
```

```
#> x = [10, 11, 12] returns  [   5.  121.    6.]
#> x = [[10, 11, 12], [1, 2, 3]] returns  [[   5.  121.    6.]
#>  [   1.    1.    9.]]
```

This can be very handy whenever you want to make a scalar function work on arrays.

`vectorize` also accepts an optional `otypes` parameter where you provide what the datatype of the output should be. It makes the vectorized function run faster.

7.2 apply_along_axis – Apply a function column wise or row wise

Let me first create a 2D array to show this.

```
# Create a 4x10 random array
np.random.seed(100)
arr_x = np.random.randint(1,10,size=[4,10])
arr_x
```

```
#> array([[9, 9, 4, 8, 8, 1, 5, 3, 6, 3],
#>        [3, 3, 2, 1, 9, 5, 1, 7, 3, 5],
#>        [2, 6, 4, 5, 5, 4, 8, 2, 2, 8],
#>        [8, 1, 3, 4, 3, 6, 9, 2, 1, 8]])
```

Let's understand this by solving the following question:

How to find the difference of the maximum and the minimum value in each row?

Well, the normal approach would be to write a for-loop that iterates along each row and then compute the max-min in each iteration.

That sounds alright but it can get cumbersome if you want to do the same column wise or want to implement a more complex computation. Besides, it can consume more keystrokes.

You can do this elegantly using the `numpy.apply_along_axis`.

It takes as arguments:

1. Function that works on a 1D vector (`fund1d`)
2. Axis along which to apply `func1d`. For a 2D array, 1 is row wise and 0 is column wise.
3. Array on which `func1d` should be applied.

Let's implement this.

```
# Define func1d
def max_minus_min(x):
    return np.max(x) - np.min(x)

# Apply along the rows
print('Row wise: ', np.apply_along_axis(max_minus_min, 1, arr=arr_x))

# Apply along the columns
print('Column wise: ', np.apply_along_axis(max_minus_min, 0, arr=arr_x))
```

```
#> Row wise:  [8 8 6 8]
#> Column wise:  [7 8 2 7 6 5 8 5 5 5]
```

7.3 searchsorted – Find the location to insert so the array will remain sorted

what does `numpy.searchsorted` do?

It gives the index position at which a number should be inserted in order to keep the array sorted.

```
# example of searchsorted
x = np.arange(10)
print('Where should 5 be inserted?: ', np.searchsorted(x, 5))
print('Where should 5 be inserted (right)?: ', np.searchsorted(x, 5, side='right'))
```

```
#> Where should 5 be inserted?:  5
#> Where should 5 be inserted (right)?:  6
```

With the smart hack [https://twitter.com/RadimRehurek/status/928671225861296128] by Radim, you can use searchsorted to do sampling elements with probabilities. It's much faster than `np.choice`.

```
# Randomly choose an item from a list based on a predefined probability
lst = range(10000)  # the list
probs = np.random.random(10000); probs /= probs.sum()  # probabilities

%timeit lst[np.searchsorted(probs.cumsum(), np.random.random())]
%timeit np.random.choice(lst, p=probs)
```

```
#> 36.6 µs ± 3.93 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
#> 1.02 ms ± 7.16 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

7.4 How to add a new axis to a numpy array?

Sometimes you might want to convert a 1D array into a 2D array (like a spreadsheet) without adding any additional data.

You might need this in order a 1D array as a single column in a csv file, or you might want to concatenate it with another array of similar shape.

Whatever the reason be, you can do this by inserting a new axis using the `np.newaxis`.

Actually, using this you can raise an array of a lower dimension to a higher dimension.

```
# Create a 1D array
x = np.arange(5)
print('Original array: ', x)

# Introduce a new column axis
x_col = x[:, np.newaxis]
print('x_col shape: ', x_col.shape)
print(x_col)

# Introduce a new row axis
x_row = x[np.newaxis, :]
print('x_row shape: ', x_row.shape)
print(x_row)
```

```
#> Original array:  [0 1 2 3 4]
#> x_col shape:  (5, 1)
#> [[0]
#>  [1]
#>  [2]
#>  [3]
#>  [4]]
#> x_row shape:  (1, 5)
#> [[0 1 2 3 4]]
```

7.5 More Useful Functions

Digitize

Use `np.digitize` to return the index position of the bin each element belongs to.

```
# Create the array and bins
x = np.arange(10)
bins = np.array([0, 3, 6, 9])

# Get bin allotments
np.digitize(x, bins)
```

```
#> array([1, 1, 1, 2, 2, 2, 3, 3, 3, 4])
```

Clip

Use np.clip to cap the numbers within a given cutoff range. All number lesser than the lower limit will be replaced by the lower limit. Same applies to the upper limit also.

```
# Cap all elements of x to lie between 3 and 8
np.clip(x, 3, 8)
```

```
#> array([3, 3, 3, 3, 4, 5, 6, 7, 8, 8])
```

Histogram and Bincount

Both `histogram()` and `bincount()` gives the frequency of occurences. But with certain differences.

While `histogram()` gives the frequency counts of the bins, `bincount()` gives the frequency count of all the elements in the range of the array between the min and max values. Including the values that did not occur.

```
# Bincount example
x = np.array([1,1,2,2,2,4,4,5,6,6,6]) # doesn't need to be sorted
np.bincount(x) # 0 occurs 0 times, 1 occurs 2 times, 2 occurs thrice, 3 occurs 0 time
s, ...

# Histogram example
counts, bins = np.histogram(x, [0, 2, 4, 6, 8])
print('Counts: ', counts)
print('Bins: ', bins)
```

```
#> Counts:  [2 3 3 3]
#> Bins:  [0 2 4 6 8]
```

8. What is missing in numpy?

So far we have covered a good number of techniques to do data manipulations with numpy. But there are a considerable number of things you can't do with numpy directly. At least to my limited knowledge. Let me list a few:

1. No direct function to merge two 2D arrays based on a common column.
2. Create pivot tables directly
3. No direct way of doing 2D cross tabulations.
4. No direct method to compute statistics (like mean) grouped by unique values in an array.
5. And more..

Well, the reason I am telling you this is these shortcomings are nicely handled by the spectacular pandas library which I will talk about in the upcoming pandas tutorial.

Meanwhile, you might want to test your skills with the numpy practice exercises (https://www.machinelearningplus.com/101-numpy-exercises-python/).

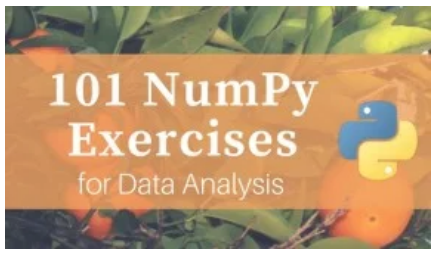
Related

(https://www.machinelearningplus.com/python/numpy-tutorial-part1-array-python-examples/)

Numpy Tutorial Part 1 - Introduction to Arrays (https://www.machinelearningplus.com/python/numpy-tutorial-part1-array-python-examples/)

February 7, 2018

In "Python"



(https://www.machinelearningplus.com/101-numpy-exercises-python/)

101 NumPy Exercises for Data Analysis (Python) (https://www.machinelearningplus.com/101-numpy-exercises-python/)

February 26, 2018

In "Python"

(https://www.machinelearningplus.com/101-pandas-exercises-python/)

101 Pandas Exercises for Data Analysis (https://www.machinelearningplus.com/101-pandas-exercises-python/)

April 27, 2018

In "Python"

You may also like:

Feedback

22_Density_Plot_Matpl
min

Top 50 matplotlib Visualizations - The Master Plots (w/ Full Python Code) | ML+

output_22_1

Linear Regression - A Complete Introduction in R with Examples

2_Bubble_Plot_Matplo
min

101 NumPy Exercises for Data Analysis (Python) - ML+

Gensim Topic Modeling - A Guide to Building Best LDA models

How Naive Bayes Algorithm Works? (with example and full code) | ML+

Tags:Data_Manipulation (https://www.machinelearningplus.com/tag/data-manipulation/), Numpy (https://www.machinelearningplus.com/tag/numpy/), Python (https://www.machinelearningplus.com/tag/python/)

Li Guangyu • 2 months ago

thank you!

Reply • Share ›

Nguyen Thi Thuy • 8 months ago

It is very useful. Thank you so much.

Reply • Share ›

Om Shankar • a year ago

Clear explanation of both Numpy series. Really useful. Thanks, a lot!

Reply • Share ›

Dr. vadivel • a year ago

very nice article !!

• Reply • Share ›

Ann • a year ago

Excellent tutorial.Helped me lots.

• Reply • Share ›

Nagendra • a year ago

very nice tutorial

• Reply • Share ›

kulvinder • a year ago

excellent compilation helped a lot !!
thx

• Reply • Share ›

Selva Prabhakaran ➚ kulvinder • a year ago

Glad to know :)

• Reply • Share ›

[https://[illegible]Julian]

Search

Recent Posts

Vector Autoregression (VAR) – Comprehensive Guide with Examples in Python (https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/)

Mahalonobis Distance – Understanding the math with examples (python) (https://www.machinelearningplus.com/statistics/mahalanobis-distance/)

datetime in Python – Simplified Guide with Clear Examples (https://www.machinelearningplus.com/python/datetime-python-examples/)

Python Logging – Simplest Guide with Full Code and Examples (https://www.machinelearningplus.com/python/python-logging-guide/)

Matplotlib Histogram – How to Visualize Distributions in Python (https://www.machinelearningplus.com/plots/matplotlib-histogram-python-examples/)

ARIMA Model – Complete Guide to Time Series Forecasting in Python (https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/)

Time Series Analysis in Python – A Comprehensive Guide with Examples (https://www.machinelearningplus.com/time-series/time-series-analysis-python/)

Matplotlib Tutorial – A Complete Guide to Python Plot w/ Examples (https://www.machinelearningplus.com/plots/matplotlib-tutorial-complete-guide-python-plot-examples/)

Topic modeling visualization – How to present the results of LDA models? (https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/)

Top 50 matplotlib Visualizations – The Master Plots (with full python code) (https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/)

List Comprehensions in Python – My Simplified Guide (https://www.machinelearningplus.com/python/list-comprehensions-in-python/)

Python @Property Explained – How to Use and When? (Full Examples) (https://www.machinelearningplus.com/python/python-property/)

How Naive Bayes Algorithm Works? (with example and full code) (https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/)

Parallel Processing in Python – A Practical Guide with Examples (https://www.machinelearningplus.com/python/parallel-processing-python/)

Cosine Similarity – Understanding the math and how it works (with python codes) (https://www.machinelearningplus.com/nlp/cosine-similarity/)

Gensim Tutorial – A Complete Beginners Guide (https://www.machinelearningplus.com/nlp/gensim-tutorial/)

Lemmatization Approaches with Examples in Python (https://www.machinelearningplus.com/nlp/lemmatization-examples-python/)

Feature Selection – Ten Effective Techniques with Examples (https://www.machinelearningplus.com/machine-learning/feature-selection/)

101 Pandas Exercises for Data Analysis (https://www.machinelearningplus.com/python/101-pandas-exercises-python/)

LDA in Python – How to grid search best topic models? (https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/)

Topic Modeling with Gensim (Python) (https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/)

Python debugging with pdb (https://www.machinelearningplus.com/python/python-debugging/)

Caret Package – A Practical Guide to Machine Learning in R (https://www.machinelearningplus.com/machine-learning/caret-package/)

101 NumPy Exercises for Data Analysis (Python) (https://www.machinelearningplus.com/python/101-numpy-exercises-python/)

Numpy Tutorial Part 2 – Vital Functions for Data Analysis (https://www.machinelearningplus.com/python/numpy-tutorial-python-part2/)

Tags

@property (https://www.machinelearningplus.com/tag/property/) Bigrams (https://www.machinelearningplus.com/tag/bigrams/)

Classification (https://www.machinelearningplus.com/tag/classification/)

Corpus (https://www.machinelearningplus.com/tag/corpus/)

Cosine Similarity (https://www.machinelearningplus.com/tag/cosine-similarity/)

Data Manipulation (https://www.machinelearningplus.com/tag/data-manipulation/)

Debugging (https://www.machinelearningplus.com/tag/debugging/) Doc2Vec (https://www.machinelearningplus.com/tag/doc2vec/)

Evaluation Metrics (https://www.machinelearningplus.com/tag/evaluation-metrics/)

FastText (https://www.machinelearningplus.com/tag/fasttext/)

Feature Selection (https://www.machinelearningplus.com/tag/feature-selection/)

Gensim (https://www.machinelearningplus.com/tag/gensim/)

klaR (https://www.machinelearningplus.com/tag/klar/) LDA (https://www.machinelearningplus.com/tag/lda/)

Lemmatization (https://www.machinelearningplus.com/tag/lemmatization/)

Linear Regression (https://www.machinelearningplus.com/tag/linear-regression/)

Logistic (https://www.machinelearningplus.com/tag/logistic/) LSI (https://www.machinelearningplus.com/tag/lsi/)

Matplotlib (https://www.machinelearningplus.com/tag/matplotlib/)

Multiprocessing (https://www.machinelearningplus.com/tag/multiprocessing/)

Naive Bayes (https://www.machinelearningplus.com/tag/naive-bayes/)

NLP (https://www.machinelearningplus.com/tag/nlp/)

NLTK (https://www.machinelearningplus.com/tag/nltk/)

Numpy (https://www.machinelearningplus.com/tag/numpy/)

Pandas (https://www.machinelearningplus.com/tag/pandas/)

Parallel Processing (https://www.machinelearningplus.com/tag/parallel-processing/)

Phraser (https://www.machinelearningplus.com/tag/phraser/)

Practice Exercise (https://www.machinelearningplus.com/tag/practice-exercise/)

Python (https://www.machinelearningplus.com/tag/python/)

R (https://www.machinelearningplus.com/tag/r/)

Regex (https://www.machinelearningplus.com/tag/regex/)

Regression (https://www.machinelearningplus.com/tag/regression/)

Residual Analysis (https://www.machinelearningplus.com/tag/residual-analysis/)

Scikit Learn (https://www.machinelearningplus.com/tag/scikit-learn/)

Significance Tests (https://www.machinelearningplus.com/tag/significance-tests/)

Soft Cosine Similarity (https://www.machinelearningplus.com/tag/soft-cosine-similarity/)

spaCy (https://www.machinelearningplus.com/tag/spacy/) Summarization (https://www.machinelearningplus.com/tag/summarization/)

TaggedDocument (https://www.machinelearningplus.com/tag/taggeddocument/)

TextBlob (https://www.machinelearningplus.com/tag/textblob/) TFIDF (https://www.machinelearningplus.com/tag/tfidf/)

Time Series (https://www.machinelearningplus.com/tag/time-series/)

Topic Modeling (https://www.machinelearningplus.com/tag/topic-modeling/)

Visualization (https://www.machinelearningplus.com/tag/visualization/)

Word2Vec (https://www.machinelearningplus.com/tag/word2vec/)

Feedback