# HNCDI Explain Quantum Training

In [1]:

```
# Default imports
import numpy as np

# Importing standard Qiskit libraries
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, transpile, Aer, IBMQ
from qiskit.compiler import transpile
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()
```

In [2]:

```
# Notebook imports
from qiskit.providers.ibmq import least_busy

# pi approximation
from qiskit import assemble
from qiskit.tools.monitor import job_monitor
import matplotlib.pyplot as plt
```

In [3]:

```
# Version Information
%qiskit_version_table
```

## Version Information

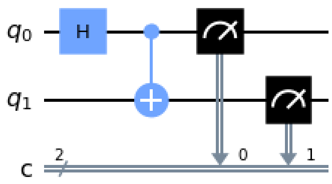
| Qiskit Software | Version |
|---|---|
| qiskit-terra | 0.19.1 |
| qiskit-aer | 0.10.2 |
| qiskit-ignis | 0.7.0 |
| qiskit-ibmq-provider | 0.18.3 |
| qiskit | 0.34.1 |
| qiskit-nature | 0.3.0 |
| qiskit-finance | 0.3.0 |
| qiskit-optimization | 0.3.0 |
| qiskit-machine-learning | 0.3.0 |
| **System information** | |
| Python version | 3.8.12 |
| Python compiler | GCC 9.4.0 |
| Python build | default, Oct 12 2021 21:59:51 |
| OS | Linux |
| CPUs | 8 |
| Memory (Gb) | 31.400043487548828 |
| | Thu Jan 20 17:21:08 2022 UTC |

# Create your Bell state circuit

In [4]:

```
# Create your Bell state circuit

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')

circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])

circuit.draw()
```

Out[4]:




## Submit Bell state circuit to a simulator

In [5]:

```
# View backends
Aer.backends()
```

Out[5]:

```
[AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator_statevector'),
 AerSimulator('aer_simulator_density_matrix'),
 AerSimulator('aer_simulator_stabilizer'),
 AerSimulator('aer_simulator_matrix_product_state'),
 AerSimulator('aer_simulator_extended_stabilizer'),
 AerSimulator('aer_simulator_unitary'),
 AerSimulator('aer_simulator_superop'),
 QasmSimulator('qasm_simulator'),
 StatevectorSimulator('statevector_simulator'),
 UnitarySimulator('unitary_simulator'),
 PulseSimulator('pulse_simulator')]
```

In [6]:

```
# Set simulator backend
simulator_backend = Aer.get_backend('aer_simulator')
```

In [7]:

```
# Submit job to simulator backend
simulator_job = simulator_backend.run(circuit, shots=1024)
```

In [8]:

```
# List measurement outcomes
simulator_job_counts = simulator_job.result().get_counts()
simulator_job_counts
```

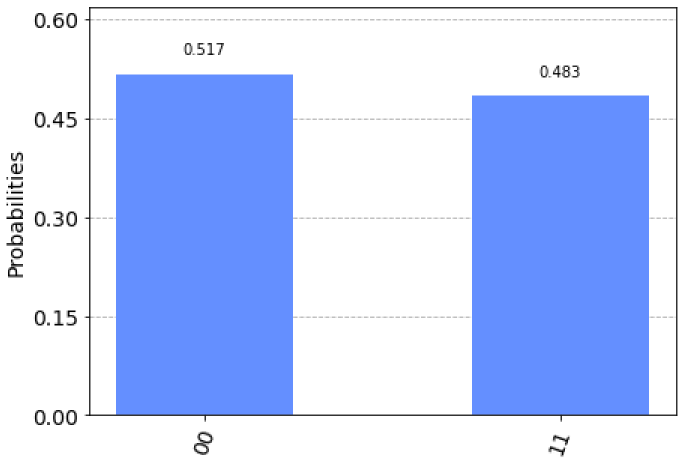
Out[8]:

```
{'00': 529, '11': 495}
```

In [9]:

```
# Plot measurement outcomes
plot_histogram(simulator_job_counts)
```

Out[9]:




## Submit your Bell state circuit to real quantum hardware

In [10]:

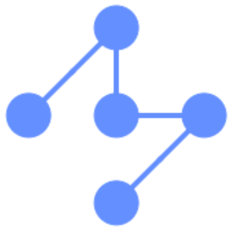
```
# View backends
provider.backends()
```

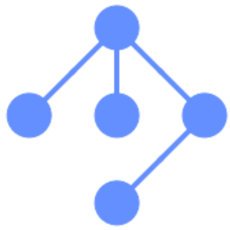
Out[10]:
```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_santiago') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_bogota') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_belem') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_quito') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQSimulator('simulator_statevector') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQSimulator('simulator_mps') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQSimulator('simulator_extended_stabilizer') from IBMQ(hub='ibm-q', group='open', project='main')
>,
 <IBMQSimulator('simulator_stabilizer') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open', project='main')>]
```

In [11]:
```
# Visual overview of available machines

%qiskit_backend_overview
```

## Backend Overview

**ibmq_manila** **ibmq_quito** **ibmq_bele**





Num. Qubits

Quantum Vol.

**5** **5** **5**

Pending Jobs

Operational

**32** **16** **16**

Least Busy

357 367 106 116 34

Avg. T1 / T2

Avg. CX Err.

True True True

Avg. Meas. Err.

False False False

**167.7 / 57.5 us** **79.4 / 88.5 us** **116.2 / 129.8**

**0.008** **0.0115** **0.0091**

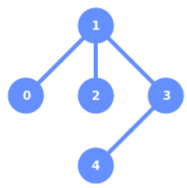
**0.0233** **0.0287** **0.0215**

In [12]:
```
# Find least busy machine to submit a job to. Set IBMQ backend to this machine.

ibmq_backend = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits >= 2
                                            and not x.configuration().simulator
                                            and x.status().operational==True))
ibmq_backend
```

## ibmq_lima

Configuration | Qubit Properties | Multi-Qubit Gates | Error Map | Job History

| **Property** | **Value** |
|---|---|
| **n_qubits** | 5 |
| **quantum_volume** | 8 |
| **operational** | True |
| **status_msg** | active |
| **pending_jobs** | 32 |
| **backend_version** | 1.0.27 |
| **basis_gates** | ['id', 'rz', 'sx', 'x', 'cx', 'reset'] |
| **max_shots** | 20000 |
| **max_experiments** | 100 |




hamiltonian

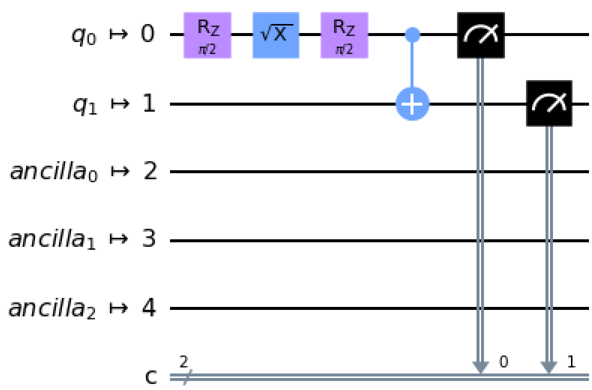
$$\mathcal{H}/\hbar = \sum_{i=0}^{4}\left(\frac{\omega_{q,i}}{2}(\mathbb{I}-\sigma_i^z)+\frac{\Delta_i}{2}(O_i^2-O_i)+\Omega_{d,i}D_i(t)\sigma_i^X\right)$$
$$+J_{0,1}(\sigma_0^+\sigma_1^-+\sigma_0^-\sigma_1^+)+J_{1,2}(\sigma_1^+\sigma_2^-+\sigma_1^-\sigma_2^+)+J_{1,3}(\sigma_1^+\sigma_3^-+\sigma_1^-\sigma_3^+)+J_{3,4}(\sigma_3^+\sigma_4^- +$$
$$+\Omega_{d,0}(U_0^{(0,1)}(t))\sigma_0^X+\Omega_{d,1}(U_1^{(1,0)}(t)+U_3^{(1,3)}(t)+U_2^{(1,2)}(t))\sigma_1^X$$

Out[12]:
```
<IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open', project='main')>
```

In [13]:
```python
# Transpile your circuit for the IBMQ backend
transpiled_circuit = transpile(circuit, ibmq_backend)
transpiled_circuit.draw()
```

Out[13]:

Global Phase: π/4



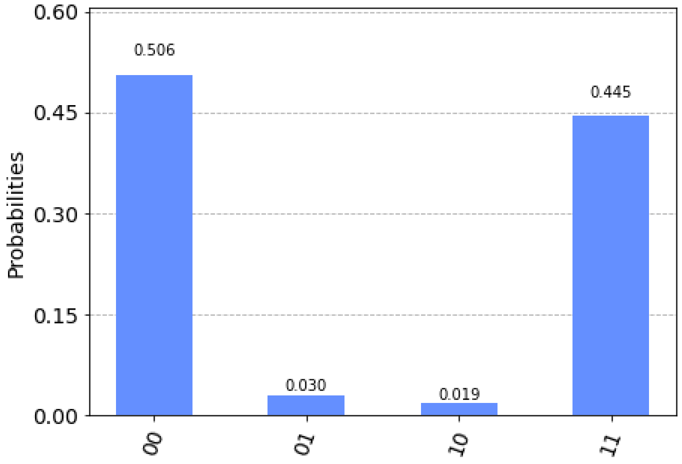

In [14]:
```python
# Submit job to IBMQ backend
ibmq_job = ibmq_backend.run(transpiled_circuit, shots=1024)
```

In [15]:
```python
# List measurement outcomes
ibmq_job_counts = ibmq_job.result().get_counts()
ibmq_job_counts
```

Out[15]:
```
{'00': 518, '01': 31, '10': 19, '11': 456}
```

In [16]:
```python
# Plot measurement outcomes
plot_histogram(ibmq_job_counts)
```

Out[16]:




## Approximating $\pi$ using IBMQ

In [17]:

```
#Compute the Inverse Quantum Fourier Transform

def qft_dagger(circ_, n_qubits):
    """n-qubit QFTdagger the first n qubits in circ"""
    for qubit in range(int(n_qubits/2)):
        circ_.swap(qubit, n_qubits-qubit-1)
    for j in range(0,n_qubits):
        for m in range(j):
            circ_.cp(-np.pi/float(2**(j-m)), m, j)
        circ_.h(j)
```

In [18]:

```
#Initial state of Quantum Phase Estimation

def qpe_pre(circ_, n_qubits):
    circ_.h(range(n_qubits))
    circ_.x(n_qubits)

    for x in reversed(range(n_qubits)):
        for _ in range(2**(n_qubits-1-x)):
            circ_.cp(1, n_qubits-1-x, n_qubits)
```

In [19]:

```
def run_job(circ, backend, shots, optimization_level=0):
    t_circ = transpile(circ, backend, optimization_level=optimization_level)
    qobj = assemble(t_circ, shots=shots)
    job = backend.run(qobj)
    job_monitor(job)
    return job.result().get_counts()
```

In [20]:

```
def get_pi_estimate(n_qubits, backend, shots):

    # create the circuit
    circ = QuantumCircuit(n_qubits + 1, n_qubits)

    # create the input state
    qpe_pre(circ, n_qubits)

    # apply a barrier
    circ.barrier()
    # apply the inverse fourier transform
    qft_dagger(circ, n_qubits)
    # apply  a barrier
    circ.barrier()
    # measure all but the last qubits
    circ.measure(range(n_qubits), range(n_qubits))

    # optionally save to a file
    if n_qubits == 3:
        circ.draw(filename='qpe')

    # run the job and get the results
    counts = run_job(circ, backend, shots, optimization_level=0)
    print('counts = ', counts)

    # get the count that occurred most frequently
    max_counts_result = max(counts, key=counts.get)
    print('max_counts_result = ', max_counts_result)
    max_counts_result = int(max_counts_result, 2)
```

```python
    print('max_counts_result = ', max_counts_result)

    # solve for pi from the measured counts
    theta = max_counts_result/2**n_qubits
    return (1./(2*theta))
```

In [21]:

```python
# estimate pi using different numbers of qubits

shots = 10000
nqs = list(range(2,12+1))
pi_estimates = []
for nq in nqs:
    thisnq_pi_estimate = get_pi_estimate(nq, simulator_backend, shots)
    pi_estimates.append(thisnq_pi_estimate)
    print(f"{nq} qubits, pi ≈ {thisnq_pi_estimate}")
```

```
Job Status: job has successfully run
counts =  {'11': 559, '01': 6569, '10': 657, '00': 2215}
max_counts_result =  01
max_counts_result =  1
2 qubits, pi ≈ 2.0
Job Status: job has successfully run
counts =  {'010': 1129, '100': 110, '110': 101, '001': 7782, '101': 93, '011': 248, '000': 408, '111':
129}
max_counts_result =  001
max_counts_result =  1
3 qubits, pi ≈ 4.0
Job Status: job has successfully run
counts =  {'1101': 57, '1010': 33, '1111': 92, '1110': 65, '1011': 39, '1000': 56, '0000': 176, '011
1': 67, '0011': 4798, '0010': 3330, '1100': 33, '0101': 206, '0001': 445, '1001': 38, '0110': 110, '01
00': 455}
max_counts_result =  0011
max_counts_result =  3
4 qubits, pi ≈ 2.6666666666666665
Job Status: job has successfully run
counts =  {'01110': 1, '10010': 1, '10101': 1, '11010': 1, '01111': 2, '10111': 1, '11100': 2, '0110
1': 2, '00001': 3, '01001': 6, '00101': 9707, '01100': 3, '00010': 12, '00110': 115, '00111': 25, '001
00': 75, '00011': 13, '11011': 2, '01000': 10, '00000': 6, '01011': 2, '01010': 5, '10001': 2, '1110
1': 3}
max_counts_result =  00101
max_counts_result =  5
5 qubits, pi ≈ 3.2
Job Status: job has successfully run
counts =  {'111001': 2, '100000': 1, '011101': 1, '110001': 2, '101101': 1, '011011': 1, '010111': 2,
'100011': 1, '010101': 2, '101111': 2, '011000': 6, '010100': 3, '000000': 1, '101011': 3, '000011': 1
0, '101001': 1, '000111': 34, '101000': 1, '001110': 17, '001101': 37, '000001': 4, '110111': 1, '0010
01': 222, '010011': 2, '000101': 8, '010110': 1, '110010': 2, '010000': 5, '001100': 101, '000010': 4,
'001010': 8933, '000100': 9, '010001': 5, '001011': 444, '001000': 74, '000110': 17, '110101': 1, '111
010': 1, '110100': 1, '001111': 10, '010010': 8, '011001': 3, '011110': 1, '101010': 1, '111110': 2,
'011010': 2, '110000': 1, '101110': 3, '100010': 1, '111111': 5}
max_counts_result =  001010
max_counts_result =  10
6 qubits, pi ≈ 3.2
Job Status: job has successfully run
counts =  {'1001100': 1, '1011000': 1, '1101110': 2, '0110110': 1, '0011011': 26, '0101110': 2, '11101
01': 1, '1100101': 1, '0001101': 18, '0001100': 11, '0010000': 35, '0000111': 10, '0010110': 305, '000
1110': 24, '0000000': 5, '0101011': 1, '1111001': 1, '0010011': 442, '0000101': 4, '1110001': 1, '0100
000': 4, '0010100': 6211, '0011000': 68, '0011100': 17, '0110010': 1, '0010101': 2162, '0000100': 7,
'0010001': 72, '0000011': 4, '1101101': 1, '0010010': 169, '0001111': 33, '0001010': 11, '0000010': 4,
'1010110': 1, '1110000': 3, '0011010': 32, '0111110': 1, '0010111': 147, '0100011': 3, '1000101': 1,
'1111010': 2, '1000011': 3, '1000100': 1, '0101100': 3, '0011110': 10, '0100100': 3, '1110011': 2, '11
10100': 3, '1111111': 1, '0001011': 9, '1111110': 1, '1101111': 1, '0101010': 2, '0000110': 4, '000100
1': 4, '0100001': 5, '0100101': 2, '0001000': 6, '0100010': 2, '1101100': 2, '0110000': 2, '1111011':
4, '1111000': 1, '0101001': 2, '1001010': 1, '1001101': 1, '0011001': 42, '0000001': 3, '1000010': 1,
'1010100': 1, '1100011': 2, '1001000': 1, '0100110': 2, '1110111': 2, '0110011': 2, '0101000': 1, '001
1101': 7, '1111100': 2, '0111000': 1, '0011111': 10, '1010010': 1, '0101111': 1}
max_counts_result =  0010100
max_counts_result =  20
7 qubits, pi ≈ 3.2
Job Status: job has successfully run
counts =  {'11111110': 1, '01011100': 1, '00011110': 2, '00011001': 2, '00111101': 1, '00010111': 4,
'00100011': 21, '00110001': 9, '00011100': 1, '00100001': 9, '00100101': 44, '00011011': 3, '0011011
0': 4, '00100010': 15, '11000000': 1, '00110100': 6, '01000011': 2, '00111111': 1, '00011010': 7, '001
01001': 8062, '00101011': 109, '00100111': 167, '00101101': 27, '11110010': 1, '00011000': 1, '0011101
0': 3, '10101011': 1, '00111001': 6, '01001000': 1, '00100110': 71, '10100110': 1, '00110101': 4, '001
01111': 17, '00011111': 7, '01100101': 1, '10000011': 1, '00100000': 5, '10001000': 1, '00100100': 21,
'01010101': 1, '00000011': 1, '10010110': 1, '01000001': 2, '00001001': 1, '11001011': 1, '00110000':
8, '11101011': 1, '00000111': 1, '00010001': 1, '11010110': 1, '00101000': 937, '00111000': 5, '000111
01': 4, '00110011': 4, '00111100': 1, '00010110': 2, '01000010': 1, '00101110': 14, '01000111': 2, '00
110111': 3, '11010011': 1, '10011110': 1, '01101001': 2, '10111010': 1, '00001111': 1, '01010100': 1,
'00110010': 3, '00010101': 2, '00101010': 303, '00000110': 1, '10111101': 1, '10010101': 1, '1111100
0': 1, '01010000': 2, '00111011': 1, '11111011': 1, '00101100': 39, '10001110': 1, '11101000': 1, '010
00110': 2, '11011011': 1}
max_counts_result =  00101001
max_counts_result =  41
8 qubits, pi ≈ 3.1219512195121952
Job Status: job has successfully run
counts =  {'110111000': 1, '111110110': 1, '010111011': 1, '000010010': 1, '001101100': 2, '01001010
0': 1, '100011100': 1, '101101011': 1, '001011011': 3, '000000101': 1, '111110000': 1, '101010110': 1,
'010110110': 1, '010101110': 1, '001101111': 2, '000011100': 1, '010100010': 1, '111111111': 1, '10011
0100': 1, '000110101': 1, '011010011': 1, '000110111': 3, '001000001': 2, '000110100': 1, '001001100':
36, '001100101': 2, '001100100': 3, '111011101': 1, '001100010': 4, '001011111': 7, '001110010': 2, '0
01011001': 20, '001011100': 11, '001110100': 4, '111010110': 2, '000010001': 1, '001000000': 4, '01000
0001': 1, '001001101': 42, '001001010': 17, '001010011': 456, '000111011': 2, '100011000': 1, '0010001
01': 5, '001010001': 4179, '000111100': 3, '001010110': 41, '010010010': 1, '001010100': 163, '0010011
11': 179, '010000100': 1, '001011101': 9, '011111000': 1, '001010000': 454, '001110111': 1, '00100101
1': 32, '001111100': 1, '000111000': 1, '000010101': 1, '000110010': 1, '001010111': 47, '111100101':
1, '000010110': 1, '000101110': 1, '001000111': 8, '001100001': 5, '111101011': 1, '000011000': 1, '00
0111010': 6, '010001100': 1, '111110001': 1, '000011001': 1, '000111101': 2, '101001101': 1, '10100100
1': 1, '000010011': 1, '001001110': 100, '001000011': 2, '000110110': 2, '111001111': 1, '001010010':
3874, '001100000': 3, '000001111': 1, '011011000': 1, '001000010': 5, '001101010': 2, '001100111': 2,
```

```
'001010101': 90, '111101000': 1, '110101110': 1, '000111110': 6, '000111111': 1, '001001001': 15, '000
011010': 1, '000000000': 1, '010001111': 2, '001101011': 2, '001110011': 2, '001101001': 3, '00101101
0': 17, '001011000': 24, '000111001': 4, '010101011': 1, '001100110': 3, '001000100': 9, '001000110':
4, '001001000': 13, '001100011': 5, '001011110': 3}
max_counts_result =  001010001
max_counts_result =  81
9 qubits, pi ≈ 3.1604938271604937
Job Status: job has successfully run
counts =  {'0010111101': 1, '0010001000': 1, '0010100100': 8, '0010100011': 9976, '0010011111': 1, '00
10100000': 1, '0010100010': 6, '0010100101': 1, '0010100001': 2, '0010110010': 1, '0010100110': 2}
max_counts_result =  0010100011
max_counts_result =  163
10 qubits, pi ≈ 3.1411042944785277
Job Status: job has successfully run
counts =  {'00101010011': 1, '00101100010': 1, '00101011001': 1, '00101001001': 2, '00101000010': 3,
'00101000000': 1, '00101001011': 1, '00100011101': 1, '00101001111': 1, '00101000110': 9921, '00101000
111': 21, '00101001000': 5, '00100111011': 1, '00101000100': 5, '00101000101': 31, '00101000011': 4}
max_counts_result =  00101000110
max_counts_result =  326
11 qubits, pi ≈ 3.1411042944785277
Job Status: job has successfully run
counts =  {'001011100110': 1, '001010011100': 1, '001010000100': 2, '001010001000': 5, '001010011111':
1, '001010010011': 4, '001010010001': 1, '001001100010': 1, '001110000001': 1, '001010001101': 88, '00
1010011010': 1, '001010001001': 10, '001010001100': 9666, '001010000000': 2, '001010001110': 17, '0010
10000110': 2, '001001011011': 1, '000111101101': 1, '001010000111': 2, '001001111010': 1, '00000101111
0': 1, '001010010000': 8, '001010111000': 1, '001010110111': 1, '001011011011': 1, '001010000101': 1,
'001001110101': 1, '001010000011': 2, '001010001011': 123, '001010011001': 1, '001010010111': 1, '0010
10010010': 2, '001101100001': 1, '001010110100': 1, '001010001111': 17, '001010001010': 28, '001100100
011': 1, '001010011011': 1}
max_counts_result =  001010001100
max_counts_result =  652
12 qubits, pi ≈ 3.1411042944785277
```

In [22]:

```python
pi = np.pi

plt.plot(nqs, [pi]*len(nqs), '--r')
plt.plot(nqs, pi_estimates, '.-', markersize=12)
plt.xlim([1.5, 12.5])
plt.ylim([1.5, 4.5])
plt.legend(['$\pi$', 'estimate of $\pi$'])
plt.xlabel('Number of qubits', fontdict={'size':20})
plt.ylabel('$\pi$ and estimate of $\pi$', fontdict={'size':20})
plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.show()
```