1. Implement the basic logic gates AND & OR using McCullough Pit s model. Draw the linear separability line for the above Boolean functions.

2.
a) Implement a linear classifier (binary) for the given input data using multi layer perceptron using TensorFlow.

b) John Successfully implemented AND & OR gates using single layer perceptron but was unable to implement XOR Gate. He got to know that it can be implemented by multi layer perceptron using Tensor Flow.

3.
a) This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them.The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no/less chance of heart attack and 1 = more chance of heart attack *Hint: Use ANN for performing classification(Optimizer: Adam)*
b) Perform ANN regression for the given data from 1c

Post: Use stochastic gradient descent optimizer for the similar classification problem above and compare the results of both the models.

4.
a) In this dataset, there are various factors given, which are involved when a patient is hospitalized. On the basis of these factors, predict whether the patient will survive or not. But it has 85 columns so, perform Dimensionality reduction using PCA.
b) Normalize the data in the given dataset and perform classification using ANN.

5. Perform Multi class classification on MNIST dataset using ANN.(Note: Use Softmax activation function for the output layer with 3-4 hidden layers consisting of ReLU activation function also evaluate the performance by using confusion matrix.)

6.

a) Build a prediction model that will perform the following:

i. Classify if a customer is going to churn or not

ii. Preferably and based on model performance, choose a model to improve the accuracy by adding some dropout layers and regularization.

7. Build CNN Model for COVID-19 Disease Detection

Post:

1) Use 3*3 filters and perform convolution operation for the matrix. The Given 3*3 matrix is:

A=[[1,1,1,0,0,0], [1,1,0,1,0,0], [1,0,1,0,1,0],[[0,1,1,0,0,1], [0,0,1,0,1,0], [1,0,0,0,0,0]]

8. Build a deep learning model which classifies cats and dogs using CNN.

9. The stock price of the present day can be predicted by the stock prices obtained for past 50 days. Using Simple RNN, train the model with "Google_stock_price_train.csv" dataset, predict the stock prices for the dates given in "Google_stock_price_train.csv" dataset and visualize the actual, predicted prices using matplotlib.

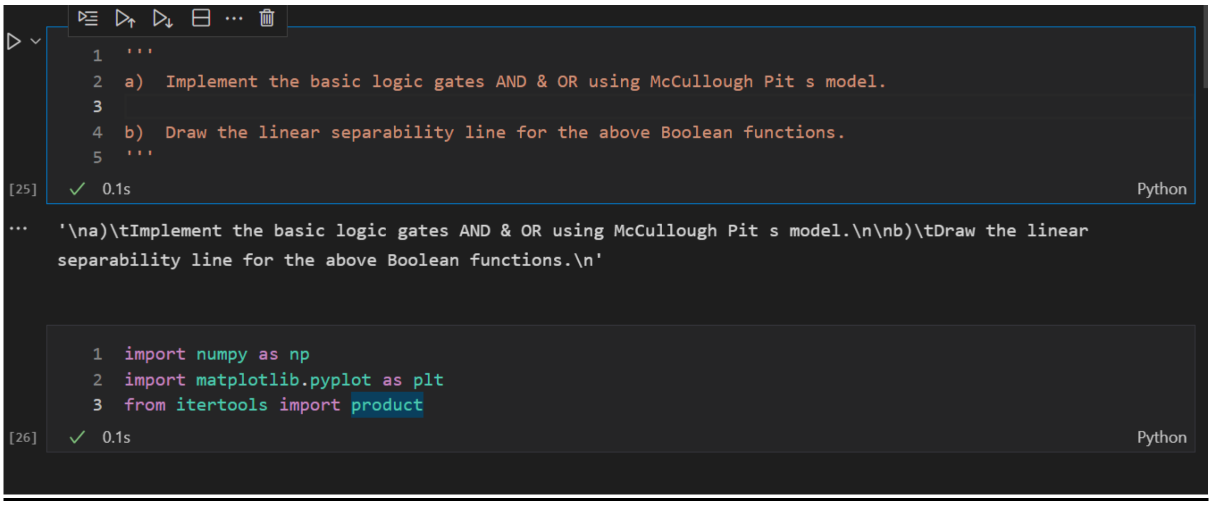
10. Using LSTM train the model with train dataset, predict the covid death cases and confirmed cases. Visualize the actual, predict data using matplotlib. Use 20days time stamp.

Post: Predict the weather for the dates given in the test dataset and visualize the actual, predict data using matplotlib. Use 20 day's time stamp.

11. Build a simple auto encoder on MNIST data using keras library and help them in building a simple Auto Encoder. (Note: Encode the image into from 784 to 32 pixels).

12. Build a simple denoisy auto encoder using keras on MNIST.

LAB-1



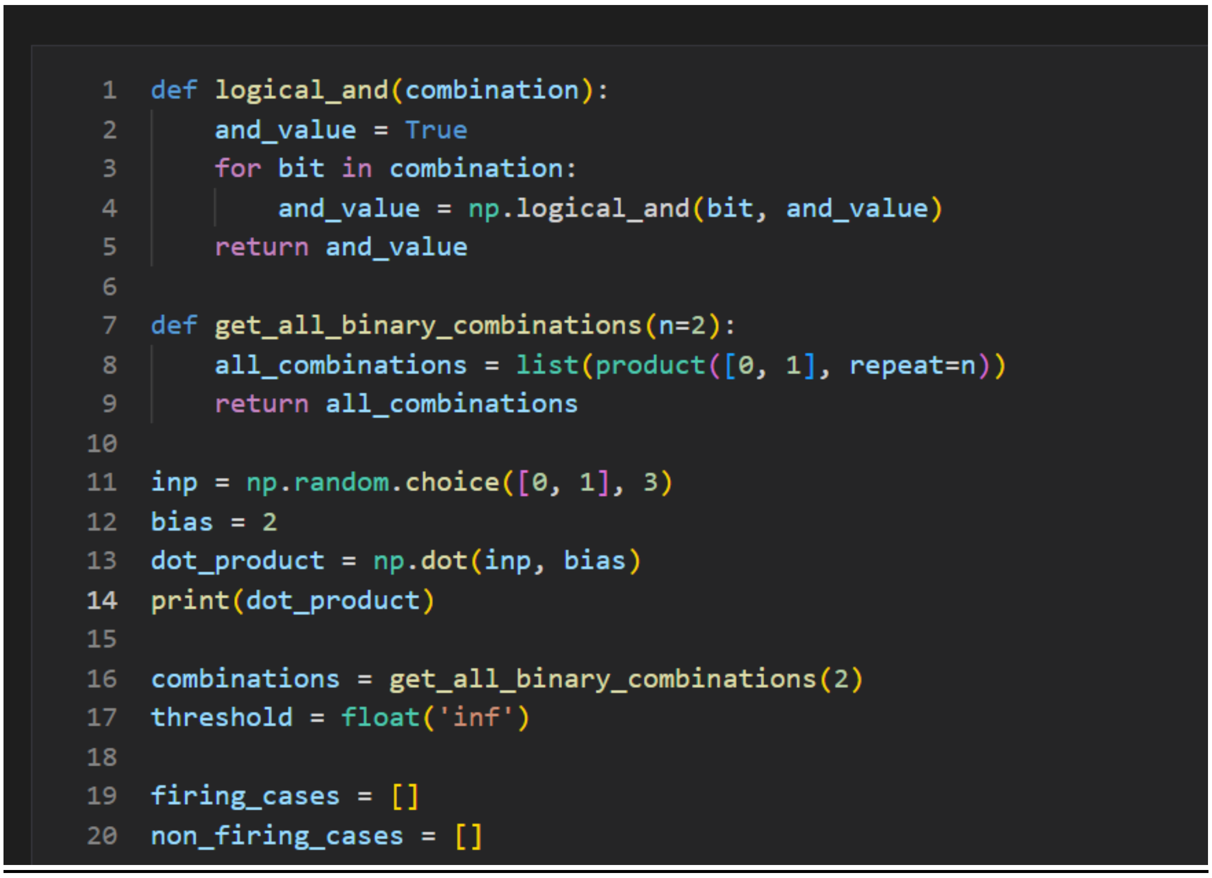
```
'''
a)  Implement the basic logic gates AND & OR using McCullough Pit s model.

b)  Draw the linear separability line for the above Boolean functions.
'''
```

[25] 0.1s Python

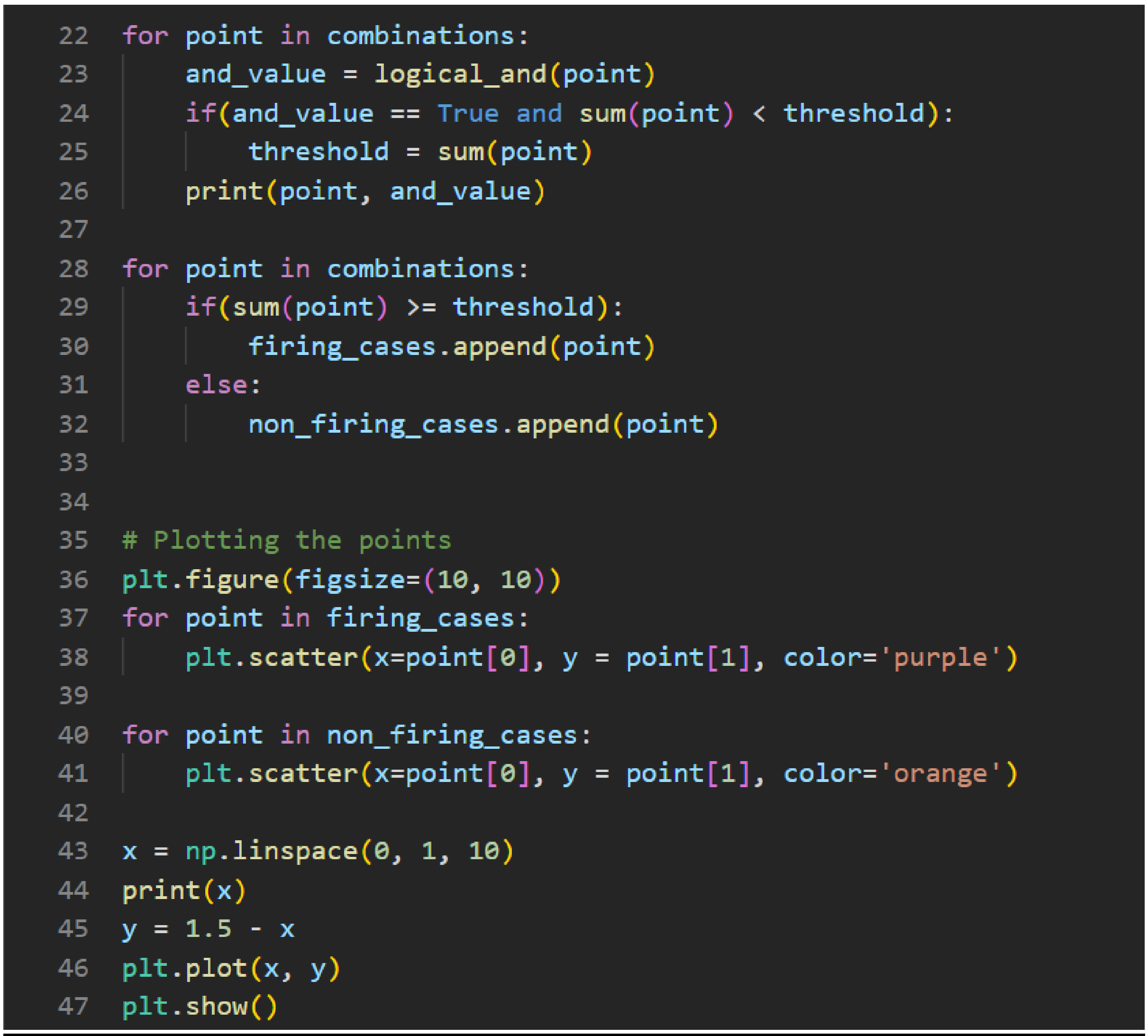
```
'\na)\tImplement the basic logic gates AND & OR using McCullough Pit s model.\n\nb)\tDraw the linear
separability line for the above Boolean functions.\n'
```

```python
import numpy as np
import matplotlib.pyplot as plt
from itertools import product
```

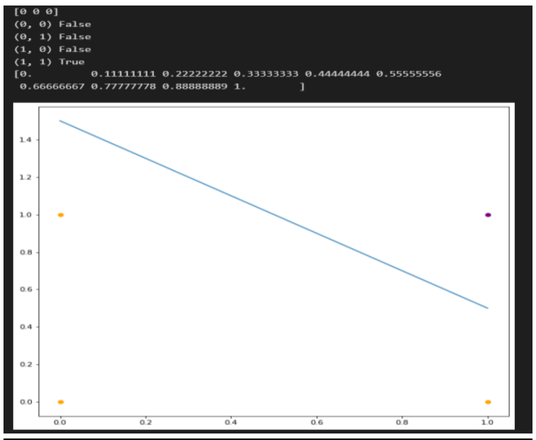
[26] 0.1s Python



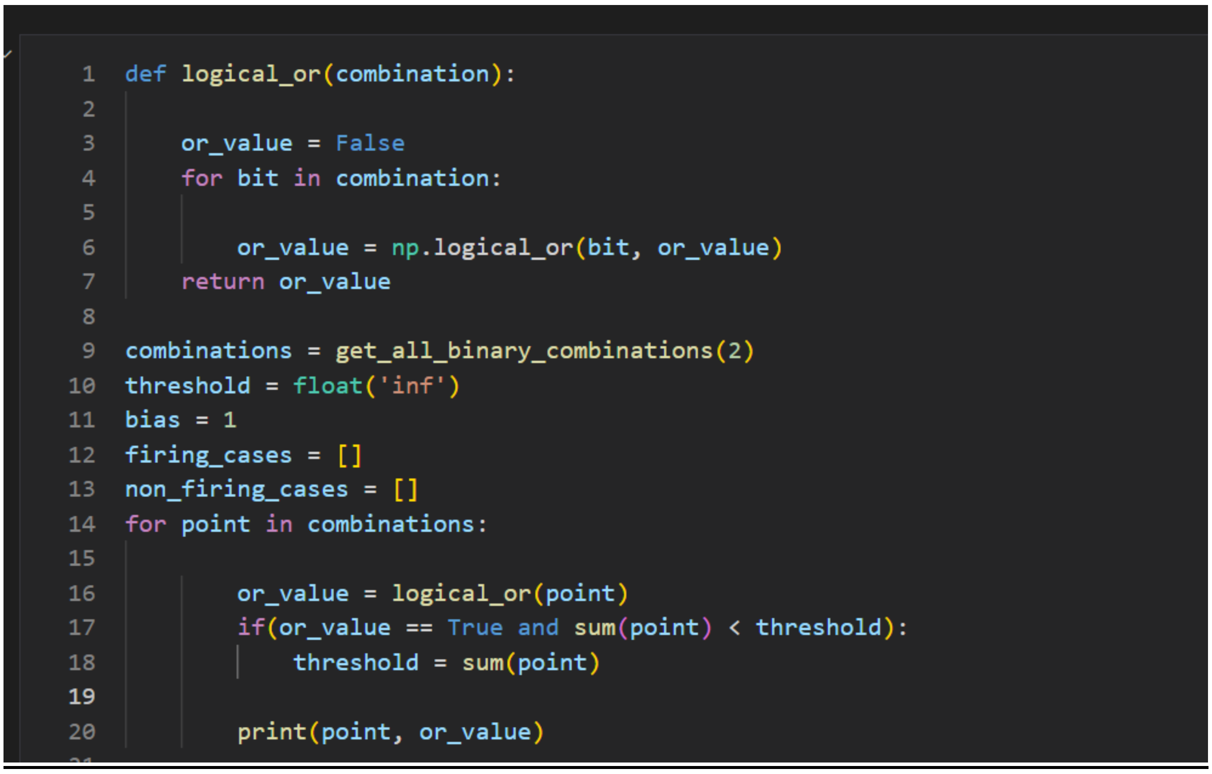
```python
def logical_and(combination):
    and_value = True
    for bit in combination:
        and_value = np.logical_and(bit, and_value)
    return and_value

def get_all_binary_combinations(n=2):
    all_combinations = list(product([0, 1], repeat=n))
    return all_combinations

inp = np.random.choice([0, 1], 3)
bias = 2
dot_product = np.dot(inp, bias)
print(dot_product)

combinations = get_all_binary_combinations(2)
threshold = float('inf')

firing_cases = []
non_firing_cases = []
```



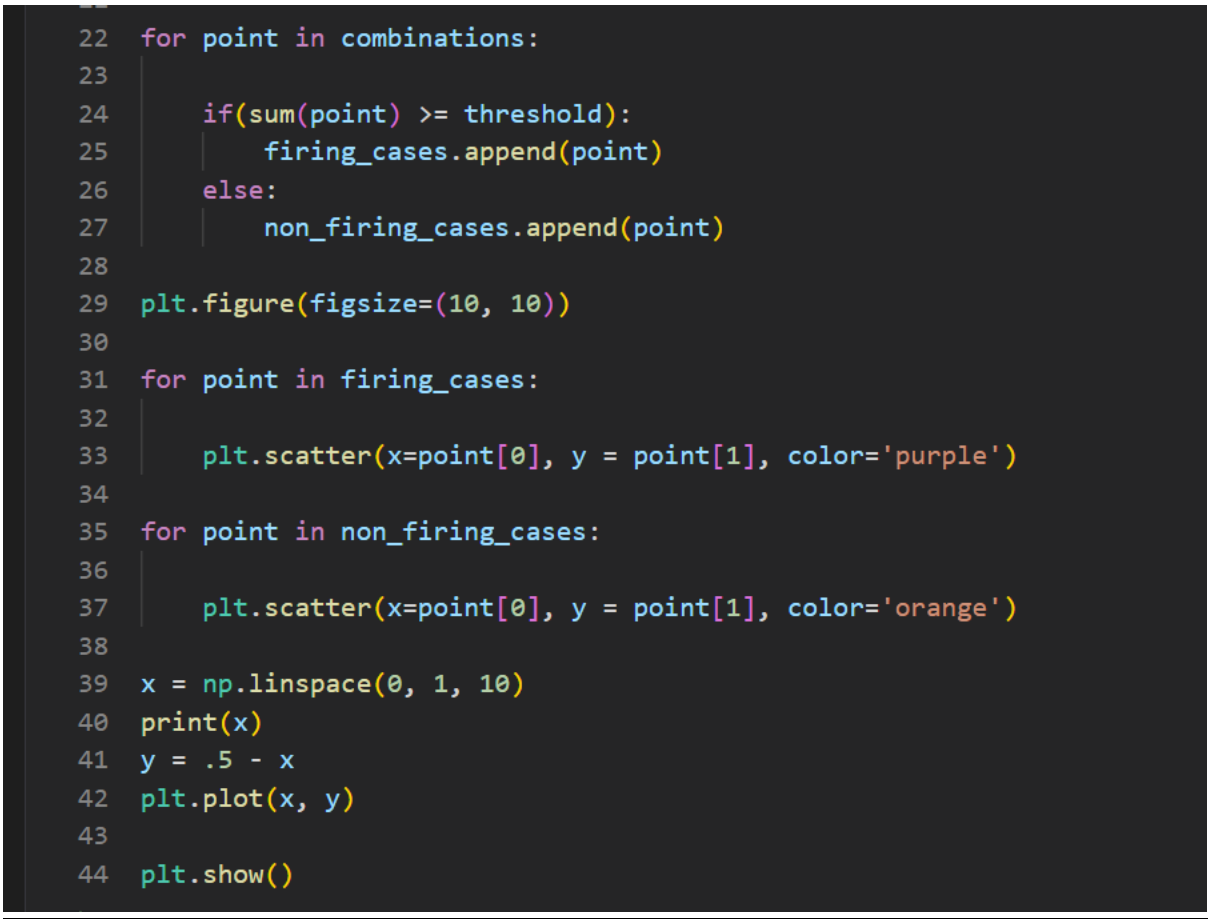
```python
for point in combinations:
    and_value = logical_and(point)
    if(and_value == True and sum(point) < threshold):
        threshold = sum(point)
    print(point, and_value)

for point in combinations:
    if(sum(point) >= threshold):
        firing_cases.append(point)
    else:
        non_firing_cases.append(point)


# Plotting the points
plt.figure(figsize=(10, 10))
for point in firing_cases:
    plt.scatter(x=point[0], y = point[1], color='purple')

for point in non_firing_cases:
    plt.scatter(x=point[0], y = point[1], color='orange')

x = np.linspace(0, 1, 10)
print(x)
y = 1.5 - x
plt.plot(x, y)
plt.show()
```



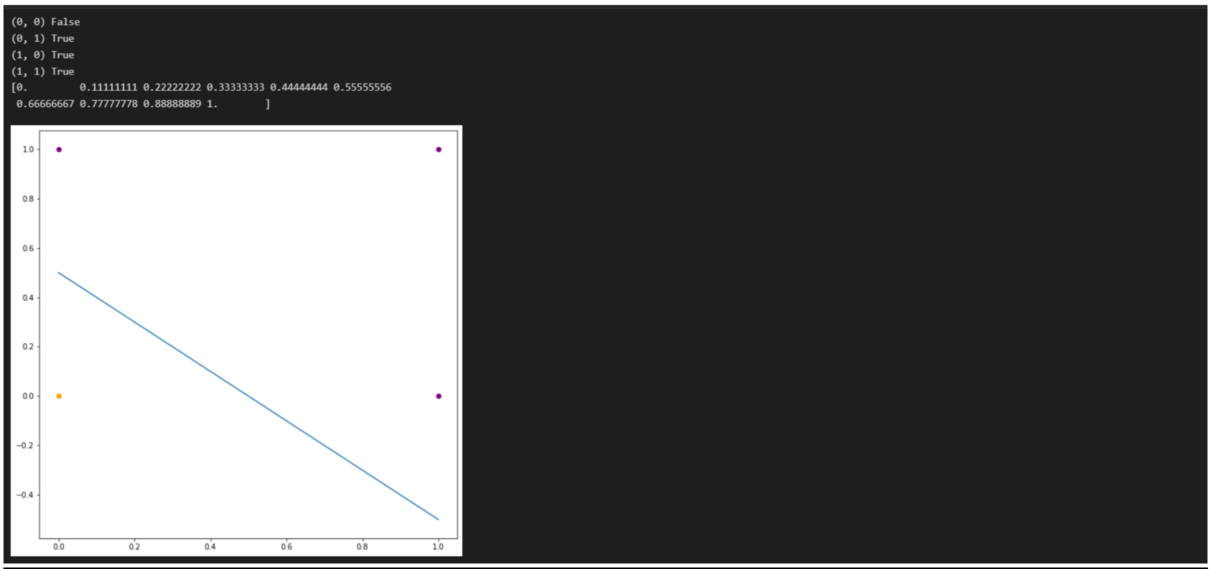
```
[0 0 0]
(0, 0) False
(0, 1) False
(1, 0) False
(1, 1) True
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
```



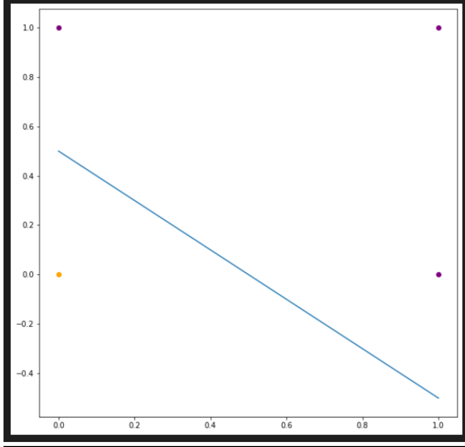
```python
def logical_or(combination):

    or_value = False
    for bit in combination:

        or_value = np.logical_or(bit, or_value)
    return or_value

combinations = get_all_binary_combinations(2)
threshold = float('inf')
bias = 1
firing_cases = []
non_firing_cases = []
for point in combinations:

        or_value = logical_or(point)
        if(or_value == True and sum(point) < threshold):
            threshold = sum(point)

        print(point, or_value)
```



```python
for point in combinations:

    if(sum(point) >= threshold):
        firing_cases.append(point)
    else:
        non_firing_cases.append(point)

plt.figure(figsize=(10, 10))

for point in firing_cases:

    plt.scatter(x=point[0], y = point[1], color='purple')

for point in non_firing_cases:

    plt.scatter(x=point[0], y = point[1], color='orange')

x = np.linspace(0, 1, 10)
print(x)
y = .5 - x
plt.plot(x, y)

plt.show()
```

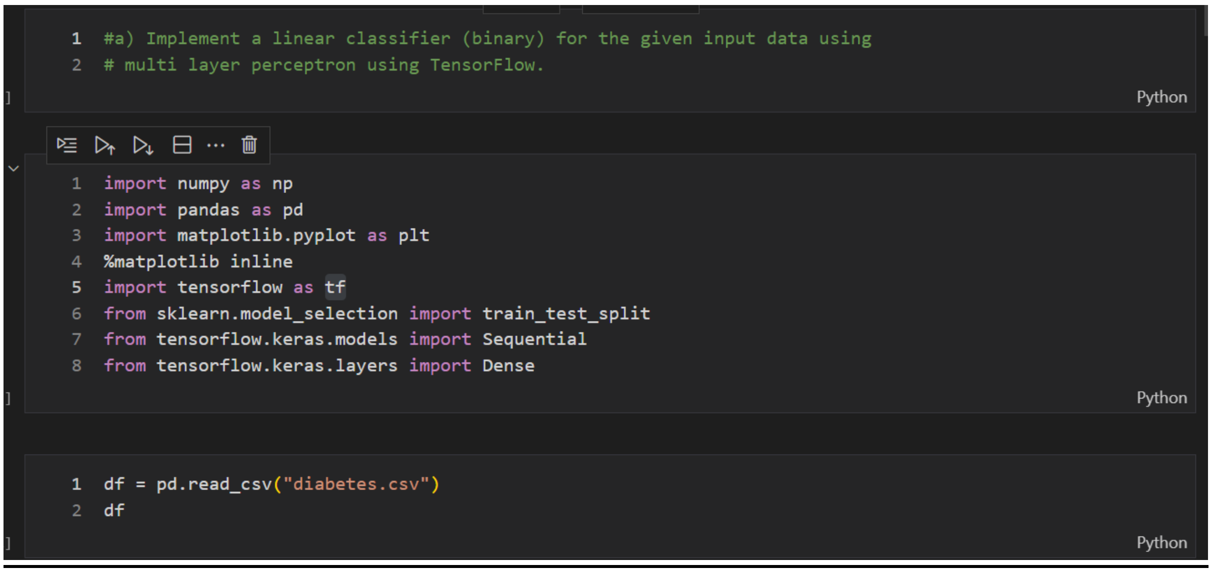


```
(0, 0) False
(0, 1) True
(1, 0) True
(1, 1) True
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
```



LAB-2

A)



```python
#a) Implement a linear classifier (binary) for the given input data using
# multi layer perceptron using TensorFlow.
```

Python

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Python

```python
df = pd.read_csv("diabetes.csv")
df
```

Python

```python
df.head(5)
df.tail(5)
df.count()
df.info()
df.describe()
df.isnull().sum()
#7) dimensions of the dataset
df.shape
```

```python
#8a) Standardise the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('Outcome',axis=1))
scaled_features = scaler.transform(df.drop('Outcome',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

```python
def normalise_dataset(data):
    return (data - data.min()) / (data.max() - data.min())


normalise_dataset(diabetes)
```

```
#9) Split train test into 80-20 ratio
x = df.drop(df[["Outcome"]], axis = 1)
y = df["Outcome"]

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.2)
```

```
model = Sequential()
model.add(Dense(8, input_shape=(8,), activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics='accuracy')
model.fit(x, y, epochs=150, batch_size=20)
accuracy = model.evaluate(x, y)
```

b) John Successfully implemented AND & OR gates using single layer perceptron but was unable to implement XOR Gate. He got to know that it can be implemented by multi layer perceptron using Tensor Flow.

```
import numpy as np
import tensorflow.compat.v1 as tf
import matplotlib.pyplot as plt

num_features = 2
num_iter = 10000
display_step = int(num_iter / 10)
learning_rate = 0.01

num_input = 2           # units in the input layer 28x28 images
num_hidden1 = 2         # units in the first hidden layer
num_output = 1          # units in the output, only one output 0 or 1
```

```
#%% mlp function

def multi_layer_perceptron_xor(x, weights, biases):

    hidden_layer1 = tf.add(tf.matmul(x, weights['w_h1']), biases['b_h1'])
    hidden_layer1 = tf.nn.sigmoid(hidden_layer1)

    out_layer = tf.add(tf.matmul(hidden_layer1, weights['w_out']),
biases['b_out'])

    return out_layer
```

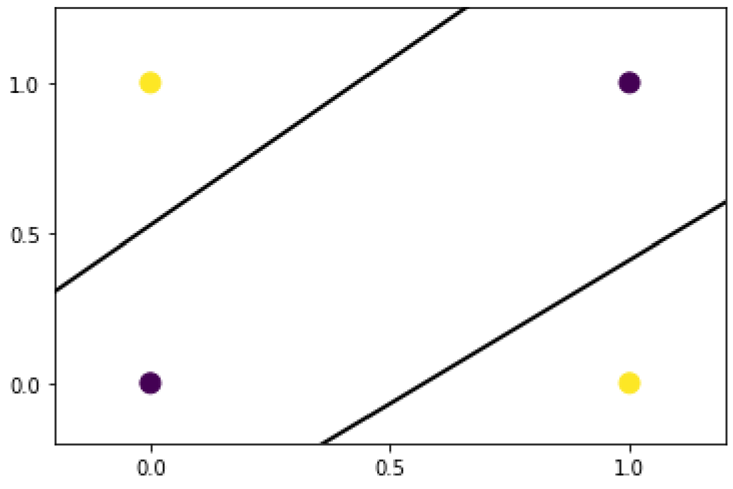
```
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], np.float32)  # 4x2, input
y = np.array([0, 1, 1, 0], np.float32)                      # 4, correct
output, AND operation
y = np.reshape(y, [4,1])                                    # convert to 4x1

# trainum_inputg data and labels
X = tf.placeholder('float', [None, num_input])     # training data
Y = tf.placeholder('float', [None, num_output])    # labels
```

```
# weights and biases
weights = {
    'w_h1' : tf.Variable(tf.random_normal([num_input, num_hidden1])), # w1,
from input layer to hidden layer 1
    'w_out': tf.Variable(tf.random_normal([num_hidden1, num_output])) # w2,
from hidden layer 1 to output layer
}
biases = {
    'b_h1' : tf.Variable(tf.zeros([num_hidden1])),
    'b_out': tf.Variable(tf.zeros([num_output]))
}

model = multi_layer_perceptron_xor(X, weights, biases)
```

```
'''
loss_func =
tf.reduce_sum(tf.nn.sigmoid_cross_entropy_with_logits(logits=model, labels=Y))
optimizer =
tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(loss_f
unc)
#train = tf.train.AdamOptimizer(0.1).minimize(e)
```

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for k in range(num_iter):
    tmp_cost, _ = sess.run([loss_func, optimizer], feed_dict={X: x, Y: y})
    if k % display_step == 0:
        #print('output: ', sess.run(model, feed_dict={X:x}))
        print('loss= ' + "{:.5f}".format(tmp_cost))

# separates the input space
W = np.squeeze(sess.run(weights['w_h1']))   # 2x2
b = np.squeeze(sess.run(biases['b_h1']))    # 2,

sess.close()
```

```
# Now plot the fitted line. We need only two points to plot the line
plot_x = np.array([np.min(x[:, 0] - 0.2), np.max(x[:, 1]+0.2)])
plot_y =  -1 / W[1, 0] * (W[0, 0] * plot_x + b[0])
plot_y = np.reshape(plot_y, [2, -1])
plot_y = np.squeeze(plot_y)

plot_y2 = -1 / W[1, 1] * (W[0, 1] * plot_x + b[1])
plot_y2 = np.reshape(plot_y2, [2, -1])
plot_y2 = np.squeeze(plot_y2)

plt.scatter(x[:, 0], x[:, 1], c=y, s=100, cmap='viridis')
plt.plot(plot_x, plot_y, color='k', linewidth=2)    # line 1
plt.plot(plot_x, plot_y2, color='k', linewidth=2)   # line 2
plt.xlim([-0.2, 1.2]); plt.ylim([-0.2, 1.25]);
#plt.text(0.425, 1.05, 'XOR', fontsize=14)
plt.xticks([0.0, 0.5, 1.0]); plt.yticks([0.0, 0.5, 1.0])
plt.show()
```



LAB-3

```
'''a)  This database contains 76 attributes, but all published experiments 
refer to
using a subset of 14 of them. The "target" field refers to the presence of 
heart disease in the patient.
 It is integer valued 0 = no/less chance of heart attack and 1 = more chance 
of heart attack
 Hint: Use ANN for performing classification( Optimizer: Adam)
https://www.kaggle.com/nareshbhat/health-care-data-set-on-heart-attack-
possibility

b)  Perform ANN regression for the given data from 1c
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

```
df = pd.read_csv('heart.csv')
df.describe()
scaler = StandardScaler()
X = df.drop('target', axis=1)

X = scaler.fit_transform(X)
Y = df['target']
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2)
```

```

model =tf.keras.Sequential([

    tf.keras.layers.Dense(64, activation=tf.nn.relu, 
input_shape=[x_train.shape[1]]),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(1)
])

optimizer = tf.keras.optimizers.RMSprop(0.001)

model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse'])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=100, batch_size=20)

model.evaluate(x_test, y_test)
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
```

```
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)
y_pred
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
conf = confusion_matrix(y_test, y_pred)
print(conf)
accuracy_score(y_test,y_pred)
```

```
#B)Perform ANN regression for the given data

import numpy as np
from keras.layers import Dense, Activation
from keras.models import Sequential
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


# Initialising the ANN
model = Sequential()

# Adding the input layer and the first hidden layer
model.add(Dense(32, activation = 'relu', input_dim = 13))

# Adding the second hidden layer
model.add(Dense(units = 32, activation = 'relu'))

# Adding the third hidden layer
model.add(Dense(units = 32, activation = 'relu'))

# Adding the output layer

model.add(Dense(units = 1))

#model.add(Dense(1))
# Compiling the ANN
```

```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the ANN to the Training set
model.fit(x_train, y_train, batch_size = 10, epochs = 100,
validation_data=(x_test, y_test),shuffle=True)

y_pred = model.predict(x_test)


plt.plot(y_test, color = 'red', label = 'Real data')
plt.plot(y_pred, color = 'blue', label = 'Predicted data')
plt.title('Prediction')
plt.legend()
plt.show()
```

POST LAB-3

```
#Use stochastic gradient descent optimizer for the similar classification
problem above and
#compare the results of both the models.
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf

df = pd.read_csv('heart.csv')
scaler = StandardScaler()
X = df.drop('target', axis=1)

X = scaler.fit_transform(X)
Y = df['target']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2)
```

```
#Using SGD optimizer
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(13,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')

])
```

```
model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=100)


X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

model =tf.keras.Sequential([

    tf.keras.layers.Dense(64, activation=tf.nn.relu,
input_shape=[X_train.shape[1]]),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(1)
])

optimizer = tf.keras.optimizers.RMSprop(0.001)

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy']
)
model.fit(X_train, Y_train, epochs=100, batch_size=20)

print("For SGD optimizer, Training and Testing accuracy are: ")

model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)

print("For ADAM optimizer, Training and Testing accuracy are: ")
model.evaluate(X_train, Y_train)
model.evaluate(X_test, Y_test)
```

LAB-4

```
a)  In this dataset, there are various factors given, which are involved when
a patient is hospitalized.
 On the basis of these factors, predict whether the patient will survive or
not.
 But it has 85 column So, perform Dimensionality reduction using PCA.
  Dataset: https://www.kaggle.com/mitishaagarwal/patient

b)  Normalize the data in the given dataset and perform classification using
ANN.

"""
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
data=pd.read_csv("dataset.csv")
data.drop(["encounter_id" ,"patient_id" ,"hospital_id","Unnamed: 83", ],axis
=1, inplace=True)
```

```
Num=[]
for col in data.columns:
    if (data[col].dtype==int)or (data[col].dtype==float):
        Num.append(col)
print (Num)
```

```
for col in data.columns:
    if col in Num :
        data[col].fillna(data[col].median(), inplace=True)
```

```
categorical=[]
for col in data.columns:
    if  (data[col].dtype==object):
        categorical.append(col)
print (categorical)
```

```
for col in data.columns:
    if col in categorical:
        data[col].fillna(data[col].mode(), inplace=True)
```

```
data["ethnicity"].mode()
```

```
data.drop("hospital_death",axis=1)
y=data["hospital_death"]
```

```
from category_encoders import CountEncoder
CE = CountEncoder(normalize=True, cols=categorical)
data = CE.fit_transform(data)
```

```
from sklearn.preprocessing import RobustScaler
RS = RobustScaler()
scale = RS.fit_transform(data)
data = pd.DataFrame(scale, columns=data.columns)
```

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(data)
pca_data = pca.transform(data)
```

```
component_names = [f"Component {i}" for i in range(1,82 )]
pca_data = pd.DataFrame(pca_data, columns=component_names)
pca_data
```

```
plt.figure(figsize=(20,10))
sns.barplot(x=list(range(1, 82)), y=pca.explained_variance_ratio_,
palette="pastel")

import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, y, test_size=0.2,
random_state=42)
```

```
model = keras.Sequential([
    keras.layers.Dense(input_shape=(81,), units=1, activation='sigmoid'),
    keras.layers.Dense(264, activation=tf.nn.tanh),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(128, activation=tf.nn.tanh),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation=tf.nn.sigmoid),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(1)
    ])
```

```
earlyStopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=3, callbacks=[earlyStopping])
```

```
model.evaluate(x_test, y_test)

from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)
cm = confusion_matrix(y_test, y_pred)
cm
```

LAB-5

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
```

```
data = keras.datasets.mnist
(X_train, Y_train) , (x_test, y_test) = data.load_data()
plt.imshow(X_train[20])
plt.show()
```

```
X_train = X_train / 255
x_test = x_test / 255
X_valid, X_train = X_train[:2000], X_train[2000:4000]
Y_valid, Y_train = Y_train[:2000], Y_train[2000:4000]
```

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

```
import pydot
import graphviz
keras.utils.plot_model(model)
```

```
weights, biases = model.layers[1].get_weights()
weights.shape, biases.shape
```

```
model.compile(loss = "sparse_categorical_crossentropy",
              optimizer = "sgd",
              metrics = ["accuracy"])
model_history = model.fit(X_train, Y_train, epochs=50,
validation_data=(X_valid, Y_valid))
pd.DataFrame(model_history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```

LAB-6

```
'''
a)  Build a prediction model that will perform the following:
i.  Classify if a customer is going to churn or not
ii. Preferably and based on model performance,
    choose a model to improve the accuracy by adding some dropout layers and
regularization.

https://www.kaggle.com/kmalit/bank-customer-churn-prediction

'''
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('sonar_dataset.csv',header=None)
df.head()
```

```
x = df.drop(60,axis='columns')
y = df[60]
x.head()
y = pd.get_dummies(y,drop_first=True)
y.sample(5) # R = 1, M = 0
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.25,random_state=1)
```

```
import tensorflow as tf
from tensorflow import keras
model = keras.Sequential([
    keras.layers.Dense(60,input_shape=(60,),activation='relu'),
    keras.layers.Dense(30,activation='relu'),
    keras.layers.Dense(15,activation='relu'),
    keras.layers.Dense(1,activation='sigmoid')
])

model.compile(optimizer='adam',loss =
'binary_crossentropy',metrics=['accuracy'])
```

```python
model.fit(x_train,y_train,epochs=100,batch_size=8)

model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
```

```python
y_pred = model.predict(x_test).reshape(-1)
y_pred[:10]
#rounding off the values to 0 or 1
y_pred = np.round(y_pred)
```

```python
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test,y_pred)
cm
```

```python
print(classification_report(y_test,y_pred))
```

```python
#Drop out layer
model = keras.Sequential([
    keras.layers.Dense(60,input_shape=(60,),activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(30,activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(15,activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1,activation='sigmoid')
    ])
```

```python
#Compile model
model.compile(optimizer='adam',loss =
'binary_crossentropy',metrics=['accuracy'])
model.fit(x_train,y_train,epochs=100,batch_size=8)
```

```python
model.evaluate(x_train, y_train)
model.evaluate(x_test, y_test)
```

```python
y_pred = model.predict(x_test).reshape(-1)
```

```
y_pred[:10]
#rounding off the values to 0 or 1
y_pred = np.round(y_pred).astype(int)
```

```

from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test,y_pred)
cm
print(classification_report(y_test,y_pred))
```

LAB-7

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.utils import load_img, img_to_array
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

```
# Load the data

train_data_gen = ImageDataGenerator(rescale=1./255, shear_range=0.2,
zoom_range=0.2, horizontal_flip=True)
test_data_gen = ImageDataGenerator(rescale=1./255)

train = train_data_gen.flow_from_directory('train', target_size=(64, 64),
batch_size=32, class_mode='binary')
test = test_data_gen.flow_from_directory('test', target_size=(64, 64),
batch_size=32, class_mode='binary')
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(64,
64, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

results = model.fit(train, epochs=20, validation_data=test, verbose=1)
```

```
# Testing
print(train.class_indices)
#Then load the image to be classified.
img = load_img('test/Normal/0101.jpeg', target_size=(64, 64))
imgplot = plt.imshow(img)
test_image = load_img('test/Normal/0101.jpeg', target_size=(64, 64))
test_image = img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
result = model.predict(test_image)

if result[0][0] == 1:
    prediction ='Normal'
else:
    prediction ='Covid-19'

plt.title('Prediction is  '+ prediction )
plt.show()
```

POST:

```
'''
Use 3*3 filters and perform convolution operation for the matrix. The Given
3*3 matrix is:
A=[[1,1,1,0,0,0], [1,1,0,1,0,0], [1,0,1,0,1,0],[ [0,1,1,0,0,1], [0,0,1,0,1,0],
[1,0,0,0,0,0]]
'''
```

```
#Applying Convolution Operation using random 3*3 filter

    # A is the input matrix and B is the filter matrix
    # A is a 2D matrix and B is a 2D matrix
    # A is of size m*n and B is of size p*q
    # m and n are the number of rows and columns of A
    # p and q are the number of rows and columns of B
    # The output matrix C is of size (m-p+1)*(n-q+1)
    # The output matrix C is the result of convolution operation


import numpy as np

def convolution(A):
    m,n = A.shape
    B=np.random.randint(-1,2,size=(3,3))
    print()
    print("The Filter is:")
    print(B)
```

```python
    p,q = B.shape
    C=np.zeros((m-p+1,n-q+1))
    for i in range(m-p+1):
        for j in range(n-q+1):
            C[i][j] = np.sum(A[i:i+p, j:j+q]*B)
    return C

A = np.array([[1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0],
                 [1, 0, 1, 0, 1, 0], [0, 1, 1, 0, 0, 1],
                 [0, 0, 1, 0, 1, 0], [1, 0, 0, 0, 0, 0]])
print("The Input Matrix is:")
print(A)

C = convolution(A)
print()
print("The Output Matrix is:")
print(C)
```

```
The Input Matrix is:

[[1 1 1 0 0 0]

 [1 1 0 1 0 0]

[1 0 1 0 1 0]

 [0 1 1 0 0 1]

[0 0 1 0 1 0]

 [1 0 0 0 0 0]]

 The Filter is: [[1 1 1] [1 1 1] [1 1 1]]

The Output Matrix is: [[7. 5. 4. 2.] [6. 5. 4. 3.] [5. 4. 5. 3.] [4.
3. 3. 2.]]
```

LAB-8

```
import os
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
from keras import layers, models, optimizers
from keras.preprocessing.image import ImageDataGenerator
```

```
train_dir = r'cats_and_dogs_small\train'
test_dir = r'cats_and_dogs_small\test'
```

```
train_data_gen = ImageDataGenerator(rescale=1./255)
test_data_gen = ImageDataGenerator(rescale=1./255)

train_gen = train_data_gen.flow_from_directory(
    train_dir,
    target_size = (150, 150),
    batch_size = 20,
    class_mode = 'binary')

test_gen = test_data_gen.flow_from_directory(
    test_dir,
    target_size = (150, 150),
    batch_size = 20,
    class_mode = 'binary')
```

```
cnn_model = models.Sequential()

cnn_model.add(layers.Conv2D(32, (3, 3), activation = 'relu', input_shape =
(150, 150, 3)))
cnn_model.add(layers.MaxPooling2D(2, 2))

cnn_model.add(layers.Conv2D(64, (3, 3), activation = 'relu'))
cnn_model.add(layers.MaxPooling2D(2, 2))

cnn_model.add(layers.Conv2D(128, (3, 3), activation = 'relu'))
cnn_model.add(layers.MaxPooling2D(2, 2))

cnn_model.add(layers.Conv2D(128, (3, 3), activation = 'relu'))
```

```python
cnn_model.add(layers.MaxPooling2D(2, 2))

cnn_model.add(layers.Flatten())

cnn_model.add(layers.Dense(512, activation = 'relu'))

cnn_model.add(layers.Dense(1, activation = 'sigmoid'))
cnn_model.summary()
```

```python
cnn_model.compile(loss = 'binary_crossentropy',
              optimizer = optimizers.RMSprop(lr = 1e-4),
              metrics = ['accuracy'])
```

```python
model_history = cnn_model.fit(train_gen,
                                        steps_per_epoch = 50,
                                        epochs = 5)
```

```python
pd.DataFrame(model_history.history).plot(figsize = (8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
cnn_model.evaluate(test_gen, steps = 50)
```

```python
img_path = "dog.jpg"
img_path1 = "cat.jpg"
img0 = image.load_img(img_path, target_size = (150, 150))
img1 = image.load_img(img_path1, target_size = (150, 150))
plt.imshow(img0)
plt.imshow(img1)
```

```python
img0 = image.img_to_array(img0)
img1 = image.img_to_array(img1)
x0 = np.expand_dims(img0, axis = 0)
x1 = np.expand_dims(img1, axis = 0)
img0 = np.vstack([x0])
img1 = np.vstack([x1])

result0 = cnn_model.predict(img0)
result1 = cnn_model.predict(img1)

# when the value is near to 1 it is classified as dog and
# when the result is near to 0 it is classified as a car
print(result0, result1)
```

LAB-9

```
"""
The stock price of the present day can be predicted by the stock prices
obtained for past 50 days.
Using Simple RNN, train the model with “Google_stock_price_train.csv” dataset,
predict the stock prices for the dates given in “Google_stock_price_train.csv”
dataset and
visualize the actual, predicted prices using matplotlib.
"""
```

```
# Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import keras
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout
```

```
# Importing the training set

dataset_train = pd.read_csv('Google_Stock_Price_Train.csv')
dataset_train.head()
```

```
training_set = dataset_train.iloc[:, 1:2].values
training_set.shape
```

```
scaler = MinMaxScaler(feature_range = (0, 1))
scaled_training_set = scaler.fit_transform(training_set)
X_train = []
y_train = []

for i in range(50, 1258):
    X_train.append(scaled_training_set[i-50:i, 0])
    y_train.append(scaled_training_set[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_train.shape
regressor = Sequential()
```

```
regressor.add(SimpleRNN(units = 50,activation='relu', return_sequences = True,
input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 50, activation='relu', return_sequences =
True))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 50, activation='tanh', return_sequences =
True))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))
regressor.compile(optimizer='adam', loss='mse')
regressor.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
actual_stock_price = dataset_test.iloc[:, 1:2].values
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis
= 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 50:].values

inputs = inputs.reshape(-1,1)
inputs = scaler.transform(inputs)

X_test = []
for i in range(50, 60):
    X_test.append(inputs[i-50:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)
plt.plot(actual_stock_price, color='red', label = 'Real Google Stock Price')
plt.plot(predicted_stock_price, color='blue', label = 'Predicted Google Stock
Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Days')
plt.ylabel('Google_Stock_Price')
plt.legend()
plt.show()
```

LAB-10

```
# Recurrent Neural Network

# Part 1 - Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```

# Importing the training set
dataset_train = pd.read_csv('covid_india_train.csv')
training_set = dataset_train.iloc[:,1:2].values
```

```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

```
# Creating a data structure with 20 timesteps and t+1 output
X_train = []
y_train = []
for i in range(20, 125):
    X_train.append(training_set_scaled[i-20:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
# Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
# Initialising the RNN
regressor = Sequential()

# Adding the input layer and the LSTM layer
regressor.add(LSTM(units = 3,activation='sigmoid',input_shape = (None, 1)))

# Adding the output layer
regressor.add(Dense(units = 1))
```

```python
# Compiling the RNN
regressor.compile(optimizer = 'rmsprop', loss = 'mean_squared_error')
```

```python
# Fitting the RNN to the Training set
regressor.fit(X_train, y_train, epochs = 1000, batch_size = 32)
dataset_test = pd.read_csv('covid_india_test.csv')
test_set = dataset_test.iloc[:,1:2].values
real_covid_cases = np.concatenate((training_set[0:125], test_set), axis = 0)
```

```python
# Getting the predicted number of cases
scaled_real_covid_cases= sc.fit_transform(real_covid_cases)
inputs = []
for i in range(126, 136):
    inputs.append(scaled_real_covid_cases[i-20:i, 0])
inputs = np.array(inputs)
inputs = np.reshape(inputs, (inputs.shape[0], inputs.shape[1], 1))
```

```python
predicted_covid_cases = regressor.predict(inputs)

predicted_covid_cases = sc.inverse_transform(predicted_covid_cases)
predicted_covid_cases=predicted_covid_cases.astype('int64')
```

```python
# Visualising the results
plt.plot(test_set, color = 'red', label = 'Real Covid Cases')
plt.plot(predicted_covid_cases, color = 'blue', label = 'Predicted Covid
Cases')
plt.title('Covid Prediction')
plt.xlabel('Time')
plt.ylabel('Predition of covid cases')
plt.legend()
plt.show()
```

```python
import math
from sklearn.metrics import mean_squared_error
rmse=math.sqrt(mean_squared_error(test_set,predicted_covid_cases))
#predict the next 2 days case
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
real_covid_cases_scaled = sc.fit_transform(real_covid_cases)
input_next_pred=real_covid_cases_scaled.reshape(-1,1)
input_next_pred = np.asarray(input_next_pred).astype(np.float32)
```

```
list_input=list(input_next_pred)
from numpy import array
lst_output=[]
n_steps=135
i=0
while(i<2):

    if(len(list_input)>135):
        #print(temp_input)
        input_next_pred=np.array(list_input[1:])
        #print("{} day input {}".format(i,x_input))
        input_next_pred=input_next_pred.reshape(1,-1)
        input_next_pred = input_next_pred.reshape((1, n_steps, 1))
        input_next_pred  = input_next_pred.astype(np.float32)
        #print(x_input)
        y_pred = regressor.predict(input_next_pred, verbose=0)
        y_pred = sc.inverse_transform(y_pred)
        y_pred=y_pred.astype('int64')

        #print("{} day output {}".format(i,yhat))
        list_input.extend(y_pred[0].tolist())
        list_input=list_input[1:]
        #print(temp_input)
        lst_output.extend(y_pred.tolist())
        i=i+1
    else:
        input_next_pred= input_next_pred.reshape((1, n_steps,1))
        y_pred = regressor.predict(input_next_pred, verbose=0)
        y_pred = sc.inverse_transform(y_pred)
        y_pred=y_pred.astype('int64')
        print(y_pred[0])
        list_input.extend(y_pred[0].tolist())
        print(len(list_input))
        lst_output.extend(y_pred.tolist())
        i=i+1
```

```
print("Total covid cases in India on 13 June will be apporx: ",lst_output[0])
print("Total covid cases in India on 14 June will be apporx: ",lst_output[1])

day_new=np.arange(1,136)
day_pred=np.arange(136,138)
day_new=day_new.reshape(-1,1)
```

```
plt.plot(day_new,real_covid_cases,color = 'blue', label = 'Real Covid Cases')
plt.plot(day_pred,lst_output,color = 'green', label = 'Next 2 days cases
prediction')
plt.title('Next 2 days covid prediction')
plt.xlabel('Time')
plt.ylabel('Predition of new covid cases')
plt.legend()
plt.show()
```

POST:

```
"""
Predict the weather for the dates given in the test dataset and visualize the
actual,
predict data using matplotlib.
Use 20 day’s time stamp.
"""
```

```
import numpy as np
import matplotlib.pyplot as plt
from pandas import read_csv
import math
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,SimpleRNN
from tensorflow.keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing
```

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
# fix random seed for reproducibility
tf.random.set_seed(7)
# load the dataset
dataframe = read_csv('weatherHistory.csv')
label_encoder = preprocessing.LabelEncoder()
dataframe['Formatted Date']=pd.to_datetime(dataframe['Formatted Date'])
```

```
dataframe['Summary']=label_encoder.fit_transform(dataframe['Summary'])
dataframe['Precip Type']=label_encoder.fit_transform(dataframe['Precip
Type'].astype(str))
dataframe['Daily Summary']=label_encoder.fit_transform(dataframe['Daily
Summary'])
```

```
dataframe['Formatted Date']=pd.to_datetime(dataframe['Formatted
Date'],utc=True)
dataframe.info()
```

```
x_train=dataframe.drop(['Daily Summary'],axis=1)[:100]
y_train=np.array(dataframe['Daily Summary'])[:70]
x_test=dataframe.drop(['Daily Summary'],axis=1)[100:200]
y_test=np.array(dataframe['Daily Summary'])[70:100]
Row_list=[]
for index, rows in x_train.iterrows():
    # Create list for the current row
    my_list =[rows.Summary, rows['Precip Type'],rows['Temperature (C)']]
    Row_list.append([my_list])
Row_list1=[]
i=0
for index, rows in x_test.iterrows():
    # Create list for the current row
    my_list =[rows.Summary, rows['Precip Type'],rows['Temperature (C)']]
    Row_list[i].append(my_list)
    i+=1
x_train=np.array(Row_list)[:70]
x_test=np.array(Row_list)[70:100]
```

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(2,3)))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
model.fit(x_train, y_train, epochs=50, batch_size=2, verbose=2)
# make predictions
trainPredict = model.predict(x_train)
testPredict = model.predict(x_test)
```

```
# calculate root mean squared error
trainScore = np.sqrt(mean_squared_error(y_train, trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = np.sqrt(mean_squared_error(y_test, testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

```
# shift train predictions for plotting
trainPredictPlot = np.empty_like(dataframe)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[1:len(trainPredict)+1, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(dataframe)
testPredictPlot[:, :] = np.nan
testPredictPlot[1:len(testPredict)+1, :] = testPredict
```

```
# plot baseline and predictions
plt.plot(testPredictPlot,color='red',label='Predicted')
plt.plot(y_test,color='blue',label='Actual')
plt.xlabel('Time')
plt.ylabel('Weather summary')
scale_factor =1
#plt.legend()
plt.show()
```

LAB-11

```
#Auto Encoders

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import tensorflow as tf
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
plt.imshow(x_train[9], cmap='gray')
x_train[9].shape
x_train = x_train/255
x_test = x_test/255
```

```
encoder_input = tf.keras.layers.Input(shape=(28,28,1),name='img')
x = tf.keras.layers.Flatten()(encoder_input)
encoder_output = tf.keras.layers.Dense(32, activation='relu')(x)

encoder = tf.keras.Model(encoder_input, encoder_output, name='encoder')

decoder_input = tf.keras.layers.Dense(784, activation='relu')(encoder_output)
decoder_output = tf.keras.layers.Reshape((28,28,1))(decoder_input)

opt = tf.keras.optimizers.Adam(lr=0.001)

autoencoder = tf.keras.Model(encoder_input, decoder_output,
name='autoencoder')
autoencoder.summary()
```

```
autoencoder.compile(optimizer=opt, loss='mse')
autoencoder.fit(x_train, x_train, epochs=3, batch_size=32,
validation_split=0.1)
example = encoder.predict([x_test[9].reshape(1,28,28,1)])[0]
print(example)
```

```
plt.imshow(example.reshape(8,4), cmap='gray')
plt.imshow(x_test[9], cmap='gray')
ae_out = autoencoder.predict([x_test[9].reshape(1,28,28,1)])[0]
plt.imshow(ae_out.reshape(28,28), cmap='gray')
```

LAB-12

```
from keras.datasets import mnist
import numpy as np
import keras
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.callbacks import TensorBoard
import matplotlib.pyplot as plt
```

```
#making the noisy images using the mnist dataset

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.


#normalizing the images
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

```
#The convolution autoencoder model
input_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

encoded = MaxPooling2D((2, 2), padding='same')(x)
```

```
#at this point representation is (7, 7, 32)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
```

```
x = UpSampling2D((2, 2))(x)

decoded =Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)
```

```
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
#optimizer='adadelta'

autoencoder.fit(x_train_noisy, x_train,
                epochs=2,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test),
                callbacks=[TensorBoard(log_dir='/tmp/autoencoder',
histogram_freq=0, write_graph=False)])

decoded_imgs = autoencoder.predict(x_test_noisy)
```

```
n = 10
plt.figure(figsize=(20, 4))

for i in range(1,n+1):
    ax = plt.subplot(2, n, i)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    ax =plt.subplot(2,n,i+n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```