Introduction to Quantum Machine Learning

Building Your First Quantum Circuit

We will be using IBM Qiskit libraries with IBM Quantum Simulation and a REAL IBM QUANTUM COMPUTER

https://www.ibm.com/quantum-computing

This Notebook Built by Kinetic Labs

Import Libraries and set up IBM Qiskit

In [1]:
```
import numpy as np
from qiskit import *
from qiskit.providers.aer import QasmSimulator
from qiskit.visualization import plot_histogram
```

Building the circuit

Building a basic circit needed for your first program is the QuantumCircuit. We begin by creating a `QuantumCircuit` comprised of 2 qubits.

In [2]:
```
circuit = QuantumCircuit(2, 2)
```

In [3]:
```
circuit.draw()
```

Out[3]:
```
q_0:

q_1:

c: 2/
```

Adding Gates

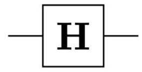
The Hadamard gate (H-gate) is a fundamental quantum gate.

It allows us to move away from the poles of the Bloch sphere and create a superposition of |0⟩ | 0 ⟩ and |1⟩ | 1 ⟩. It has the matrix: We can see that this performs the transformations below:

We create the circuit with its registers, you can add gates ("operations") to manipulate the registers.below is an example of a quantum circuit that makes a two-qubit GHZ state

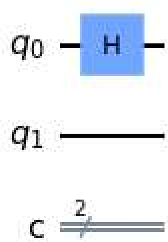
$$|\psi\rangle = (|00\rangle + |11\rangle) / \sqrt{2}.$$

To create such a state, we start with a two-qubit quantum register. By default, each qubit is initialized to $|0\rangle$. To make the GHZ state, we apply the following gates:

Hadamard (H)  $\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$

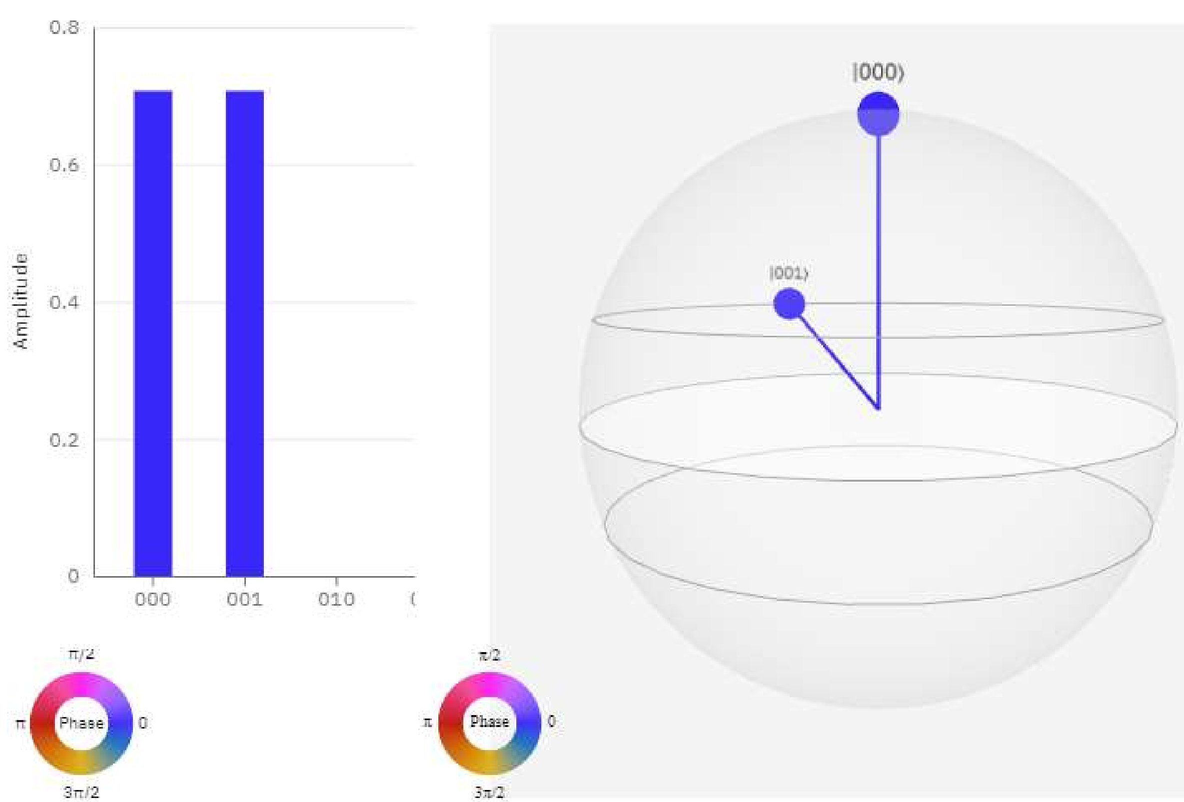
In [4]:
```
circuit.h(0)
circuit.draw(output="mpl")
```

Out[4]:



Bellow is an illastration on how the Hadamard H-Gate Creates a superposition of |0⟩ | 0 ⟩ and |1⟩ | 1 ⟩ on the Qubit.

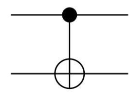
This visulazation illastration can be found using the IBM composer at https://quantum-computing.ibm.com/composer



Complete the Circuit

The below output of the complete circuit shows 2x QuBits q0 and q1 with the circuit.measure([0,1], [0,1]) so that the H-gate with the addition of a CNOT gate will allow quantum output from the QuBits to be translated to Classical Bits.

Controlled Not (CNOT, CX)

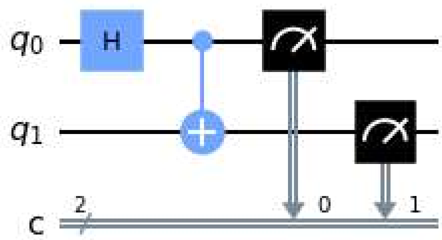


$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In [5]:
```
%matplotlib inline
```

In [6]:
```
circuit.cx(0,1) # 0-> # control qubit, 1-> target qubit
circuit.measure([0,1], [0,1])
circuit.draw(output="mpl")
```

Out[6]:



Attaching the built Circuit to IBM Simulated Quantum Simulator

In [7]:
```
# List avalible IBM quantum simulation computers
Aer.backends()
```

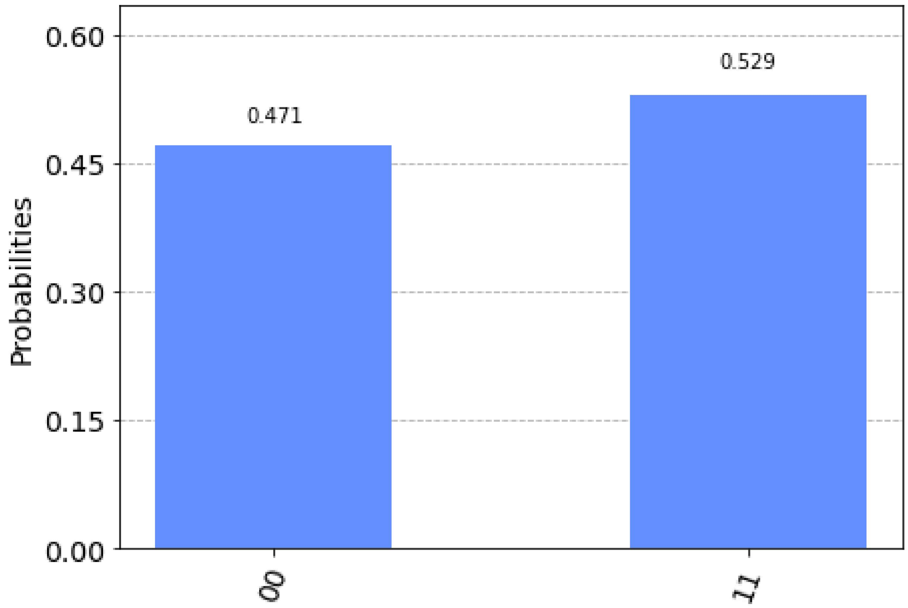
Out[7]:
```
[AerSimulator('aer_simulator'),
 AerSimulator('aer_simulator_statevector'),
 AerSimulator('aer_simulator_density_matrix'),
 AerSimulator('aer_simulator_stabilizer'),
 AerSimulator('aer_simulator_matrix_product_state'),
 AerSimulator('aer_simulator_extended_stabilizer'),
 AerSimulator('aer_simulator_unitary'),
 AerSimulator('aer_simulator_superop'),
 QasmSimulator('qasm_simulator'),
 StatevectorSimulator('statevector_simulator'),
 UnitarySimulator('unitary_simulator'),
 PulseSimulator('pulse_simulator')]
```

In [8]:
```
# Build a simulator from IBM List of available simulators
simulator = Aer.get_backend("qasm_simulator")
```

In [9]:
```
result = execute(circuit,backend=simulator).result()
```

In [10]:
```
plot_histogram(result.get_counts(circuit))
```

Out[10]:



Above we have even distobution |0⟩ | 0 ⟩ and |1⟩ | 1 ⟩

Kinetic Data Science

making data meaningful