x=5

y="hello , world !"

print(x)

print(y)

Output:



x=-1

if x > 0:

    print("x is positive")

else:

    print("x is non-positive")

Output:



for i in range(1,5):

    print(i)

i=0

while i < 5:

    print(i)

    i+=1

Output:



def welcome(name):

    return "hello ,"+name

print(welcome("sai"))

Output:



class person:

    def \_\_init\_\_(self,name,age):

        self.name=name

        self.age=age

    def greet(self):

        return "hello,my name is " + self.name ,    "my age is "+ self.age

p1=person("sai","23")

print(p1.greet())

Output:



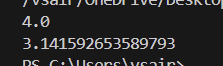
import math

print(math.sqrt(16))

from math import pi

print(pi)

Output:



try:

    print(10/1)

except ZeroDivisionError:

    print("cannot divide by zero")

with open('example.txt','w') as file:

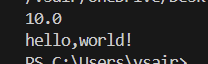
    file.write("hello,world!")

with open('example.txt','r')as file:

    content=file.read()

    print(content)

Output:



mylist=[1,2,3,4,5]

print(mylist)

mylist[2]=7

print(mylist)

mytuple=(1,2,3,4,5)

print(mytuple)

mylist=[1,2,3,4,5]

mylist.append(6)

print(mylist)

mylist=[1,2,3]

mylist.append(4)

print(mylist)

mylist.remove(2)

print(mylist)

mytuple=(1,2,3,2)

print(mytuple.count(2))

print(mytuple.index(3))

mylist=[1,2,3]

mytuple=tuple(mylist)

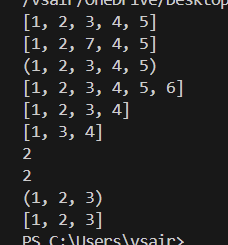
print(mytuple)

mytuple=(1,2,3)

mylist=list(mytuple)

print(mylist)

Output:



class Dog:

    def \_\_init\_\_(self,name,age):

        self.name = name

        self.age = age

def bark(self):

        return f"{self.name} says woof!"

mydog=Dog("buddy",5)

print(mydog.bark())

Output:



class animal:

    def \_\_init\_\_(self,name):

        self.name=name

    def speak(self):

        pass

class dog(animal):

    def speak(self):

        return f"{self.name} says woof!"

class cat(animal):

    def speak(self):

        return f"{self.name} says meow!"

Dog = dog("buddy")

Cat=cat("whiskers")

print(Dog.speak())

print(Cat.speak())

Output:



class person:

    def \_\_init\_\_(self,name,age):

        self.name=name

        self.age=age

    def getage(self):

        return self.age

    def setage(self,age):

        if age>0:

            self.age=age

Person=person("sai",23)

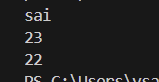
print(Person.name)

print(Person.getage())

Person.setage(22)

print(Person.getage())

Output:



class bird:

    def speak(self):

        return "some generic bird sound"

class parrot(bird):

    def speak(self):

        return "parrot says hello!"

class sparrow(bird):

    def speak(self):

        return "sparrow chirps!"

def makebirdspeak(bird):

    print(bird.speak())

Parrot=parrot()

Sparrow=sparrow()

makebirdspeak(Parrot)

makebirdspeak(Sparrow)

Output:



from abc import ABC, abstractmethod

class Shape(ABC):

    @abstractmethod

    def area(self):

        pass

class Rectangle(Shape):

     def \_\_init\_\_(self, width, height):

        self.width = width

        self.height = height

def area(self):

    return self.width \* self.height

class Circle(Shape):

    def \_\_init\_\_(self,radius):

        self.radius=radius

    def area(self):

        return 3.14 \* self.radius \*\* 2

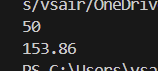
rectangle = Rectangle(5,10)

circle=Circle(7)

print(rectangle.area())

print(circle.area())

Output:



import numpy as np

arr=np.array([1,2,3,4,5])

print(arr \* 2)

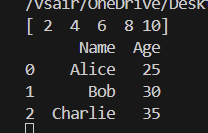
import pandas as pd

data={'Name':['Alice','Bob','Charlie'],'Age':[25,30,35]}

df=pd.DataFrame(data)

print(df)

Output:



import matplotlib.pyplot as plt

plt.plot([1,2,3],[4,5,6])

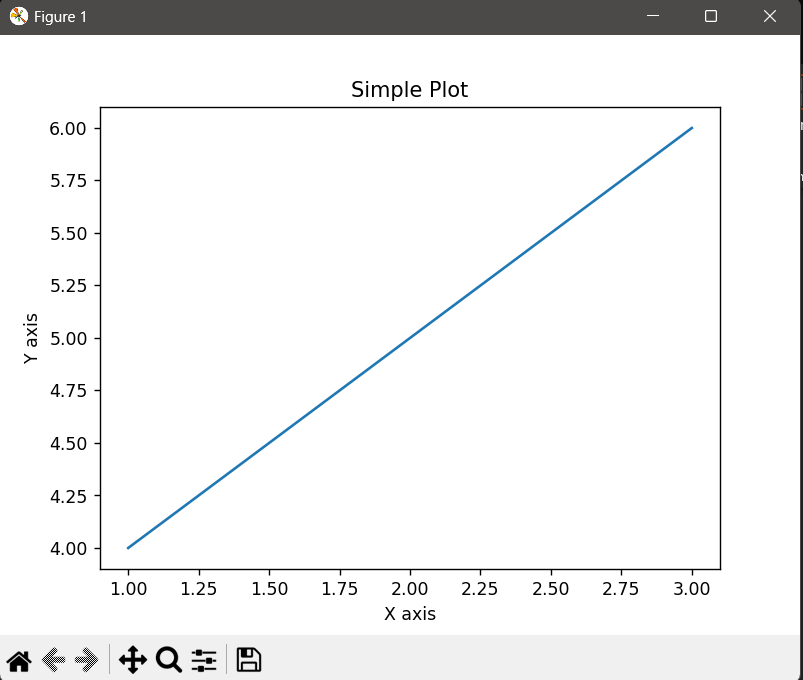
plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.title('Simple Plot')

plt.show()

Output:



import matplotlib.pyplot as plt

import seaborn as sns

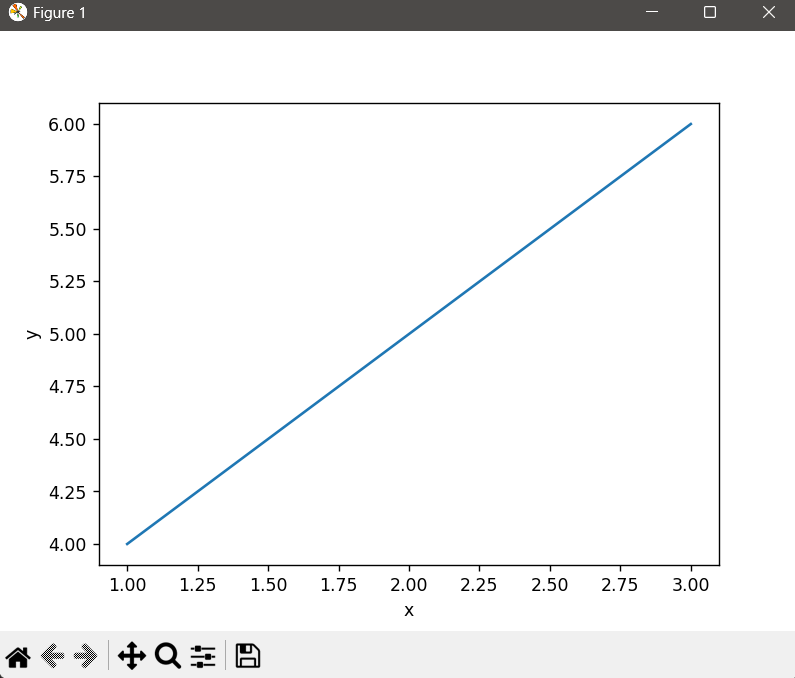
import pandas as pd

df=pd.DataFrame({'x':[1,2,3],'y':[4,5,6]})

sns.lineplot(data=df,x='x',y='y')

plt.show()

Output:



from scipy import stats

t\_stat,p\_value=stats.ttest\_1samp([1,2,3,4,5],3)

print(f"T-statistic:{t\_stat},P\_value:{p\_value}")

Output:



from sklearn.linear\_model import LinearRegression

import numpy as np

model=LinearRegression()

x= np.array([[1],[2],[3],[4],[5]])

y= np.array([1,2,3,4,5])

model.fit(x,y)

print(model.predict([[6]]))

Output:



import numpy as np

import pandas as pd

import statsmodels.api as sm

x=pd.DataFrame({'feature1':[1,2,3],'feature':[4,5,6]})

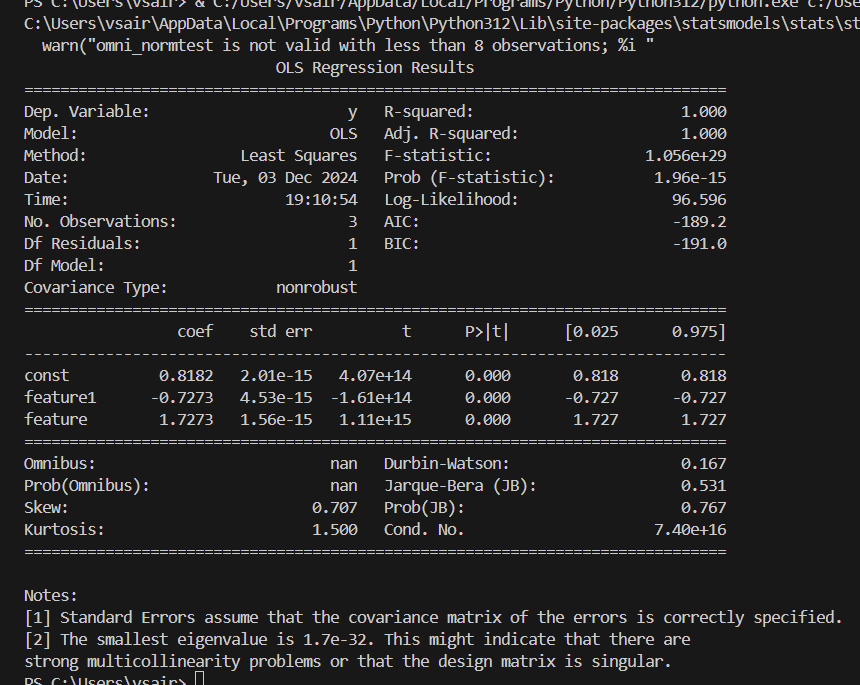
y=pd.Series([7,8,9])

x=sm.add\_constant(x)

model=sm.OLS(y,x).fit()

print(model.summary())

Output:



import pandas as pd

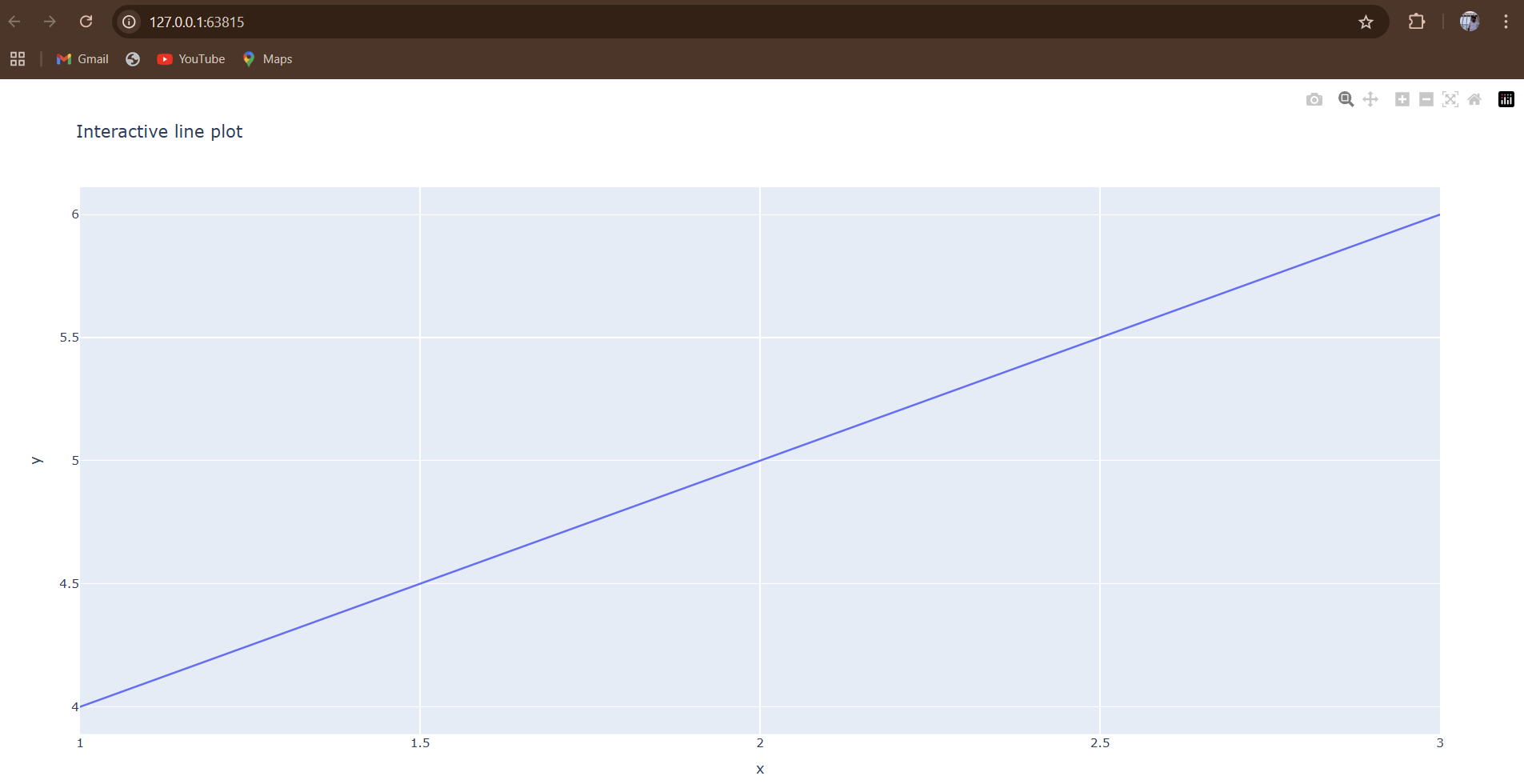
import plotly.express as px

df=pd.DataFrame({'x':[1,2,3],'y':[4,5,6]})

fig=px.line(df,x='x',y='y',title='Interactive line plot')

fig.show()

Output:



import tensorflow as tf

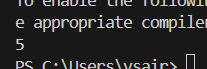
a=tf.constant(2)

b=tf.constant(3)

c=a+b

print(c.numpy())

Output:



import torch as th

a=th.tensor(2)

b=th.tensor(3)

c=a+b

print(c)

Output:

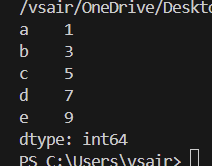


import pandas as pd

s=pd.Series([1,3,5,7,9],index=['a','b','c','d','e'])

print(s)

Output:



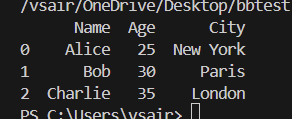
import pandas as pd

data={'Name':['Alice','Bob','Charlie'],'Age':[25,30,35],'City':['New York','Paris','London']}

df=pd.DataFrame(data)

print(df)

Output:



import pandas as pd

data={'Name':['Alice','Bob','Charlie','Alice'],'Age':[25,30,35,25],'City':['New York','Paris','London','New York']}

df=pd.DataFrame(data)

print(df)

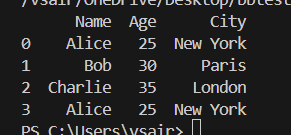
df['Age'].fillna(df['Age'].mean(),inplace=True) #fill missing values

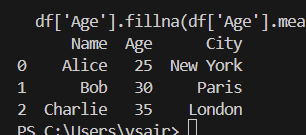
df.drop\_duplicates(inplace=True) #remove duplicates

df['Age']=df['Age'].astype(int) #convert data types

print(df)

Output:





import pandas as pd

data={'Product':['A','B','A','B','C'],

      'Category':['Electronics', 'Furniture', 'Electronics', 'Furniture', 'Kitchen'],

        'Sales': [100, 200, 150, 300, 250]}

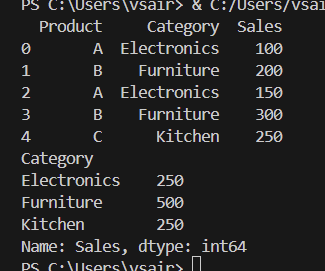
df = pd.DataFrame(data)

print(df)

category\_sales=df.groupby('Category')['Sales'].sum()

print(category\_sales)

Output:



import pandas as pd

customers1 = pd.DataFrame({'CustomerID':[1,2,3],

                         'Name':['Alice','Bob','Charlie'],

                         'City':['New York','Paris','London']})

customers2 = pd.DataFrame({'CustomerID':[2, 3, 4],

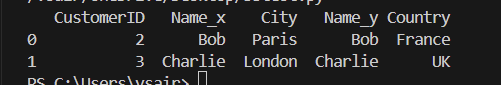
                           'Name': ['Bob', 'Charlie', 'David'],

                           'Country': ['France', 'UK', 'USA']})

merged\_customers= pd.merge(customers1, customers2, on='CustomerID', how='inner')

print(merged\_customers)

Output:



import pandas as pd

date\_range = pd.date\_range(start='1/1/2020', periods=5, freq='D')

stock\_prices = pd.DataFrame({'Date': date\_range, 'Price': [100, 110, 105, 115, 120]})

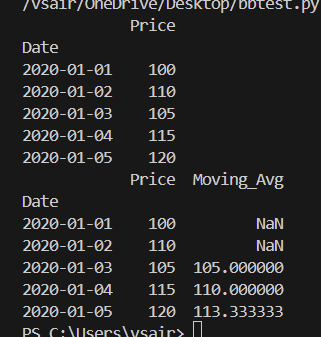
stock\_prices.set\_index('Date', inplace=True)

print(stock\_prices)

stock\_prices['Moving\_Avg'] = stock\_prices['Price'].rolling(window=3).mean()

print(stock\_prices)

Output:



import matplotlib.pyplot as plt

import pandas as pd

data = {'Product': ['A', 'B', 'A', 'B', 'C'],

        'Category': ['Electronics', 'Furniture', 'Electronics', 'Furniture', 'Kitchen'],

        'Sales': [100, 200, 150, 300, 250]}

df = pd.DataFrame(data)

category\_sales = df.groupby('Category')['Sales'].sum()

category\_sales.plot(kind='bar')

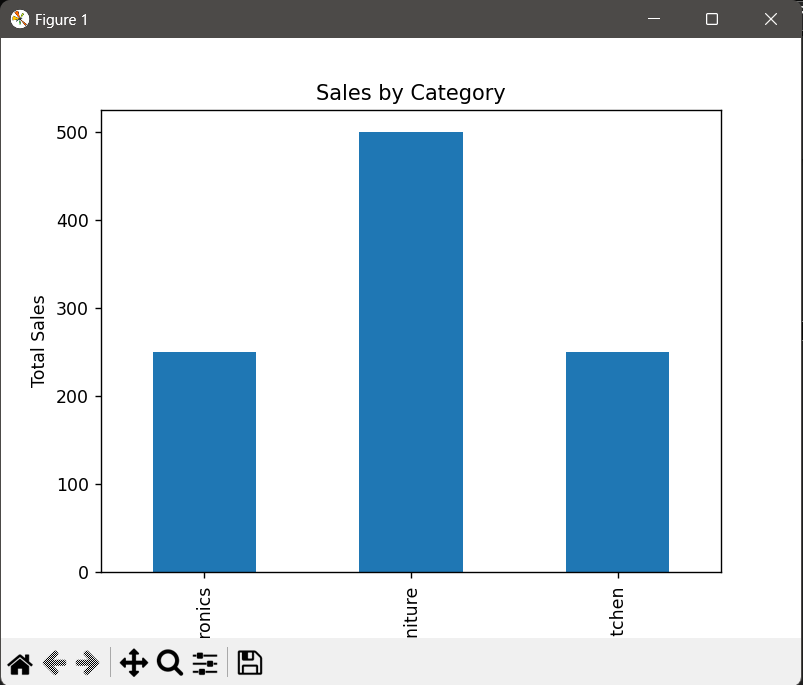
plt.xlabel('Category')

plt.ylabel('Total Sales')

plt.title('Sales by Category')

plt.show()

Output:



import pandas as pd

data = {'A': [1, 2, 3], 'B': [4, 5, 6]}

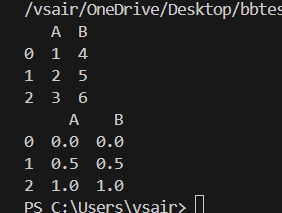
df = pd.DataFrame(data)

print(df)

df\_normalized = df.apply(lambda x: (x - x.min()) / (x.max() - x.min()))

print(df\_normalized)

Output:



import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

train\_images = train\_images.reshape((train\_images.shape[0], 28, 28, 1))

test\_images = test\_images.reshape((test\_images.shape[0], 28, 28, 1))

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

model.fit(train\_images, train\_labels, epochs=5)

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f"Test accuracy: {test\_acc}")

predictions = model.predict(test\_images)

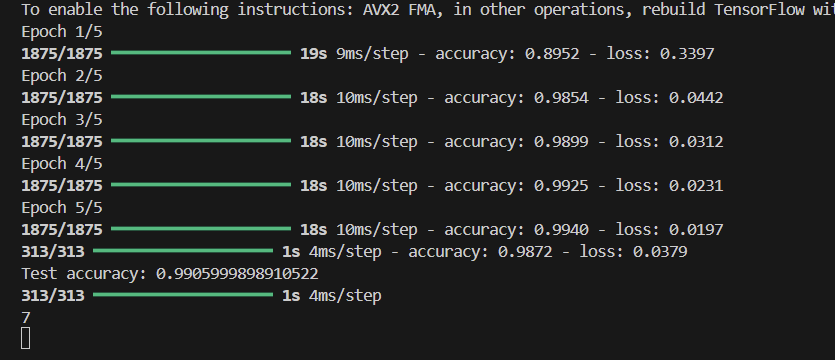
print(np.argmax(predictions[0]))

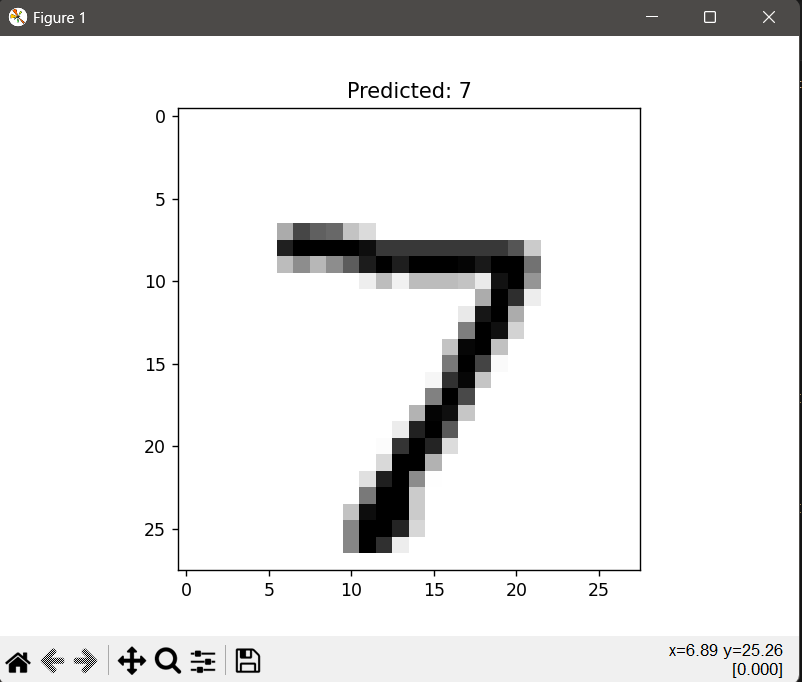
plt.imshow(test\_images[0].reshape(28, 28), cmap=plt.cm.binary)

plt.title(f"Predicted: {np.argmax(predictions[0])}")

plt.show()

Output:





import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

trainset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

trainloader = DataLoader(trainset, batch\_size=64, shuffle=True)

testset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

testloader = DataLoader(testset, batch\_size=64, shuffle=False)

class CNN(nn.Module):

    def \_\_init\_\_(self):

        super(CNN, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(1, 32, 3, 1)

        self.conv2 = nn.Conv2d(32, 64, 3, 1)

        self.fc1 = nn.Linear(24\*24\*64, 128)

        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):

        x = self.conv1(x)

        x = torch.relu(x)

        x = self.conv2(x)

        x = torch.relu(x)

        x = torch.flatten(x, 1)

        x = self.fc1(x)

        x = torch.relu(x)

        x = self.fc2(x)

        return torch.log\_softmax(x, dim=1)

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model = CNN().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 5

for epoch in range(epochs):

    model.train()

    running\_loss = 0.0

    for images, labels in trainloader:

        images, labels = images.to(device), labels.to(device)

        optimizer.zero\_grad()

        output = model(images)

        loss = criterion(output, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

    print(f"Epoch {epoch+1}/{epochs}, Loss: {running\_loss/len(trainloader)}")

model.eval()

correct = 0

total = 0

with torch.no\_grad():

    for images, labels in testloader:

        images, labels = images.to(device), labels.to(device)

        output = model(images)

        \_, predicted = torch.max(output.data, 1)

        total += labels.size(0)

        correct += (predicted == labels).sum().item()

print(f"Test accuracy: {100 \* correct / total}%")

images, labels = next(iter(testloader))

images, labels = images.to(device), labels.to(device)

output = model(images)

\_, predicted = torch.max(output.data, 1)

print(f"Predicted: {predicted[0].item()}")

plt.imshow(images[0].cpu().numpy().squeeze(), cmap='gray')

plt.title(f"Predicted: {predicted[0].item()}")

plt.show()

Output:

