In [1]:
```
import numpy as np
import scipy.linalg
```

1 Python Machine Learning Stack (Anaconda)

Task 1

> In your terminal, run

In [2]:
```
!conda info
```

```
     active environment : dl
    active env location : C:\Users\kdmen\miniconda3\envs\dl
            shell level : 2
       user config file : C:\Users\kdmen\.condarc
 populated config files : C:\Users\kdmen\.condarc
          conda version : 23.3.1
    conda-build version : not installed
         python version : 3.9.16.final.0
       virtual packages : __archspec=1=x86_64
                          __win=0=0
       base environment : C:\Users\kdmen\miniconda3  (writable)
      conda av data dir : C:\Users\kdmen\miniconda3\etc\conda
  conda av metadata url : None
           channel URLs : https://conda.anaconda.org/conda-forge/win-64
                          https://conda.anaconda.org/conda-forge/noarch
                          https://repo.anaconda.com/pkgs/main/win-64
                          https://repo.anaconda.com/pkgs/main/noarch
                          https://repo.anaconda.com/pkgs/r/win-64
                          https://repo.anaconda.com/pkgs/r/noarch
                          https://repo.anaconda.com/pkgs/msys2/win-64
                          https://repo.anaconda.com/pkgs/msys2/noarch
          package cache : C:\Users\kdmen\miniconda3\pkgs
                          C:\Users\kdmen\.conda\pkgs
                          C:\Users\kdmen\AppData\Local\conda\conda\pkgs
       envs directories : C:\Users\kdmen\miniconda3\envs
                          C:\Users\kdmen\.conda\envs
                          C:\Users\kdmen\AppData\Local\conda\conda\envs
               platform : win-64
             user-agent : conda/23.3.1 requests/2.29.0 CPython/3.9.16 Windows/10 Win
dows/10.0.22621
          administrator : False
             netrc file : None
           offline mode : False
```

3 Transition from MATLAB to Python

Task 2

> Run all of Python commands in the table “Linear Algebra Equivalents” in Numpy for MATLAB Users.

In [3]:
```
a = np.array([[1., 2., 3.], [4., 5., 6.]])
b = np.array([[7., 8., 9.], [10., 11., 12.]])
c = np.array([[2., 2., 2.], [4., 4., 4.]])
d = np.array([[1., 1., 1.], [3., 3., 3.]])

n = 1

v = np.array([1, 1, 10])

x = np.array([[5, 5, 5], [11, 12, 13]])
```

In [4]:
```
np.ndim(a)
```

Out[4]:
```
2
```

In [5]:
```
np.size(a)
```

Out[5]:
```
6
```

In [6]:
```
np.shape(a)
```

Out[6]:
```
(2, 3)
```

In [7]:
```
a.shape[n-1]
```

Out[7]:
```
2
```

In [8]:
```
np.array([[1., 2., 3.], [4., 5., 6.]])
```

Out[8]:
```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [9]:
```
np.block([[a, b], [c, d]])
```

Out[9]:
```
array([[ 1.,  2.,  3.,  7.,  8.,  9.],
       [ 4.,  5.,  6., 10., 11., 12.],
       [ 2.,  2.,  2.,  1.,  1.,  1.],
       [ 4.,  4.,  4.,  3.,  3.,  3.]])
```

In [10]:
```
a[-1]
```

Out[10]:
```
array([4., 5., 6.])
```

In [11]:
```
# a[1, 4]
a[1, 2]
```

Out[11]: 6.0

In [12]:
```python
# a[1] or a[1, :]
a[1, :]
```

Out[12]:
```
array([4., 5., 6.])
```

In [13]:
```python
# a[0:5] or a[:5] or a[0:5, :]
a[0:5, :]
```

Out[13]:
```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [14]:
```python
a[-5:]
```

Out[14]:
```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [15]:
```python
a[0:3, 4:9]
```

Out[15]:
```
array([], shape=(2, 0), dtype=float64)
```

In [16]:
```python
# a[np.ix_([1, 3, 4], [0, 2])]
a[np.ix_([0, 1], [0, 1, 2])]
```

Out[16]:
```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [17]:
```python
a[2:21:2,:]
```

Out[17]:
```
array([], shape=(0, 3), dtype=float64)
```

In [18]:
```python
a[::2, :]
```

Out[18]:
```
array([[1., 2., 3.]])
```

In [19]:
```python
a[::-1,:]
```

Out[19]:
```
array([[4., 5., 6.],
       [1., 2., 3.]])
```

In [20]:
```python
a[np.r_[:len(a),0]]
```

Out[20]:
```
array([[1., 2., 3.],
       [4., 5., 6.],
       [1., 2., 3.]])
```

In [21]:
```python
a.T
```

Out[21]:
```
array([[1., 4.],
       [2., 5.],
       [3., 6.]])
```

In [22]:
```python
a.conj().T
```

```
Out[22]: array([[1., 4.],
                [2., 5.],
                [3., 6.]])
```

```
In [23]: # a @ b
         a @ b.T
```

```
Out[23]: array([[ 50.,  68.],
                [122., 167.]])
```

```
In [24]: a * b
```

```
Out[24]: array([[ 7., 16., 27.],
                [40., 55., 72.]])
```

```
In [25]: a/b
```

```
Out[25]: array([[0.14285714, 0.25      , 0.33333333],
                [0.4       , 0.45454545, 0.5       ]])
```

```
In [26]: a**3
```

```
Out[26]: array([[  1.,   8.,  27.],
                [ 64., 125., 216.]])
```

```
In [27]: (a > 0.5)
```

```
Out[27]: array([[ True,  True,  True],
                [ True,  True,  True]])
```

```
In [28]: np.nonzero(a > 0.5)
```

```
Out[28]: (array([0, 0, 0, 1, 1, 1], dtype=int64),
          array([0, 1, 2, 0, 1, 2], dtype=int64))
```

```
In [29]: a[:,np.nonzero(v > 0.5)[0]]
```

```
Out[29]: array([[1., 2., 3.],
                [4., 5., 6.]])
```

```
In [30]: a[:, v.T > 0.5]
```

```
Out[30]: array([[1., 2., 3.],
                [4., 5., 6.]])
```

```
In [31]: a[a < 0.5]=0
```

```
In [32]: a * (a > 0.5)
```

```
Out[32]: array([[1., 2., 3.],
                [4., 5., 6.]])
```

```
In [33]: a[:] = 3
```

```
In [34]: y = x.copy()
```

In [35]:
```
y = x[1, :].copy()
```

In [36]:
```
y = x.flatten()
```

In [37]:
```
# np.arange(1., 11.) or np.r_[1.:11.] or np.r_[1:10:10j]
np.arange(1., 11.)
```

Out[37]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

In [38]:
```
# np.arange(10.) or np.r_[:10.] or np.r_[:9:10j]
np.arange(10.)
```

Out[38]:
```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

In [39]:
```
np.arange(1.,11.)[:, np.newaxis]
```

Out[39]:
```
array([[ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.],
       [ 6.],
       [ 7.],
       [ 8.],
       [ 9.],
       [10.]])
```

In [40]:
```
np.zeros((3, 4))
```

Out[40]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [41]:
```
np.zeros((3, 4, 5))
```

Out[41]:
```
array([[[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]])
```

In [42]:
```
np.ones((3, 4))
```

Out[42]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [43]:
```
np.eye(3)
```

Out[43]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [44]:
```
np.diag(a)
```

Out[44]:
```
array([3., 3.])
```

In [45]:
```
np.diag(v, 0)
```

Out[45]:
```
array([[ 1,  0,  0],
       [ 0,  1,  0],
       [ 0,  0, 10]])
```

In [46]:
```
from numpy.random import default_rng
rng = default_rng(42)
rng.random((3, 4))
```

Out[46]:
```
array([[0.77395605, 0.43887844, 0.85859792, 0.69736803],
       [0.09417735, 0.97562235, 0.7611397 , 0.78606431],
       [0.12811363, 0.45038594, 0.37079802, 0.92676499]])
```

In [47]:
```
np.linspace(1,3,4)
```

Out[47]:
```
array([1.        , 1.66666667, 2.33333333, 3.        ])
```

In [48]:
```
# np.mgrid[0:9.,0:6.] or np.meshgrid(r_[0:9.],r_[0:6.])
np.mgrid[0:9.,0:6.]
```

Out[48]:
```
array([[[0., 0., 0., 0., 0., 0.],
        [1., 1., 1., 1., 1., 1.],
        [2., 2., 2., 2., 2., 2.],
        [3., 3., 3., 3., 3., 3.],
        [4., 4., 4., 4., 4., 4.],
        [5., 5., 5., 5., 5., 5.],
        [6., 6., 6., 6., 6., 6.],
        [7., 7., 7., 7., 7., 7.],
        [8., 8., 8., 8., 8., 8.]],

       [[0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.],
        [0., 1., 2., 3., 4., 5.]]])
```

In [49]:
```
# ogrid[0:9.,0:6.] or np.ix_(np.r_[0:9.],np.r_[0:6.]
np.ogrid[0:9.,0:6.]
```

Out[49]:
```
[array([[0.],
        [1.],
        [2.],
        [3.],
        [4.],
        [5.],
        [6.],
        [7.],
        [8.]]),
 array([[0., 1., 2., 3., 4., 5.]])]
```

In [50]:
```
np.meshgrid([1,2,4],[2,4,5])
```

Out[50]:
```
[array([[1, 2, 4],
        [1, 2, 4],
        [1, 2, 4]]),
 array([[2, 2, 2],
        [4, 4, 4],
        [5, 5, 5]])]
```

In [51]:
```
np.ix_([1,2,4],[2,4,5])
```

Out[51]:
```
(array([[1],
        [2],
        [4]]),
 array([[2, 4, 5]]))
```

In [52]:
```
m = 3
np.tile(a, (m, n))
```

Out[52]:
```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

In [53]:
```
# np.concatenate((a,b),1) or np.hstack((a,b)) or np.column_stack((a,b)) or np.c_[a,
np.concatenate((a,b),1)
```

Out[53]:
```
array([[ 3.,  3.,  3.,  7.,  8.,  9.],
       [ 3.,  3.,  3., 10., 11., 12.]])
```

In [54]:
```
# np.concatenate((a,b)) or np.vstack((a,b)) or np.r_[a,b]
np.concatenate((a,b))
```

Out[54]:
```
array([[ 3.,  3.,  3.],
       [ 3.,  3.,  3.],
       [ 7.,  8.,  9.],
       [10., 11., 12.]])
```

In [55]:
```
a.max()
```

Out[55]: 3.0

In [56]:
```
a.max(0)
```

Out[56]:
```
array([3., 3., 3.])
```

In [57]:
```
a.max(1)
```

Out[57]:
```
array([3., 3.])
```

In [58]:
```
np.maximum(a, b)
```

Out[58]:
```
array([[ 7.,  8.,  9.],
       [10., 11., 12.]])
```

In [59]:
```
# np.sqrt(v @ v) or np.linalg.norm(v)
np.linalg.norm(v)
```

Out[59]:
```
10.099504938362077
```

In [60]:
```
np.logical_and(a,b)
```

Out[60]:
```
array([[ True,  True,  True],
       [ True,  True,  True]])
```

In [61]:
```
np.logical_or(a,b)
```

Out[61]:
```
array([[ True,  True,  True],
       [ True,  True,  True]])
```

In [62]:
```
aa = np.array([5, 3, 7], dtype=np.uint8)
bb = np.array([3, 6, 7], dtype=np.uint8)

#a & b
np.bitwise_and(aa, bb)
```

Out[62]:
```
array([1, 2, 7], dtype=uint8)
```

In [63]:
```
#a | b
np.bitwise_or(aa, bb)
```

Out[63]:
```
array([7, 7, 7], dtype=uint8)
```

In [64]:
```
sq_mtrx = np.array([[1,2,3], [4,5,6], [7,8,9]])
np.linalg.inv(sq_mtrx)
```

Out[64]:
```
array([[-4.50359963e+15,  9.00719925e+15, -4.50359963e+15],
       [ 9.00719925e+15, -1.80143985e+16,  9.00719925e+15],
       [-4.50359963e+15,  9.00719925e+15, -4.50359963e+15]])
```

In [65]:
```
np.linalg.pinv(sq_mtrx)
```

Out[65]:
```
array([[-6.38888889e-01, -1.66666667e-01,  3.05555556e-01],
       [-5.55555556e-02,  3.46944695e-17,  5.55555556e-02],
       [ 5.27777778e-01,  1.66666667e-01, -1.94444444e-01]])
```

In [66]:
```
np.linalg.matrix_rank(sq_mtrx)
```

Out[66]:
```
2
```

In [67]:
```
# linalg.solve(a, b) if a is square; linalg.lstsq(a, b) otherwise
np.linalg.solve(sq_mtrx, sq_mtrx)
```

Out[67]:
```
array([[1. , 0.5, 0. ],
       [0. , 0. , 0. ],
       [0. , 0.5, 1. ]])
```

In [68]:
```
# MATLAB: b/a --> NUMPY: Solve a.T x.T = b.T instead
```

In [69]:
```
U, S, Vh = np.linalg.svd(a); V = Vh.T
```

In [70]:
```
A = np.array([[4, -1, 2],
              [-1, 5, 3],
              [2, 3, 6]])

# Check if the matrix is positive definite
is_positive_definite = np.all(np.linalg.eigvals(A) > 0)

np.linalg.cholesky(A)
```

Out[70]:
```
array([[ 2.        ,  0.        ,  0.        ],
       [-0.5       ,  2.17944947,  0.        ],
       [ 1.        ,  1.60591014,  1.55597321]])
```

In [71]:
```
D,V = np.linalg.eig(sq_mtrx)
```

In [72]:
```
D,V = np.linalg.eig((sq_mtrx, sq_mtrx))
```

In [73]:
```
# Compute all eigenvalues and eigenvectors
D, V = np.linalg.eigh(A)

# Get the k (three) largest eigenvalues and their corresponding eigenvectors
k = 3
top_eigenvalues = D[-k:]
top_eigenvectors = V[:, -k:]
```

In [74]:
```
Q,R = np.linalg.qr(A)
```

In [75]:
```
# P,L,U = linalg.lu(a) where a == P@L@U
import scipy
P,L,U = scipy.linalg.lu(A)
```

In [76]:
```
np.fft.fft(a)
```

Out[76]:
```
array([[9.+0.j, 0.+0.j, 0.+0.j],
       [9.+0.j, 0.+0.j, 0.+0.j]])
```

In [77]:
```
np.fft.ifft(a)
```

Out[77]:
```
array([[3.+0.j, 0.+0.j, 0.+0.j],
       [3.+0.j, 0.+0.j, 0.+0.j]])
```

In [78]:
```
# np.sort(a) or a.sort(axis=0)
np.sort(a)
```

Out[78]:
```
array([[3., 3., 3.],
       [3., 3., 3.]])
```

In [79]:
```
# np.sort(a, axis=1) or a.sort(axis=1)
np.sort(a, axis=1)
```

Out[79]:
```
array([[3., 3., 3.],
       [3., 3., 3.]])
```

In [80]:
```
I = np.argsort(a[:, 0]); b = a[I,:]
```

In [81]:
```
Z = np.array([[1, 2], [3, 4], [5, 6]])
y = np.array([7, 8, 9])
np.linalg.lstsq(Z, y, rcond=None)[0]
```

Out[81]:
```
array([-6. ,  6.5])
```

In [82]:
```
from scipy.signal import resample
x = np.array([1, 2, 3, 4, 5])
resample(x, int(np.ceil(len(x) / 2)))
```

Out[82]:
```
array([2.        , 2.30801829, 4.69198171])
```

In [83]:
```
np.unique(a)
```

Out[83]:
```
array([3.])
```

In [84]:
```
a.squeeze()
```

Out[84]:
```
array([[3., 3., 3.],
       [3., 3., 3.]])
```

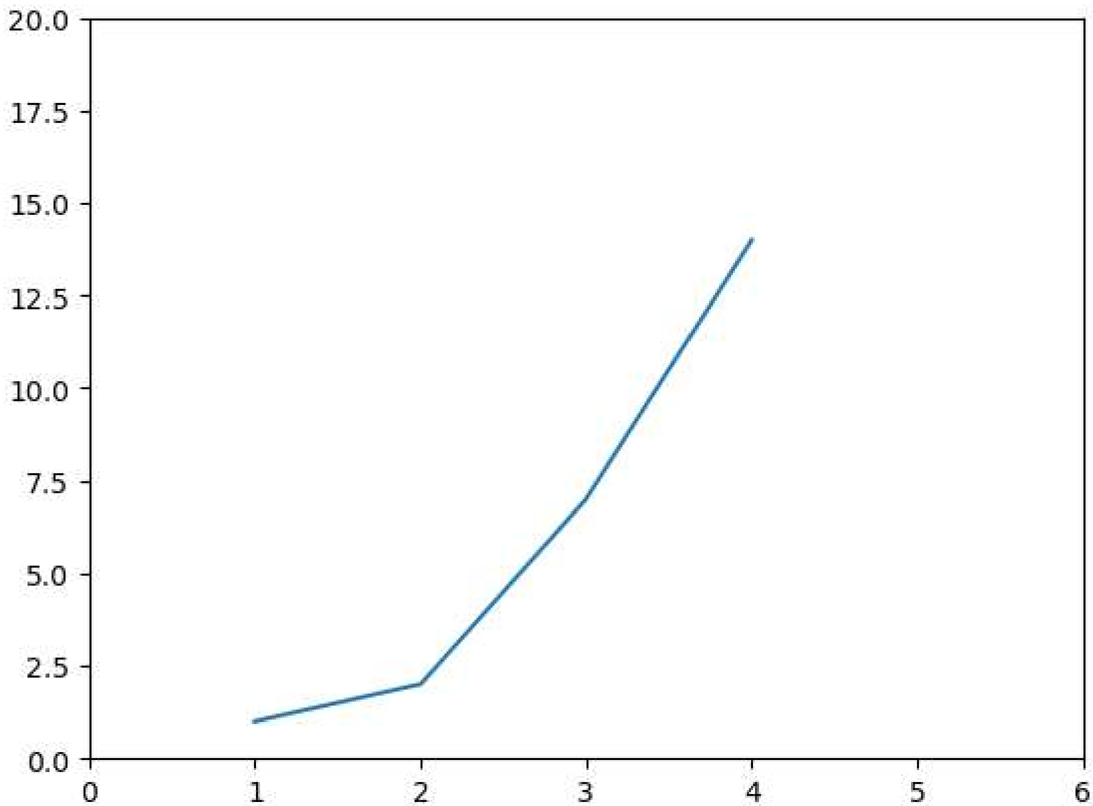
4 Plotting (MatPlotLib / PyPlot)

Task 3

> Run the following script in IPython and paste the figure created by the script into your report.

In [85]:

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,2,7,14])
plt.axis([0, 6, 0, 20])
plt.show()
```



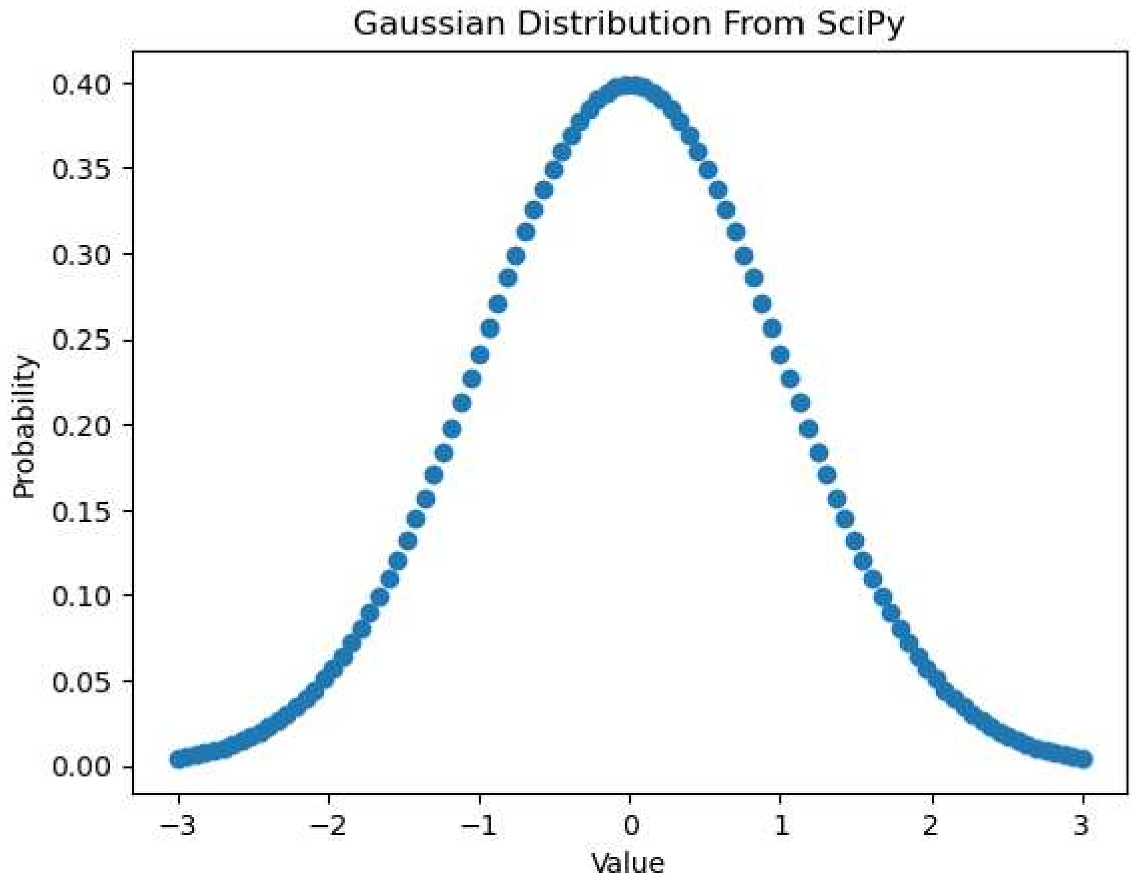
Task 4

> Use Matplotlib to create a figure of your choice in IPython. Paste your code and figure into your report.

In [86]:

```
import scipy.stats as stats

mean = 0
variance = 1
std = np.sqrt(variance)
x = np.linspace(mean-3*std, mean+3*std, 100)
plt.scatter(x, stats.norm.pdf(x, mean, std))
plt.title("Gaussian Distribution From SciPy")
plt.xlabel("Value")
plt.ylabel("Probability")
plt.show()
```



5 Version Control System (BitBucket/GitHub)

Task 5:

> Paste your VCS account into your report.

- Platform: Github
- Username: kdmalc
- Link: https://github.com/kdmalc

6 Integrated Development Environment (PyCharm)

Task 6:

> Start a new project in Pycharm. Commit and push your project to Bitbucket/GitHub as a public project. Paste the link of your project in your report.

- https://github.com/kdmalc/deeplearning576

```
In [ ]:
```