**Name**- Divyam K Deep

**Reg. No**.- RA1911031010096

**Year/Section**- 3rd Yr L1 Section

**Faculty Name**- Dr. P Supraja

# EX 8 : IMPLEMENTATION OF LEARNING ALGORITHMS FOR AN APPLICATION

# Aim:

To implement a supervised learning algorithm.

# Problem Description:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

# Program:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

import seaborn as sns

%matplotlib inline

companies=pd.read\_csv('sample\_data/LinearRegressionExample.csv') x=companies.iloc[:,:-1].values

y=companies.iloc[:,4].values companies.head()

print(x) print(y)

sns.heatmap(companies.corr())

from sklearn.preprocessing import LabelEncoder, OneHotEncoder from sklearn.compose import ColumnTransformer

#Encode State Column labelencoder = LabelEncoder()

x[:,3] = labelencoder.fit\_transform(x[:,3])

ct = ColumnTransformer([("State", OneHotEncoder(), [3])], remainder = 'passthrough') x = ct.fit\_transform(x)

print(x)

x=x[:,1:]

print(x)

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

from sklearn.linear\_model import LinearRegression model\_fit=LinearRegression() model\_fit.fit(x\_train,y\_train)

y\_pred=model\_fit.predict(x\_test) print(y\_pred)

print(model\_fit.coef\_) print(model\_fit.intercept\_)

from sklearn.metrics import r2\_score r2\_score(y\_test, y\_pred)

x\_state = companies.State y\_profit = companies.Profit

from sklearn.preprocessing import LabelEncoder labelencoder = LabelEncoder()

x\_state = labelencoder.fit\_transform(x\_state) print(x\_state)

x\_state = np.array(x\_state).reshape(-1,1) y\_profit = np.array(y\_profit).reshape(-1,1)

print(x\_state.shape) print(y\_profit.shape)

X\_train\_1, X\_test\_1, Y\_train\_1, Y\_test\_1 = train\_test\_split(x\_state, y\_profit, test\_size = 0.2, random\_state=5)

print(X\_train\_1.shape) print(X\_test\_1.shape) print(Y\_train\_1.shape) print(Y\_test\_1.shape)

from sklearn.metrics import mean\_squared\_error reg\_1 = LinearRegression()

reg\_1.fit(X\_train\_1, Y\_train\_1)

y\_train\_predict\_1 = reg\_1.predict(X\_train\_1)

rmse = (np.sqrt(mean\_squared\_error(Y\_train\_1, y\_train\_predict\_1))) r2 = round(reg\_1.score(X\_train\_1, Y\_train\_1),2)

print("The model performance for training set") print('RMSE is {}'.format(rmse))

print('R2 score is {}'.format(r2)) print("\n")

y\_pred\_1 = reg\_1.predict(X\_test\_1)

rmse = (np.sqrt(mean\_squared\_error(Y\_test\_1, y\_pred\_1))) r2 = round(reg\_1.score(X\_test\_1, Y\_test\_1),2)

print("The model performance for testing set") print(" ") print("Root Mean Squared Error: {}".format(rmse)) print("R^2: {}".format(r2))

print("\n")

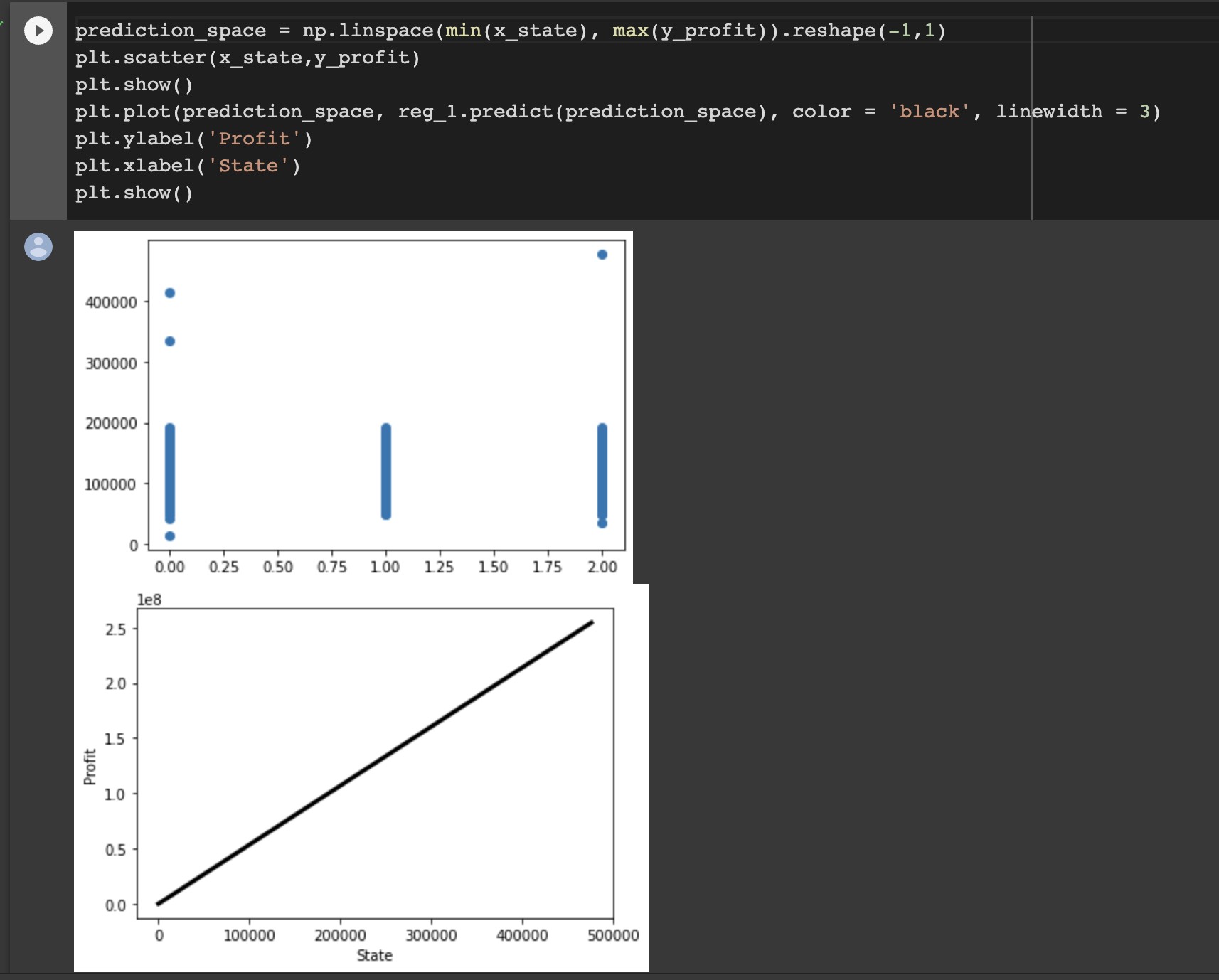
prediction\_space = np.linspace(min(x\_state), max(y\_profit)).reshape(-1,1) plt.scatter(x\_state,y\_profit)

plt.show()

plt.plot(prediction\_space, reg\_1.predict(prediction\_space), color = 'black', linewidth = 3) plt.ylabel('Profit')

plt.xlabel('State') plt.show()

# Output Screenshot:



**Result:** The program was executed successfully.

# EX 9 : IMPLEMENTATION OF NLP PROGRAM TEXT CLASSIFICATION

**Aim:**

Solve NLP program for text classification

## Problem Description:

Text clarification is the process of categorizing the text into a group of words. By using NLP, text classification can automatically analyze text and then assign a set of predefined tags or categories based on its context. NLP is used for sentiment analysis, topic detection, and language detection.

# Program:

import pandas as pd import numpy as np

from nltk.tokenize import word\_tokenize from nltk import pos\_tag

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer from sklearn.preprocessing import LabelEncoder from collections import defaultdict

from nltk.corpus import wordnet as wn

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn import model\_selection, naive\_bayes, svm from sklearn.metrics import accuracy\_score np.random.seed(500)

Corpus = pd.read\_csv(r"https://raw.githubusercontent.com/Gunjitbedi/Text- Classification/master/corpus.csv",encoding='latin-1')

# Step - a : Remove blank rows if any.

Corpus['text'].dropna(inplace=True)

# Step - b : Change all the text to lower case. This is required as python interprets 'dog' and 'DOG' differently

Corpus['text'] = [entry.lower() for entry in Corpus['text']]

# Step - c : Tokenization : In this each entry in the corpus will be broken into set of words print(Corpus['text'])

import nltk nltk.download('punkt')

# Step - c : Tokenization : In this each entry in the corpus will be broken into set of words Corpus['text']= [word\_tokenize(entry) for entry in Corpus['text']]

print(Corpus['text']) import nltk nltk.download('wordnet')

# Step - d : Remove Stop words, Non-Numeric and perfom Word Stemming/Lemmenting.

# WordNetLemmatizer requires Pos tags to understand if the word is noun or verb or adjective etc. By default it is set to Noun

tag\_map = defaultdict(lambda : wn.NOUN) tag\_map['J'] = wn.ADJ

tag\_map['V'] = wn.VERB tag\_map['R'] = wn.ADV import nltk

nltk.download('averaged\_perceptron\_tagger') import nltk

nltk.download('stopwords')

for index,entry in enumerate(Corpus['text']):

# Declaring Empty List to store the words that follow the rules for this step Final\_words = []

# Initializing WordNetLemmatizer() word\_Lemmatized = WordNetLemmatizer()

# pos\_tag function below will provide the 'tag' i.e if the word is Noun(N) or Verb(V) or something else.

for word, tag in pos\_tag(entry):

# Below condition is to check for Stop words and consider only alphabets if word not in stopwords.words('english') and word.isalpha():

word\_Final = word\_Lemmatized.lemmatize(word,tag\_map[tag[0]]) Final\_words.append(word\_Final)

# The final processed set of words for each iteration will be stored in 'text\_final' Corpus.loc[index,'text\_final'] = str(Final\_words)

print(Corpus['text\_final'])

Train\_X, Test\_X, Train\_Y, Test\_Y = model\_selection.train\_test\_split(Corpus['text\_final'],Corpus['label'],test\_size=0.3)

Encoder = LabelEncoder()

Train\_Y = Encoder.fit\_transform(Train\_Y) Test\_Y = Encoder.fit\_transform(Test\_Y) Tfidf\_vect = TfidfVectorizer(max\_features=5000) Tfidf\_vect.fit(Corpus['text\_final'])

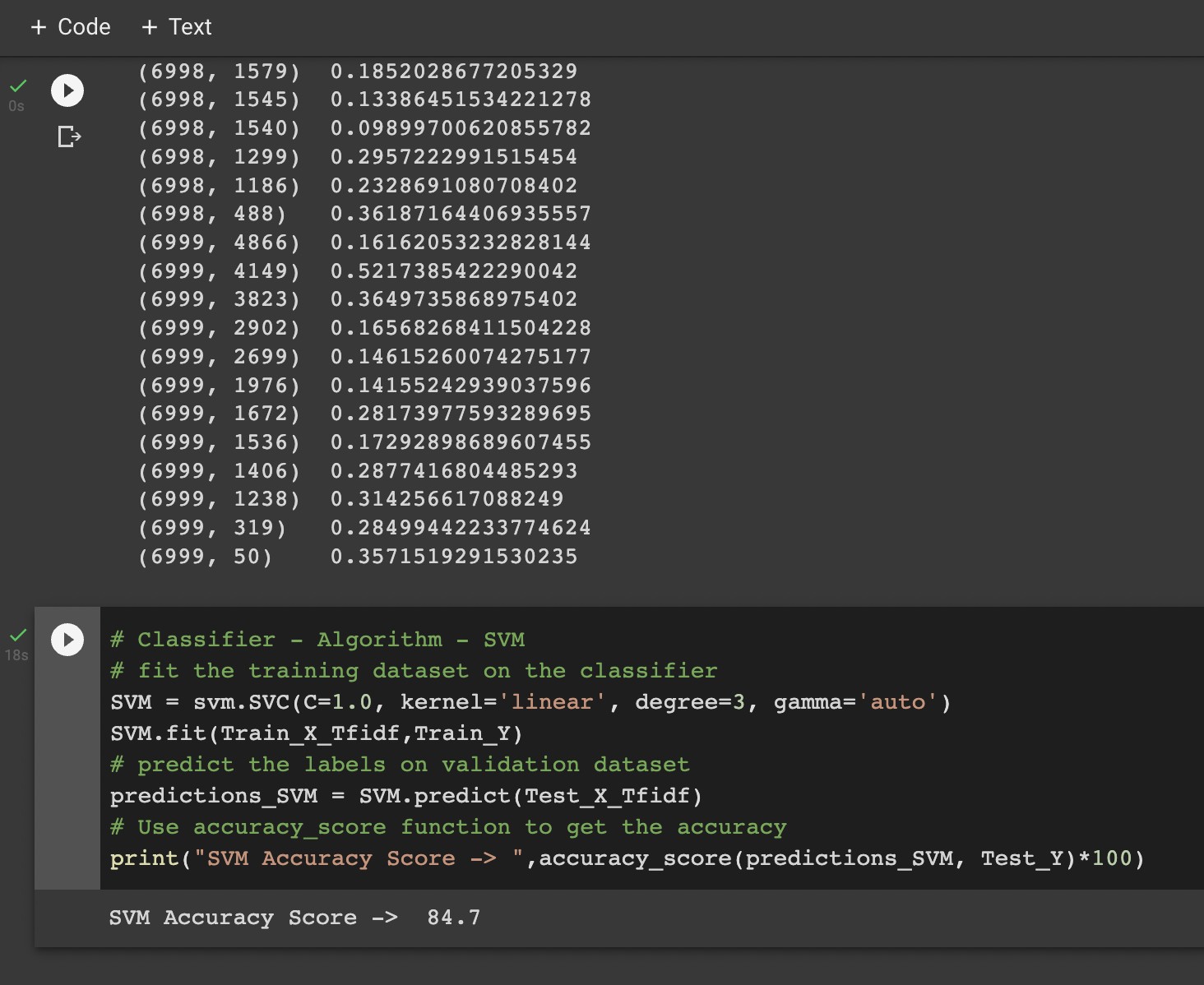
Train\_X\_Tfidf = Tfidf\_vect.transform(Train\_X) Test\_X\_Tfidf = Tfidf\_vect.transform(Test\_X) print(Tfidf\_vect.vocabulary\_) print(Train\_X\_Tfidf)

SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto') SVM.fit(Train\_X\_Tfidf,Train\_Y)

# predict the labels on validation dataset predictions\_SVM = SVM.predict(Test\_X\_Tfidf) # Use accuracy\_score function to get the accuracy

print("SVM Accuracy Score -> ",accuracy\_score(predictions\_SVM, Test\_Y)\*100)

# Output Screenshot:



**Result:** The program was executed successfully and accuracy was found.

# EX10 : APPLYING DEEP LEARNING METHODS TO SOLVE AN APPLICATION - CNN ALGORITHM

**Aim:** To write a convolutional neural network algorithm program using deep learning.

**Problem Description: I**n deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery.

# Program:

import tensorflow as tf

from tensorflow.keras import datasets, layers, models import matplotlib.pyplot as plt

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

# Normalize pixel values to be between 0 and 1

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10)) for i in range(25):

plt.subplot(5,5,i+1) plt.xticks([])

plt.yticks([]) plt.grid(False)

plt.imshow(train\_images[i])

# The CIFAR labels happen to be arrays, # which is why you need the extra index plt.xlabel(class\_names[train\_labels[i][0]])

plt.show()

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(layers.MaxPooling2D((2, 2)))

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

model.add(layers.MaxPooling2D((2, 2)))

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

model.summary()

model.add(layers.Flatten()) model.add(layers.Dense(64, activation='relu')) model.add(layers.Dense(10))

model.summary()

model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), metrics=['accuracy'])

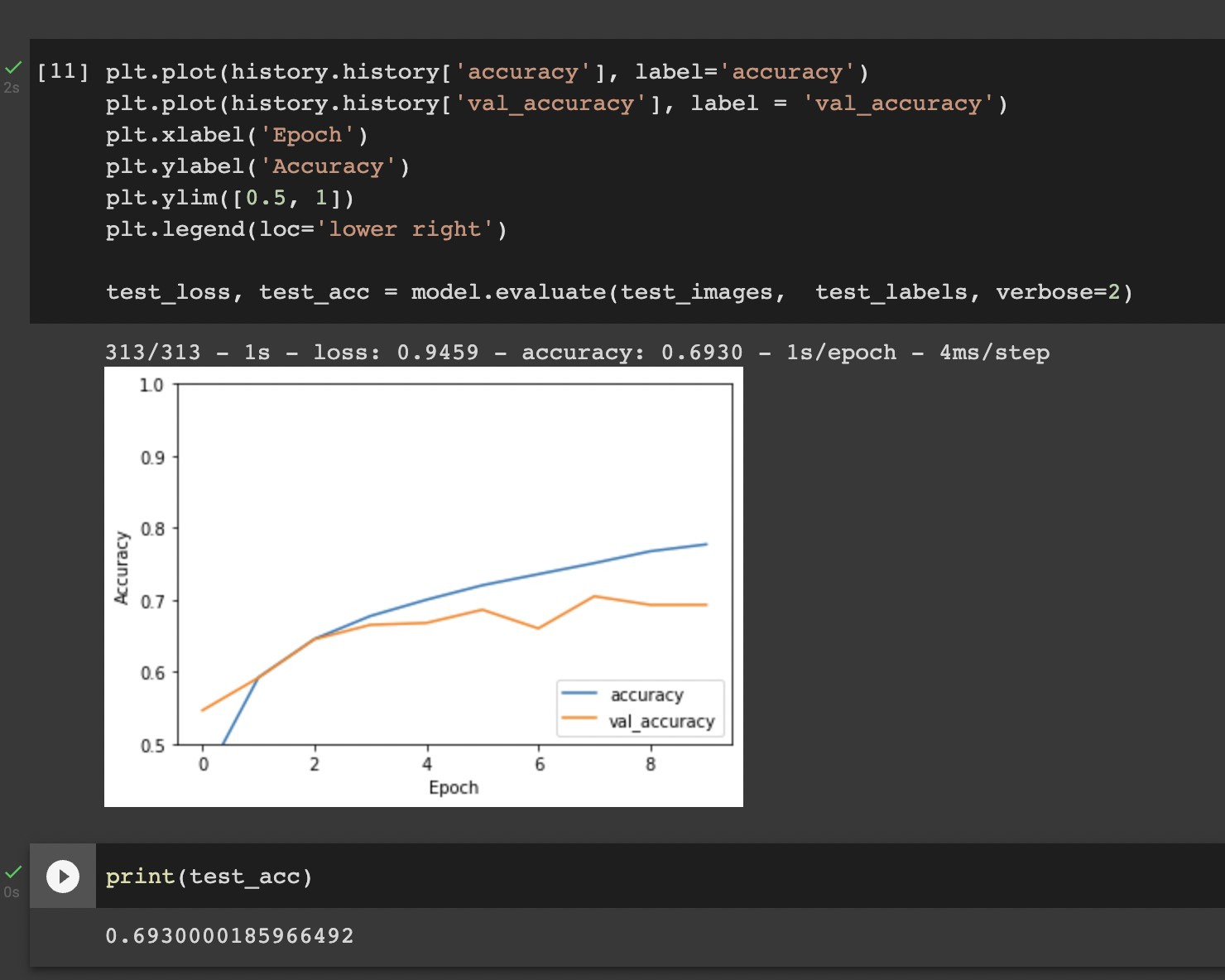
history = model.fit(train\_images, train\_labels, epochs=10, validation\_data=(test\_images, test\_labels))

plt.plot(history.history['accuracy'], label='accuracy') plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy') plt.xlabel('Epoch')

plt.ylabel('Accuracy') plt.ylim([0.5, 1]) plt.legend(loc='lower right')

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2) print(test\_acc)

# Output Screenshot:



**Result:** The program was executed successfully and accuracy was found.