Installing Libraries

!pip install qiskit

!pip install pylatexenc

!pip install qiskit-aer

!pip install qiskit-ibmq-provider

# Importing Libraries

from qiskit import \*

from qiskit.tools.monitor import job\_monitor

from qiskit.tools.visualization import plot\_state\_qsphere

from qiskit.visualization import plot\_histogram, plot\_bloch\_multivector, array\_to\_latex

3 Types of Output -> Simulator

1. Statevector
2. Unitary
3. Results after measurement

# Result After Measurement

# Build a Circuit

## Create a circuit

circuit = QuantumCircuit(1,1)

#here we are taking 1 qbit measure and 1 cbit to store the result in

## Apply Gate

circuit.h(0)

# applying Hadamard gate gets the qbit into a superposition which can be used

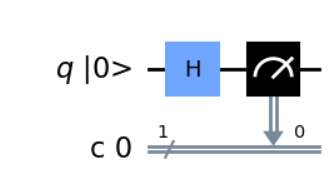
# for fixed state

# apply measurement

circuit.measure(0, 0)

# draw circuit

circuit.draw(output = 'mpl', initial\_state=True)



# Select Simulator

# Select a Simulator

simulator = Aer.get\_backend('aer\_simulator')

# assemble circuit

# by assembling we get a quantum object

qobj = assemble(circuit)

# run on simulator

job = simulator.run(qobj)

# monitor job

job\_monitor(job)

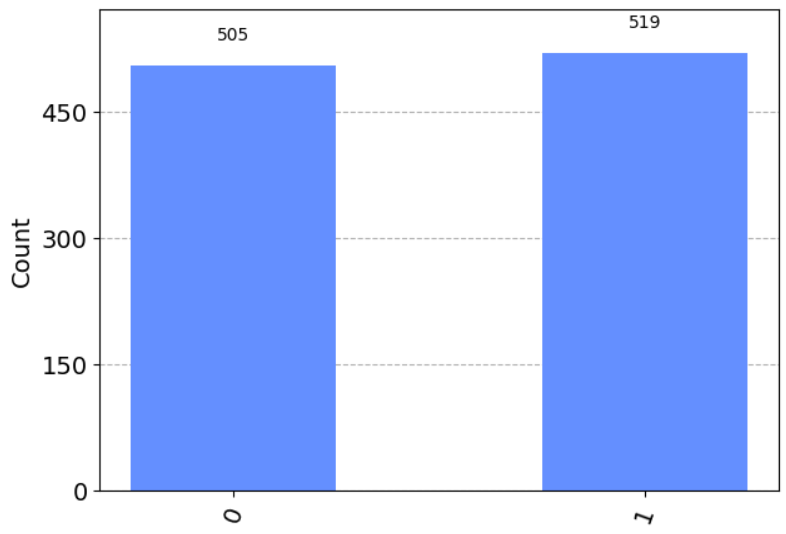
# get result

result = job.result()

# getting the histogram data

count = result.get\_counts()

Job Status: job has successfully run



In simulation wise, it shouldve given 50/50 result, in reality, the quantum computers are near perfection but not perfect. There are still errors and its being fixed and updated, the theory is also updated with time.

# Build A ****Circuit****

## Creating the same circuit again (Not Necessary)

circuit = QuantumCircuit(1,1)

#here we are taking 1 qbit measure and 1 cbit to store the result in

## Apply Gate

circuit.h(0)

# applying Hadamard gate gets the qbit into a superposition which can be used

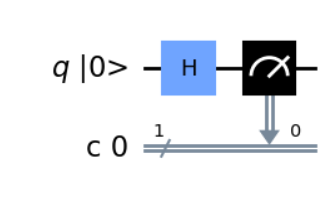
# for fixed state

# apply measurement

circuit.measure(0, 0)

# draw circuit

circuit.draw(output = 'mpl', initial\_state=True)



# Run on Real Quantum Device

## import library for QC (Quantum Computer)

from qiskit import IBMQ

## save API Token

IBMQ.save\_account('1192be126933d579aedd7f316c9a298a4be662cf797f468631995e273bc5d39c9b7cfa3288700afdb3b74428cb88b96b77ea6c7ac21f606397212ea6b3fabb07')

## load account

IBMQ.load\_account()

# select provider

provider = IBMQ.get\_provider('ibm-q')

## select quantum device

# you can change the computer based on the names on IBM website

qcomp = provider.get\_backend('simulator\_stabilizer')

## run on real device

# to execute we call the execute F^n that has two arguments

# one is the circuit we want and the other is the platform we want to

# simulate on, in our case its a real device

# this will return a job

job = execute(circuit, backend = qcomp)

## monitor the job

job\_monitor(job)

## get result

result = job.result()

## get histogram data

count = result.get\_counts()

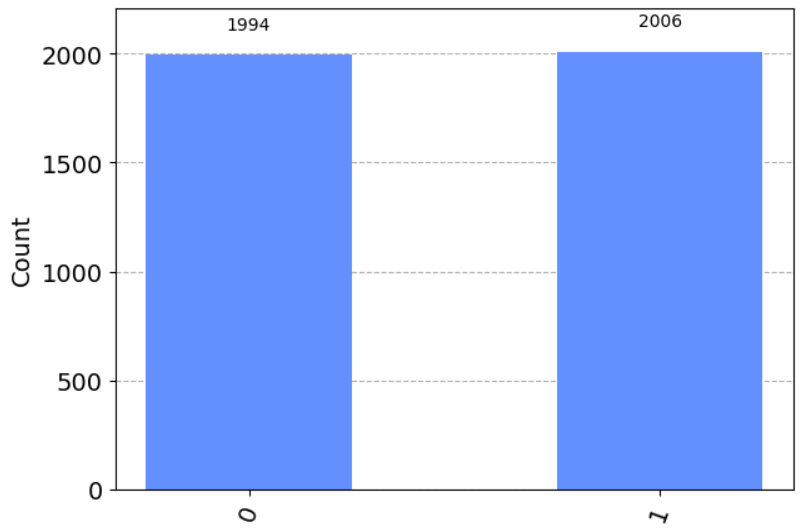
# Note: if it takes longer to execute the job, just change the

# machine, Tip => use the one with 5000 qubits!!

Job Status: job has successfully run

# Visualize the result after measurement

plot\_histogram(count)



# New Circuit

Trying to build a bit complex circuit...

## Create a circuit

circuit = QuantumCircuit(2, 2)

#here we are taking 2 qbit measure and 2 cbit to store the result

## Apply Gate

circuit.h(0)

# applying Hadamard gate gets the qbit into a superposition which can be used

# for fixed state

## Applying another gate here i.e., c not gate

# C not gate or cx gate is a multi qbit gate for that

# we need more than one qbit circuit

# its kinda like a not gate, the state of target qbit is

# changed to another state and stored into anothe qbit

circuit.cx(0, 1)

# here the argument of cx gate is, the first qbit is the control qbit

# and the second one is the target qbit

# in our case control qbit is 0 and target is 1

# apply measurement

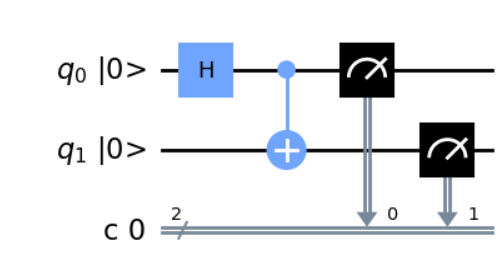
# for more than one qbit we have to measure in the following way

circuit.measure(0, 0)

circuit.measure(1, 1)

# draw circuit

circuit.draw(output = 'mpl', initial\_state=True)



# Simulating

# Select a Simulator

simulator = Aer.get\_backend('aer\_simulator')

# assemble circuit

# by assembling we get a quantum object

qobj = assemble(circuit)

# run on simulator

job = simulator.run(qobj)

# monitor job

job\_monitor(job)

# get result

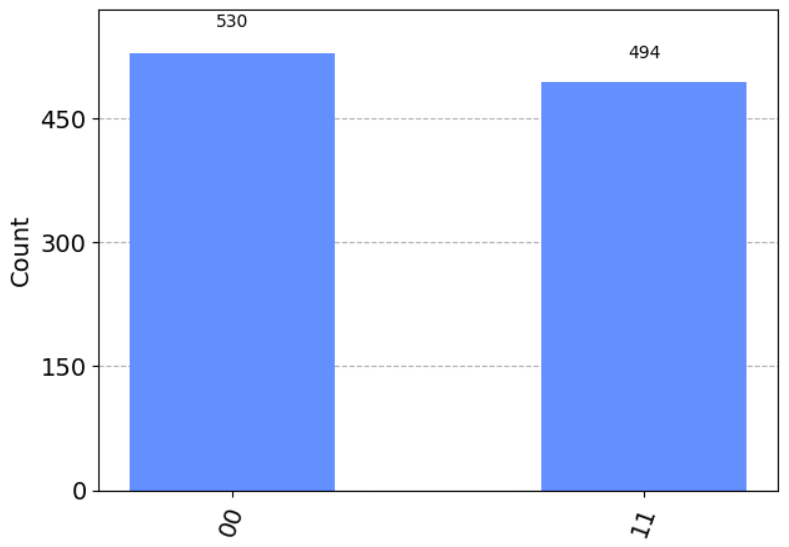
result = job.result()

# getting the histogram data

count = result.get\_counts()

Job Status: job has successfully run

# Visualizing



# Now running this circuit in a real Quantum Computer

## import library for QC (Quantum Computer)

from qiskit import IBMQ

## save API Token

IBMQ.save\_account('1192be126933d579aedd7f316c9a298a4be662cf797f468631995e273bc5d39c9b7cfa3288700afdb3b74428cb88b96b77ea6c7ac21f606397212ea6b3fabb07')

## load account

IBMQ.load\_account()

# select provider

provider = IBMQ.get\_provider('ibm-q')

## select quantum device

# you can change the computer based on the names on IBM website

qcomp = provider.get\_backend('simulator\_stabilizer')

## run on real device

# to execute we call the execute F^n that has two arguments

# one is the circuit we want and the other is the platform we want to

# simulate on, in our case its a real device

# this will return a job

job = execute(circuit, backend = qcomp)

## monitor the job

job\_monitor(job)

## get result

result = job.result()

## get histogram data

count = result.get\_counts()

# Note: if it takes longer to execute the job, just change the

# machine, Tip => use the one with 5000 qubits!!

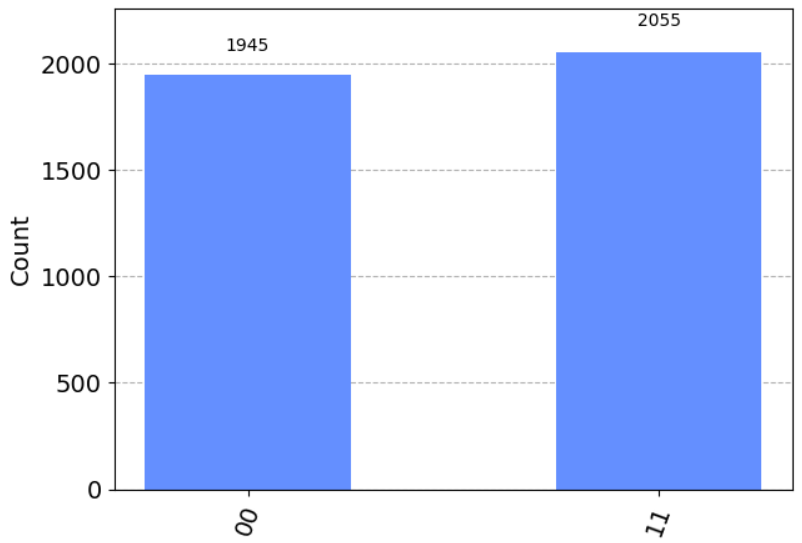
Job Status: job has successfully run

plot\_histogram(count)

# sometimes, all the states are shown i.e., 00 01 10 11 states

# this means that the super positions is broken

# and are into a bell state



# select simulator

simulator = Aer.get\_backend('aer\_simulator')

# save statevector

circuit.save\_statevector()

# assemble circuit

qobj = assemble(circuit)

# run simulator

job = simulator.run(qobj)

# get result

result = job.result()

# get final state

final\_state = result.get\_statevector()

plot\_bloch\_multivector(final\_state)

