Python_Tutorial

February 12, 2020

1 Introduction

1.1 Environment setup

Required software: - Python 3.6+ - Jupyter Notebook - Python libraries: numpy, pandas, matplotlib, scikit-learn, tensorflow, Keras

Setup options:

- Option #1 - install the listed packages separately
- Option #2 - install Anaconda Distribution (includes all necessary packages): https://www.anaconda.com/distribution/
- Option #3 - Google Colab: https://colab.research.google.com

1.2 Where to find datasets

Kaggle: https://www.kaggle.com/

Google Dataset Search: https://datasetsearch.research.google.com/

UCI Machine learning repository: http://archive.ics.uci.edu/ml/index.php

2 15-minute Python tutorial

2.1 Data types

2.1.1 Basic data types

[1]:
```
# Integers
integer = 123
very_large_int = 9999999999999999999999
print(integer)
print(very_large_int)
```

```
123
9999999999999999999999
```

[2]:
```
# Floats
# Accurate up to 15 decimal points:
```

```python
float1 = 0.852
float2 = 1 / 3
print(float1)
print(float2)
```

```
0.852
0.3333333333333333
```

[3]:
```python
# Booleans
this_is_true = True
this_is_also_true = 1 != 2
this_is_false = 1 == 2

print(this_is_true)
print(this_is_also_true)
print(this_is_false)
```

```
True
True
False
```

[4]:
```python
# Strings
word1 = "Hello"      # You can use either single or double quotes
word2 = 'world'
words = word1 + ' ' + word2
print(words)

num = 123
message = f'The value is: {num}'     # String formatting (f-strings)
print(message)
```

```
Hello world
The value is: 123
```

2.1.2 Lists

- Python equivalent of an array
- Ordered collection of values
- Zero-indexed
- Resizable
- Can contain elements of different types

[5]:
```python
# Creating a list and adding/removing values
list_demo = [1, 2, 3, 4, 5]
list_demo.append(6)          # At the end of the list
list_demo.insert(2, 999)     # At the specified position
```

```
list_demo.pop(0)
list_demo
```

[5]: [2, 999, 3, 4, 5, 6]

[6]:
```
# Getting a range of values with slices
list_demo = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list_demo[5])
print(list_demo[:5])
print(list_demo[5:])
print(list_demo[2:8])
```

```
5
[0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7]
```

[7]:
```
# Length of the list
list_demo = [5, 1, 4, 2, 3]
len(list_demo)
```

[7]: 5

[8]:
```
# Iterate through the list
numbers = [1, 2, 3, 4, 5]
squares = []
for number in numbers:
    square = number ** 2
    squares.append(square)
print(squares)
```

```
[1, 4, 9, 16, 25]
```

[9]:
```
# List comprehensions
numbers = [1, 2, 3, 4, 5]
squares = [number ** 2 for number in numbers]
print(squares)
```

```
[1, 4, 9, 16, 25]
```

2.1.3 Tuples

- Ordered list of values, similar to list
- Immutable
- Often used to return multiple values from a function

[10]:
```
# Creating a tuple:
tuple_demo = (1, 2, 3,)
```

```
tuple_demo
```

[10]:
```
(1, 2, 3)
```

[11]:
```
# Once created, it cannot be modified!
tuple_demo = (1, 2, 3,)
tuple_demo.append(999)
```

```
        ␣
↪---------------------------------------------------------------------------

      AttributeError                            Traceback (most recent call␣
↪last)

      <ipython-input-11-b0d41d3c4698> in <module>
        1 # Once created, it cannot be modified!
        2 tuple_demo = (1, 2, 3,)
  ----> 3 tuple_demo.append(999)


      AttributeError: 'tuple' object has no attribute 'append'
```

2.1.4 Dictionaries

- Key/value pairs
- Similar to Map datatype in Java

[14]:
```
# Creating a dictionary and setting/getting its values
dict_demo = {'a': 1, 'b': 2}

print(dict_demo['a'])
dict_demo['a'] = 100
dict_demo['c'] = 999
print(dict_demo)
```

```
1
{'a': 100, 'b': 2, 'c': 999}
```

[15]:
```
# Iterating through a dictionary
dict_demo = {'a': 1, 'b': 2, 'c': 3}
for key, value in dict_demo.items():
    print(key, value)
```

```
a 1
b 2
c 3
```

2.2 Control flow

2.2.1 Conditional statements

[16]:
```
# 'if-elif-else'
x = 0
if x == 0:
    print('x is zero')
elif x > 0:
    print('x is positive')
elif x < 0:
    print('x is negative')
else:
    print('not possible!')
```

```
x is zero
```

[17]:
```
# Empty lists, dicts, sets and tuples are treated as False
empty_list = []
empty_dict = {}
empty_set = set()
empty_tuple = tuple()

if empty_list:
    print('list')
if empty_dict:
    print('dict')
if empty_set:
    print('set')
if empty_tuple:
    print('tuple')
```

2.2.2 Loops

[18]:
```
# 'For' loop
some_list = [1, 2, 3, 4]
for item in some_list:
    print(item)
```

```
1
2
3
4
```

[19]:
```
# 'While' loop
x = 0
while True:        # Any boolean expression can be used here
```

```
    print(x)
    x += 1
    if x > 4:
        break
```

```
0
1
2
3
4
```

2.3 Functions

- Functions are declared with `def` keyword
- No return type and no parameter types are specified
- Same indentation rules apply as in other control flow blocks (`if` statement, `for` loop, etc)
- If function has no `return` statement, it returns `None` by default
- Functions can return multiple values
- Functions can be nested

[20]:
```
# Function definition
def add_numbers(num1, num2):
    res = num1 + num2
    return res

result = add_numbers(3, 7)
print(result)
```

```
10
```

[21]:
```
# Function can return multiple values at once (as a tuple)
def return_multiple():
    val1 = 1
    val2 = 'text'
    val3 = False
    return val1, val2, val3

# res1, res2, res3 = return_multiple()
# print(res1, res2, res3)
results = return_multiple()
res1, res2, res3 = results
print(res1, res2, res3)
```

```
1 text False
```

[22]:
```python
# Optional arguments
def some_func(first, second='text'):
    print(f'1st argument: {first}, 2nd argument: {second}')

some_func(123)
some_func(123, second=9999999)
```

```
1st argument: 123, 2nd argument: text
1st argument: 123, 2nd argument: 9999999
```

2.4 Classes and objects

[23]:
```python
class ComplexNumber:

    # Constructor
    def __init__(self, real, imag):
        # Instance variables
        self.real = real
        self.imag = imag

    # Instance method
    def add(self, other):
        sum_real = self.real + other.real
        sum_imag = self.imag + other.imag
        return ComplexNumber(sum_real, sum_imag)

# First argument ('self') is required and refers to the current instance

# Creating objects (class instances)
c1 = ComplexNumber(1, 2)
c2 = ComplexNumber(100, 200)

# Calling methods
c3 = c1.add(c2)

print(c3.real, c3.imag)
```

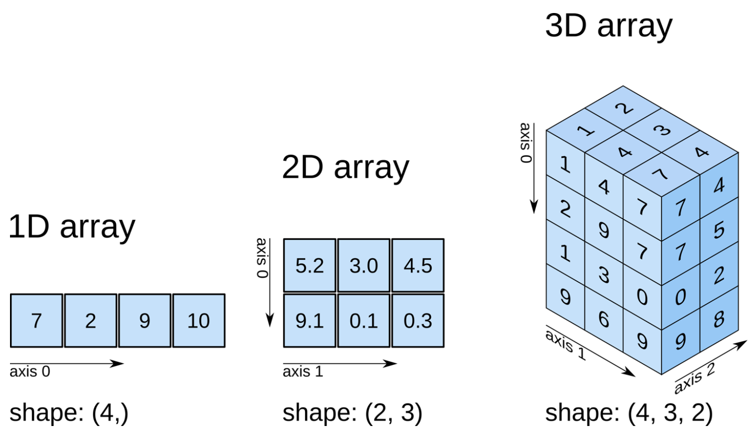
```
101 202
```

3 NumPy

NumPy is a Python C extension library for array-oriented computing: * Efficient * In-memory * Contiguous * Homogeneous

[24]:
```python
import numpy as np
```

3.1 Multidimensional array



Source: https://www.oreilly.com/library/view/elega
scipy/9781491922927/ch01.html

3.1.1 Array creation

[25]:
```python
# Creating numpy arrays from Python lists:

vector = np.array([1, 2, 3, 4, 5, 6])
matrix = np.array([
    [6, 7, 8, 9, 10, 11],
    [11, 12, 13, 14, 15, 16],
])
print(type(vector))
print(type(matrix))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

[26]:
```python
# Converting numpy arrays back to Python lists:
print(vector.tolist())
print(matrix.tolist())
```

```
[1, 2, 3, 4, 5, 6]
[[6, 7, 8, 9, 10, 11], [11, 12, 13, 14, 15, 16]]
```

[27]:
```python
# Using built-in numpy functions to
# initialize array with some constant:

zero_matrix = np.zeros((3, 5))
ones_matrix = np.ones((3, 5))
number_matrix = np.full((3, 5), 3.14)

print(zero_matrix)
print(ones_matrix)
print(number_matrix)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
[[3.14 3.14 3.14 3.14 3.14]
 [3.14 3.14 3.14 3.14 3.14]
 [3.14 3.14 3.14 3.14 3.14]]
```

[28]:

```
# Generating sequences

# Similar to built-in Python range() function:
sequence_ints = np.arange(0, 20, 2)

# Evenly spaced numbers over some interval:
sequence_floats = np.linspace(0, 1, 5)

print(sequence_ints)
print(sequence_floats)
```

```
[ 0  2  4  6  8 10 12 14 16 18]
[0.   0.25 0.5  0.75 1.  ]
```

[29]:

```
# Random values

# Floating-point values between 0 and 1:
random_floats = np.random.random((3, 5))

# Integer values between 0 and 100:
random_integers = np.random.randint(0, 100, (3, 5))

print(random_floats)
print(random_integers)
```

```
[[0.18155313 0.04598169 0.36782286 0.18814205 0.62813224]
 [0.15928699 0.63660025 0.63030445 0.11659768 0.074069  ]
 [0.52025724 0.23446715 0.05585017 0.42650664 0.03891379]]
[[85 28 65 67 67]
 [14 62 99 63 34]
 [75 94 33 14 60]]
```

[30]:

```
# Other array creation routines

# Identity matrices:
identity_matrix = np.eye(5)
```

```
# Allocate array without initializing its values:
uninitialized_matrix = np.empty((3, 5))

print(identity_matrix)
print(uninitialized_matrix)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
[[0.18155313 0.04598169 0.36782286 0.18814205 0.62813224]
 [0.15928699 0.63660025 0.63030445 0.11659768 0.074069  ]
 [0.52025724 0.23446715 0.05585017 0.42650664 0.03891379]]
```

3.1.2 Shapes

[31]:
```
vector = np.array([1, 2, 3, 4, 5, 6])
matrix = np.array([
    [6, 7, 8, 9, 10, 11],
    [11, 12, 13, 14, 15, 16],
])

# Getting array shape:
print(vector.shape)
print(matrix.shape)

# Number of dimensions:
print(vector.ndim)
print(matrix.ndim)
```

```
(6,)
(2, 6)
1
2
```

[32]:
```
# Arrays can be reshaped:
print(matrix)
print(matrix.reshape(3, 4))
```

```
[[ 6  7  8  9 10 11]
 [11 12 13 14 15 16]]
[[ 6  7  8  9]
 [10 11 11 12]
 [13 14 15 16]]
```

[33]:
```python
# Also possible to reshape into a different number of dimensions:

# 2D to 1D
print(matrix.reshape(12))

# 1D to 2D
print(vector.reshape((2, 3)))
```

```
[ 6  7  8  9 10 11 11 12 13 14 15 16]
[[1 2 3]
 [4 5 6]]
```

[34]:
```python
# Converting multi-dimensional array into 1D:

# This returns a new 1D array:
print(matrix.flatten())

# This returns a view into the original array:
print(matrix.ravel())
```

```
[ 6  7  8  9 10 11 11 12 13 14 15 16]
[ 6  7  8  9 10 11 11 12 13 14 15 16]
```

3.1.3 Data types

bool_	Boolean (True or False) stored as a byte
int_	Default integer type
intc	Identical to C int e.g int32 in64
intp	Integer used for indexing
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Source: https://medium.com/@vsvaibhav2016/basics-of-numpy-python-for-data-analysis-45b0c43f591b

[35]:
```
vector = np.array([1, 2, 3, 4, 5, 6])
matrix = np.array([
    [6, 7, 8, 9, 10, 11],
    [11, 12, 13, 14, 15, 16],
])

# Inspecting array data type:
print(matrix.dtype)
```

```
int64
```

[36]:
```
# Data type is implicitly determined
# during array creation:

vector_int = np.array([1, 2, 3])
vector_float = np.array([1.0, 2.0, 3.0])

print(vector_int.dtype)
print(vector_float.dtype)
```

```
int64
float64
```

[37]:
```
# It can also be specified explicitly:
vector_small_int = np.array([1, 2, 3], dtype=np.uint8)
vector_small_float = np.array([1.0, 2.0, 3.0], dtype=np.float32)

print(vector_small_int.dtype)
print(vector_small_float.dtype)
```

```
uint8
float32
```

[38]:
```
# Data type conversions
# Be careful with casting into a narrower data type!

print(vector_int.astype(np.float32))
print(vector_float.astype(np.int32))
```

```
[1. 2. 3.]
[1 2 3]
```

3.1.4 NumPy arrays vs. Python lists

[39]:
```
# Once the numpy array is created, you cannot
# increase/decrease its size - a new array
# has to be created instead
some_vector = np.array([1, 2, 3, 4, 5])

try:
    some_vector.append(6)
except:
    print('Regular append() wont work...')

# This returns a new array:
new_vector = np.append(some_vector, 7)
print(new_vector)
```

```
Regular append() wont work...
[1 2 3 4 5 7]
```

[40]:
```
# Some operations not possible with Python lists
# can be done with numpy arrays:

some_vector = np.array([1, 2, 3, 4, 5])
```

```
# Wouldn't work with regular lists:
some_vector += 2

print(some_vector)
```

```
[3 4 5 6 7]
```

[41]:
```
# Adding two arrays together does not concatenate them,
# but adds their elements together.
# Same goes for other arithmetic operations.

vec1 = np.array([1, 2, 3])
vec2 = np.array([4, 5, 6])
vec1 + vec2
```

[41]:
```
array([5, 7, 9])
```

3.2 Indexing

[42]:
```
# Accessing elements of numpy arrays:
vector = np.array([1, 2, 3, 4, 5, 6])
matrix = np.array([
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
])

print(vector[3])

# Both options work:
print(matrix[2][2])
print(matrix[2, 2])
```

```
4
11
11
```

3.2.1 Slicing

[43]:
```
# Regular list slicing works in any number of dimensions:
print(vector[0::2])    # Every second element
print(matrix[:, 0::2]) # Every second column and all rows
```

```
[1 3 5]
[[ 1  3]
 [ 5  7]
 [ 9 11]]
```

3.2.2 Integer indexing

[44]:
```
# You can use integer lists (or numpy arrays)
# to access certain elements of numpy array

# Either a regular Python list or a numpy array:
indices_py = [1, 3, 5]
indices_np = np.array([1, 3, 5])

print(vector)
print(vector[indices_py])
print(vector[indices_np])
```

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

[45]:
```
# Integer indexing: 2D example

# The tuple below must contain the same number of lists
# as there are dimensions in the numpy array.

# Each list represents indices for each array axis:
indices = (
    [0, 2],    # First axis (rows)
    [0, 3],    # Second axis (columns)
)

# Integer indices can be used both for access and assignment:
print(matrix[indices])
matrix[indices] = 9999
print(matrix)
```

```
[ 1 12]
[[9999    2    3    4]
 [   5    6    7    8]
 [   9   10   11 9999]]
```

3.2.3 Boolean indexing

[46]:
```
# Numpy array elements can also be accessed
# by using a boolean array of the same dimension:

# Once again, both Python lists and numpy arrays work fine:
mask_py = [False, True, False, True, False, True]
mask_np = np.array(mask_py)
```

```python
print(vector[mask_py])
print(vector[mask_np])
```

```
[2 4 6]
[2 4 6]
```

[47]:
```python
# Boolean indexing is most often used
# together with comparison operators:

matrix = np.array([
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
])

# Using numpy array with some comparison operator
# will return boolean array of the same dimension.
mask = matrix > 5
print(mask)
print(matrix[mask])
```

```
[[False False False False]
 [False  True  True  True]
 [ True  True  True  True]]
[ 6  7  8  9 10 11 12]
```

[48]:
```python
# In some cases you might want to use comparison operator,
# but get integer indices, instead of boolean mask.

# np.where() can be used in such case:
indices = np.where(matrix > 5)

print(indices)
print(matrix[indices])
```

```
(array([1, 1, 1, 2, 2, 2, 2]), array([1, 2, 3, 0, 1, 2, 3]))
[ 6  7  8  9 10 11 12]
```

3.3 Array manipulation

3.3.1 Sorting

[49]:
```python
# Simplest way of sorting arrays:

unsorted_vector = np.array([100, 400, 200, 500, 300, 600])

# This happens in-place:
```

```
print(unsorted_vector.sort())
print(unsorted_vector)
```

```
None
[100 200 300 400 500 600]
```

[50]:
```
# Returning the element indices in particular order,
# without actually sorting the array:

unsorted_vector = np.array([100, 400, 200, 500, 300, 600])
order = np.argsort(unsorted_vector)

print(order)
print(unsorted_vector[order])

# Useful when you have more than one array
# and want to sort them all in specific order
```

```
[0 2 4 1 3 5]
[100 200 300 400 500 600]
```

3.3.2 Joining

[51]:
```
# Concatenating arrays:

a1 = np.random.randint(0, 100, (2, 10))
a2 = np.random.randint(0, 100, (3, 10))

# This joins multiple arrays along existing axis:
concatenated = np.concatenate((a1, a2))
concatenated.shape
```

[51]:
```
(5, 10)
```

[52]:
```
# Stacking arrays:

a1 = np.random.randint(0, 100, (10,))
a2 = np.random.randint(0, 100, (10,))

# This joins multiple arrays along a new axis:
stacked = np.stack((a1, a2))
stacked.shape
```

[52]:
```
(2, 10)
```

3.4 Operations

[53]:
```
# Numpy module itself contains a large number
# of various mathematical functions

np.sin(np.pi / 2)
```

[53]:
```
1.0
```

3.4.1 Basic statistics

[54]:
```
# Dealing with 1D arrays is simple:
print('Min:            ', vector.min())
print('Max:            ', vector.max())
print('Sum:            ', vector.sum())
print('Std. deviation: ', vector.std())
print('Variance:       ', vector.var())
print('Mean:           ', vector.mean())
print('Median:         ', np.median(vector))
```

```
Min:             1
Max:             6
Sum:             21
Std. deviation:  1.707825127659933
Variance:        2.9166666666666665
Mean:            3.5
Median:          3.5
```

[55]:
```
# When working with two (or more) dimensional arrays,
# axis can be specified along which the operation is applied:

# This just returns the sum of all elements:
print(matrix.sum())

# This computes sum of each column:
print(matrix.sum(axis=0))

# This computes sum of each row:
print(matrix.sum(axis=1))
```

```
78
[15 18 21 24]
[10 26 42]
```

3.4.2 Linear algebra

[56]:
```
# Matrix transpose:

print(matrix)
print(matrix.T)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]
```

[57]:
```
# Matrix determinant:

square_matrix = np.random.randint(0, 100, (5, 5))
np.linalg.det(square_matrix)
```

[57]:
```
69778666.99999999
```

[58]:
```
# Matrix inverse
np.linalg.inv(square_matrix)
```

[58]:
```
array([[-0.18690518, -0.02720151, -0.0179004 ,  0.03976834,  0.11522754],
       [ 0.11206488,  0.00438082,  0.02232668, -0.0235441 , -0.05994077],
       [ 0.24465062,  0.02752506,  0.01467512, -0.03466365, -0.14244576],
       [ 0.20608372,  0.04424871, -0.00041802, -0.04542914, -0.11083876],
       [-0.17161714, -0.01283308, -0.00796559,  0.02993626,  0.09382266]])
```

[59]:
```
# Matrix multiplication

# Note the dimensions!
m1 = np.random.randint(0, 100, (2, 5))
m2 = np.random.randint(0, 100, (5, 3))

np.dot(m1, m2)
```

[59]:
```
array([[ 6415, 17846, 19828],
       [ 8904, 12509, 19249]])
```

[60]:
```
# Dot product vs. element-size multiplication
# Don't confuse them!
sm1 = np.random.randint(0, 100, (4, 4))
sm2 = np.random.randint(0, 100, (4, 4))

print(np.dot(sm1, sm2))
print(sm1 * sm2)
```

```
[[ 1686  1120  1696  2442]
 [ 5520  9396 11850  3184]
 [ 4628  6897  7228  2746]
 [13398 10071 12743  8937]]
[[ 210  962    0   72]
 [ 696   80 1092  864]
 [  87  237 3822  105]
 [9408 5475  464 5766]]
```

4 Pandas

[61]:
```
import pandas as pd
```

4.1 Data structures

4.1.1 DataFrame

[62]:
```
# DataFrame - a two (or more) dimensional
# data structure, similar to a database table.

# One way of creating pandas dataframe is to
# pass Python dictionary, where its keys represent
# columns and values represent data for each column:

df_demo = pd.DataFrame({
    'some_integers': [1, 2, 3, 4],
    'some_floats': [10.5, 11.5, 12.5, 13.5],
    'some_strings': ['some', 'text', 'goes', 'here'],
})
df_demo
```

[62]:

	some_integers	some_floats	some_strings
0	1	10.5	some
1	2	11.5	text
2	3	12.5	goes
3	4	13.5	here

[63]:
```
# Pandas dataframes can also be created from numpy arrays
# (and vice versa)

# Numpy array -> Pandas dataframe:
df_demo2 = pd.DataFrame(
    np.random.randint(0, 100, (5, 2)),
    columns=['Column1', 'Column2'])

df_demo2
```

[63]:
```
   Column1  Column2
0       69       43
1       36       78
2       28       17
3       73       46
4       98       66
```

[64]:
```
# Pandas dataframe -> numpy array

df_demo2.values
# OR:
df_demo2.to_numpy()
```

[64]:
```
array([[69, 43],
       [36, 78],
       [28, 17],
       [73, 46],
       [98, 66]])
```

4.1.2 Series

[65]:
```
# Series - a one dimensional data structure
# that can store values.

sr_demo = pd.Series([1, 2, 3])
sr_demo
```

[65]:
```
0    1
1    2
2    3
dtype: int64
```

4.2 Input/output

4.2.1 CSV

[66]:
```
# Reading data:

df_countries = pd.read_csv('datasets/countries of the world.csv', decimal=',',␣
↪skipinitialspace=True)

# [IGNORE THIS FOR NOW] Removing trailing whitespace:
df_countries['Region'] = df_countries['Region'].str.strip()
df_countries['Country'] = df_countries['Country'].str.strip()

df_countries
```

[66]:
```
           Country               Region  Population  Area (sq. mi.)  \
0      Afghanistan  ASIA (EX. NEAR EAST)    31056997          647500
```

```
1           Albania       EASTERN EUROPE     3581655            28748
2           Algeria      NORTHERN AFRICA    32930091          2381740
3    American Samoa              OCEANIA       57794              199
4           Andorra       WESTERN EUROPE       71201              468
..              ...                  ...         ...              ...
222       West Bank            NEAR EAST     2460492             5860
223  Western Sahara      NORTHERN AFRICA      273008           266000
224           Yemen            NEAR EAST    21456188           527970
225          Zambia   SUB-SAHARAN AFRICA    11502010           752614
226        Zimbabwe   SUB-SAHARAN AFRICA    12236805           390580

     Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
0                          48.0                          0.00          23.06
1                         124.6                          1.26          -4.93
2                          13.8                          0.04          -0.39
3                         290.4                         58.29         -20.71
4                         152.1                          0.00           6.60
..                          ...                           ...            ...
222                       419.9                          0.00           2.98
223                         1.0                          0.42            NaN
224                        40.6                          0.36           0.00
225                        15.3                          0.00           0.00
226                        31.3                          0.00           0.00

     Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  \
0                                163.07               700.0          36.0
1                                 21.52              4500.0          86.5
2                                 31.00              6000.0          70.0
3                                  9.27              8000.0          97.0
4                                  4.05             19000.0         100.0
..                                  ...                 ...           ...
222                               19.62               800.0           NaN
223                                 NaN                 NaN           NaN
224                               61.50               800.0          50.2
225                               88.29               800.0          80.6
226                               67.69              1900.0          90.7

     Phones (per 1000)  Arable (%)  Crops (%)  Other (%)  Climate  Birthrate  \
0                  3.2       12.13       0.22      87.65      1.0      46.60
1                 71.2       21.09       4.42      74.49      3.0      15.11
2                 78.1        3.22       0.25      96.53      1.0      17.14
3                259.5       10.00      15.00      75.00      2.0      22.46
4                497.2        2.22       0.00      97.78      3.0       8.71
..                 ...         ...        ...        ...      ...        ...
222              145.2       16.90      18.97      64.13      3.0      31.67
223                NaN        0.02       0.00      99.98      1.0        NaN
224               37.2        2.78       0.24      96.98      1.0      42.89
```

```
225                8.2        7.08       0.03       92.90      2.0       41.00
226               26.8        8.32       0.34       91.34      2.0       28.01

     Deathrate  Agriculture  Industry  Service
0        20.34        0.380     0.240    0.380
1         5.22        0.232     0.188    0.579
2         4.61        0.101     0.600    0.298
3         3.27          NaN       NaN      NaN
4         6.25          NaN       NaN      NaN
..         ...          ...       ...      ...
222       3.92        0.090     0.280    0.630
223        NaN          NaN       NaN    0.400
224       8.30        0.135     0.472    0.393
225      19.93        0.220     0.290    0.489
226      21.84        0.179     0.243    0.579

[227 rows x 20 columns]
```

4.2.2 JSON

[67]:
```python
# Reading data:
# 'lines=True' means that one line contains one data sample in JSON format
df_nyt2 = pd.read_json('datasets/Sarcasm_Headlines_Dataset.json', lines=True)
df_nyt2
```

[67]:
```
                                            article_link  \
0      https://www.huffingtonpost.com/entry/versace-b...
1      https://www.huffingtonpost.com/entry/roseanne-...
2      https://local.theonion.com/mom-starting-to-fea...
3      https://politics.theonion.com/boehner-just-wan...
4      https://www.huffingtonpost.com/entry/jk-rowlin...
...                                                  ...
26704  https://www.huffingtonpost.com/entry/american-...
26705  https://www.huffingtonpost.com/entry/americas-...
26706  https://www.huffingtonpost.com/entry/reparatio...
26707  https://www.huffingtonpost.com/entry/israeli-b...
26708  https://www.huffingtonpost.com/entry/gourmet-g...

                                                headline  is_sarcastic
0      former versace store clerk sues over secret 'b...             0
1      the 'roseanne' revival catches up to our thorn...             0
2      mom starting to fear son's web series closest ...             1
3      boehner just wants wife to listen, not come up...             1
4      j.k. rowling wishes snape happy birthday in th...             0
...                                                  ...           ...
26704               american politics in moral free-fall             0
26705                            america's best 20 hikes             0
```

```
26706                           reparations and obama          0
26707  israeli ban targeting boycott supporters raise...          0
26708                     gourmet gifts for the foodie 2014          0

[26709 rows x 3 columns]
```

4.2.3 SQL

[68]:
```
# Reading data:
import sqlite3

conn = sqlite3.connect('datasets/iris.sqlite')
df_iris = pd.read_sql_query('SELECT * FROM Iris', conn)
df_iris
```

[68]:
```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

4.3 Viewing data

[69]:
```
# Getting several first rows:
df_countries.head(3)
```

[69]:
```
       Country                Region  Population  Area (sq. mi.)  \
0  Afghanistan  ASIA (EX. NEAR EAST)    31056997          647500
1      Albania        EASTERN EUROPE     3581655           28748
2      Algeria       NORTHERN AFRICA    32930091         2381740

   Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
0                        48.0                          0.00          23.06
1                       124.6                          1.26          -4.93
2                        13.8                          0.04          -0.39

   Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  \
0                              163.07               700.0          36.0
1                               21.52              4500.0          86.5
2                               31.00              6000.0          70.0

   Phones (per 1000)  Arable (%)  Crops (%)  Other (%)  Climate  Birthrate  \
0                3.2       12.13       0.22      87.65      1.0      46.60
1               71.2       21.09       4.42      74.49      3.0      15.11
2               78.1        3.22       0.25      96.53      1.0      17.14

   Deathrate  Agriculture  Industry  Service
0      20.34        0.380     0.240    0.380
1       5.22        0.232     0.188    0.579
2       4.61        0.101     0.600    0.298
```

[70]:
```
# Getting several last rows:
df_countries.tail(3)
```

[70]:
```
       Country              Region  Population  Area (sq. mi.)  \
224      Yemen           NEAR EAST    21456188          527970
225     Zambia  SUB-SAHARAN AFRICA    11502010          752614
226   Zimbabwe  SUB-SAHARAN AFRICA    12236805          390580

     Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
224                        40.6                          0.36            0.0
225                        15.3                          0.00            0.0
226                        31.3                          0.00            0.0

     Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  \
224                               61.50               800.0          50.2
225                               88.29               800.0          80.6
226                               67.69              1900.0          90.7

     Phones (per 1000)  Arable (%)  Crops (%)  Other (%)  Climate  Birthrate  \
```

```
224              37.2        2.78        0.24        96.98       1.0        42.89
225               8.2        7.08        0.03        92.90       2.0        41.00
226              26.8        8.32        0.34        91.34       2.0        28.01

     Deathrate  Agriculture  Industry  Service
224       8.30        0.135     0.472    0.393
225      19.93        0.220     0.290    0.489
226      21.84        0.179     0.243    0.579
```

[71]:
```
# Shape attribute is available, as in numpy arrays:
df_countries.shape
```

[71]:
```
(227, 20)
```

[72]:
```
# Getting column list:
df_countries.columns
```

[72]:
```
Index(['Country', 'Region', 'Population', 'Area (sq. mi.)',
       'Pop. Density (per sq. mi.)', 'Coastline (coast/area ratio)',
       'Net migration', 'Infant mortality (per 1000 births)',
       'GDP ($ per capita)', 'Literacy (%)', 'Phones (per 1000)', 'Arable (%)',
       'Crops (%)', 'Other (%)', 'Climate', 'Birthrate', 'Deathrate',
       'Agriculture', 'Industry', 'Service'],
      dtype='object')
```

[73]:
```
# Dataframe summary:
df_countries.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 20 columns):
Country                               227 non-null object
Region                                227 non-null object
Population                            227 non-null int64
Area (sq. mi.)                        227 non-null int64
Pop. Density (per sq. mi.)            227 non-null float64
Coastline (coast/area ratio)          227 non-null float64
Net migration                         224 non-null float64
Infant mortality (per 1000 births)    224 non-null float64
GDP ($ per capita)                    226 non-null float64
Literacy (%)                          209 non-null float64
Phones (per 1000)                     223 non-null float64
Arable (%)                            225 non-null float64
Crops (%)                             225 non-null float64
Other (%)                             225 non-null float64
Climate                               205 non-null float64
Birthrate                             224 non-null float64
Deathrate                             223 non-null float64
Agriculture                           212 non-null float64
Industry                              211 non-null float64
```

```
Service                             212 non-null float64
dtypes: float64(16), int64(2), object(2)
memory usage: 35.6+ KB
```

[74]:
```
# Basic statistics of all columns
df_countries.describe()
```

[74]:
```
         Population  Area (sq. mi.)  Pop. Density (per sq. mi.)  \
count  2.270000e+02    2.270000e+02                  227.000000
mean   2.874028e+07    5.982270e+05                  379.047137
std    1.178913e+08    1.790282e+06                 1660.185825
min    7.026000e+03    2.000000e+00                    0.000000
25%    4.376240e+05    4.647500e+03                   29.150000
50%    4.786994e+06    8.660000e+04                   78.800000
75%    1.749777e+07    4.418110e+05                  190.150000
max    1.313974e+09    1.707520e+07                16271.500000

       Coastline (coast/area ratio)  Net migration  \
count                    227.000000     224.000000
mean                      21.165330       0.038125
std                       72.286863       4.889269
min                        0.000000     -20.990000
25%                        0.100000      -0.927500
50%                        0.730000       0.000000
75%                       10.345000       0.997500
max                      870.660000      23.060000

       Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  \
count                          224.000000          226.000000    209.000000
mean                            35.506964         9689.823009     82.838278
std                             35.389899        10049.138513     19.722173
min                              2.290000          500.000000     17.600000
25%                              8.150000         1900.000000     70.600000
50%                             21.000000         5550.000000     92.500000
75%                             55.705000        15700.000000     98.000000
max                            191.190000        55100.000000    100.000000

       Phones (per 1000)  Arable (%)   Crops (%)   Other (%)     Climate  \
count         223.000000  225.000000  225.000000  225.000000  205.000000
mean          236.061435   13.797111    4.564222   81.638311    2.139024
std           227.991829   13.040402    8.361470   16.140835    0.699397
min             0.200000    0.000000    0.000000   33.330000    1.000000
25%            37.800000    3.220000    0.190000   71.650000    2.000000
50%           176.200000   10.420000    1.030000   85.700000    2.000000
75%           389.650000   20.000000    4.440000   95.440000    3.000000
max          1035.600000   62.110000   50.680000  100.000000    4.000000

        Birthrate   Deathrate  Agriculture    Industry     Service
```

```
count  224.000000  223.000000   212.000000  211.000000  212.000000
mean    22.114732    9.241345     0.150844    0.282711    0.565283
std     11.176716    4.990026     0.146798    0.138272    0.165841
min      7.290000    2.290000     0.000000    0.020000    0.062000
25%     12.672500    5.910000     0.037750    0.193000    0.429250
50%     18.790000    7.840000     0.099000    0.272000    0.571000
75%     29.820000   10.605000     0.221000    0.341000    0.678500
max     50.730000   29.740000     0.769000    0.906000    0.954000
```

[75]:
```
# Basic statistics for selected column:

print('Count:          ', df_countries['Population'].count())
print('Sum:            ', df_countries['Population'].sum())
print('Mean:           ', df_countries['Population'].mean())
print('Median:         ', df_countries['Population'].median())
print('Max:            ', df_countries['Population'].max())
print('Min:            ', df_countries['Population'].min())
print('Std. deviation: ', df_countries['Population'].std())
print('Variance:       ', df_countries['Population'].var())
```

```
Count:           227
Sum:             6524044551
Mean:            28740284.365638766
Median:          4786994.0
Max:             1313973713
Min:             7026
Std. deviation:  117891326.54347652
Variance:        1.3898364874180612e+16
```

4.4 Selecting data

4.4.1 Columns

[76]:
```
# Selecting a single column: returns a Series object

# Column can be accessed as an attribute or by dictionary key:
print(type(df_countries.Country))
print(type(df_countries['Country']))

df_countries['Country']
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[76]:
```
0          Afghanistan
1              Albania
2              Algeria
3       American Samoa
```

```
4             Andorra
                ...
222         West Bank
223    Western Sahara
224             Yemen
225            Zambia
226          Zimbabwe
Name: Country, Length: 227, dtype: object
```

[77]:
```
# Selecting multiple columns: returns a DataFrame object

# Note the double brackets!
df_countries[['Country', 'Region', 'Population']]
```

[77]:
```
            Country               Region  Population
0       Afghanistan  ASIA (EX. NEAR EAST)    31056997
1           Albania       EASTERN EUROPE     3581655
2           Algeria      NORTHERN AFRICA    32930091
3    American Samoa              OCEANIA       57794
4           Andorra       WESTERN EUROPE       71201
..              ...                  ...         ...
222       West Bank            NEAR EAST     2460492
223  Western Sahara      NORTHERN AFRICA      273008
224           Yemen            NEAR EAST    21456188
225          Zambia   SUB-SAHARAN AFRICA    11502010
226        Zimbabwe   SUB-SAHARAN AFRICA    12236805

[227 rows x 3 columns]
```

4.4.2 Rows

Two DataFrame attributes for selecting rows: * .loc - selects by index/label or boolean expression * .iloc - selects by integer indices (slices)

[78]:
```
# Selecting rows by name



# .loc[<row_label(s)>, <column_label(s)>]

# Before using .loc, make sure that DataFrame has index!
df_countries_indexed = df_countries.set_index('Country')

# Single row / single column
print(df_countries_indexed.loc['Lithuania', 'Area (sq. mi.)'])

# Multiple rows / multiple columns
print(df_countries_indexed.loc[['Lithuania', 'Latvia'], ['Population', 'Area␣
↪(sq. mi.)']])
```

```
65200
           Population  Area (sq. mi.)
Country
Lithuania     3585906           65200
Latvia        2274735           64589
```

[79]:
```python
# Selecting rows by name

# .loc attribute can also be used with a boolean array,
# similar to boolean masks in numpy arrays

df_countries.loc[df_countries['Region'] == 'BALTICS']
```

[79]:
```
       Country   Region  Population  Area (sq. mi.)  \
64     Estonia  BALTICS     1324333           45226
114     Latvia  BALTICS     2274735           64589
120  Lithuania  BALTICS     3585906           65200

     Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
64                         29.3                          8.39          -3.16
114                        35.2                          0.82          -2.23
120                        55.0                          0.14          -0.71

     Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  \
64                                 7.87             12300.0          99.8
114                                9.55             10200.0          99.8
120                                6.89             11400.0          99.6

     Phones (per 1000)  Arable (%)  Crops (%)  Other (%)  Climate  Birthrate  \
64               333.8       16.04       0.45      83.51      3.0      10.04
114              321.4       29.67       0.47      69.86      3.0       9.24
120              223.4       45.22       0.91      53.87      NaN       8.75

     Deathrate  Agriculture  Industry  Service
64       13.25        0.040     0.294    0.666
114      13.66        0.040     0.261    0.699
120      10.98        0.055     0.325    0.620
```

[80]:
```python
# Selecting rows by numerical index

# Usual slicing syntax:
df_countries.iloc[10:15]
```

[80]:
```
        Country               Region  Population  Area (sq. mi.)  \
10        Aruba  LATIN AMER. & CARIB       71891             193
11    Australia              OCEANIA    20264082         7686850
12      Austria       WESTERN EUROPE     8192880           83870
13   Azerbaijan  C.W. OF IND. STATES     7961619           86600
14  Bahamas, The  LATIN AMER. & CARIB      303770           13940
```

```
    Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
10                       372.5                         35.49           0.00
11                         2.6                          0.34           3.98
12                        97.7                          0.00           2.00
13                        91.9                          0.00          -4.90
14                        21.8                         25.41          -2.20

    Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  \
10                                5.89             28000.0          97.0
11                                4.69             29000.0         100.0
12                                4.66             30000.0          98.0
13                               81.74              3400.0          97.0
14                               25.21             16700.0          95.6

    Phones (per 1000)  Arable (%)  Crops (%)  Other (%)  Climate  Birthrate  \
10              516.1       10.53       0.00      89.47      2.0      11.03
11              565.5        6.55       0.04      93.41      1.0      12.14
12              452.2       16.91       0.86      82.23      3.0       8.74
13              137.1       19.63       2.71      77.66      1.0      20.74
14              460.6        0.80       0.40      98.80      2.0      17.57

    Deathrate  Agriculture  Industry  Service
10       6.68        0.004     0.333    0.663
11       7.51        0.038     0.262    0.700
12       9.76        0.018     0.304    0.678
13       9.75        0.141     0.457    0.402
14       9.05        0.030     0.070    0.900
```

4.5 Manipulating data

[81]:
```
# Applying functions

# Add a new column to dataframe:
df_countries['CountryUppercase'] = df_countries['Country'].apply(lambda s: s.
 ↪upper())
df_countries[['Country', 'CountryUppercase']]
```

[81]:
```
             Country CountryUppercase
0        Afghanistan      AFGHANISTAN
1            Albania          ALBANIA
2            Algeria          ALGERIA
3     American Samoa   AMERICAN SAMOA
4            Andorra          ANDORRA
..               ...              ...
222        West Bank        WEST BANK
223   Western Sahara   WESTERN SAHARA
```

```
224          Yemen            YEMEN
225         Zambia           ZAMBIA
226       Zimbabwe         ZIMBABWE

[227 rows x 2 columns]
```

[82]:
```
# Grouping
df_countries.groupby(['Region']).sum()['Population']
```

[82]:
```
Region
ASIA (EX. NEAR EAST)    3687982236
BALTICS                    7184974
C.W. OF IND. STATES      280081548
EASTERN EUROPE           119914717
LATIN AMER. & CARIB      561824599
NEAR EAST                195068377
NORTHERN AFRICA          161407133
NORTHERN AMERICA         331672307
OCEANIA                   33131662
SUB-SAHARAN AFRICA       749437000
WESTERN EUROPE           396339998
Name: Population, dtype: int64
```

[83]:
```
# Merging dataframes
# (equivalent of joins in SQL)

df_countries_short = pd.DataFrame({
    'Country':      ['Germany', 'France', 'Italy'],
    'CountryShort': ['DE', 'FR', 'IT'],
})
print(df_countries_short)

# Inner join
df_countries.merge(df_countries_short)

# Other types of joins (i.e. left outer) can be used
# by providing 'how' keyword argument.
```

```
   Country CountryShort
0  Germany           DE
1   France           FR
2    Italy           IT
```

[83]:
```
   Country          Region  Population  Area (sq. mi.)  \
0   France  WESTERN EUROPE    60876136          547030
1  Germany  WESTERN EUROPE    82422299          357021
2    Italy  WESTERN EUROPE    58133509          301230

   Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
```

```
0                111.3                              0.63          0.66
1                230.9                              0.67          2.18
2                193.0                              2.52          2.07

   Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  ...  \
0                                4.26             27600.0          99.0  ...
1                                4.16             27600.0          99.0  ...
2                                5.94             26700.0          98.6  ...

   Crops (%)  Other (%)  Climate  Birthrate  Deathrate  Agriculture  Industry  \
0       2.07      64.40      4.0      11.99       9.14        0.022     0.214
1       0.59      65.56      3.0       8.25      10.62        0.009     0.296
2       9.53      62.68      NaN       8.72      10.40        0.021     0.291

   Service  CountryUppercase  CountryShort
0    0.764            FRANCE            FR
1    0.695           GERMANY            DE
2    0.688             ITALY            IT

[3 rows x 22 columns]
```

[84]:
```python
# Sorting

df_countries.sort_values('Population', ascending=False).head(5)
# Note that the index values are also reordered!
```

[84]:
```
            Country               Region  Population  Area (sq. mi.)  \
42            China  ASIA (EX. NEAR EAST)  1313973713         9596960
94            India  ASIA (EX. NEAR EAST)  1095351995         3287590
214   United States      NORTHERN AMERICA   298444215         9631420
95        Indonesia  ASIA (EX. NEAR EAST)   245452739         1919440
27           Brazil   LATIN AMER. & CARIB   188078227         8511965

     Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
42                        136.9                          0.15          -0.40
94                        333.2                          0.21          -0.07
214                        31.0                          0.21           3.41
95                        127.9                          2.85           0.00
27                         22.1                          0.09          -0.03

     Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  \
42                                24.18              5000.0          90.9
94                                56.29              2900.0          59.5
214                                6.50             37800.0          97.0
95                                35.60              3200.0          87.9
27                                29.61              7600.0          86.4

     ...  Arable (%)  Crops (%)  Other (%)  Climate  Birthrate  Deathrate  \
```

```
42   ...     15.40     1.25     83.35     1.5     13.25     6.97
94   ...     54.40     2.74     42.86     2.5     22.01     8.18
214  ...     19.13     0.22     80.65     3.0     14.14     8.26
95   ...     11.32     7.23     81.45     2.0     20.34     6.25
27   ...      6.96     0.90     92.15     2.0     16.56     6.17

     Agriculture  Industry  Service  CountryUppercase
42         0.125     0.473    0.403             CHINA
94         0.186     0.276    0.538             INDIA
214        0.010     0.204    0.787     UNITED STATES
95         0.134     0.458    0.408         INDONESIA
27         0.084     0.400    0.516            BRAZIL

[5 rows x 21 columns]
```

[85]:

```python
# In certain cases it might be necessary to reset index
# after sorting or other dataframe manipulation operations:

df_countries.sort_values('Population', ascending=False).reset_index(drop=True).
 ↪head(5)
# Note the updated index
```

[85]:

```
         Country               Region  Population  Area (sq. mi.)  \
0          China  ASIA (EX. NEAR EAST)  1313973713         9596960
1          India  ASIA (EX. NEAR EAST)  1095351995         3287590
2  United States      NORTHERN AMERICA   298444215         9631420
3      Indonesia  ASIA (EX. NEAR EAST)   245452739         1919440
4         Brazil    LATIN AMER. & CARIB   188078227         8511965

   Pop. Density (per sq. mi.)  Coastline (coast/area ratio)  Net migration  \
0                       136.9                          0.15          -0.40
1                       333.2                          0.21          -0.07
2                        31.0                          0.21           3.41
3                       127.9                          2.85           0.00
4                        22.1                          0.09          -0.03

   Infant mortality (per 1000 births)  GDP ($ per capita)  Literacy (%)  ...  \
0                               24.18              5000.0          90.9  ...
1                               56.29              2900.0          59.5  ...
2                                6.50             37800.0          97.0  ...
3                               35.60              3200.0          87.9  ...
4                               29.61              7600.0          86.4  ...

   Arable (%)  Crops (%)  Other (%)  Climate  Birthrate  Deathrate  \
0       15.40       1.25      83.35      1.5      13.25       6.97
1       54.40       2.74      42.86      2.5      22.01       8.18
2       19.13       0.22      80.65      3.0      14.14       8.26
3       11.32       7.23      81.45      2.0      20.34       6.25
```

```
4         6.96        0.90       92.15       2.0        16.56        6.17

   Agriculture  Industry  Service  CountryUppercase
0        0.125     0.473    0.403             CHINA
1        0.186     0.276    0.538             INDIA
2        0.010     0.204    0.787     UNITED STATES
3        0.134     0.458    0.408         INDONESIA
4        0.084     0.400    0.516            BRAZIL

[5 rows x 21 columns]
```

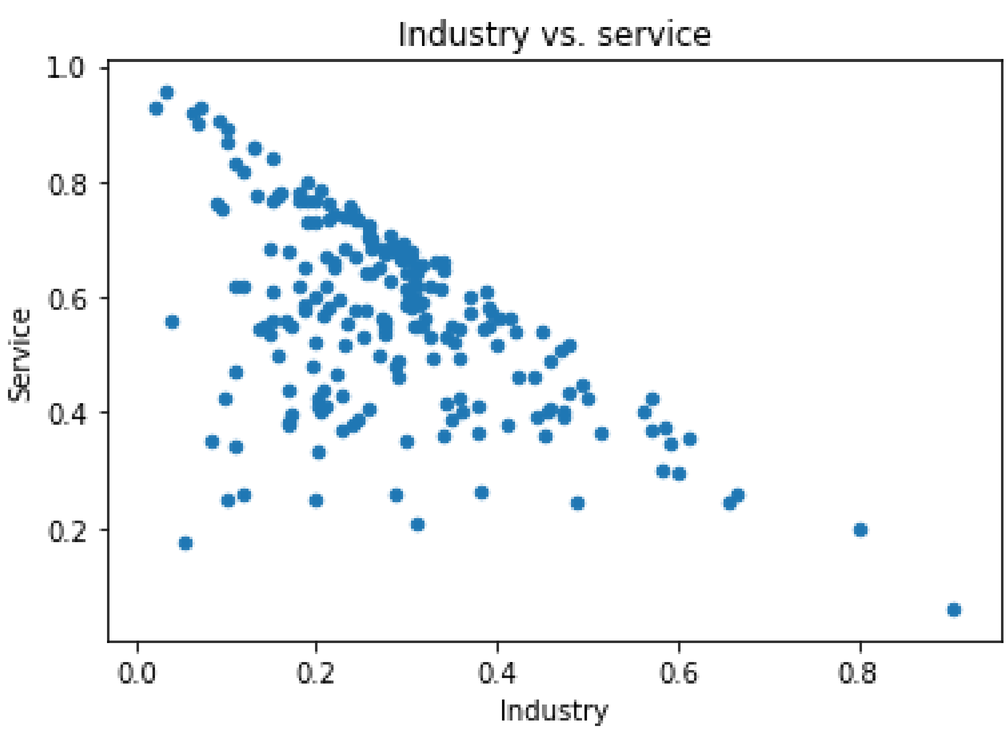
4.6 Basic plots

[86]:
```python
# Line plots
df_countries_sorted = df_countries.sort_values('Industry').reset_index()
df_countries_sorted[['Industry', 'Service']].plot(title='Industry vs. service')
```

[86]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9c22d0978>
```

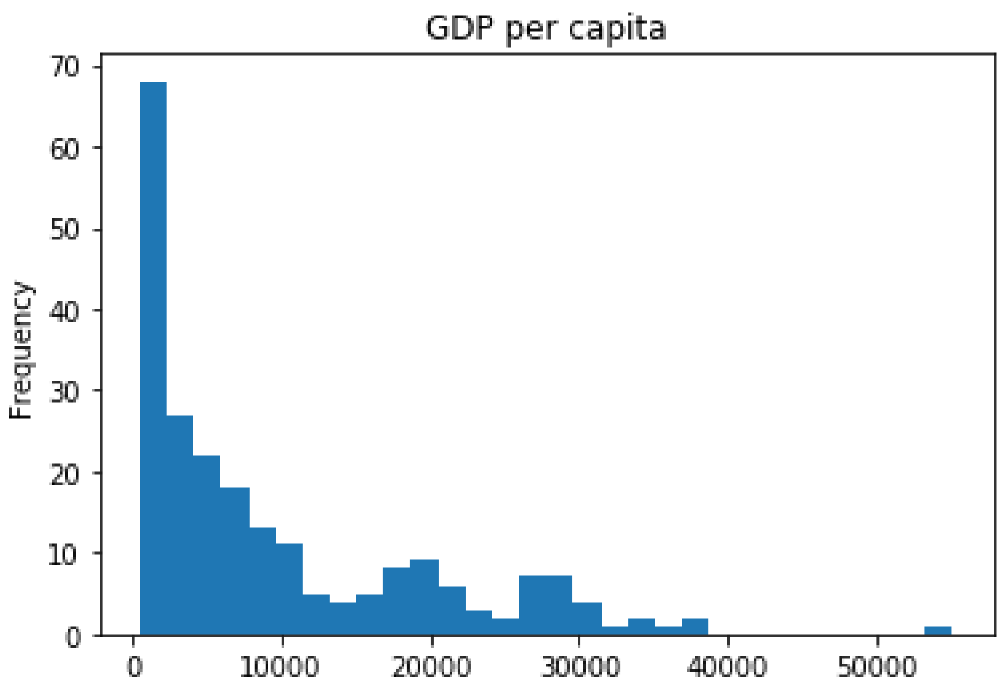
[87]:
```python
# Scatter plots
df_countries.plot(kind='scatter', x='Industry', y='Service', title='Industry vs.
 ↪ service')
```

[87]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9c25f58d0>
```



[88]:
```
# Histograms
df_countries['GDP ($ per capita)'].plot(kind='hist', bins=30, title='GDP per␣
↪capita')
```

[88]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe9c21ecf98>



5 Matplotlib

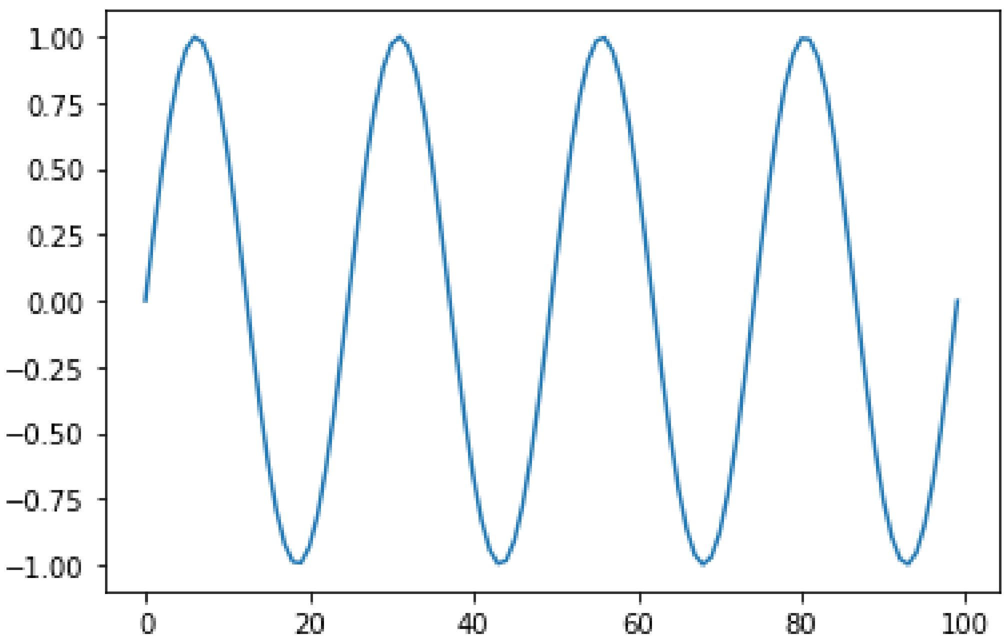
https://matplotlib.org/

5.1 Basics

[89]:
```
# Generate some data first:
import numpy as np

x = np.linspace(-4 * np.pi, 4 * np.pi, 100)
wave1 = np.sin(x)
wave2 = np.cos(x)

print(wave1.shape)
print(wave2.shape)
```

```
(100,)
(100,)
```

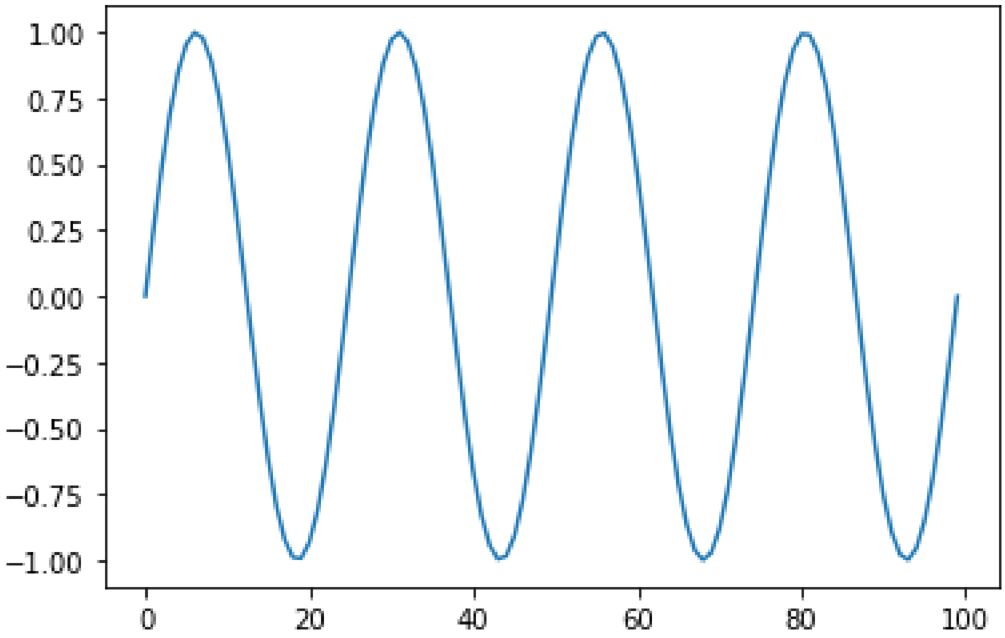
[90]:
```python
# The standard way of importing matplotlib:
import matplotlib.pyplot as plt

# The simplest plot:
plt.plot(wave1)

# The following is not strictly required in Jupyter notebooks,
# but is important in scripts or Python interpreter,
# as it opens a window containing the figure.
plt.show()
```



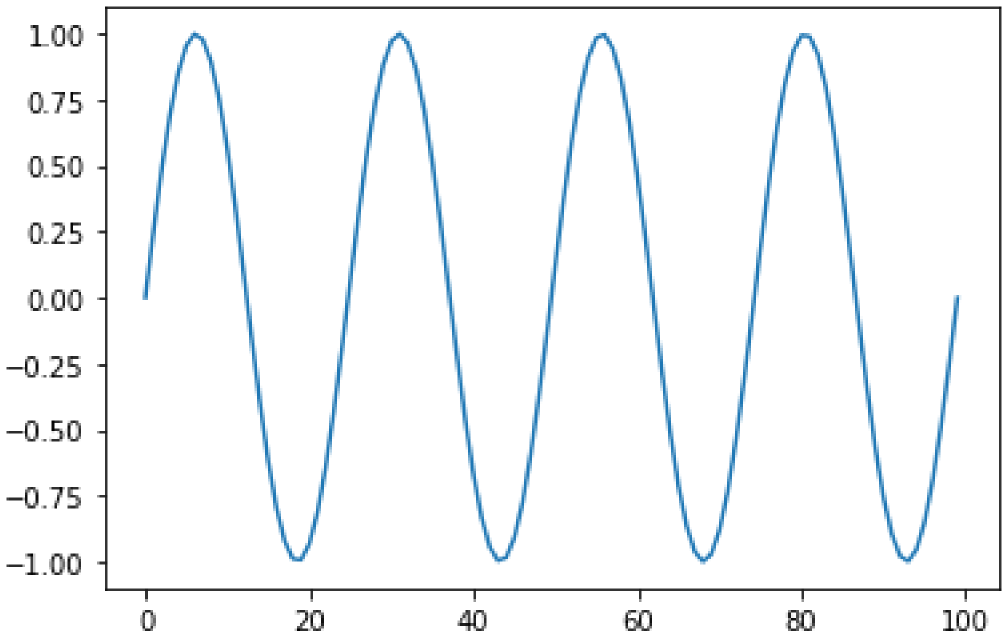
[91]:
```python
# Matplotlib provides two interfaces: stateful and stateless.

# Stateful interface (based on MATLAB) is used by accessing
# 'pyplot' module.

# Stateless interface is object-oriented and is used
# by calling methods on figure and axis objects.
# Example:

fig, ax = plt.subplots()
ax.plot(wave1)
fig.show()
```

```
/home/tomas/.pyenv/versions/3.7.2/lib/python3.7/site-
packages/ipykernel_launcher.py:12: UserWarning: Matplotlib is currently using
```

```
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  if sys.path[0] == '':
```

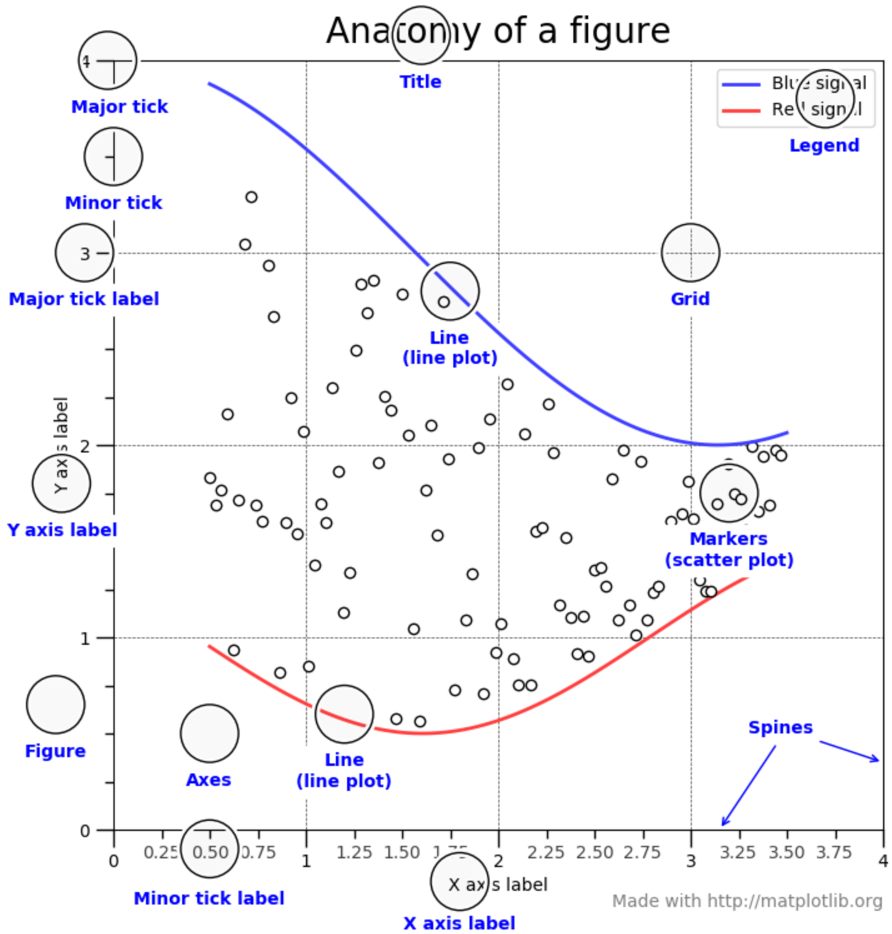


[92]:
```
# Saving a figure to a file:

plt.figure()
plt.plot(wave1)
plt.savefig('test.png')
```



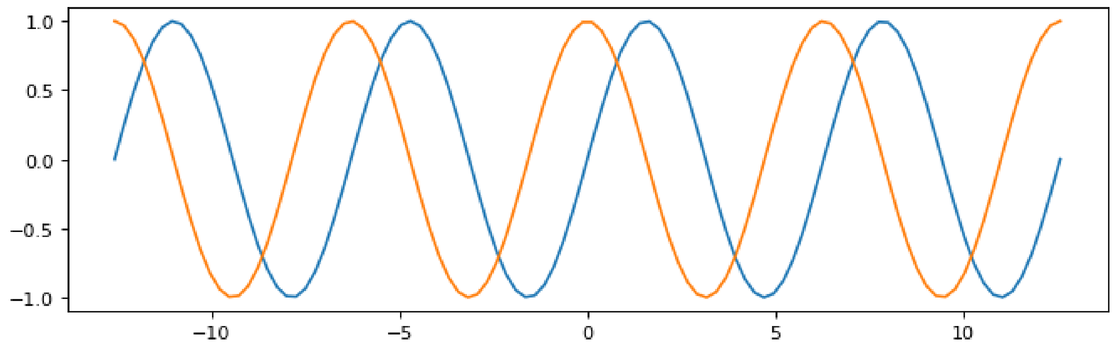
5.2 Plot customizations

Matplotlib figure components:

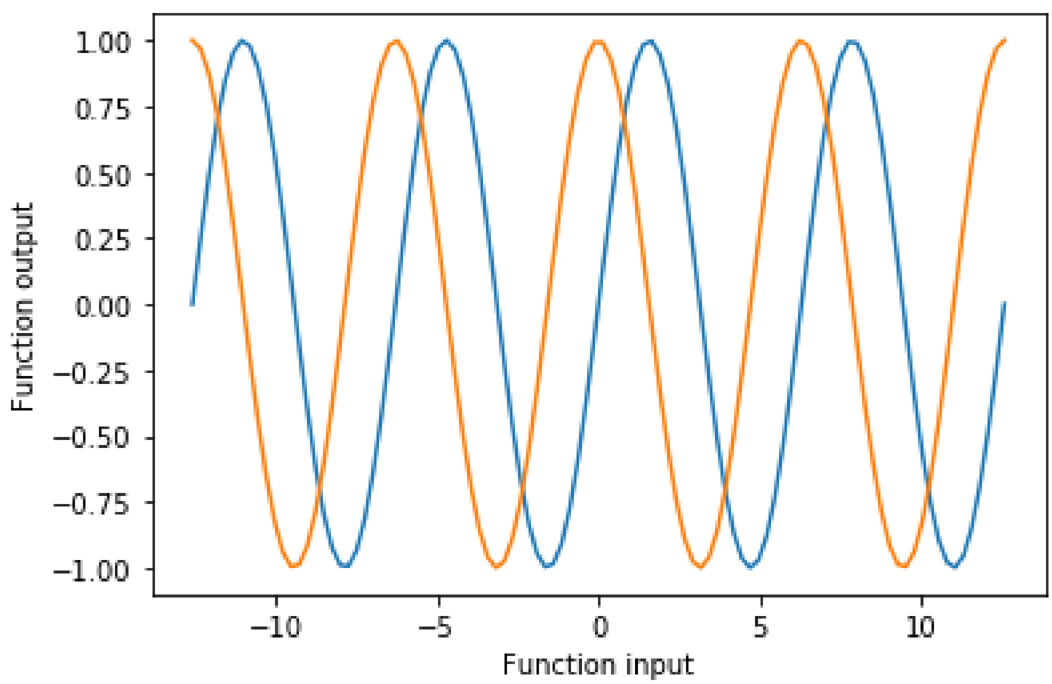


anatomy.png

[93]:
```
# Starting with two simple line plots:

# Figure size (in inches) optionally specified by 'figsize':
# DPI - dots per inch
plt.figure(figsize=(10, 3), dpi=80)

# plot() function can take 2 positional arguments,
# instead of 1, as seen previously.
# In this case, first argument represent X values,
# the second one - Y values

# Multiple line plots can be put drawn in one figure:
plt.plot(x, wave1)
plt.plot(x, wave2)
plt.show()
```

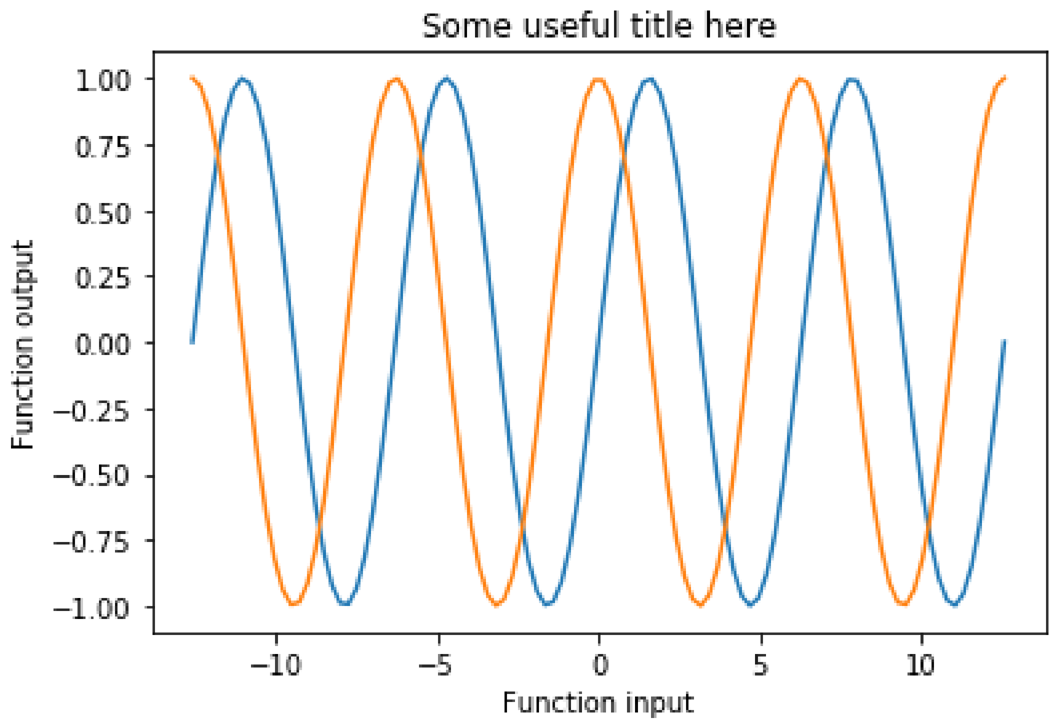


[94]:
```
# Axis labels

plt.figure()
plt.plot(x, wave1)
plt.plot(x, wave2)
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.show()
```

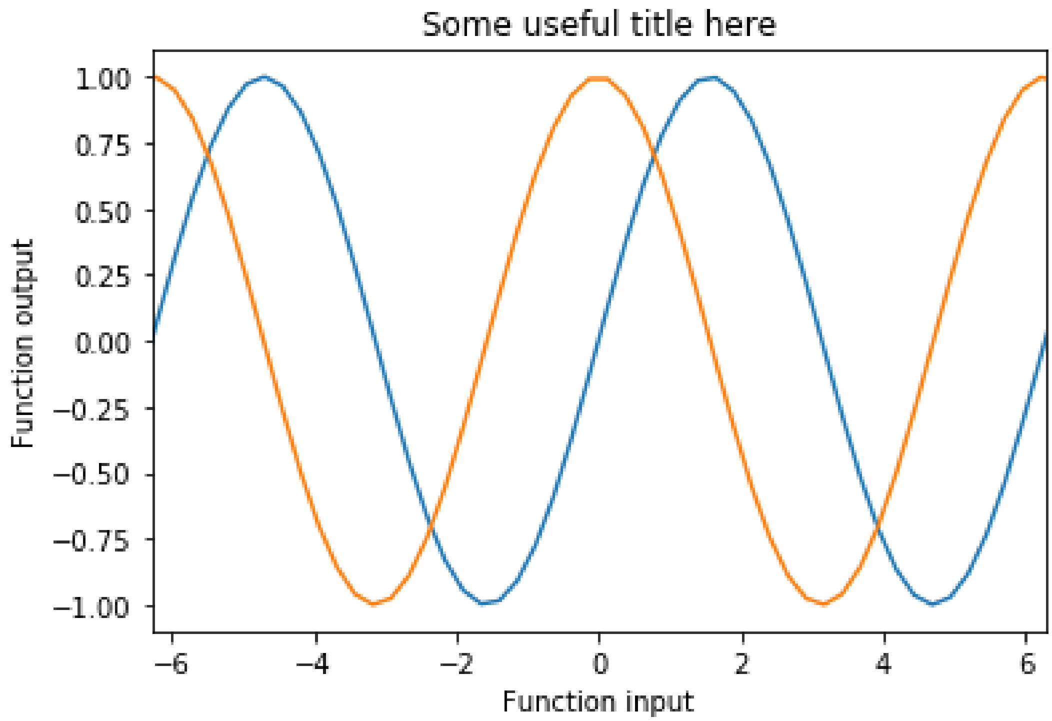


[95]:
```
# Plot title

plt.figure()
```

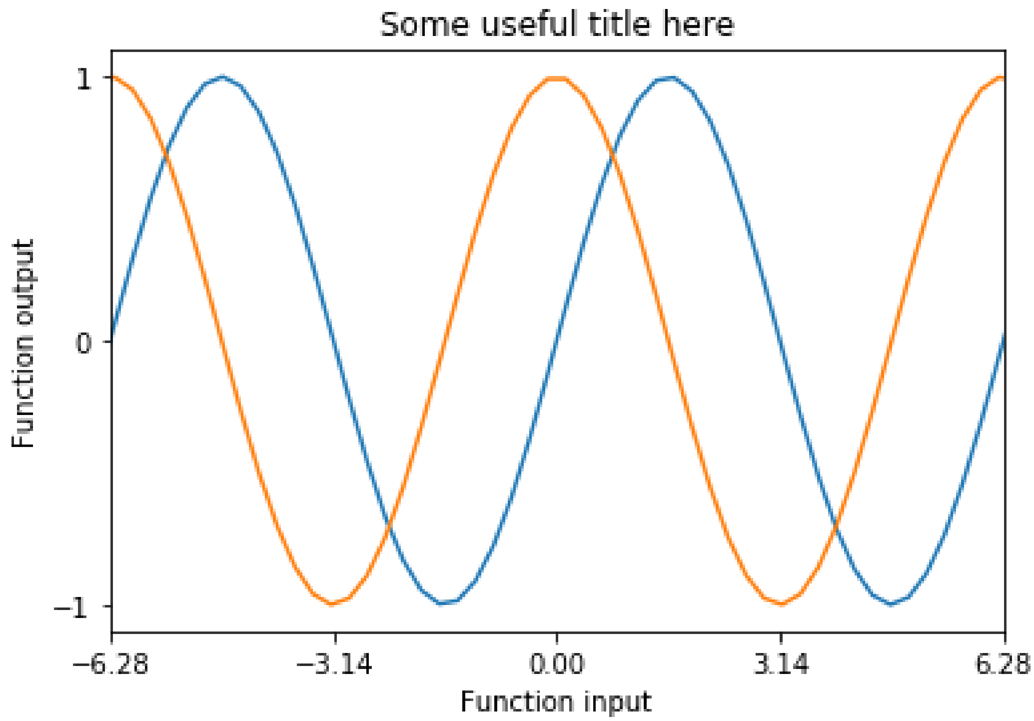
```python
plt.plot(x, wave1)
plt.plot(x, wave2)
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
plt.show()
```



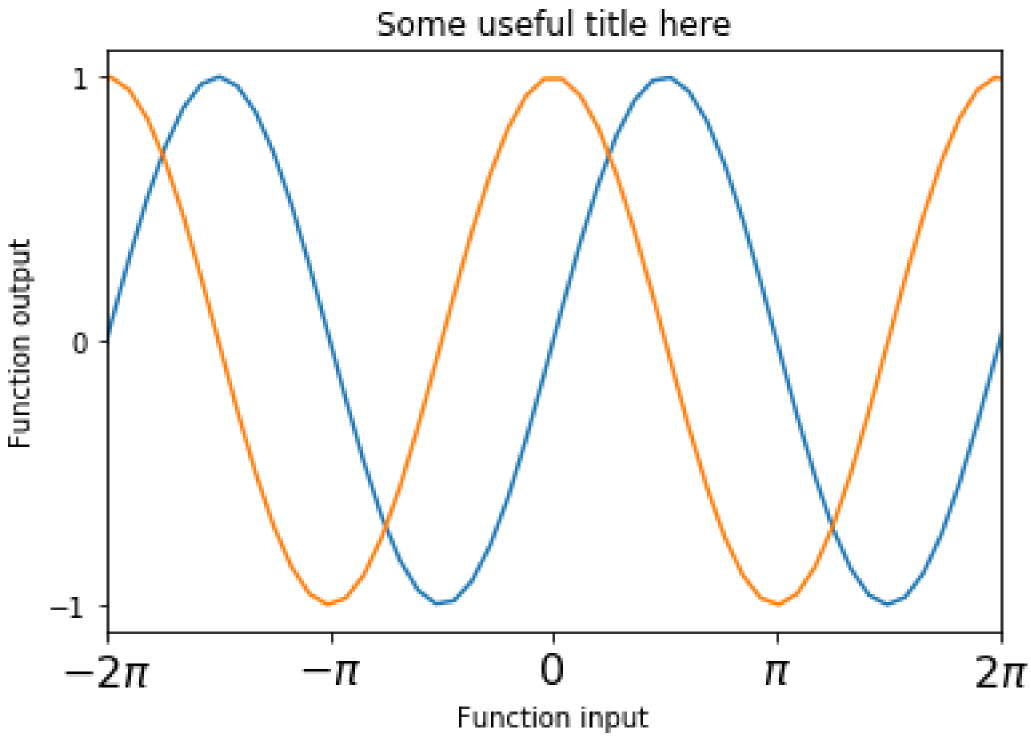
[96]:
```python
# Axis limits

plt.figure()
plt.plot(x, wave1)
plt.plot(x, wave2)
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
# 2-tuples passed here:
plt.xlim((-np.pi * 2, np.pi * 2))
# Use plt.ylim() to set limits to Y axis
plt.show()
```



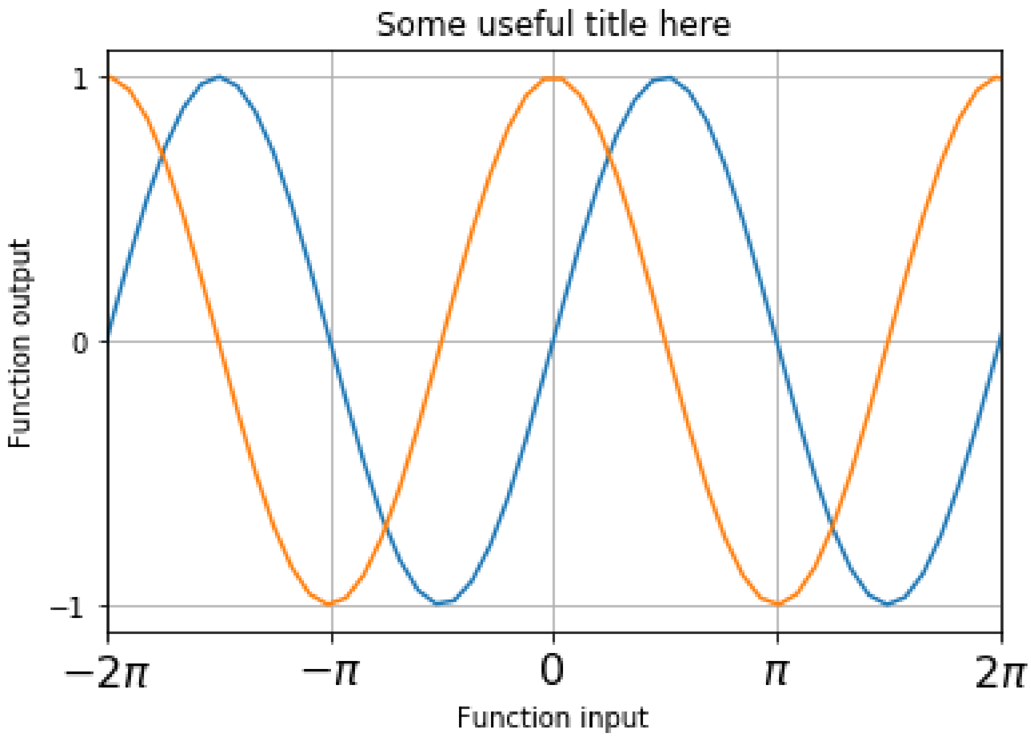
[97]:
```
# Axis ticks

plt.figure()
plt.plot(x, wave1)
plt.plot(x, wave2)
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
plt.xlim((-np.pi * 2, np.pi * 2))
plt.xticks([-np.pi * 2, -np.pi, 0, np.pi, np.pi * 2])
plt.yticks([-1, 0, 1])
plt.show()
```



[98]:
```
# Axis tick labels

plt.figure()
plt.plot(x, wave1)
plt.plot(x, wave2)
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
plt.xlim((-np.pi * 2, np.pi * 2))
plt.xticks(
    [-np.pi * 2, -np.pi, 0, np.pi, np.pi * 2],
    # Custom tick labels:
    ['$-2\pi$', '$-\pi$', '0', '$\pi$', '$2\pi$'],
    fontsize=16)
plt.yticks([-1, 0, 1])
plt.show()
```

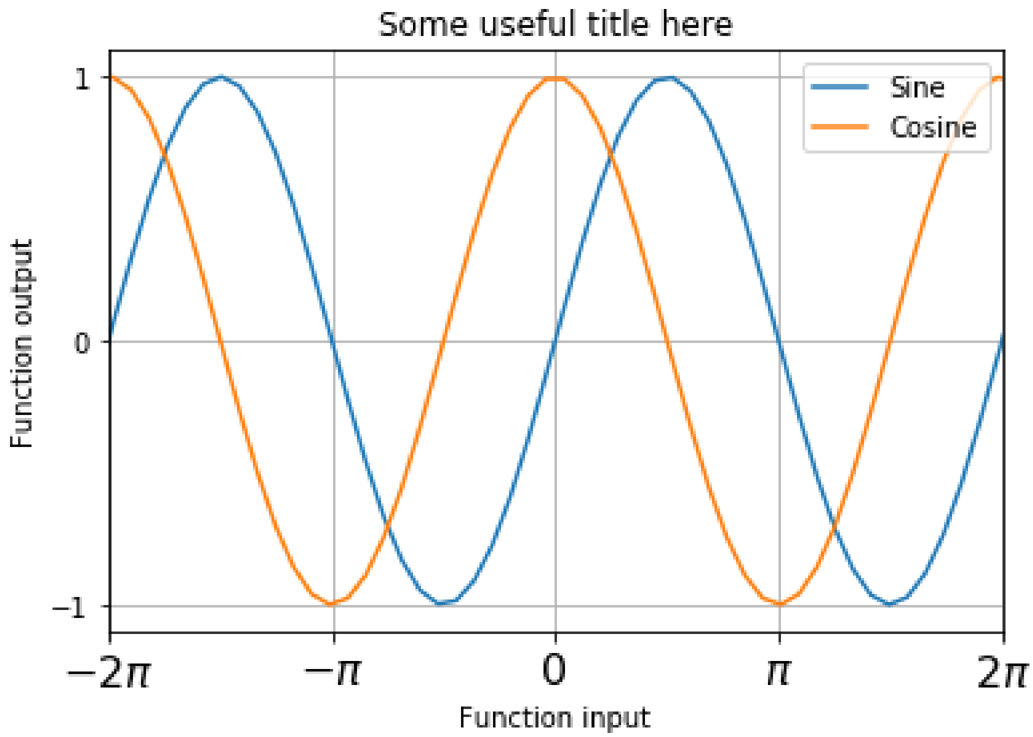


[99]:
```
# Grid

plt.figure()
plt.plot(x, wave1)
plt.plot(x, wave2)
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
plt.xlim((-np.pi * 2, np.pi * 2))
plt.xticks(
    [-np.pi * 2, -np.pi, 0, np.pi, np.pi * 2],
    ['$-2\pi$', '$-\pi$', '0', '$\pi$', '$2\pi$'],
    fontsize=16)
plt.yticks([-1, 0, 1])
plt.grid()
plt.show()
```



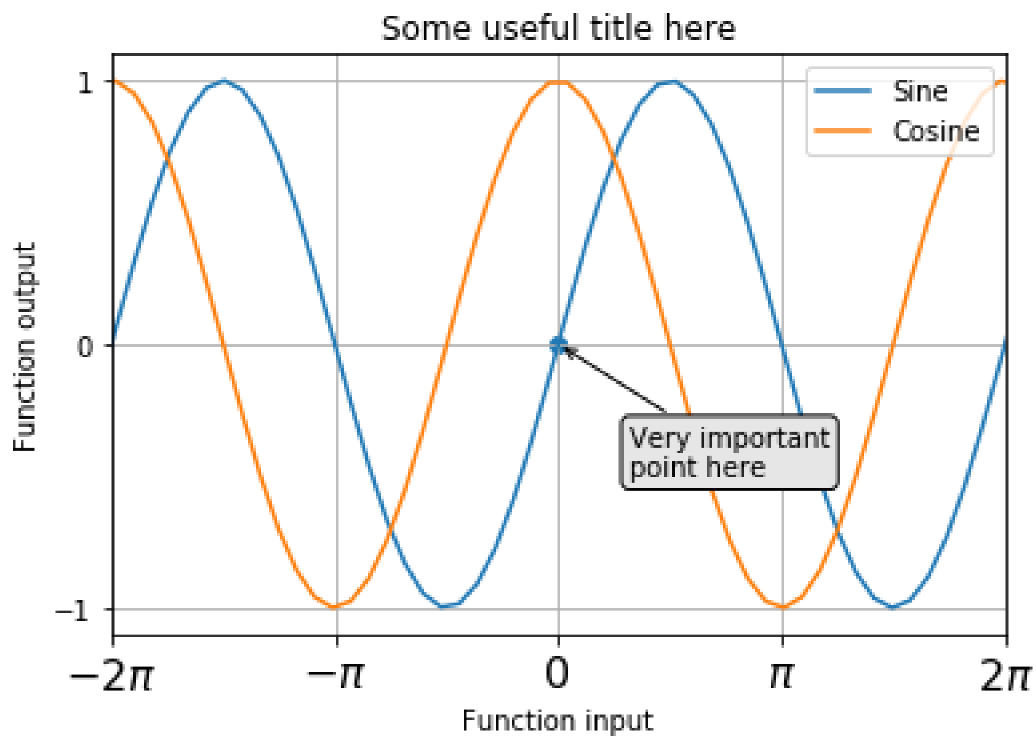
[100]:

```
# Legend

plt.figure()
# Need to specify label for each line:
plt.plot(x, wave1, label='Sine')
plt.plot(x, wave2, label='Cosine')
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
plt.xlim((-np.pi * 2, np.pi * 2))
plt.xticks(
    [-np.pi * 2, -np.pi, 0, np.pi, np.pi * 2],
    ['$-2\pi$', '$-\pi$', '0', '$\pi$', '$2\pi$'],
    fontsize=16)
plt.yticks([-1, 0, 1])
plt.grid()
# 'loc' keyword can be omitted and matplotlib
# will then pick the best place for the legend
plt.legend(loc='upper right')
plt.show()
```



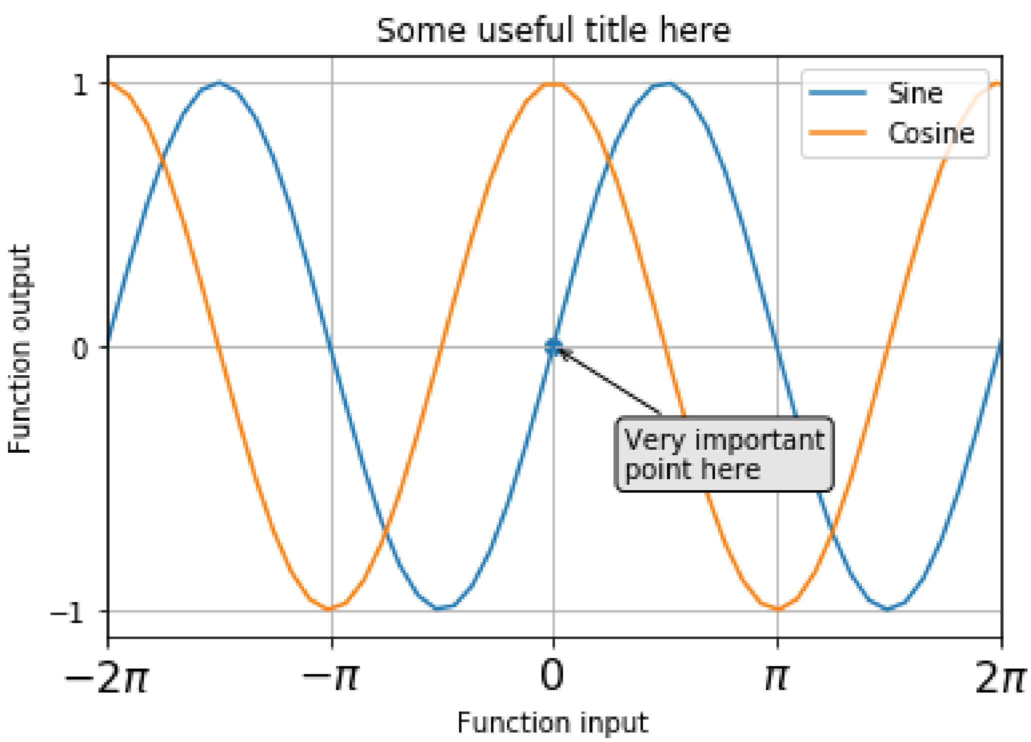
[101]:
```
# Annotations

plt.figure()
plt.plot(x, wave1, label='Sine')
plt.plot(x, wave2, label='Cosine')
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
plt.xlim((-np.pi * 2, np.pi * 2))
plt.xticks(
    [-np.pi * 2, -np.pi, 0, np.pi, np.pi * 2],
    ['$-2\pi$', '$-\pi$', '0', '$\pi$', '$2\pi$'],
    fontsize=16)
plt.yticks([-1, 0, 1])
plt.grid()
plt.legend(loc='upper right')

plt.annotate('Very important\npoint here',
    # Coordinates of annotated point:
    xy=(0, 0),
    # Text offset (from the annotated point):
    xytext=(1, -0.5),
    # Annotation text can be put into a box:
```

```
    bbox=dict(boxstyle="round", fc="0.9"),
    # Arrows:
    arrowprops=dict(arrowstyle='->'),)

# Annotated point can be marked separately,
# using scatter():
plt.scatter([0,], [0,])

plt.show()
```



[102]:
```
# Line styling

plt.figure()
plt.plot(x, wave1, label='Sine')
plt.plot(x, wave2, label='Cosine')
plt.xlabel('Function input')
plt.ylabel('Function output')
plt.title('Some useful title here')
plt.xlim((-np.pi * 2, np.pi * 2))
plt.xticks(
    [-np.pi * 2, -np.pi, 0, np.pi, np.pi * 2],
    ['$-2\pi$', '$-\pi$', '0', '$\pi$', '$2\pi$'],
    fontsize=16)
```

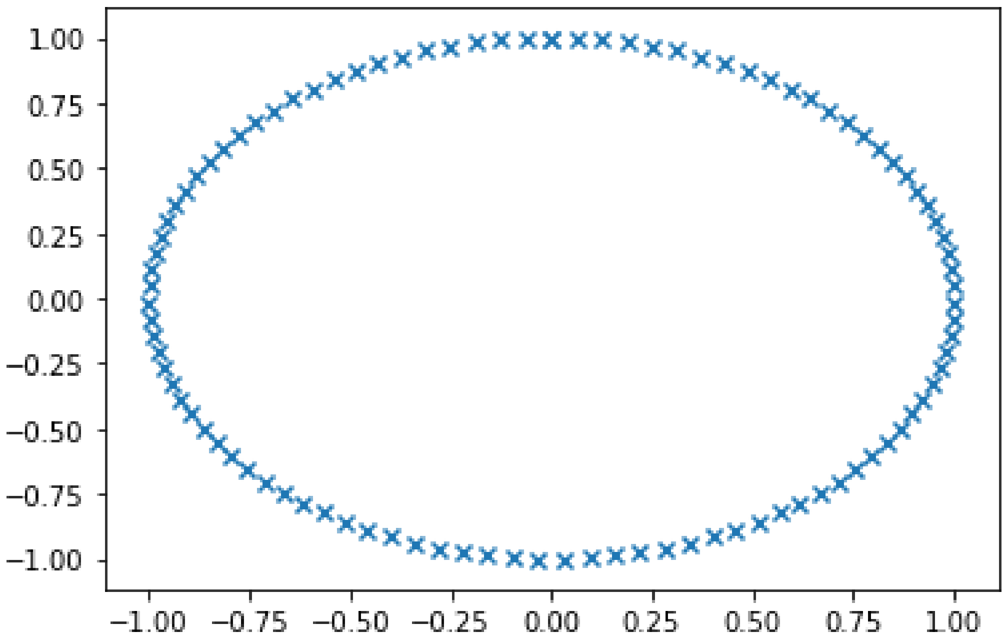
```
plt.yticks([-1, 0, 1])
plt.grid()
plt.legend(loc='upper right')

plt.annotate('Very important\npoint here',
    # Coordinates of annotated point:
    xy=(0, 0),
    # Text offset (from the annotated point):
    xytext=(1, -0.5),
    # Annotation text can be put into a box:
    bbox=dict(boxstyle="round", fc="0.9"),
    # Arrows:
    arrowprops=dict(arrowstyle='->'),)

# Annotated point can be marked separately,
# using scatter():
plt.scatter([0,], [0,])

plt.show()
```



5.3 Plot types

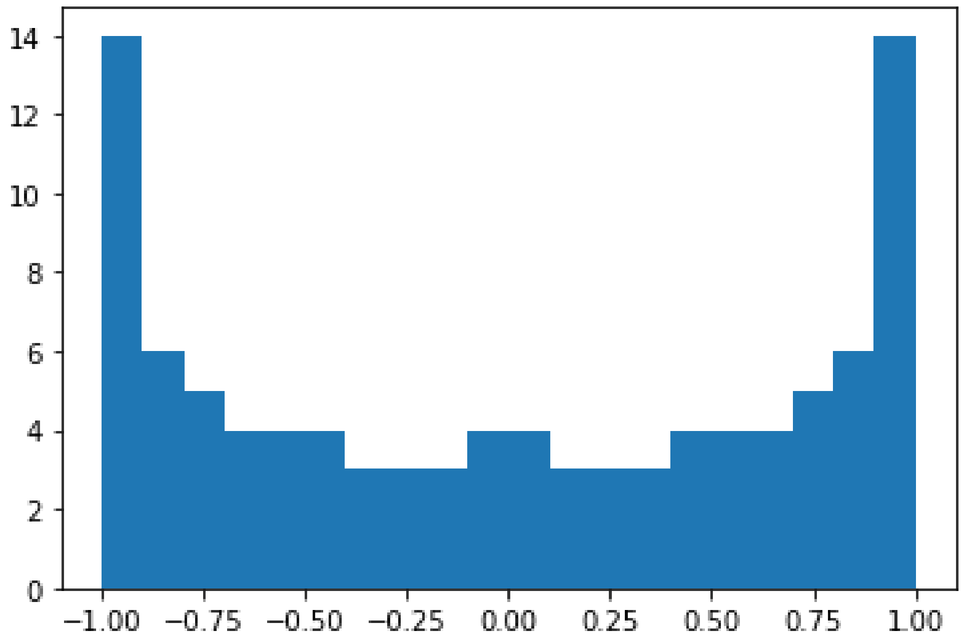
[103]:

```
# Scatter plots

plt.figure()
# Passing X and Y coordinates as positional arguments:
plt.scatter(wave1, wave2, marker='x')
plt.show()
```

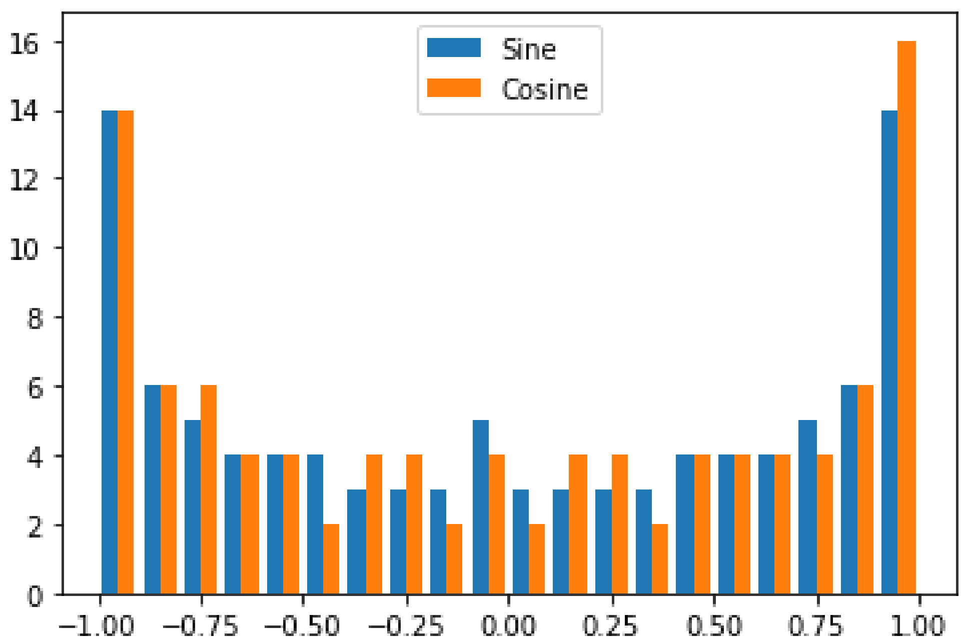


[104]:

```
# Histograms

plt.figure()
plt.hist(wave1, bins=20)
plt.show()
```



[105]:
```
# Multiple histograms
plt.figure()
plt.hist([wave1, wave2], bins=20, label=['Sine', 'Cosine'])
plt.legend()
plt.show()
```



[106]:
```
# Subplots

x = np.arange(1, 20)
y = np.log(x)

# Specifying number of rows and columns:
fig, axs = plt.subplots(2, 2)

# This is not the only way - you can also use
# fig.add_subplot() to get axes for selected subplot.

# Using axis objects to draw each subplot separately:
axs[0, 0].plot(x, y, marker='o', color='black')
axs[0, 1].plot(x, y, marker='x', color='red')
axs[1, 0].plot(x, y, marker='^', color='green')
axs[1, 1].plot(x, y, marker='v', color='blue')
```

[106]:
```
[<matplotlib.lines.Line2D at 0x7fe9c25e7c18>]
```