AMA546 Statistical Data Mining
Tutorial 1: Python

Ruijian HAN

Email: ruijian.han@polyu.edu.hk
Office: **TU827**
Office Hours: **Wednesdays 16:00-17:00**

Python

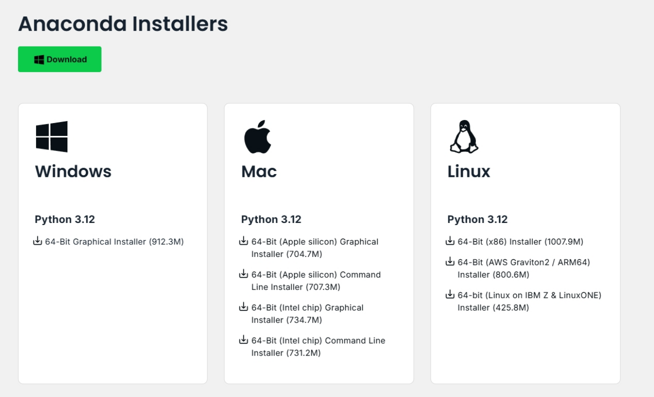
Python is a free software environment for general-purpose programming. It compiles and runs on a wide variety of platforms including UNIX, Windows and Linux.

- An interpreter-based programming, graphics and statistics package.
- Free, stable, can be extended.
- Can easily perform standard statistical and numerical analysis.
- Can be programmed to handle non-standard cases.
- For complex tasks, it is often used as a first step to interface with `C` or `FORTRAN`.
- Widely used by the deep learning communities and industry.

Installing Python

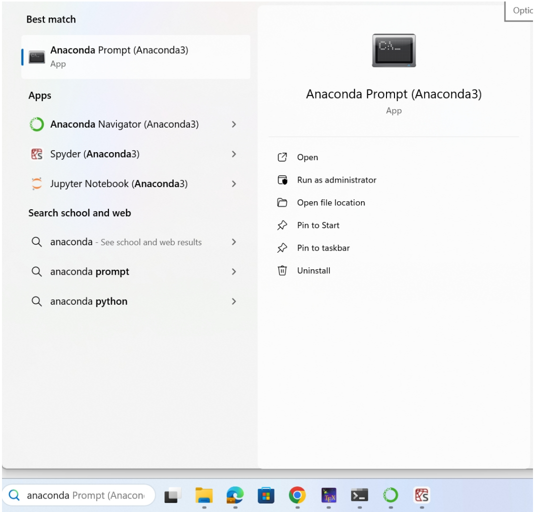
Anaconda is recommended.

See https://www.anaconda.com/download/success

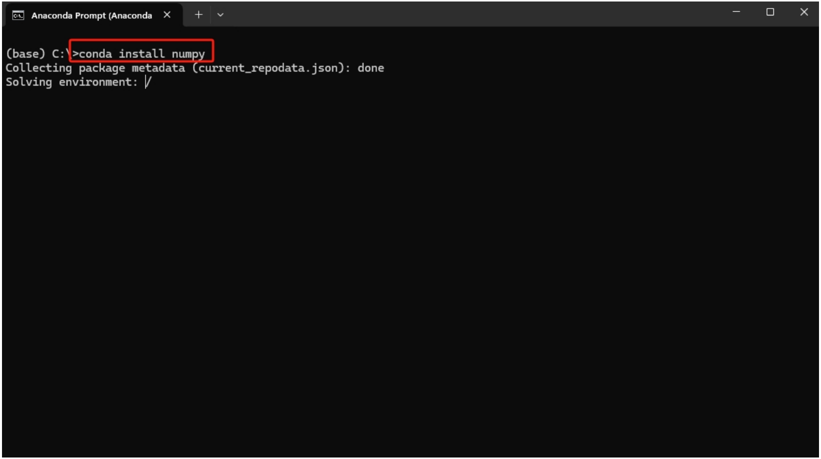


Installing Package

Launch the "Anaconda Prompt".



Launch Python. You may use `conda` or `pip` to install packages.



Create environments

It is recommended to create the environment. Try

```
conda create -n AMA546 python=3.9
conda activate AMA546
```

More details related to environments:

https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

Exercise

Try to install packages `bs4`, `requests`, `selenium`:

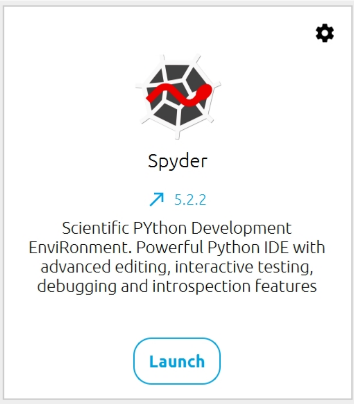
- `conda install conda-forge::bs4`
- `conda install anaconda::requests`
- `conda install conda-forge::selenium`

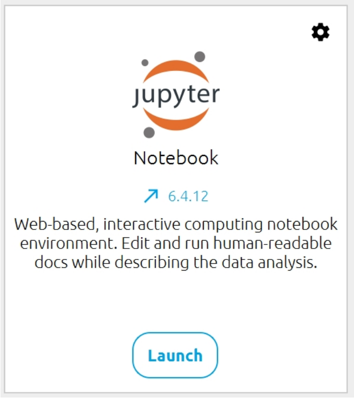
Questions:

Q1: What is the purpose of these packages?

Q2: What is the prerequisite to use selenium?

Spyder & Jupyter Notebook

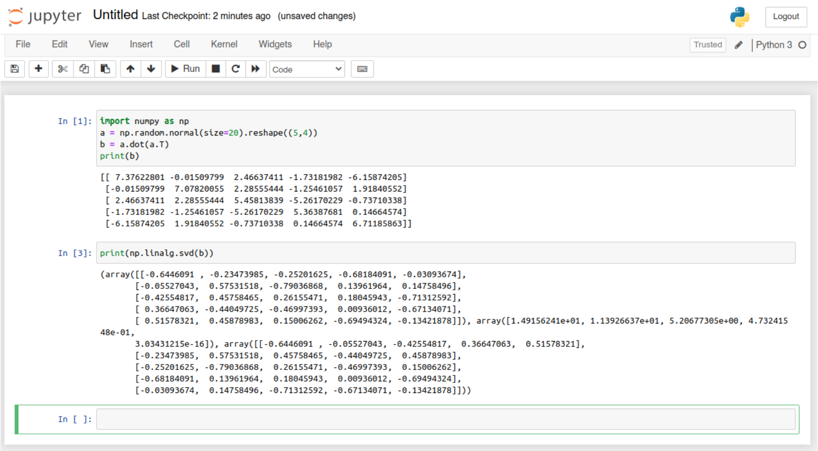




Spyder

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Feb  3 12:05:08 2025

@author: RJ
"""

import os
os.chdir(os.getcwd())
os.getcwd()

import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression



data = np.genfromtxt('vendordata.txt', delimiter = '')
X = data[:,2:4]
Y = data[:,1]
b = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
X_full =np.c_[np.ones([15]).T, X]
b = np.linalg.inv(X_full.T.dot(X_full)).dot(X_full.T).dot(Y)
model = LinearRegression()
model.fit(X_full, Y)
# another statement
model = LinearRegression().fit(X_full, Y)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
y_pred = model.predict(X_full)
print('predicted response:', y_pred)


ss = StandardScaler()
std_data = ss.fit_transform(np.c_[Y, X])
std_X = np.c_[np.ones([15]).T, std_data[:,1:3]]
std_Y = std_data[:,0]
model = LinearRegression().fit(std_X, std_Y)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
y_pred = model.predict(std_X)
print('predicted response:', y_pred)
plt.scatter(std_X[:,1], std_Y,  color='black')
plt.scatter(std_X[:,2], std_Y,  color='black')
```

Jupyter Notebook



Some simple commands

- All arithmetic operations are represented via standard symbols (+ − ∗ /) and have the usual order of precedence.

```
>>> 3 + 4 * 2
11
>>> math.sin(math.pi / 6)
0.49999999999999994
>>> from math import *
>>> exp(log(2) + log(3))
6.0
>>> atan(inf) / pi
0.5
```

Data type: Numbers

In Python, "`int`" stands for integers, and "`float`" stands for floating-point numbers.

```
>>> a = 17
>>> type(a)
<class 'int'>
>>> print(a)
17
>>> print(type(a))
<class 'int'>
>>> a + 1
18
>>> a - 1
16
>>> -a
-17
>>> a * a
289
>>> a ** 2 # THE NUMBER a RAISED TO POWER 2. DO NOT USE "a^2".
289
>>> a ** 100
110889937278078364130611171587509496643601716764987
952440276984127888758050136669771242469425600509358
924845150306839760800l
>>> type(a ** 100)
<class 'int'>
```

```
>>> type(3.14)
<class 'float'>
>>> 3.14 ** 20
8681463855.993662
>>> 3.14 ** 3000
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: (34, 'Numerical result out of range')
```

Data type: Boolean

Python has implemented Boolean logic. Remember that Python uses English words "and", "or", and "not" to represent Boolean operations.

```
>>> True
True
>>> False
False
>>> TRUE # PYTHON IS CASE SENSITIVE. CAPITAL LETTERS AND SMALL LETTERS
ARE DIFFERENT
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'TRUE' is not defined
>>> false
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'false' is not defined
>>> type(True)
<class 'bool'>
>>> True and False
False
>>> True or False
True
>>> not True
False
>>> not not True
True
```

Data type: String

```
>>> a = "hello" # ONE USES DOUBLE QUOTES
>>> type(a)
<class 'str'>

>>> b = 'world' # OR SINGLE QUOTES, IT DOES NOT MATTER.
>>> print(a)
hello

>>> print(len(a))
5

>>> print(a + " " + b) # STRING CONCATENATION, GLUE SEVERAL STRINGS
TOGETHER.
hello world

>>> str = "%s %f %.10f" % (a, 3.14 ** 5, 3.14 ** 5) # STRING FORMATTING
>>> str
'hello 305.244776 305.2447761824'

>>> "%.100f" % 3.15
'3.149999999999999911182158029987476766109466552
7343750000000000000000000000000000000000000000000000'
```

```
>>> (3.14).__add__(3)
6.140000000000001

>>> a = "Hello world!"
>>> a.upper()
'HELLO WORLD!'

>>> "Hello world!".lower()
'hello world!'

>>> a.rjust(5) # RIGHT-JUSTIFY A STRING
'Hello world!'
>>> a.rjust(20)
'        Hello world!'
>>> a.ljust(20)
'Hello world!        '
>>> a.center(20)
'    Hello world!    '

>>> a.replace("l", "[L]")
'He[L][L]o wor[L]d!'
>>> a.center(20).strip() # STRIPPING LEADING AND TAILING WHITESPACES.
'Hello world!'
```

Containers: list

Python implements some container data structures: “lists”, “dictionaries”, “sets”, and “tuples”.

```
>>> a = [2, 3, 5, 7, 11] # GENERATE A LIST
>>> print(a[0], a[4], a) # THE INDEX OF PYTHON AND C STARTS FROM 0;
2 11 [2, 3, 5, 7, 11]
>>> # WHILE THE INDEX OF R AND JULIA STARTS FROM 1.
>>> a[-1] # NEGATIVE INDEXES GIVE ELEMENTS COUNTING FROM THE END
11
>>> a[2] = "hello" # CHANGE VALUE

>>> a.append("world") # ADD A NEW ELEMENT TO THE END
>>> a
[2, 3, 'hello', 7, 11, 'world']
>>> b = a.pop() # REMOVE THE LAST ELEMENT OF a
>>> a
[2, 3, 'hello', 7, 11]
>>> b
'world'
```

From the above demonstration, we see that

- Different elements of a list may have different data type;
- The size of a list may change from time to time.

One may cut out a part from a list.

```
>>> a = list(range(10)) # MAKE A LIST FROM 0 TO 9 (NOT TO 10)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> a[1:5]
[1, 2, 3, 4]
>>> a[:5]
[0, 1, 2, 3, 4]
>>> a[5:]
[5, 6, 7, 8, 9]
>>> a[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[:-1]
[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> a[3:5] = ["a", "b", "c"] # ASSIGN VALUES TO THE SUB-LIST
>>> a
[0, 1, 2, 'a', 'b', 'c', 5, 6, 7, 8, 9]
```

Making a loop with a list

*Remember that **indentation** is an important part of the syntax of Python.*

```
>>> fruits = ["apple", "orange", "banana", "peach"]
>>> for x in fruits:
...   print(x)
...
apple
orange
banana
peach

>>> y = []
>>> for x in fruits:
...   y.append(len(x))
...
>>> y
[5, 6, 6, 5]
```

One may also use **list comprehension**

```
>>> z = ["I like " + x for x in fruits]
>>> z
['I like apple', 'I like orange', 'I like banana', 'I like peach']
```

Containers: dictionary

A dictionary is a container that stores “(key, value)” pairs.

```
>>> capital = {'China': 'Beijing', 'UK': 'London', 'Japan': 'Tokyo'}
>>> capital['UK']
'London'
>>> 'China' in capital
True
>>> capital['USA'] = 'Washington, D.C.'
>>> capital
{'China': 'Beijing', 'UK': 'London', 'Japan': 'Tokyo', 'USA':
'Washington, D.C.'}
>>> capital.get('Australia', 'N/A')
'N/A'
>>> capital.get('UK', 'N/A')
'London'
>>> del capital['UK']
>>> capital.get('UK', 'N/A')
'N/A'
```

One may also make loops over the keys in a dictionary.

```
>>> for country in capital:
...   a = capital[country]
...   print('The capital of %s is %s.' % (country, a))
...
The capital of China is Beijing.
The capital of Japan is Tokyo.
The capital of USA is Washington, D.C..

>>> for a, b in capital.items():
...   print('The capital of %s is %s.' % (a, b))
...
The capital of China is Beijing.
The capital of Japan is Tokyo.
The capital of USA is Washington, D.C..
```

There are a lot of techniques that we will have no time to cover. Students are encouraged to teach themselves, especially when they are working on Python-related projects.

5.6. Looping Techniques

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the `items()` method.

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the `enumerate()` function.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

To loop over two or more sequences at the same time, the entries can be paired with the `zip()` function.

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

To loop over a sequence in reverse, first specify the sequence in a forward direction and then call the `reversed()` function.

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

To loop over a sequence in sorted order, use the `sorted()` function which returns a new sorted list while leaving the source unaltered.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for i in sorted(basket):
...     print(i)
...
apple
apple
banana
orange
orange
pear
```

screenshot captured from: https://docs.python.org/3/tutorial/datastructures.html

Containers: tuple

A tuple is similar to a list, except:

- A tuple is **immutable**, which means you can not change its value or its length.
- A tuple can be used as a key of dictionary and as an element of set, but a list can not.
- For more subtleties, please study Python documentation.

```
>>> tupleExample = tuple(range(5))
>>> print(tupleExample)
(0, 1, 2, 3, 4)
>>> type(tupleExample)
<class 'tuple'>
```

Containers: set

Concept

Set: a collection of objects satisfying:

1. Rigorous membership: whether an object is in one set is well defined.
2. Uniqueness: all the objects in a set are distinct.
3. No order: there is not any order among the objects in a set.

Examples:

- $A = \{$All the students in the class classCode in Fall 2025$\}$
- $B = \{\sqrt{2}, 34, \pi\}$
- $C = \{x : 2x^2 - 5x - 3 = 0\} = \{3, -\frac{1}{2}\}$
- $D = \{x : -1.5 < x \leq 3\} = (-1.5, 3]$

The data type “set” in Python well implements the mathematical definition of a set.

```
>>> a1 = {'monkey', 'python', 'whale'}
>>> a2 = {'monkey', 'frog', 'lizard'}
>>> 'whale' in a1
True
>>> 'whale' in a2
False
>>> a1.add('monkey')
>>> a1.add('bat')
>>> a1
{'bat', 'python', 'whale', 'monkey'}
>>> len(a1)
4
>>> a2.remove('frog')
>>> a2.remove('whale')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'whale'
>>> a2.discard('whale')
>>> a2
{'lizard', 'monkey'}
>>> a1[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
```

```
>>> a1
{'bat', 'python', 'whale', 'monkey'}
>>> a2
{'lizard', 'monkey'}

>>> a1.union(a2)
{'whale', 'monkey', 'bat', 'python', 'lizard'}
>>> a1.intersection(a2)
{'monkey'}

>>> a1.difference(a2)
{'bat', 'python', 'whale'}

>>> a1.symmetric_difference(a2)
{'bat', 'python', 'whale', 'lizard'}
```

Functions

The definition of a function starts with the keyword "`def`".

```
>>> def like(name, strong = False):
...   if strong:
...     print("I like " + name + "!!!")
...   else:
...     print("I like " + name + ".")
...
>>> like("vegetable")
I like vegetable.
>>> like("fruits")
I like fruits.
>>> like("McDonald's", strong = True)
I like McDonald's!!!
```

Recall that the Fibonacci sequence $\{F_k\}_{k=1}^{\infty}$ is defined via $F_1 = F_2 = 1$ and

$$F_k = F_{k-1} + F_{k-2}, \quad \text{for any } k \geq 3.$$

```
>>> def Fibonacci(a):
...   if(a <= 2):
...     return(1)
...   else:
...     return(Fibonacci(a - 1) + Fibonacci(a - 2))
...
>>> Fibonacci(15)
610
>>> Fibonacci(30)
832040
```

This function "`Fibonacci`" is defined through **recursion**. We decompose the task of computing F_k for $k \geq 3$, into two smaller problems of computing F_{k-1} and F_{k-2}. Therefore, a recursive function often saves the time of the devlopers. A recursive function would however, usually cost too much time, CPU load, and computer memory.

Example

Use loop to find

$$\sum_{k=1}^{100} k.$$

If you get the answer 5050, your code is probably correct.

```
>>> # METHOD 1
>>> a = list(range(1, 101))
>>> b = 0
>>> for i in a:
...   b = b + i
...
>>> b
5050
>>> # METHOD 2
>>> sum(range(1, 101))
5050
```

Package NumPy for Linear Algebra

NumPy

- is a well known library for the Python programming language
- is written in Python and C
- was created in 2005
- is supervised by The NumPy Steering Council of leading experts

See `https://numpy.org/`

One makes vectors with NumPy.

```
>>> import numpy as np

>>> a = np.array([2, 3, 5]) # MAKING A VECTOR
>>> type(a)
<class 'numpy.ndarray'>
>>> a.shape
(3,)
>>> a.dtype
dtype('int64')
>>> print(a[0], a[2])
2 5
>>> print(a)
[2 3 5]
>>> np.sin(a)
array([ 0.90929743,  0.14112001, -0.95892427])
>>> np.sqrt(a)
array([1.41421356, 1.73205081, 2.23606798])
>>> np.exp(a)
array([  7.3890561 ,  20.08553692, 148.4131591 ])
```

One may hit the "Tab" key to explore more attributes and methods of one object.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a.
a.T                 a.dumps(            a.reshape(
a.all(              a.fill(             a.resize(
a.any(              a.flags             a.round(
a.argmax(           a.flat              a.searchsorted(
a.argmin(           a.flatten(          a.setfield(
a.argpartition(     a.getfield(         a.setflags(
a.argsort(          a.imag              a.shape
a.astype(           a.item(             a.size
a.base              a.itemset(          a.sort(
a.byteswap(         a.itemsize          a.squeeze(
a.choose(           a.max(              a.std(
a.clip(             a.mean(             a.strides
a.compress(         a.min(              a.sum(
a.conj(             a.nbytes            a.swapaxes(
a.conjugate(        a.ndim              a.take(
a.copy(             a.newbyteorder(     a.tobytes(
a.ctypes            a.nonzero(          a.tofile(
a.cumprod(          a.partition(        a.tolist(
a.cumsum(           a.prod(             a.tostring(
a.data              a.ptp(              a.trace(
a.diagonal(         a.put(              a.transpose(
a.dot(              a.ravel(            a.var(
a.dtype             a.real              a.view(
a.dump(             a.repeat(
```

Let's explore some linear algebraic operations.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> a.T # MATRIX TRANSPOSE
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> a.reshape(6)
array([1, 2, 3, 4, 5, 6])
>>> a.T.reshape(6)
array([1, 4, 2, 5, 3, 6])
>>> b = a.T.reshape(6)
>>> b.reshape([2, 3])
array([[1, 4, 2],
       [5, 3, 6]])
>>> b.reshape([3, 2])
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> a.dot(a.T) # MATRIX PRODUCT
array([[14, 32],
       [32, 77]])
>>> a.T.dot(a)
array([[17, 22, 27],
       [22, 29, 36],
       [27, 36, 45]])
>>> a.dot(a) # THIS MATRIX PRODUCT WILL NOT WORK
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

Some special matrices

```
>>> a = np.zeros([2, 3]) # MAKE A MATRIX OF SIZE 2-BY-3, OF ZEROS
>>> a
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> b = np.ones([3, 2]) # MAKE A MATRIX OF SIZE 2-BY-3, OF ONES
>>> b
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
>>> c = np.eye(3) # MAKE AN IDENTITY MATRIX
>>> c
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> d = np.random.uniform(low = 0, high = 1, size = 9).reshape((3,3))
>>> # MAKE A RANDOM MATRIX I.I.D. FROM THE UNIFORM DISTRIBUTION ON
[0,1]
>>> d
array([[0.83096397, 0.37767234, 0.77237302],
       [0.7487696 , 0.6571644 , 0.53969859],
       [0.44210463, 0.24029824, 0.75236625]])
```

The universe of random number generators implemented by NumPy

Distributions

beta (a, b[, size])	Draw samples from a Beta distribution.
binomial (n, p[, size])	Draw samples from a binomial distribution.
chisquare (df[, size])	Draw samples from a chi-square distribution.
dirichlet (alpha[, size])	Draw samples from the Dirichlet distribution.
exponential ([scale, size])	Draw samples from an exponential distribution.
f (dfnum, dfden[, size])	Draw samples from an F distribution.
gamma (shape[, scale, size])	Draw samples from a Gamma distribution.
geometric (p[, size])	Draw samples from the geometric distribution.
gumbel ([loc, scale, size])	Draw samples from a Gumbel distribution.
hypergeometric (ngood, nbad, nsample[, size])	Draw samples from a Hypergeometric distribution.
laplace ([loc, scale, size])	Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).
logistic ([loc, scale, size])	Draw samples from a logistic distribution.
lognormal ([mean, sigma, size])	Draw samples from a log-normal distribution.
logseries (p[, size])	Draw samples from a logarithmic series distribution.
multinomial (n, pvals[, size])	Draw samples from a multinomial distribution.
multivariate_hypergeometric (colors, nsample)	Generate variates from a multivariate hypergeometric distribution.
multivariate_normal (mean, cov[, size, ...])	Draw random samples from a multivariate normal distribution.
negative_binomial (n, p[, size])	Draw samples from a negative binomial distribution.
noncentral_chisquare (df, nonc[, size])	Draw samples from a noncentral chi-square distribution.
noncentral_f (dfnum, dfden, nonc[, size])	Draw samples from the noncentral F distribution.
normal ([loc, scale, size])	Draw random samples from a normal (Gaussian) distribution.

screenshot from https://numpy.org/doc/stable/reference/random/generator.html

About random seeds

NumPy generates random numbers according to a random seed. For more mathematics of random number generating techniques, see different volumes of the famous book *The art of computer programming* by Donald Knuth.

```
>>> np.random.seed(123)
>>> np.random.normal(size = 5)
array([-1.0856306 ,  0.99734545,  0.2829785 , -1.50629471, -0.57860025])
>>> np.random.seed(123)
>>> np.random.normal(size = 5)
array([-1.0856306 ,  0.99734545,  0.2829785 , -1.50629471, -0.57860025])
>>> np.random.normal(size = 5)
array([ 1.65143654, -2.42667924, -0.42891263,  1.26593626, -0.8667404 ])
```

Matrix algebra

One needs some preparation of linear algebra to understand the code below.

```
>>> np.random.seed(123)
>>> a = np.random.normal(size = 9).reshape([3, 3])
>>> b = np.linalg.inv(a) # GUESS, WHAT IS THE PROBABILITY THAT a IS
INVERTIBLE?
>>> b.dot(a)
array([[ 1.00000000e+00, -7.10501327e-17,  1.05538526e-16],
       [ 6.35890891e-17,  1.00000000e+00,  5.89712384e-17],
       [ 6.18471591e-17, -1.11765992e-16,  1.00000000e+00]])
>>> np.linalg.eig(a) # EIGEN DECOMPOSITION
(array([ 0.33539593+1.42242617j,  0.33539593-1.42242617j,
       -1.06908645+0.j        ]), array([[ 0.16650179-0.35533468j,
0.16650179+0.35533468j,
         0.69819494+0.j        ],
       [ 0.54901727-0.26881708j,  0.54901727+0.26881708j,
        -0.18466968+0.j        ],
       [ 0.68726402+0.j        ,  0.68726402-0.j        ,
         0.69167979+0.j        ]]))
>>> c = np.linalg.svd(a) # SINGULAR VALUE DECOMPOSITION
>>> c[0].dot(np.diag(c[1])).dot(c[2]) - a
array([[-4.44089210e-16,  5.55111512e-16, -4.99600361e-16],
       [-2.22044605e-16, -4.44089210e-16,  2.22044605e-16],
       [ 0.00000000e+00,  3.33066907e-16, -4.44089210e-16]])
```

The universe of linear algebraic operations implemented by NumPy

Matrix and vector products

dot(a, b[, out])	Dot product of two arrays.
linalg.multi_dot(arrays, *[, out])	Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order.
vdot(a, b)	Return the dot product of two vectors.
inner(a, b)	Inner product of two arrays.
outer(a, b[, out])	Compute the outer product of two vectors.
matmul(x1, x2, /[, out, casting, order, ...])	Matrix product of two arrays.
tensordot(a, b[, axes])	Compute tensor dot product along specified axes.
einsum(subscripts, *operands[, out, dtype, ...])	Evaluates the Einstein summation convention on the operands.
einsum_path(subscripts, *operands[, optimize])	Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays.
linalg.matrix_power(a, n)	Raise a square matrix to the (integer) power *n*.
kron(a, b)	Kronecker product of two arrays.

Decompositions

linalg.cholesky(a)	Cholesky decomposition.
linalg.qr(a[, mode])	Compute the qr factorization of a matrix.
linalg.svd(a[, full_matrices, compute_uv, ...])	Singular Value Decomposition.

Matrix eigenvalues

linalg.eig(a)	Compute the eigenvalues and right eigenvectors of a square array.
linalg.eigh(a[, UPLO])	Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix.
linalg.eigvals(a)	Compute the eigenvalues of a general matrix.
linalg.eigvalsh(a[, UPLO])	Compute the eigenvalues of a complex Hermitian or real symmetric matrix.

Norms and other numbers

linalg.norm(x[, ord, axis, keepdims])	Matrix or vector norm.
linalg.cond(x[, p])	Compute the condition number of a matrix.
linalg.det(a)	Compute the determinant of an array.
linalg.matrix_rank(M[, tol, hermitian])	Return matrix rank of array using SVD method
linalg.slogdet(a)	Compute the sign and (natural) logarithm of the determinant of an array.
trace(a[, offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.

Solving equations and inverting matrices

linalg.solve(a, b)	Solve a linear matrix equation, or system of linear scalar equations.
linalg.tensorsolve(a, b[, axes])	Solve the tensor equation `a x = b` for x.
linalg.lstsq(a, b[, rcond])	Return the least-squares solution to a linear matrix equation.
linalg.inv(a)	Compute the (multiplicative) inverse of a matrix.
linalg.pinv(a[, rcond, hermitian])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
linalg.tensorinv(a[, ind])	Compute the 'inverse' of an N-dimensional array.

For more functions, see
`https://numpy.org/doc/stable/reference/routines.linalg.html`.

... and keep Googling or ask Large Language Models.

Some good resources to improve your coding skills: leetcode.