**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN 1**

-----o0o----

**BÁO CÁO TIỂU LUẬN CUỐI KỲ MÔN**

**PHÁT TRIỂN CÁC HỆ THỐNG THÔNG MINH**

|  |  |
| --- | --- |
| **Giảng viên:** | PGS.TS Trần Đình Quế |
| **Sinh viên:** | Phạm Hồng Đại |
| **Mã sinh viên:** | B19DCCN162 |
| **Lớp:** | INT14151 nhóm 02 |
| **SĐT:** | 0388560668 |

**Hà Nội, tháng 1 năm 2023**

**MỤC LỤC**

[PHẦN I: MACHINELEARNING CƠ BẢN 3](#_Toc123844463)

[CHƯƠNG 1: XỬ LÝ DỮ LIỆU 3](#_Toc123844464)

[CHƯƠNG 2. HỌC CÓ GIÁM SÁT 15](#_Toc123844471)

[1. Linear Regression 15](#_Toc123844472)

[2. Classification Using Logistic Regression 19](#_Toc123844473)

[3. Classification Using Support Vector Machines 24](#_Toc123844474)

[4. Classification Using K-Nearest Neighbors(KNN) 26](#_Toc123844475)

[CHƯƠNG 3. HỌC KHÔNG GIÁM SÁT 29](#_Toc123844476)

[1. Unsupervised Learning Using K-Means 30](#_Toc123844477)

[CHƯƠNG 4. CASE STYDY VÀ DEPLOY 33](#_Toc123844478)

[PHẦN II: DEEPLEARNING 38](#_Toc123844479)

[CHƯƠNG 5: BIỂU DIỄN DỮ LIỆU VÀ TENSORFLOW 38](#_Toc123844480)

[CHƯƠNG 6: HỌC SÂU VÀ KERAS 45](#_Toc123844489)

[PHẦN III: Bài tập áp dụng 54](#_Toc123844490)

[Bài 1: Dự đoán bệnh tiểu đường. 54](#_Toc123844491)

[Bài 2: Dự đoán điểm thi. 60](#_Toc123844492)

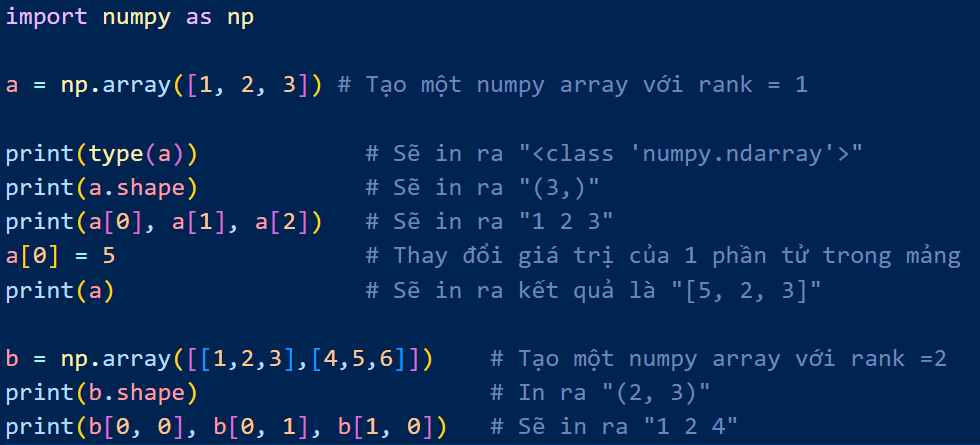
# **PHẦN I: MACHINE LEARNING CƠ BẢN**

## **CHƯƠNG 1: XỬ LÝ DỮ LIỆU**

1. **Numpy**

* **Giới thiệu:** 
  + Numpy là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.
* **Arrays**
  + Một mảng numpy là một lưới các giá trị, và tất cả các giá trị có dùng kiểu giá trị, và được lập chỉ mục bởi một số nguyên không âm, số chiều được gọi là **rank** của mảng Numpy, và **shape** là một **tuple** các số nguyên đưa ra kích thước của mảng theo mỗi chiều.

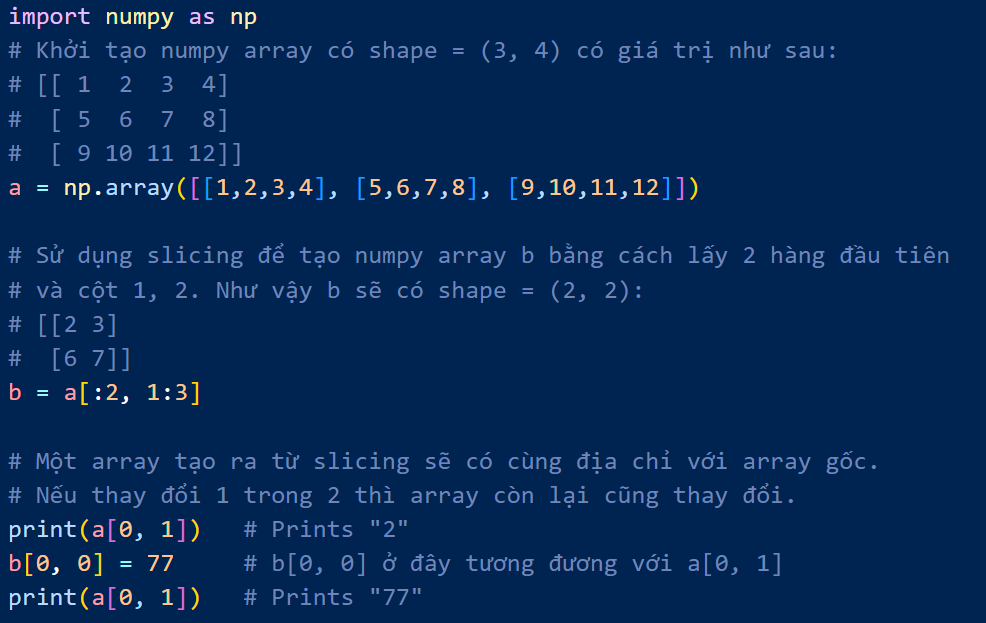
Có thể khởi tạo numpy arrays từ nested Python lists, và dùng dấu ngoặc vuông để truy cập từng phần tử



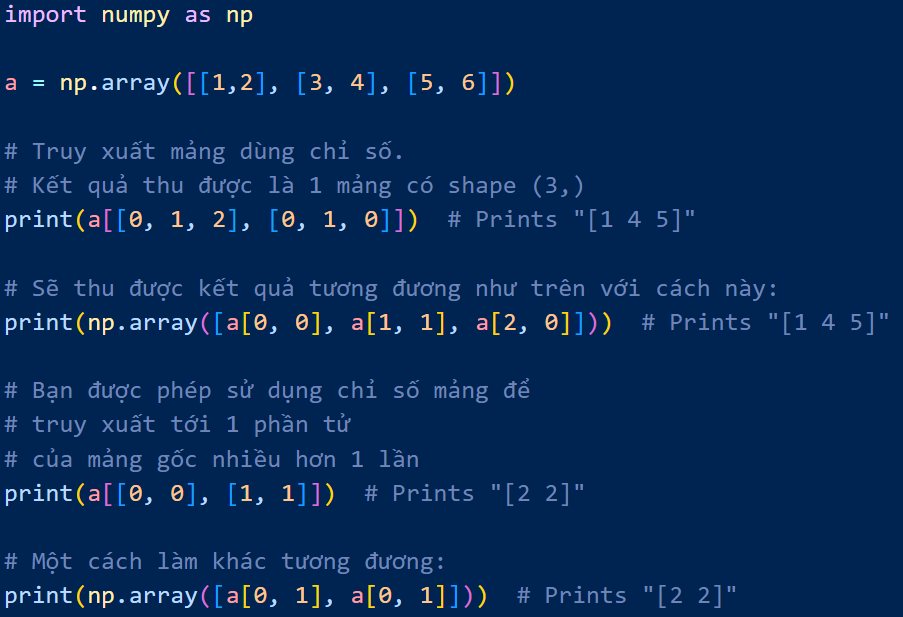
* **Array indexing**

Numpy cung cấp một số cách để truy xuất phần tử trong mảng

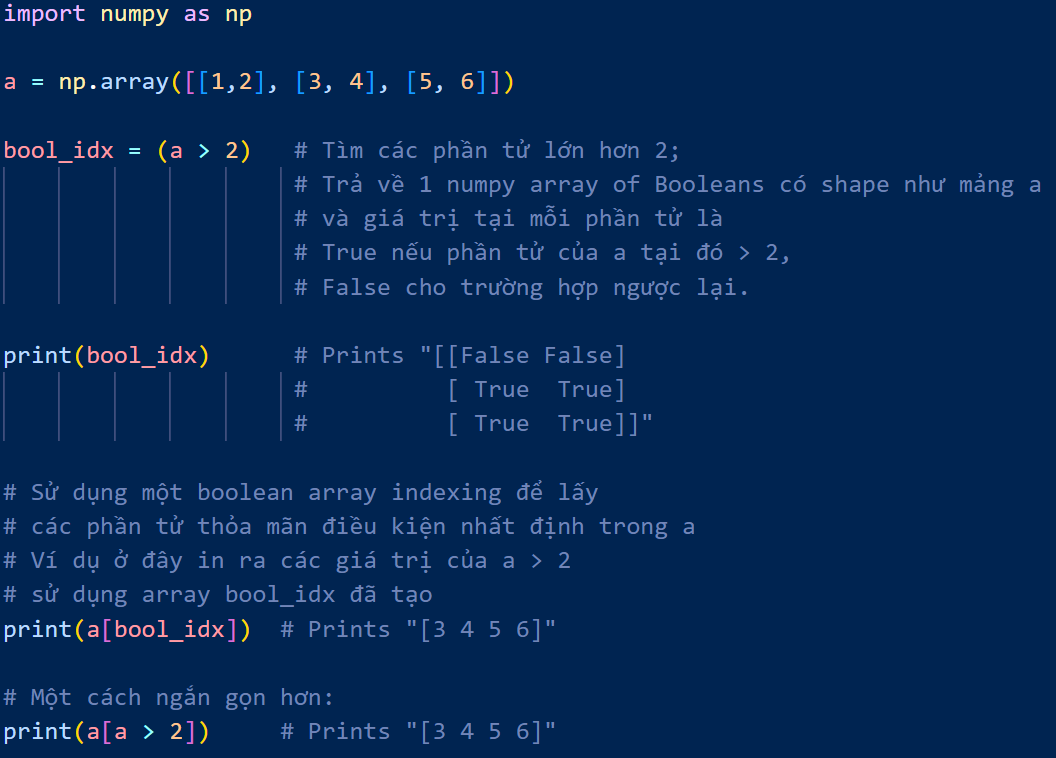
* + **Slicing**: Tương tự như list trong python, numpy arrays cũng có thể được cắt.



* + **Integer array indexing**: Khi truy xuất mảng dùng slicing, kết quả trả về sẽ là mảng con của mảng ban đầu, tuy nhiên sử dụng chỉ số mảng cho phép xây dựng mảng tùy ý từ một mảng khác

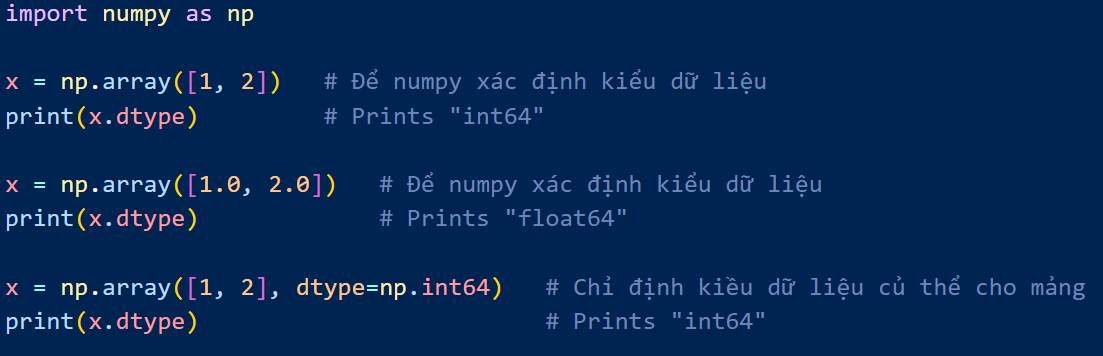


* + **Boolean array indexing**: Cho phép chọn ra các phần tử tùy ý của một mảng, thường được sử dụng để chọn ra các phần tử thỏa mãn điều kiện nào đó



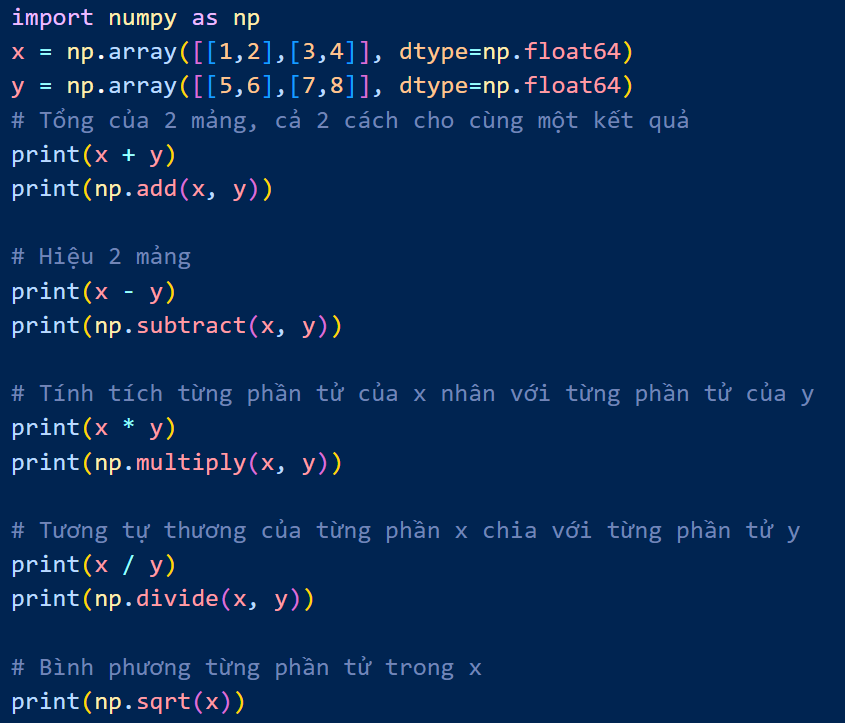
* **Datatypes**

Mỗi numpy array là một lưới các phần tử cùng kiểu dữ liệu. Numpy cung cấp một tập hợp lớn các kiểu dữ liệu số mà bạn có thể sử dụng để xây dựng các mảng. Numpy cố gắng đoán một kiểu dữ liệu khi bạn tạo một mảng, nhưng các hàm xây dựng các mảng thường cũng bao gồm một đối số tùy chọn để chỉ định rõ ràng kiểu dữ liệu



* **Array math**

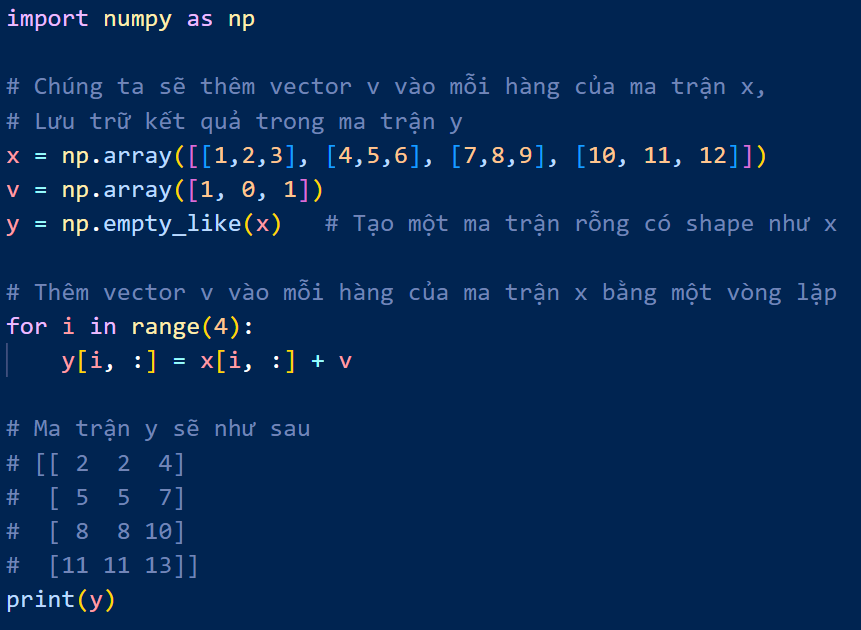
Ghép, cộng, nhân, hoán vị chỉ với một dòng code. Dưới đây là một số ví dụ về các phép toán số học và nhân khác nhau với các mảng Numpy



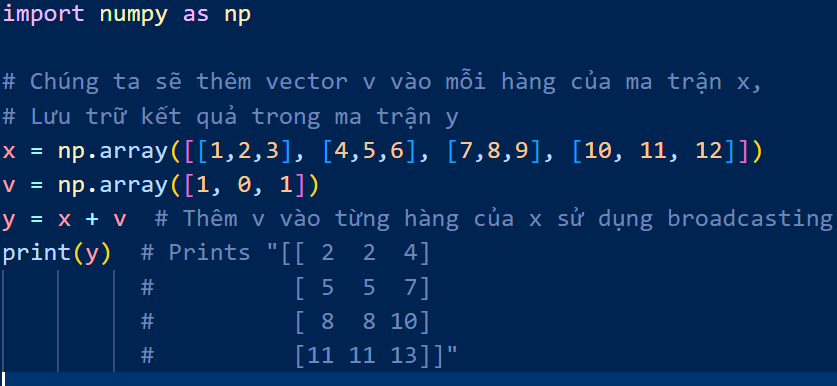
* **Broadcasting**

Broadcasting là một cơ chế mạnh mẽ cho phép thực thi các phép toán số học trên các numpy array có kích thước khác nhau. Chúng ta thường có một mảng nhỏ hơn và một mảng lớn hơn và chúng tôi muốn sử dụng mảng nhỏ hơn nhiều lần để thực hiện một số thao tác trên mảng lớn hơn.

Ví dụ: Giả sử rằng chúng ta muốn thêm một vectơ không đổi vào mỗi hàng của ma trận. Chúng ta có thể làm như thế này



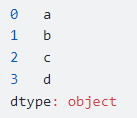
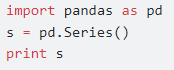
Cách này hoạt động bình thường, khi ma trận x quá lớn, việc sử dụng vòng lặp này sẽ rất chậm. Numpy broadcasting cho phép chúng ta thực thi tính toán này mà không cần phải tạo ra nhiều bản sao của v. Và đây là code khi sử dụng broadcasting



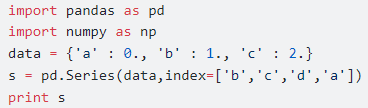
1. **Pandas**

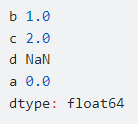
* **Giới thiệu**
  + Pandas là một thư viện mã nguồn mở được xây dựng dựa trên NumPy, sử dụng thao tác và phân tích dữ liệu, được thiết kế để cho phép bạn làm việc với dữ liệu được gắn nhãn hoặc quan hệ theo cách trực quan hơn
  + Pandas có ba cấu trúc dữ liệu: **Series**, **DataFrame**, **Panel**. Trong đó Panel sử dụng không phổ biến. Ta đề cập đến 2 loại là Series là mảng một chiều và DataFrame là mảng 2 chiều
* **Series:**
  + Series là mảng một chiều giống như mảng Numpy, hay như một cột của một bảng, nhưng nó bao gồm thêm một bảng đánh label. Series có thể được khởi tạo thông qua NumPy, kiểu Dict hoặc các dữ liệu vô hướng bình thường
  + Series có thể được tạo bằng cách sử dụng hàm tạo sau:

pandas.Series( data, index, dtype, copy)



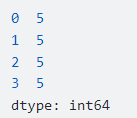
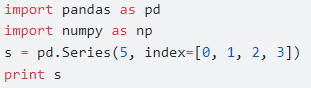
### **Tạo một Series từ dict**





### **Tạo một Series từ Scalar**

Nếu dữ liệu là một giá trị **scalar**, **index** phải được cung cấp. Giá trị sẽ được lặp lại để phù hợp với độ dài của **index**



### **Truy cập dữ liệu từ Series với Position**

Dữ liệu trong series có thể được truy cập tương tự như dữ liệu trong một ndarray.

* **DataFrame**

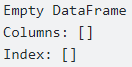
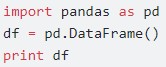
Một **Data frame** là một cấu trúc dữ liệu 2 chiều, dữ liệu được sắp xếp theo kiểu bảng trong các hàng và cột có các đặc trưng sau:

* + Các cột dữ liệu là các kiểu không đồng nhất: float64, int, bool, …
  + Kích thước table có thể thay đổi: các cột có thể thêm hoặc xoá đi
  + Các trục được dán nhãn (hàng và cột)
  + Có thể thực hiện các phép toán số học trên các hàng và cột

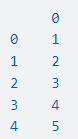
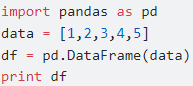
Một DataFrame có thể được tạo bằng cách sử dụng hàm tạo sau:

pandas.DataFrame( data, index, columns, dtype, copy)

### **Tạo một DataFrame rỗng**

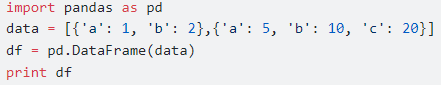


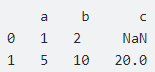
### **Tạo một DataFrame từ Lists**



### **Tạo một DataFrame từ một danh sách của Dicts**

Danh sách Dictionaries có thể được chuyển qua dưới dạng dữ liệu đầu vào để tạo DataFrame. Các keys của Dictionaries theo mặc định được lấy làm tên cột





NaN (Not a Number) được thêm vào các khu vực bị thiếu chọn cột, xóa cột, thêm cột, chọn hàng, xóa hàng, thêm hàng

1. **Science kit**

**Linear regression**

Hồi quy tuyền tính là một cách tiếp cận tuyến tính để mô hình hóa mối quan hệ giữa một biến phụ thuộc vô hướng y và một hoặc nhiều biến giải thích

import matplotlib.pyplot as plt

# represents the heights of a group of people in meters

heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]

# represents the weights of a group of people in kgs

weights = [[60], [65], [72.3], [75], [80]]

plt.title('Weights plotted against heights')

plt.xlabel('Heights in meters')

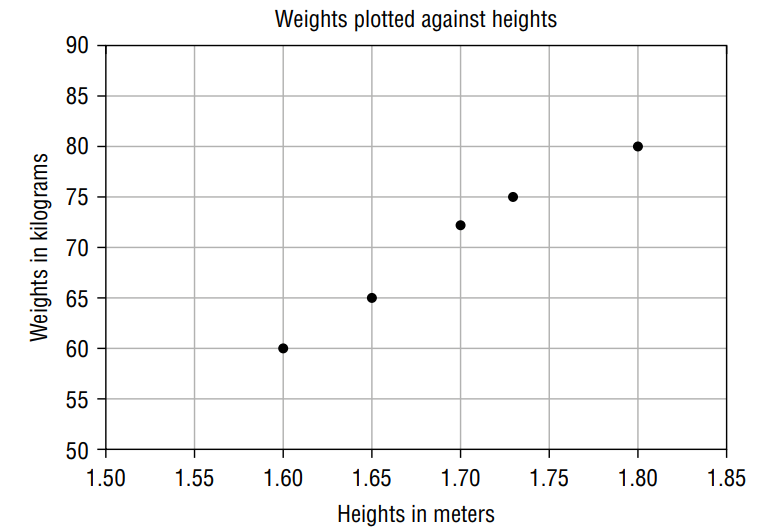
plt.ylabel('Weights in kilograms')

plt.plot(heights, weights, 'k.')

# axis range for x and y

plt.axis([1.5, 1.85, 50, 90])

plt.grid(True)



Từ biểu đồ, bạn có thể thấy rằng có mối tương quan giữa cân nặng và chiều cao của nhóm người này. Bạn có thể vẽ một đường thẳng thông qua các điểm và sử dụng điểm đó để dự đoán cân nặng của một người khác.

**Data Cleansing**

Data Cleaning (Làm sạch dữ liệu) là quá trình thay đổi hoặc loại bỏ dữ liệu không chính xác, trùng lặp, bị hỏng hoặc không đầy đủ bên trong cơ sở dữ liệu (database). Nếu dữ liệu không chính xác, các thuật toán và kết quả cho ra không đáng tin cậy (dù cho nó có vẻ đúng). Quy trình Data Cleaning không chỉ đơn thuần quan tâm đến việc xóa dữ liệu để tăng dung lượng cho dữ liệu mới. Mà còn tìm ra phương pháp tối đa hóa tính xác thực của tập dữ liệu mà không cần phải xóa thông tin.

Cleaning Rows with NaNs

Thay thế bằng giá trị trung bình của cột

import pandas as pd  
df = pd.read\_csv('NaNDataset.csv')  
df.isnull().sum()

A B C  
0 1 2.0 3  
1 4 11.0 6  
2 7 11.0 9  
3 10 11.0 12  
4 13 14.0 15  
5 16 17.0 18   
xóa hàng

df = pd.read\_csv('NaNDataset.csv')  
df = df.dropna() # drop all rows with NaN  
print(df)

A B C  
0 1 2.0 3  
3 10 11.0 12  
4 13 14.0 15  
5 16 17.0 18

1. **Matplotlib**

Để thực hiện các suy luận thống kê cần thiết, cần phải trực quan hóa dữ liệu của bạn và Matplotlib là một trong những giải pháp như vậy cho người dùng Python. Nó là một thư viện vẽ đồ thị rất mạnh mẽ hữu ích cho những người làm việc với Python và NumPy. Module được sử dụng nhiều nhất của Matplotib là Pyplot cung cấp giao diện như MATLAB nhưng thay vào đó, nó sử dụng Python và nó là nguồn mở.

Plot

%matplotlib inline

import matplotlib.pyplot as plt

from matplotlib import style

style.use("ggplot")

plt.plot(

[1,2,3,4,5,6,7,8,9,10],

[2,4.5,1,2,3.5,2,1,2,3,2]

)

plt.plot(

[1,2,3,4,5,6,7,8,9,10],

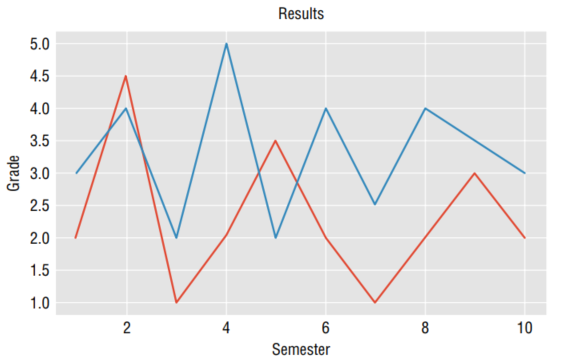
[3,4,2,5,2,4,2.5,4,3.5,3]

)

plt.title("Results") # sets the title for the chart

plt.xlabel("Semester") # sets the label to use for the x-axis

plt.ylabel("Grade") # sets the label to use for the y-axis



Biểu đồ cột

%matplotlib inline

import matplotlib.pyplot as plt

from matplotlib import style

style.use("ggplot")

plt.bar(

[1,2,3,4,5,6,7,8,9,10],

[2,4.5,1,2,3.5,2,1,2,3,2],

label = "Jim",

color = "m", # m for magenta

align = "center"

)

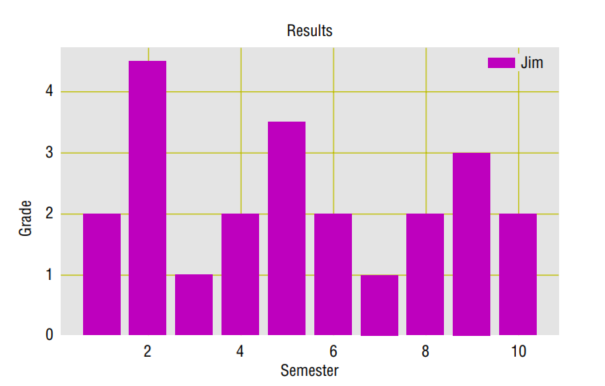
plt.title("Results")

plt.xlabel("Semester")

plt.ylabel("Grade")

plt.legend()

plt.grid(True, color="y")



Biểu đồ tròn

%matplotlib inline

import matplotlib.pyplot as plt

labels = ["Chrome", "Internet Explorer",

"Firefox", "Edge","Safari",

"Sogou Explorer","Opera","Others"]

marketshare = [61.64, 11.98, 11.02, 4.23, 3.79, 1.63, 1.52, 4.19]

explode = (0,0,0,0,0,0,0,0)

plt.pie(marketshare,

explode = explode, # fraction of the radius with which to

# offset each wedge

labels = labels,

autopct="%.1f%%", # string or function used to label the

# wedges with their numeric value

shadow=True,

startangle=45) # rotates the start of the pie chart by

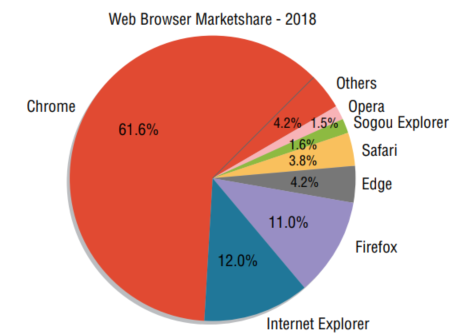
# angle degrees counterclockwise from the

# x-axis

plt.axis("equal") # turns off the axis lines and labels

plt.title("Web Browser Marketshare - 2018")

plt.show()



## **CHƯƠNG 2. HỌC CÓ GIÁM SÁT**

1. **Linear Regression(hồi quy tuyến tính)**

Trong học máy, hồi quy tuyến tính là một trong những thuật toán đơn giản nhất mà bạn có thể áp dụng cho tập dữ liệu để lập mô hình mối quan hệ giữa các tính năng và nhãn.

**Sử dụng Boston Dataset**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn.datasets import load\_boston

dataset = load\_boston()

print(dataset.data)

[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00] [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00] [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00] ... [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00] [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00] [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]

**Các thuộc tính của dataset**

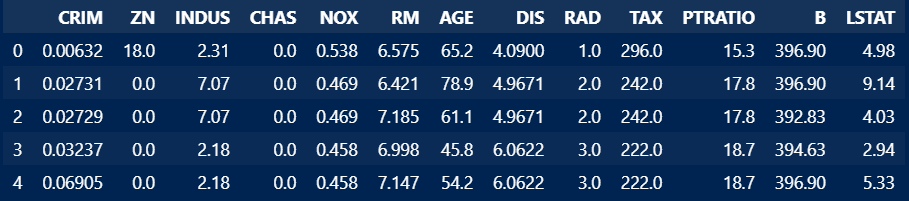
print(dataset.feature\_names)

['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO' 'B' 'LSTAT']

**Tải dữ liệu sử dụng Pandas DataFrame**

df = pd.DataFrame(dataset.data, columns=dataset.feature\_names)

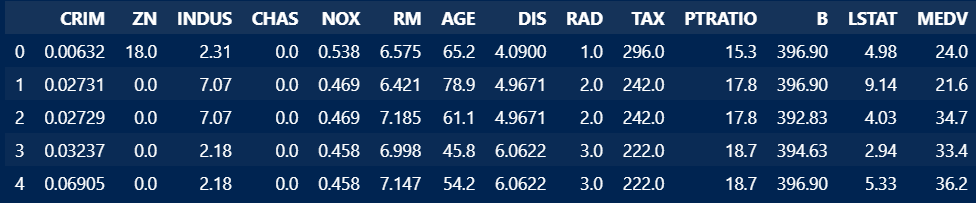
df.head()



**Nếu muốn thêm giá nhà vào DataFrame, thêm một cột gọi là MEDV**

df['MEDV'] = dataset.target

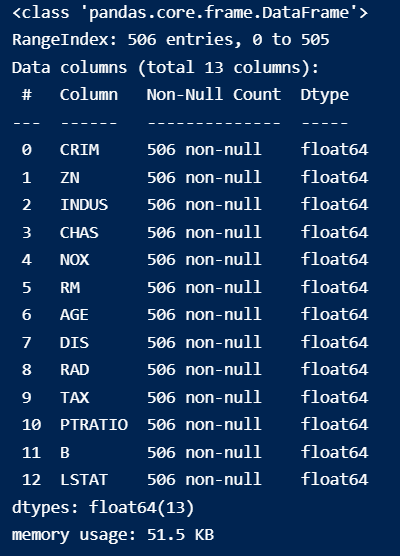
df.head()



**Data Cleansing**

Bước tiếp theo sẽ là làm sạch dữ liệu và thực hiện bất kỳ chuyển đổi nào nếu cần. Đầu tiên, hãy sử dụng info () để kiểm tra kiểu dữ liệu của từng trường:

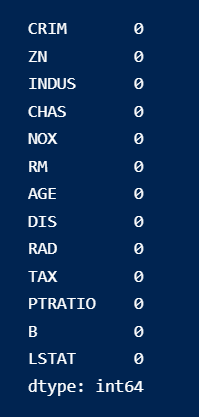
df.info()



Vì Scikit-learning chỉ hoạt động với các trường là số, bạn cần mã hóa các giá trị chuỗi thành các giá trị số. May mắn thay, tập dữ liệu chứa tất cả các giá trị số và vì vậy không cần mã hóa.

Tiếp theo, chúng ta cần kiểm tra xem có thiếu giá trị nào không. Để làm như vậy, hãy sử dụng isnull ():

df.isnull().sum()



Tập dữ liệu tốt, vì nó không có bất kỳ giá trị nào bị thiếu.

**Feature Selection**

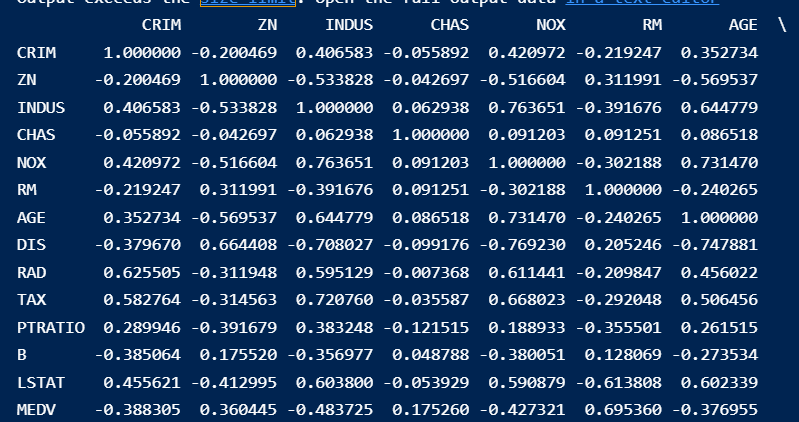
Vì có 13 tính năng trong tập dữ liệu, ta không muốn sử dụng tất cả các tính năng này để đào tạo mô hình của mình, bởi vì không phải tất cả chúng đều phù hợp. Thay vào đó, muốn chọn những đặc điểm ảnh hưởng trực tiếp đến kết quả (tức là giá nhà) để đào tạo mô hình. Đối với điều này, chúng ta có thể sử dụng hàm corr (). Các hàm corr () tính toán mối tương quan theo cặp của các cột.

df = pd.DataFrame(dataset.data, columns=dataset.feature\_names)

df['MEDV'] = dataset.target

corr = df.corr()

print(corr)



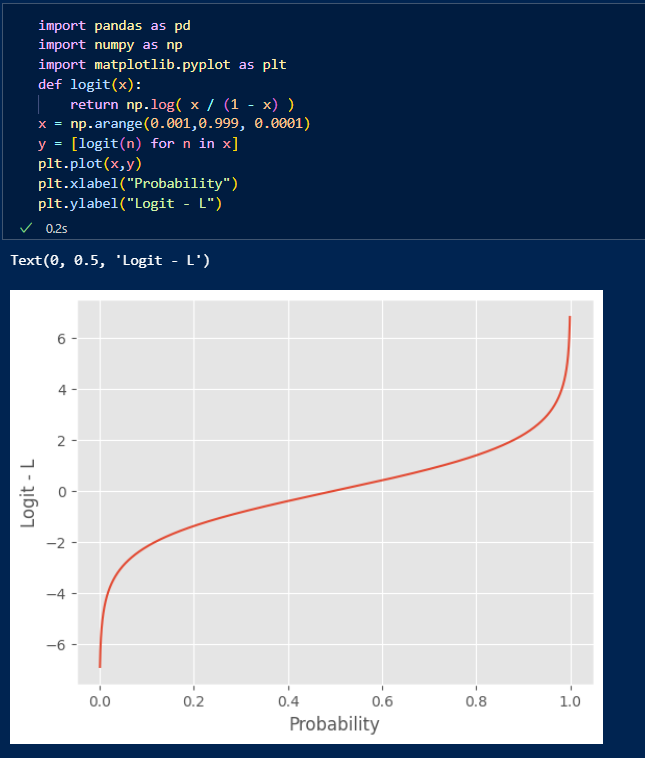
1. **Classification Using Logistic Regression**

Khác với hồi quy tuyến tính, hồi quy logistic không cố gắng dự đoán giá trị của một biến số đã cho một tập hợp các đầu vào. Thay vào đó, đầu ra của hồi quy logistic là xác suất của một điểm đầu vào nhất định thuộc về một lớp cụ thể. Đầu ra của hồi quy logistic luôn nằm trong [0,1].

**Hàm Logit**

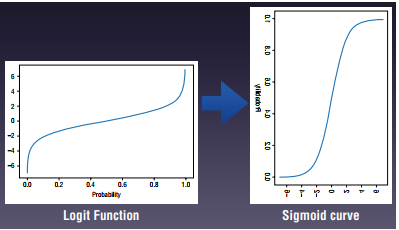


Hàm logit chuyển một biến trên (0, 1) thành một biến mới trên (–∞, ∞).



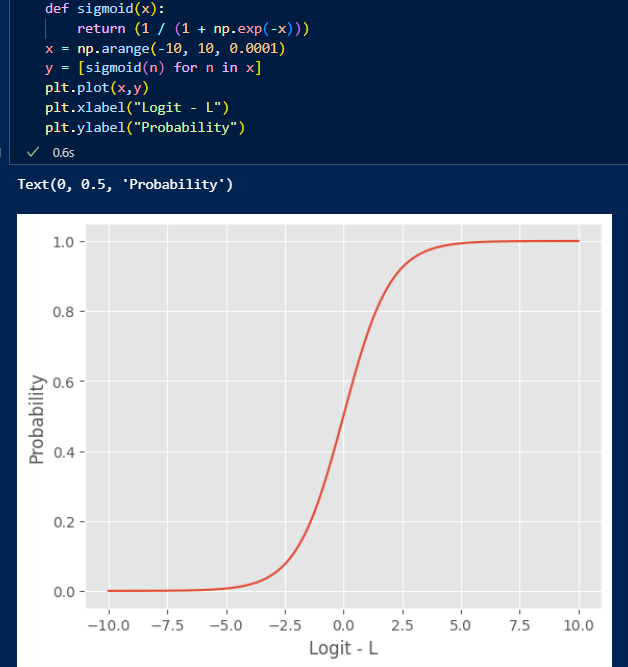
**Đường cong Sigmoid**

Đối với đường cong logit, hãy quan sát rằng trục x là xác suất và trục y là dãy số thực. Đối với hồi quy logistic, điều bạn thực sự muốn là một hàm ánh xạ các số trên hệ thống số thực với xác suất, đó là chính xác những gì bạn nhận được khi lật các trục của đường cong logit.



Khi bạn lật các trục, đường cong mà bạn nhận được được gọi là đường cong sigmoid. Các đường cong sigmoid thu được bằng cách sử dụng hàm Sigmoid, là hàm nghịch đảo của chức năng đăng nhập. Hàm Sigmoid được sử dụng để biến đổi các giá trị trên (–∞, ∞) thành số trên (0, 1).





import matplotlib.pyplot as plt

from sklearn.datasets import load\_breast\_cancer

cancer = load\_breast\_cancer()

#---copy from dataset into a 2-d list---

X = []

for target in range(2):

    X.append([[], []])

    for i in range(len(cancer.data)): # target is 0 or 1

        if cancer.target[i] == target:

            X[target][0].append(cancer.data[i][0]) # first feature -mean radius

            X[target][1].append(cancer.data[i][1]) # second feature —mean texture

colours = ("r", "b") # r: malignant, b: benign

fig = plt.figure(figsize=(10,8))

ax = fig.add\_subplot(111)

for target in range(2):

    ax.scatter(X[target][0],

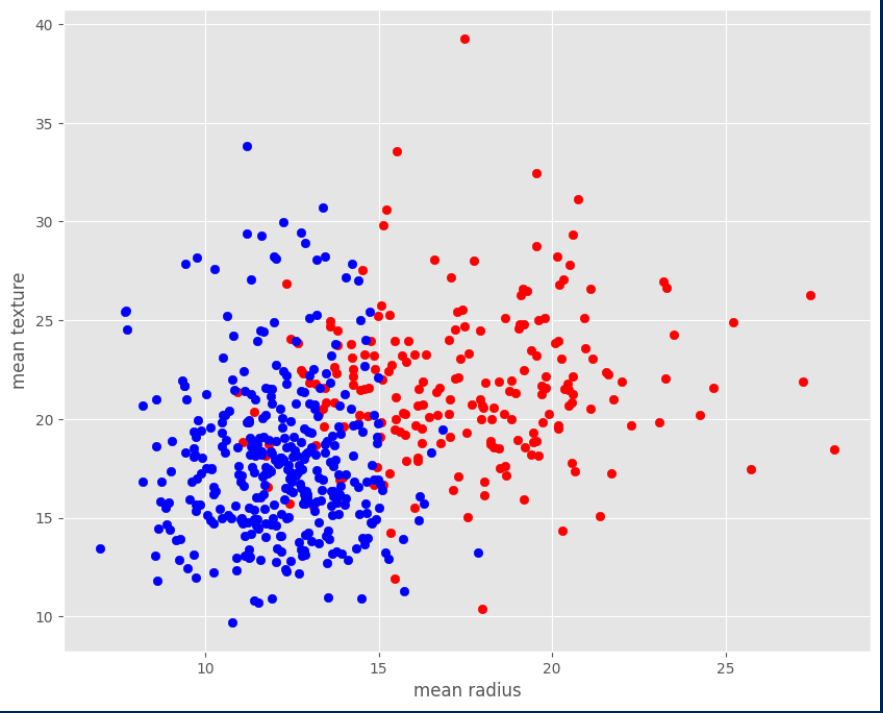
        X[target][1],

        c=colours[target])

ax.set\_xlabel("mean radius")

ax.set\_ylabel("mean texture")

plt.show()



Hiển thị khối u ác tính có màu đỏ và khối u lành tính có màu xanh lam. Từ biểu đồ phân tán này, bạn có thể thu thập dữ liệu đó khi khối u phát triển về bán kính và kết cấu càng tăng thì càng có nhiều khả năng được chẩn đoán là ác tính.

**Training Using One Feature**

Bây giờ chúng ta hãy sử dụng hồi quy logistic để cố gắng dự đoán xem một khối u có phải là ung thư hay không. để có được đã bắt đầu, hãy chỉ sử dụng tính năng đầu tiên của tập dữ liệu: bán kính trung bình. Đoạn mã sau vẽ biểu đồ phân tán cho biết khối u là ác tính hay lành tính dựa trên bán kính trung bình.

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches

from sklearn.datasets import load\_breast\_cancer

cancer = load\_breast\_cancer() # Load dataset

x = cancer.data[:,0] # mean radius

y = cancer.target # 0: malignant, 1: benign

colors = {0:'red', 1:'blue'} # 0: malignant, 1: benign

plt.scatter(x,y,facecolors='none',

    edgecolors=pd.DataFrame(cancer.target)[0].apply(lambda x:colors[x]),

    cmap=colors)

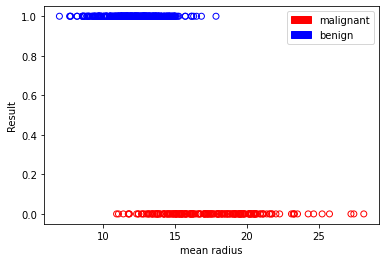
plt.xlabel("mean radius")

plt.ylabel("Result")

red = mpatches.Patch(color='red', label='malignant')

blue = mpatches.Patch(color='blue', label='benign')

plt.legend(handles=[red, blue], loc=1)



1. **Classification Using Support Vector Machines**

Giống như hồi quy logistics, SVM cũng là một thuật toán phân loại.

Ý tưởng chính đằng sau SVM là vẽ một đường thẳng giữa hai hoặc nhiều lớp trong cách tốt nhất có thể.

Khi đường được vẽ để phân tách các lớp, bạn có thể sử dụng nó để dự đoán

dữ liệu trong tương lai. Ví dụ, với chiều dài mõm và hình dạng tai của một loài mới động vật chưa biết, bây giờ bạn có thể sử dụng đường phân chia làm công cụ phân loại để dự đoán nếu con vật là một con chó hoặc một con mèo.

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

import numpy as np

from sklearn.datasets import make\_circles

#---X is features and c is the class labels---

X, c = make\_circles(n\_samples=500, noise=0.09)

rgb = np.array(['r', 'g'])

plt.scatter(X[:, 0], X[:, 1], color=rgb[c])

plt.show()

fig = plt.figure(figsize=(18,15))

ax = fig.add\_subplot(111, projection='3d')

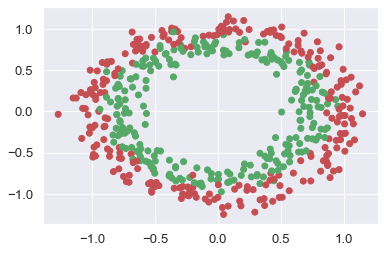
z = X[:,0]\*\*2 + X[:,1]\*\*2

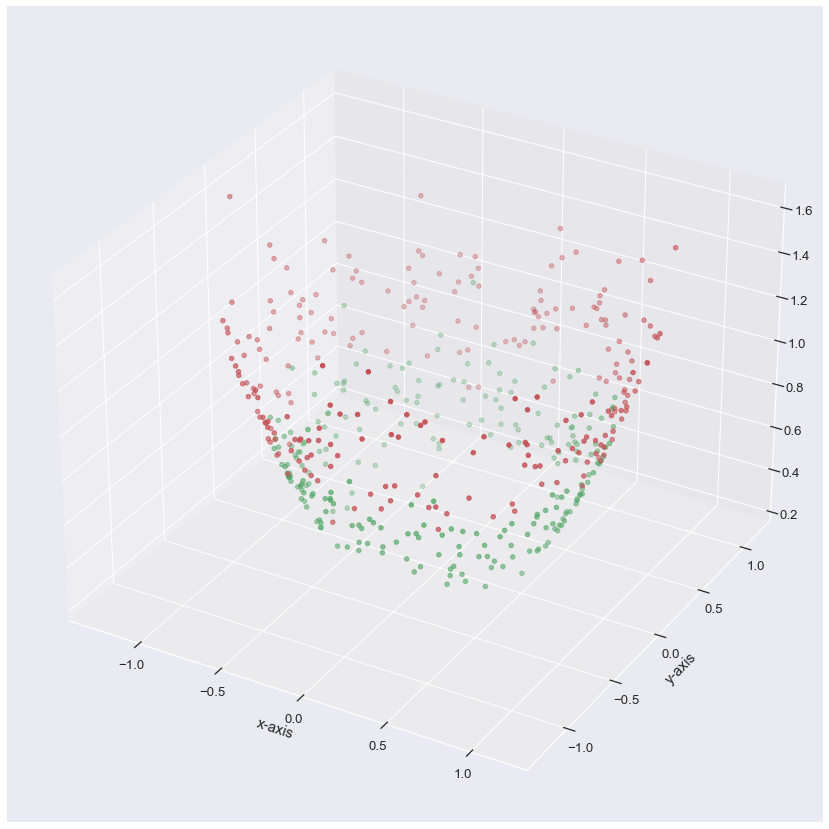
ax.scatter(X[:, 0], X[:, 1], z, color=rgb[c])

plt.xlabel("x-axis")

plt.ylabel("y-axis")

plt.show()





1. **Classification Using K-Nearest Neighbors(KNN)**

KNN là một thuật toán tương đối đơn giản so với các thuật toán khác mà chúng tôi đã thảo luận trong các chương trước. Nó hoạt động bằng cách so sánh phiên bản truy vấn khoảng cách đến các mẫu đào tạo khác và chọn K-hàng xóm gần nhất (do đó tên của nó). Sau đó, nó lấy phần lớn các lớp K-láng giềng này làm lớp dự đoán của trường hợp truy vấn.

from sklearn import svm, datasets

import matplotlib.pyplot as plt

iris = datasets.load\_iris()

X = iris.data[:, :2] # take the first two features

y = iris.target

#---plot the points---

colors = ['red', 'green', 'blue']

for color, i, target in zip(colors, [0, 1, 2], iris.target\_names):

plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)

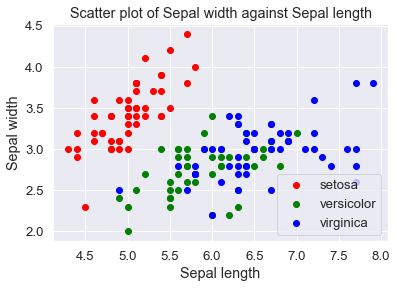
plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.legend(loc='best', shadow=False, scatterpoints=1)

plt.title('Scatter plot of Sepal width against Sepal length')

plt.show()



Hình trên cho thấy đồ thị phân tán của chiều rộng vách ngăn so với chiều dài vách ngăn.

Bây giờ chúng ta có thể sử dụng lớp KNeighborsClassifier của Scikit-learn để giúp chúng ta huấn luyện một mô hình trên bộ dữ liệu Iris bằng KNN. Để bắt đầu, hãy sử dụng k là 1.

from sklearn.neighbors import KNeighborsClassifier

k = 1

#---instantiate learning model---

knn = KNeighborsClassifier(n\_neighbors=k)

#---fitting the model---

knn.fit(X, y)

#---min and max for the first feature---

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

#---min and max for the second feature---

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

#---step size in the mesh---

h = (x\_max / x\_min)/100

#---make predictions for each of the points in xx,yy---

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),np.arange(y\_min, y\_max, h))

Z = knn.predict(np.c\_[xx.ravel(), yy.ravel()])

#---draw the result using a color plot---

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Accent, alpha=0.8)

#---plot the training points---

colors = ['red', 'green', 'blue']

for color, i, target in zip(colors, [0, 1, 2], iris.target\_names):

plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

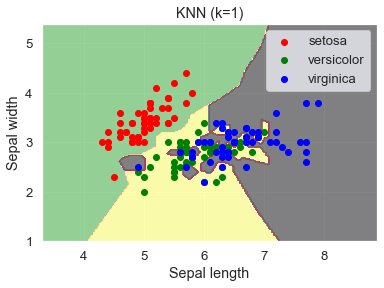
plt.title(f'KNN (k={k})')

plt.legend(loc='best', shadow=False, scatterpoints=1)

predictions = knn.predict(X)

#--classifications based on predictions---

print(np.unique(predictions, return\_counts=True))



Hình trên cho thấy ranh giới phân loại sử dụng k bằng 1. Lưu ý rằng đối với k = 1, bạn thực hiện dự đoán của mình chỉ dựa trên một mẫu duy nhất—láng giềng gần nhất. Điều này làm cho dự đoán của bạn rất nhạy cảm với tất cả các loại biến dạng, chẳng hạn như ngoại lệ, dán nhãn sai, v.v. Nói chung, đặt k = 1 thường dẫn tới quá khớp, và kết quả là dự đoán của bạn thường không chính xác lắm.

## **CHƯƠNG 3. HỌC KHÔNG GIÁM SÁT**

Cho đến nay, tất cả các thuật toán học máy mà bạn đã thấy đều là học có giám sát. Có nghĩa là, tất cả các bộ dữ liệu đã được dán nhãn, phân lớp hoặc phân loại.

Các tập dữ liệu đã được gắn nhãn được gọi là dữ liệu được gắn nhãn, trong khi các tập dữ liệu chưa được gắn nhãn được gọi là dữ liệu chưa được gắn nhãn.

1. **Unsupervised Learning Using K-Means**

Vì không có nhãn nào trong dữ liệu không được gắn nhãn, do đó chúng ta quan tâm đến việc chúng ta có thể tìm thấy các mẫu trong dữ liệu không được gắn nhãn đó. Kỹ thuật tìm kiếm các mẫu này trong dữ liệu không được gắn nhãn được gọi là phân cụm(clustering). Mục đích chính của phân cụm là tách các nhóm có các đặc điểm giống nhau và phân chúng thành các nhóm (thường được biết đến như các cụm-cluster).

Một trong những thuật toán phổ biến được sử dụng để phân cụm là thuật toán K-Means.

K-Means clustering là một loại học tập không giám sát:

* Được sử dụng khi bạn có dữ liệu chưa được gắn nhãn
* Mục tiêu là tìm các nhóm trong dữ liệu, với số lượng các nhóm được đại diện bởi K

Trong thuật toán K-means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thể nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau.

Mục tiêu của phân cụm K-Means là đạt được những điều sau:

* K centroid đại diện cho trung tâm của các cụm
* Nhãn cho dữ liệu đào tạo

**Tóm tắt thuật toán**

**Đầu vào:** Dữ liệu XX và số lượng cluster cần tìm KK.

**Đầu ra:** Các center MM và label vector cho từng điểm dữ liệu YY.

1. Chọn KK điểm bất kỳ làm các center ban đầu.
2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất.
3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.
4. Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất các các điểm dữ liệu đã được gán vào cluster đó sau bước 2.
5. Quay lại bước 2.

Chúng ta có thể đảm bảo rằng thuật toán sẽ dừng lại sau một số hữu hạn vòng lặp. Thật vậy, vì hàm mất mát là một số dương và sau mỗi bước 2 hoặc 3, giá trị của hàm mất mát bị giảm đi. Theo kiến thức về dãy số trong chương trình cấp 3: *nếu một dãy số giảm và bị chặn dưới thì nó hội tụ!* Hơn nữa, số lượng cách phân nhóm cho toàn bộ dữ liệu là hữu hạn nên đến một lúc nào đó, hàm mất mát sẽ không thể thay đổi, và chúng ta có thể dừng thuật toán tại đây.

**How Clustering in K-Means Works**

from \_\_future\_\_ import print\_function

import numpy as np

import matplotlib.pyplot as plt

from scipy.spatial.distance import cdist

np.random.seed(11)

means = [[2, 2], [8, 3], [3, 6]]

cov = [[1, 0], [0, 1]]

N = 500

X0 = np.random.multivariate\_normal(means[0], cov, N)

X1 = np.random.multivariate\_normal(means[1], cov, N)

X2 = np.random.multivariate\_normal(means[2], cov, N)

X = np.concatenate((X0, X1, X2), axis = 0)

K = 3

original\_label = np.asarray([0]\*N + [1]\*N + [2]\*N).T

def kmeans\_display(X, label):

K = np.amax(label) + 1

X0 = X[label == 0, :]

X1 = X[label == 1, :]

X2 = X[label == 2, :]

plt.plot(X0[:, 0], X0[:, 1], 'b^', markersize = 4, alpha = .8)

plt.plot(X1[:, 0], X1[:, 1], 'go', markersize = 4, alpha = .8)

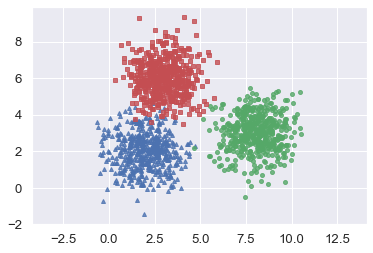
plt.plot(X2[:, 0], X2[:, 1], 'rs', markersize = 4, alpha = .8)

plt.axis('equal')

plt.plot()

plt.show()

kmeans\_display(X, original\_label)



Trong đồ thị trên, mỗi cluster tương ứng với một màu. Có thể nhận thấy rằng có một vài điểm màu đỏ bị lẫn sang phần cluster màu xanh.

Từ kết quả này chúng ta thấy rằng thuật toán K-means clustering làm việc khá thành công, các centers tìm được khá gần với kỳ vọng ban đầu. Các điểm thuộc cùng một cluster hầu như được phân vào cùng một cluster (trừ một số diểm màu đỏ ban đầu đã bị phân nhầm vào cluster màu xanh da trời, nhưng tỉ lệ là nhỏ và có thể chấp nhận được).

## **CHƯƠNG 4. CASE STYDY VÀ DEPLOY**

Mục tiêu chính của học máy là tạo ra một mô hình mà bạn có thể sử dụng để đưa ra dự đoán. Một cách hay để triển khai mô hình học máy của bạn là xây dựng một REST (Representational State Transfer) API, để những người khác có thể truy cập mô hình có thể không quen thuộc với cách thức hoạt động của máy học. Sử dụng REST, bạn có thể xây dựng các ứng dụng front-end đa nền tảng (như iOS, Android, Windows, v.v.) và chuyển dữ liệu đến mô hình để xử lý. Kết quả sau đó có thể được trả lại cho ứng dụng.

Trong chương này, chúng ta sẽ đi qua một nghiên cứu điển hình, xây dựng một máy học mô hình và sau đó triển khai nó dưới dạng dịch vụ REST. Cuối cùng, chúng ta sẽ xây dựng một giao diện điều khiển ứng dụng front-end sử dụng Python để cho phép người dùng đưa ra một số dự đoán.

**CASE STUDY**

Đối với nghiên cứu trường hợp này, chúng tôi sẽ giúp dự đoán khả năng một người được chẩn đoán mắc bệnh tiểu đường dựa trên một số phép đo chẩn đoán người đó.

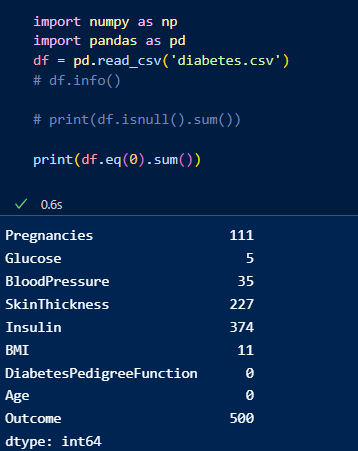
Bộ dữ liệu mà bạn sẽ sử dụng trong chương này là từ cơ sở dữ liệu này: <https://www.kaggle.com/uciml/pima-indians-diabetes-database>. tập dữ liệu này chứa một số yếu tố dự đoán độc lập về y tế và một mục tiêu. Các tính năng của nó

bao gồm những điều sau đây:

* Mang thai: Số lần mang thai
* Glucose: Nồng độ glucose huyết tương sau 2 giờ trong glucose uống

kiểm tra khả năng chịu đựng

* Huyết áp: Huyết áp tâm trương (mm Hg)
* SkinThickness: Độ dày nếp gấp da cơ tam đầu (mm)
* Insulin: Insulin huyết thanh 2 giờ (mu U/ml)
* BMI: Chỉ số khối cơ thể (cân nặng tính bằng kg/(chiều cao tính bằng m)^2)
* DiabetesPedigreeFunction: Chức năng phả hệ bệnh tiểu đường
* Tuổi: Tuổi (năm)
* Kết quả: 0 (không mắc bệnh tiểu đường) hoặc 1 (bệnh tiểu đường)



Có nhiều cách để xử lý trường hợp 0 đối với các tính năng, nhưng để đơn giản, hãy thay thế các giá trị 0 bằng NaN:

df[['Glucose','BloodPressure','SkinThickness',

'Insulin','BMI','DiabetesPedigreeFunction','Age']] = \

df[['Glucose','BloodPressure','SkinThickness',

'Insulin','BMI','DiabetesPedigreeFunction','Age']].replace

(0,np.NaN)

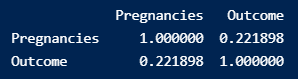
Khi các giá trị NaN đã thay thế các số 0 trong DataFrame, bây giờ có thể thay thế chúng bằng giá trị trung bình của mỗi cột như sau

df.fillna(df.mean(), inplace = True)

**Examining the Correlation Between the Features**

Bước tiếp theo là kiểm tra các tính năng độc lập khác nhau ảnh hưởng như thế nào đến kết quả (cho dù bệnh nhân có bị tiểu đường hay không). Để làm điều đó, bạn có thể gọi hàm corr() trên DataFrame:

corr = df.corr()



Hàm corr() tính toán mối tương quan theo cặp của các cột. Ví dụ,

**Plotting the Correlation Between Features**

import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 10))

cax = ax.matshow(corr,cmap='coolwarm', vmin=-1, vmax=1)

fig.colorbar(cax)

ticks = np.arange(0,len(df.columns),1)

ax.set\_xticks(ticks)

ax.set\_xticklabels(df.columns)

plt.xticks(rotation = 90)

ax.set\_yticklabels(df.columns)

ax.set\_yticks(ticks)

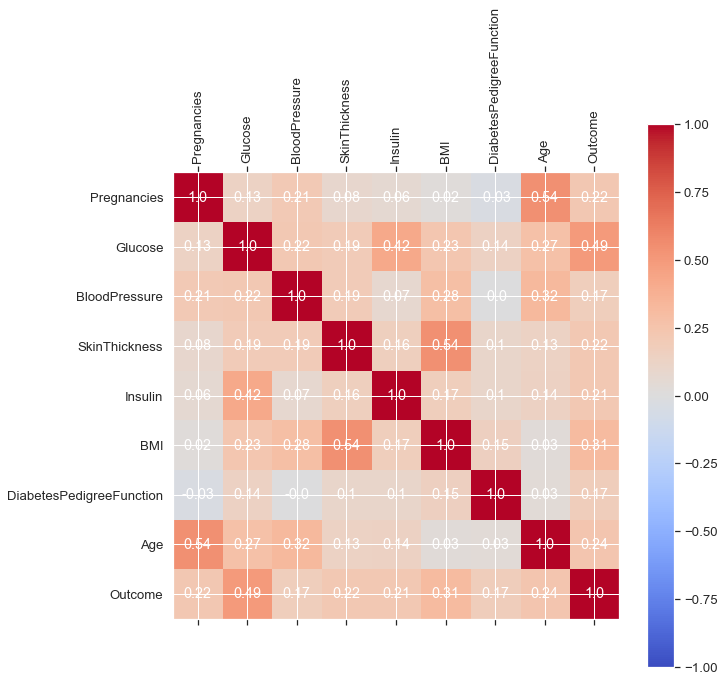
#---print the correlation factor---

for i in range(df.shape[1]):

    for j in range(9):

        text = ax.text(j, i, round(corr.iloc[i][j],2),ha="center", va="center", color="w")

plt.show()



**Đánh giá các thuật toán**

* Hồi quy logistic
* K-Láng giềng gần nhất (KNN)
* Support Vector Machines (SVM)—Hạt nhân tuyến tính và RBF

**Hồi quy logistic**

Đối với thuật toán đầu tiên, chúng tôi sẽ sử dụng hồi quy logistic. Thay vì chia nhỏ tập dữ liệu thành các tập huấn luyện và kiểm tra, chúng tôi sẽ sử dụng xác thực chéo 10 lần để lấy điểm trung bình của thuật toán được sử dụng:

from sklearn import linear\_model

from sklearn.model\_selection import cross\_val\_score

#---features---

X = df[['Glucose','BMI','Age']]

#---label---

y = df.iloc[:,8]

log\_regress = linear\_model.LogisticRegression()

log\_regress\_score = cross\_val\_score(log\_regress, X, y, cv=10,

scoring=accuracy.mean()

print(log\_regress\_score)

Kết quả đào tạo mô hình nên sử dụng giá trị trung bình là 0,7617737525632263.

0.7669856459330144

Chúng tôi cũng sẽ lưu kết quả này vào một danh sách để có thể so sánh với

điểm số của các thuật toán khác:

result = []

result.append(log\_regress\_score)

**K-Hàng xóm gần nhất**

Thuật toán tiếp theo mà chúng ta sẽ sử dụng là K-Hàng xóm gần nhất (KNN). Trong ngoài việc sử dụng xác thực chéo 10 lần để có được điểm trung bình của thuật toán, chúng ta cũng cần thử các giá trị khác nhau của k để có được giá trị tối ưu k để chúng ta có được độ chính xác tốt nhất

from sklearn.neighbors import KNeighborsClassifier

#---empty list that will hold cv (cross-validates) scores---

cv\_scores = []

#---number of folds---

folds = 10

#---creating odd list of K for KNN---

ks = list(range(1,int(len(X) \* ((folds - 1)/folds)), 2))

#---perform k-fold cross validation---

for k in ks:

knn = KNeighborsClassifier(n\_neighbors=k)

score = cross\_val\_score(knn, X, y, cv=folds, scoring='accuracy').mean()

cv\_scores.append(score)

#---get the maximum score---

knn\_score = max(cv\_scores)

#---find the optimal k that gives the highest score---

optimal\_k = ks[cv\_scores.index(knn\_score)]

print(f"The optimal number of neighbors is {optimal\_k}")

print(knn\_score)

result.append(knn\_score)

----> 0.7721462747778537

So sánh với giá trị kết quả bên trên. Ta thấy được rằng thuật toán KNN có kết quả tối ưu hơn

Ta sẽ quyết định trainning data bằng KNN với k=19

# 

# **PHẦN II: DEEPLEARNING**

## **CHƯƠNG 5: BIỂU DIỄN DỮ LIỆU VÀ TENSORFLOW**

Tensorflow là gì – Với sự bùng nổ của lĩnh vực Trí Tuệ Nhân Tạo – A.I. trong thập kỷ vừa qua, machine learning và deep learning rõ ràng cũng phát triển theo cùng. Và ở thời điểm hiện tại, TensorFlow chính là thư viện mã nguồn mở cho machine learning nổi tiếng nhất thế giới, được phát triển bởi các nhà nghiên cứu từ Google. Việc hỗ trợ mạnh mẽ các phép toán học để tính toán trong machine learning và deep learning đã giúp việc tiếp cận các bài toán trở nên đơn giản, nhanh chóng và tiện lợi hơn nhiều.

Các hàm được dựng sẵn trong thư viện cho từng bài toán cho phép TensorFlow xây dựng được nhiều neural network. Nó còn cho phép bạn tính toán song song trên nhiều máy tính khác nhau, thậm chí trên nhiều CPU, GPU trong cùng 1 máy hay tạo ra các dataflow graph – đồ thị luồng dữ liệu để dựng nên các model. Nếu bạn muốn chọn con đường sự nghiệp trong lĩnh vực A.I. này, nắm rõ những điều cơ bản của TensorFlow thực sự rất quan trọng.

Được viết bằng C++ và thao tác interface bằng Python nên phần performance của TensorFlow cực kỳ tốt. Đối tượng sử dụng nó cũng đa dạng không kém: từ các nhà nghiên cứu, nhà khoa học dữ liệu và dĩ nhiên không thể thiếu các lập trình viên.

### **Tensor**

Tên của TensorFlow được đưa ra trực tiếp là nhờ vào framework cốt lõi của nó: Tensor. Trong TensorFlow, tất cả các tính toán đều liên quan tới các tensor. 1 tensor là 1 **vector** hay **ma trận** của n-chiều không gian đại diện cho tất cả loại dữ liệu. Tất cả giá trị trong 1 tensor chứa đựng loại dữ liệu giống hệt nhau với 1 **shape** đã biết (hoặc đã biết 1 phần). Shape của dữ liệu chính là chiều của ma trận hay mảng.  
  
1 tensor có thể được bắt nguồn từ dữ liệu input hay kết quả của 1 tính toán. Trong TensorFlow, tất cả các hoạt động được tiến hành bên trong 1 **graph** – biểu đồ. Biểu đồ là 1 tập hợp tính toán được diễn ra liên tiếp. Mỗi operation được gọi là 1 **op node** (operation node) và được kết nối với nhau.  
  
Biểu đồ phát thảo các op và kết nối giữa các node. Tuy nhiên, nó không hiển thị các giá trị. Phần edge của các node chính là tensor, 1 cách để nhập operation với dữ liệu.

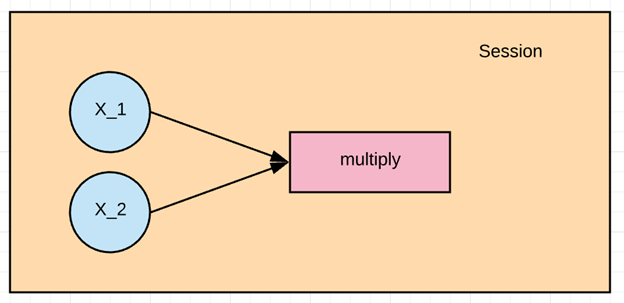
### **Graph**

TensorFlow sử dụng framework dạng biểu đồ. Biểu đồ tập hợp và mô tả tất cả các chuỗi tính toán được thực hiện trong quá trình training. Biểu đồ cũng mang rất nhiều lợi thế:  
  
– Nó được làm ra để chạy trên nhiều CPU hay GPU, ngay cả các hệ điều hành trên thiết bị điện thoại.  
– Tính di động của biểu đồ cho phép bảo toàn các tính toán để bạn sử dụng ngay hay sau đó. Biểu đồ có thể được lưu lại để thực thi trong tương lai.  
– Tất cả tính toán trong biểu đồ được thực hiện bằng cách kết nối các tensor lại với nhau. 1 **tensor** có 1 **node** và 1 **edge**. Node mang operation toán học và sản xuất các output ở đầu cuối. Các edge giải thích mối quan hệ input/output giữa các node.

import numpy as np  
import tensorflow as tf

Trong 2 line code đầu, ta đã import tensorflow là tf. Với Python, nó chỉ là 1 bài tập thông dụng khi dùng tên viết tắt cho 1 thư viện. Lợi thế là để tránh gõ đầy đủ tên của thư viện khi ta cần sử dụng nó.

Ví dụ: chúng ta có thể import tensorflow là tf, và gọi tf khi chúng ta muốn dùng 1 chức năng TensorFlow.  
  
Ta hãy luyện tập quy trình làm việc cơ bản của TensorFlow với 1 ví dụ cơ bản sau đây: Tạo 1 đồ thị tính toán nhân 2 số lại với nhau.  
  
Chúng ta sẽ nhân X\_1 và X\_2 lại với nhau. TensorFlow sẽ tạo 1 node để kết nối với operation. Trong ví dụ này, nó được gọi là multiply. Khi đồ thị được xác định, engine tính toán TensorFlow sẽ nhân X\_1 và X\_2 lại cùng nhau.

  
Cuối cùng, chúng ta chạy 1 session TensorFlow mà sẽ chạy đồ thị tính toán với giá trị của X\_1 và X\_2 và print phần kết quả của phép nhân.

Chúng ta có một cách cơ bản đó là dùng constant. Ta có ví dụ

import tensorflow as tf

x1=tf.constant(1, name='cons\_1')

print(x1)

x2=tf.constant(2.)

print(x2)

x3=tf.constant(3,dtype='float32')

print(x3)

v1=tf.constant([1,2,3],dtype=tf.float32)

print(v1)

sess=tf.Session()

sess.run(v1)

y=x2+x3

sess.run(y)

cách khác.

Hãy xác định các input node của X\_1 và X\_2. Khi ta tạo 1 node trong TensorFlow, chúng ta phải chọn nó là loại node gì. Các node X1 và X2 sẽ là node placeholder – node giữ chỗ. Placeholder chỉ định 1 giá trị mới mỗi khi ta làm 1 phép tính. Ta sẽ tạo chúng như là 1 node TF chấm placeholder.

### **Bước 1: Xác định giá trị**

X\_1 = tf.placeholder(tf.float32, name = "X\_1")  
X\_2 = tf.placeholder(tf.float32, name = "X\_2")

Khi ta tạo 1 node placeholder, ta phải gửi loại dữ liệu sẽ được thêm số tại đây nên ta có thể sử dụng 1 loại dữ liệu floating-point, hãy dùng tf.float32. Chúng ta cũng cần phải cho node này 1 cái tên. Tên này sẽ xuất hiện khi ta nhìn vào phần trực quan dạng đồ thị của model. Hãy đặt tên node X\_1 này bằng cách nhập 1 tham số được gọi bằng tên với 1 giá trị của X\_1 và bây giờ hãy xác định X\_2 theo cách tương tự.

### **Bước 2: Xác định phần tính toán**

multiply = tf.multiply(X\_1, X\_2, name = "multiply")

Giờ ta có thể xác định node sẽ thực hiện operation phép nhân. Trong TensorFlow, chúng ta có thể làm điều đó bằng cách tạo 1 node tf.multiply.  
  
Ta sẽ nhập node X\_1 và X\_2 tới node nhân. Nó sẽ nói với TensorFlow để liên kết những node đó trong đồ thị tính toán, nên ta đang yêu cầu nó để pull các giá trị từ x và y và nhân phần kết quả. Hãy cho node nhân cái tên multiply. Nó là toàn bộ định nghĩa cho đồ thị tính toán đơn giản này.

### **Bước 3: Thực thi operation**

Để thực thi các operation trong đồ thị, ta phải tạo 1 session. Trong TensorFlow, nó được thực hiện bằng tf.Session(). Giờ ta có 1 session ta có thể hỏi session để chạy operation trên đồ thị tính toán của ta bằng cách gọi session. Để chạy phần tính toán, chúng ta sẽ dùng run.  
  
Khi operation bổ sung chạy, nó sẽ thấy rằng nó cần để lấy các giá trị của node X\_1 và X\_2, nên chúng ta cũng cần cung cấp các trị cho X\_1 và X\_2. Ta có thể dùng điều đó bằng cách cung cấp 1 tham số được gọi là feed\_dict. Chúng ta chuyển giá trị 1,2,3 cho X\_1 và 4,5,6 cho X\_2.  
  
Chúng ta print phần kết quả với print(result). Chúng ta sẽ thấy 4,10 và 18 cho 1×4, 2×5 và 3,6.

X\_1 = tf.placeholder(tf.float32, name = "X\_1")  
X\_2 = tf.placeholder(tf.float32, name = "X\_2")  
  
multiply = tf.multiply(X\_1, X\_2, name = "multiply")  
with tf.Session() as session:  
 result = session.run(multiply, feed\_dict={X\_1:[1,2,3], X\_2:[4,5,6]})  
 print(result)

[ 4. 10. 18.]

Variables

## Các option tải dữ liệu vào TensorFlow

Bước đầu tiên trước khi train 1 thuật toán machine learning là load dữ liệu. Có 2 cách thông dụng để load dữ liệu:  
  
1. Load dữ liệu vào bộ nhớ: đây là phương pháp đơn giản nhất. Bạn load tất cả dữ liệu vào bộ nhớ như 1 mảng đơn. Bạn cũng có thể viết code bằng Python. Những dòng code này không liên quan gì tới TensorFlow.

2. Pipeline dữ liệu TensorFlow. TensorFlow sở hữu built-in API và nó sẽ giúp bạn load dữ liệu, thực thi các operation và feed thuật toán machine learning 1 cách dễ dàng. Phương pháp này hoạt động tốt đặc biệt khi bạn có 1 dataset lớn. Ví dụ: các hình ảnh thu được được biết khá là khổng lồ và không thể fit vào bộ nhớ. Pipeline dữ liệu sẽ tự quản lý phần bộ nhớ.

### Giải pháp sẽ là gì?

#### **Load dữ liệu vào bộ nhớ**

Nếu dataset của bạn không quá lớn, chẳng hạn như dưới 10 GB, bạn có thể dùng phương pháp đầu tiên. Dữ liệu có thể dễ dàng fit vào bộ nhớ. Bạn cũng có thể dùng 1 thư viện nổi tiếng có tên là Pandas để import các tệp CSV. 1

#### **Load dữ liệu với TensorFlow pipeline**

Phương pháp thứ 2 sẽ hoạt động tốt nhất nếu bạn có 1 dataset lớn. Ví dụ: nếu bạn có 1 dataset nặng 50 GB và máy tính của bạn chỉ có 16GB dung lượng thì rõ ràng là máy sẽ crash thôi.

Trong tình huống này, bạn cần dựng 1 TensorFlow pipeline. Đường ống sẽ load dữ liệu trong batch, hay chunk nhỏ. Mỗi batch sẽ được push tới pipeline và sẵn sàng cho việc training. Dựng 1 pipeline là 1 giải pháp tuyệt vời vì nó còn cho bạn sử dụng phép tính toán song song. Nghĩa là TensorFlow sẽ train model qua nhiều CPU. Thúc đẩy sự tính toán và cho phép training mạng lưới thần kinh mạnh mẽ hơn.  
  
Tóm tắt:

-Nếu bạn có 1 set dữ liệu nhỏ, bạn có thể load dữ liệu trong bộ nhớ với thư viện Pandas.

-Nếu bạn có 1 set dữ liệu lớn và muốn sử dụng nhiều CPU, thì bạn sẽ thoải mái hơn khi làm việc cùng TensorFlow pipeline.

Tạo đường ống TensorFlow

Trong ví dụ trước, ta đã thêm thủ công 3 giá trị cho X\_1 và X\_2. Giờ chúng ta sẽ xem cách load dữ liệu tới TensorFlow.

#### **Bước 1: Tạo dữ liệu**

Đầu tiên, hãy dùng thư viện numpy để tạo ra 2 giá trị ngẫu nhiên.

import numpy as np  
x\_input = np.random.sample((1,2))  
print(x\_input)

#### **Bước 2: Tạo placeholder**

Giống như ví dụ trước, ta tạo 1 placeholder với tên là X. Ta cần phải chỉ định shape của TensorFlow 1 cách rõ ràng. Trong vài trường hợp, ta sẽ tải 1 mảng với chỉ 2 giá trị. Ta có thể viết shape như là shape=[1,2]

# using a placeholder  
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')

#### **Bước 3: Xác định phương pháp cho set dữ liệu**

Tiếp theo, chúng ta cần phải xác định set dữ liệu nơi ta có thể nhập giá trị của placeholder x. Chúng ta cần dùng phương pháp tf.data.Dataset.from\_tensor\_slices

dataset = tf.data.Dataset.from\_tensor\_slices(x)

#### **Bước 4: Tạo đường ống**

Trong bước này, ta cần phải khởi tạo pipeline nơi dữ liệu chạy qua. Ta cần tạo 1 iterator với make\_initializable\_iterator. Đặt tên cho nó là iterator. Rồi ta cần gọi iterator này để feed cho batch dữ liệu tiếp theo, get\_next. Ta đặt tên bước này là get\_next. Hãy lưu ý điều này trong ví dụ của mình, vì chỉ có duy nhất 1 batch dữ liệu với 2 giá trị.

iterator = dataset.make\_initializable\_iterator()   
get\_next = iterator.get\_next()

#### **Bước 5: Thực thi operation**

Bước cuối cũng tương tự như ví dụ trước. Ta khởi tạo 1 session và chạy operation iterator. Ta feed cái feed\_dict với giá trị được tạo ra bởi numpy. 2 giá trị này sẽ nhập vào placeholder x. Rồi chúng ta chạy get\_next để print kết quả.

with tf.Session() as sess:  
 # feed the placeholder with data  
 sess.run(iterator.initializer, feed\_dict={ x: x\_input })   
 print(sess.run(get\_next)) # output [ 0.52374458 0.71968478]

Linear Regression

8 bước để xử lý vấn đề học máy

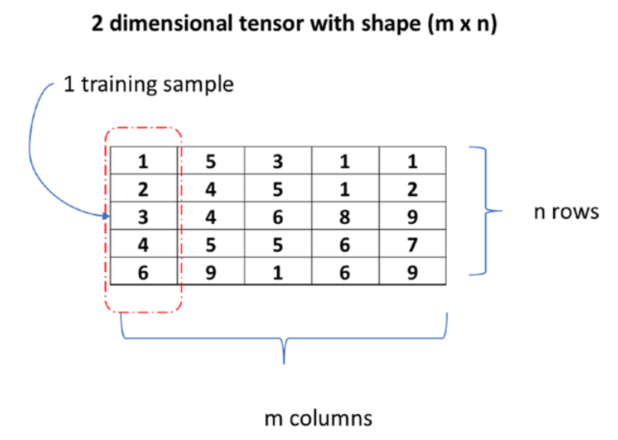
* Hiểu dữ liệu
* Tiền xử lý dữ liệu
* Chọn các đặc trưng trong dữ liệu
* Chọn thuật toán để học
* Đào tạo
* Tinh chỉnh dữ liệu
* Đánh giá
* Deployment

## **CHƯƠNG 6: HỌC SÂU VÀ KERAS**

DNN là dữ liệu đầu vào, nơ-ron, hàm kích hoạt, lớp (tức là nhóm nơ-ron), kết nối giữa các nơ-ron hoặc cạnh, một lớp học. thủ tục (nghĩa là thuật toán lan truyền ngược) và lớp đầu ra.

Dữ liệu đầu vào

Dữ liệu đầu vào cho thuật toán DL có thể thuộc nhiều loại khác nhau. Về cơ bản, mô hình hiểu dữ liệu là “tensor”. Các tenxơ không là gì ngoài một dạng chung cho vectơ, hay theo thuật ngữ kỹ thuật máy tính, một ma trận n chiều đơn giản. Dữ liệu dưới mọi hình thức cuối cùng được biểu diễn dưới dạng ma trận số thuần nhất. Vì vậy, nếu dữ liệu ở dạng bảng, nó sẽ là một tenxơ hai chiều trong đó mỗi cột đại diện cho một mẫu huấn luyện và toàn bộ bảng/ ma trận sẽ là m mẫu. Để hiểu điều này tốt hơn, hãy xem hình minh họa trực quan sau đây.

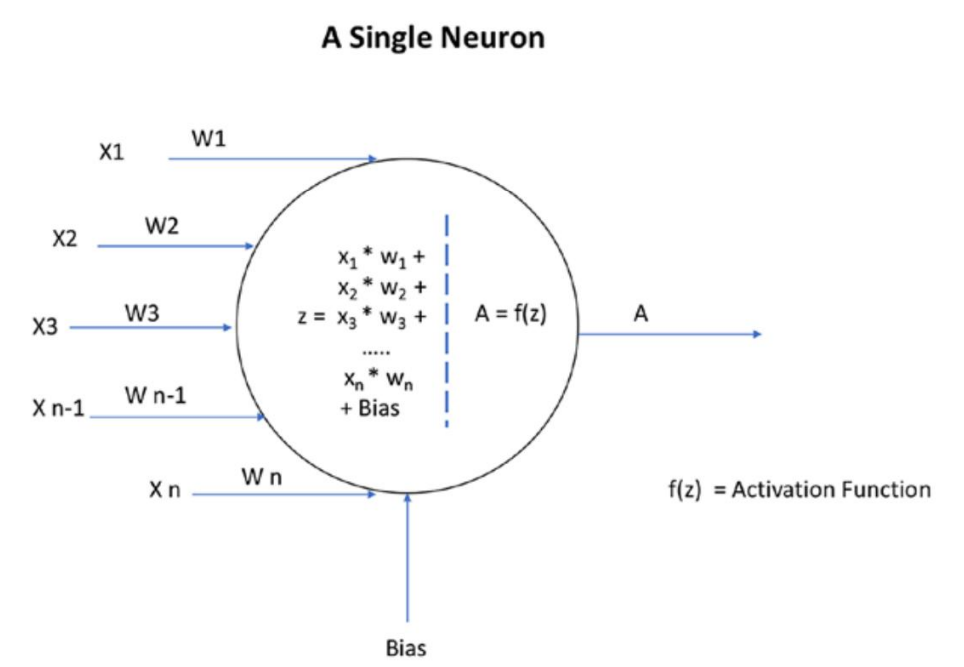


các mô hình DL chỉ có thể giải thích dữ liệu số. Nếu tập dữ liệu có bất kỳ dữ liệu phân loại nào như “giới tính” với các giá trị “nam” và “nữ”, chúng tôi sẽ cần chuyển đổi chúng thành các biến được mã hóa một chiều (nghĩa là chỉ biểu thị các cột có giá trị 0 hoặc 1, trong đó 0 sẽ biểu thị “nam” và 1 sẽ đại diện cho “nữ” hoặc ngược lại)

Dữ liệu hình ảnh cũng cần được chuyển đổi thành một tensor n chiều. Một hình ảnh được lưu trữ trong dữ liệu dưới dạng tenxơ ba chiều trong đó hai chiều xác định các giá trị pixel trên mặt phẳng 2D và chiều thứ ba xác định các giá trị cho các kênh màu RGB. Vì vậy, về cơ bản, một hình ảnh trở thành một tensor ba chiều và n hình ảnh sẽ là một tensor bốn chiều, trong đó chiều thứ tư sẽ xếp chồng một hình ảnh tensor 3D làm mẫu huấn luyện. Do đó, nếu chúng ta có 100 hình ảnh với độ phân giải 512 × 512 pixel, chúng sẽ được biểu diễn dưới dạng tensor 4D có hình dạng 512 × 512 × 3 × 100

nên chuẩn hóa, chuẩn hóa hoặc chia tỷ lệ đầu vào giá trị trước khi đào tạo. Việc chuẩn hóa các giá trị sẽ đưa tất cả các giá trị trong tenxơ đầu vào vào phạm vi 0–1 và quá trình chuẩn hóa sẽ đưa các giá trị vào một phạm vi trong đó giá trị trung bình là 0 và độ lệch chuẩn là 1. Điều này giúp giảm tính toán khi việc học được cải thiện bởi một lợi nhuận lớn và hiệu suất cũng vậy, vì các chức năng kích hoạt (được trình bày ở phần sau) hoạt động phù hợp hơn.

**Neuron**   
Tại cốt lõi của DNN, chúng tôi có các nơ-ron nơi thực hiện tính toán cho một đầu ra. Một nơ-ron nhận một hoặc nhiều đầu vào từ các nơ-ron ở lớp trước. Nếu các nơron ở lớp ẩn đầu tiên, chúng sẽ nhận dữ liệu từ luồng dữ liệu đầu vào. Trong tế bào thần kinh sinh học, tín hiệu điện được đưa ra dưới dạng đầu ra khi nó nhận được đầu vào có ảnh hưởng cao hơn. Để ánh xạ chức năng đó trong nơ-ron toán học, chúng ta cần có một hàm hoạt động dựa trên tổng đầu vào nhân với các trọng số tương ứng (ký hiệu là f(z) trong hình dưới đây) và phản hồi bằng một giá trị phù hợp dựa trên đầu vào. Nếu nhận được đầu vào ảnh hưởng cao hơn, đầu ra sẽ cao hơn và ngược lại. Nó tương tự như tín hiệu kích hoạt (nghĩa là ảnh hưởng cao hơn -> sau đó kích hoạt, nếu không thì hủy kích hoạt). Hàm hoạt động trên dữ liệu đầu vào được tính toán được gọi là hàm kích hoạt.



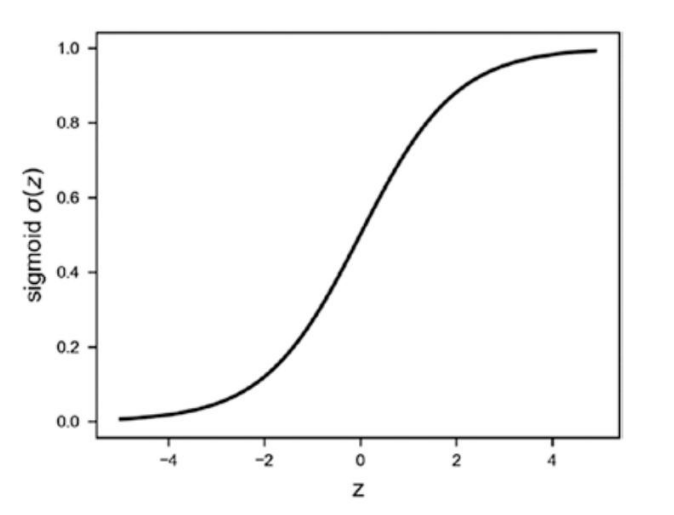
Chức năng kích hoạt

Hàm kích hoạt là hàm lấy đầu vào kết hợp z như minh họa trong hình minh họa trước, áp dụng một hàm trên đó và chuyển giá trị đầu ra, do đó cố gắng bắt chước hàm kích hoạt/hủy kích hoạt.

Có nhiều lựa chọn có sẵn để sử dụng làm chức năng kích hoạt. Chúng ta có thể nhập cái này vào Python chỉ bằng lệnh nhập: Một hàm sigmoid được định nghĩa là Những cái phổ biến nhất là hàm sigmoid và ReLU (đơn vị tuyến tính được chỉnh lưu).

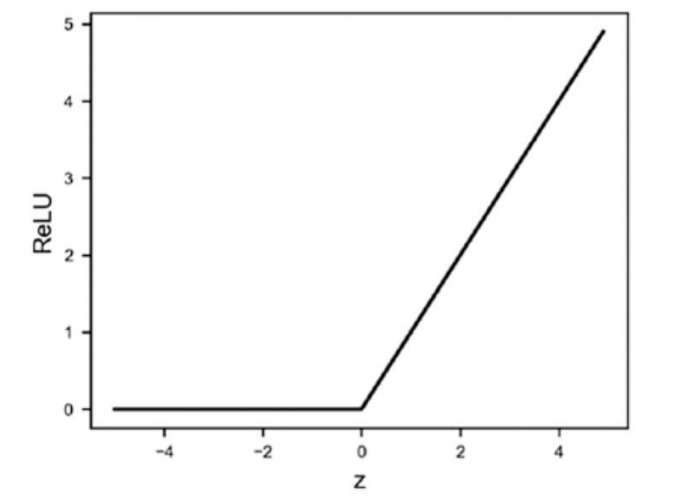
**Sigmoid**

Chúng ta có thể nhập cái này vào Python chỉ bằng lệnh nhập: keras.activations.sigmoid



**ReLU**

ReLU sử dụng hàm f(z) = max(0,z), có nghĩa là nếu đầu ra là dương thì nó sẽ cho ra cùng một giá trị, ngược lại nó sẽ cho ra 0. Phạm vi đầu ra của hàm được hiển thị trong hình ảnh sau



**Model**   
Cấu trúc tổng thể của DNN được phát triển bằng cách sử dụng đối tượng mô hình trong Keras. Điều này cung cấp một cách đơn giản để tạo một chồng các lớp bằng cách lần lượt thêm các lớp mới.

from keras.models import Sequential  
from keras.layers import Dense, Activation  
model = Sequential()  
model.add(Dense(10, input\_dim=15))  
model.add(Activation('relu'))

**Layers**   
Một lớp trong DNN được định nghĩa là một nhóm các nơ-ron hoặc một nhóm được phân tách hợp lý trong cấu trúc mạng phân cấp.

lớp lõi

Có một vài lớp quan trọng mà chúng tôi sẽ sử dụng trong hầu hết các trường hợp sử dụng.

**Dense Layer**   
là lớp DNN thông thường kết nối mọi nơ-ron trong lớp được xác định với mọi nơ-ron trong lớp trước đó. Chẳng hạn, nếu Lớp 1 có 5 nơ-ron và Lớp 2 (lớp dày đặc) có 3 nơ-ron, thì tổng số kết nối giữa Lớp 1 và Lớp 2 sẽ là 15 (5 × 3).

model = Sequential()  
model.add(Dense(5,input\_dim=10,activation = "sigmoid"))  
model.add(Dense(1,activation = "sigmoid"))   
**Dropout Layer**   
giúp giảm tình trạng thừa bằng cách đưa các khả năng chính quy hóa và khái quát hóa vào mô hình. c loại bỏ một số nơ-ron hoặc đặt chúng thành 0 và giảm tính toán trong quá trình đào tạo. Quá trình tùy ý loại bỏ nơ-ron hoạt động khá tốt trong việc giảm trang bị quá mức.

keras.layers.Dropout(rate, noise\_shape=None, seed=None)

**Loss**   
Hàm mất mát là số liệu giúp mạng hiểu liệu nó có đang học đúng hướng hay không. Nói một cách đơn giản về hàm mất mát, hãy coi nó như điểm kiểm tra mà bạn đạt được trong một kỳ thi. Giả sử bạn đã xuất hiện trong một số bài kiểm tra về cùng một chủ đề: bạn sẽ sử dụng thước đo nào để hiểu hiệu suất của mình trong mỗi bài kiểm tra? Rõ ràng là điểm thi. Giả sử bạn đạt 56, 60, 78, 90 và 96 trên 100 trong năm bài kiểm tra ngôn ngữ liên tiếp.

Bạn sẽ thấy rõ ràng rằng điểm kiểm tra được cải thiện là dấu hiệu cho thấy bạn đang thể hiện tốt như thế nào. Nếu điểm kiểm tra ngày càng giảm, thì phán quyết sẽ là hiệu suất của bạn đang giảm và bạn cần phải thay đổi phương pháp hoặc tài liệu học tập của mình để cải thiện.

Tương tự, làm thế nào để một mạng hiểu liệu nó có đang cải thiện quá trình học tập của mình trong mỗi lần lặp lại hay không? Nó sử dụng hàm mất mát, tương tự như điểm kiểm tra. Hàm mất mát về cơ bản đo lường sự mất mát từ mục tiêu. Giả sử bạn đang phát triển một mô hình để dự đoán liệu một học sinh sẽ đậu hay rớt và cơ hội đậu hay rớt được xác định bằng xác suất. Vì vậy, 1 có nghĩa là anh ta sẽ vượt qua với 100% chắc chắn và 0 có nghĩa là anh ta chắc chắn sẽ trượt

Mô hình học từ dữ liệu và dự đoán học sinh đạt điểm 0,87. Vì vậy, tổn thất thực tế ở đây sẽ là 1,00 – 0,87 = 0,13. Nếu nó lặp lại bài tập với một số cập nhật tham số để cải thiện và hiện đạt mức lỗ 0,40, thì nó sẽ hiểu rằng những thay đổi mà nó đã thực hiện không giúp mạng học một cách thích hợp. Ngoài ra, mức giảm mới là 0,05 sẽ chỉ ra rằng các bản cập nhật hoặc thay đổi từ quá trình học đang đi đúng hướng.

**Optimizers**   
Phần quan trọng nhất của đào tạo mô hình là trình tối ưu hóa. Cho đến thời điểm này, chúng tôi đã giải quyết quá trình đưa ra phản hồi cho mô hình thông qua một thuật toán gọi là truyền ngược; đây thực sự là một thuật toán tối ưu hóa.

Một mạng có trọng số ngẫu nhiên và xác định cấu trúc là điểm khởi đầu cho một mô hình. Mô hình có thể đưa ra dự đoán vào thời điểm này, nhưng hầu như nó sẽ không có giá trị. Mạng lấy một mẫu đào tạo và sử dụng các giá trị của nó làm đầu vào cho các nơ-ron ở lớp đầu tiên, sau đó tạo ra đầu ra với hàm kích hoạt đã xác định. Đầu ra bây giờ trở thành đầu vào cho lớp tiếp theo, v.v. Đầu ra của lớp cuối cùng sẽ là dự đoán cho mẫu đào tạo. Đây là nơi chức năng mất đi vào hình ảnh. Hàm mất mát giúp mạng hiểu mức độ tốt hay kém của bộ trọng số hiện tại đã thực hiện trên mẫu đào tạo. Bước tiếp theo của mô hình là giảm tổn thất. Làm thế nào để nó biết những bước hoặc cập nhật mà nó nên thực hiện trên các trọng số để giảm tổn thất? Chức năng tối ưu hóa giúp nó hiểu bước này. Hàm tối ưu hóa là một thuật toán toán học sử dụng đạo hàm, đạo hàm riêng và quy tắc chuỗi trong phép tính để hiểu mức độ thay đổi mà mạng sẽ thấy trong hàm mất mát bằng cách tạo ra một thay đổi nhỏ về trọng lượng của các nơ-ron. Sự thay đổi trong hàm mất mát, có thể là tăng hoặc giảm, sẽ giúp trong việc xác định hướng thay đổi cần thiết trong trọng lượng của kết nối. Việc tính toán một mẫu huấn luyện từ lớp đầu vào đến lớp đầu ra được gọi là vượt qua. Thông thường, việc đào tạo sẽ được thực hiện theo đợt do hạn chế về bộ nhớ trong hệ thống. Một lô là một tập hợp các mẫu đào tạo từ toàn bộ đầu vào. Mạng cập nhật trọng số của nó sau khi xử lý tất cả các mẫu trong một đợt. Điều này được gọi là một phép lặp (nghĩa là tất cả các mẫu trong một đợt đều vượt qua thành công, sau đó là cập nhật trọng số trong mạng). Việc tính toán tất cả các mẫu đào tạo được cung cấp trong dữ liệu đầu vào với các cập nhật trọng số theo từng đợt được gọi là một kỷ nguyên. Trong mỗi lần lặp lại, mạng tận dụng chức năng tối ưu hóa để thực hiện một thay đổi nhỏ đối với các tham số trọng số của nó (được khởi tạo ngẫu nhiên lúc đầu) để cải thiện dự đoán kết thúc bằng cách giảm hàm mất mát. Từng bước một, với một số lần lặp lại và sau đó là một số kỷ nguyên, mạng cập nhật trọng số của nó và học cách đưa ra dự đoán chính xác cho các mẫu đào tạo đã cho

**Metrics**   
Tương tự như hàm mất mát, chúng tôi cũng xác định số liệu cho mô hình trong Keras. Nói một cách đơn giản, các số liệu có thể được hiểu là chức năng được sử dụng để đánh giá hiệu suất của mô hình trên một tập dữ liệu vô hình khác, còn được gọi là tập dữ liệu xác thực. Sự khác biệt duy nhất giữa số liệu và hàm mất mát là kết quả từ số liệu không được sử dụng để đào tạo mô hình liên quan đến tối ưu hóa. Chúng chỉ được sử dụng để xác nhận kết quả kiểm tra trong khi báo cáo.

**Cấu hình**

Để biên dịch một mô hình, chúng ta cần cung cấp ba tham số: hàm tối ưu hóa, hàm mất mát và chỉ số để mô hình đo lường hiệu suất trên tập dữ liệu xác thực

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation

# Generate dummy training dataset

np.random.seed(2018)

x\_train = np.random.random((6000,10))

y\_train = np.random.randint(2, size=(6000, 1))

# Generate dummy validation dataset

x\_val = np.random.random((2000,10))

y\_val = np.random.randint(2, size=(2000, 1))

# Generate dummy test dataset

x\_test = np.random.random((2000,10))

y\_test = np.random.randint(2, size=(2000, 1))

#Define the model architecture

model = Sequential()

model.add(Dense(64, input\_dim=10,activation = "relu")) #Layer 1

model.add(Dense(32,activation = "relu")) #Layer 2

model.add(Dense(16,activation = "relu")) #Layer 3

model.add(Dense(8,activation = "relu")) #Layer 4

model.add(Dense(4,activation = "relu")) #Layer 5

model.add(Dense(1,activation = "sigmoid")) #Output Layer

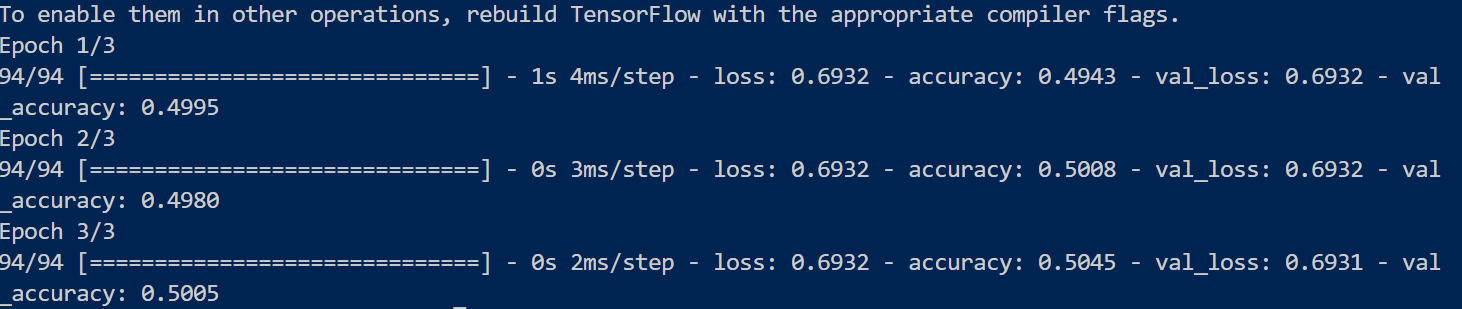
#Configure the model

model.compile(optimizer='Adam',loss='binary\_crossentropy',metrics=['accuracy'])

#Train the model

model.fit(x\_train, y\_train, batch\_size=64, epochs=3,

validation\_data=(x\_val,y\_val))



Chúng ta có thể thấy rằng sau mỗi kỷ nguyên, mô hình in ra độ chính xác và độ mất mát đào tạo trung bình cũng như độ chính xác và độ mất xác thực. Chúng ta có thể sử dụng các kết quả trung gian này để đưa ra đánh giá về hiệu suất của mô hình. Chúng tôi đã không thảo luận về hiệu suất mô hình cho đến nay. Hiểu mức độ hiệu quả của mô hình của bạn trên tập dữ liệu thử nghiệm vô hình là điều tối quan trọng. Trong hầu hết các trường hợp sử dụng DL lớn, chúng tôi sẽ có một số kỷ nguyên để đào tạo. Đó là một phương pháp hay để theo dõi hiệu suất của mô hình với các chỉ số mà chúng tôi đã định cấu hình theo các khoảng thời gian để xem kết quả sau một vài kỷ nguyên. Nếu kết quả dường như không có lợi cho bạn, bạn nên dừng đào tạo và xem lại cấu hình và kiến trúc mô hình.

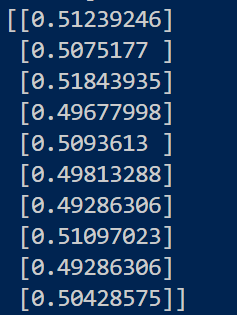
**Đánh giá mô hình**

Keras cung cấp đối tượng mô hình được trang bị tính năng đánh giá mô hình sẵn có và một chức năng khác để dự đoán kết quả từ tập dữ liệu thử nghiệm. Chúng ta hãy xem xét cả hai điều này bằng cách sử dụng mô hình được đào tạo và dữ liệu thử nghiệm giả được tạo trong ví dụ trước.

print(model.evaluate(x\_test,y\_test))

Ngoài ra, bạn có thể sử dụng phương pháp dự đoán của mô hình và tận dụng các dự đoán thực tế có thể là xác suất (đối với trường hợp sử dụng này, do phân loại nhị phân):

pred = model.predict(x\_test)

pred[:10]

**Thêm ví dụ nữa**

chèn vào các thư viện cần thiết

from keras.models import Sequential

from keras.layers import Dense

import numpy as np

tạo bộ dữ liệu giả

train\_data = np.random.random((1000, 3))

test\_data  = np.random.random((500, 3))

tạo nhãn

labels = np.random.randint(2, size=(1000, 1))

model = Sequential()

# noron đầu tiên 5 node, input đầu vào là 3, hàm relu

model.add(Dense(5, input\_dim=3, activation='relu'))

lớp thứ 2 có 4 node

model.add(Dense(4, activation='relu'))

lớp cuối có 1 node ra

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam',metrics=['accuracy'])

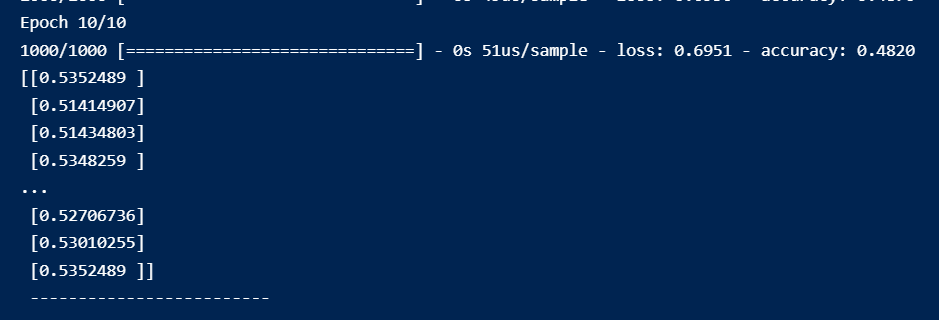
đào tạo tập train và gắn nhãn, epochs=10, batch\_size bằng 32

model.fit(train\_data, labels, epochs=10, batch\_size=32)

đưa ra dự đoán từ tập train

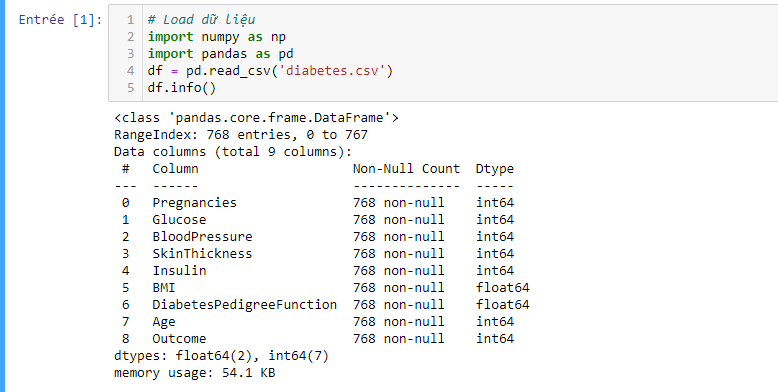
predictions = model.predict(test\_data)

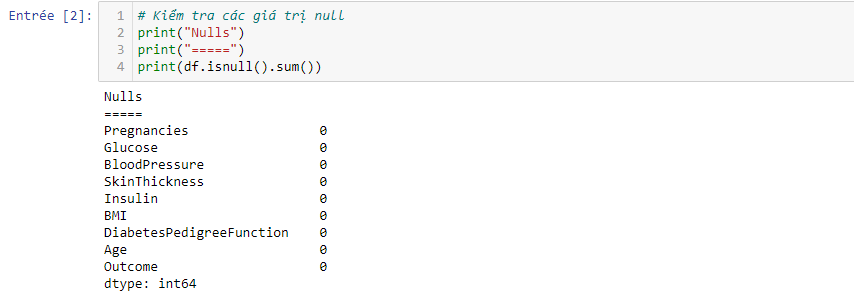
print(predictions)

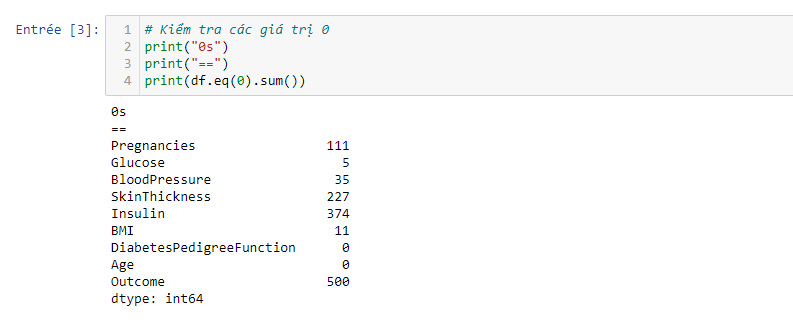
kết quả

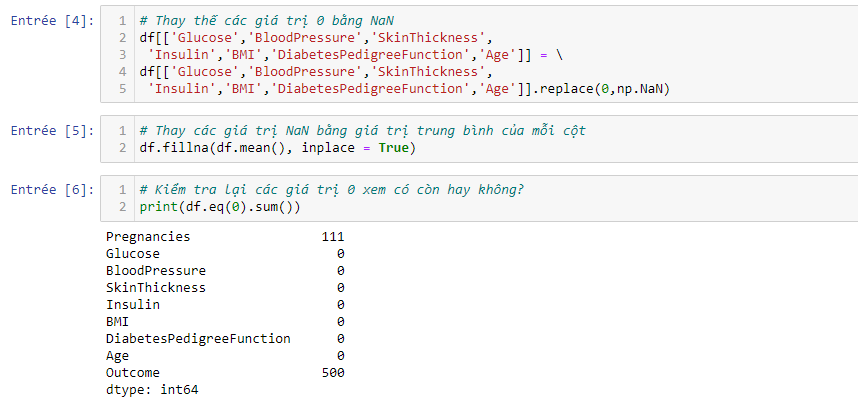
# **PHẦN III: Bài tập áp dụng**

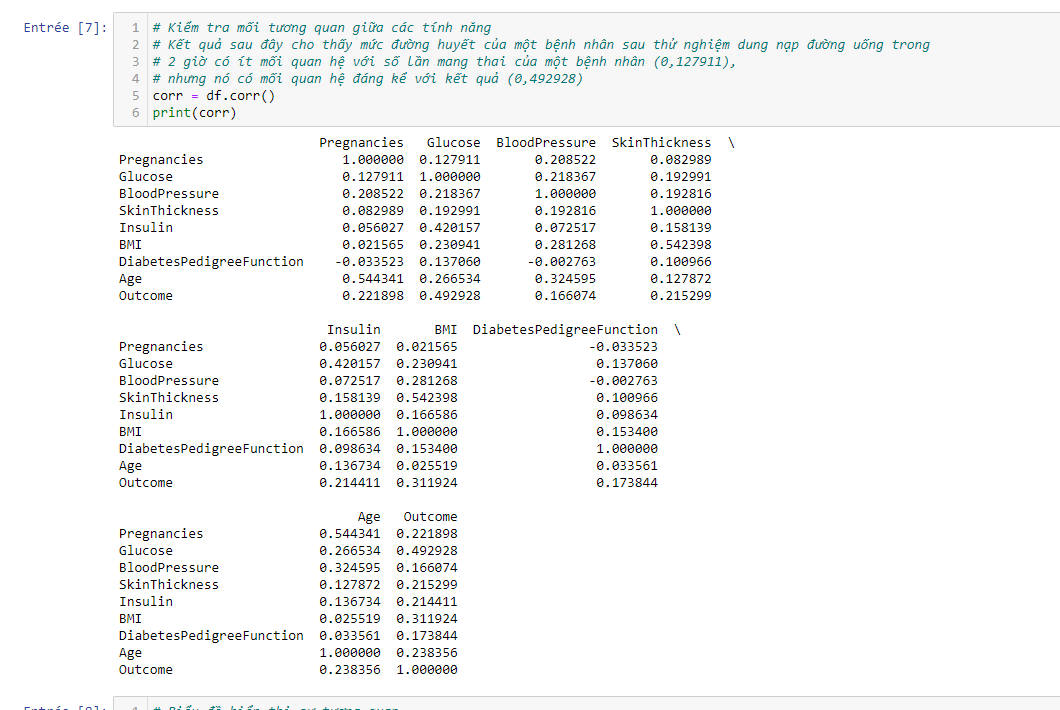
## **Bài 1: Dự đoán bệnh tiểu đường**

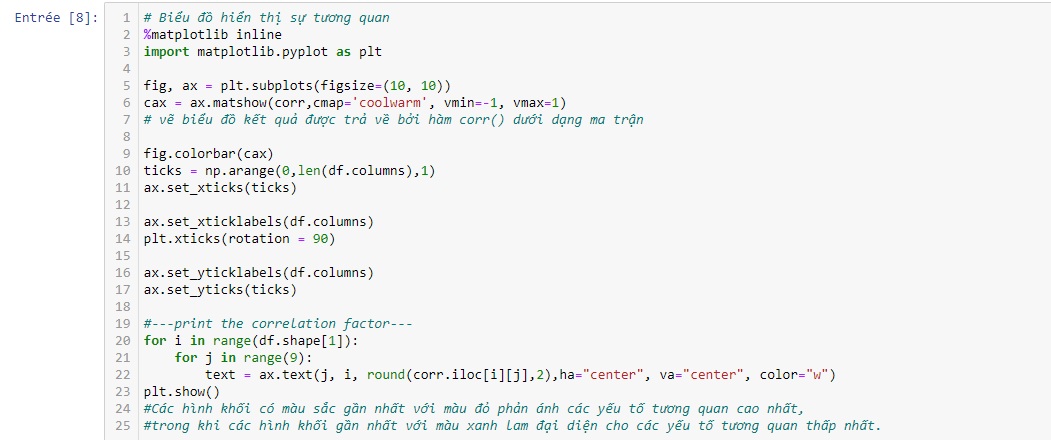
****

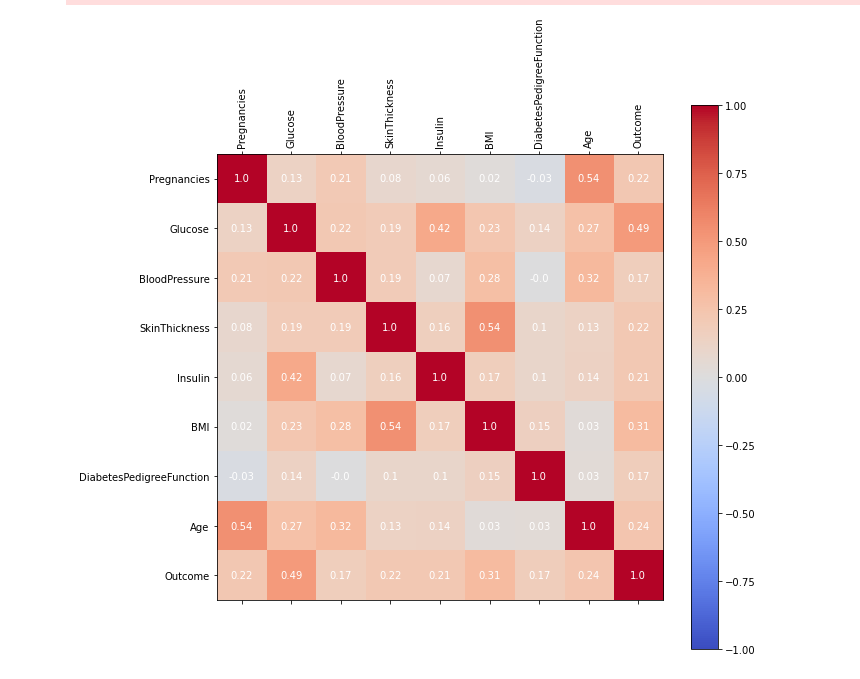
****

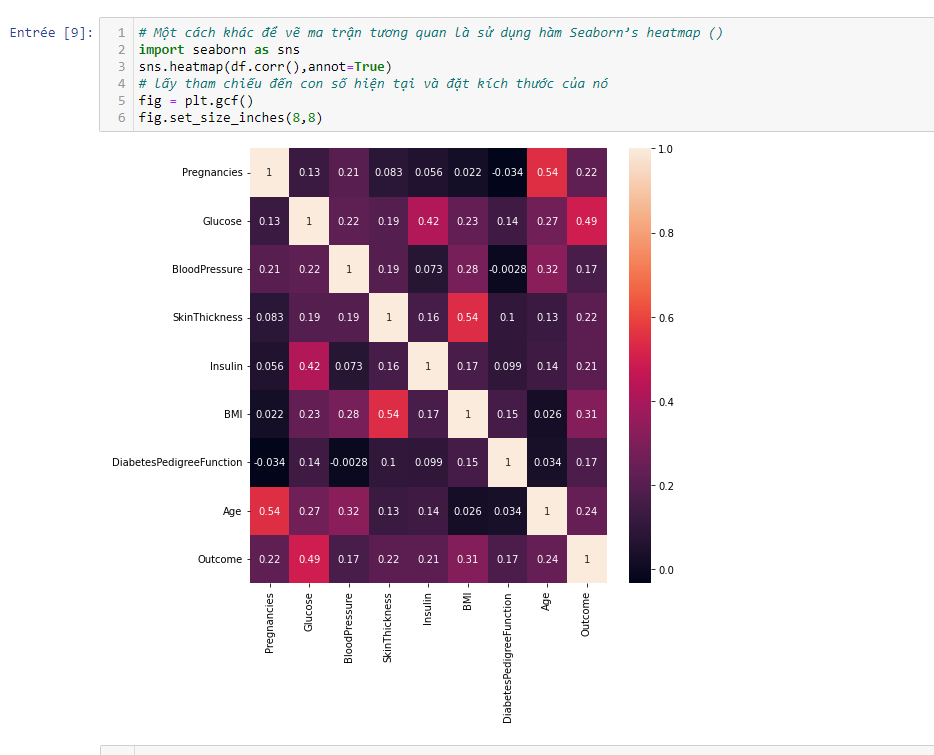
****

****

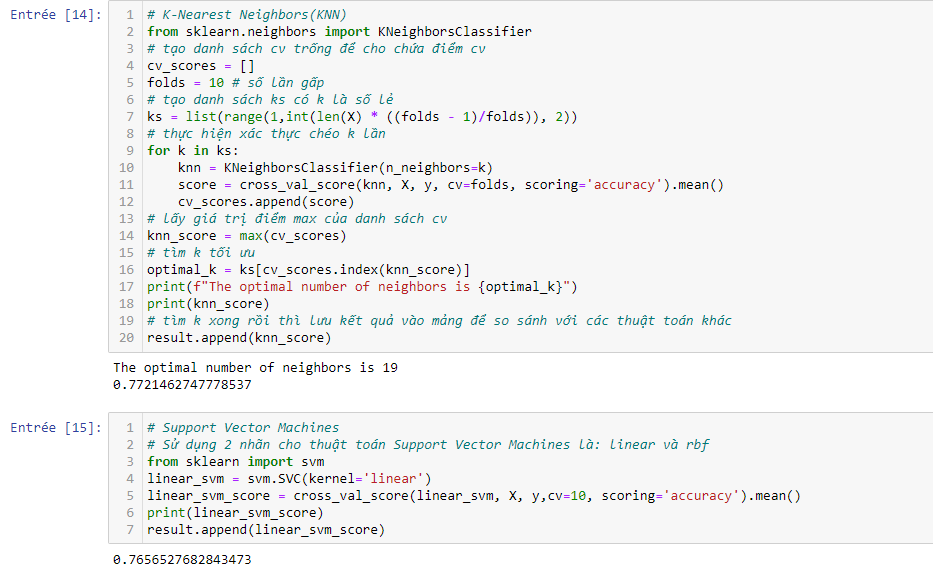
****

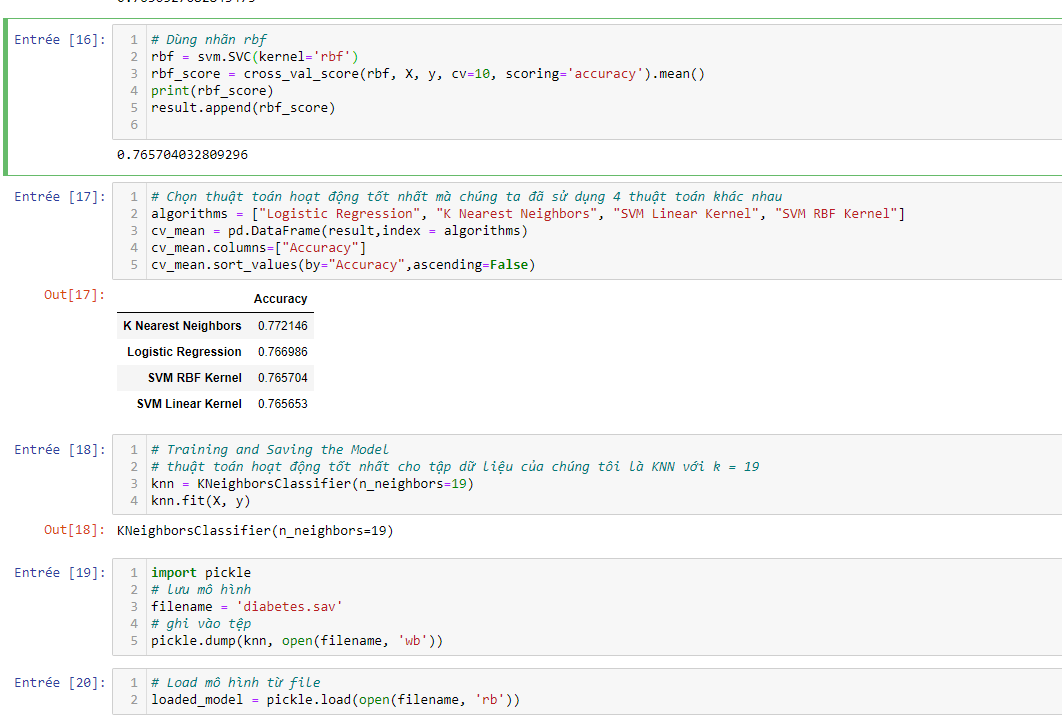
****

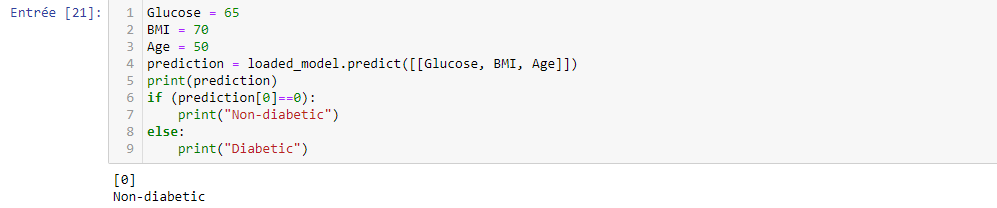
****

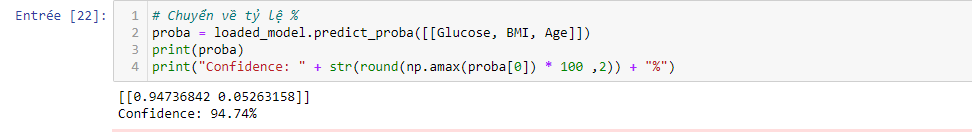
****

****

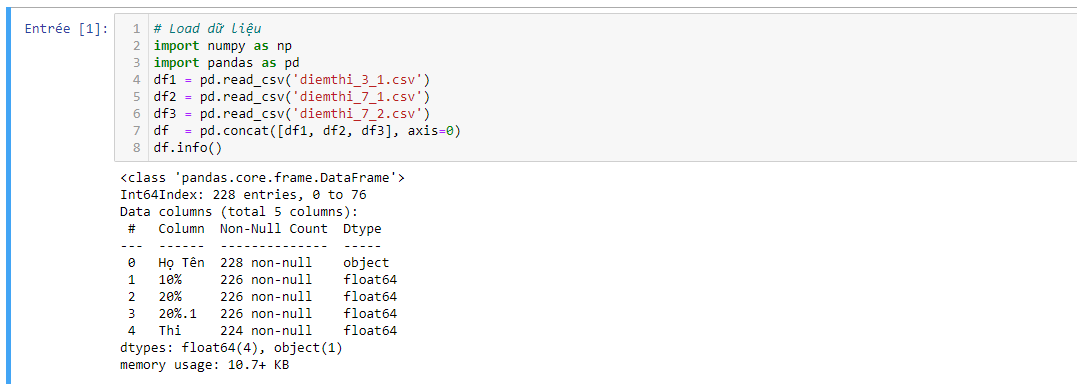
****

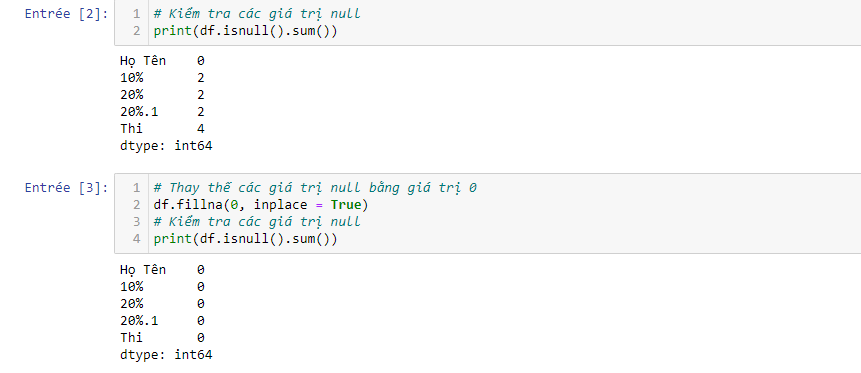
****

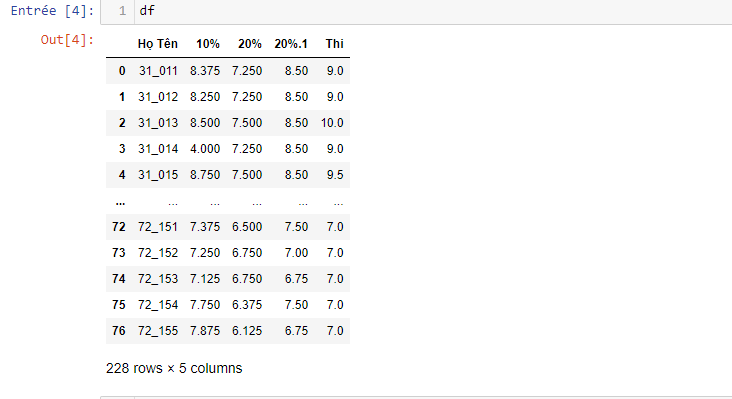
****

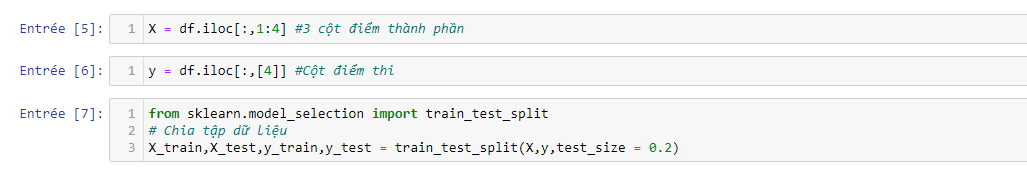
****

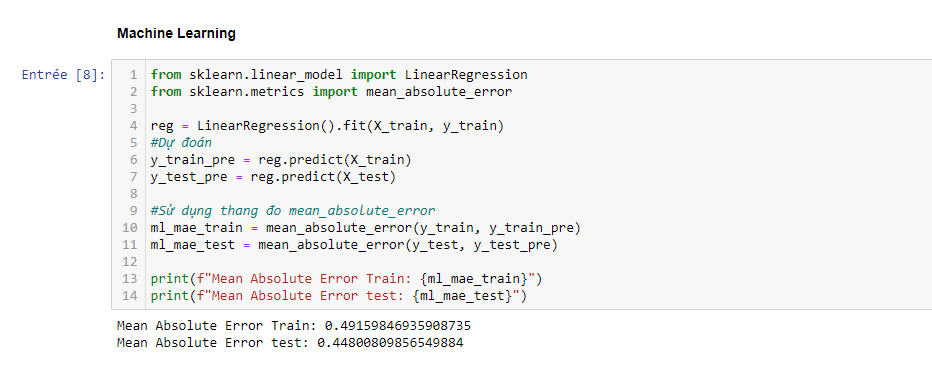
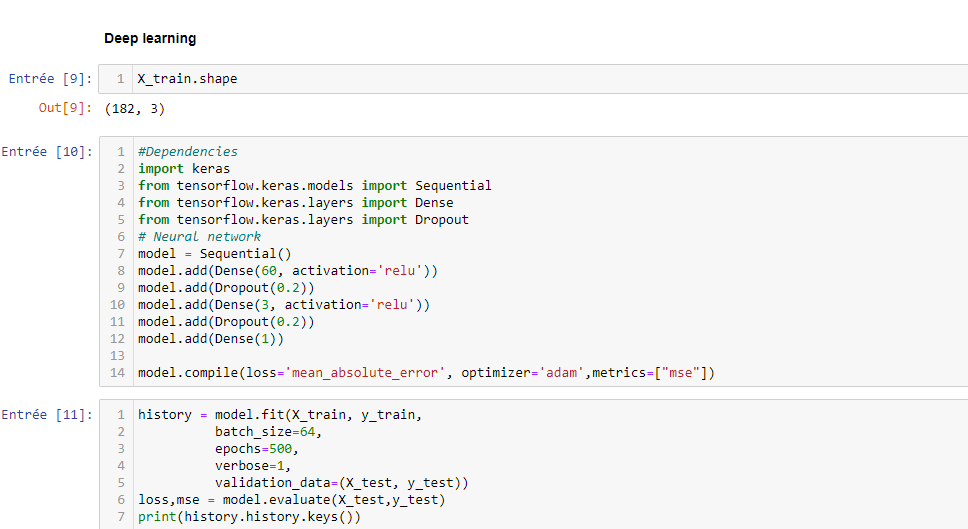
## **Bài 2: Dự đoán điểm thi.**

****

****

****

****

**** **** 