Scipy

What is SciPy?

SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely. SciPy was created by NumPy's creator Travis Olliphant.

Why Use SciPy?

If SciPy uses NumPy underneath, why can we not just use NumPy? SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

Which Language is SciPy Written in?

SciPy is predominantly written in Python, but a few segments are written in C.

Constants in SciPy

As SciPy is more focused on scientific implementations, it provides many built-in scientific constants. These constants can be helpful when you are working with Data Science. PI is an example of a scientific constant.

In [1]:
```
from scipy import constants
import numpy as np
```

In [2]:
```
constants.pi
```

Out[2]:
```
3.141592653589793
```

Constant Units

A list of all units under the constants module can be seen using the dir( ) function.

In [3]:
```
print(dir(constants))
```

```
['Avogadro', 'Boltzmann', 'Btu', 'Btu_IT', 'Btu_th', 'ConstantWarning', 'G',
'Julian_year', 'N_A', 'Planck', 'R', 'Rydberg', 'Stefan_Boltzmann', 'Wien',
'__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__path__', '__spec__', '_codata', '_constants',
'_obsolete_constants', 'acre', 'alpha', 'angstrom', 'arcmin', 'arcminute', 'a
rcsec', 'arcsecond', 'astronomical_unit', 'atm', 'atmosphere', 'atomic_mass',
'atto', 'au', 'bar', 'barrel', 'bbl', 'blob', 'c', 'calorie', 'calorie_IT',
'calorie_th', 'carat', 'centi', 'codata', 'constants', 'convert_temperature',
'day', 'deci', 'degree', 'degree_Fahrenheit', 'deka', 'dyn', 'dyne', 'e', 'e
V', 'electron_mass', 'electron_volt', 'elementary_charge', 'epsilon_0', 'er
g', 'exa', 'exbi', 'femto', 'fermi', 'find', 'fine_structure', 'fluid_ounce',
'fluid_ounce_US', 'fluid_ounce_imp', 'foot', 'g', 'gallon', 'gallon_US', 'gal
lon_imp', 'gas_constant', 'gibi', 'giga', 'golden', 'golden_ratio', 'grain',
'gram', 'gravitational_constant', 'h', 'hbar', 'hectare', 'hecto', 'horsepowe
r', 'hour', 'hp', 'inch', 'k', 'kgf', 'kibi', 'kilo', 'kilogram_force', 'km
h', 'knot', 'lambda2nu', 'lb', 'lbf', 'light_year', 'liter', 'litre', 'long_t
on', 'm_e', 'm_n', 'm_p', 'm_u', 'mach', 'mebi', 'mega', 'metric_ton', 'micr
o', 'micron', 'mil', 'mile', 'milli', 'minute', 'mmHg', 'mph', 'mu_0', 'nan
o', 'nautical_mile', 'neutron_mass', 'nu2lambda', 'ounce', 'oz', 'parsec', 'p
ebi', 'peta', 'physical_constants', 'pi', 'pico', 'point', 'pound', 'pound_fo
rce', 'precision', 'proton_mass', 'psi', 'pt', 'short_ton', 'sigma', 'slinc
h', 'slug', 'speed_of_light', 'speed_of_sound', 'stone', 'survey_foot', 'surv
ey_mile', 'tebi', 'tera', 'test', 'ton_TNT', 'torr', 'troy_ounce', 'troy_poun
d', 'u', 'unit', 'value', 'week', 'yard', 'year', 'yobi', 'yocto', 'yotta',
'zebi', 'zepto', 'zero_Celsius', 'zetta']
```

Unit Categories

The units are placed under these categories: ● Metric ● Binary ● Mass ● Angle ● Time ● Length ● Pressure ● Volume ● Speed ● Temperature ● Energy ● Power ● Force

Metric (SI) Prefixes:

Return the specified unit in meter (e.g. cent i returns 0.0 1)

In [4]:
```
constants.yotta
```

Out[4]: 1e+24

In [5]:
```
constants.peta
```

Out[5]: 1000000000000000.0

In [6]:
```
constants.mega
```

Out[6]: 1000000.0

In [7]:
```
constants.tera
```
Out[7]: 1000000000000.0

In [8]:
```
constants.giga
```
Out[8]: 1000000000.0

In [9]:
```
constants.kilo
```
Out[9]: 1000.0

In [10]:
```
constants.pico
```
Out[10]: 1e-12

In [11]:
```
constants.milli
```
Out[11]: 0.001

In [12]:
```
constants.micro
```
Out[12]: 1e-06

Binary Prefixes:

Return the specified unit in bytes (e.g. kibi returns 1024)

In [13]:
```
constants.kibi
```
Out[13]: 1024

In [14]:
```
constants.mebi
```
Out[14]: 1048576

In [15]:
```
constants.gibi
```
Out[15]: 1073741824

In [16]:
```
constants.tebi
```
Out[16]: 1099511627776

In [17]:
```
constants.pebi
```
Out[17]: 1125899906842624

In [18]:
```
constants.exbi
```

Out[18]: 1152921504606846976

In [19]:
```
constants.zebi
```

Out[19]: 1180591620717411303424

In [20]:
```
constants.yobi
```

Out[20]: 1208925819614629174706176

Mass:

Return the specified unit in kg (e.g. gram returns 0.001)

In [21]:
```
constants.gram
```

Out[21]: 0.001

In [22]:
```
constants.metric_ton
```

Out[22]: 1000.0

In [23]:
```
constants.grain
```

Out[23]: 6.479891e-05

In [24]:
```
print (constants.lb)
```

```
0.45359236999999997
```

In [25]:
```
print (constants.pound)
```

```
0.45359236999999997
```

In [26]:
```
print (constants.oz)
```

```
0.028349523124999998
```

In [27]:
```
print (constants.ounce)
```

```
0.028349523124999998
```

In [28]:
```
print (constants.stone)
```

```
6.3502931799999995
```

In [29]:
```
print (constants.long_ton)
```

```
1016.0469088
```

In [30]:
```
print (constants.short_ton)
```

```
907.1847399999999
```

In [31]:
```
print (constants.troy_ounce)
```

```
0.031103476799999998
```

In [32]:
```
print (constants.troy_pound)
```

```
0.37324172159999996
```

In [33]:
```
print (constants.carat)
```

```
0.0002
```

In [34]:
```
print (constants.atomic_mass)
```

```
1.6605390666e-27
```

In [35]:
```
print (constants.m_u)
```

```
1.6605390666e-27
```

In [36]:
```
print (constants.u)
```

```
1.6605390666e-27
```

Angle:

Return the specified unit in radians (e.g. degre e returns 0.01745329251994329 5)

In [37]:
```
constants.degree
```

Out[37]:
```
0.017453292519943295
```

In [38]:
```
constants.arcmin
```

Out[38]:
```
0.0002908882086657216
```

In [39]:
```
constants.arcminute
```

Out[39]:
```
0.0002908882086657216
```

In [40]:
```
constants.arcsecond
```
Out[40]:
```
4.84813681109536e-06
```

In [41]:
```
constants.arcsec
```
Out[41]:
```
4.84813681109536e-06
```

Time:

Return the specified unit in seconds (e.g. hou r returns 3600. 0)

In [42]:
```
constants.minute
```
Out[42]:
```
60.0
```

In [43]:
```
constants.hour
```
Out[43]:
```
3600.0
```

In [44]:
```
constants.week
```
Out[44]:
```
604800.0
```

In [45]:
```
constants.day
```
Out[45]:
```
86400.0
```

In [46]:
```
constants.year
```
Out[46]:
```
31536000.0
```

In [47]:
```
constants.Julian_year
```
Out[47]:
```
31557600.0
```

Currently, the Julian calendar is 13 days behind the Gregorian calendar. So, to convert from the Julian calendar to the Gregorian calendar, add 13 days; to convert in the opposite direction, subtract 13 days. The gap between the two calendar systems will increase to 14 days in the year 2100.m

Length:

Return the specified unit in meters (e.g. nautical_mile returns 1852.0)

In [48]:
```
constants.inch
```
Out[48]:
```
0.0254
```

In [49]:
```
constants.foot
```
Out[49]: 0.30479999999999996

In [50]:
```
constants.yard
```
Out[50]: 0.9143999999999999

In [51]:
```
constants.mile
```
Out[51]: 1609.3439999999998

In [52]:
```
constants.mil
```
Out[52]: 2.5399999999999997e-05

In [53]:
```
constants.pt
```
Out[53]: 0.00035277777777777776

In [54]:
```
constants.point
```
Out[54]: 0.00035277777777777776

In [55]:
```
constants.survey_foot
```
Out[55]: 0.3048006096012192

In [56]:
```
constants.survey_mile
```
Out[56]: 1609.3472186944373

In [57]:
```
constants.nautical_mile
```
Out[57]: 1852.0

In [58]:
```
constants.fermi
```
Out[58]: 1e-15

In [59]:
```
constants.angstrom
```
Out[59]: 1e-10

In [60]:
```
constants.micron
```
Out[60]: 1e-06

In [61]:
```
constants.au
```
Out[61]: 149597870700.0

In [62]:
```
constants.astronomical_unit
```
Out[62]: 149597870700.0

In [63]:
```
constants.light_year
```
Out[63]: 9460730472580800.0

In [64]:
```
constants.parsec
```
Out[64]: 3.085677581491367e+16

Pressure:

Return the specified unit in pascals (e.g. psi returns 6894.757293168361)

In [65]:
```
constants.atm
```
Out[65]: 101325.0

In [66]:
```
constants.atmosphere
```
Out[66]: 101325.0

In [67]:
```
constants.bar
```
Out[67]: 100000.0

In [68]:
```
constants.torr
```
Out[68]: 133.32236842105263

In [69]:
```
constants.mmHg
```
Out[69]: 133.32236842105263

In [70]:
```
constants.psi
```
Out[70]: 6894.757293168361

Area:

Return the specified unit in square meters(e.g. hectare returns 10000.0)

In [71]:
```
constants.hectare
```
Out[71]: 10000.0

In [72]:
```
constants.acre
```
Out[72]: 4046.8564223999992

Volume:

Return the specified unit in cubic meters (e.g. lite r returns 0.00 1)

In [73]:
```
constants.liter
```
Out[73]: 0.001

In [74]:
```
constants.litre
```
Out[74]: 0.001

In [75]:
```
constants.gallon
```
Out[75]: 0.0037854117839999997

In [76]:
```
constants.gallon_imp
```
Out[76]: 0.00454609

In [77]:
```
constants.gallon_US
```
Out[77]: 0.0037854117839999997

In [78]:
```
constants.fluid_ounce
```
Out[78]: 2.9573529562499998e-05

In [79]:
```
constants.fluid_ounce_imp
```
Out[79]: 2.84130625e-05

In [80]:
```
constants.fluid_ounce_US
```
Out[80]: 2.9573529562499998e-05

In [81]:
```
constants.barrel
```
Out[81]: 0.15898729492799998

In [82]:
```
constants.bbl
```
Out[82]: 0.15898729492799998

Speed:

Return the specified unit in meters per second (e.g. speed_of_sound returns 340.5)

In [83]:
```
constants.kmh
```
Out[83]: 0.2777777777777778

In [84]:
```
constants.mph
```
Out[84]: 0.44703999999999994

In [85]:
```
constants.mach
```
Out[85]: 340.5

In [86]:
```
constants.speed_of_sound
```
Out[86]: 340.5

In [87]:
```
constants.knot
```
Out[87]: 0.5144444444444445

Temperature:

Return the specified unit in Kelvin (e.g. zero_Celsiu s returns 273.15)

In [88]:
```
constants.zero_Celsius
```
Out[88]: 273.15

In [89]:
```
constants.degree_Fahrenheit
```
Out[89]: 0.5555555555555556

Energy:

Return the specified unit in joules (e.g. calori e returns 4.18 4)

In [90]:
```
constants.eV
```
Out[90]: 1.602176634e-19

In [91]:
```
constants.electron_volt
```
Out[91]: 1.602176634e-19

In [92]:
```
constants.calorie
```
Out[92]: 4.184

In [93]:
```
constants.calorie_th
```
Out[93]: 4.184

In [94]:
```
constants.erg
```
Out[94]: 1e-07

In [95]:
```
constants.Btu
```
Out[95]: 1055.05585262

In [96]:
```
constants.Btu_IT
```
Out[96]: 1055.05585262

In [97]:
```
constants.Btu_th
```
Out[97]: 1054.3502644888888

In [98]:
```
constants.ton_TNT
```
Out[98]: 4184000000.0

Power:

Return the specified unit in watts (e.g. horsepowe r returns 745.6998715822701)

In [99]:
```
constants.hp
```
Out[99]: 745.6998715822701

In [100]:
```
constants.horsepower
```
Out[100]: 745.6998715822701

Force:

Return the specified unit in newton (e.g. kilogram_forc e returns 9.80665)

In [101]:
```
constants.dyn
```
Out[101]: 1e-05

In [102]:
```
constants.dyne
```
Out[102]: 1e-05

In [103]:
```
constants.lbf
```
Out[103]: 4.4482216152605

In [104]:
```
constants.pound_force
```
Out[104]: 4.4482216152605

In [105]:
```
constants.kgf
```
Out[105]: 9.80665

In [106]:
```
constants.kilogram_force
```
Out[106]: 9.80665

Optimizers in SciPy while plotting graph, non linear relation

Optimizers are a set of procedures defined in SciPy that either find the minimum value of a function, or the root of an equation.

Optimizing Functions

Essentially, all of the algorithms in Machine Learning are nothing more than a complex equation that needs to be minimized with the help of given data.

Find root of the equation x + cos(x) :

In [107]:
```
from scipy.optimize import root
from math import cos
def eqn(x):
  return x + cos(x)
myroot = root(eqn, 0 )
print (myroot.x)
```

```
[-0.73908513]
```

In [108]:
```
print(myroot)
```

```
 message: The solution converged.
 success: True
  status: 1
     fun: [ 0.000e+00]
       x: [-7.391e-01]
    nfev: 9
    fjac: [[-1.000e+00]]
       r: [-1.674e+00]
     qtf: [-2.668e-13]
```

SciPy Statistical Significance Tests

In statistics, statistical significance means that the result that was produced has a reason behind it, it was not produced randomly, or by chance. SciPy provides us with a module called scipy.stats , which has functions for performing statistical significance tests.

T-test

T-tests are used to determine if there is significant deference between means of two variables. and lets us know if they belong to the same distribution. It is a two tailed test. The function ttest_ind( ) takes two samples of same size and produces a tuple of t-statistic and p-value.

In [109]:
```
from scipy.stats import ttest_ind
v1 = np.random.normal(size=100 )
v2 = np.random.normal(size=100 )
res = ttest_ind(v1, v2)
print (res)
```

```
Ttest_indResult(statistic=-0.3738548097876047, pvalue=0.7089123532194355)
```

For displaying only p-value;

In [110]:
```
res = ttest_ind(v1, v2).pvalue
print (res)
```

```
0.7089123532194355
```

KS Test (One Sample Kolmogorov Smirnov)

KS test is used to check if given values follow a distribution.

In [111]:
```
from scipy.stats import kstest

v = np.random.normal(size=100 )

res = kstest(v, 'norm' )

print (res)
```

```
KstestResult(statistic=0.05052455282364693, pvalue=0.9491571411974371, statis
tic_location=-0.7739660057965958, statistic_sign=1)
```

Statistical Description of Data

In order to see a summary of values in an array, we can use the describe( ) function. It returns the following description:

1. number of observations (nobs)
2. minimum and maximum values = minmax
3. mean
4. variance
5. skewness
6. kurtosis

In [112]:
```
from scipy.stats import describe
v = np.random.normal(size=100 )
res = describe(v)
print (res)
```

```
DescribeResult(nobs=100, minmax=(-1.9912420833029174, 2.4022302843980583), me
an=0.04005500292261657, variance=0.9411881219089001, skewness=0.3174416467763
3824, kurtosis=-0.3628862799655348)
```

Normality Tests (Skewness and Kurtosis)

In [113]:
```
from scipy.stats import skew, kurtosis

v = np.random.normal(size=100 )

print (skew(v))

print (kurtosis(v))
```

```
0.021842538200524866
-0.9272259281501878
```

Find if the data comes from a normal distribution:

In [114]:
```
from scipy.stats import normaltest

v = np.random.normal(size=100 )

print (normaltest(v))
```

```
NormaltestResult(statistic=4.183675276251119, pvalue=0.12346005233335727)
```