from sklearn import datasets

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

from sklearn.metrics import accuracy\_score

from sklearn import preprocessing,svm

from sklearn.decomposition import PCA

# import some data to play with

digits = datasets.load\_digits()

X\_digits = digits.data

y\_digits = digits.target

print(X\_digits.shape)

print(y\_digits.shape)

# 1 Split the Digits Dataset (load\_digits) into training and testing sets with ratio 9:1

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X\_digits, y\_digits,test\_size=0.1)

# 2 Standardize the data for each dimension

scaler = preprocessing.StandardScaler().fit(Xtrain)

Xtrain\_new = scaler.transform(Xtrain)

Xtest\_new = scaler.transform(Xtest)

# 3 Reduce the dimension to 32 using PCA

pca = PCA(32)

pca.fit(Xtrain\_new)

Xtrain\_new = pca.transform(Xtrain\_new)

Xtest\_new = pca.transform(Xtest\_new)

print(Xtrain\_new.shape)

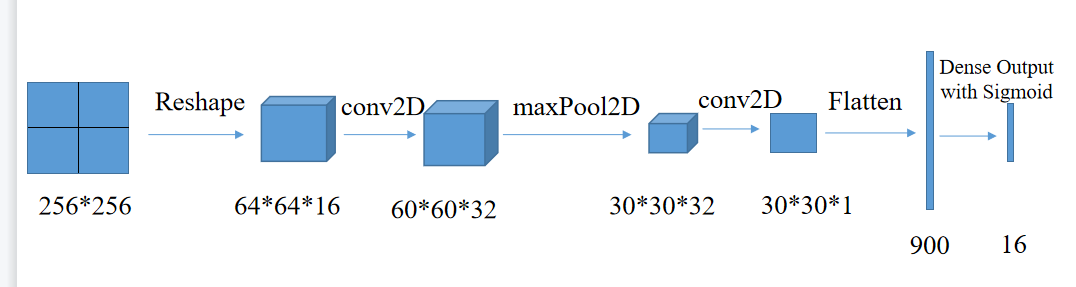
# 4 Train a SVM classifier

model = svm.SVC()

model.fit(Xtrain\_new, ytrain)

y\_pred = model.predict(Xtest\_new)

print(accuracy\_score(y\_pred,ytest))



from keras.models import Sequential

from keras.layers import Dense,Conv2D,MaxPool2D,Input,Reshape,Flatten

import keras

import numpy as np

layers = [Input(shape=(256,256,1)),

      Reshape((64,64,-1)),

      Conv2D(filters=32,kernel\_size=5),

      MaxPool2D(2),

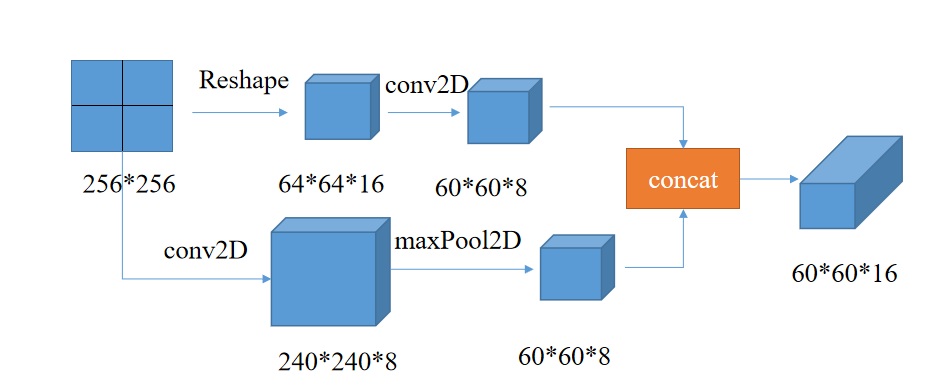
      Conv2D(filters=1,kernel\_size=1),

      Flatten(),

      Dense(16,activation='sigmoid')]

model = Sequential(layers)

model.summary()



def my\_model():

  # define layers

    input\_tensor = Input(shape=(256,256,1))

    reshape\_layer = Reshape((64,64,-1))

    conv2d\_1 = Conv2D(filters=8,kernel\_size=5)

    conv2d\_2 = Conv2D(filters=8,kernel\_size=17)

    max\_pool\_layer = MaxPool2D(4)

    input\_tensor\_reshape = reshape\_layer(input\_tensor)

    conv1\_out = conv2d\_1(input\_tensor\_reshape)

    conv2\_out = conv2d\_2(input\_tensor)

    max\_pool\_out = max\_pool\_layer(conv2\_out)

    cat\_out = concatenate([conv1\_out,max\_pool\_out],axis=-1)

    model = Model(inputs=input\_tensor, outputs=cat\_out)

    return model

my\_model = my\_model()

my\_model.summary()

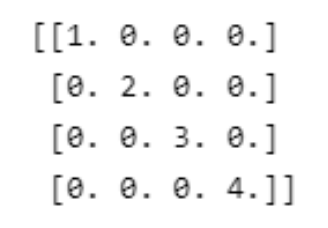
import numpy as np

a = [[1, 2], [3, 4]]

print(a)

a\_pad = np.pad(a, ((1, 9), (3, 3)), 'constant')

print(a\_pad)



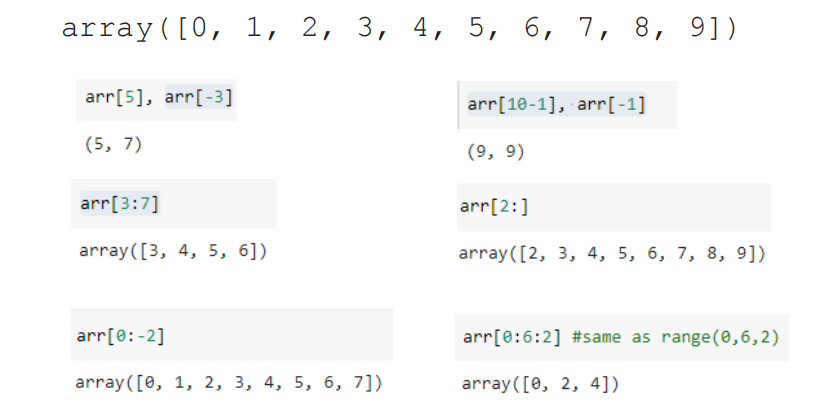
import numpy as np

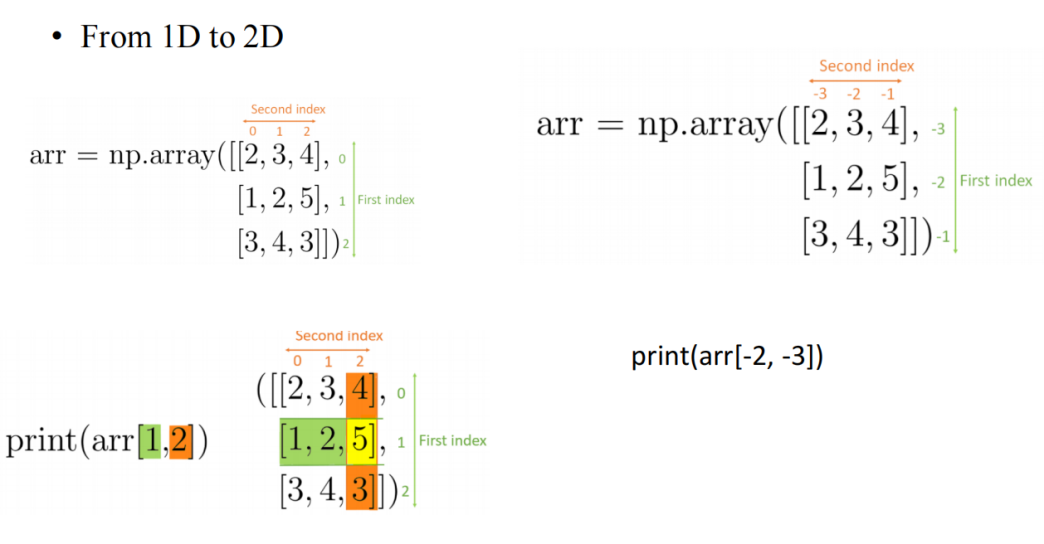
a=np.zeros([4,4])

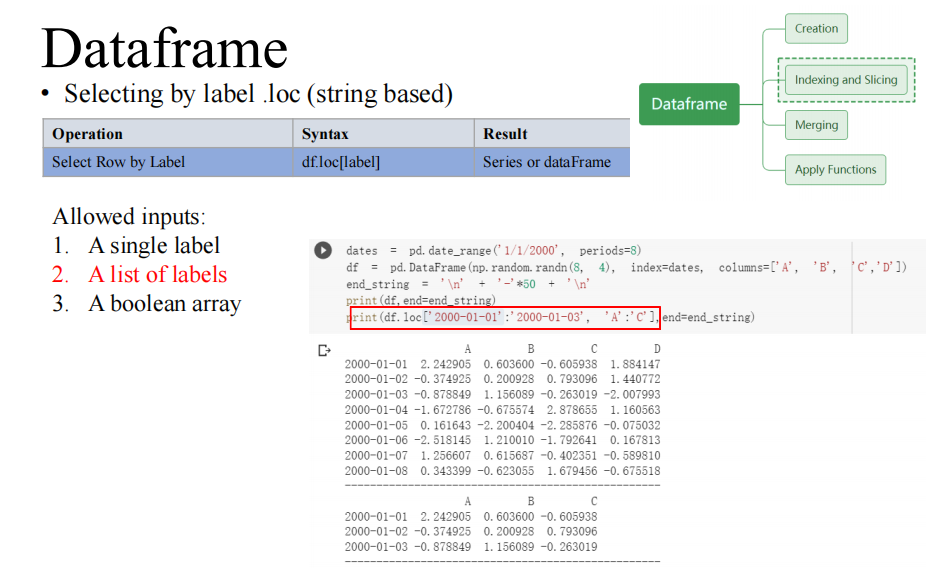
for i in range(4):

    a[i,i]=i+1

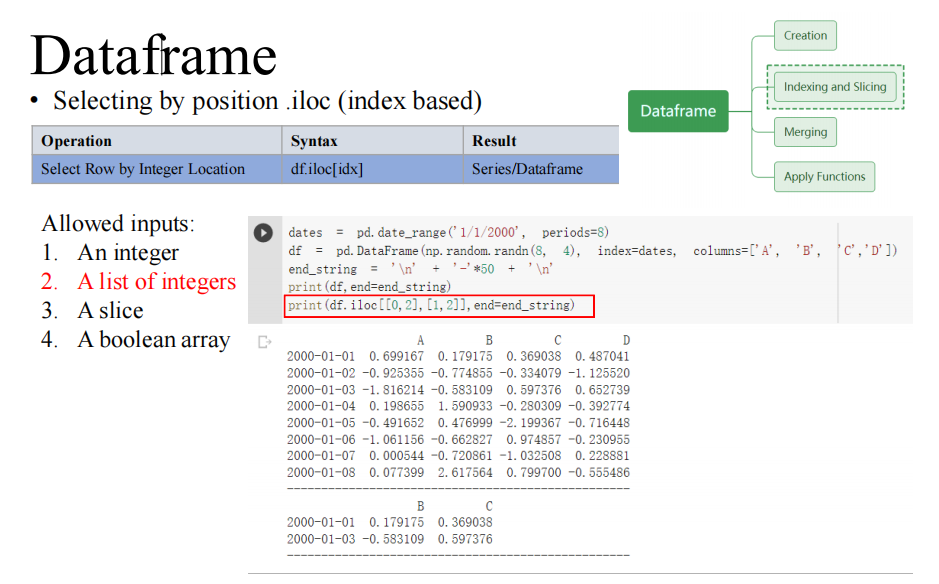
print(a)



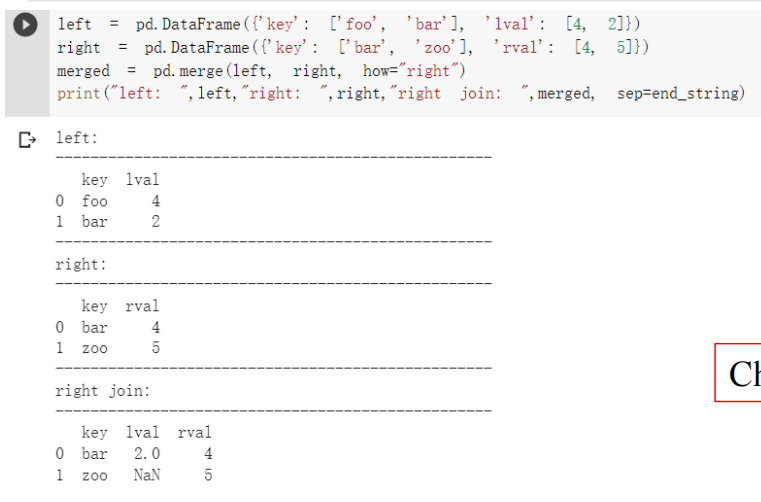




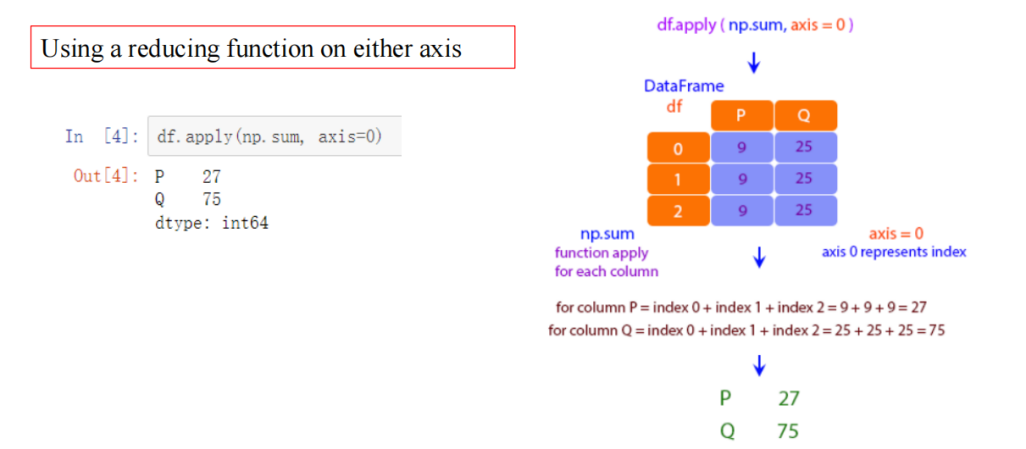
df.loc[行，列]



df.iloc[rowIndex, columnIndex]



pd.merge(left,right,how=’right’), 自动找可以连起来的键



df.apply(function, axis=0列，axis=1行),可以通过loc.apply配合实现我们的逻辑

aRowColumnMax = df.loc[‘a’].apply(thisRowColumnMax)

Data visualization

