Numpy Array

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming.

NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication

For any scientific project, NumPy is the tool to know. It has been built to work with the N-dimensional array, linear algebra, random number, Fourier transform, etc. It can be integrated to C/C++ and Fortran.

NumPy is a programming language that deals with multi-dimensional arrays and matrices.

On top of the arrays and matrices, NumPy supports a large number of mathematical operations.

Here, we will review the essential functions that you need to know for the tutorial on 'TensorFlow.'

Why use NumPy?

NumPy is memory efficiency, meaning it can handle the vast amount of data more accessible than any other library.

Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping.

On top of that, NumPy is fast. In fact, TensorFlow and Scikit learn to use NumPy array to compute the matrix multiplication in the back end.

- Complete the import statement to load the numpy **package**.
- Use numpy's array **class** to define arr.
- Use arr's sort **method** to sort the numpy array.

```
# import the numpy package

import numpy as np

# create an array class object

arr = np.array([8, 6, 7, 5, 3, 0, 9])

# use the sort

methodarr.sort()
```

```
# Import `numpy` as `np`
import numpy as np

# Make the array `my_array`
my_array = np.array([[1,2,3,4], [5,6,7,8]],
dtype=np.int64)

# Print `my_array`
print(my_array)
```

<script.py> output:

```
[[1 2 3 4]
 [5 6 7 8]]
```

```
# Create an array of ones
np.ones((3,4))

# Create an array of zeros
np.zeros((2,3,4),dtype=np.int16)

# Create an array with random values
np.random.random((2,2))

# Create an empty array
np.empty((3,2))

print(np.empty((3,2)))
# Create a full array
np.full((2,2),7)

# Create an array of evenly-spaced values
np.arange(10,25,5)
print(np.arange(10,25,5))

# Create an array of evenly-spaced values
np.linspace(0,2,9)
print(np.linspace(0,2,9))
```

```
import numpy as np

# This is your data in the text file
# Value1  Value2  Value3
# 0.2536  0.1008  0.3857
# 0.4839  0.4536  0.3561
# 0.1292  0.6875  0.5929
# 0.1781  0.3049  0.8928
# 0.6253  0.3486  0.8791

x, y, z = np.loadtxt('data.txt',
                     skiprows=1,
                     unpack=True)

print(x,y,z)
```

#output

```
runfile('D:/python_examples/untitled16.py',
wdir='D:/python_examples')
[0.2536 0.4839 0.1292 0.1781 0.6253] [0.1008
0.4536 0.6875 0.3049 0.3486] [0.3857 0.3561
0.5929 0.8928 0.8791]
```

```
import numpy as np

# Your data in the text file
# Value1  Value2  Value3
# 0.4839  0.4536  0.3561
# 0.1292  0.6875  MISSING
# 0.1781  0.3049  0.8928
# MISSING 0.5801  0.2038
# 0.5993  0.4357  0.7410


my_array2 = np.genfromtxt('data.txt',
                          skip_header=1,
                          filling_values=-999)
print(my_array2)
```

#OUTPUT

```
runfile('D:/python_examples/numpy_missing_value
s example.py', wdir='D:/python_examples')
[[ 2.536e-01  1.008e-01  3.857e-01]
 [ 4.839e-01  4.536e-01  3.561e-01]
 [ 1.292e-01  6.875e-01  5.929e-01]
 [-9.990e+02  3.049e-01  8.928e-01]
 [ 6.253e-01  3.486e-01  8.791e-01]]
```

Your First NumPy Array

In this session, we're going to dive into the world of baseball. Along the way, you'll get comfortable with the basics of numpy, a powerful package to do data science.

A list baseball has already been defined in the Python script, representing the height of some baseball players in centimeters. Can you add some code here and there to create a numpy array from it?

baseball = [180, 215, 210, 210, 188, 176, 209, 200]

Import the numpy package as np, so that you can refer to numpy with np.

Use np.array() to create a numpy array from baseball.

Name this array np_baseball.

Print out the type of np_baseball to check that you got it right.

```
# Create list baseball
baseball = [180, 215, 210, 210, 188, 176, 209, 200]

# Import the numpy package as np
import numpy as np

# Create a Numpy array from baseball: np_baseball
np_baseball = np.array(baseball)

# Print out type of np_baseball
print(type(np_baseball))
```

Create a numpy array from height. Name this new array np_height.

Print np_height.

Multiply np_height with 0.0254 to convert all height measurements from inches to meters. Store the new values in a new array, np_height_m.

Print out np_height_m and check if the output makes sense.

Baseball player's BMI

The MLB also offers to let you analyze their weight data.

Again, both are available as regular Python lists: height and weight. height is in inches and weight is in pounds.

It's now possible to calculate the BMI of each baseball player.

Python code to convert height to a numpy array with the correct units is already available in the workspace.

Follow the instructions step by step and finish the game

```
# height and weight are available as a regular
lists

# Import numpy
import numpy as np

# Create array from height with correct units:
np_height_m
np_height_m = np.array(height) * 0.0254

# Create array from weight with correct units:
np_weight_kg
np_weight_kg=np.array(weight)*0.453592

# Calculate the BMI: bmi
bmi=np_weight_kg/np_height_m**2

# Print out bmi
print(bmi)
```

Mathematical Operations on an Array

You could perform mathematical operations like additions, subtraction, division and multiplication on an array. The syntax is the array name followed by the operation (+.-,*,/) followed by the operand

Example:

numpy_array_from_list + 10

Output:

array([11, 19, 18, 13])

This operation adds 10 to each element of the numpy array.

2 Dimension Array

You can add a dimension with a ","coma

Note that it has to be within the bracket []

```
### 2 dimension
c = np.array([(1,2,3),
              (4,5,6)])
print(c.shape)
(2, 3)
```

3 Dimension Array

Higher dimension can be constructed as follow:

```
### 3 dimension
d = np.array([
    [[1, 2,3],
       [4, 5, 6]],
    [[7, 8,9],
       [10, 11, 12]]
])
print(d.shape)
(2, 2, 3)
```

```
import numpy as np

e = np.array([(1,2,3), (4,5,6)])

 print(e) e.reshape(3,2)
```

Flatten Data

When you deal with some neural network like convnet, you need to flatten the array. You can use flatten(). The syntax is

```
numpy.flatten(order='C')


[1,2,3,4,5,6] // flattened array
```

numpy.hstack() and numpy.vstack() in Python with Example

In this architecture we are using Monolith architecture i.e. all collaborating components combine all in one application.

What is hstack?

With hstack you can appened data horizontally. This is a very convinient function in Numpy.
Lets do it with an example:

```
## Horitzontal Stack
import numpy as np
f = np.array([1,2,3])
g = np.array([4,5,6])

print('Horizontal Append:', np.hstack((f, g)))
```

What is vstack in numpy?

With vstack you can appened data vertically.

Lets do it with an example:

```
## Vertical Stack
import numpy as np
f = np.array([1,2,3])
g = np.array([4,5,6])

print('Vertical Append:', np.vstack((f, g)))
```

```
Vertical Append: [[1 2 3]
 [4 5 6]]
```

Generate Random Numbers

To generate random numbers for Gaussian distribution use
numpy.random.normal(loc, scale, size)

Here
•Loc: the mean. The center of distribution
•scale: standard deviation.
•Size: number of returns

•## Generate random nmber from normal distribution
•normal_array = np.random.normal(5, 0.5, 10)
•print(normal_array)
•[5.56171852 4.84233558 4.65392767 4.946659 4.85165567 5.61211317 4.46704244 5.22675736
4.49888936 4.68731125]

Mathematical functions

a.sum()	Array-wise sum
a.min()	Array-wise minimum value
b.max(axis=0)	Maximum value of an array row
b.cumsum(axis=1)	Cumulative sum of the elements
a.mean()	Mean
b.median()	Median
a.corrcoef()	Correlation coefficient
np.std(b)	Standard deviation

How Do Array Mathematics Work?

You've seen that broadcasting is handy when you're doing arithmetic operations. In this section, you'll discover some of the functions that you can use to do mathematics with arrays.

As such, it probably won't surprise you that you can just use `+`, `-`, `*`, `/` or `%` to add, subtract, multiply, divide or calculate the remainder of two (or more) arrays. However, a big part of why NumPy is so handy, is because it also has functions to do this. The equivalent functions of the operations that you have seen just now are, respectively, `np.add()`, `np.subtract()`, `np.multiply()`, `np.divide()` and `np.remainder()`.

You can also easily do exponentiation and taking the square root of your arrays with `np.exp()` and `np.sqrt()`, or calculate the sines or cosines of your array with `np.sin()` and `np.cos()`. Lastly, its' also useful to mention that there's also a way for you to calculate the natural logarithm with `np.log()` or calculate the dot product by applying the `dot()` to your array.

numpy.dot(): Dot Product in Python using Numpy

Dot Product

Numpy is powerful library for matrices computation. For instance, you can compute the dot product with np.dot

```
## Linear algebra
### Dot product: product of two arrays
f = np.array([1,2])
g = np.array([4,5])
### 1*4+2*5
np.dot(f, g)
```

NumPy Matrix Multiplication with np.matmul() Example

Matrix Multiplication

The Numpu matmul() function is used to return the matrix product of 2 arrays. Here is how it works

1) 2-D arrays, it returns normal product

2) Dimensions > 2, the product is treated as a stack of matrix

3) 1-D array is first promoted to a matrix, and then the product is calculated

Statistical function

Numpy is equipped with the robust statistical function as listed below

Check out this small list of aggregate functions:

a.sum()	Array-wise sum
a.min()	Array-wise minimum value
b.max(axis=0)	Maximum value of an array row
b.cumsum(axis=1)	Cumulative sum of the elements
a.mean()	Mean
b.median()	Median
a.corrcoef()	Correlation coefficient
np.std(b)	Standard deviation

```
import numpy as np
normal_array = np.random.normal(5, 0.5, 10)
print(normal_array)

### Min
print(np.min(normal_array))

### Max
print(np.max(normal_array))

### Mean
print(np.mean(normal_array))

### Median
print(np.median(normal_array))

### Sd
print(np.std(normal_array))
```

How To Manipulate Arrays

Performing mathematical operations on your arrays is one of the things that you'll be doing, but probably most importantly to make this and the broadcasting work is to know how to manipulate your arrays.

Below are some of the most common manipulations that you'll be doing.

How To Transpose Your Arrays

What transposing your arrays actually does is permuting the dimensions of it. Or, in other words, you switch around the shape of the array. Let's take a small example to show you the effect of transposition:

script.py

```
# Print `my_2d_array`
print(my_2d_array)

# Transpose `my_2d_array`
print(np.transpose(my_2d_array))

# Or use `T` to transpose `my_2d_array`
print(my_2d_array.T)
```

IPython Shell

```
[[1 2 3 4]
 [5 6 7 8]]
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
```

```
### Matmul: matruc product of two arrays
h = [[1,2],[3,4]]
i = [[5,6],[7,8]]
### 1*5+2*7 = 19
np.matmul(h, i)


#ouput
array([[19, 22],
       [43, 50]])
```

Use np.array() to create a 2D numpy array from baseball. Name it np_baseball.

Print out the type of np_baseball.

Print out the shape attribute of np_baseball. Use np_baseball.shape.

```
# Create baseball, a list of lists
baseball = [[180, 78.4],
            [215, 102.7],
            [210, 98.5],
            [188, 75.2]]

# Import numpy
import numpy as np

# Create a 2D numpy array from baseball:
np_baseball

np_baseball=np.array(baseball)

# Print out the type of np_baseball

print(type(np_baseball))

# Print out the shape of np_baseball

print(np_baseball.shape)
```

Import numpy as np.

Declare variable my_matrix and set it to [[1,2,3,4], [5,6,7,8]].

Declare a function called return_array(), which takes a list matrix as input, and returns an array object as output. In the body, declare a variable array set it to np.array(matrix, dtype = float).

Call return_array() on the my_matrix list, and print out the output.

```
# Import numpy as np
import numpy as np

# List input: my_matrix
my_matrix = [[1,2,3,4], [5,6,7,8]]

# Function that converts lists to arrays: return_array
def return_array(matrix):
    array = np.array(matrix, dtype = float)
    return array

# Call return_array on my_matrix, and print the output
print(return_array(my_matrix))
```

Reshaping Versus Resizing Your Arrays

You might have read in the broadcasting section that the dimensions of your arrays need to be compatible if you want them to be good candidates for arithmetic operations. But the question of what you should do when that is not the case, was not answered yet.

Well, this is where you get the answer!

What you can do if the arrays don't have the same dimensions, is resize your array. You will then return a new array that has the shape that you passed to the `np.resize()` function. If you pass your original array together with the new dimensions, and if that new array is larger than the one that you originally had, the new array will be filled with copies of the original array that are repeated as many times as is needed.

However, if you just apply `np.resize()` to the array and you pass the new shape to it, the new array will be filled with zeros.

Let's try this out with an example:

script.py

```
1   # Print the shape of `x`
2   print(x.shape)
3
4   # Resize `x` to ((6,4))
5   np.resize(x, (6,4))
6
7   # Try out this as well
8   x.resize((6,4))
9
10  # Print out `x`
11  print(x)
```

IPython Shell

```
(3, 4)
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]

In [1]:
```

Boolean array indexing:

Boolean array indexing lets you pick out arbitrary elements of an array.

Frequently this type of indexing is used to select the elements of an array that satisfy some condition. Here is an example:

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                     # this returns a numpy array of Booleans of the same
                     # shape as a, where each slot of bool_idx tells
                     # whether that element of a is > 2.

print(bool_idx)      # Prints "[[False False]
                     #         [ True  True]
                     #         [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

Broadcasting

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations.

Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

For example, suppose that we want to add a constant vector to each row of a matrix. We could do it like this:

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

# Now y is the following
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
print(y)
```

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))   # Stack 4 copies of v on top of each other
print(vv)                 # Prints "[[1 0 1]
                          #         [1 0 1]
                          #         [1 0 1]
                          #         [1 0 1]]"
y = x + vv  # Add x and vv elementwise
print(y)  # Prints "[[ 2  2  4
          #         [ 5  5  7]
          #         [ 8  8 10]
          #         [11 11 13]]"
```

Numpy broadcasting allows us to perform this computation without actually creating multiple copies of v. Consider this version, using broadcasting:

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v
# Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
        #         [ 5  5  7]
        #         [ 8  8 10]
        #         [11 11 13]]"
```

Broadcasting two arrays together follows these rules:

1.If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s until both shapes have the same length.

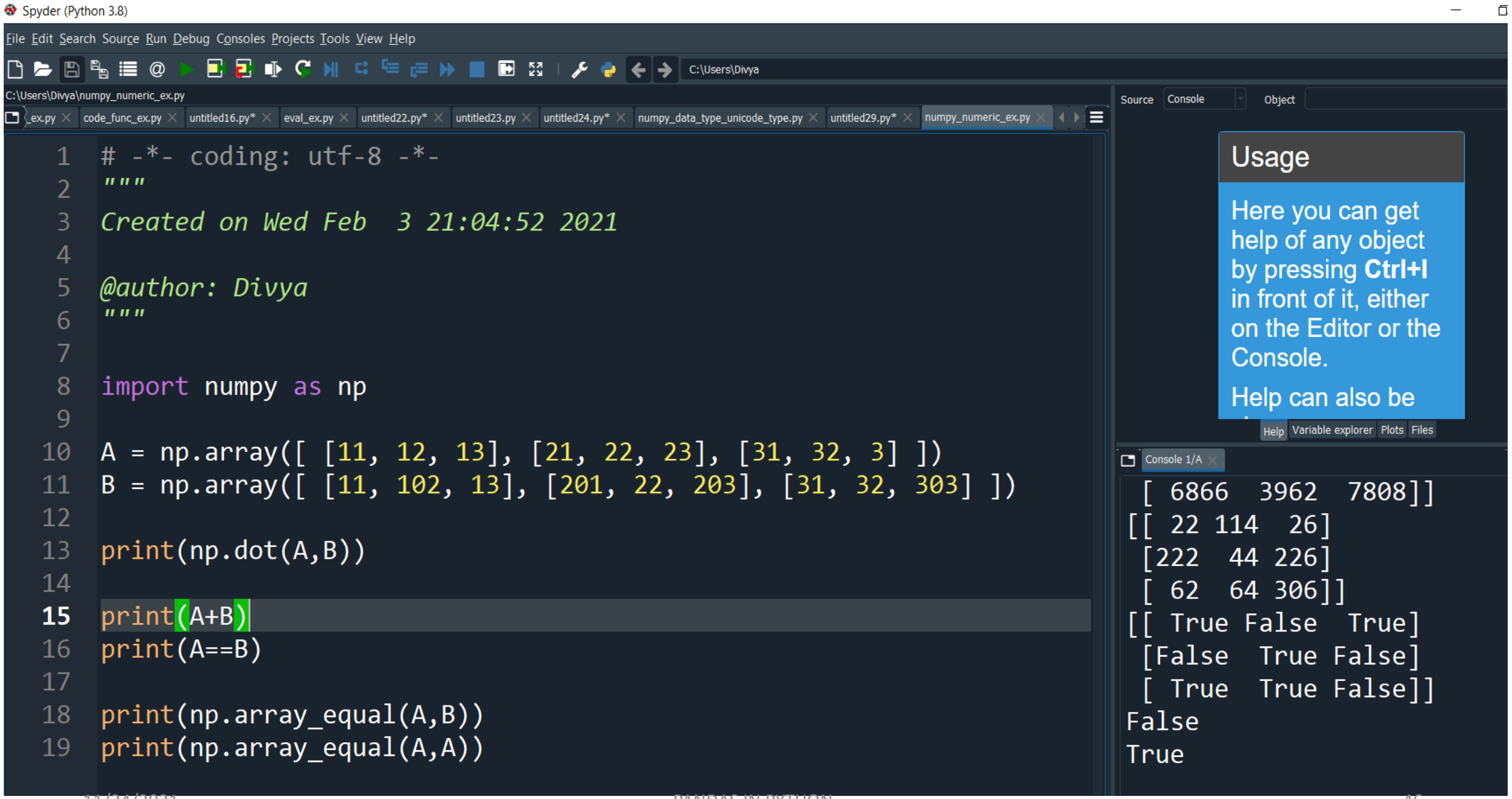
2.The two arrays are said to be *compatible* in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.

3.The arrays can be broadcast together if they are compatible in all dimensions.

4.After broadcasting, each array behaves as if it had shape equal to the elementwise maximum of shapes of the two input arrays.

5.In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension

```
# Create arrays
import numpy as np
my_house = np.array([18.0, 20.0, 10.75, 9.50])
your_house = np.array([14.0, 24.0, 14.25, 9.0])

# my_house greater than 18.5 or smaller than 10

np.logical_and(my_house > 18.5,
               my_house < 10)
# Both my_house and your_house smaller than 11

np.logical_or(my_house <11,
              your_house < 11)
```

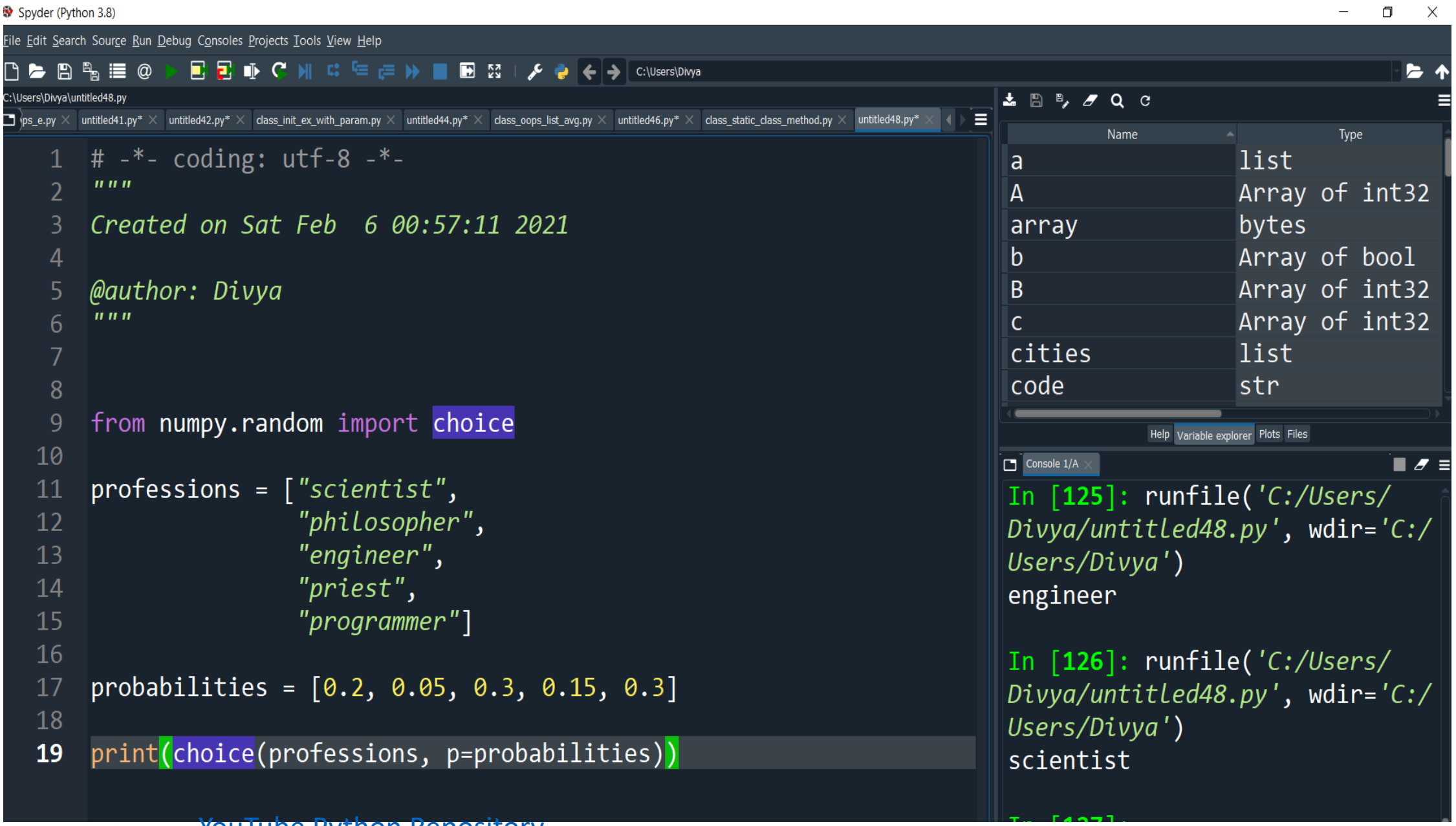

Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Divya
C:\Users\Divya\numpy_numeric_ex.py
code_func_ex.py
untitled16.py*
eval_ex.py
untitled22.py*
untitled23.py
untitled24.py*
numpy_data_type_unicode_type.py
untitled29.py*
numpy_numeric_ex.py
-*- coding: utf-8 -*-
"""
Created on Wed Feb 3 21:04:52 2021
@author: Divya
"""
import numpy as np
A = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 3]])
B = np.array([[11, 102, 13], [201, 22, 203], [31, 32, 303]])
print(np.dot(A,B))
print(A+B)
print(A==B)
print(np.array_equal(A,B))
print(np.array_equal(A,A))
Source
Console
Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be
Help
Variable explorer
Plots
Files
Console 1/A
[6866 3962 7808]]
[[22 114 26]
[222 44 226]
[62 64 306]]
[[True False True]
[False True False]
[True True False]]
False
True

Weighted Random Choice with Numpy

To produce a weighted choice of an array like object, we can also use the choice function of the numpy.random package. Actually, you should use functions from well-established module like 'NumPy' instead of reinventing the wheel by writing your own code. In addition the 'choice' function from NumPy can do even more. It generates a random sample from a given 1-D array or array like object like a list, tuple and so on. The function can be called with four parameters:

```
choice(a, size=None, replace=True, p=None)
```

Parameter	Meaning
a	a 1-dimensional array-like object or an int. If it is an array-like object, the function will return a random sample from the elements. If it is an int, it behaves as if we called it with np.arange(a)
size	This is an optional parameter defining the output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. The Default is None, in which case a single value will be returned.
replace	An optional boolean parameter. It is used to define whether the output sample will be with or without replacements.
p	An optional 1-dimensional array-like object, which contains the probabilities associated with each entry in a. If it is not given the sample assumes a uniform distribution over all entries in a.


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Divya
C:\Users\Divya\untitled48.py
ps_e.py
untitled41.py*
untitled42.py*
class_init_ex_with_param.py
untitled44.py*
class_oops_list_avg.py
untitled46.py*
class_static_class_method.py
untitled48.py*
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sat Feb 6 00:57:11 2021
4
5 @author: Divya
6 """
7
8
9 from numpy.random import choice
10
11 professions = ["scientist",
12 "philosopher",
13 "engineer",
14 "priest",
15 "programmer"]
16
17 probabilities = [0.2, 0.05, 0.3, 0.15, 0.3]
18
19 print(choice(professions, p=probabilities))
Name
Type
a
list
A
Array of int32
array
bytes
b
Array of bool
B
Array of int32
c
Array of int32
cities
list
code
str
Help
Variable explorer
Plots
Files
Console 1/A
In [125]: runfile('C:/Users/Divya/untitled48.py', wdir='C:/Users/Divya')
engineer
In [126]: runfile('C:/Users/Divya/untitled48.py', wdir='C:/Users/Divya')
scientist

- Import numpy using the standard alias np.
- Assign the numerical values in the DataFrame df to an array np_vals using the attribute values.
- Pass np_vals into the NumPy method log10() and store the results in np_vals_log10.
- Pass the entire df DataFrame into the NumPy method log10() and store the results in df_log10.
- Inspect the output of the print() code to see the type() of the variables that you created.

Spyder (Python 3.8)

Your screen sharing is paused Stop Share

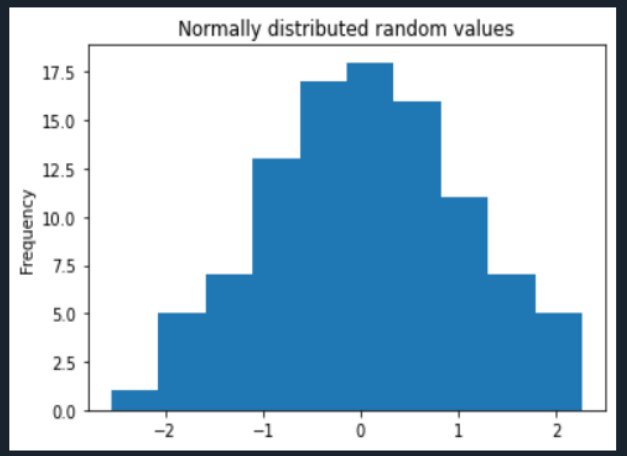
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Divya

C:\Users\Divya\untitled28.py

untitled21.py* | untitled22.py* | untitled23.py* | numpy_2d_array_analysis.py | untitled25.py* | untitled26.py* | numpy_Structured data_type.py | untitled28.py*

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Feb  8 21:39:22 2021
4
5  @author: Divya
6  """
7
8  import numpy as np
9
10 a=np.array([1,2,3,4,5,6,7])
11 b=np.array([])
12
13 print(a+b)
```



125 %

Help | Variable explorer | Plots | Files

Console 1/A

```
  File "C:\Users\Divya\untitled28.py",
line 13, in <module>
    print(a+b)

ValueError: operands could not be
broadcast together with shapes (7,)
(0,)


In [57]:
```

Spyder (Python 3.8)

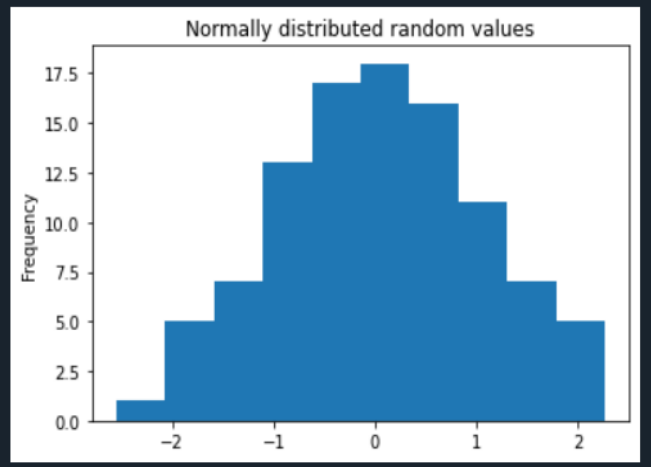
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Divya

C:\Users\Divya\untitled31.py

mpy_2d_array_analysis.py | untitled25.py* | untitled26.py* | numpy_Structured data_type.py | untitled28.py* | untitled29.py* | untitled30.py* | untitled31.py*

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Feb  8 21:57:45 2021
4
5  @author: Divya
6  """
7
8  import numpy as np
9  str="hello am python"
10 print(np.char.splitlines("Welcome\nto\npython"))
11
12 np.char.replace()
```



125 %

Help | Variable explorer | Plots | Files

Console 1/A

```
0

In [68]: runfile('C:/Users/Divya/
untitled31.py', wdir='C:/Users/Divya
['Welcome', 'to', '\\python']

In [69]: runfile('C:/Users/Divya/
untitled31.py', wdir='C:/Users/Divya
['Welcome', 'to', 'python']

In [70]:
```

How To Append Arrays

When you append arrays to your original array, they are "glued" to the end of that original array. If you want to make sure that what you append does not come at the end of the array, you might consider inserting it. Go to the next section if you want to know more.

Appending is a pretty easy thing to do thanks to the NumPy library; You can just make use of the `np.append()`.

Check how it's done in the code chunk below. Don't forget that you can always check which arrays are loaded in by typing, for example, `my_array` in the IPython shell and pressing ENTER.

script.py | solution.py

```
# Append a 1D array to your `my_array`
new_array = np.append(my_array, [7, 8, 9, 10])

# Print `new_array`
print(new_array)

# Append an extra column to your `my_2d_array`
new_2d_array = np.append(my_2d_array, [[7], [8]], axis=1)

# Print `new_2d_array`
print(new_2d_array)
```

IPython Shell

```
In [1]:
```

YouTube Python Repository

How To Join And Split Arrays

You can also 'merge' or join your arrays. There are a bunch of functions that you can use for that purpose and most of them are listed below.

Try them out, but also make sure to test out what the shape of the arrays is in the IPython shell. The arrays that have been loaded are `x`, `my_array`, `my_resized_array` and `my_2d_array`.

script.py

```
# Concatentate `my_array` and `x`
print(np.concatenate((my_array,x)))

# Stack arrays row-wise
print(np.vstack((my_array, my_2d_array)))

# Stack arrays row-wise
print(np.r_[my_resized_array, my_2d_array])

# Stack arrays horizontally
print(np.hstack((my_resized_array, my_2d_array)))

# Stack arrays column-wise
print(np.column_stack((my_resized_array, my_2d_array)))

# Stack arrays column-wise
print(np.c_[my_resized_array, my_2d_array])
```

IPython Shell

```
[ 1.  2.  3.  4.  1.  1.  1.  1.]
[[1 2 3 4]
 [1 2 3 4]
 [5 6 7 8]]
[[1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]
 [5 6 7 8]]
[[1 2 3 4 1 2 3 4]
 [1 2 3 4 5 6 7 8]]
[[1 2 3 4 1 2 3 4]
 [1 2 3 4 5 6 7 8]]
[[1 2 3 4 1 2 3 4]
 [1 2 3 4 5 6 7 8]]

In [1]:
```

How To Visualize NumPy Arrays

Lastly, something that will definitely come in handy is to know how you can plot your arrays. This can especially be handy in data exploration, but also in later stages of the data science workflow, when you want to visualize your arrays.

With `np.histogram()`

Contrary to what the function might suggest, the `np.histogram()` function doesn't draw the histogram but it does compute the occurrences of the array that fall within each bin; This will determine the area that each bar of your histogram takes up.

What you pass to the `np.histogram()` function then is first the input data or the array that you're working with. The array will be flattened when the histogram is computed.

script.py

```
# Import `numpy` as `np`
import numpy as np

# Initialize your array
my_3d_array = np.array([[[1,2,3,4], [5,6,7,8]], [[1,2,3
,4], [9,10,11,12]]], dtype=np.int64)

# Pass the array to `np.histogram()`
print(np.histogram(my_3d_array))
```

IPython Shell

```
(array([4, 2, 2, 1, 1, 1, 1, 1, 1, 2]), array([  1. ,   2.1,
    3.2,   4.3,   5.4,   6.5,   7.6,   8.7,   9.8,
       10.9,  12. ]))
(array([0, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2]), array([ 0,  1,  2
    ,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]))

In [1]:
```

Numpy provides the following functions to perform the different algebraic calculations on the input data.

SN	Function	Definition
1	dot()	It is used to calculate the dot product of two arrays.
2	vdot()	It is used to calculate the dot product of two vectors.
3	inner()	It is used to calculate the inner product of two arrays.
4	matmul()	It is used to calculate the matrix multiplication of two arrays.
5	det()	It is used to calculate the determinant of a matrix.
6	solve()	It is used to solve the linear matrix equation.
7	inv()	It is used to calculate the multiplicative inverse of the matrix.

Numpy provides the following bitwise operators.

SN	Operator	Description
1	bitwise_and	It is used to calculate the bitwise and operation between the corresponding array elements.
2	bitwise_or	It is used to calculate the bitwise or operation between the corresponding array elements.
3	invert	It is used to calculate the bitwise not the operation of the array elements.
4	left_shift	It is used to shift the bits of the binary representation of the elements to the left.
5	right_shift	It is used to shift the bits of the binary representation of the elements to the right.

```
1.import numpy as np
2.
3.b = np.array([12, 90, 380, 12, 211])
4.
5.print(np.where(b>12))
6.
7.c = np.array([[20, 24],[21, 23]])
8.
9.print(np.where(c>20))
```

NULL value Operations

`None`

```
None or True # Same for False
True
```

```
None + True # For all operators
TypeError: unsupported operand
None / 3 # For all operators
TypeError: unsupported operand
```

`np.nan`

```
import numpy as np
np.nan or True  # Same for False
nan
```

```
np.nan * True # For all operators
nan
np.nan - 3 # For all operators
nan
```

NULL value Operations

`None`

```
None or True # Same for False
True
```

```
None + True # For all operators
TypeError: unsupported operand
None / 3 # For all operators
TypeError: unsupported operand
```

```
type(None)
NoneType
```

`np.nan`

```
import numpy as np
np.nan or True  # Same for False
nan
```

```
np.nan * True # For all operators
nan
np.nan - 3 # For all operators
nan
```

```
type(np.nan)
float
```

script.py

```
try:
  # Print the sum of two None's
  print("Add operation output of 'None': ", None + np.nan)

except TypeError:
  # Print if error
  print("'None' does not support Arithmetic Operations!!")
```

script.py solution.py

```
1  try:
2      # Print the sum of two None's
3      print("Add operation output of 'None': ", None + None)
4
5  except TypeError:
6      # Print if error
7      print("'None' does not support Arithmetic Operations!!")
```

script.py

```
try:
    # Print the sum of two np.nan's
    print("Add operation output of 'np.nan': ", np.nan+np.nan)

except TypeError:
    # Print if error
    print("'np.nan' does not support Arithmetic Operations!!")
```

```
Add operation output of 'np.nan':  nan
```

script.py | solution.py

```python
try:
  # Print the output of logical OR of two None's
  print("OR operation output of 'None': ", None or None)

except TypeError:
  # Print if error
  print("'None' does not support Logical Operations!!")
```

```
'None' does not support Logical Operations!!
```

script.py

```
try:
  # Print the comparison of two 'np.nan's
  print("'np.nan' comparison output: ", np.nan==np.nan)

except TypeError:
  # Print if error
  print("'np.nan' does not support this operation!!")
```

script.py

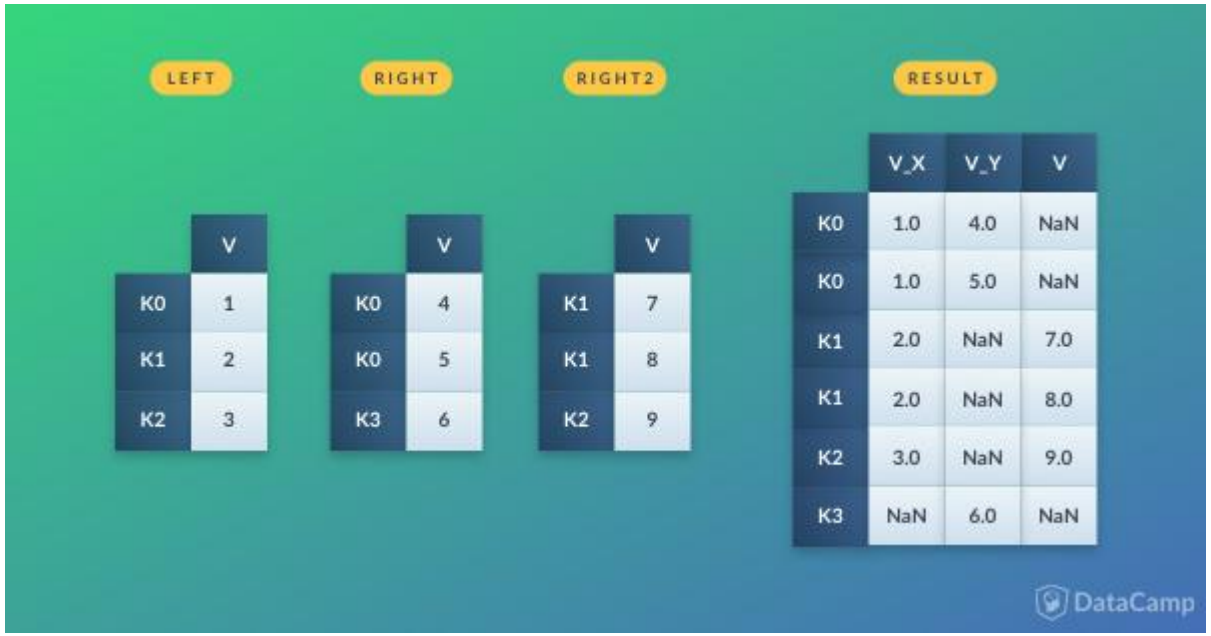
```
try:
  # Print the comparison of two 'None's
  print("'None' comparison output: ", None==None)

except TypeError:
  # Print if error
  print("'None' does not support this operation!!")
```

script.py

```
try:
    # Check if 'np.nan' is 'NaN'
    print("Is 'np.nan' same as nan? ", np.isnan(np.nan))

except TypeError:
    # Print if error
    print("Function 'np.isnan()' does not support this Type!!")
```

```
Is 'np.nan' same as nan?  True
```

Pandas is a popular Python package for data science, and with good reason: it offers powerful, expressive and flexible data structures that make data manipulation and analysis easy, among many other things. The DataFrame is one of these structures.



pandas is built on NumPy and Matplotlib

Rectangular data

Name	Breed	Color	Height (cm)	Weight (kg)	Date of Birth
Bella	Labrador	Brown	56	25	2013-07-01
Charlie	Poodle	Black	43	23	2016-09-16
Lucy	Chow Chow	Brown	46	22	2014-08-25
Cooper	Schnauzer	Gray	49	17	2011-12-11
Max	Labrador	Black	59	29	2017-01-20
Stella	Chihuahua	Tan	18	2	2015-04-20
Bernie	St. Bernard	White	77	74	2018-02-27

What Are Pandas Data Frames?

Data Frames

Those who are familiar with R know the data frame as a way to store data in rectangular grids that can easily be overviewed. Each row of these grids corresponds to measurements or values of an instance, while each column is a vector containing data for a specific variable. This means that a data frame's rows do not need to contain, but can contain, the same type of values: they can be numeric, character, logical, etc.

Now, DataFrames in Python are very similar: they come with the Pandas library, and they are defined as two-dimensional labeled data structures with columns of potentially different types.

In general, you could say that the Pandas DataFrame consists of three main components: the data, the index, and the columns.

Firstly, the DataFrame can contain data that is:

- ***a Pandas DataFrame***
- ***a Pandas Series: a one-dimensional labeled array capable of holding any data type with axis labels or index. An example of a Series object is one column from a DataFrame.***
- ***a NumPy ndarray, which can be a record or structured a two-dimensional ndarray***
- ***dictionaries of one-dimensional ndarray's, lists, dictionaries or Series.***

Note the difference between np.ndarray and np.array() . The former is an actual data type, while the latter is a function to make arrays from other data structures.

Structured arrays allow users to manipulate the data by named fields: in the example below, a structured array of three tuples is created. The first element of each tuple will be called foo and will be of type int, while the second element will be named bar and will be a float.

Record arrays, on the other hand, expand the properties of structured arrays. They allow users to access fields of structured arrays by attribute rather than by index. You see below that the foo values are accessed in the r2 record array.

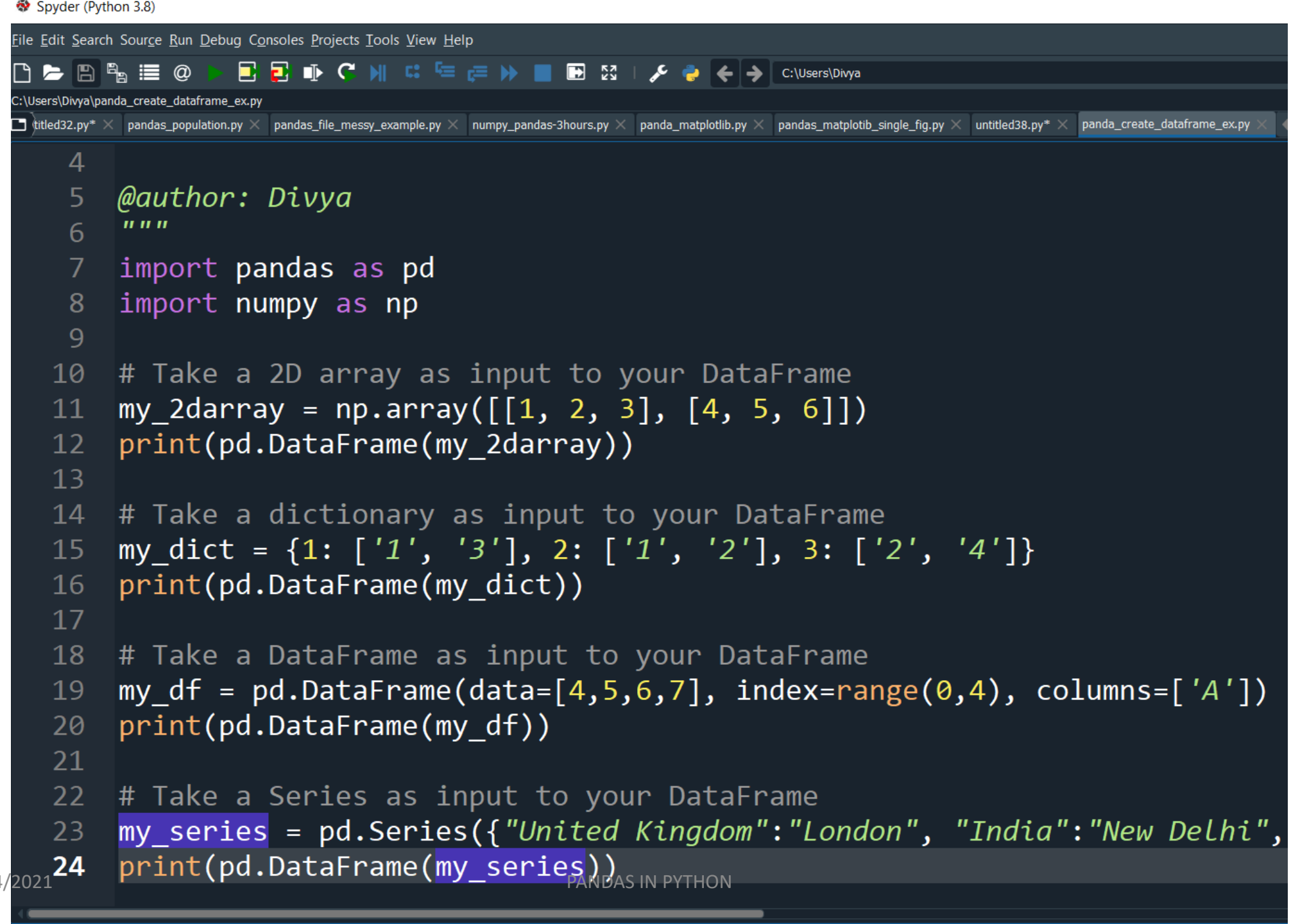
An example:

```
# A structured array
my_array = np.ones(3, dtype=[('foo', int), ('bar', float)]))
print(my_array['foo'])

# A record array
my_array2 = my_array.view(np.recarray)
print(my_array2.foo)
```

Besides data, you can also specify the index and column names for your DataFrame. The index, on the one hand, indicates the difference in rows, while the column names indicate the difference in columns.

The libraries that you need have already been loaded in. The Pandas library is usually imported under the alias pd, while the NumPy library is loaded as np. Remember that when you code in your own data science environment, you shouldn't forget this import step, which you write just like this:

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


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Divya
C:\Users\Divya\panda_create_dataframe_ex.py
titled32.py*
pandas_population.py
pandas_file_messy_example.py
numpy_pandas-3hours.py
panda_matplotlib.py
pandas_matplotib_single_fig.py
untitled38.py*
panda_create_dataframe_ex.py
4
5 @author: Divya
6 """
7 import pandas as pd
8 import numpy as np
9
10 # Take a 2D array as input to your DataFrame
11 my_2darray = np.array([[1, 2, 3], [4, 5, 6]])
12 print(pd.DataFrame(my_2darray))
13
14 # Take a dictionary as input to your DataFrame
15 my_dict = {1: ['1', '3'], 2: ['1', '2'], 3: ['2', '4']}
16 print(pd.DataFrame(my_dict))
17
18 # Take a DataFrame as input to your DataFrame
19 my_df = pd.DataFrame(data=[4,5,6,7], index=range(0,4), columns=['A'])
20 print(pd.DataFrame(my_df))
21
22 # Take a Series as input to your DataFrame
23 my_series = pd.Series({"United Kingdom":"London", "India":"New Delhi",
24 print(pd.DataFrame(my_series))

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



df=pd.DataFrame(data=list(range(10)),index=range(10,110,10)
                ,columns=['a'])

print(df)
```

```
Console 1/A
6  6
7  7
8  8
9  9

IPdb [6]: runfile
matam/.spyder-py3
     a
10   0
20   1
30   2
40   3
50   4
60   5
70   6
80   7
90   8
100  9

IPdb [7]:
```

IPython Shell

```
<script.py> output:
       0  1  2
    0  1  2  3
    1  4  5  6
       1  2  3
    0  1  1  2
    1  3  2  4
       A
    0  4
    1  5
    2  6
    3  7
                              0
    Belgium           Brussels
    India            New Delhi
    United Kingdom      London
    United States   Washington
```

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[]

.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`.

This all might seem very complicated. Let's illustrate all of this with a small example:


C:\Users\Divya\untitled48.py
ed31.py*
untitled32.py*
pandas_file_messy_example.py
untitled38.py*
untitled43.py*
austin_weather_pandas.py
pandas_with_timeseries.py
untitled46.py*
untitled47.py*
untitled48.py*
-*- coding: utf-8 -*-
"""
Created on Wed Feb 10 08:26:57 2021
@author: Divya
"""
import pandas as pd
import numpy as np
a=np.array([[111,20,30,50],[999,30,70,60]])
print(type(a))
print(a)
df=pd.DataFrame(a,columns=['a','b','c','d'],index=range(10,12))
print(df)
print(df.loc[10])
Name
file_clean str
file_messy str
height list
list_to_drop list
my_2darray Array
my_array Array
my_df DataF
my_dict dict
Help
Variable explorer
Plots
Files
Console 1/A
a b c d
10 111 20 30 50
11 999 30 70 60
a 111
b 20
c 30
d 50
Name: 10, dtype: int32
In [135]:

C:\Users\Divya\panda_iloc_example.py

das-3hours.py | panda_matplotlib.py | pandas_matplotib_single_fig.py | untitled38.py* | panda_create_dataframe_ex.py | panda_iloc_example.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Feb  9 01:55:43 2021
4
5  @author: Divya
6  """
7  import pandas as pd
8  import numpy as np
9
10 df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5,
11
12 # Pass `2` to `loc`
13 print(df.loc[2])
14
15 # Pass `2` to `iloc`
16 print(df.iloc[2])
17
18
```

DataFrame with range()

```
import numpy as np
import pandas as pd



df=pd.DataFrame(data=list(range(10)),index=range(10,110,10)
                ,columns=['a'])

print(df)
```

```
Console 1/A
Solar     4
Wind      0
Temp      0
dtype: flo

IPdb [110]
wdir='C:/U
     a
10   0
20   1
30   2
40   3
50   4
60   5
70   6
80   7
90   8
100  9
```

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, just like this:

script.py | solution.py

```
1  df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7,
   8, 9]]), columns=['A', 'B', 'C'])
2
3  # Use `.index`
4  df['D'] = df.index
5
6  # Print `df`
7  print(df)
```

IPython Shell

```
<script.py> output:
      A  B  C  D
   0  1  2  3  0
   1  4  5  6  1
   2  7  8  9  2

In [1]:
```

However, if you want to append columns to your DataFrame, you could also follow the same approach as when you would add an index to your DataFrame: you use `.loc[]` or `.iloc[]`. In this case, you add a Series to an existing DataFrame with the help of `.loc[]`:

script.py | solution.py

```
# Study the DataFrame `df`
print(df)

# Append a column to `df`
df.loc[:, 4] = pd.Series(['5', '6'], index=df.index)

# Print out `df` again to see the changes
print(df)
```

IPython Shell

```
In [1]:
```

script.py | solution.py

```
# Study the DataFrame `df`
print(df)

# Append a column to `df`
df.loc[:, 4] = pd.Series(['5', '6'], index=df.index)

# Print out `df` again to see the changes
print(df)
```

IPython Shell

```
<script.py> output:
       1  2  3
    0  1  1  2
    1  3  2  4
       1  2  3  4
    0  1  1  2  5
    1  3  2  4  6

In [1]:
```

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. You can easily do this with `.reset_index()`. However, you should still watch out, as you can pass several arguments that can make or break the success of your reset:

script.py

```
# Check out the weird index of your dataframe
print(df)

# Use `reset_index()` to reset the values.
df_reset = df.__________(level=0, drop=True)

# Print `df_reset`
print(df_reset)
```

IPython Shell

```
In [1]:
```

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. You can easily do this with `.reset_index()`. However, you should still watch out, as you can pass several arguments that can make or break the success of your reset:

script.py | solution.py

```
# Check out the weird index of your dataframe
print(df)

# Use `reset_index()` to reset the values
df_reset = df.reset_index(level=0, drop=True)

# Print `df_reset`
print(df_reset)
```

IPython Shell

```
<script.py> output:
          48  49  50
   2.5     1   2   3
   12.6    4   5   6
   4.8     7   8   9
      48  49  50
   0   1   2   3
   1   4   5   6
   2   7   8   9

In [1]:
```

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

Now that you have seen how to select and add indices, rows, and columns to your DataFrame, it's time to consider another use case: removing these three from your data structure.

Deleting an Index from Your DataFrame

If you want to remove the index from your DataFrame, you should reconsider because DataFrames and Series always have an index.

However, what you *can* do is, for example:

- resetting the index of your DataFrame (go back to the previous section to see how it is done) or
- remove the index name, if there is any, by executing `del df.index.name`,
- remove duplicate index values by resetting the index, dropping the duplicates of the index column that has been added to your DataFrame and reinstating that duplicateless column again as the index:

script.py | solution.py | IPython Shell

```
df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [40, 50, 60], [23, 35, 37]]),
                  index= [2.5, 12.6, 4.8, 4.8, 2.5],
                  columns=[48, 49, 50])

df.reset_index().drop_duplicates(subset='index', keep='last').set_index('index')
```

C:\Users\Divya\panda_drop_duplicates.py

otlib.py | pandas_matplotib_single_fig.py | untitled38.py* | panda_create_dataframe_ex.py | panda_iloc_example.py | panda_drop_duplicates.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Feb  9 02:04:35 2021
4
5  @author: Divya
6  """
7  import pandas as pd
8  import numpy as np
9
10 df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5,
11                    index= [2.5, 12.6, 4.8, 4.8, 2.5]
12                    columns=[48, 49, 50])
13
14 df.reset_index().drop_duplicates(subset='index', ke
15
16 print(df)
```

Name	Type
a	int
arr	Array of
axes	Array of
b	int
baseball	list
bmi	Array of
c	Array of
cities	list

Help | Variable explorer | Plots

Console 1/A

```
In [94]: runfile('C:/Users/
panda_drop_duplicates.py',
Divya')
      48  49  50
2.5    1   2   3
12.6   4   5   6
4.8    7   8   9
4.8   40  50  60
2.5   23  35  37

In [95]:
```

Deleting a Column from Your DataFrame

To get rid of (a selection of) columns from your DataFrame, you can use the `drop()` method:

script.py | solution.py

```
# Check out the DataFrame `df`
print(df)

# Drop the column with label 'A'
df.drop('A', axis=1, inplace=True)

# Drop the column at position 1
df.drop(df.columns[[1]], axis=1)
```

IPython Shell

```
In [1]:
```

You might think now: well, this is not so straightforward; There are some extra arguments that are passed to the `drop()` method!

- The `axis` argument is either 0 when it indicates rows and 1 when it is used to drop columns.
- You can set `inplace` to True to delete the column without having to reassign the DataFrame.

script.py | solution.py

```
# Check out the DataFrame `df`
print(df)

# Drop the column with label 'A'
df.drop('A', axis=1, inplace=True)

# Drop the column at position 1
df.drop(df.columns[[1]], axis=1)
```

IPython Shell

```
<script.py> output:
      A  B  C
   0  1  2  3
   1  4  5  6
   2  7  8  9

In [1]:
```

Removing a Row from Your DataFrame

You can remove duplicate rows from your DataFrame by executing `df.drop_duplicates()`. You can also remove rows from your DataFrame, taking into account only the duplicate values that exist in one column.

Check out this example:

script.py | solution.py

```
1  # Check out your DataFrame `df`
2  print(df)
3
4  # Drop the duplicates in `df`
5  df.drop_duplicates([48], keep='last')
```

IPython Shell

```
<script.py> output:
          48  49  50  50
    2.5    1   2   3   4
    12.6   4   5   6   5
    4.8    7   8   9   6
    4.8   23  50  60   7
    2.5   23  35  37  23

In [1]:
```

If there is no uniqueness criterion to the deletion that you want to perform, you can use the `drop()` method, where you use the `index` property to specify the index of which rows you want to remove from your DataFrame:

script.py | solution.py

```
# Check out the DataFrame `df`
print(df)

# Drop the index at position 1
print(df.drop(df.index[1]))
```

IPython Shell

```
<script.py> output:
      A  B  C
   0  1  2  3
   1  4  5  6
   2  7  8  9
      A  B  C
   0  1  2  3
   2  7  8  9

In [1]:
```

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

To give the columns or your index values of your dataframe a different value, it's best to use the `.rename()` method.

script.py | solution.py

```
# Check out your DataFrame `df`
print(df)

# Define the new names of your columns
newcols = {
    'A': 'new_column_1',
    'B': 'new_column_2',
    'C': 'new_column_3'
}

# Use `rename()` to rename your columns
df.rename(columns=newcols, inplace=True)

# Rename your index
df.rename(index={1: 'a'})
```

IPython Shell

```
In [1]:
```

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

To give the columns or your index values of your dataframe a different value, it's best to use the `.rename()` method.

script.py | solution.py

```
# Check out your DataFrame `df`
print(df)

# Define the new names of your columns
newcols = {
    'A': 'new_column_1',
    'B': 'new_column_2',
    'C': 'new_column_3'
}

# Use `rename()` to rename your columns
df.rename(columns=newcols, inplace=True)

# Rename your index
df.rename(index={1: 'a'})
```

IPython Shell

```
<script.py> output:
      A  B  C
   0  1  2  3
   1  4  5  6
   2  7  8  9

In [1]:
```

6. How To Format The Data in Your Pandas DataFrame

Most of the times, you will also want to be able to do some operations on the actual values that are contained within your DataFrame. In the following sections, you'll cover several ways in which you can format 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.

Just like this:

script.py

```
# Study the DataFrame `df` first
_____(df)

# Replace the strings by numerical values (0-4)
df._______(['Awful', 'Poor', 'OK', 'Acceptable',
'Perfect'], [0, 1, 2, 3, 4])
```

IPython Shell

```
In [1]:
```

7. How To Create an Empty DataFrame

The function that you will use is the Pandas `Dataframe()` function: it requires you to pass the data that you want to put in, the indices and the columns.

Remember that the data that is contained within the data frame doesn't have to be homogenous. It can be of different data types!

There are several ways in which you can use this function to make an empty DataFrame. Firstly, you can use `numpy.nan` to initialize your data frame with `NaN`s. Note that `numpy.nan` has type `float`.

script.py

```
1  df = pd.DataFrame(np.nan, index=[0,1,2,3], columns=['A'])
2  print(df)
```

IPython Shell

```
<script.py> output:
     A
0  NaN
1  NaN
2  NaN
3  NaN
```

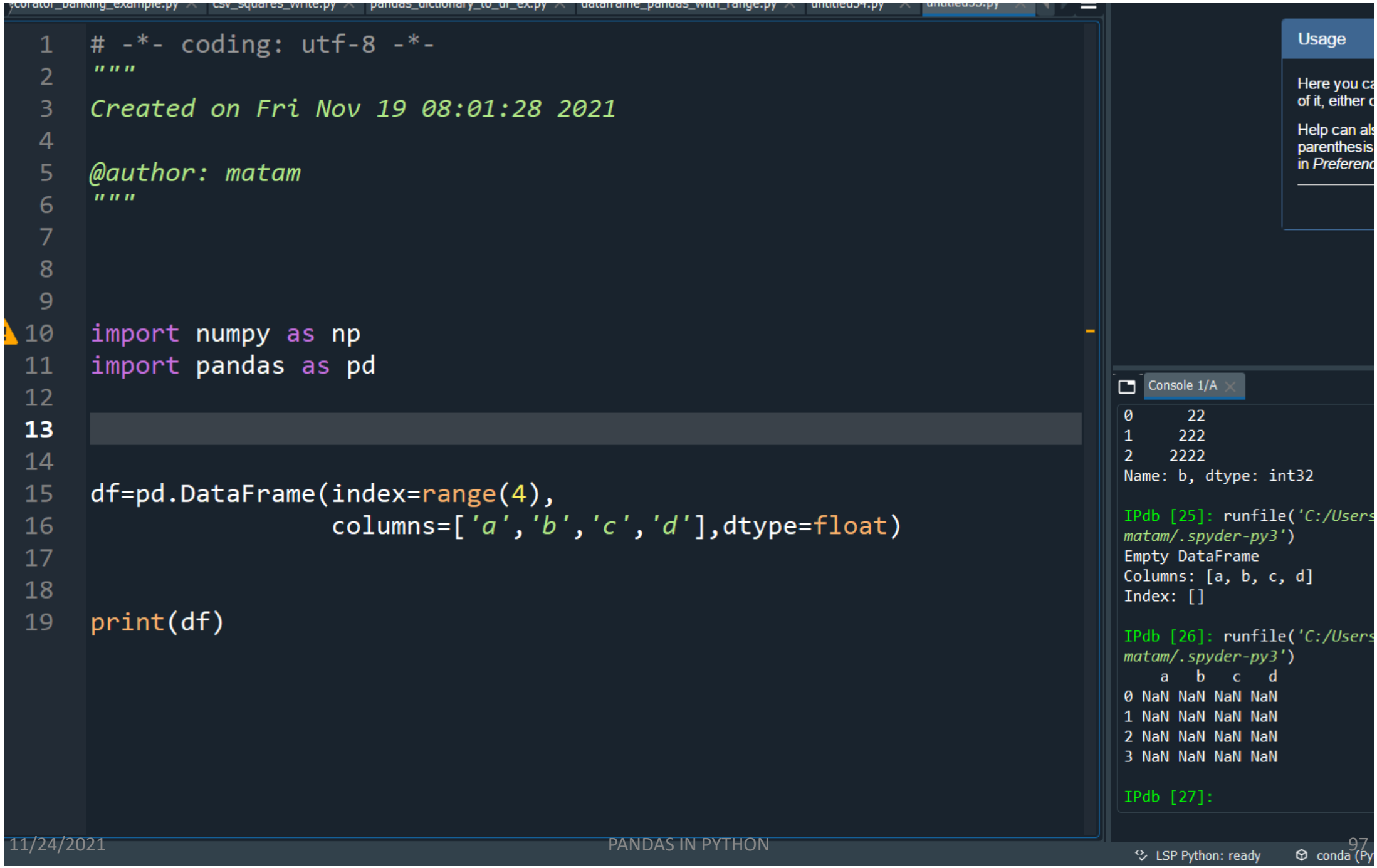
Right now, the data type of the data frame is inferred by default: because `numpy.nan` has type float, the data frame will also contain values of type float. You can, however, also force the DataFrame to be of a particular type by adding the attribute `dtype` and filling in the desired type. Just like in this example:

script.py

```
1  df = pd.DataFrame(index=range(0,4),columns=['A'], dtype
   ='float')
2  print(df)
```

IPython Shell

```
<script.py> output:
        A
    0 NaN
    1 NaN
    2 NaN
    3 NaN

In [1]:
```


1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Nov 19 08:01:28 2021
4
5 @author: matam
6 """
7
8
9
10 import numpy as np
11 import pandas as pd
12
13
14
15 df=pd.DataFrame(index=range(4),
16 columns=['a','b','c','d'],dtype=float)
17
18
19 print(df)
Usage
Here you ca
of it, either
Help can als
parenthesis
in Preferenc
Console 1/A
0 22
1 222
2 2222
Name: b, dtype: int32
IPdb [25]: runfile('C:/Users
matam/.spyder-py3')
Empty DataFrame
Columns: [a, b, c, d]
Index: []
IPdb [26]: runfile('C:/Users
matam/.spyder-py3')
a b c d
0 NaN NaN NaN NaN
1 NaN NaN NaN NaN
2 NaN NaN NaN NaN
3 NaN NaN NaN NaN
IPdb [27]:
LSP Python: ready
conda (Py

8. Does Pandas Recognize Dates When Importing Data?

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:

```
import pandas as pd
pd.read_csv('yourFile', parse_dates=True)


# or this option:
pd.read_csv('yourFile', parse_dates=['columnName'])
```

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

No worries! 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.

script.py | solution.py

```
# Read the dataset 'college.csv'
college = pd.read_csv('college.csv')
print(college.head())

# Print the info of college
print(college.info())

# Store unique values of 'csat' column to 'csat_unique'
csat_unique = college.csat.unique()

# Print the sorted values of csat_unique
print(np.sort(csat_unique))
```

```
import pandas as pd
dateparser = lambda x: pd.datetime.strptime(x, '%Y-%m-%d %H:%M:%S')

# Which makes your read command:
pd.read_csv(infile, parse_dates=['columnName'], date_parser=dateparse)

# Or combine two columns into a single DateTime column
pd.read_csv(infile, parse_dates={'datetime': ['date', 'time']}, date_parser=dateparse)
```

Replacing hidden missing values

script.py | solution.py

```
# Print the description of the data
print(diabetes.describe())
```

IPython Shell | Slides

```
       pregnancies     glucose  diastolic  triceps      insulin         bmi  \
mean      3.845052  121.686763  72.405184  29.153420   155.548223   31.992578
std       3.369578   30.535641  12.382158  10.476982   118.775855    7.884160
min       0.000000   44.000000  24.000000   7.000000    14.000000    0.000000
25%       1.000000   99.000000  64.000000  22.000000    76.250000   27.300000
50%       3.000000  117.000000  72.000000  29.000000   125.000000   32.000000
75%       6.000000  141.000000  80.000000  36.000000   190.000000   36.600000
max      17.000000  199.000000 122.000000  99.000000   846.000000   67.100000

       dpf_score         age
mean    0.471876   33.240885
std     0.331329   11.760232
min     0.078000   21.000000
25%     0.243750   24.000000
50%     0.372500   29.000000
75%     0.626250   41.000000
max     2.420000   81.000000

In [1]:
```

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

Reshaping your DataFrame is 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. But how would you reshape your DataFrame?

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

Pivotting 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:

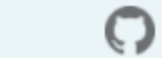
1. `values`: this argument allows you to specify which values of your original DataFrame you want to see in your pivot table.
2. `columns`: whatever you pass to this argument will become a column in your resulting table.
3. `index`: whatever you pass to this argument will become an index in your resulting table.

script.py | solution.py

```
# Import pandas
import pandas as pd

# Create your DataFrame
products = pd.DataFrame({'category': ['Cleaning', 'Cleaning', 'Entertainment', 'Entertainment', 'Tech', 'Tech'],
        'store': ['Walmart', 'Dia', 'Walmart', 'Fnac', 'Dia','Walmart'],
        'price':[11.42, 23.50, 19.99, 15.95, 55.75, 111.55],
        'testscore': [4, 3, 5, 7, 5, 8]})

# Use `pivot()` to pivot the DataFrame
pivot_products = products.pivot(index='category', columns='store', values='price')

# Check out the result
print(pivot_products)
```

IPython Shell

```
In [1]:
```



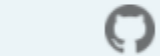
When you don't specifically fill in what values you expect to be present in your resulting table, you will pivot by multiple columns:

script.py

```
# Import the Pandas library
import ______ as pd

# Construct the DataFrame
products = pd.DataFrame({'category': ['Cleaning',
'Cleaning', 'Entertainment', 'Entertainment', 'Tech',
'Tech'],
                         'store': ['Walmart', 'Dia',
'Walmart', 'Fnac', 'Dia','Walmart'],
                         'price':[11.42, 23.50, 19.99, 15
.95, 55.75, 111.55],
                         'testscore': [4, 3, 5, 7, 5, 8]})

# Use `pivot()` to pivot your DataFrame
pivot_products = products._____(index='category', columns
='store')

# Check out the results
print(pivot_products)
```

IPython Shell

```
In [1]:
```



When you don't specifically fill in what values you expect to be present in your resulting table, you will pivot by multiple columns:

script.py | solution.py

```
# Import the Pandas library
import pandas as pd

# Construct the DataFrame
products = pd.DataFrame({'category': ['Cleaning', 'Cleaning', 'Entertainment', 'Entertainment', 'Tech', 'Tech'],
                         'store': ['Walmart', 'Dia', 'Walmart', 'Fnac', 'Dia','Walmart'],
                         'price':[11.42, 23.50, 19.99, 15.95, 55.75, 111.55],
                         'testscore': [4, 3, 5, 7, 5, 8]})

# Use `pivot()` to pivot your DataFrame
pivot_products = products.pivot(index='category', columns='store')

# Check out the results
print(pivot_products)
```

IPython Shell

```
In [1]:
```

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:

script.py

```
# Import the Pandas library
import ______ as pd

# Your DataFrame
products = pd.DataFrame({'category': ['Cleaning', 'Cleaning', 'Entertainment', 'Entertainment', 'Tech', 'Tech'],
                         'store': ['Walmart', 'Dia', 'Walmart', 'Fnac', 'Dia','Walmart'],
                         'price':[11.42, 23.50, 19.99, 15.95, 19.99, 111.55],
                         'testscore': [4, 3, 5, 7, 5, 8]})

# Pivot your `products` DataFrame with `pivot_table()`
pivot_products = products.__________(index='category', columns='store', values='price', aggfunc='mean')

# Check out the results
print(pivot_products)
```

IPython Shell

```
In [1]:
```

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:

script.py | solution.py

```
# Import the Pandas library
import pandas as pd

# Your DataFrame
products = pd.DataFrame({'category': ['Cleaning', 'Cleaning', 'Entertainment', 'Entertainment', 'Tech', 'Tech'],
                         'store': ['Walmart', 'Dia', 'Walmart', 'Fnac', 'Dia','Walmart'],
                         'price':[11.42, 23.50, 19.99, 15.95, 19.99, 111.55],
                         'testscore': [4, 3, 5, 7, 5, 8]})

# Pivot your `products` DataFrame with `pivot_table()`
pivot_products = products.pivot_table(index='category', columns='store', values='price', aggfunc='mean')

# Check out the results
print(pivot_products)
```

IPython Shell

```
In [1]:
```

```python
import pandas as pd


data ={
       "category":
         ['mobile','tv','laptop','gadget','hardware','hardware','tech','tech'],
       'store':['filpkart','flipkart','amazon','CLIQ','alibaba','alibaba','croma','croma'],
       'price':[10000,80000,100000,20000,5000,6000,9000,10000]}


df=pd.DataFrame(data)

print(df)

data_pivot=df.pivot_table(index="category",columns='store',aggfunc='mean',fill_value=0)

print(data_pivot)
```

Variable explorer | Help | Plots | Files

Console 1/A

```
   category     store   price
0    mobile  filpkart   10000
1        tv  flipkart   80000
2    laptop    amazon  100000
3    gadget      CLIQ   20000
4  hardware   alibaba    5000
5  hardware   alibaba    6000
6      tech     croma    9000
7      tech     croma   10000
           price
store       CLIQ alibaba  amazon croma filpkart flipkart
category
gadget     20000       0       0     0        0        0
hardware       0    5500       0     0        0        0
laptop         0       0  100000     0        0        0
mobile         0       0       0     0    10000        0
tech           0       0       0  9500        0        0
tv             0       0       0     0        0    80000

IPdb [187]:
```

script.py | solution.py

```
1   # Import the Pandas library
2   import pandas as pd
3
4   # Your DataFrame
5   products = pd.DataFrame({'category': ['Cleaning',
    'Cleaning', 'Entertainment', 'Entertainment', 'Tech',
    'Tech'],
6                            'store': ['Walmart', 'Dia',
    'Walmart', 'Fnac', 'Dia','Walmart'],
7                            'price':[11.42, 23.50, 19.99, 15
    .95, 19.99, 111.55],
8                            'testscore': [4, 3, 5, 7, 5, 8]})
9
10  # Pivot your `products` DataFrame with `pivot_table()`
11  pivot_products = products.pivot_table(index='category',
    columns='store', values='price', aggfunc='mean')
12
13  # Check out the results
14  print(pivot_products)
```

IPython Shell

```
<script.py> output:
    store            Dia   Fnac  Walmart
    category
    Cleaning       23.50    NaN    11.42
    Entertainment    NaN  15.95    19.99
    Tech           19.99    NaN   111.55

In [1]:
```

Stacking DataFrames

RESHAPING DATA WITH PANDAS

Row multi-indices

		height	weight
Last	First		
Wick	John	185	68
	Julien	164	61
Shelley	Mary	164	59
	Frank	155	58

Setting the index

```
churn.set_index(['country', 'age'], inplace=True)
```

```
             credit_score num_products exited
age  country
 43   France           619            1    Yes
 34  Germany           608            0     No
 23   France           502            1    Yes
```

MultiIndex DataFrames

		2019		2020	
		height	weight	height	weight
Last	First				
Wick	John	185	68	185	70
	Julien	164	61	164	60
Shelley	Mary	164	59	164	60
	Frank	155	65	155	58

MultiIndex DataFrames

```
index = pd.MultiIndex.from_arrays([['Wick', 'Wick', 'Shelley', 'Shelley'],
                                   ['John', 'Julien', 'Mary', 'Frank']],
                          names=['last', 'first'])
columns = pd.MultiIndex.from_arrays([['2019', '2019', '2020', '2020'],
                                     ['age', 'weight', 'age', 'weight']],
                            names=['year', 'feature'])
```

MultiIndex DataFrames

```
index = pd.MultiIndex.from_arrays([['Wick', 'Wick', 'Shelley', 'Shelley'],
                                   ['John', 'Julien', 'Mary', 'Frank']],
                          names=['last', 'first'])
columns = pd.MultiIndex.from_arrays([['2019', '2019', '2020', '2020'],
                                     ['age', 'weight', 'age', 'weight']],
                            names=['year', 'feature'])
patients = pd.DataFrame(data, index=index, columns=columns)
patients
```

year		2019		2020	
feature		age	weight	age	weight
last	first				
Wick	John	25	68	26	72
	Julien	31	72	32	73
Shelley	Mary	41	68	42	69
	Frank	32	75	33	74

The .stack() method

		height	weight
Last	First		
Wick	John	185	68
	Julien	164	61
Shelley	Mary	164	59
	Frank	155	58

Last	First		
Wick	John	height	185
		weight	68
	Julien	height	164
		weight	61
Shelley	Mary	height	164
		weight	59
	Frank	height	155
		weight	58

df.stack()

The .stack() method

> Rearrange a level of the columns to obtain a reshaped DataFrame with a new inner-most level row index

		height	weight
Last	First		
Wick	John	185	68
	Julien	164	61
Shelley	Mary	164	59
	Frank	155	58

Last	First		
Wick	John	height	185
		weight	68
	Julien	height	164
		weight	61
Shelley	Mary	height	164
		weight	59
	Frank	height	155
		weight	58

Stack into a series

```
churn
```

	credit_score	age	country	num_products	exited
0	619	43	France	1	Yes
1	608	34	Germany	0	No
2	502	23	France	1	Yes

```
churned_stacked = churn.stack()
churned_stacked.head(10)
```

member	credit_card		
yes	no	credit_score	619
		age	43
		country	France
		num_products	1
		churn	Yes
no	yes	credit_score	608
		age	34
		country	Germany
		num_products	0
		churn	No

Stack into a DataFrame

```
patients
```

		year	2019		2020	
		feature	age	weight	age	weight
last	first					
Wick	John		25	68	26	72
	Julien		31	72	32	73
Shelley	Mary		41	68	42	69
	Frank		32	75	33	74

```
patients_stacked = patients.stack()
patients_stacked
```

		year	2019	2020
last	first	feature		
Wick	John	age	25	26
		weight	68	72
	Julien	age	31	32
		weight	72	73
Shelley	Mary	age	41	42
		weight	68	69
	Frank	age	32	33
		weight	75	74

Unstack Series

```
churn_stacked
```

```
member  credit_card
yes     no           credit_score       619
                     age                 43
                     country          France
                     num_products          1
                     churn               Yes
no      yes          credit_score       608
                     age                 34
                     country         Germany
                     num_products          0
                     churn                No
yes     yes          credit_score       502
                     age                 23
                     country          France
                     num_products          1
                     churn               Yes
```

Unstack Series

```
churned_stacked.unstack()
```

```
                   credit_score age  country  num_products exited
member credit_card
    no         yes           608  34  Germany             0     No
   yes          no           619  43   France             1    Yes
                yes           502  23   France             1    Yes
```

Unstacking a DataFrame

```
patients_stacked
```

```
                 year  2019 2020
  first   last feature
   Wick   John     age    25   26
                weight    68   72
         Julien     age    31   32
                weight    72   73
Shelley   Mary     age    41   42
                weight    68   69
          Frank     age    32   33
                weight    75   74
```

Unstack a level

Last	First		
Johnson	Louis	age	32
		weight	68
	Mary	age	42
		weight	61
Smith	Louis	age	20
		weight	59
	Mary	age	32
		weight	58

	First	Louis	Mary
Last			
Johnson	age	32	42
	weight	68	61
Smith	age	20	32
	weight	59	58

df.unstack(level=1) or df.unstack(level='First')

Unstack level by number

```
churn_stacked.head(10)
```

```
member  credit_card
yes     no           credit_score        619
                     age                  43
                     country          France
                     num_products          1
                     churn               Yes
no      yes          credit_score        608
                     age                  34
                     country         Germany
                     num_products          0
                     churn                No
```

```
churn_stacked.unstack(level=0)
```

```
                    member     no    yes
credit_card
         no credit_score     NaN    619
                     age     NaN     43
                 country     NaN France
            num_products     NaN      1
                   churn     NaN    Yes
        yes credit_score     608    502
                     age      34     23
                 country Germany France
            num_products       0      1
                   churn      No    Yes
```

Unstack level by name

```
churn_stacked.head(10)
```

```
member  credit_card
yes     no           credit_score        619
                     age                  43
                     country          France
                     num_products          1
                     churn               Yes
no      yes          credit_score        608
                     age                  34
                     country         Germany
                     num_products          0
                     churn                No
```

```
churn_stacked.unstack(level='credit_card')
```

```
                credit_card      no     yes
member
    no  credit_score     NaN     608
                age     NaN      34
            country     NaN Germany
        num_products     NaN       0
              churn     NaN      No
    yes credit_score     619     NaN
                age      43     NaN
            country  France     NaN
        num_products       1     NaN
              churn     Yes     NaN
```

Sort index

```
patients_stacked.unstack().sort_index(ascending=False)
```

	year	2019		2020	
	feature	age	weight	age	weight
last	first				
Wick	Julien	31	72	32	73
	John	25	68	26	72
Shelley	Mary	41	68	42	69
	Frank	32	75	33	74

Rearranging levels

```
patients_stacked
```

```
                year  2019 2020
  first   last feature
   Wick   John     age    25   26
                weight    68   72
         Julien     age    31   32
                weight    72   73
Shelley   Mary     age    41   42
                weight    68   69
          Frank     age    32   33
                weight    75   74
```

```
patients_stacked.unstack(level=1).stack(level=0)
```

```
first                Frank  John  Julien  Mary
 last   feature year
Shelley age     2019   32.0   NaN     NaN  41.0
                2020   33.0   NaN     NaN  42.0
        weight  2019   75.0   NaN     NaN  68.0
                2020   74.0   NaN     NaN  69.0
Wick    age     2019    NaN  25.0    31.0   NaN
                2020    NaN  26.0    32.0   NaN
        weight  2019    NaN  68.0    72.0   NaN
                2020    NaN  72.0    73.0   NaN
```

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\matam\.spyder-py3

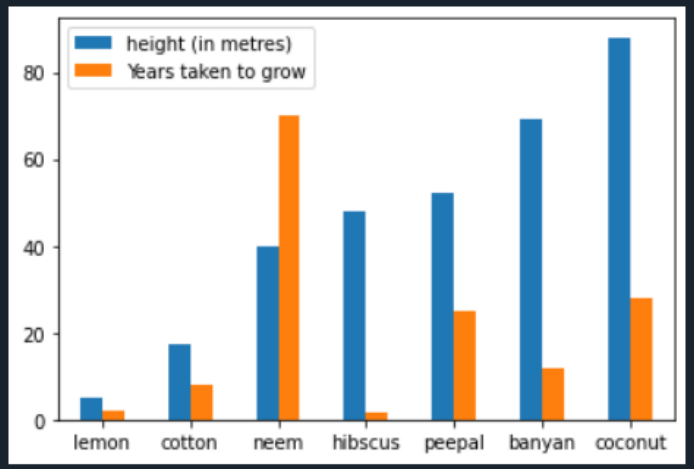
C:\Users\matam\.spyder-py3\untitled83.py

pandas_pivot_multiple_columns.py | untitled79.py* | pivot_table_duplicate_index.py | untitled81.py* | pandas_stacking_unstacking_level.py | untitled83.py*

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Nov 2
4
5  @author: matam
6  """
7
8  import pandas as pd
9
10
11 height = [5, 17.5, 40, 48, 52, 69, 88]
12
13 years_to_fully_grow = [2, 8, 70, 1.5, 25, 12, 28]
14
15 index = ['lemon', 'cotton', 'neem',
16
17         'hibscus', 'peepal', 'banyan', 'coconut']
18
19 df = pd.DataFrame({'height (in metres)': height,
20                    'Years taken to grow': years_to_fully_grow}, index=index)
21
22
23 print(df)
24
25 ax = df.plot.bar(rot=0)
```

No documentation available

Click anywhere in this tooltip for additional help



126 %

Variable explorer | Help | Plots | Files

Console 1/A

```
Cathrine Semester1     45       67
         Semester2     89       90
Jack     Semester1     67       44
         Semester2     56       55

IPdb [193]: runfile('C:/Users/matam/.spyder-py3/untitled83.py', wd
Users/matam/.spyder-py3')

IPdb [194]: runfile('C:/Users/matam/.spyder-py3/untitled83.py', wd
Users/matam/.spyder-py3')
         height (in metres)  Years taken to grow
lemon                   5.0                  2.0
cotton                 17.5                  8.0
neem                   40.0                 70.0
hibscus                48.0                  1.5
peepal                 52.0                 25.0
banyan                 69.0                 12.0
coconut                88.0                 28.0

IPdb [195]:
```

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\matam\.spyder-py3

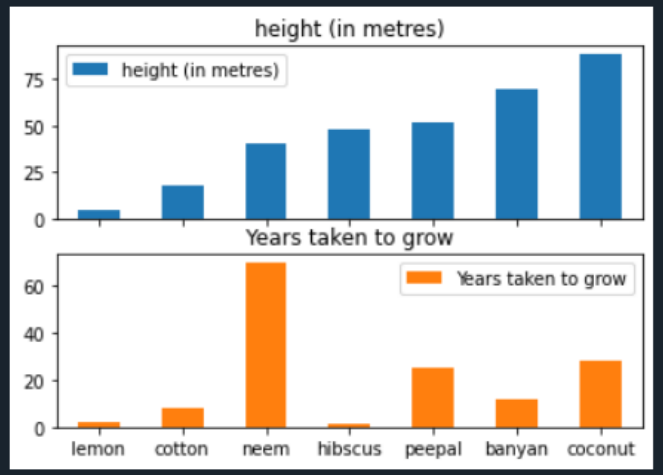
C:\Users\matam\.spyder-py3\untitled83.py

pandas_pivot_multiple_columns.py | untitled79.py* | pivot_table_duplicate_index.py | untitled81.py* | pandas_stacking_unstacking_level.py | untitled83.py*

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Nov
4
5  @author: matam
6  """
7
8  import pandas as pd
9
10
11 height = [5, 17.5, 40, 48, 52, 69, 88]
12
13 years_to_fully_grow = [2, 8, 70, 1.5, 25, 12, 28]
14
15 index = ['lemon', 'cotton', 'neem',
16
17         'hibscus', 'peepal', 'banyan', 'coconut']
18
19 df = pd.DataFrame({'height (in metres)': height,
20                    'Years taken to grow': years_to_fully_grow}, index=index)
21
22
23 print(df)
24
25 ax = df.plot.bar(rot=0)
26
27 axes = df.plot.bar(rot=0, subplots=True)
```

No documentation available

Click anywhere in this tooltip for additional help



Variable explorer | Help | Plots | Files

Console 1/A

```
lemon                5.0                   2.0
cotton              17.5                   8.0
neem                40.0                  70.0
hibscus             48.0                   1.5
peepal              52.0                  25.0
banyan              69.0                  12.0
coconut             88.0                  28.0

IPdb [195]: runfile('C:/Users/matam/.spyder-py3/untitled83.py',
Users/matam/.spyder-py3')
         height (in metres)  Years taken to grow
lemon                   5.0                   2.0
cotton                 17.5                   8.0
neem                   40.0                  70.0
hibscus                48.0                   1.5
peepal                 52.0                  25.0
banyan                 69.0                  12.0
coconut                88.0                  28.0
```

Reshape Your DataFrame With `melt()`

Melting is considered useful in cases where you have 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 have two non-identifier columns, namely, 'variable' and 'value'.

Let's illustrate this with an example:

script.py

```
# The `people` DataFrame
people = pd.DataFrame({'FirstName' : ['John', 'Jane'],
                       'LastName' : ['Doe', 'Austen'],
                       'BloodType' : ['A-', 'B+'],
                       'Weight' : [90, 64]})

# Use `melt()` on the `people` DataFrame
print(pd.____(people, id_vars=['FirstName', 'LastName'],
var_name='measurements'))
```

IPython Shell

```
In [1]:
```

Let's illustrate this with an example:

script.py | solution.py

```
# The `people` DataFrame
people = pd.DataFrame({'FirstName' : ['John', 'Jane'],
                       'LastName' : ['Doe', 'Austen'],
                       'BloodType' : ['A-', 'B+'],
                       'Weight' : [90, 64]})

# Use `melt()` on the `people` DataFrame
print(pd.melt(people, id_vars=['FirstName', 'LastName'],
var_name='measurements'))
```

IPython Shell

```
In [1]:
```

Reshaping data

- Transforming a DataFrame or Series structure to adjust it for analysis
  - Transposing a DataFrame

```
fifa_players.set_index('club')[['name', 'nationality']].transpose()
```

```
club         Barcelona          Juventus     Saint-Germain
name       Lionel Messi  Cristiano Ronaldo   Neymar da Silva
nationality   Argentina           Portugal            Brazil
```

Pivot method

	Name	Year	Weight
0	John	2013	80
1	Mary	2013	65
2	Mary	2014	68
3	John	2014	83
4	Laura	2014	71

Name	John	Mary	Laura
Year			
2013	80	65	NaN
2014	83	68	71

Pivot method

	Name	Year	Weight
0	John	2013	80
1	Mary	2013	65
2	Mary	2014	68
3	John	2014	83
4	Laura	2014	71

Name	John	Mary	Laura
Year			
2013	80	65	NaN
2014	83	68	71

```
df.pivot(index="Year", columns="Name", values="Weight")
```

Pivoting multiple columns

```
fifa.pivot(index='name', columns='variable', values=['metric_system', 'imperial_system'])
```

	metric_system		imperial_system	
variable	height	weight	height	weight
name				
Cristiano Ronaldo	187	83	6.13	183.0
J. Oblak	188	87	6.16	191.0

Pivoting multiple columns

	Name	Year	Weight	Age
0	John	2013	80	30
1	Mary	2013	65	28
2	Mary	2014	68	29
3	John	2014	83	31
4	Laura	2014	71	34

	Weight			Age		
Name	John	Mary	Laura	John	Mary	Laura
Year						
2013	80	65	NaN	30	28	NaN
2014	83	68	71	31	29	34

```
df.pivot(index="Year", columns="Name")
```

Duplicate entries error

```
another_fifa.pivot(index="name", columns="variable")
```

```
ValueError: Index contains duplicate entries, cannot reshape
```

```
another_fifa = another_fifa.drop(4, axis=0)
another_fifa.pivot(index="name", columns="variable")
```

script.py | solution.py

```
1  # Drop the fifth row to delete all repeated rows
2  fifa_no_rep = fifa_players.drop(4, axis=0)
3
4  # Print fifa_pivot
5  print(fifa_no_rep)
```

Pivot table

	Name	Year	Weight
0	John	2013	80
1	John	2013	81
2	Mary	2013	67
3	Mary	2013	66
4	John	2014	82
5	John	2014	84
6	Mary	2014	69
7	Mary	2014	67

Name	John	Mary
Year		
2013	80.5	66.5
2014	83	68

Pivot table

	Name	Year	Weight
0	John	2013	80
1	John	2013	81
2	Mary	2013	67
3	Mary	2013	66
4	John	2014	82
5	John	2014	84
6	Mary	2014	69
7	Mary	2014	67

Name	John	Mary
Year		
2013	80.5	66.5
2014	83	68

```
df.pivot_table(index="Year", columns="Name", values="Weight" , aggfunc="mean" )
```

Pivot or pivot table?

Does the DataFrame have more than one value for each index/column pair?

Do you need to have a multi-index in your resulting pivoted DataFrame?

Do you need summary statistics of your large DataFrame?

Yes! Use `.pivot_table()`

Melt

	first	last	age	height	weight
0	John	Wick	50	185	70
1	Mary	Shelley	25	164	60
2	Alice	Liddell	16	155	58

	first	last	variable	value
0	John	Wick	age	50
1	Mary	Shelley	age	25
2	Alice	Liddell	age	16
3	John	Wick	height	185
4	Mary	Shelley	height	164
5	Alice	Liddell	height	155
6	John	Wick	weight	70
7	Mary	Shelley	weight	60
8	Alice	Liddell	weight	58

```
df.melt(id_vars=["first", "last"])
```

Values and variables

	first	last	age	height	weight
0	John	Wick	50	185	70
1	Mary	Shelley	25	164	60
2	Alice	Liddell	16	155	58

	first	last	feature	amount
0	John	Wick	age	50
1	Mary	Shelley	age	25
2	Alice	Liddell	age	16
3	John	Wick	height	185
4	Mary	Shelley	height	164
5	Alice	Liddell	height	155

```
df.melt(id_vars=["first","last"], value_vars=["age","height"], var_name="feature", value_name="amount")
```

Specifying values to melt

```
books.melt(id_vars='title', value_vars=['language_code', 'num_pages'])
```

	title	variable	value
0	Mostly Harmless	language	eng
1	The Hitchhiker's Guide	language	eng
2	El restaurante del fin del mundo	language	spa
3	Mostly Harmless	pages	260
4	The Hitchhiker's Guide	pages	215
5	El restaurante del fin del mundo	pages	250

Naming values and variables

```
books.melt(id_vars='title', value_vars=['language_code', 'isbn'], var_name='feature', value_name='code')
```

```
                          title   feature    code
0                 Mostly Harmless      isbn     074
1           The Hitchhiker's Guide      isbn     072
2  El restaurante del fin del mundo      isbn     071
3                 Mostly Harmless  language     eng
4           The Hitchhiker's Guide  language     eng
5  El restaurante del fin del mundo  language     spa
```

10. How To Iterate Over a Pandas DataFrame

You can iterate over the rows of your DataFrame with the help of a `for` loop in combination with an `iterrows()` call on your DataFrame:

script.py

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

for index, row in df.iterrows() :
    print(row['A'], row['B'])
```

IPython Shell

```
In [1]:
```

10. How To Iterate Over a Pandas DataFrame

You can iterate over the rows of your DataFrame with the help of a `for` loop in combination with an `iterrows()` call on your DataFrame:

script.py | solution.py

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

for index, row in df.iterrows() :
    print(row['A'], row['B'])
```

IPython Shell

```
<script.py> output:
   1 2
   4 5
   7 8

In [1]:
```

11. 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.

Output a DataFrame to CSV

To write a DataFrame as a CSV file, you can use `to_csv()`:

```
import pandas as pd

df.to_csv('myDataFrame.csv')
```

That piece of code seems quite simple, but this is just where the difficulties begin for most people because you will have specific requirements for the output of your data. Maybe you don't want a comma as a delimiter, or you want to specify a specific encoding, ...

Don't worry! You can pass some additional arguments to `to_csv()` to make sure that your data is outputted the way you want it to be!

- To delimit by a tab, use the `sep` argument:

```
import pandas as pd
df.to_csv('myDataFrame.csv', sep='\t')
```

- To use a specific character encoding, you can use the `encoding` argument:

```
import pandas as pd
df.to_csv('myDataFrame.csv', sep='\t', encoding='utf-8')
```

- Furthermore, you can specify how you want your `NaN` or missing values to be represented, whether or not you want to output the header, whether or not you want to write out the row

Writing a DataFrame to Excel

Similarly to what you did to output your DataFrame to CSV, you can use `to_excel()` to write your table to Excel. However, it is a bit more complicated:

```
import pandas as pd
writer = pd.ExcelWriter('myDataFrame.xlsx')
df.to_excel(writer, 'DataFrame')
writer.save()
```

Building DataFrames from scratch

DataFrames from dict (1)

```
import pandas as pd
data = {'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],
            'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],
            'visitors': [139, 237, 326, 456],
            'signups': [7, 12, 3, 5]}
users = pd.DataFrame(data)
print(users)
```

```
 weekday    city  visitors  signups
0     Sun  Austin       139        7
1     Sun  Dallas       237       12
2     Mon  Austin       326        3
3     Mon  Dallas       456        5
```

Building DataFrames from scratch

DataFrames from dict (2)

```
import pandas as pd
cities = ['Austin', 'Dallas', 'Austin', 'Dallas']
signups = [7, 12, 3, 5]
visitors = [139, 237, 326, 456]
weekdays = ['Sun', 'Sun', 'Mon', 'Mon']
list_labels = ['city', 'signups', 'visitors', 'weekday']
list_cols = [cities, signups, visitors, weekdays]
zipped = list(zip(list_labels, list_cols))
```

Building DataFrames from scratch

DataFrames from dict (3)

```
print(zipped)
```

```
[('city', ['Austin', 'Dallas', 'Austin', 'Dallas']),
('signups', [7, 12, 3, 5]),
('visitors', [139, 237, 326, 456]),
('weekday', ['Sun', 'Sun', 'Mon', 'Mon'])]
```

```
data = dict(zipped)
users = pd.DataFrame(data)
print(users)
```

```
  weekday    city  visitors  signups
0     Sun  Austin       139        7
1     Sun  Dallas       237       12
2     Mon  Austin       326        3
3     Mon  Dallas       456        5
```

Broadcasting

```
users['fees'] = 0 # Broadcasts to entire column
print(users)
```

```
     city  signups  visitors weekday  fees
0  Austin        7       139     Sun     0
1  Dallas       12       237     Sun     0
2  Austin        3       326     Mon     0
3  Dallas        5       456     Mon     0
```

Building DataFrames from scratch

Broadcasting with a dict

```
import pandas as pd
heights = [ 59.0, 65.2, 62.9, 65.4, 63.7, 65.7, 64.1 ]
data = {'height': heights, 'sex': 'M'}
results = pd.DataFrame(data)
print(results)
```

```
   height sex
0    59.0   M
1    65.2   M
2    62.9   M
3    65.4   M
4    63.7   M
5    65.7   M
6    64.1   M
```

Building DataFrames from scratch

Index and columns

```
results.columns = ['height (in)', 'sex']
results.index = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
print(results)
```

```
   height (in) sex
A         59.0   M
B         65.2   M
C         62.9   M
D         65.4   M
E         63.7   M
F         65.7   M
G         64.1   M
```

- Zip the 2 lists list_keys and list_values together into one list of (key, value) tuples. Be sure to convert the zip object into a list, and store the result in zipped.
- Inspect the contents of zipped using print(). This has been done for you.
- Construct a dictionary using zipped. Store the result as data.
- Construct a DataFrame using the dictionary. Store the result as df.

script.py

```
# Zip the 2 lists together into one list of (key,value) tuples: zipped
zipped = list(zip(list_keys,list_values))

# Inspect the list using print()
print(zipped)

# Build a dictionary with the zipped list: data
data = dict(zipped)

# Build and inspect a DataFrame from the dictionary: df
df = pd.DataFrame(data)
print(df)
```

Building DataFrames with broadcasting

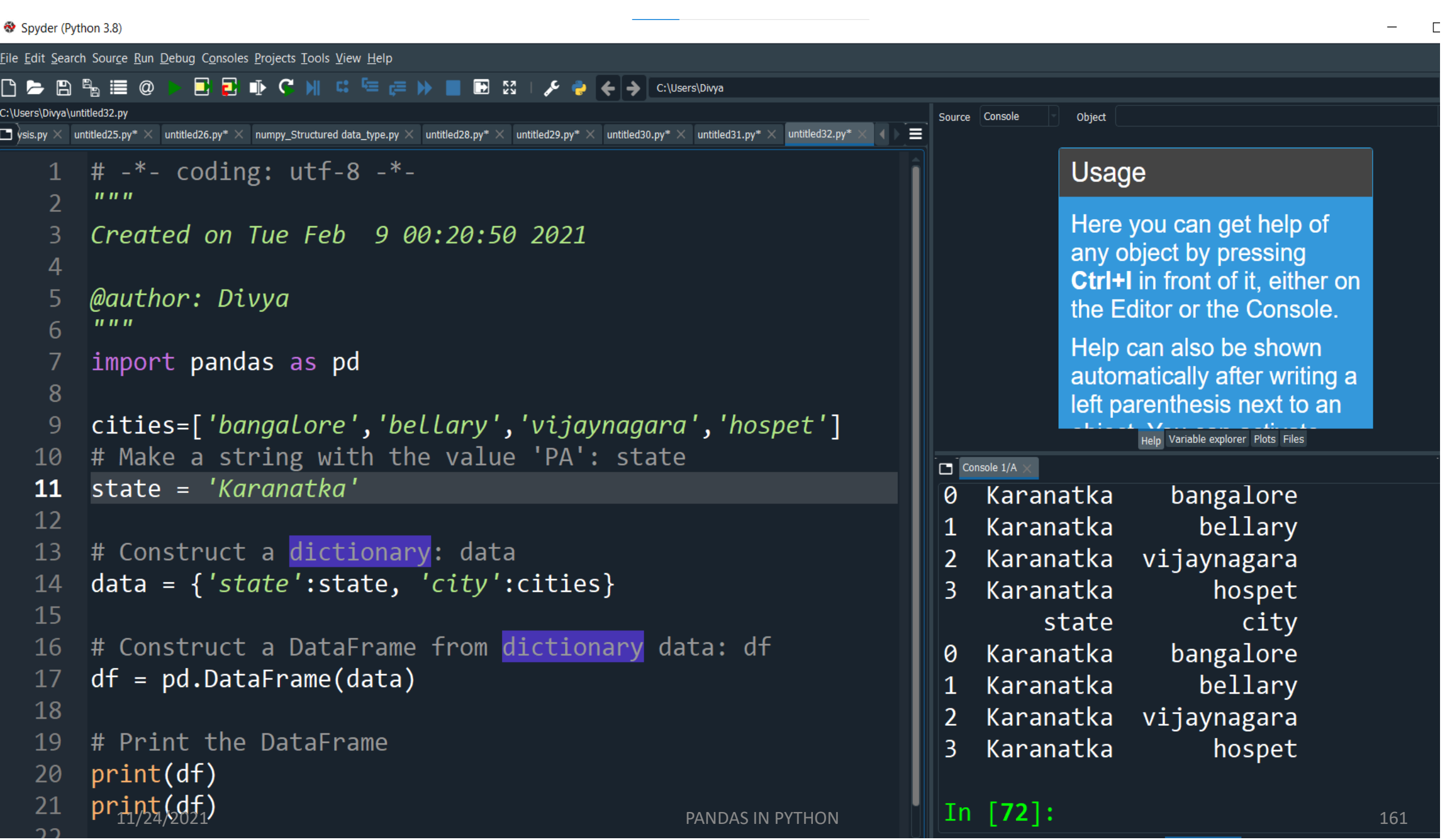
You can implicitly use 'broadcasting', a feature of NumPy, when creating pandas DataFrames.

In this exercise, you're going to create a DataFrame of cities in Pennsylvania that contains the city name in one column and the state name in the second.

We have imported the names of 15 cities as the list `cities`.

Your job is to construct a DataFrame from the list of cities and the string `'PA'`.

- Make a string object with the value 'PA' and assign it to state.
- Construct a dictionary with 2 key:value pairs: 'state':state and 'city':cities.
- Construct a pandas DataFrame from the dictionary you created and assign it to df.


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Divya
C:\Users\Divya\untitled32.py
ysis.py
untitled25.py*
untitled26.py*
numpy_Structured data_type.py
untitled28.py*
untitled29.py*
untitled30.py*
untitled31.py*
untitled32.py*
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Feb 9 00:20:50 2021
4
5 @author: Divya
6 """
7 import pandas as pd
8
9 cities=['bangalore','bellary','vijaynagara','hospet']
10 # Make a string with the value 'PA': state
11 state = 'Karanatka'
12
13 # Construct a dictionary: data
14 data = {'state':state, 'city':cities}
15
16 # Construct a DataFrame from dictionary data: df
17 df = pd.DataFrame(data)
18
19 # Print the DataFrame
20 print(df)
21 print(df)
Source
Console
Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an
Help
Variable explorer
Plots
Files
Console 1/A
0 Karanatka bangalore
1 Karanatka bellary
2 Karanatka vijaynagara
3 Karanatka hospet
state city
0 Karanatka bangalore
1 Karanatka bellary
2 Karanatka vijaynagara
3 Karanatka hospet
In [72]:

- Use pd.read_csv() with the string data_file to read the CSV file into a DataFrame and assign it to df1.

- Create a list of new column labels - 'year', 'population' - and assign it to the variable new_labels.

- Reread the same file, again using pd.read_csv(), but this time, add the keyword arguments header=0 and names=new_labels. Assign the resulting DataFrame to df2.

- Print both the df1 and df2 DataFrames to see the change in column names. This has already been done for you.

Delimiters, headers, and extensions in pandas

- Not all data files are clean and tidy. Pandas provides methods for reading those not-so-perfect data files that you encounter far too often.
- In this exercise, you have monthly stock data for four companies downloaded from **Yahoo Finance**.
- The data is stored as one row for each company and each column is the end-of-month closing price. The file name is given to you in the variable `file_messy`.
- In addition, this file has three aspects that may cause trouble for lesser tools: multiple header lines, comment records (rows) interleaved throughout the data rows, and space delimiters instead of commas.
- Your job is to use pandas to read the data from this problematic `file_messy` using non-default input options with `read_csv()` so as to tidy up the mess at read time.
- Then, write the cleaned up data to a CSV file with the variable `file_clean` that has been prepared for you, as you might do in a real data workflow.
- You can learn about the option input parameters needed by using `help()` on the pandas function `pd.read_csv()`

YouTube Python Repository

- Use pd.read_csv() *without using any keyword arguments* to read file_messy into a pandas DataFrame df1.

- Use .head() to print the first 5 rows of df1 and see how messy it is. Do this in the IPython Shell first so you can see how modifying read_csv() can clean up this mess.

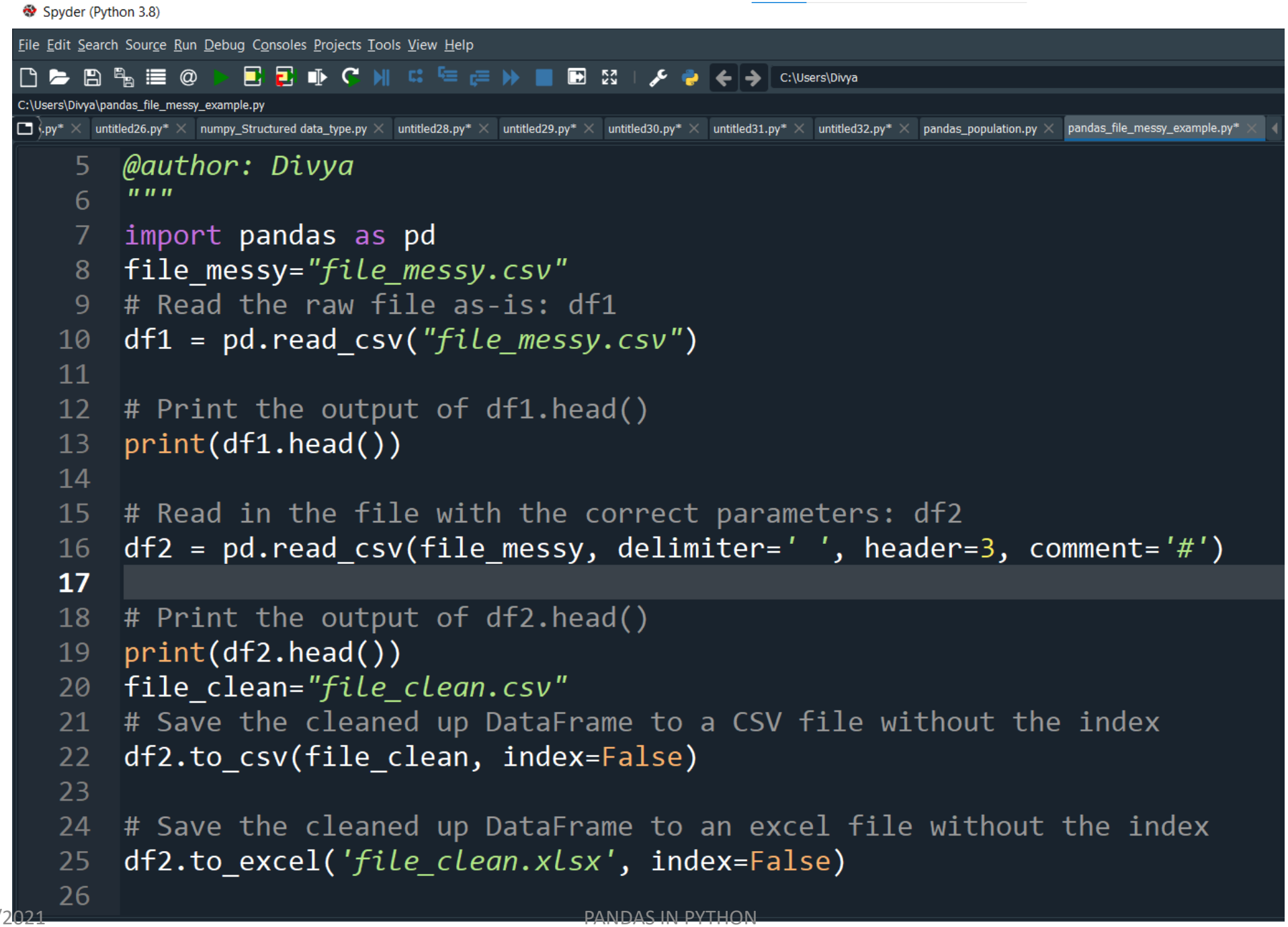
- Using the keyword arguments delimiter=' ', header=3 and comment='#', use pd.read_csv() again to read file_messy into a new DataFrame df2.

- Print the output of df2.head() to verify the file was read correctly.

- Use the DataFrame method .to_csv() to save the DataFrame df2 to the variable file_clean. Be sure to specify index=False.

- Use the DataFrame method .to_excel() to save the DataFrame df2 to the file 'file_clean.xlsx'. Again, remember to specify index=False.

YouTube Python Repository


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Divya
C:\Users\Divya\pandas_file_messy_example.py
untitled26.py*
numpy_Structured data_type.py
untitled28.py*
untitled29.py*
untitled30.py*
untitled31.py*
untitled32.py*
pandas_population.py
pandas_file_messy_example.py*
@author: Divya
"""
import pandas as pd
file_messy="file_messy.csv"
# Read the raw file as-is: df1
df1 = pd.read_csv("file_messy.csv")
# Print the output of df1.head()
print(df1.head())
# Read in the file with the correct parameters: df2
df2 = pd.read_csv(file_messy, delimiter=' ', header=3, comment='#')
# Print the output of df2.head()
print(df2.head())
file_clean="file_clean.csv"
# Save the cleaned up DataFrame to a CSV file without the index
df2.to_csv(file_clean, index=False)
# Save the cleaned up DataFrame to an excel file without the index
df2.to_excel('file_clean.xlsx', index=False)

Plotting with pandas

PANDAS FOUNDATIONS

YouTube Python Repository

AAPL stock data

```
import pandas as pd
import matplotlib.pyplot as plt
aapl = pd.read_csv('aapl.csv', index_col='date',
                   parse_dates=True)
aapl.head(6)
```

date	adj_close	close	high	low	open	volume
2000-03-01	31.68	130.31	132.06	118.50	118.56	38478000
2000-03-02	29.66	122.00	127.94	120.69	127.00	11136800
2000-03-03	31.12	128.00	128.23	120.00	124.87	11565200
2000-03-06	30.56	125.69	129.13	125.00	126.00	7520000
2000-03-07	29.87	122.87	127.44	121.12	126.44	9767600
2000-03-08	29.66	122.00	123.94	118.56	122.87	9690800

Plotting arrays (matplotlib)

```
close_arr = aapl['close'].values
type(close_arr)
```

```
numpy.ndarray
```

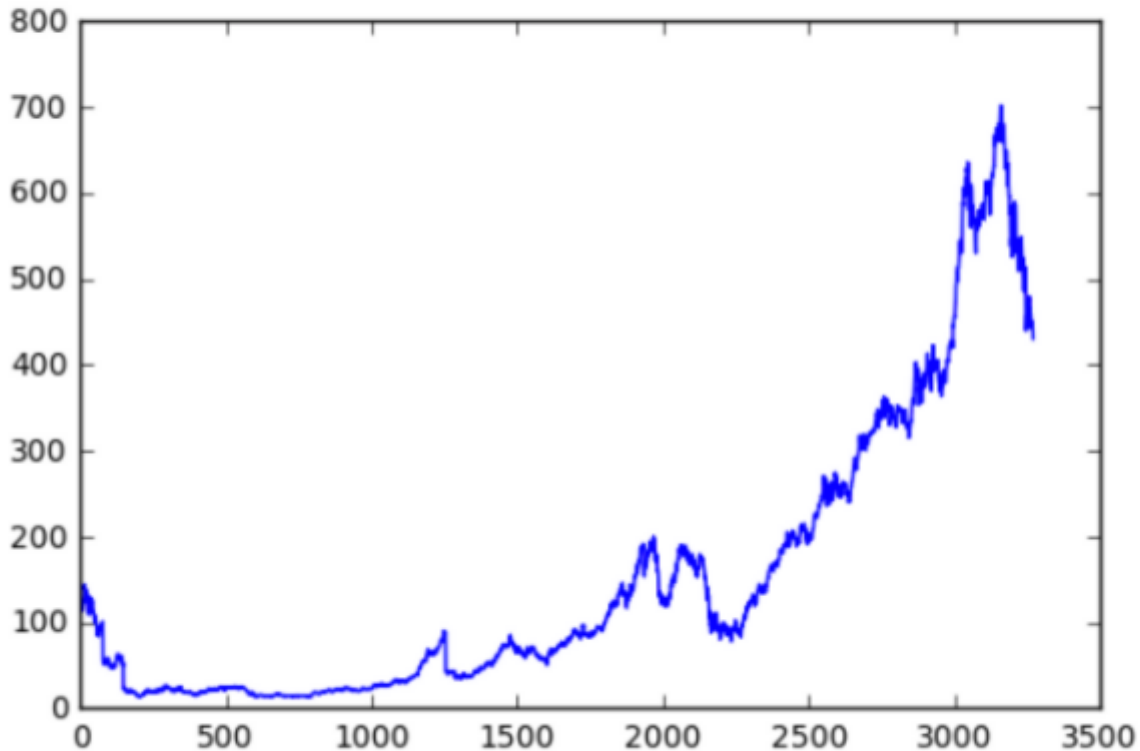
```
plt.plot(close_arr)
```

```
[<matplotlib.lines.Line2D at 0x115550358>]
```

```
plt.show()
```


YouTube Python Repository

Plotting arrays (matplotlib)



YouTube Python Repository

Plotting Series (matplotlib)

```
close_series = aapl['close']
type(close_series)
```

```
pandas.core.series.Series
```

```
plt.plot(close_series)
```

```
[<matplotlib.lines.Line2D at 0x11801cd30>]
```

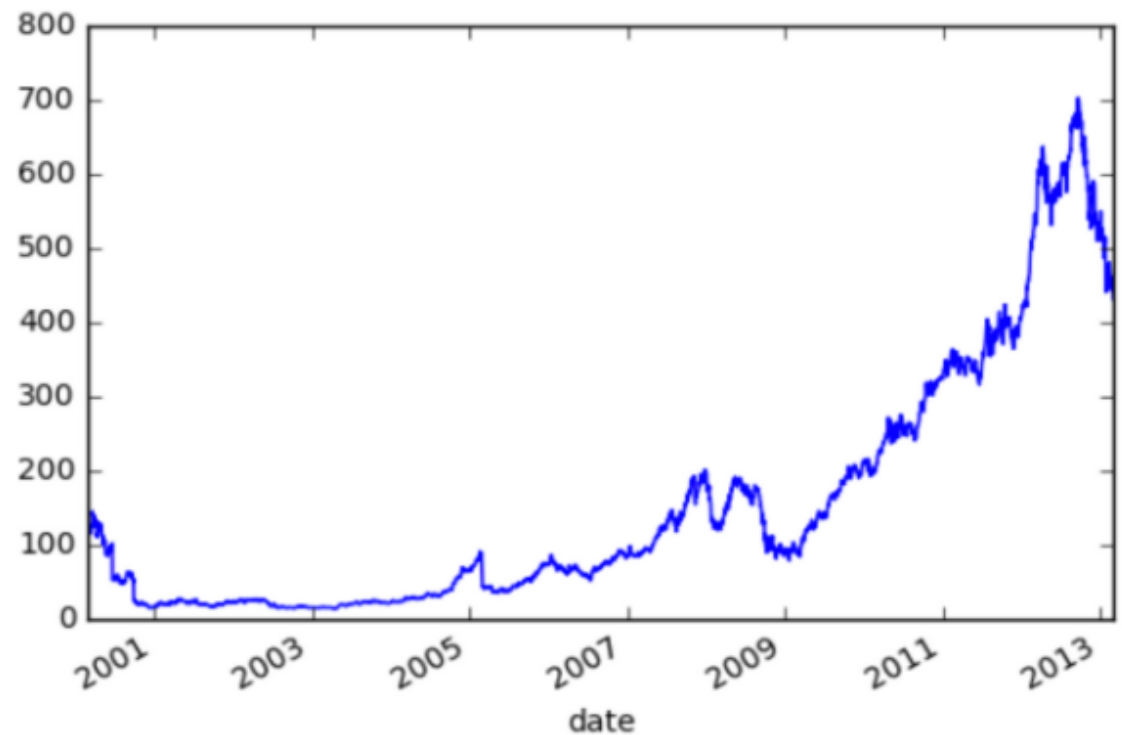
```
plt.show()
```


YouTube Python Repository

Plotting Series (pandas)

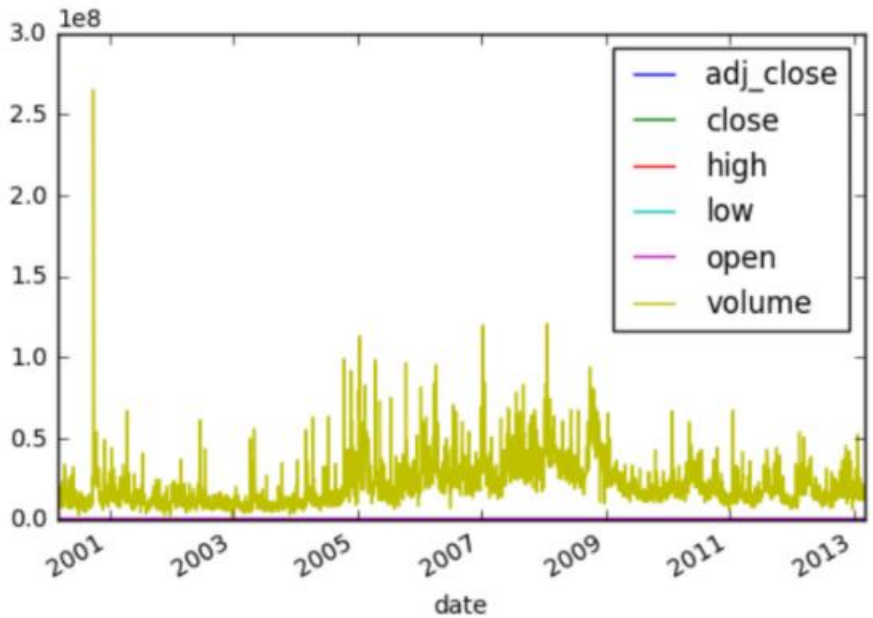
```
close_series.plot()  # plots Series directly
plt.show()
```

Plotting Series (pandas)



YouTube Python Repository

Plotting DataFrames (pandas)



YouTube Python Repository

Customizing plots

```
aapl['open'].plot(color='b', style='.-', legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>
```

```
aapl['close'].plot(color='r', style='.', legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>
```

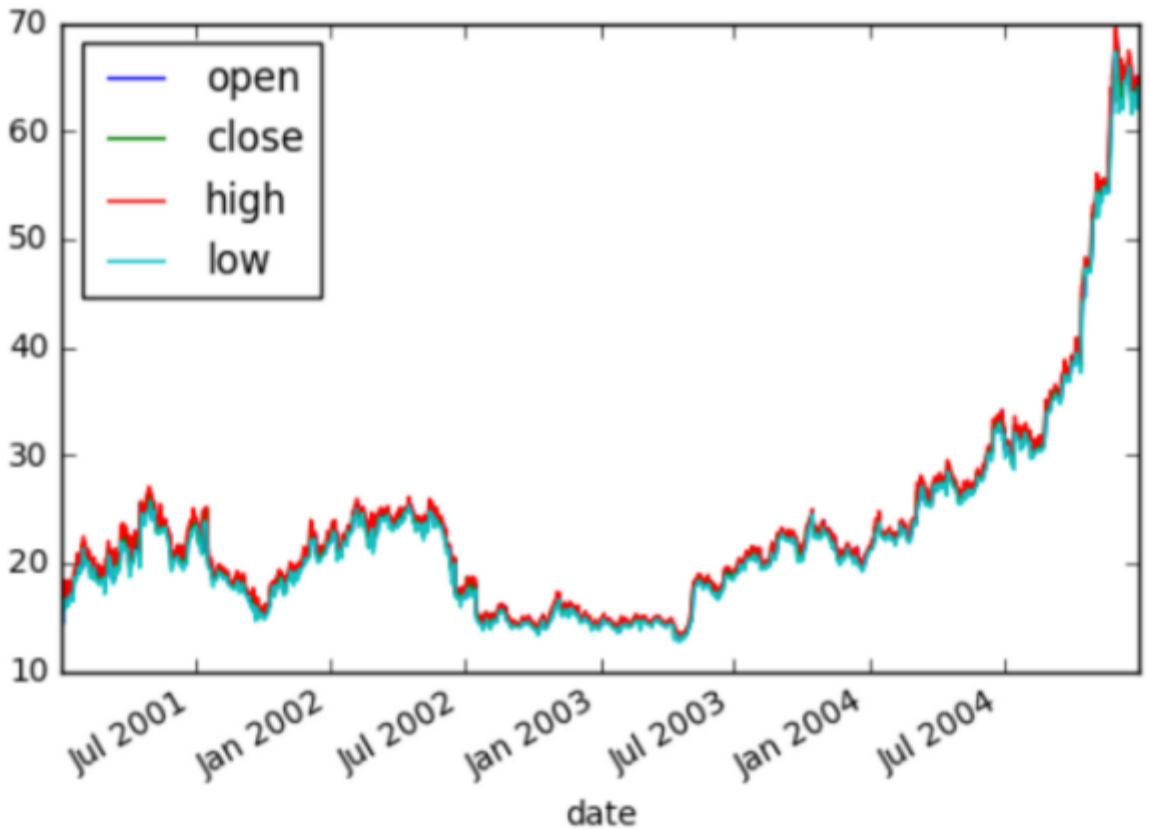
```
plt.axis(('2001', '2002', 0, 100))
```

```
'2001', '2002', 0, 100)
```

```
plt.show()
```


YouTube Python Repository

Saving plots



[YouTube Python Repository]

Plotting series using pandas

Data visualization is often a very effective first step in gaining a rough understanding of a data set to be analyzed.

Pandas provides data visualization by both depending upon and interoperating with the matplotlib library.

You will now explore some of the basic plotting mechanics with pandas as well as related matplotlib options.

We have pre-loaded a pandas DataFrame `df` which contains the data you need.
Your job is to use the DataFrame method `df.plot()` to visualize the data, and then explore the optional matplotlib input parameters that this `.plot()` method accepts.

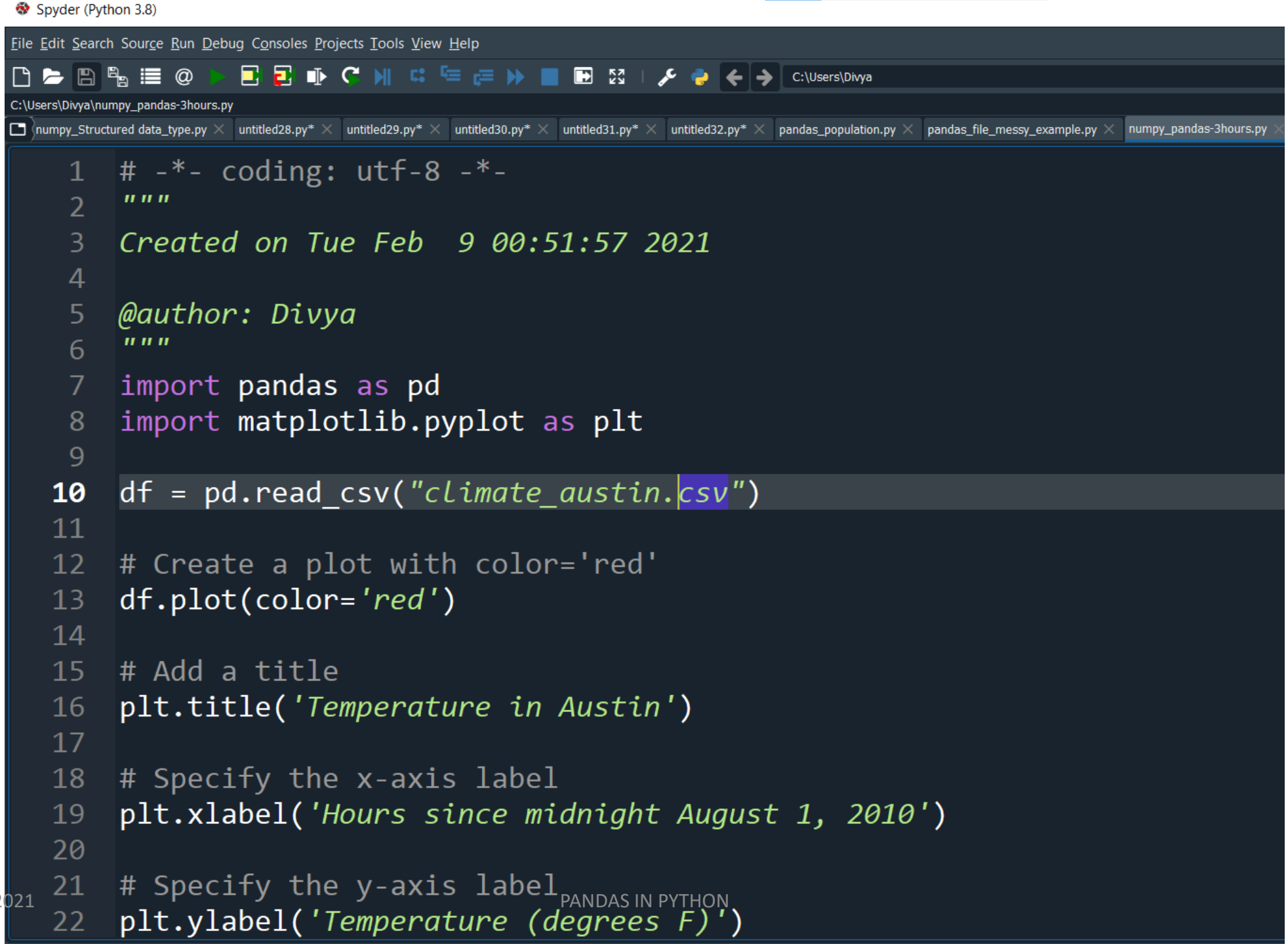
The pandas `.plot()` method makes calls to matplotlib to construct the plots.

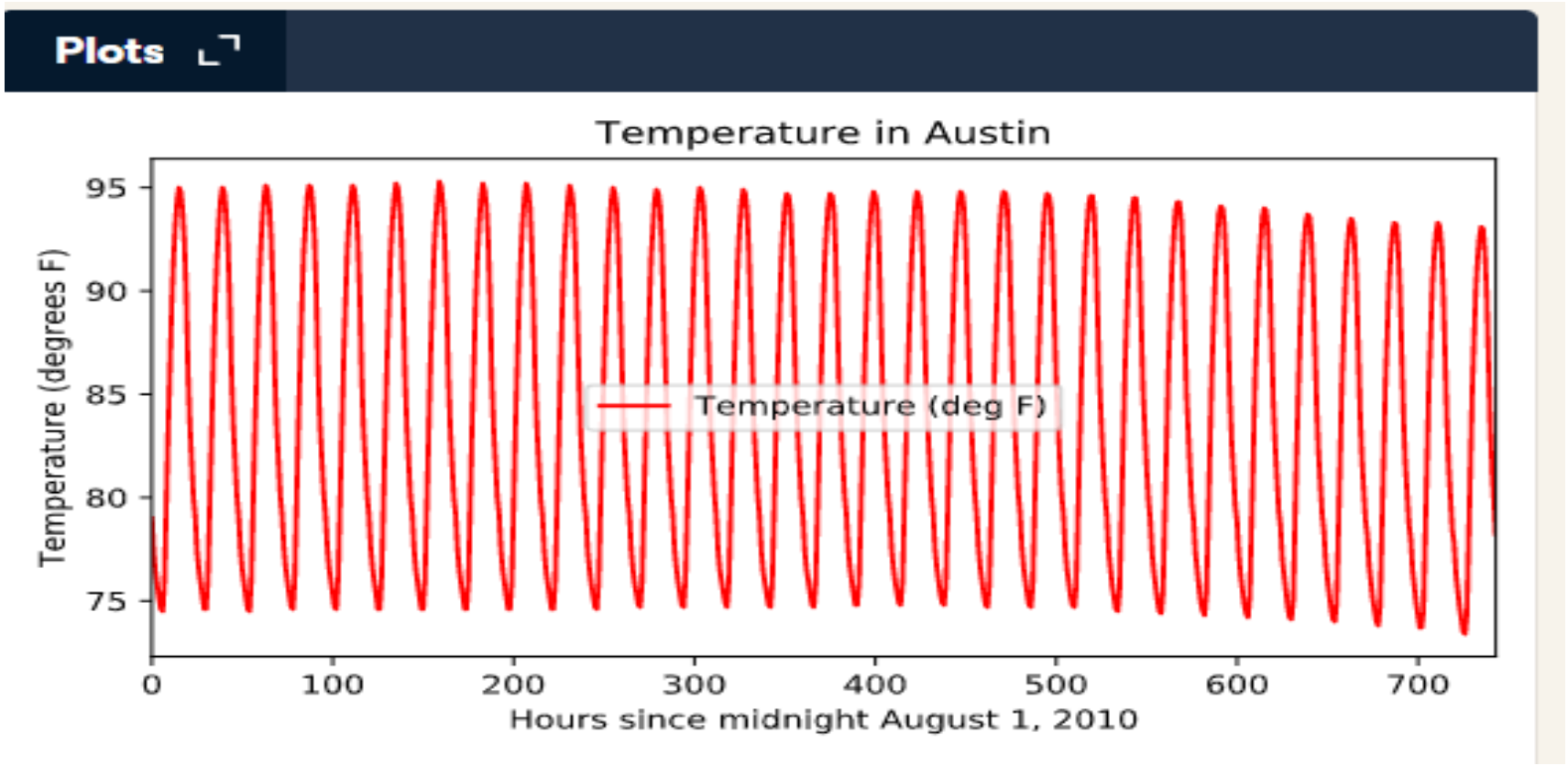
This means that you can use the skills you've learned in previous visualization courses to customize the plot. In this exercise, you'll add a custom title and axis labels to the figure.

YouTube Python Repository

- Create the plot with the DataFrame method df.plot(). Specify a color of 'red'.
  - *Note*: c and color are interchangeable as parameters here, but we ask you to be explicit and specify color.
- Use plt.title() to give the plot a title of 'Temperature in Austin'.
- Use plt.xlabel() to give the plot an x-axis label of 'Hours since midnight August 1, 2010'.
- Use plt.ylabel() to give the plot a y-axis label of 'Temperature (degrees F)'.
- Finally, display the plot using plt.show().

YouTube Python Repository


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Divya
C:\Users\Divya\numpy_pandas-3hours.py
numpy_Structured data_type.py
untitled28.py*
untitled29.py*
untitled30.py*
untitled31.py*
untitled32.py*
pandas_population.py
pandas_file_messy_example.py
numpy_pandas-3hours.py
-*- coding: utf-8 -*-
"""
Created on Tue Feb 9 00:51:57 2021
@author: Divya
"""
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("climate_austin.csv")
# Create a plot with color='red'
df.plot(color='red')
# Add a title
plt.title('Temperature in Austin')
# Specify the x-axis label
plt.xlabel('Hours since midnight August 1, 2010')
# Specify the y-axis label
plt.ylabel('Temperature (degrees F)')



YouTube Python Repository

Plotting DataFrames

- Comparing data from several columns can be very illuminating. Pandas makes doing so easy with multi-column DataFrames. By default, calling `df.plot()` will cause pandas to over-plot all column data, with each column as a single line.

- In this exercise, we have pre-loaded three columns of data from a weather data set - temperature, dew point, and pressure - but the problem is that pressure has different units of measure.

- The pressure data, measured in Atmospheres, has a different vertical scaling than that of the other two data columns, which are both measured in degrees Fahrenheit.

- Your job is to plot all columns as a multi-line plot, to see the nature of vertical scaling problem.

- Then, use a list of column names passed into the DataFrame `df[column_list]` to limit plotting to just one column, and then just 2 columns of data.

- When you are finished, you will have created 4 plots. You can cycle through them by clicking on the 'Previous Plot' and 'Next Plot' buttons.

YouTube Python Repository

pandas scatter plots

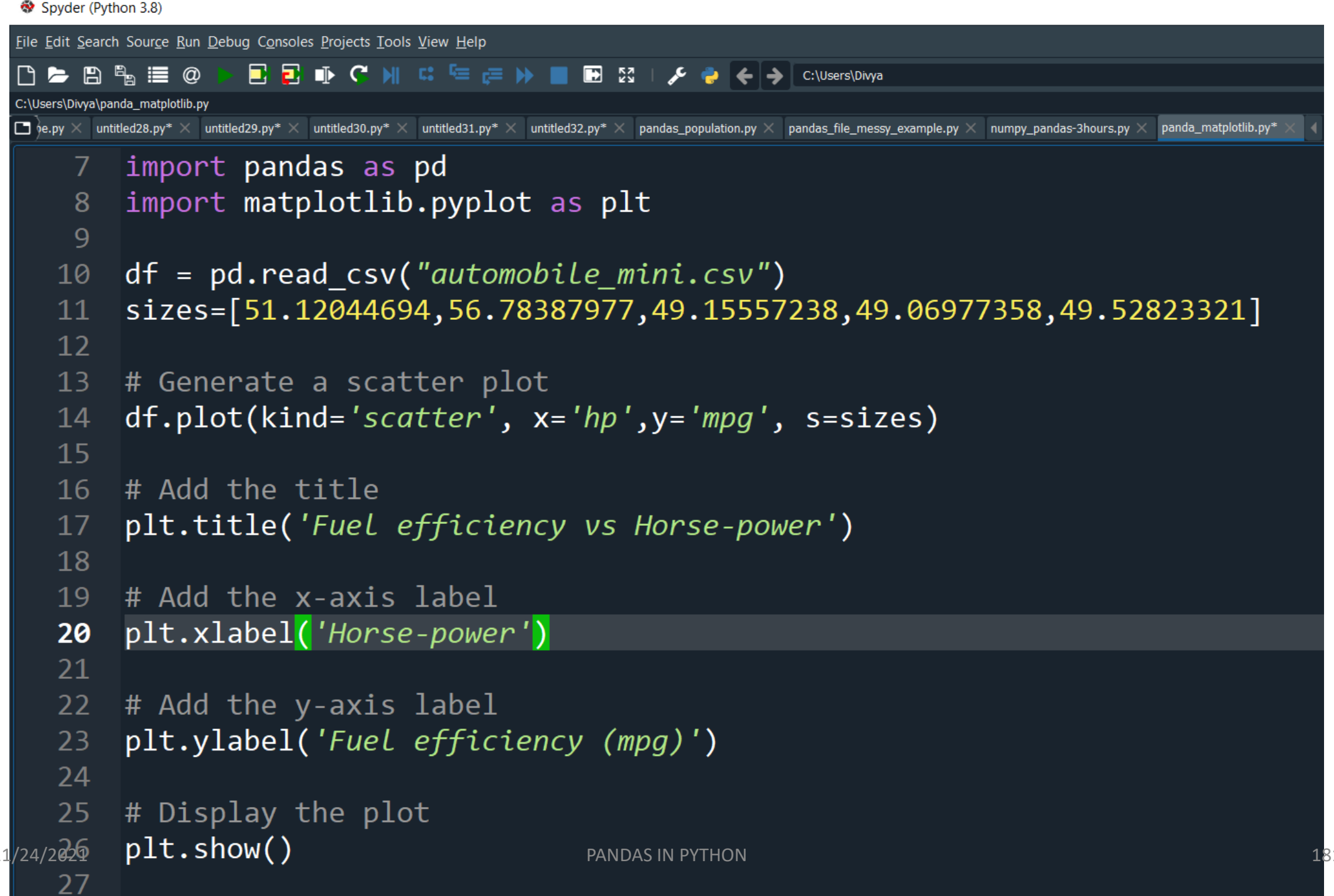
Pandas scatter plots are generated using the kind='scatter' keyword argument. Scatter plots require that the x and y columns be chosen by specifying the x and y parameters inside .plot().

Scatter plots also take an s keyword argument to provide the radius of each circle to plot in pixels.

In this exercise, you're going to plot fuel efficiency (miles-per-gallon) versus horse-power for 392 automobiles manufactured from 1970 to 1982 from the **UCI Machine Learning Repository**.

The size of each circle is provided as a NumPy array called sizes. This array contains the normalized 'weight' of each automobile in the dataset.

YouTube Python Repository


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Divya
C:\Users\Divya\panda_matplotlib.py
untitled28.py*
untitled29.py*
untitled30.py*
untitled31.py*
untitled32.py*
pandas_population.py
pandas_file_messy_example.py
numpy_pandas-3hours.py
panda_matplotlib.py*
7 import pandas as pd
8 import matplotlib.pyplot as plt
9
10 df = pd.read_csv("automobile_mini.csv")
11 sizes=[51.12044694,56.78387977,49.15557238,49.06977358,49.52823321]
12
13 # Generate a scatter plot
14 df.plot(kind='scatter', x='hp',y='mpg', s=sizes)
15
16 # Add the title
17 plt.title('Fuel efficiency vs Horse-power')
18
19 # Add the x-axis label
20 plt.xlabel('Horse-power')
21
22 # Add the y-axis label
23 plt.ylabel('Fuel efficiency (mpg)')
24
25 # Display the plot
26 plt.show()
27

pandas box plots

While pandas can plot multiple columns of data in a single figure, making plots that share the same x and y axes, there are cases where two columns cannot be plotted together because their units do not match.

The `.plot()` method can generate subplots for each column being plotted.

Here, each plot will be scaled independently.

In this exercise your job is to generate box plots for fuel efficiency (mpg) and weight from the automobiles data set. To do this in a single figure, you'll specify `subplots=True` inside `.plot()` to generate two separate plots.

YouTube Python Repository

- Make a list called cols of the column names to be plotted: 'weight' and 'mpg’.

- Call plot on df[cols] to generate a box plot of the two columns in a single figure. To do this, specify subplots=True

YouTube Python Repository

Spyder (Python 3.8)

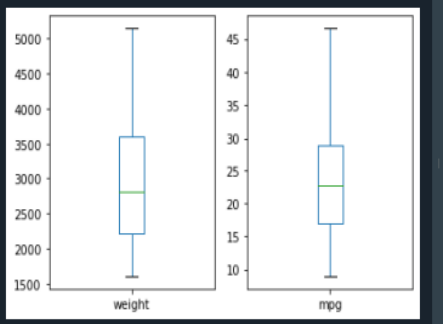
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Divya

C:\Users\Divya\pandas_matplotib_single_fig.py

ed29.py* untitled30.py* untitled31.py* untitled32.py* pandas_population.py pandas_file_messy_example.py numpy_pandas-3hours.py panda_matplotlib.py pandas_matplotib_single_fig.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Feb  9 01:33:23 2021
4
5  @author: Divya
6  """
7  import pandas as pd
8  import matplotlib.pyplot as plt
9  df = pd.read_csv("automobile.csv")
10
11
12 cols = ['weight','mpg']
13
14 # Generate the box plots
15 df[cols].plot(kind='box',subplots=True)
16
17 # Display the plot
18 plt.show()
```



90 %

Help Variable explorer Plots Files

Console 1/A

```
Users/Divya/
panda_matplotlib.py'
wdir='C:/Users/Divya

In [84]: runfile('C:
Users/Divya/
pandas_matplotib_sin
fig.py', wdir='C:/Us
Divya')

In [85]:
```

- Import matplotlib.pyplot and pandas as its usual alias.
- Read data using panda
- Add a 'blue' line plot of the % of degrees awarded to women in the Physical Sciences (physical_sciences) from 1970 to 2011 (year). Note that the x-axis should be specified first.
- Add a 'red' line plot of the % of degrees awarded to women in Computer Science (computer_science) from 1970 to 2011 (year).
- Use plt.show() to display the figure with the curves on the same axes.

```
# Import matplotlib.pyplot
import matplotlib.pyplot as plt

import pandas as pd

degrees=pd.read_csv('bachelor_degree.csv')

year=degrees['Year']
physical_sciences=degrees['Physical Sciences']
computer_science=degrees['Computer Science']

# Plot in blue the % of degrees awarded to women in the Physical Sciences
plt.plot(year, physical_sciences, color='blue')

# Plot in red the % of degrees awarded to women in Computer Science
plt.plot(year, computer_science, color='red')

# Display the plot
plt.show()
```

Spyder (Python 3.8)

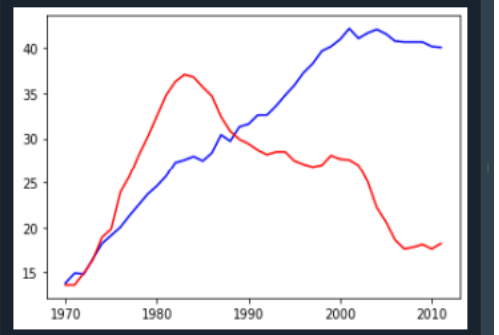
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Divya

C:\Users\Divya\untitled39.py

ed31.py* | untitled32.py* | pandas_population.py | pandas_file_messy_example.py | numpy_pandas-3hours.py | panda_matplotlib.py | pandas_matplotib_single_fig.py | untitled38.py* | untitled39.py*

```
7
8  # Import matplotlib.pyplot
9  import matplotlib.pyplot as plt
10
11 import pandas as pd
12
13 degrees=pd.read_csv('bachelor_degree.csv')
14
15 year=degrees['Year']
16 physical_sciences=degrees['Physical Sciences']
17 computer_science=degrees['Computer Science']
18
19 # Plot in blue the % of degrees awarded to women in the Physical Scienc
20 plt.plot(year, physical_sciences, color='blue')
21
22 # Plot in red the % of degrees awarded to women in Computer Science
23 plt.plot(year, computer_science, color='red')
24
25 # Display the plot
26 plt.show()
27
```



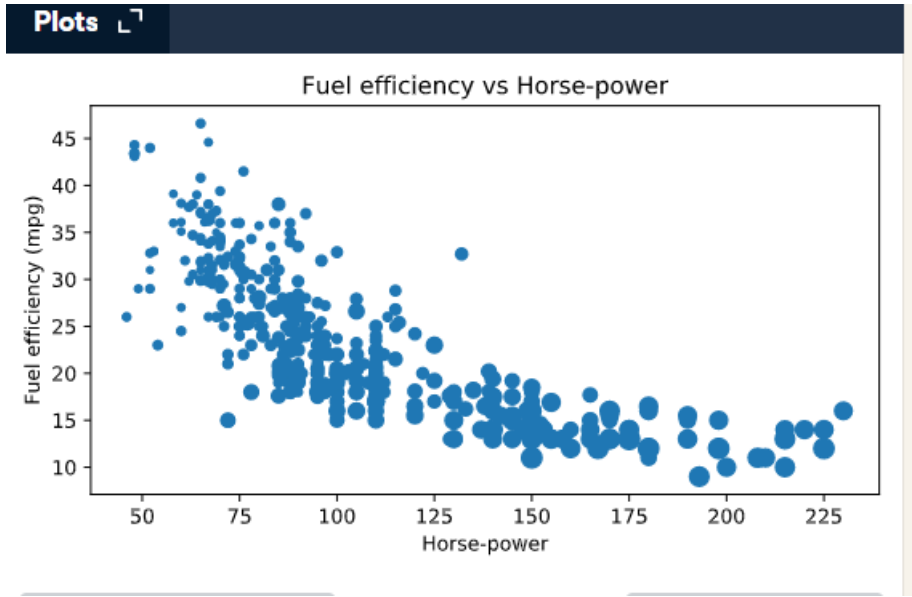
93 %

Help | Variable explorer | Plots | Files

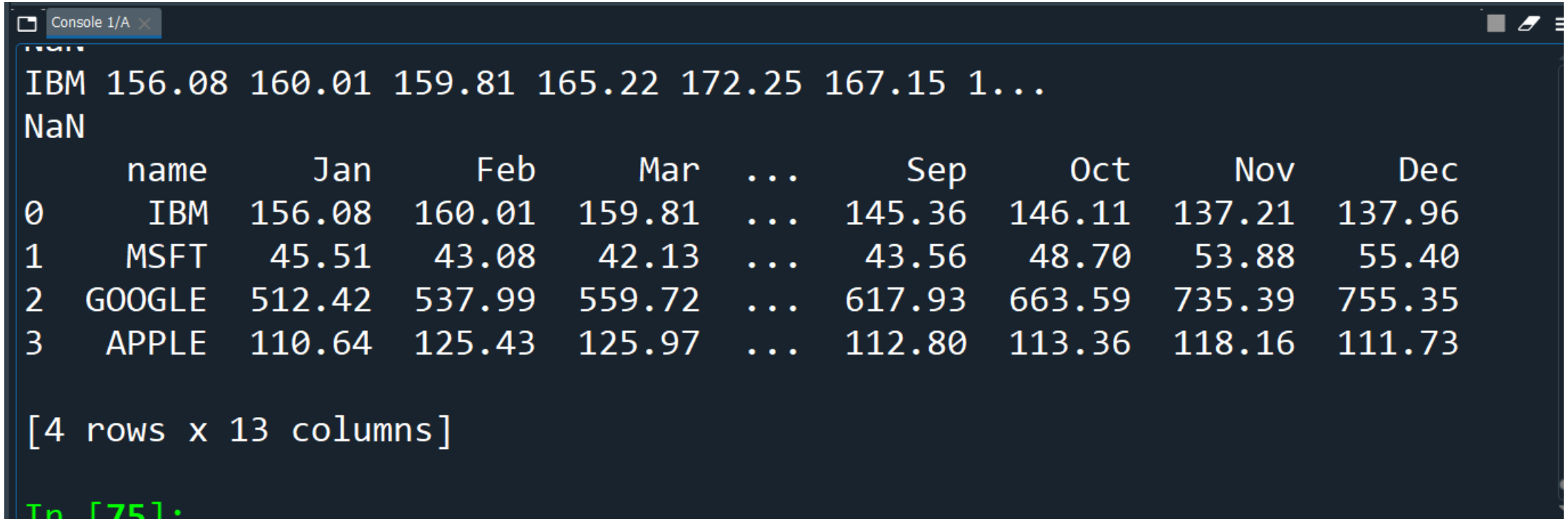
Console 1/A

```
AttributeError:
'DataFrame' object h
no attribute 'fracti

In [86]: runfile('C:
Users/Divya/
untitled39.py',
wdir='C:/Users/Divya
```


YouTube Python Repository



YouTube Python Repository



YouTube Python Repository

Here, you have the same three arrays year, physical_sciences, and computer_science representing percentages of degrees awarded to
women over a range of years.

You will use plt.axes() to create separate sets of axes in which you will draw each line plot.

and calling plt.axes([xlo, ylo, width, height]), a set of axes is created and made active with lower corner at coordinates (xlo, ylo) of the specified width and height. Note that these coordinates can be passed to plt.axes() in the form of a list or a tuple.

The coordinates and lengths are values between 0 and 1 representing lengths relative to the dimensions of the figure. After issuing a plt.axes() command, plots generated are put in that set of axes.

Create a set of plot axes with lower corner xlo and ylo of 0.05 and 0.05, width of 0.425, and height of 0.9 (in units relative to the figure dimension).

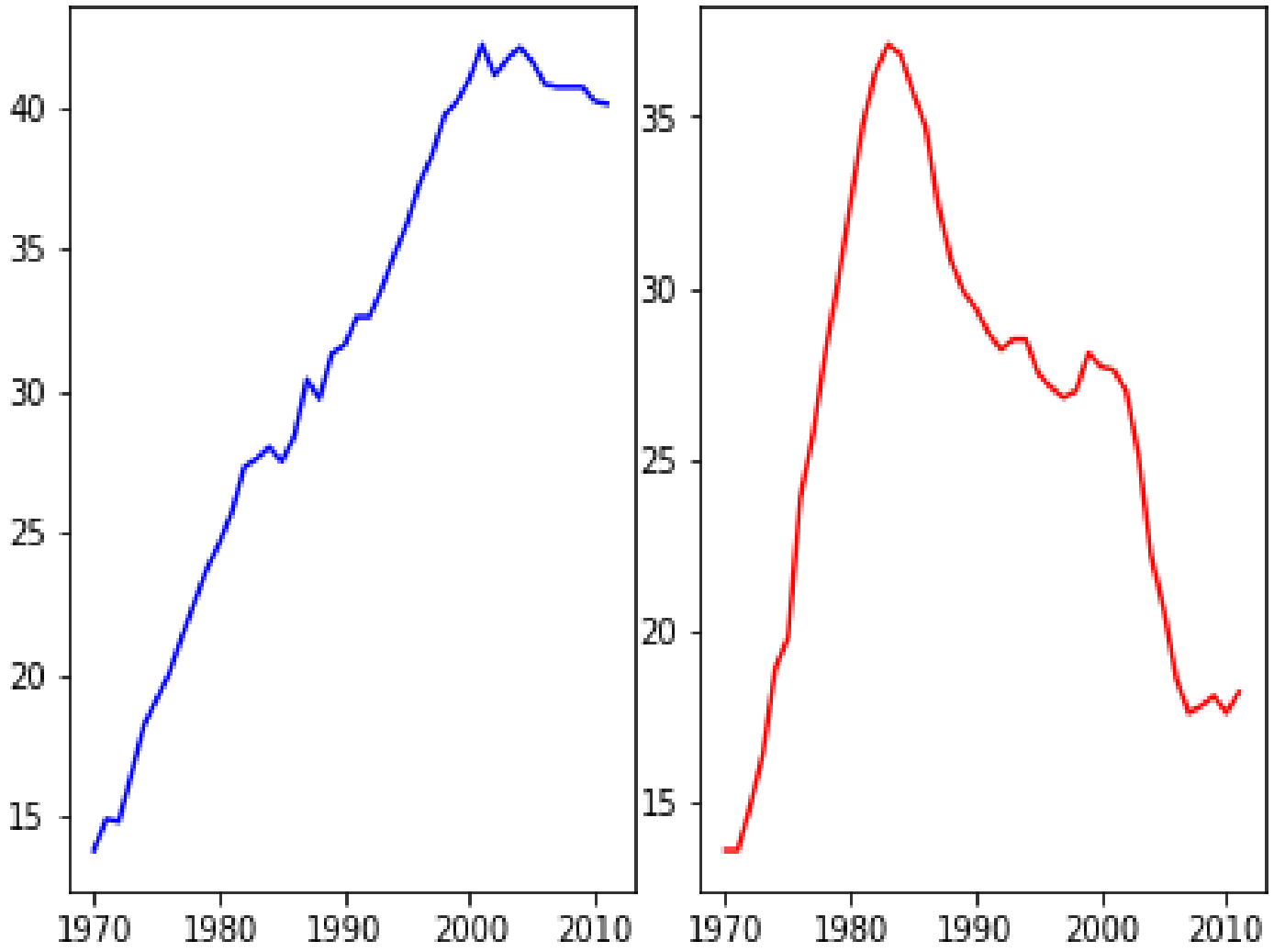
Note: Remember to pass these coordinates to plt.axes() in the form of a list: [xlo, ylo, width, height].

Plot the percentage of degrees awarded to women in Physical Sciences in blue in the active axes just created.

Create a set of plot axes with lower corner xlo and ylo of 0.525 and 0.05, width of 0.425, and height of 0.9 (in units relative to the figure dimension).

Plot the percentage of degrees awarded to women in Computer Science in red in the active axes just created

```
import matplotlib.pyplot as plt
import pandas as pd

degrees=pd.read_csv('D:/python_docs/bachelor_degree_women_data.csv')
year=degrees['Year']
physical_sciences=degrees['Physical Sciences']
computer_science=degrees['Computer Science']

# Create plot axes for the first line plot
plt.axes([0.05,0.05,0.425,0.9])

# Plot in blue the % of degrees awarded to women in the Physical Sciences
plt.plot(year, physical_sciences, color='blue')

# Create plot axes for the second line plot
plt.axes([0.525,0.05,0.425,0.9])

# Plot in red the % of degrees awarded to women in Computer Science
plt.plot(year, computer_science, color='red')

# Display the plot
plt.show()
```


40
35
30
25
20
15
1970
1980
1990
2000
2010
35
30
25
20
15
1970
1980
1990
2000
2010

Climate normals of Austin, TX from

	Temperature	DewPoint	Pressure	Date
0	46.2	37.5	1.0	20100101 00:00
1	44.6	37.1	1.0	20100101 01:00
2	44.1	36.9	1.0	20100101 02:00
3	43.8	36.9	1.0	20100101 03:00
4	43.5	36.8	1.0	20100101 04:00
5	43.0	36.5	1.0	20100101 05:00
6	43.1	36.3	1.0	20100101 06:00
7	42.3	35.9	1.0	20100101 07:00
8	42.5	36.2	1.0	20100101 08:00
9	45.9	37.8	1.0	20100101 09:00

Source: National Oceanic & Atmospheric Administration,
www.noaa.gov/climate

What method should we use to read the data?

The first step in our analysis is to read in the data. Upon inspection with a certain system tool, we find that the data appears to be ASCII encoded with comma delimited columns, but has no header and no column labels. Which of the following is the best method to start with to read the data files?

Answer the question 50XP

Possible Answers

- (●) `pd.read_csv()` press 1
- ( ) `pd.to_csv()` press 2
- ( ) `pd.read_hdf()` press 3
- ( ) `np.load()` press 4

Reading in a data file

Now that you have identified the method to use to read the data, let's try to read one file.

- The problem with real data such as this is that the files are almost never formatted in a convenient way.
- In this exercise, there are several problems to overcome in reading the file.
- First, there is no header, and thus the columns don't have labels.
- There is also no obvious index column, since none of the data columns contain a full date or time.
- ***Your job is to read the file into a DataFrame using the default arguments.***
- After inspecting it, you will re-read the file specifying that there are no headers supplied.

- Import pandas as pd.
- Read the file data_file into a DataFrame called df.
- Print the output of df.head(). This has been done for you. Notice the formatting problems in df.
- Re-read the data using specifying the keyword argument header=None and assign it to df_headers.
- Print the output of df_headers.head(). This has already been done for you. Hit 'Submit Answer' and see how this resolves the formatting issues.

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Divya

C:\Users\Divya\untitled44.py

9.py* untitled30.py* untitled31.py* untitled32.py* pandas_file_messy_example.py untitled38.py* untitled43.py* untitled44.py*

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Feb  9 14:57:39 2021
4
5  @author: Divya
6  """
7
8  # Import pandas
9  import pandas as pd
10
11 data_file="Austin_weathers.csv"
12 # Read in the data file: df
13 df = pd.read_csv(data_file)
14
15 # Print the output of df.head()
16 print(df.head())
17
18 # Read in the data file with header=None: df_headers
19 df_headers = pd.read_csv(data_file, header=None)
20
21 # Print the output of df_headers.head()
22 print(df_headers.head())
```

Source Editor Object

data_file

Definition : data_file(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result

Help Variable explorer Plots Files

Console 1/A

```
Date
1          46.2       37.5       1.0  20100101
00:00
2          44.6       37.1       1.0  20100101
01:00
3          44.1       36.9       1.0  20100101
02:00
4          43.8       36.9       1.0  20100101
03:00

In [111]:
```


YouTube Python Repository

Re-assigning column names

After the initial step of reading in the data, the next step is to clean and tidy it so that it is easier to work with.

In this exercise, you will begin this cleaning process by re-assigning column names and dropping unnecessary columns.

```
[5 rows x 44 columns]
    Wban      date  Time  ...  wind_direction station_pressure sea_level_pressure
0  13904  20110101   153  ...             340             29.49              30.01
1  13904  20110101   253  ...             010             29.49              30.01
2  13904  20110101   353  ...             350             29.51              30.03
3  13904  20110101   453  ...             020             29.51              30.04
4  13904  20110101   553  ...             010             29.53              30.06

[5 rows x 17 columns]
```

Cleaning and tidying datetime data

In order to use the full power of pandas time series, you must construct a DatetimeIndex.

To do so, it is necessary to clean and transform the date and time columns.

The DataFrame df_dropped you created in the last exercise is provided for you and pandas has been imported as pd.

Your job is to clean up the date and Time columns and combine them into a datetime collection to be used as the Index.

Signal min, max, median

Now that you have the data read and cleaned, you can begin with statistical EDA. First, you will analyze the 2011 Austin weather data.

Your job in this exercise is to analyze the `'dry_bulb_faren'` column and print the median temperatures for specific time ranges. You can do this using *partial datetime string* selection.

The cleaned dataframe is provided in the workspace as `df_clean`.

Analyze the amount of missingness

DEALING WITH MISSING DATA IN PYTHON

Nullity DataFrame

- Use either `.isnull()` or `.isna()` methods on the DataFrame

```
airquality_nullity = airquality.isnull()
airquality_nullity.head()
```

Date	Ozone	Solar	Wind	Temp
1976-05-01	False	False	False	False
1976-05-02	False	False	False	False
1976-05-03	False	False	False	False
1976-05-04	False	False	False	False
1976-05-05	True	True	False	False

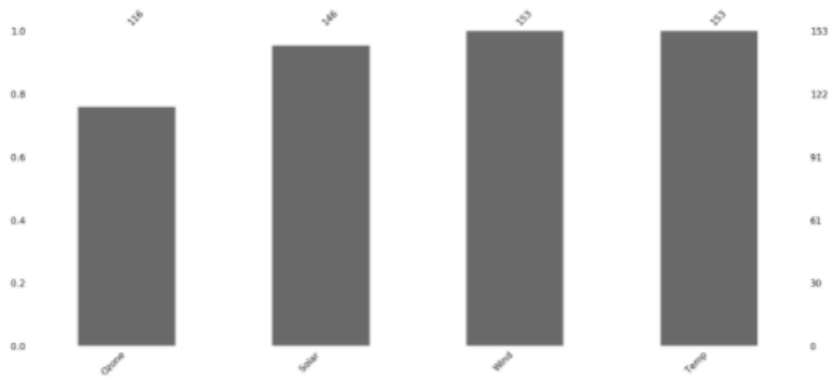
Nullity Bar

Missingno package

- Package for graphical analysis of missing values

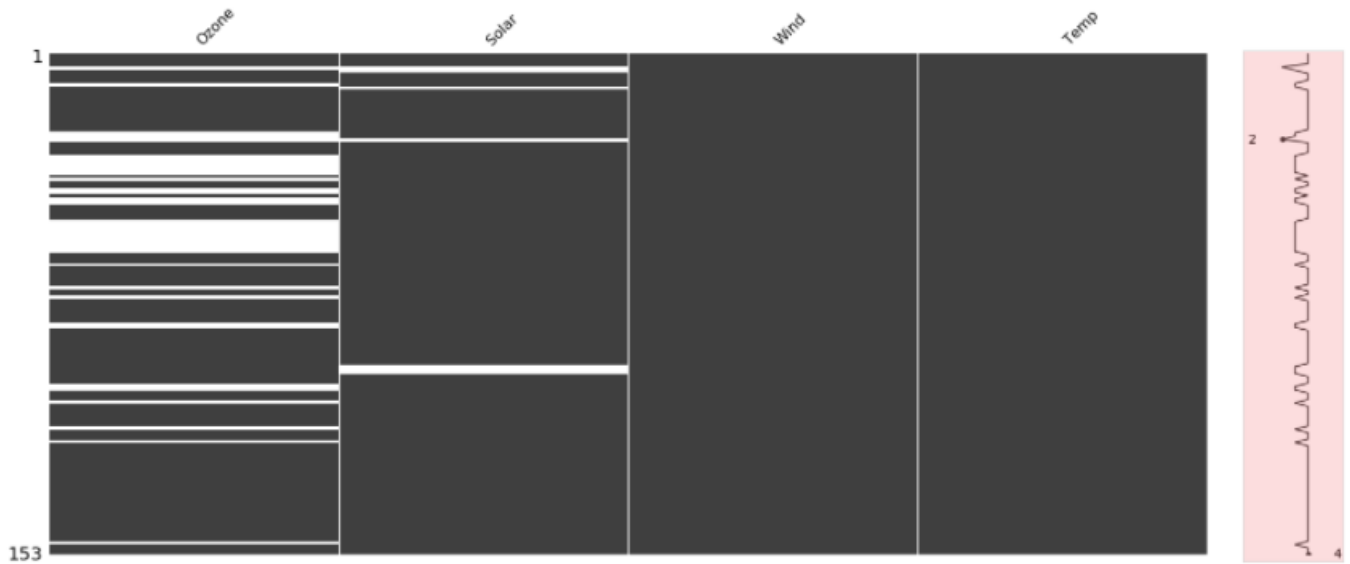
```
import missingno as msno
```

```
msno.bar(airquality)
```



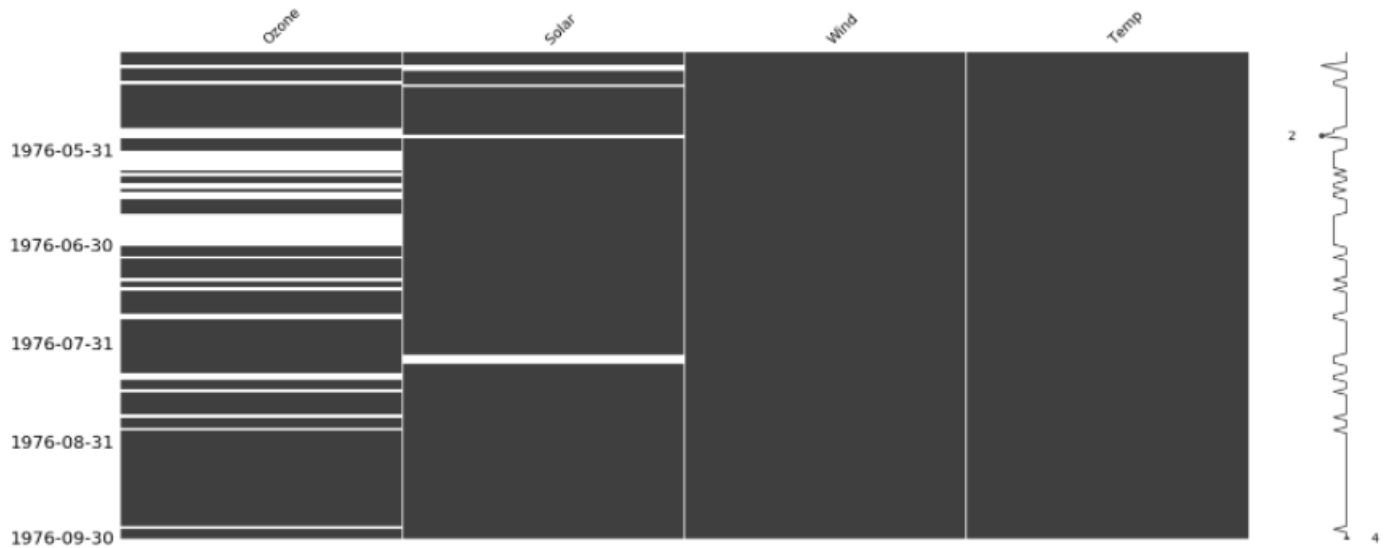
Nullity Matrix

```
msno.matrix(airquality)
```



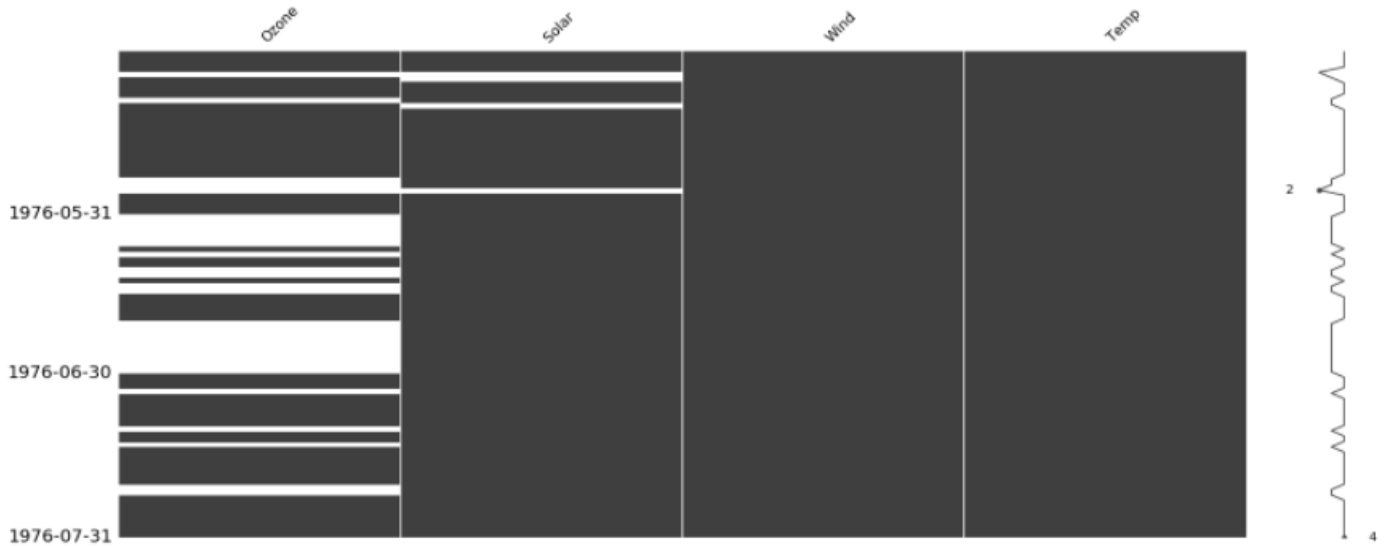
Nullity Matrix for time-series data

```
msno.matrix(airquality, freq='M')
```



Fine tuning the matrix

```
msno.matrix(airquality.loc['May-1976': 'Jul-1976'], freq='M')
```



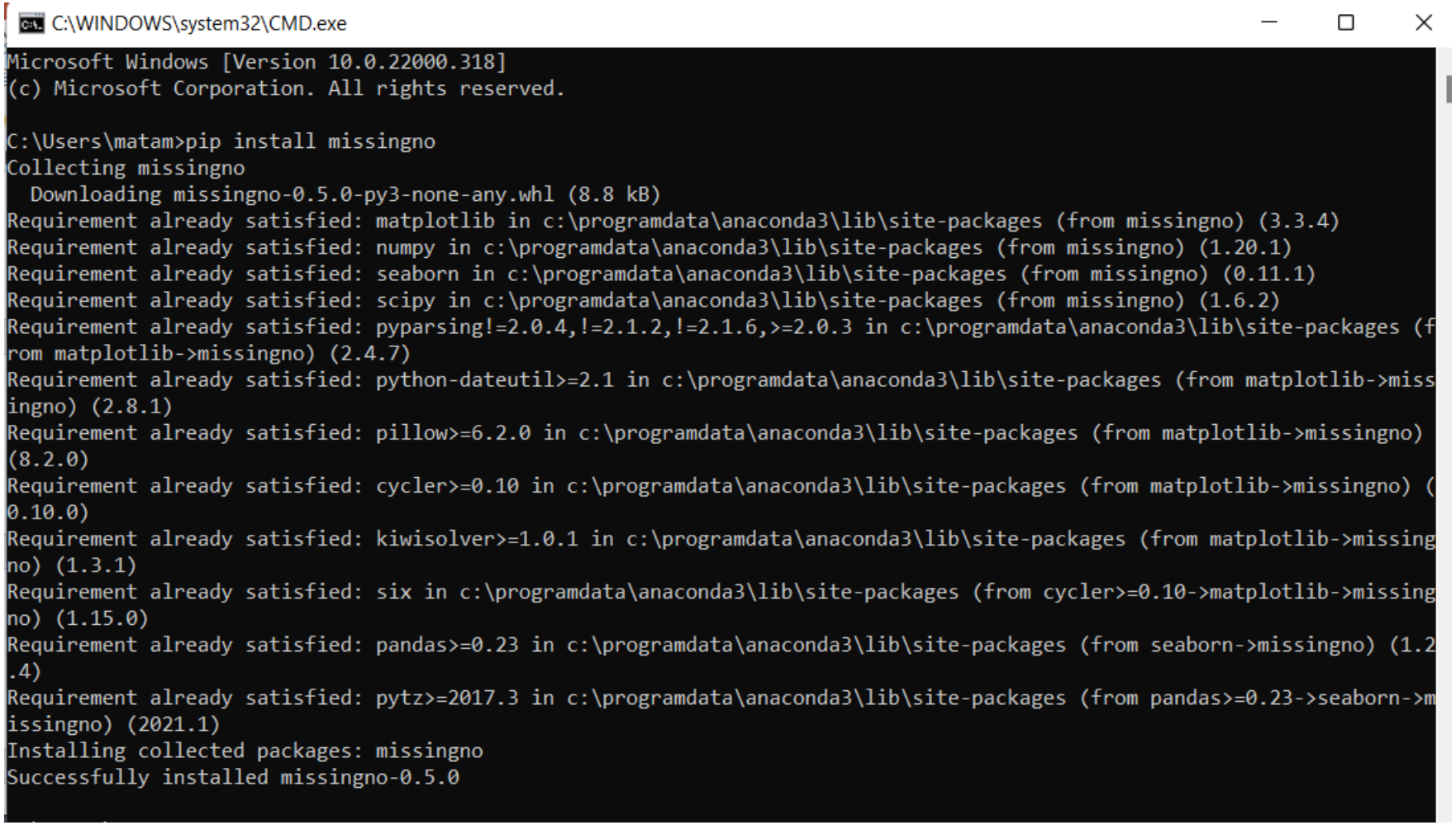
- Load 'air-quality.csv' into a pandas DataFrame while parsing the 'Date' column and setting it to the index column as well.

```python
import pandas as pd
# Load the airquality dataset
airquality = pd.read_csv('air-quality.csv', parse_dates=['Date'], index_col='Date')

# Create a nullity DataFrame airquality_nullity
airquality_nullity = airquality.isnull()
print(airquality_nullity.head())
```

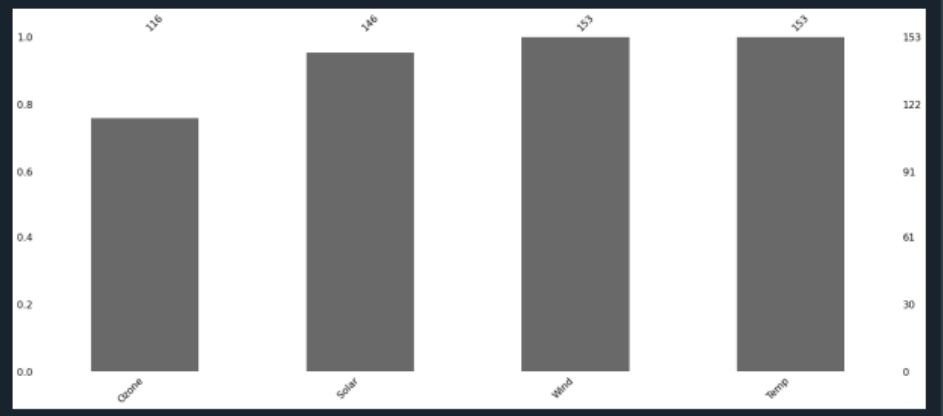
Variable explorer

Console 1/A

```
IPdb [26]: runfile('C:/Users/matam/.spyder
matam/.spyder-py3')
    a   b   c   d
0 NaN NaN NaN NaN
1 NaN NaN NaN NaN
2 NaN NaN NaN NaN
3 NaN NaN NaN NaN

IPdb [27]: runfile('C:/Users/matam/.spyder
matam/.spyder-py3')
            Ozone  Solar   Wind   Temp
Date
1976-05-01  False  False  False  False
1976-05-02  False  False  False  False
1976-05-03  False  False  False  False
1976-05-04  False  False  False  False
1976-05-05   True   True  False  False

IPdb [28]:
```


C:\WINDOWS\system32\CMD.exe
Microsoft Windows [Version 10.0.22000.318]
(c) Microsoft Corporation. All rights reserved.
C:\Users\matam>pip install missingno
Collecting missingno
Downloading missingno-0.5.0-py3-none-any.whl (8.8 kB)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from missingno) (3.3.4)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from missingno) (1.20.1)
Requirement already satisfied: seaborn in c:\programdata\anaconda3\lib\site-packages (from missingno) (0.11.1)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from missingno) (1.6.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->missingno) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->missingno) (2.8.1)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->missingno) (8.2.0)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->missingno) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->missingno) (1.3.1)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib->missingno) (1.15.0)
Requirement already satisfied: pandas>=0.23 in c:\programdata\anaconda3\lib\site-packages (from seaborn->missingno) (1.2.4)
Requirement already satisfied: pytz>=2017.3 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.23->seaborn->missingno) (2021.1)
Installing collected packages: missingno
Successfully installed missingno-0.5.0

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 19 16:41:06 2021

@author: matam
"""

# Import missingno as msno
import missingno as msno

import pandas as pd
# Load the airquality dataset
airquality = pd.read_csv('air-quality.csv', parse_dates=['Date'], index_col='Date')

# Create a nullity DataFrame airquality_nullity
airquality_nullity = airquality.isnull()
print(airquality_nullity.head())

# Plot amount of missingness
msno.bar(airquality)
```



Variable explorer | Help | Plots | Files

Console 1/A

```
IPdb [30]: runfile('C:/Users/matam/.spyder-py3/untitled57.py', wdir='C:/Users/
matam/.spyder-py3')
            Ozone  Solar   Wind   Temp
Date
1976-05-01  False  False  False  False
1976-05-02  False  False  False  False
1976-05-03  False  False  False  False
1976-05-04  False  False  False  False
1976-05-05   True   True  False  False
```

```
Figures now render in the Plots pane by default. To make them also ap
in the Console, uncheck "Mute Inline Plotting" under the Plots pane o
```

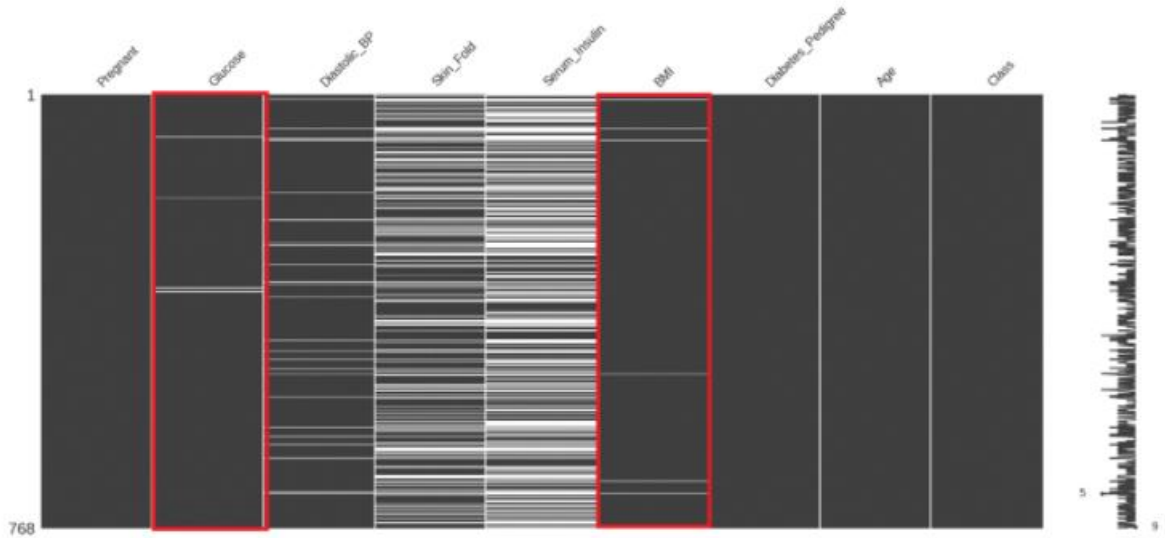
Missing Completely at Random(MCAR)

Definition:

> "Missingness has no relationship between any values, observed or missing"

MCAR - An example

```
msno.matrix(diabetes)
```



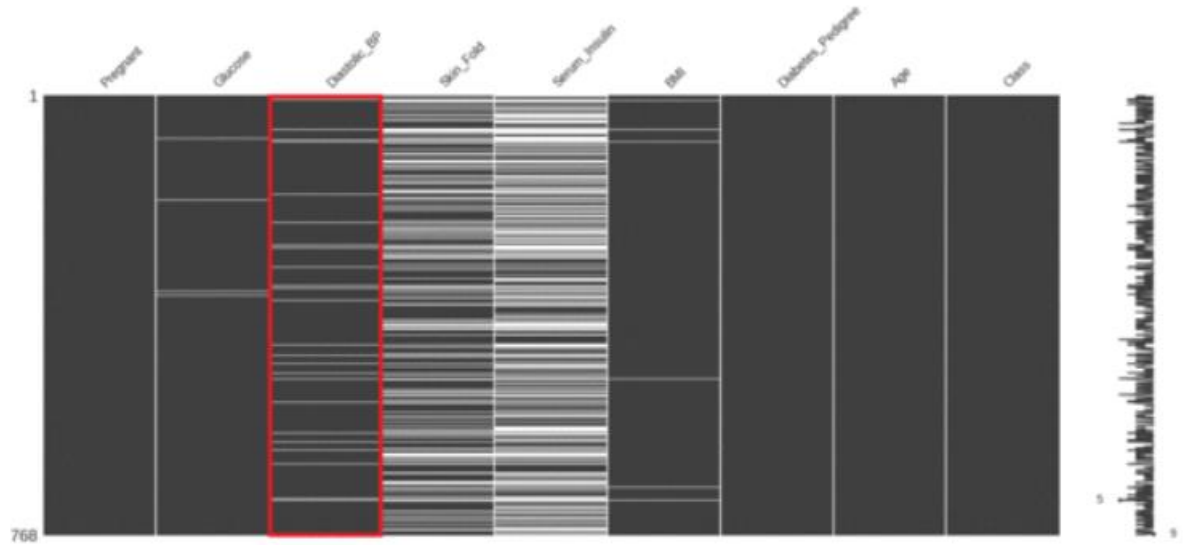
Missing at Random(MAR)

Definition:

> "There is a systematic relationship between missingness and other observed data, but not the missing data"

MAR - An example

```
msno.matrix(diabetes)
```



Missing not at Random(MNAR)

Definition:

> "There is a relationship between missingness and its values, missing or non-missing"

MNAR - An example

- Missingness pattern of the `diabetes` sorted by `Serum_Insulin`

```
sorted = diabetes.sort_values('Serum_Insulin')
msno.matrix(sorted)
```