

Họ và tên: Quách Xuân Phúc

MSV: B20DCCN513

## Bài tập 2

### 2.1 Split a set into training and testing sets:

Phân tách thử nghiệm đào tạo được sử dụng để ước tính hiệu suất của các thuật toán học máy có thể áp dụng cho Thuật toán/Ứng dụng dựa trên dự đoán. Phương pháp này là một quy trình thực hiện nhanh chóng và dễ dàng để chúng ta có thể so sánh kết quả mô hình học máy của chính mình với kết quả của máy. Theo mặc định, Tập kiểm tra được chia thành 30% dữ liệu thực tế và tập huấn luyện được chia thành 70% dữ liệu thực tế.

Chúng ta cần chia tập dữ liệu thành tập huấn luyện và tập kiểm tra để đánh giá mô hình học máy của chúng ta hoạt động tốt như thế nào. Tập đoàn tàu được sử dụng để phù hợp với mô hình và số liệu thống kê của tập đoàn tàu đã được biết. Tập thứ hai được gọi là tập dữ liệu thử nghiệm, tập này chỉ được sử dụng để dự đoán.

#### \* Dataset Splitting:

Thư viện scikit-learn cung cấp cho chúng ta mô-đun `model_selection` trong đó chúng ta có hàm phân tách `train_test_split()`.

```
train_test_split(*arrays, test_size=None, train_size=None,
random_state=None, shuffle=True, stratify=None)
```

#### \* Tầm quan trọng của việc chia tách dữ liệu:

Học máy có giám sát là việc tạo ra các mô hình ánh xạ chính xác các đầu vào nhất định (biến độc lập hoặc yếu tố dự đoán) với đầu ra nhất định (biến phụ thuộc hoặc phản hồi).

Cách bạn đo lường độ chính xác của mô hình tùy thuộc vào loại vấn đề bạn đang cố gắng giải quyết. Trong phân tích hồi quy, bạn thường sử dụng hệ số xác định, sai số bình phương trung bình gốc, sai số tuyệt đối trung bình hoặc các đại lượng tương tự. Đối với các bài toán phân loại, bạn thường áp dụng độ chính xác, độ chính xác, khả năng thu hồi, điểm F1 và các chỉ số liên quan.

Các giá trị số được chấp nhận để đo độ chính xác sẽ khác nhau tùy theo từng trường. Bạn có thể tìm thấy lời giải thích chi tiết từ Thống kê của Jim, Quora và nhiều tài nguyên khác.

Điều quan trọng nhất cần hiểu là bạn thường cần đánh giá khách quan để sử dụng đúng các biện pháp này, đánh giá hiệu suất dự đoán của mô hình của bạn và xác thực mô hình.

Điều này có nghĩa là bạn không thể đánh giá hiệu suất dự đoán của một mô hình với cùng dữ liệu bạn đã sử dụng để đào tạo. Bạn cần đánh giá mô hình bằng dữ liệu mới mà mô

hình chưa từng thấy trước đó. Bạn có thể thực hiện điều đó bằng cách chia nhỏ tập dữ liệu của mình trước khi sử dụng.

\* Đào tạo, xác nhận và kiểm tra:

Việc chia tách tập dữ liệu của bạn là điều cần thiết để đánh giá khách quan về hiệu suất dự đoán. Trong hầu hết các trường hợp, việc chia ngẫu nhiên tập dữ liệu của bạn thành ba tập hợp con là đủ:

Tập huấn luyện được áp dụng để huấn luyện hoặc điều chỉnh mô hình của bạn. Ví dụ: bạn sử dụng tập huấn luyện để tìm trọng số hoặc hệ số tối ưu cho hồi quy tuyến tính, hồi quy logistic hoặc mạng thần kinh.

Bộ xác thực được sử dụng để đánh giá mô hình không thiên vị trong quá trình điều chỉnh siêu tham số. Ví dụ: khi bạn muốn tìm số lượng nơ-ron tối ưu trong mạng nơ-ron hoặc hạt nhân tốt nhất cho máy vector hỗ trợ, bạn thử nghiệm với các giá trị khác nhau. Đối với mỗi cài đặt siêu tham số được xem xét, bạn điều chỉnh mô hình với tập huấn luyện và đánh giá hiệu suất của nó bằng tập xác thực.

Bộ kiểm tra là cần thiết để đánh giá khách quan mô hình cuối cùng. Bạn không nên sử dụng nó để điều chỉnh hoặc xác nhận.

Trong những trường hợp ít phức tạp hơn, khi bạn không phải điều chỉnh siêu tham số, bạn có thể chỉ làm việc với các tập huấn luyện và kiểm tra.

\* Underfitting and Overfitting

Việc chia tập dữ liệu cũng có thể quan trọng để phát hiện xem mô hình của bạn có gặp phải một trong hai vấn đề rất phổ biến hay không, được gọi là Underfitting and Overfitting:

1.Underfitting: thường là hậu quả của việc một mô hình không thể gói gọn các mối quan hệ giữa các dữ liệu. Ví dụ, điều này có thể xảy ra khi cố gắng biểu diễn các mối quan hệ phi tuyến bằng mô hình tuyến tính. Các mô hình không được trang bị đầy đủ có thể sẽ có hiệu suất kém với cả tập huấn luyện và tập kiểm tra.

2.Overfitting: thường diễn ra khi một mô hình có cấu trúc quá phức tạp và tìm hiểu cả mối quan hệ hiện có giữa dữ liệu và nhiễu. Những mô hình như vậy thường có khả năng khái quát hóa kém. Mặc dù chúng hoạt động tốt với dữ liệu huấn luyện nhưng chúng thường mang lại hiệu suất kém với dữ liệu (kiểm tra) không nhìn thấy được.

\* Học máy được giám sát với `train_test_split()`

Bây giờ là lúc xem `train_test_split()` hoạt động khi giải quyết các vấn đề học có giám sát. Bạn sẽ bắt đầu với một bài toán hồi quy nhỏ có thể được giải bằng hồi quy tuyến tính trước khi xem xét một bài toán lớn hơn. Bạn cũng sẽ thấy rằng bạn cũng có thể sử dụng `train_test_split()` để phân loại.

\* Train-Test split evaluation

Phân chia thử nghiệm đào tạo là một kỹ thuật để đánh giá hiệu suất của thuật toán học máy.

Nó có thể được sử dụng cho các vấn đề phân loại hoặc hồi quy và có thể được sử dụng cho bất kỳ thuật toán học có giám sát nào.

Quy trình này bao gồm việc lấy một tập dữ liệu và chia nó thành hai tập hợp con. Tập hợp con đầu tiên được sử dụng để phù hợp với mô hình và được gọi là tập dữ liệu huấn luyện. Tập hợp con thứ hai không được sử dụng để huấn luyện mô hình; thay vào đó, phần tử đầu vào của tập dữ liệu được cung cấp cho mô hình, sau đó các dự đoán được đưa ra và so sánh với các giá trị mong đợi. Tập dữ liệu thứ hai này được gọi là tập dữ liệu thử nghiệm.

Train Dataset: Được sử dụng để phù hợp với mô hình machine learning.

Test Dataset: Được sử dụng để đánh giá mô hình học máy phù hợp.

Mục tiêu là ước tính hiệu suất của mô hình học máy trên dữ liệu mới: dữ liệu không được sử dụng để huấn luyện mô hình.

Đây là cách chúng tôi mong đợi sử dụng mô hình trong thực tế. Cụ thể, để phù hợp với dữ liệu có sẵn với đầu vào và đầu ra đã biết, sau đó đưa ra dự đoán về các ví dụ mới trong tương lai khi chúng ta không có đầu ra hoặc giá trị mục tiêu dự kiến.

Quy trình kiểm tra huấn luyện phù hợp khi có sẵn tập dữ liệu đủ lớn.

\* Khi nào nên sử dụng Train-Test split

Ý tưởng “đủ lớn” là cụ thể cho từng vấn đề về mô hình dự đoán. Điều đó có nghĩa là có đủ dữ liệu để chia tập dữ liệu thành tập dữ liệu huấn luyện và tập dữ liệu kiểm tra, đồng thời mỗi tập dữ liệu huấn luyện và tập dữ liệu kiểm tra đều là sự thể hiện phù hợp của miền vấn đề. Điều này đòi hỏi tập dữ liệu gốc cũng phải là sự thể hiện phù hợp của miền vấn đề.

Một biểu diễn phù hợp của miền vấn đề có nghĩa là có đủ bản ghi để bao gồm tất cả các trường hợp phổ biến và hầu hết các trường hợp không phổ biến trong miền. Điều này có thể có nghĩa là sự kết hợp của các biến đầu vào được quan sát trong thực tế. Nó có thể yêu cầu hàng nghìn, hàng trăm nghìn hoặc hàng triệu ví dụ.

Ngược lại, quy trình train-test không phù hợp khi tập dữ liệu có sẵn nhỏ. Lý do là khi tập dữ liệu được chia thành tập huấn luyện và tập kiểm tra, sẽ không có đủ dữ liệu trong tập dữ liệu huấn luyện để mô hình học cách ánh xạ đầu vào thành đầu ra một cách hiệu quả. Cũng sẽ không có đủ dữ liệu trong bộ thử nghiệm để đánh giá hiệu quả hiệu suất của mô hình. Hiệu suất ước tính có thể quá lạc quan (tốt) hoặc quá bi quan (xấu).

Nếu bạn không có đủ dữ liệu thì quy trình đánh giá mô hình thay thế phù hợp sẽ là quy trình xác thực chéo k-Fold.

Ngoài kích thước tập dữ liệu, một lý do khác để sử dụng quy trình đánh giá phân tách thử nghiệm huấn luyện là hiệu quả tính toán.

Một số mô hình rất tốn kém để đào tạo và trong trường hợp đó, việc đánh giá lặp đi lặp lại được sử dụng trong các quy trình khác là khó thực hiện được. Một ví dụ có thể là các mô hình mạng lưới thần kinh sâu. Trong trường hợp này, quy trình thử tàu thường được sử dụng.

Ngoài ra, một dự án có thể có một mô hình hiệu quả và một tập dữ liệu khổng lồ, mặc dù có thể yêu cầu ước tính hiệu suất mô hình một cách nhanh chóng. Một lần nữa, thủ tục phân chia thử nghiệm tàu được tiếp cận trong tình huống này.

Các mẫu từ tập dữ liệu huấn luyện ban đầu được chia thành hai tập con bằng cách sử dụng lựa chọn ngẫu nhiên. Điều này nhằm đảm bảo rằng tập dữ liệu huấn luyện và tập dữ liệu kiểm tra đại diện cho tập dữ liệu gốc.

#### \* Cách định cấu hình Train-Test Split

Quy trình này có một tham số cấu hình chính, đó là kích thước của tập hợp thử nghiệm và đoàn tàu. Điều này thường được biểu thị dưới dạng phần trăm từ 0 đến 1 đối với tập dữ liệu huấn luyện hoặc tập dữ liệu thử nghiệm. Ví dụ: tập huấn luyện có kích thước 0,67 (67 phần trăm) có nghĩa là phần trăm còn lại 0,33 (33 phần trăm) được gán cho tập kiểm tra.

Không có tỷ lệ phân chia tối ưu.

Bạn phải chọn tỷ lệ phân chia đáp ứng các mục tiêu của dự án với những cân nhắc bao gồm:

Chi phí tính toán trong quá trình huấn luyện mô hình.

Chi phí tính toán trong việc đánh giá mô hình.

Tính đại diện của tập huấn luyện.

Kiểm tra tính đại diện của tập hợp.

Tuy nhiên, tỷ lệ phân chia phổ biến bao gồm:

Đào tạo: 80%, Kiểm tra: 20%

Tàu: 67%, Kiểm tra: 33

Tàu: 50%, Kiểm tra: 50%

Bây giờ chúng ta đã quen với quy trình đánh giá mô hình phân tách thử nghiệm tàu, hãy xem cách chúng ta có thể sử dụng quy trình này trong Python.

\* Ví dụ:

1.

```

# import packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# importing data
df = pd.read_csv('headbrain1.csv')

# head of the data
print(df.head())

X= df['Head Size(cm^3)']
y=df['Brain Weight(grams)']

# using the train test split function
X_train, X_test,
y_train, y_test = train_test_split(X,y ,
                                    random_state=104,
                                    test_size=0.25,
                                    shuffle=True)

# printing out train and test sets

print('X_train : ')
print(X_train.head())
print('')
print('X_test : ')
print(X_test.head())
print('')
print('y_train : ')
print(y_train.head())
print('')
print('y_test : ')
print(y_test.head())

```

#### Output:

```

      Head Size(cm^3)  Brain Weight(grams)
0             4512             1530
1             3738             1297
2             4261             1335
3             3777             1282
4             4177             1590
X_train :
99      3478
52      4270
184     3479
139     3171
107     3399
Name: Head Size(cm^3), dtype: int64
(177,)

X_test :
66      3415
113     3594
135     3436
227     4204
68      4430
Name: Head Size(cm^3), dtype: int64
(60,)

```

```

y_train :
99      1270
52      1335
184     1160
139     1127
107     1226
Name: Brain Weight(grams), dtype: int64
(177,)

y_test :
66      1310
113     1290
135     1235
227     1380
68      1510
Name: Brain Weight(grams), dtype: int64
(60,)

```

2.

```

# import packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# importing data
df = pd.read_csv('headbrain1.csv')
print(df.shape)

# head of the data
print(df.head())

X= df['Head Size(cm^3)']
y=df['Brain Weight(grams)']

# using the train test split function
X_train, X_test, y_train,
y_test = train_test_split(X,y ,
                          random_state=104,
                          train_size=0.8, shuffle=True)

# printing out train and test sets
print('X_train : ')
print(X_train.head())
print(X_train.shape)
print('')
print('X_test : ')
print(X_test.head())
print(X_test.shape)
print('')
print('y_train : ')
print(y_train.head())
print(y_train.shape)
print('')
print('y_test : ')
print(y_test.head())
print(y_test.shape)

```

## Output:

```
(237, 2)
   Head Size(cm^3)  Brain Weight(grams)
0              4512              1530
1              3738              1297
2              4261              1335
3              3777              1282
4              4177              1590
```

X\_train :

```
110    3695
164    3497
58     3935
199    3297
182    4005
```

Name: Head Size(cm^3), dtype: int64

(189,)

X\_test :

```
66     3415
113    3594
135    3436
227    4204
68     4430
```

Name: Head Size(cm^3), dtype: int64

(48,)

y\_train :

```
110    1310
164    1280
58     1330
199    1220
182    1280
```

Name: Brain Weight(grams), dtype: int64

(189,)

y\_test :

```
66     1310
113    1290
135    1235
227    1380
68     1510
```

Name: Brain Weight(grams), dtype: int64

(48,)

3.

```
# demonstrate that the train-test split procedure is repeatable
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_blobs(n_samples=100)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# summarize first 5 rows
print(X_train[:5, :])
# split again, and we should see the same split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# summarize first 5 rows
print(X_train[:5, :])
```

```
[[ -6.64863704  1.76866075]
 [ -7.6394962   0.60046715]
 [  0.38082536 -3.20107332]
 [  9.04448027 -7.36013098]
 [  0.52776559 -5.15858726]]
[[ -6.64863704  1.76866075]
 [ -7.6394962   0.60046715]
 [  0.38082536 -3.20107332]
 [  9.04448027 -7.36013098]
 [  0.52776559 -5.15858726]]
```

4.

```
# split imbalanced dataset into train and test sets without stratification
from collections import Counter
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_classification(n_samples=100, weights=[0.94], flip_y=0, random_state=1)
print(Counter(y))
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=1)
print(Counter(y_train))
print(Counter(y_test))
```

```
Counter({0: 94, 1: 6})
Counter({0: 45, 1: 5})
Counter({0: 49, 1: 1})
```



5.

```
# train-test split evaluation random forest on the sonar dataset
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/sonar.csv'
dataframe = read_csv(url, header=None)
data = dataframe.values
# split into inputs and outputs
X, y = data[:, :-1], data[:, -1]
print(X.shape, y.shape)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# fit the model
model = RandomForestClassifier(random_state=1)
model.fit(X_train, y_train)
# make predictions
yhat = model.predict(X_test)
# evaluate predictions
acc = accuracy_score(y_test, yhat)
print('Accuracy: %.3f' % acc)
```

```
(208, 60) (208,)
(139, 60) (69, 60) (139,) (69,)
Accuracy: 0.783
```

## 2.2 Clean data

### \* Data Cleaning là gì?

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.

Data Cleaning không chỉ đơn thuần là loại bỏ dữ liệu, mà còn bao gồm sửa lỗi cú pháp và chính tả, sửa đổi các lỗi như thiếu mã, trường trống, xác định các điểm dữ liệu trùng lặp và chuẩn hóa tập dữ liệu. Data Cleaning đóng một phần quan trọng trong việc phát triển các câu trả lời đáng tin cậy và trong quá trình phân tích. Đồng thời được coi là một tính năng cơ bản của kiến thức cơ bản về khoa học dữ liệu. Động cơ của dịch vụ Data Cleaning là xây dựng các bộ dữ liệu thống nhất và được tiêu chuẩn hóa. Nó cho

phép các công cụ phân tích dữ liệu và thông tin kinh doanh dễ dàng truy cập và nhận thức dữ liệu chính xác cho từng vấn đề.

### **\* Làm thế nào để clean Data?**

Một công cụ Data Cleaning sẽ thay đổi hầu hết các khía cạnh của chương trình Data Cleaning chung, nhưng công cụ Data Cleaning này chỉ là một phần của biện pháp khắc phục liên tục để làm sạch dữ liệu. Sơ lược về các bước Data Cleaning như sau:

- Xác định các trường dữ liệu quan trọng: Xác định loại trường dữ liệu nào là quan trọng đối với dự án dự định.
- Thu thập dữ liệu: Dữ liệu có trong các trường dữ liệu được liệt kê ngắn gọn được thu thập, phân loại và tổ chức.
- Loại bỏ các giá trị trùng lặp: Các số liệu trùng lặp được nhận dạng, loại bỏ và không chính xác được giải quyết.
- Giải quyết các giá trị trống: Các công cụ Data Cleaning tìm kiếm và lấp đầy các giá trị còn thiếu đó để hoàn thiện tập dữ liệu và tránh các khoảng trống trong thông tin.
- Quy trình làm sạch tiêu chuẩn: Quy trình Data Cleaning phải được chuẩn hóa theo các thử nghiệm và các phương pháp lặp đi lặp lại đã chứng minh là tạo ra dữ liệu chất lượng, sau này sẽ giúp dễ dàng sao chép và nhất quán. Quy trình và tần suất Data Cleaning phải được chuẩn hóa và việc xem xét dữ liệu thường được sử dụng nhất.
- Xem xét, điều chỉnh, lặp lại: Thời gian cụ thể phải được dành ra hàng tuần hoặc hàng tháng để phân tích cẩn thận các lỗi, phương pháp hoạt động tốt, chỗ để cải thiện, lỗi và trục trặc đang xảy ra.

### **\* Các thành phần của dữ liệu chất lượng**

Việc xác định tiêu chuẩn thông tin đòi hỏi phải xem xét kỹ lưỡng các đặc điểm của nó, sau đó đo lường các đặc điểm đó theo mức độ quan trọng của nó và ứng dụng của chúng trong tổ chức. Năm đặc điểm của dữ liệu chất lượng phải có là:

- Tính hợp lệ: Mức độ phù hợp với các ràng buộc và quy tắc kinh doanh đã xác định mà dữ liệu cung cấp.
- Độ chính xác: Dữ liệu phải có khả năng mô tả các giá trị thực và tốt nhất.
- Tính đầy đủ: Mức độ quen thuộc của tất cả dữ liệu được yêu cầu.
- Tính nhất quán: Tính nhất quán về dữ liệu trong cùng một cơ sở dữ liệu và trên các tập dữ liệu khác nhau.
- Tính đồng nhất: Mức độ dữ liệu được tuân theo cùng một đơn vị đo lường.

### **\* Lợi ích của Data Cleaning**

Việc kiểm tra dữ liệu sạch sẽ và chất lượng cuối cùng chắc chắn sẽ tăng năng suất tổng thể và cho phép thông tin chất lượng cao để ra quyết định nhanh chóng và đúng.

Khi nhiều nguồn dữ liệu đang phát, các lỗi sẽ được loại bỏ để hoạt động trơn tru.

Có ít hoặc không có sai sót làm cho khách hàng vui vẻ, hài lòng và giảm bớt căng thẳng cho nhân viên.

Khả năng lập bản đồ các chức năng khác nhau và dữ liệu của bạn dự định làm gì.

Theo dõi các lỗi và tiêu chuẩn báo cáo cao hơn và xác định chính xác nguồn gốc của lỗi giúp dễ dàng gỡ lỗi dữ liệu sai cho ứng dụng trong tương lai.

Các công cụ Data Cleaning giúp cho hoạt động kinh doanh hiệu quả hơn, đồng thời cho phép ra quyết định nhanh chóng và dễ dàng.

Tăng doanh thu: Hoạt động kinh doanh vận hành linh hoạt và hiệu quả hơn, dẫn đến hiệu suất tốt hơn. Từ đó, giúp tăng trưởng trong tổ chức, cuối cùng dẫn đến tăng doanh thu.

Hiệu quả về chi phí: Làm việc với Database phù hợp cho hoạt động Marketing sẽ giúp tiết kiệm chi phí

Tăng năng suất: Với những địa chỉ liên hệ đã hết hạn hoặc những khách hàng không có tiềm năng, nhân viên đỡ tốn thời gian hơn trong việc liên hệ họ

Danh tiếng: Sự tin tưởng và danh tiếng chắc chắn sẽ tốt đối với các công ty liên quan đến việc chia sẻ dữ liệu với công chúng.

### **\* Tầm quan trọng của Data Cleaning?**

Tạo dữ liệu đáng tin cậy: Khi dữ liệu đã được làm sạch, bạn có thể tin tưởng vào kết quả phân tích dữ liệu hơn. Điều này quan trọng đặc biệt trong việc đưa ra quyết định kinh doanh dựa trên dữ liệu.

Tránh các quyết định sai lầm: Dữ liệu không chính xác hoặc thiếu sót có thể dẫn đến quyết định sai lầm. Ví dụ, nếu dữ liệu bị nhiễu, một dự đoán hoặc phân tích có thể không chính xác và dẫn đến chi phí hoặc rủi ro không mong muốn.

Tối ưu hóa hiệu suất phân tích: Data Cleaning giúp tạo ra dữ liệu chất lượng và đồng nhất, giúp cho quá trình phân tích dữ liệu diễn ra nhanh chóng hơn và dễ dàng hơn.

Thúc đẩy hợp nhất dữ liệu: Trong các tổ chức lớn có nhiều nguồn dữ liệu khác nhau, Data Cleaning giúp hợp nhất dữ liệu từ các nguồn này để tạo ra một tập dữ liệu thống nhất và đáng tin cậy.

Duy trì danh tiếng và sự tin tưởng: Một tập dữ liệu sạch sẽ thể hiện sự chuyên nghiệp và tôn trọng đối với dữ liệu của bạn, làm tăng danh tiếng và sự tin tưởng của khách hàng, đối tác và nhân viên.

Tiết kiệm thời gian và nguồn lực: Data Cleaning có thể tiết kiệm thời gian và nguồn lực của tổ chức bằng cách giảm đi sự cần thiết phải xử lý các vấn đề dữ liệu sau khi chúng đã gây ra tác động tiêu cực.

Việc Data Cleaning là một phần quan trọng của chu trình làm việc với dữ liệu và đóng vai trò quan trọng trong việc đảm bảo dữ liệu chất lượng và đáng tin cậy cho các ứng dụng khoa học dữ liệu và phân tích dữ liệu.

### **\* Các phương pháp Data Cleaning cụ thể?**

Sử dụng ngôn ngữ lập trình và công cụ: Nhiều ngôn ngữ lập trình và công cụ phổ biến đã được phát triển để thực hiện các tác vụ Data Cleaning một cách hiệu quả, chẳng hạn như Python với thư viện pandas và numpy, hoặc công cụ chuyên nghiệp như OpenRefine.

Sử dụng biểu đồ và trực quan hóa: Biểu đồ và trực quan hóa dữ liệu có thể giúp phát hiện các biểu hiện ngoại lai và các mẫu trong tập dữ liệu, giúp trong quá trình Data Cleaning.

Sử dụng kỹ thuật Machine Learning: Một số mô hình học máy có thể được sử dụng để dự đoán giá trị bị thiếu hoặc xác định các giá trị bất thường trong dữ liệu.

Quy trình kiểm tra dữ liệu trước và sau Data Cleaning:

Kiểm tra dữ liệu trước khi làm sạch: Trước khi bắt đầu quá trình Data Cleaning, việc kiểm tra dữ liệu ban đầu là quan trọng để hiểu tình trạng của nó và xác định các vấn đề cụ thể cần được giải quyết.

Kiểm tra dữ liệu sau khi làm sạch: Sau khi hoàn thành quá trình Data Cleaning, bạn nên kiểm tra lại dữ liệu để đảm bảo rằng nó đã được làm sạch một cách hiệu quả và không có lỗi còn sót lại.

Tính bảo mật và quyền riêng tư: Trong quá trình Data Cleaning, đảm bảo rằng thông tin nhạy cảm đã được ẩn danh hoặc che kín để đảm bảo tính riêng tư và tuân thủ các quy định về bảo mật dữ liệu.

Phân phối và lưu trữ dữ liệu: Lưu ý cách dữ liệu đã được phân phối và lưu trữ sau khi được làm sạch. Điều này có thể liên quan đến việc sử dụng cơ sở dữ liệu trực tuyến hoặc lưu trữ dữ liệu trong các kho lưu trữ dữ liệu có cấu trúc.

Liên quan đến quản lý dữ liệu: Data Cleaning thường được thực hiện bởi các chuyên gia dữ liệu hoặc nhóm quản lý dữ liệu chuyên nghiệp, những người đảm bảo rằng quy trình này tuân thủ các tiêu chuẩn và quy định của tổ chức và ngành công nghiệp.

Thách thức và khó khăn: Data Cleaning có thể đối mặt với các thách thức như sự phức tạp của dữ liệu, mất mát dữ liệu, và sự phụ thuộc vào nguồn dữ liệu bên ngoài. Việc hiểu và vượt qua các thách thức này là quan trọng để đảm bảo sự thành công của quá trình làm sạch dữ liệu.

\* **VD:**

```
import pandas as pd

# Tạo một DataFrame giả định với dữ liệu raw
data = {'Tên': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Frank'],
        'Tuổi': [25, -30, 28, 35, None, 40],
        'Điểm': [85, 92, 78, 102, 75, 110]}

df = pd.DataFrame(data)

# Hiển thị tập dữ liệu ban đầu
print("Dữ liệu ban đầu:")
print(df)

# Loại bỏ giá trị âm ở cột 'Tuổi'
df = df[df['Tuổi'] > 0]

# Điền giá trị thiếu ở cột 'Điểm' bằng giá trị trung bình
mean_score = df['Điểm'].mean()
df['Điểm'].fillna(mean_score, inplace=True)

# Hiển thị tập dữ liệu sau khi đã làm sạch
print("\nDữ liệu sau khi đã làm sạch:")
print(df)
```

```
Dữ liệu ban đầu:
   Tên  Tuổi  Điểm
0  Alice  25.0   85
1   Bob -30.0   92
2 Charlie  28.0   78
3  David  35.0  102
4   Eva   NaN   75
5  Frank  40.0  110

Dữ liệu sau khi đã làm sạch:
   Tên  Tuổi  Điểm
0  Alice  25.0   85
2 Charlie  28.0   78
3  David  35.0  102
5  Frank  40.0  110
```

## 2.3 Wrangling data: Các bước Wrangling data.

Data Wrangling, còn được gọi là "điều khiển dữ liệu," là quá trình biến đổi và ánh xạ dữ liệu từ một định dạng dữ liệu "nguyên thô" sang một định dạng khác để làm cho nó phù hợp và có giá trị hơn cho các mục đích tiếp theo như phân tích dữ liệu. Mục tiêu của Data Wrangling là đảm bảo chất lượng và tính hữu ích của dữ liệu. Thông thường, các nhà phân tích dữ liệu dành phần lớn thời gian của họ trong quá trình Data Wrangling so với việc thực hiện phân tích thực sự.

Quá trình Data Wrangling có thể bao gồm việc biến đổi dữ liệu, trực quan hóa dữ liệu, tổng hợp dữ liệu, huấn luyện một mô hình thống kê và nhiều mục đích sử dụng khác. Data Wrangling thường tuân theo một loạt các bước chung, bắt đầu từ việc trích xuất dữ liệu nguyên thô từ nguồn dữ liệu, "điều khiển" dữ liệu nguyên thô (ví dụ: sắp xếp) hoặc phân tích cú pháp dữ liệu thành cấu trúc dữ liệu được định trước và cuối cùng đưa nội dung kết quả vào một nơi lưu trữ và sử dụng trong tương lai.

Data Wrangling thường đi kèm với việc Mapping. Thuật ngữ "Data Mapping" liên quan đến phần của quá trình điều khiển dữ liệu mà liên quan đến việc xác định các trường dữ liệu nguồn và các trường dữ liệu đích tương ứng của chúng. Trong khi Data Wrangling chuyên về việc biến đổi dữ liệu, Mapping là về việc kết nối các yếu tố khác nhau.

Quá trình Data Wrangling thường bao gồm các bước sau:

**Khám phá (Discovery):** Trước khi bắt đầu quá trình Data Wrangling, cần xem xét mục tiêu bạn muốn đạt được với dữ liệu và thu thập dữ liệu tương ứng. Điều này bao gồm xác định mục tiêu và sử dụng dữ liệu sau khi đã hoàn thành quá trình Wrangling.

**Tổ chức (Organization):** Dữ liệu thô thường rất phức tạp và đa dạng về loại và nguồn gốc. Bước này liên quan đến cấu trúc lại dữ liệu để làm cho nó dễ quản lý hơn.

**Làm sạch (Cleaning):** Sau khi dữ liệu đã được tổ chức, bạn có thể bắt đầu làm sạch dữ liệu. Quá trình làm sạch dữ liệu bao gồm loại bỏ các giá trị ngoại lai, định dạng các giá trị bị thiếu và loại bỏ dữ liệu trùng lặp.

**Bổ sung dữ liệu (Data Enrichment):** Bước này đòi hỏi bạn xem xét xem có đủ dữ liệu để tiếp tục không. Việc hoàn thành quá trình Wrangling mà thiếu dữ liệu có thể ảnh hưởng đến hiểu biết thu được từ phân tích tiếp theo.

**Kiểm tra (Validation):** Sau khi xác định đã thu thập đủ dữ liệu, bạn cần áp dụng các quy tắc kiểm tra dữ liệu. Quy tắc kiểm tra này xác nhận tính nhất quán của dữ liệu trong toàn bộ tập dữ liệu và đảm bảo tính chất lượng cũng như bảo mật.

**Xuất bản (Publishing):** Bước cuối cùng của quá trình Data Wrangling là xuất bản dữ liệu. Điều này bao gồm việc chuẩn bị dữ liệu cho sử dụng trong tương lai và tạo tài liệu về quá trình Wrangling.

Ưu điểm của Data Wrangling bao gồm tính nhất quán của dữ liệu, cải thiện hiểu biết từ dữ liệu, hiệu quả về chi phí và khả năng sử dụng dữ liệu dễ dàng hơn.

Công cụ Data Wrangling bao gồm nhiều phần mềm và công cụ khác nhau để thu thập, nhập, cấu trúc và làm sạch dữ liệu trước khi sử dụng chúng cho phân tích và ứng dụng Business Intelligence. Các công cụ này có thể được tự động hóa để kiểm tra và sửa lỗi trong quá trình biến đổi dữ liệu.

Với Data Wrangling, bạn có thể nắm bắt toàn bộ tiềm năng của dữ liệu của mình và làm cho nó trở nên hữu ích và chất lượng hơn. Quá trình này giúp làm sạch, cấu trúc và tổ chức dữ liệu để chuẩn bị cho việc phân tích và ra quyết định kinh doanh cụ thể.

Use Case of Data Wrangling (Các Ứng Dụng của Data Wrangling):

Data Wrangling có nhiều ứng dụng trong thực tế, và sau đây là một số ví dụ:

Phát hiện gian lận (Fraud Detection): Data Wrangling giúp xác định gian lận doanh nghiệp bằng cách phân tích thông tin chi tiết như email hoặc cuộc trò chuyện web đa bên và đa tầng. Điều này giúp hỗ trợ an ninh dữ liệu và duyệt qua hàng tỷ nhiệm vụ an ninh hàng ngày.

Phân tích Hành vi Khách hàng (Customer Behavior Analysis): Công cụ Data Wrangling có thể giúp đội ngũ tiếp thị hiểu rõ hơn về hành vi của khách hàng. Điều này giúp họ ra quyết định kinh doanh dựa trên thông tin khách hàng.

Data Wrangling Tools (Công cụ Data Wrangling):

Có nhiều công cụ và phần mềm dùng cho Data Wrangling, từ công cụ thủ công đến tự động. Dưới đây là một số ví dụ về các công cụ phổ biến:

Spreadsheets / Excel Power Query: Đây là công cụ cơ bản để thực hiện Data Wrangling bằng tay.

OpenRefine: Một công cụ tự động giúp làm sạch dữ liệu, nhưng yêu cầu kỹ năng lập trình.

Google DataPrep: Dịch vụ dành riêng cho việc khám phá, làm sạch và chuẩn bị dữ liệu.

Data Wrangler: Công cụ giúp làm sạch và biến đổi dữ liệu.

Plotly (data wrangling with Python): Công cụ hữu ích để làm việc với dữ liệu biểu đồ và bản đồ bằng Python.

Lợi ích của Data Wrangling (Benefits of Data Wrangling):

Việc sử dụng Data Wrangling mang lại nhiều lợi ích, bao gồm tính nhất quán của dữ liệu, cải thiện hiểu biết từ dữ liệu, hiệu quả về chi phí và khả năng sử dụng dữ liệu dễ dàng

hơn. Việc tự động hóa Data Wrangling đặc biệt quan trọng khi làm việc với các tập dữ liệu lớn.

Các Định dạng của Data Wrangling (Data Wrangling Formats):

Dựa vào loại dữ liệu bạn sử dụng, kết quả cuối cùng của Data Wrangling có thể thuộc một trong bốn định dạng sau: giao dịch không chuẩn, bảng cơ sở phân tích (ABT), dữ liệu chuỗi thời gian hoặc thư viện tài liệu.

Giao dịch không chuẩn (Transactional data): Bao gồm thông tin về giao dịch doanh nghiệp. Dữ liệu này có thể bao gồm thông tin chi tiết về giao dịch, tương tác khách hàng, biên lai và ghi chú về giao dịch ngoại trừ.

Bảng cơ sở phân tích (Analytical Base Table - ABT): Bảng dữ liệu ABT chứa thông tin về các yếu tố khác nhau của doanh nghiệp. Loại dữ liệu này phù hợp để sử dụng trong AI và ML.

Dữ liệu chuỗi thời gian (Time-series data): Dữ liệu chuỗi thời gian liên quan đến thời gian hoặc các dữ liệu có mối quan hệ với thời gian.

Thư viện tài liệu (Document library): Thông tin trong thư viện tài liệu chứa nhiều dữ liệu văn bản, đặc biệt là văn bản trong tài liệu.

## 2.4 Evaluate ML models:

Khi bạn đã xây dựng một mô hình học máy, bước tiếp theo quan trọng là đánh giá mô hình để đảm bảo rằng nó hoạt động tốt và đáng tin cậy trước khi triển khai nó vào môi trường thực tế. Đánh giá mô hình là quá trình xác định hiệu suất của mô hình trên dữ liệu chưa nhìn thấy trước. Dưới đây là một số khái niệm quan trọng và phương pháp để đánh giá các mô hình học máy:

Tập kiểm tra và Tập huấn luyện: Trong quá trình đánh giá mô hình, dữ liệu thường được chia thành hai phần chính: tập kiểm tra và tập huấn luyện. Tập huấn luyện được sử dụng để huấn luyện mô hình, trong khi tập kiểm tra được sử dụng để đánh giá hiệu suất. Đảm bảo rằng việc chia dữ liệu được thực hiện một cách ngẫu nhiên và cân đối để tránh các hiện tượng như overfitting hoặc underfitting.

Metrics (Độ đo) phổ biến:

Accuracy (Độ chính xác): Đây là tỷ lệ số lượng dự đoán đúng trên tổng số dự đoán. Tuy nhiên, độ chính xác có thể bị lừa dối trong trường hợp các lớp trong tập dữ liệu không cân bằng.



**Precision (Độ chính xác dương tính):** Tỷ lệ số lượng dự đoán đúng positive trên tổng số positive dự đoán. Được sử dụng trong bài toán phân loại để đo lường khả năng dự đoán đúng các trường hợp positive.

**Recall (Độ nhớ):** Tỷ lệ số lượng dự đoán đúng positive trên tổng số positive thực tế. Được sử dụng để đo lường khả năng tìm ra tất cả các trường hợp positive.

**F1-Score:** Kết hợp giữa Precision và Recall thành một con số duy nhất, giúp đánh giá hiệu suất tổng thể của mô hình.

**Confusion Matrix (Ma trận nhầm lẫn):** Ma trận nhầm lẫn là một công cụ quan trọng để xem xét hiệu suất của mô hình phân loại. Nó bao gồm các phần tử như True Positive (TP), True Negative (TN), False Positive (FP), và False Negative (FN) để đánh giá số lượng dự đoán đúng và sai trong từng lớp.

**Receiver Operating Characteristic (ROC) và Area Under the Curve (AUC):** Đây là một phương pháp đánh giá mô hình phân loại trong trường hợp có nhiều ngưỡng quyết định khác nhau. Biểu đồ ROC và giá trị AUC được sử dụng để đánh giá khả năng phân biệt của mô hình giữa các lớp.

**Cross-Validation (Kiểm tra chéo):** Kiểm tra chéo là một phương pháp quan trọng để đánh giá mô hình một cách tổng quan hơn. Nó bao gồm chia dữ liệu thành nhiều phần, huấn luyện và kiểm tra mô hình trên các phần khác nhau và tính trung bình kết quả. Loại kiểm tra chéo phổ biến bao gồm Kiểm tra chéo K-Fold và Leave-One-Out (LOO) Cross-Validation.

**Bias và Variance:** Đánh giá mô hình cũng liên quan đến việc kiểm tra xem mô hình có tend to overfit (quá khớp) hoặc underfit (thiếu khớp) trên dữ liệu hay không. Overfitting xuất hiện khi mô hình quá phức tạp và fit quá tốt với dữ liệu huấn luyện, nhưng không tổng quát hóa tốt trên dữ liệu mới. Underfitting xảy ra khi mô hình quá đơn giản và không thể fit dữ liệu huấn luyện một cách đủ tốt.

**Tuning Hyperparameters (Điều chỉnh siêu tham số):** Siêu tham số là các tham số mà bạn có thể điều chỉnh để cải thiện hiệu suất mô hình. Điều này bao gồm việc thử nghiệm các giá trị khác nhau của các siêu tham số và đánh giá tác động của chúng đối với hiệu suất mô hình.

**Interpretability (Khả năng giải thích):** Đối với một số ứng dụng quan trọng, đánh giá mô hình không chỉ liên quan đến hiệu suất mà còn đến khả năng giải thích. Mô hình cần có khả năng giải thích dự đoán của nó để người dùng có thể hiểu lý do tại sao mô hình ra quyết định như vậy.

Validation Set (Tập kiểm tra): Đôi khi, bạn cần tạo ra một tập kiểm tra riêng để đánh giá cuối cùng của mô hình sau khi điều chỉnh và tinh chỉnh các siêu tham số. Tập kiểm tra này không được sử dụng trong quá trình huấn luyện hoặc điều chỉnh siêu tham số.

Đánh giá các mô hình học máy là một phần quan trọng của quy trình phát triển mô hình và giúp bạn xác định mô hình nào hoạt động tốt nhất cho vấn đề cụ thể của bạn.

VD1: Đánh mô hình phân loại bằng cross-validation và tính toán confusion matrix.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
# Load dữ liệu Iris là một ví dụ
iris = load_iris()
X = iris.data
y = iris.target
# Chọn một mô hình phân loại (ví dụ: RandomForestClassifier)
clf = RandomForestClassifier()
# Sử dụng cross-validation để đánh giá mô hình
y_pred = cross_val_predict(clf, X, y, cv=5)
confusion = confusion_matrix(y, y_pred)
print("Confusion Matrix:")
print(confusion)
```

```
Confusion Matrix:
[[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]
```

VD2:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score

# Tạo mô hình RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train) # Huấn luyện mô hình trước

# Dự đoán trên dữ liệu kiểm tra
y_pred = clf.predict(X_test)

# Tính độ chính xác và độ nhớ lại
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')

print("Precision (Macro):", precision)
print("Recall (Macro):", recall)
```

```
Precision (Macro): 0.8185344827586207
Recall (Macro): 0.8105042016806723
```

VD3:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error

# Tạo mô hình hồi quy
regressor = LinearRegression()

# Chuyển đổi các giá trị chuỗi sang số thực
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Huấn luyện mô hình bằng dữ liệu huấn luyện (X_train và y_train_encoded)
regressor.fit(X_train, y_train_encoded)

# Sau đó, thực hiện dự đoán và tính toán Mean Absolute Error
y_pred = regressor.predict(X_test)
mae = mean_absolute_error(y_test_encoded, y_pred) # Sử dụng y_test_encoded
print("Mean Absolute Error:", mae)
```

Mean Absolute Error: 0.4108629881550926

VD4:

```
from sklearn.metrics import f1_score
# Sử dụng mô hình phân loại đã huấn luyện (clf) để dự đoán
y_pred = clf.predict(X_test)
# Tính toán F1-score với average='micro'
f1_micro = f1_score(y_test, y_pred, average='micro')
# Tính toán F1-score với average='macro'
f1_macro = f1_score(y_test, y_pred, average='macro')

# Tính toán F1-score với average='weighted'
f1_weighted = f1_score(y_test, y_pred, average='weighted')
print("F1-score (micro):", f1_micro)
print("F1-score (macro):", f1_macro)
print("F1-score (weighted):", f1_weighted)
```

F1-score (micro): 0.8115942028985508  
F1-score (macro): 0.8101587301587302  
F1-score (weighted): 0.810397975615367

## 2.5

### a. k-nearest

KNN (K-Nearest Neighbors) là một thuật toán học máy trong lĩnh vực phân loại và dự đoán. Thuật toán KNN được sử dụng để xác định lớp hoặc giá trị dự đoán của một điểm dữ liệu mới dựa trên các điểm dữ liệu trong tập huấn luyện.

Cách thức hoạt động của thuật toán KNN như sau:

- Giả sử chúng ta có một tập dữ liệu huấn luyện gồm các điểm dữ liệu đã được gán nhãn (có lớp hoặc giá trị dự đoán).
- Khi có một điểm dữ liệu mới cần phân loại hoặc dự đoán, thuật toán KNN tìm ra K điểm dữ liệu gần nhất (K là một số nguyên dương đã xác định trước) trong tập huấn luyện.
- Thuật toán tính toán khoảng cách giữa điểm dữ liệu mới và các điểm dữ liệu trong tập huấn luyện, thông thường sử dụng khoảng cách Euclidean.
- Sau đó, thuật toán xác định lớp hoặc giá trị dự đoán cho điểm dữ liệu mới dựa trên đa số lớp hoặc giá trị của K điểm gần nhất. Trong trường hợp phân loại, lớp được xác định bằng cách đếm số lượng điểm thuộc từng lớp và chọn lớp có số lượng nhiều nhất. Trong trường hợp dự đoán giá trị số, giá trị dự đoán thường là trung bình của giá trị của K điểm gần nhất.
- Cuối cùng, thuật toán trả về lớp hoặc giá trị dự đoán của điểm dữ liệu mới.

Triển khai KNN:

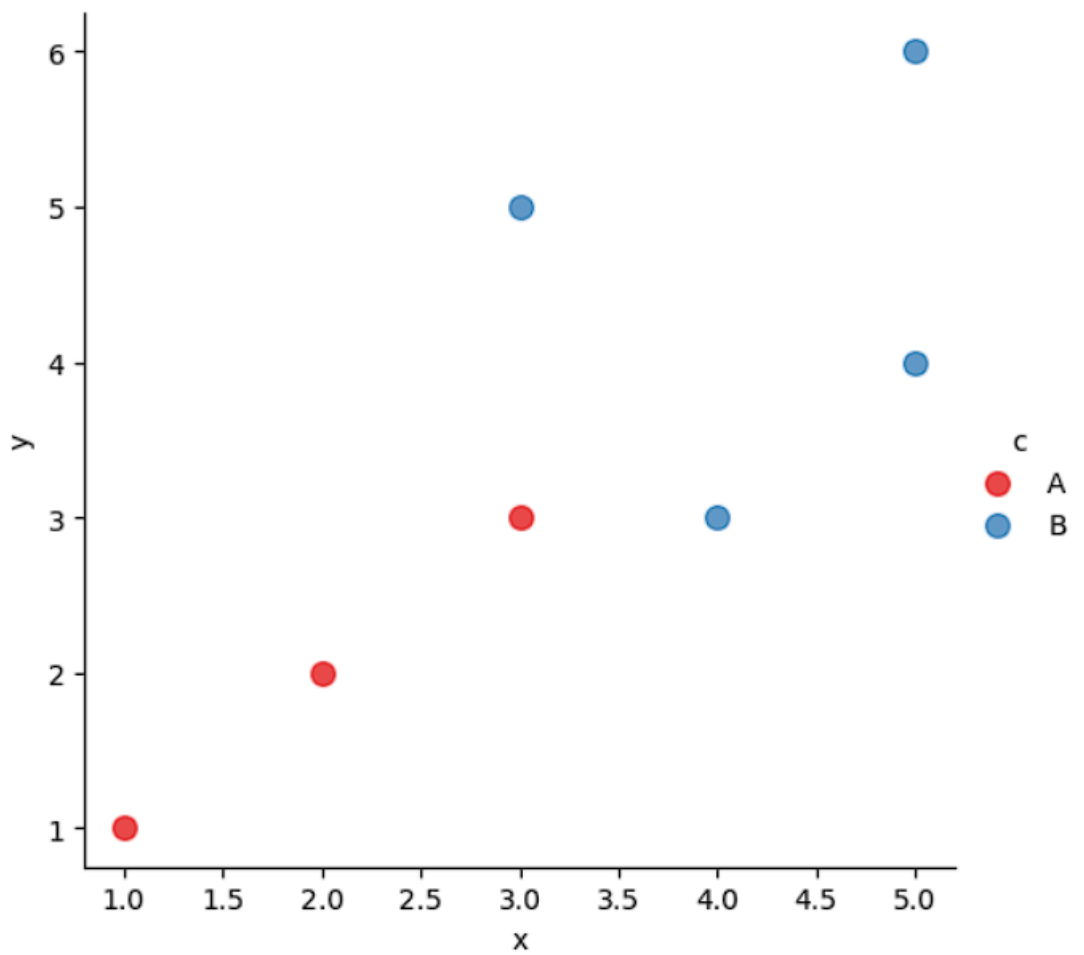
File knn.csv

```
x,y,c
1,1,A
2,2,A
4,3,B
3,3,A
3,5,B
5,6,B
5,4,B
```

Vẽ các điểm:

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

data = pd.read_csv("knn.csv")
sns.lmplot(x='x', y='y', data=data, hue='c', palette='Set1', fit_reg=False, scatter_kws={"s": 70})
plt.show()
```



Tính khoảng cách giữa các điểm:

```
def euclidean_distance(pt1, pt2, dimension):  
    distance = 0  
    for x in range(dimension):  
        distance += np.square(pt1[x] - pt2[x])  
    return np.sqrt(distance)
```

Triển khai KNN:

```
def knn(training_points, test_point, k):
    distances = {}
    #---the number of axes we are dealing with---
    dimension = test_point.shape[1]
    #--calculating euclidean distance between each
    # point in the training data and test data
    for x in range(len(training_points)):
        dist = euclidean_distance(test_point, training_points.iloc[x], dimension)
        #---record the distance for each training points---
        distances[x] = dist[0]

    #---sort the distances---
    sorted_d = sorted(distances.items(), key=operator.itemgetter(1))

    #---to store the neighbors---
    neighbors = []
    #---xtract the top k neighbors---
    for x in range(k):
        neighbors.append(sorted_d[x][0])

    #---for each neighbor found, find out its class---
    class_counter = {}
    for x in range(len(neighbors)):
        #---find out the class for that particular point---
        cls = training_points.iloc[neighbors[x]][-1]
        if cls in class_counter:
            class_counter[cls] += 1
        else:
            class_counter[cls] -= 1

    #---sort the class_counter in descending order---
    sorted_counter = sorted(class_counter.items(), key=operator.itemgetter(1), reverse=True)

    #---return the class with the most count, as well as the neighbors found
    return sorted_counter[0][0], neighbors
```

Đưa ra dự đoán:

```
import operator
#---test point---
test_set = [[3, 3.9]]
test = pd.DataFrame(test_set)
cls,neighbors = knn(data, test, 5)
print("Predicted Class: " + cls)

Predicted Class: B
```

Hình dung các giá trị khác nhau của K:

```
#---generate the color map for the scatter plot---
#---if column 'c' is A, then use Red, else use Blue---
colors = ['r' if i == 'A' else 'b' for i in data['c']]
ax = data.plot(kind='scatter', x='x', y='y', c=colors)
plt.xlim(0, 7)
plt.ylim(0, 7)

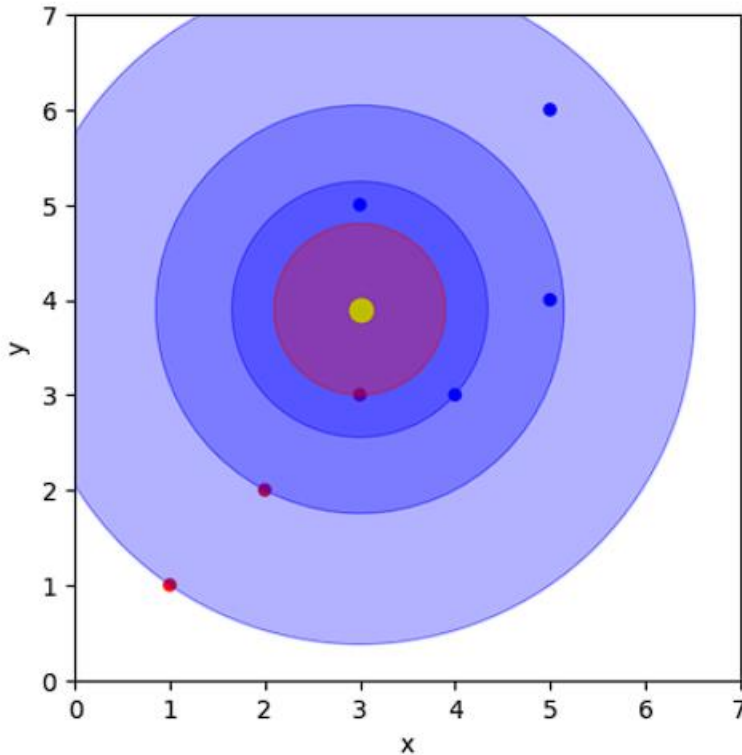
#---plot the test point---
plt.plot(test_set[0][0], test_set[0][1], "yo", markersize=9)

test_df = pd.DataFrame(test_set)
for k in range(7, 0, -2):
    cls, neighbors = knn(data, test_df, k)
    print("=====")
    print("k=", k)
    print("Class:", cls)
    print("Neighbors:")
    print(data.iloc[neighbors])
    furthest_point = data.iloc[neighbors].tail(1)

    #---draw a circle connecting the test point and the furthest point---
    radius = euclidean_distance(test_df, furthest_point.iloc[0], 2)

    #---display the circle in red if classification is A, else display circle in blue---
    c = 'r' if cls == 'A' else 'b'
    circle = plt.Circle((test_set[0][0], test_set[0][1]), radius, color=c, alpha=0.3)
    ax.add_patch(circle)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```

```
=====
k= 7
Class: B
Neighbors:
   x  y  c
3  3  3  A
4  3  5  B
2  4  3  B
6  5  4  B
1  2  2  A
5  5  6  B
0  1  1  A
=====
k= 5
Class: B
Neighbors:
   x  y  c
3  3  3  A
4  3  5  B
2  4  3  B
6  5  4  B
1  2  2  A
=====
k= 3
Class: B
Neighbors:
   x  y  c
3  3  3  A
4  3  5  B
2  4  3  B
=====
k= 1
Class: A
Neighbors:
   x  y  c
3  3  3  A
```



#### b. k-means

K-means là một thuật toán phân cụm trong machine learning, được sử dụng để phân loại dữ liệu vào các nhóm khác nhau dựa trên các đặc trưng của chúng.

Thuật toán này hoạt động bằng cách tìm ra các trung tâm cụm sao cho tổng bình phương khoảng cách từ các điểm dữ liệu đến trung tâm cụm gần nhất là nhỏ nhất.

Triển khai k-means:

File kmeans.csv:

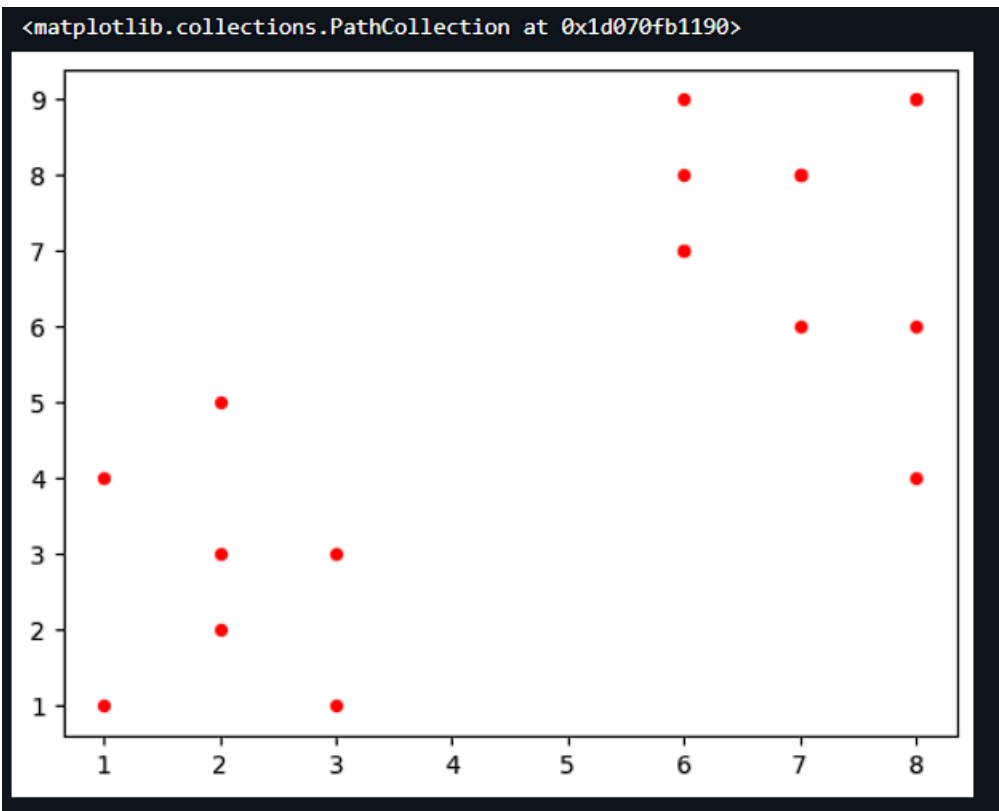
1	x,y
2	1,1
3	2,2
4	2,3
5	1,4
6	3,3
7	6,7
8	7,8
9	6,8
10	7,6
11	6,9
12	2,5
13	7,8
14	8,9
15	6,7
16	7,8
17	3,1
18	8,4
19	8,6
20	8,9



Tải tệp CSV vào khung dữ liệu Pandas và vẽ biểu đồ phân tán hiển thị các điểm:

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

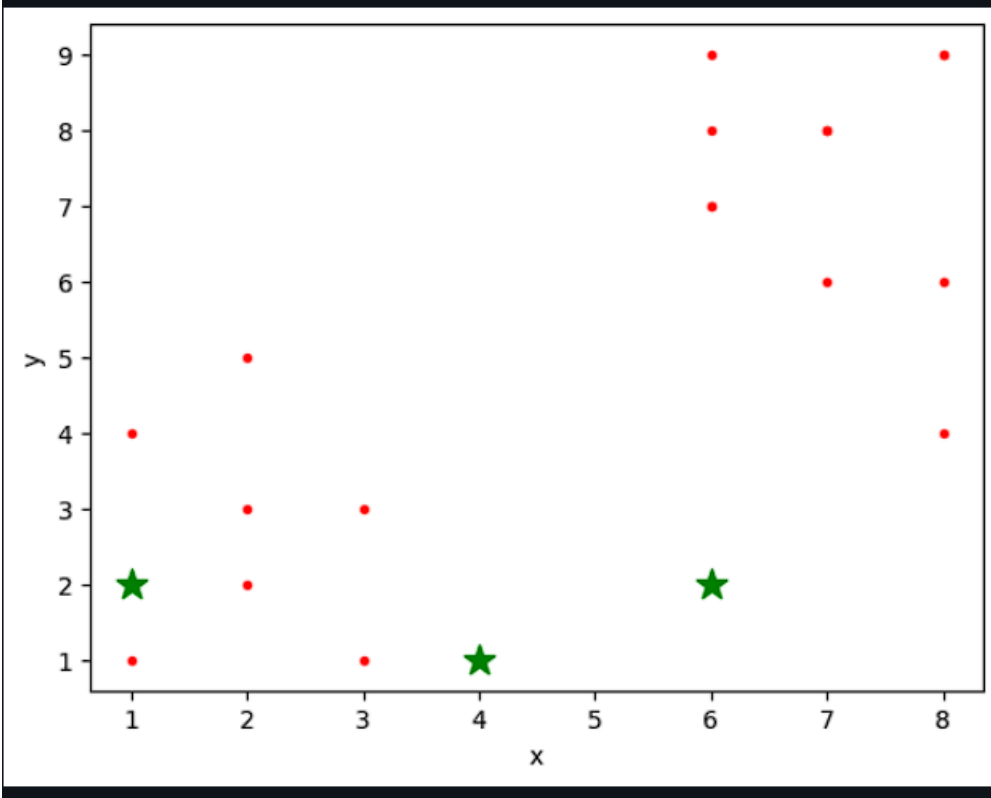
df = pd.read_csv("kmeans.csv")
plt.scatter(df['x'], df['y'], c='r', s=18)
```



Giả sử  $K=3$ , tạo 3 trọng tâm ngẫu nhiên và đánh dấu trên đồ thị

```
#---let k assume a value---
k = 3
#---create a matrix containing all points---
X = np.array(list(zip(df['x'], df['y'])))
#---generate k random points (centroids)---
Cx = np.random.randint(np.min(X[:,0]), np.max(X[:,0]), size = k)
Cy = np.random.randint(np.min(X[:,1]), np.max(X[:,1]), size = k)
#---represent the k centroids as a matrix---
C = np.array(list(zip(Cx, Cy)), dtype=np.float64)
print(C)
#---plot the original points as well as the k centroids---
plt.scatter(df['x'], df['y'], c='r', s=8)
plt.scatter(Cx, Cy, marker='*', c='g', s=160)
plt.xlabel("x")
plt.ylabel("y")
```

```
[[6. 2.]  
 [4. 1.]  
 [1. 2.]  
 Text(0, 0.5, 'y')
```



## Triển khai k-means:

```
from copy import deepcopy

#---to calculate the distance between two points---
def euclidean_distance (a, b, ax=1):
    return np.linalg.norm (a - b, axis=ax)

#---create a matrix of 0 with same dimension as C (centroids) ---
C_prev= np.zeros(C. shape)

#---to store the cluster each point belongs to---
clusters = np.zeros(len(X))

#---C is the random centroids and C_prev is all 00---
#---measure the distance between the centroids and prev---
distance_differences = euclidean_distance (C, C_prev)

#---Loop as long as there is still a difference in
# distance between the previous and current centroids---
while distance_differences.any() != 0:
    #---assign each value to its closest cluster---
    for i in range (len (X)):
        distances = euclidean_distance (X[i], C)

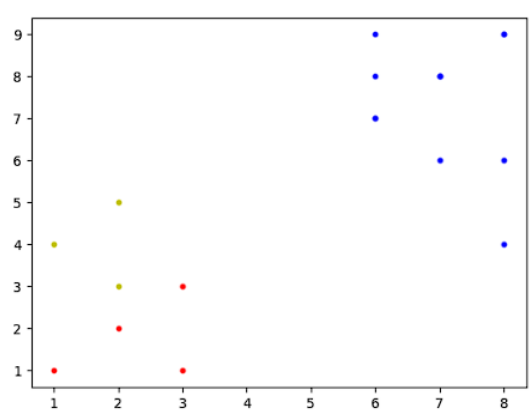
        #---returns the indices of the minimum values along an axis---
        cluster = np.argmin(distances)
        clusters [i] = cluster

    #---store the prev centroids---
    C_prev = deepcopy(C)

    #---find the new centroids by taking the average value---
    for i in range(k): #---k is the number of clusters---
        #take all the points in cluster i---
        points = [X[j] for j in range (len (X)) if clusters [j] == i]
        if len (points) != 0:
            C[i] = np.mean (points, axis=0)

    #---find the distances between the old centroids and the new centroids..
    distance_differences = euclidean_distance (C, C_prev)

#---plot the scatter plot---
colors= ['b', 'r', 'y', 'g', 'c', 'm']
for i in range (k):
    points= np.array([X[j] for j in range (len(X)) if clusters[j] == i])
    if len (points) > 0:
        plt.scatter(points[:, 0], points[:, 1], s=10, c=colors[i])
    else:
        # this means that one of the clusters has no points
        print("Please regenerate your centroids again.")
        plt.scatter(points[:, 0], points[:, 1], s=10, c=colors[i])
        plt.scatter(C[:, 0], C[:, 1], marker='*', s=100, c='black')
```



In ra các cụm mà mỗi điểm thuộc về:

```
for i, cluster in enumerate(clusters):  
    print("Point " + str(x[i]), "Cluster " + str(int(cluster)))
```

```
Point [1 1] Cluster 0  
Point [2 2] Cluster 0  
Point [2 3] Cluster 0  
Point [1 4] Cluster 0  
Point [3 3] Cluster 0  
Point [6 7] Cluster 0  
Point [7 8] Cluster 0  
Point [6 8] Cluster 0  
Point [7 6] Cluster 0  
Point [6 9] Cluster 0  
Point [2 5] Cluster 0  
Point [7 8] Cluster 0  
Point [8 9] Cluster 0  
Point [6 7] Cluster 0  
Point [7 8] Cluster 0  
Point [3 1] Cluster 0  
Point [8 4] Cluster 0  
Point [8 6] Cluster 0  
Point [8 9] Cluster 0
```

## 2.6

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier

# Load the Iris dataset
iris = datasets.load_iris()
x = iris.data[:, :2] # X-Axis: sepal length-width
y = iris.target # Y-Axis: species

# Define plot Limits
x_min, x_max = x[:, 0].min() - 0.5, x[:, 0].max() + 0.5
y_min, y_max = x[:, 1].min() - 0.5, x[:, 1].max() + 0.5

# Create a mesh grid for the decision boundary
h = 0.02 # Step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Create a K-nearest neighbors classifier
knn = KNeighborsClassifier()
knn.fit(x, y)

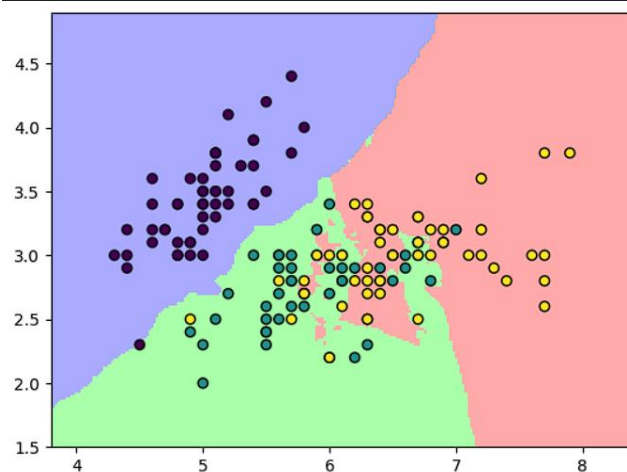
# Predict the class for each point in the mesh grid
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Create a colormap for the decision regions
cmap_light = ListedColormap(['#AAAAFF', '#AAFFAA', '#FFAAAA'])

# Create a figure and plot the decision boundaries
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot the training points
plt.scatter(x[:, 0], x[:, 1], c=y, edgecolor='k')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plt.show()
```



## \* Giải thích:

Import các thư viện và modules cần thiết:

- `numpy`: Được sử dụng cho việc xử lý mảng và tính toán.
- `matplotlib.pyplot`: Được sử dụng để vẽ biểu đồ.
- `matplotlib.colors.ListedColormap`: Được sử dụng để định nghĩa các màu cho các vùng phân loại.
- `sklearn.datasets`: Được sử dụng để tải bộ dữ liệu Iris.
- `sklearn.neighbors.KNeighborsClassifier`: Là một mô hình phân loại dựa trên K-Nearest Neighbors.

Tải bộ dữ liệu Iris từ `sklearn.datasets` và chọn ra hai đặc trưng (sepal length và sepal width) để đặt trên trục X và Y.

Xác định giới hạn của biểu đồ (điểm bắt đầu và kết thúc trên trục X và Y) để đảm bảo biểu đồ có đủ không gian để vẽ.

Tạo một mạng lưới (mesh grid) bằng cách sử dụng `np.meshgrid` với các điểm nằm trên trục X và Y. Mạng lưới này sẽ được sử dụng để vẽ biểu đồ quyết định (decision boundary).

Tạo một mô hình `KNeighborsClassifier` và huấn luyện nó trên dữ liệu huấn luyện (sepal length và sepal width).

Sử dụng mô hình đã được huấn luyện để dự đoán nhãn lớp cho mỗi điểm trên mạng lưới mesh grid. Điều này sẽ giúp chúng ta tạo ra một biểu đồ quyết định cho việc phân loại.

Định nghĩa các màu cho các vùng phân loại bằng cách sử dụng `ListedColormap`.

Vẽ biểu đồ quyết định bằng cách sử dụng `plt.pcolormesh` để hiển thị các vùng phân loại trên mạng lưới mesh grid.

Vẽ các điểm dữ liệu huấn luyện trên biểu đồ bằng cách sử dụng `plt.scatter` và sử dụng màu sắc khác nhau cho từng loại hoa Iris.

Đặt giới hạn của trục X và Y để đảm bảo biểu đồ hiển thị đầy đủ dữ liệu.

Hiển thị biểu đồ với `plt.show()`.

Kết quả là bạn sẽ thấy một biểu đồ với các vùng màu khác nhau cho từng loại hoa Iris và các điểm dữ liệu được hiển thị trên đó. Biểu đồ này minh họa cách mô hình KNN phân loại dữ liệu vào các lớp khác nhau dựa trên khoảng cách đến các điểm láng giềng gần nhất.

## 2.7

\* Linear regression:

Hồi quy tuyến tính (Linear regression) là một trong những phương pháp thường được sử dụng trong thống kê và học máy để mô hình hóa mối quan hệ tuyến tính giữa một biến độc lập (hoặc nhiều biến độc lập) và một biến phụ thuộc trong dữ liệu. Mục tiêu của hồi quy tuyến tính là xác định một mô hình tuyến tính (linear model) có thể dự đoán một biến phụ thuộc dựa trên các biến độc lập.

Cụ thể, mô hình tuyến tính xây dựng một đường thẳng (trong trường hợp đơn giản với một biến độc lập) hoặc một siêu mặt phẳng (trong trường hợp nhiều biến độc lập) sao cho khoảng cách giữa các điểm dữ liệu và đường thẳng (hoặc siêu mặt phẳng) này là nhỏ nhất. Đường thẳng (hoặc siêu mặt phẳng) này được sử dụng để dự đoán giá trị của biến phụ thuộc dựa trên giá trị của các biến độc lập.

Công thức của mô hình hồi quy tuyến tính cho trường hợp một biến độc lập (simple linear regression) có dạng:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Trong đó:

- $Y$  là biến phụ thuộc cần dự đoán.
- $X$  là biến độc lập.
- $\beta_0$  là hệ số chặn (intercept).
- $\beta_1$  là hệ số của biến độc lập (slope).
- $\epsilon$  là sai số (error term) đại diện cho sự biến đổi không thể dự đoán hoặc các yếu tố khác mà mô hình không xác định.

Mục tiêu khi xây dựng mô hình hồi quy tuyến tính là tìm các giá trị  $\beta_0$  và  $\beta_1$  tối ưu sao cho mô hình phù hợp nhất với dữ liệu huấn luyện và có khả năng dự đoán chính xác giá trị của  $Y$  cho các giá trị  $X$  mới.

Hồi quy tuyến tính có nhiều biến thể khác nhau, bao gồm hồi quy tuyến tính đa biến (multiple linear regression) cho trường hợp có nhiều biến độc lập, và hồi quy tuyến tính có chức năng (polynomial regression) để mô hình mối quan hệ bậc cao hơn giữa biến độc lập và biến phụ thuộc.

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets

diabetes = datasets.load_diabetes()
x_train = diabetes.data[:-20]
y_train = diabetes.target[:-20]
x_test = diabetes.data[-20:]
y_test = diabetes.target[-20:]

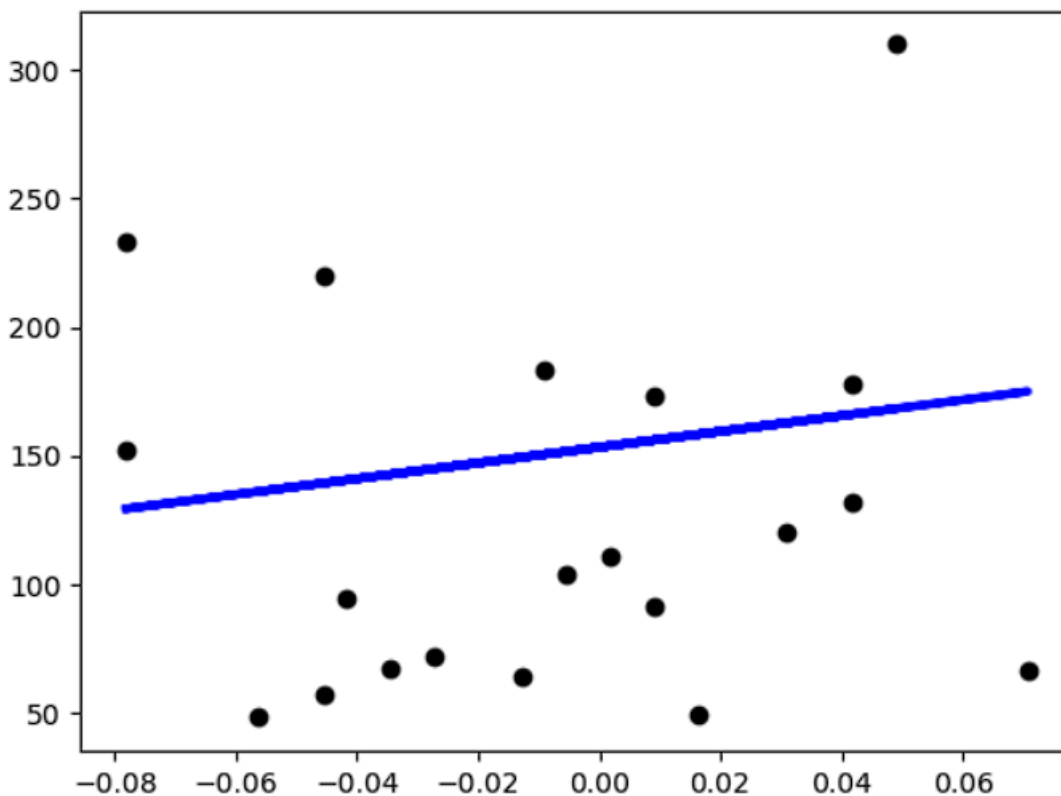
x0_test = x_test[:, 0]
x0_train = x_train[:, 0]

x0_test = x0_test[:, np.newaxis]
x0_train = x0_train[:, np.newaxis]

linreg = linear_model.LinearRegression()
linreg.fit(x0_train, y_train)
y = linreg.predict(x0_test)

plt.scatter(x0_test, y_test, color='k')
plt.plot(x0_test, y, color='b', linewidth=3)
plt.show()

```





\* Giải thích:

Dòng 1-3: Nhập các thư viện cần thiết.

Dòng 4: Tải dữ liệu về bệnh tiểu đường từ dataset mẫu được cung cấp bởi `datasets.load_diabetes()` và lưu trữ nó vào biến `diabetes`.

Dòng 5 - 8: Chia dữ liệu thành tập huấn luyện và tập kiểm tra. Dữ liệu của tập huấn luyện là `x_train` và `y_train`, bao gồm 20 mẫu cuối cùng của dữ liệu tiểu đường. Dữ liệu kiểm tra là `x_test` và `y_test`, bao gồm 20 mẫu đóng vai trò là tập kiểm tra.

Dòng 9 - 12: Chọn chỉ một đặc trưng từ tập dữ liệu, cụ thể là đặc trưng đầu tiên (`x0`) và chuyển chúng thành mảng 2D sử dụng `np.newaxis`. Điều này làm cho chúng có thể được sử dụng trong mô hình hồi quy tuyến tính.

Dòng 13: Tạo một mô hình hồi quy tuyến tính bằng cách sử dụng `linear_model.LinearRegression()` và lưu trữ nó trong biến `linreg`.

Dòng 14: Huấn luyện mô hình hồi quy tuyến tính (`linreg`) bằng cách sử dụng tập huấn luyện `x0_train` và `y_train`.

Dòng 15: dự đoán các giá trị mục tiêu (`y`) cho tập kiểm tra `x0_test` bằng cách sử dụng mô hình hồi quy tuyến tính đã huấn luyện.

Dòng 16: Vẽ biểu đồ scatter plot của dữ liệu thực tế từ tập kiểm tra bằng `plt.scatter()` với màu đen (`color='k'`).

Dòng 17: Vẽ đường hồi quy dự đoán bằng cách sử dụng `plt.plot()` với màu xanh (`color='b'`) và độ rộng đường là 3 (`linewidth=3`).

Dòng 18: hiển thị biểu đồ đã tạo bằng `plt.show()`

## 2.8

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets

# Tải dữ liệu về tiểu đường
diabetes = datasets.load_diabetes()

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
x_train = diabetes.data[:-20]
y_train = diabetes.target[:-20]
x_test = diabetes.data[-20:]
y_test = diabetes.target[-20:]

# Tạo một hình vẽ với kích thước 8x12 inches
plt.figure(figsize=(8, 12))

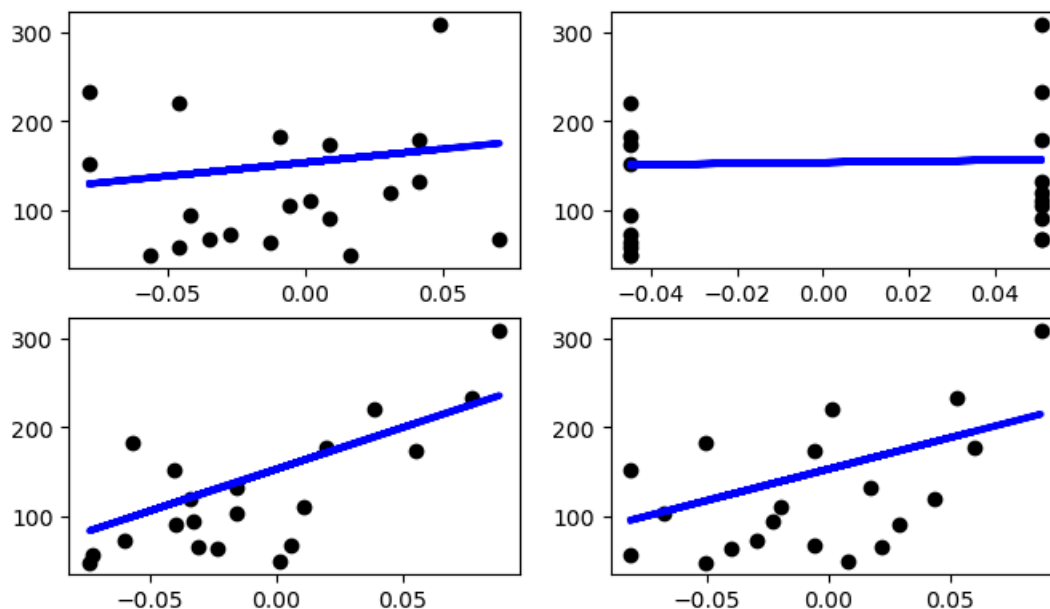
# Lặp qua từng đặc trưng từ 0 đến 9
for f in range(0, 10):
    xi_test = x_test[:, f]
    xi_train = x_train[:, f]
    xi_test = xi_test[:, np.newaxis]
    xi_train = xi_train[:, np.newaxis]

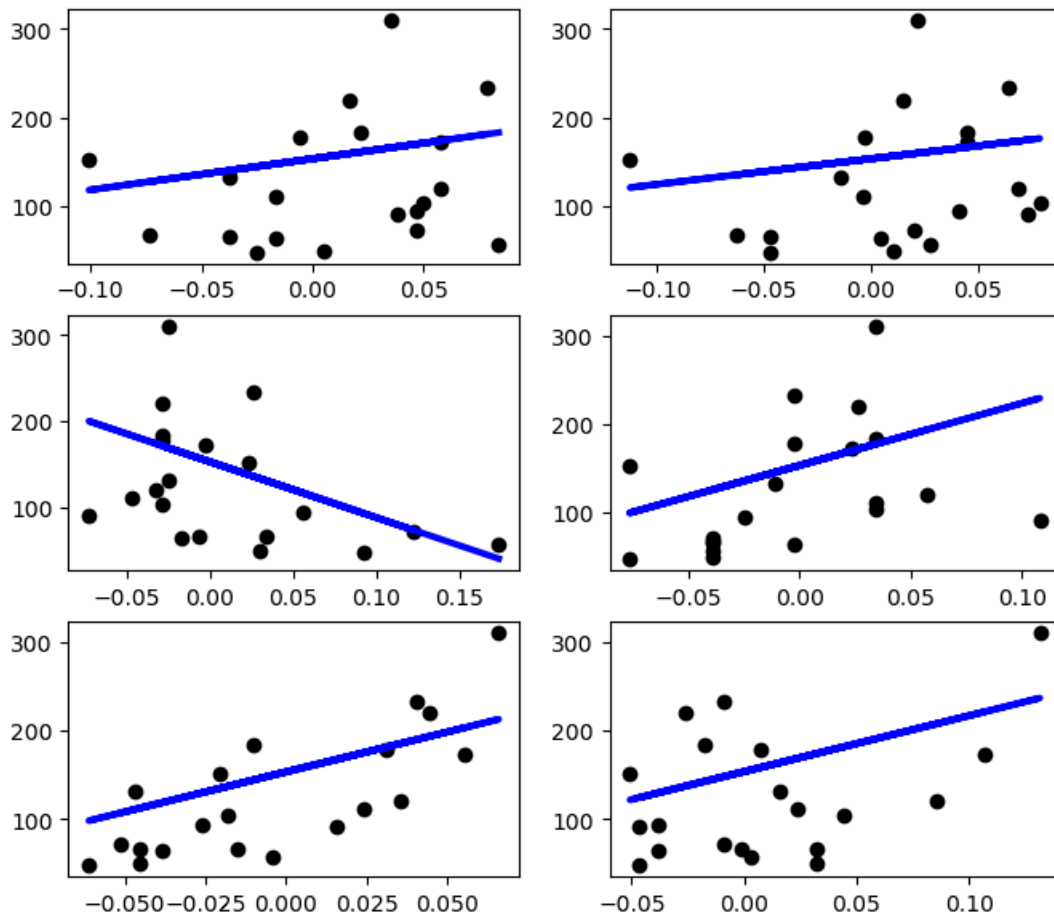
    # Tạo và huấn luyện mô hình hồi quy tuyến tính cho từng đặc trưng
    linreg = linear_model.LinearRegression()
    linreg.fit(xi_train, y_train)

    # Dự đoán giá trị
    y = linreg.predict(xi_test)

    # Vẽ biểu đồ cho từng đặc trưng
    plt.subplot(5, 2, f+1)
    plt.scatter(xi_test, y_test, color='k')
    plt.plot(xi_test, y, color='b', linewidth=3)

# Hiển thị biểu đồ
plt.show()
```





\* Giải thích:

Dòng 1-3: Nhập các thư viện cần thiết. numpy được nhập với tên viết tắt là np. matplotlib.pyplot được nhập với tên plt. linear\_model và datasets được nhập từ sklearn để sử dụng trong việc xây dựng mô hình hồi quy tuyến tính và tải dữ liệu mẫu.

Dòng 4: Tải dữ liệu về bệnh tiểu đường từ dataset mẫu được cung cấp bởi datasets.load\_diabetes() và lưu trữ nó vào biến diabetes.

Dòng 5-8: Chia dữ liệu thành tập huấn luyện và tập kiểm tra. Dữ liệu của tập huấn luyện là x\_train và y\_train, bao gồm 20 mẫu cuối cùng của dữ liệu tiểu đường. Dữ liệu kiểm tra là x\_test và y\_test, bao gồm 20 mẫu đóng vai trò là tập kiểm tra.

Dòng 9: Tạo một hình vẽ với kích thước 8x12 inches bằng plt.figure(figsize=(8, 12)).

Dòng 10: Bắt đầu một vòng lặp for để lặp qua từng đặc trưng từ 0 đến 9.

Dòng 11-14: Chọn chỉ một đặc trưng từ tập dữ liệu, cụ thể là đặc trưng thứ f và chuyển chúng thành mảng 2D sử dụng np.newaxis. Điều này làm cho chúng có thể được sử dụng trong mô hình hồi quy tuyến tính.

Dòng 15-16: Tạo một mô hình hồi quy tuyến tính (linreg) và sau đó huấn luyện mô hình bằng tập huấn luyện `xi_train` và `y_train` tương ứng với đặc trưng thứ `f`.

Dòng 17: Dự đoán giá trị `y` bằng cách sử dụng mô hình hồi quy tuyến tính đã huấn luyện (linreg) trên tập kiểm tra `xi_test`.

Dòng 18-21: Vẽ biểu đồ cho từng đặc trưng trong một cửa sổ con của biểu đồ lớn. `plt.subplot(5, 2, f+1)` tạo một ô con 5x2 và di chuyển qua từng ô con trong mỗi vòng lặp. `plt.scatter` vẽ biểu đồ scatter plot của dữ liệu thực tế từ tập kiểm tra với màu đen (`color='k'`), và `plt.plot` vẽ đường hồi quy dự đoán với màu xanh (`color='b'`) và độ rộng đường là 3 (`linewidth=3`).

Dòng 22: Hiện thị biểu đồ hoàn chỉnh bằng `plt.show()`.

## 2.9

\* Logistic regression:

Hồi quy logistic là một thuật toán học máy có giám sát, chủ yếu được sử dụng cho các nhiệm vụ phân loại trong đó mục tiêu là dự đoán xác suất một thể hiện thuộc về một lớp nhất định. Nó được sử dụng cho các thuật toán phân loại, tên của nó là hồi quy logistic. nó được gọi là hồi quy vì nó lấy đầu ra của hàm hồi quy tuyến tính làm đầu vào và sử dụng hàm sigmoid để ước tính xác suất cho lớp đã cho. Sự khác biệt giữa hồi quy tuyến tính và hồi quy logistic là đầu ra hồi quy tuyến tính là giá trị liên tục có thể là bất kỳ giá trị nào trong khi hồi quy logistic dự đoán xác suất một thể hiện có thuộc về một lớp nhất định hay không.

- Hồi quy logistic:

Nó được sử dụng để dự đoán biến phụ thuộc phân loại bằng cách sử dụng một tập hợp các biến độc lập nhất định.

Hồi quy logistic dự đoán đầu ra của một biến phụ thuộc phân loại. Do đó, kết quả phải là một giá trị phân loại hoặc rời rạc.

Nó có thể là Có hoặc Không, 0 hoặc 1, đúng hoặc Sai, v.v. nhưng thay vì đưa ra giá trị chính xác là 0 và 1, nó đưa ra các giá trị xác suất nằm trong khoảng từ 0 đến 1.

Hồi quy logistic rất giống với Hồi quy tuyến tính ngoại trừ cách chúng được sử dụng. Hồi quy tuyến tính được sử dụng để giải quyết các vấn đề hồi quy, trong khi hồi quy logistic được sử dụng để giải quyết các vấn đề phân loại.

Trong hồi quy logistic, thay vì khớp đường hồi quy, chúng tôi khớp hàm logistic hình chữ “S”, dự đoán hai giá trị tối đa (0 hoặc 1).

Đường cong từ hàm logistic cho biết khả năng xảy ra điều gì đó chẳng hạn như tế bào có bị ung thư hay không, chuột có bị béo phì hay không dựa trên trọng lượng của nó, v.v.

Hồi quy logistic là một thuật toán học máy quan trọng vì nó có khả năng cung cấp xác suất và phân loại dữ liệu mới bằng cách sử dụng các bộ dữ liệu liên tục và rời rạc.

Hồi quy logistic có thể được sử dụng để phân loại các quan sát bằng cách sử dụng các loại dữ liệu khác nhau và có thể dễ dàng xác định các biến hiệu quả nhất được sử dụng để phân loại.

- Hàm logistic (Hàm sigmoid):

Hàm sigmoid là một hàm toán học được sử dụng để ánh xạ các giá trị dự đoán thành xác suất.

Nó ánh xạ bất kỳ giá trị thực nào vào một giá trị khác trong phạm vi 0 và 1. o Giá trị của hồi quy logistic phải nằm trong khoảng từ 0 đến 1, không thể vượt quá giới hạn này, do đó, nó tạo thành một đường cong giống như dạng “S”.

Đường cong dạng S được gọi là hàm Sigmoid hoặc hàm logistic.

Trong hồi quy logistic, chúng tôi sử dụng khái niệm giá trị ngưỡng, xác định xác suất bằng 0 hoặc 1. Chẳng hạn như các giá trị trên giá trị ngưỡng có xu hướng là 1 và giá trị dưới giá trị ngưỡng có xu hướng là 0.

- Loại hồi quy logistic:

Dựa trên các loại, hồi quy logistic có thể được phân thành ba loại:

Nhị thức: Trong hồi quy logistic nhị thức, chỉ có thể có hai loại biến phụ thuộc, chẳng hạn như 0 hoặc 1, Đạt hoặc Thất bại, v.v.

Đa thức: Trong hồi quy logistic đa thức, có thể có 3 hoặc nhiều loại biến phụ thuộc không có thứ tự, chẳng hạn như “mèo”, “chó” hoặc “cừu”

Thứ tự: Trong hồi quy logistic thứ tự, có thể có 3 loại biến phụ thuộc được sắp xếp theo thứ tự trở lên, chẳng hạn như “thấp”, “Trung bình” hoặc “Cao”.

Sr.No	Linear Regression	Logistic Regression
1	Linear regression is used to predict the continuous dependent variable using a given set of independent variables.	Logistic regression is used to predict the categorical dependent variable using a given set of independent variables.
2	Linear regression is used for solving Regression problem.	It is used for solving classification problems.
3	In this we predict the value of continuous variables	In this we predict values of categorical variables
4	In this we find best fit line.	In this we find S-Curve .
5	Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for Estimation of accuracy.
6	The output must be continuous value,such as price,age,etc.	Output is must be categorical value such as 0 or 1, Yes or no, etc.
7	It required linear relationship between dependent and independent variables.	It not required linear relationship.
8	There may be collinearity between the independent variables.	There should not be collinearity between independent variable.

- Các thuật ngữ liên quan đến hồi quy logistic:

Dưới đây là một số thuật ngữ phổ biến liên quan đến hồi quy logistic:

+ Biến độc lập: Các đặc điểm đầu vào hoặc các yếu tố dự đoán được áp dụng cho các dự đoán của biến phụ thuộc.

+ Biến phụ thuộc: Biến mục tiêu trong mô hình hồi quy logistic mà chúng tôi đang cố gắng dự đoán.

+ Hàm logistic: Công thức được sử dụng để biểu thị mối liên hệ giữa các biến độc lập và phụ thuộc. Hàm logistic biến đổi các biến đầu vào thành giá trị xác suất trong khoảng từ 0 đến 1, biểu thị khả năng biến phụ thuộc là 1 hoặc 0.

+ Tỷ lệ: Đó là tỷ lệ giữa điều gì đó xảy ra và điều gì đó không xảy ra. nó khác với xác suất vì xác suất là tỷ lệ giữa điều gì đó xảy ra với mọi điều có thể xảy ra.

+ Tỷ lệ cược log: Tỷ lệ cược log, còn được gọi là hàm logit, là logarit tự nhiên của tỷ lệ cược. Trong hồi quy logistic, tỷ lệ log của biến phụ thuộc được mô hình hóa dưới dạng kết hợp tuyến tính của các biến độc lập và biến bị chặn.

+ Hệ số: Các tham số ước tính của mô hình hồi quy logistic, cho thấy các biến độc lập và phụ thuộc có liên quan với nhau như thế nào.

+ Chặn: Một thuật ngữ không đổi trong mô hình hồi quy logistic, biểu thị tỷ lệ logarit khi tất cả các biến độc lập đều bằng 0.

+ Ước tính khả năng tối đa: Phương pháp được sử dụng để ước tính các hệ số của mô hình hồi quy logistic, giúp tối đa hóa khả năng quan sát dữ liệu được cung cấp cho mô hình.

- Hồi quy logistic hoạt động như thế nào?

Mô hình hồi quy logistic biến đổi đầu ra giá trị liên tục của hàm hồi quy tuyến tính thành đầu ra giá trị phân loại bằng cách sử dụng hàm sigmoid, hàm này ánh xạ bất kỳ tập hợp biến độc lập có giá trị thực nào đầu vào thành giá trị từ 0 đến 1. Hàm này được gọi là hàm logistic.

Đặt các tính năng đầu vào độc lập

$$X = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ x_{21} & \dots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \end{bmatrix}$$

và biến phụ thuộc là Y chỉ có giá trị nhị phân tức là 0 hoặc 1

$$Y = \begin{cases} 0 & \text{if } Class\ 1 \\ 1 & \text{if } Class\ 2 \end{cases}$$

sau đó áp dụng hàm đa tuyến tính cho các biến đầu vào X

$$z = (\sum_{i=1}^n w_i x_i) + b$$

$x_i$  là quan sát thứ i của X

$w_i = [w_1, w_2, w_3, \dots, w_m]$  là trọng số hoặc Hệ số và b là thuật ngữ độ lệch còn được gọi là phần chặn. đơn giản điều này có thể được biểu diễn dưới dạng tích số chấm của trọng số và độ lệch.

$$z = w \cdot X + b$$

VD1:

```

# import the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load the breast cancer dataset
X, y = load_breast_cancer(return_X_y=True)
# split the train and test dataset
X_train, X_test, \
    y_train, y_test = train_test_split(X, y,
                                       test_size=0.20,
                                       random_state=23)

# LogisticRegression
clf = LogisticRegression(random_state=0)
clf.fit(X_train, y_train)
# Prediction
y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):", acc*100)

```

**Logistic Regression model accuracy (in %): 94.73684210526315**

\* Giải thích code:

Đoạn code này thực hiện việc sử dụng Logistic Regression để xây dựng một mô hình phân loại trên tập dữ liệu ung thư vú (breast cancer) từ thư viện scikit-learn. Dưới đây là giải thích từng phần của đoạn code:

Import các thư viện và modules cần thiết:

load\_breast\_cancer từ sklearn.datasets: Dùng để tải tập dữ liệu ung thư vú.

LogisticRegression từ sklearn.linear\_model: Dùng để tạo mô hình Logistic Regression.

train\_test\_split từ sklearn.model\_selection: Dùng để chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.

accuracy\_score từ sklearn.metrics: Dùng để tính toán độ chính xác của mô hình.

Tải tập dữ liệu ung thư vú bằng load\_breast\_cancer và gán nó vào biến X cho các đặc trưng và y cho các nhãn lớp.

Chia tập dữ liệu thành tập huấn luyện (X\_train và y\_train) và tập kiểm tra (X\_test và y\_test) bằng cách sử dụng train\_test\_split. Trong trường hợp này, 80% dữ liệu được sử dụng cho huấn luyện và 20% cho kiểm tra.

Tạo một mô hình Logistic Regression bằng cách sử dụng LogisticRegression với random\_state=0 để đảm bảo kết quả nhất quán.

Huấn luyện mô hình Logistic Regression trên tập huấn luyện bằng cách sử dụng fit.

Sử dụng mô hình đã huấn luyện để dự đoán lớp cho tập kiểm tra bằng cách sử dụng predict.



Tính toán độ chính xác của mô hình bằng cách so sánh dự đoán ( $y_{pred}$ ) với nhãn thực tế ( $y_{test}$ ) và in ra màn hình.

Kết quả cuối cùng sẽ là độ chính xác của mô hình Logistic Regression trên tập kiểm tra, được in ra dưới dạng phần trăm. Điều này thể hiện khả năng phân loại của mô hình trên tập dữ liệu ung thư vú.

VD2:

```
from sklearn.model_selection import train_test_split
from sklearn import datasets, linear_model, metrics

# Load the digit dataset
digits = datasets.load_digits()

# defining feature matrix(X) and response vector(y)
X = digits.data
y = digits.target

# splitting X and y into training and testing sets
X_train, X_test, \
    y_train, y_test = train_test_split(X, y,
                                       test_size=0.4,
                                       random_state=1)

# create logistic regression object
reg = linear_model.LogisticRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# making predictions on the testing set
y_pred = reg.predict(X_test)

# comparing actual response values (y_test)
# with predicted response values (y_pred)
print("Logistic Regression model accuracy(in %):",
      metrics.accuracy_score(y_test, y_pred)*100)
```

```
Logistic Regression model accuracy(in %): 96.52294853963839
```

\* Giải thích code:

Đoạn code này thực hiện việc sử dụng Logistic Regression để xây dựng một mô hình phân loại trên tập dữ liệu "digits" (chữ số viết tay) từ thư viện scikit-learn. Dưới đây là giải thích từng phần của đoạn code:

Import các thư viện và modules cần thiết:

`train_test_split` từ `sklearn.model_selection`: Dùng để chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.

`datasets` từ `sklearn`: Dùng để tải tập dữ liệu "digits".

linear\_model từ sklearn: Dùng để tạo mô hình Logistic Regression.

metrics từ sklearn: Dùng để tính toán các chỉ số đánh giá mô hình.

Tải tập dữ liệu "digits" bằng datasets.load\_digits() và gán nó vào biến digits.

Xác định ma trận đặc trưng (feature matrix) X và vector phản ứng (response vector) y từ dữ liệu "digits". X chứa các hình ảnh chữ số viết tay và y chứa nhãn (chữ số tương ứng) của từng hình ảnh.

Chia tập dữ liệu thành tập huấn luyện (X\_train và y\_train) và tập kiểm tra (X\_test và y\_test) bằng train\_test\_split. Trong trường hợp này, 60% dữ liệu được sử dụng cho huấn luyện và 40% cho kiểm tra.

Tạo một mô hình Logistic Regression bằng cách sử dụng linear\_model.LogisticRegression() và gán nó vào biến reg.

Huấn luyện mô hình Logistic Regression trên tập huấn luyện bằng cách sử dụng fit.

Sử dụng mô hình đã huấn luyện để dự đoán lớp cho tập kiểm tra bằng cách sử dụng predict và gán kết quả vào biến y\_pred.

Sử dụng metrics.accuracy\_score để tính toán độ chính xác của mô hình bằng cách so sánh kết quả dự đoán (y\_pred) với nhãn thực tế (y\_test). Kết quả được in ra dưới dạng phần trăm.

Kết quả cuối cùng sẽ là độ chính xác của mô hình Logistic Regression trên tập kiểm tra, thể hiện khả năng phân loại các chữ số viết tay từ tập dữ liệu "digits."

## 2.10

\* SVM là gì ?

- SVM (Support Vector Machine) là một thuật toán máy học được sử dụng trong các bài toán phân loại và hồi quy. Nó là một trong những thuật toán phổ biến và mạnh mẽ trong lĩnh vực học máy. SVM được sử dụng rộng rãi trong nhiều lĩnh vực như xử lý ảnh, xử lý ngôn ngữ tự nhiên, phát hiện spam email, phân loại văn bản, dự đoán, và nhiều ứng dụng khác.

- Đặc điểm quan trọng của SVM bao gồm:

Phân loại tuyến tính và phi tuyến tính: SVM có thể áp dụng cho cả bài toán phân loại tuyến tính và phi tuyến tính. Trong trường hợp phân loại tuyến tính, SVM tìm một đường biên (hyperplane) tốt nhất để phân tách giữa các lớp dữ liệu. Trong trường hợp phân loại phi tuyến tính, SVM sử dụng các hàm biến đổi (kernel functions) để ánh xạ dữ liệu vào một không gian cao chiều hơn để có thể tìm ra đường biên tốt nhất.

**Hỗ trợ vector:** SVM tập trung vào các điểm dữ liệu gần đường biên của lớp, được gọi là "hỗ trợ vector." Các hỗ trợ vector chính xác định vị trí của đường biên và có vai trò quan trọng trong việc xác định lựa chọn đường biên tối ưu.

**Tối ưu hóa margin:** Mục tiêu chính của SVM là tối ưu hóa margin, tức là khoảng cách giữa đường biên và các điểm dữ liệu gần nhất của hai lớp. Điều này giúp tăng khả năng tổng quát hóa của mô hình và giảm nguy cơ overfitting.

**Cách làm việc với dữ liệu không cân bằng:** SVM có khả năng làm việc tốt với dữ liệu không cân bằng, tức là khi có một lớp dữ liệu có số lượng quan sát lớn hơn rất nhiều so với lớp khác.

**Kernel trick:** SVM sử dụng kernel trick để ánh xạ dữ liệu vào không gian cao chiều hơn mà không cần tính toán tất cả các chi tiết trong không gian đó. Điều này giúp SVM làm việc tốt trên dữ liệu phi tuyến tính.

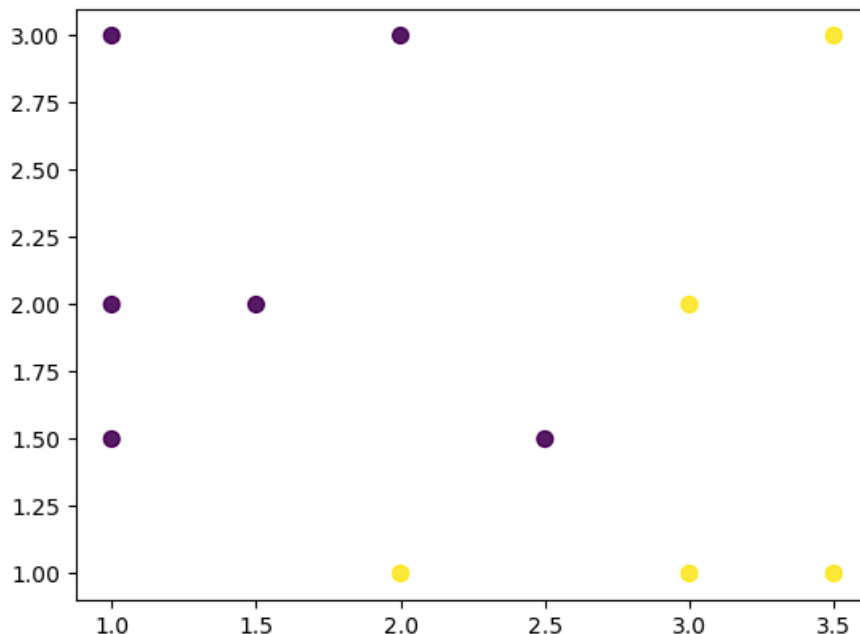
SVM là một thuật toán mạnh mẽ với khả năng phân loại tốt, đặc biệt trong các bài toán có dữ liệu tương đối lớn và phức tạp. Tuy nhiên, SVM cũng đòi hỏi nắm vững kiến thức về cách chọn kernel và các siêu tham số liên quan để có thể đạt được kết quả tốt nhất.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# Tạo dữ liệu và nhãn
x = np.array([[1, 3], [1, 2], [1, 1.5], [1.5, 2], [2, 3], [2.5, 1.5], [2, 1], [3, 1], [3, 2], [3.5, 1], [3.5, 3]])
y = [0] * 6 + [1] * 5

# Vẽ biểu đồ scatter plot với c và s tương ứng với nhãn và kích thước điểm dữ liệu
plt.scatter(x[:, 0], x[:, 1], c=y, s=50, alpha=0.9)

# Hiển thị biểu đồ
plt.show()
```



\* Giải thích:

Dòng 1-3: Nhập các thư viện cần thiết.

Dòng 5-6: Tạo một mảng x chứa các điểm dữ liệu với hai đặc trưng (hoành độ và tung độ).

Dòng 7: Tạo một mảng y chứa nhãn của các điểm dữ liệu. Trong trường hợp này, 6 điểm đầu tiên thuộc nhóm 0 và 5 điểm cuối thuộc nhóm 1.

Dòng 9: Vẽ biểu đồ scatter plot.  $x[:, 0]$  và  $x[:, 1]$  là hoành độ và tung độ của các điểm dữ liệu.  $c=y$  là để chỉ định màu sắc của điểm dữ liệu dựa trên nhãn từ mảng y.  $s=50$  là kích thước của điểm trên biểu đồ.  $\alpha=0.9$  là độ trong suốt của điểm trên biểu đồ.

Dòng 11: Hiện thị biểu đồ.

## 2.11

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

# Tải dữ liệu về tiểu đường từ datasets
diabetes = datasets.load_diabetes()
x_train = diabetes.data[:-20]
y_train = diabetes.target[:-20]
x_test = diabetes.data[-20:]
y_test = diabetes.target[-20:]

# Chọn đặc trưng thứ 3 (0-indexed) từ tập dữ liệu
x0_test = x_test[:, 2]
x0_train = x_train[:, 2]

# Chuyển đổi đặc trưng thành mảng 2D
x0_test = x0_test[:, np.newaxis]
x0_train = x0_train[:, np.newaxis]

# Sắp xếp mảng x0_test theo trục dọc (axis=0)
x0_test.sort(axis=0)

# Tăng giá trị của các điểm lên 100 lần
x0_test = x0_test * 100
x0_train = x0_train * 100

# Tạo mô hình Support Vector Regression với kernel tuyến tính và tham số C=1000
svr = svm.SVR(kernel='linear', C=1000)

# Tạo mô hình Support Vector Regression với kernel đa thức bậc 2 và tham số C=1000
svr2 = svm.SVR(kernel='poly', C=1000, degree=2)

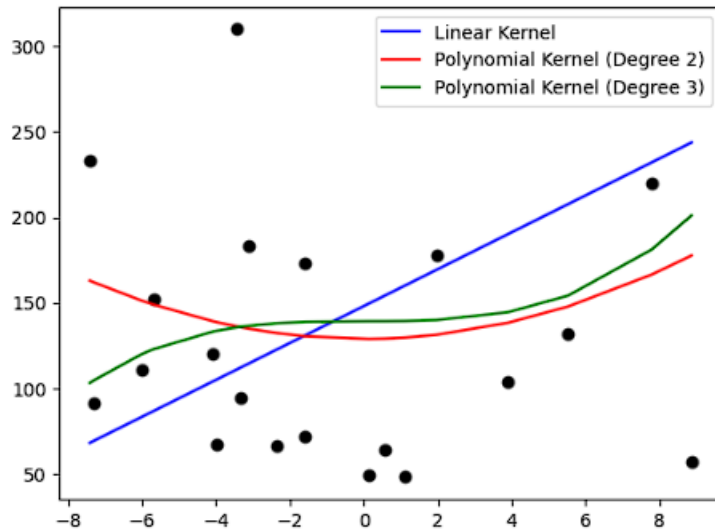
# Tạo mô hình Support Vector Regression với kernel đa thức bậc 3 và tham số C=1000
svr3 = svm.SVR(kernel='poly', C=1000, degree=3)

# Huấn luyện các mô hình SVR bằng dữ liệu huấn luyện
svr.fit(x0_train, y_train)
svr2.fit(x0_train, y_train)
svr3.fit(x0_train, y_train)

# Dự đoán giá trị của các mẫu kiểm tra bằng các mô hình đã huấn luyện
y = svr.predict(x0_test)
y2 = svr2.predict(x0_test)
y3 = svr3.predict(x0_test)

# Vẽ biểu đồ scatter plot của dữ liệu kiểm tra và đường hồi quy của ba mô hình khác nhau
plt.scatter(x0_test, y_test, color='k') # Dữ liệu kiểm tra
plt.plot(x0_test, y, color='b', label='Linear Kernel') # Mô hình với kernel tuyến tính
plt.plot(x0_test, y2, color='r', label='Polynomial Kernel (Degree 2)') # Mô hình với kernel đa thức bậc 2
plt.plot(x0_test, y3, color='g', label='Polynomial Kernel (Degree 3)') # Mô hình với kernel đa thức bậc 3

# Thêm chú thích và hiện thị biểu đồ
plt.legend()
plt.show()
```



\* Giải thích:

Dòng 1-3: Nhập các thư viện cần thiết.

Dòng 5-10: Tải dữ liệu về tiểu đường từ dataset mẫu và chia thành tập huấn luyện và tập kiểm tra.

Dòng 12-13: Chọn đặc trưng thứ 3 từ tập dữ liệu và chuyển đổi chúng thành mảng 2D.

Dòng 15: Sắp xếp mảng `x0_test` theo trục dọc để chuẩn bị cho việc vẽ biểu đồ.

Dòng 17-18: Tăng giá trị của các điểm lên 100 lần để thay đổi tỷ lệ trên biểu đồ.

Dòng 20-22: Tạo ba mô hình Support Vector Regression với các tham số khác nhau, bao gồm kernel tuyến tính và kernel đa thức bậc 2 và 3.

Dòng 24-26: Huấn luyện ba mô hình SVR bằng dữ liệu huấn luyện.

Dòng 28-30: Dự đoán giá trị của các mẫu kiểm tra bằng ba mô hình đã huấn luyện.

Dòng 32-35: Vẽ biểu đồ scatter plot của dữ liệu kiểm tra và đường hồi quy của ba mô hình khác nhau. Các đường hồi quy có màu khác nhau và được ghi chú trên biểu đồ.

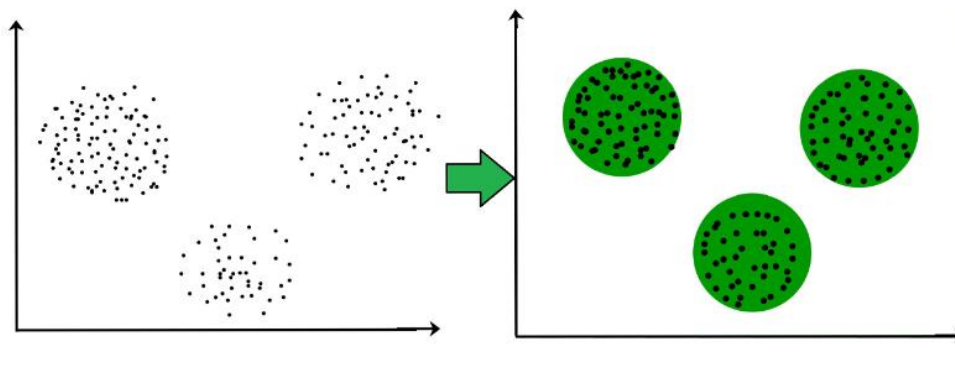
## 2.12

\* Clustering là gì?

- Giới thiệu về Phân cụm: Về cơ bản nó là một loại phương pháp học không giám sát. Phương pháp học không giám sát là phương pháp trong đó chúng tôi rút ra các tham chiếu từ bộ dữ liệu bao gồm dữ liệu đầu vào mà không có phản hồi được gắn nhãn. Nói chung, nó được sử dụng như một quá trình để tìm ra cấu trúc có ý nghĩa, các quy trình cơ bản có tính giải thích, các đặc điểm tổng quát và các nhóm vốn có trong một tập hợp các ví dụ.

- Phân cụm là nhiệm vụ chia tổng thể hoặc các điểm dữ liệu thành một số nhóm sao cho các điểm dữ liệu trong cùng một nhóm giống với các điểm dữ liệu khác trong cùng một nhóm và không giống với các điểm dữ liệu trong các nhóm khác. Về cơ bản nó là một tập hợp các đối tượng trên cơ sở sự giống nhau và khác nhau giữa chúng.

Ví dụ: Các điểm dữ liệu trong biểu đồ bên dưới được nhóm lại với nhau có thể được phân loại thành một nhóm duy nhất. Chúng ta có thể phân biệt các cụm và xác định được có 3 cụm trong hình bên dưới.



- DBSCAN: Phân cụm không gian dựa trên mật độ các ứng dụng có nhiều

Những điểm dữ liệu này được phân cụm bằng cách sử dụng khái niệm cơ bản rằng điểm dữ liệu nằm trong giới hạn nhất định từ trung tâm cụm. Các phương pháp và kỹ thuật khoảng cách khác nhau được sử dụng để tính toán các giá trị ngoại lệ.

- Tại sao phải phân cụm?

Việc phân cụm rất quan trọng vì nó quyết định việc phân nhóm nội tại giữa các dữ liệu chưa được gắn nhãn hiện có. Không có tiêu chí nào cho việc phân cụm tốt. Nó phụ thuộc vào người dùng và tiêu chí nào họ có thể sử dụng để đáp ứng nhu cầu của họ. Ví dụ: chúng ta có thể quan tâm đến việc tìm đại diện cho các nhóm đồng nhất (giảm dữ liệu), tìm “cụm tự nhiên” và mô tả các thuộc tính chưa biết của chúng (loại dữ liệu “tự nhiên”), tìm các nhóm hữu ích và phù hợp (các lớp dữ liệu “hữu ích”) hoặc trong việc tìm kiếm các đối tượng dữ liệu bất thường (phát hiện ngoại lệ). Thuật toán này phải đưa ra một số giả định cấu thành nên sự giống nhau của các điểm và mỗi giả định tạo ra các cụm khác nhau và có giá trị như nhau.

- Phương pháp phân cụm:

+ Phương pháp dựa trên mật độ: Các phương pháp này coi các cụm là vùng dày đặc có một số điểm tương đồng và khác biệt so với vùng dày đặc thấp hơn của không gian. Các phương pháp này có độ chính xác tốt và khả năng hợp nhất hai cụm. Ví dụ DBSCAN (Phân cụm ứng dụng không gian dựa trên mật độ có nhiều), OPTICS (Sắp xếp các điểm để xác định cấu trúc phân cụm), v.v.

+ Phương pháp dựa trên phân cấp: Các cụm được hình thành trong phương pháp này tạo thành cấu trúc kiểu cây dựa trên phân cấp. Các cụm mới được hình thành bằng cách sử dụng cụm đã hình thành trước đó. Nó được chia thành hai loại

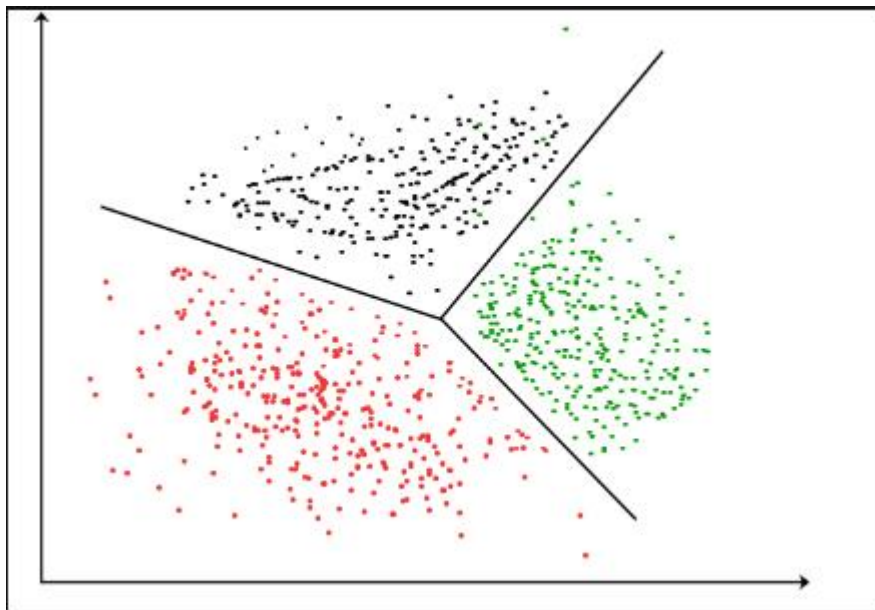
- Tích tụ (cách tiếp cận từ dưới lên)

- Phân chia (cách tiếp cận từ trên xuống)

- Phương pháp phân vùng: Các phương pháp này phân chia các đối tượng thành k cụm và mỗi phân vùng tạo thành một cụm. Phương pháp này được sử dụng để tối ưu hóa hàm tương tự tiêu chí khách quan, chẳng hạn như khi khoảng cách là tham số chính, ví dụ K-means, CLARANS (Phân cụm các ứng dụng lớn dựa trên tìm kiếm ngẫu nhiên), v.v.

- Phương pháp dựa trên lưới: Trong phương pháp này, không gian dữ liệu được tạo thành một số lượng ô hữu hạn tạo thành cấu trúc giống như lưới. Tất cả các hoạt động phân cụm được thực hiện trên các lưới này đều nhanh và độc lập với số lượng đối tượng dữ liệu, ví dụ STING (Lưới thông tin thông kê), cụm sóng, CLIQUE (CLustering In Quest), v.v.

- Thuật toán phân cụm: Thuật toán phân cụm K-mean – Đây là thuật toán học không giám sát đơn giản nhất để giải quyết vấn đề phân cụm. Thuật toán K-means phân chia n quan sát thành k cụm trong đó mỗi quan sát thuộc về cụm có giá trị trung bình gần nhất đóng vai trò là nguyên mẫu của cụm.



- Ứng dụng của Clustering trong các lĩnh vực khác nhau:

Tiếp thị: Nó có thể được sử dụng để mô tả và khám phá các phân khúc khách hàng cho mục đích tiếp thị.

Sinh học: Nó có thể được sử dụng để phân loại giữa các loài thực vật và động vật khác nhau.

**Thư viện:** Nó được sử dụng để phân cụm các cuốn sách khác nhau trên cơ sở các chủ đề và thông tin.

**Bảo hiểm:** Nó được sử dụng để thừa nhận khách hàng, chính sách của họ và xác định các gian lận.

**Quy hoạch thành phố:** Nó được sử dụng để tạo ra các nhóm nhà và nghiên cứu giá trị của chúng dựa trên vị trí địa lý và các yếu tố khác hiện có.

**Nghiên cứu động đất:** Bằng cách tìm hiểu các khu vực bị ảnh hưởng bởi động đất, chúng ta có thể xác định các khu vực nguy hiểm.

**Xử lý hình ảnh:** Phân cụm có thể được sử dụng để nhóm các hình ảnh tương tự lại với nhau, phân loại hình ảnh dựa trên nội dung và xác định các mẫu trong dữ liệu hình ảnh.

**Di truyền học:** Phân cụm được sử dụng để nhóm các gen có kiểu biểu hiện tương tự nhau và xác định các mạng lưới gen hoạt động cùng nhau trong các quá trình sinh học.

**Tài chính:** Phân cụm được sử dụng để xác định các phân khúc thị trường dựa trên hành vi của khách hàng, xác định các mẫu trong dữ liệu thị trường chứng khoán và phân tích rủi ro trong danh mục đầu tư.

**Dịch vụ khách hàng:** Phân cụm được sử dụng để nhóm các yêu cầu và khiếu nại của khách hàng thành các danh mục, xác định các vấn đề phổ biến và phát triển các giải pháp có mục tiêu.

**Sản xuất:** Phân cụm được sử dụng để nhóm các sản phẩm tương tự lại với nhau, tối ưu hóa quy trình sản xuất và xác định các khiếm khuyết trong quy trình sản xuất.

**Chẩn đoán y tế:** Phân cụm được sử dụng để nhóm các bệnh nhân có triệu chứng hoặc bệnh tương tự, giúp đưa ra chẩn đoán chính xác và xác định phương pháp điều trị hiệu quả.

**Phát hiện gian lận:** Phân cụm được sử dụng để xác định các mô hình đáng ngờ hoặc điểm bất thường trong giao dịch tài chính, có thể giúp phát hiện gian lận hoặc các tội phạm tài chính khác.

**Phân tích giao thông:** Phân cụm được sử dụng để nhóm các mẫu dữ liệu giao thông tương tự nhau, chẳng hạn như giờ cao điểm, tuyến đường và tốc độ, có thể giúp cải thiện quy hoạch và cơ sở hạ tầng giao thông.

**Phân tích mạng xã hội:** Phân cụm được sử dụng để xác định các cộng đồng hoặc nhóm trong mạng xã hội, điều này có thể giúp hiểu được hành vi, ảnh hưởng và xu hướng xã hội.

**An ninh mạng:** Phân cụm được sử dụng để nhóm các mô hình lưu lượng mạng hoặc hành vi hệ thống tương tự nhau, có thể giúp phát hiện và ngăn chặn các cuộc tấn công mạng.



**Phân tích khí hậu:** Phân cụm được sử dụng để nhóm các mẫu dữ liệu khí hậu tương tự, chẳng hạn như nhiệt độ, lượng mưa và gió, có thể giúp hiểu rõ về biến đổi khí hậu và tác động của nó đối với môi trường.

**Phân tích thể thao:** Phân cụm được sử dụng để nhóm các mẫu dữ liệu hiệu suất của cầu thủ hoặc đội tương tự, có thể giúp phân tích điểm mạnh và điểm yếu của cầu thủ hoặc đội và đưa ra các quyết định chiến lược.

**Phân tích tội phạm:** Phân cụm được sử dụng để nhóm các mẫu dữ liệu tội phạm tương tự, chẳng hạn như địa điểm, thời gian và loại, có thể giúp xác định các điểm nóng tội phạm, dự đoán xu hướng tội phạm trong tương lai và cải thiện chiến lược phòng chống tội phạm.

\* Ví dụ:

Chúng ta sử dụng hàm `make_blobs` từ `scikit-learn` để tạo ra dữ liệu mẫu gồm 300 điểm dữ liệu và 4 clusters (centers).

Chúng ta tạo một mô hình K-Means clustering với 4 clusters bằng cách sử dụng `KMeans(n_clusters=4)`.

Mô hình được huấn luyện trên dữ liệu sử dụng `kmeans.fit(data)`.

Chúng ta dự đoán nhãn của từng điểm dữ liệu bằng cách sử dụng `kmeans.predict(data)`.

Các tọa độ của các cluster centers được trích xuất bằng cách sử dụng `kmeans.cluster_centers_`.

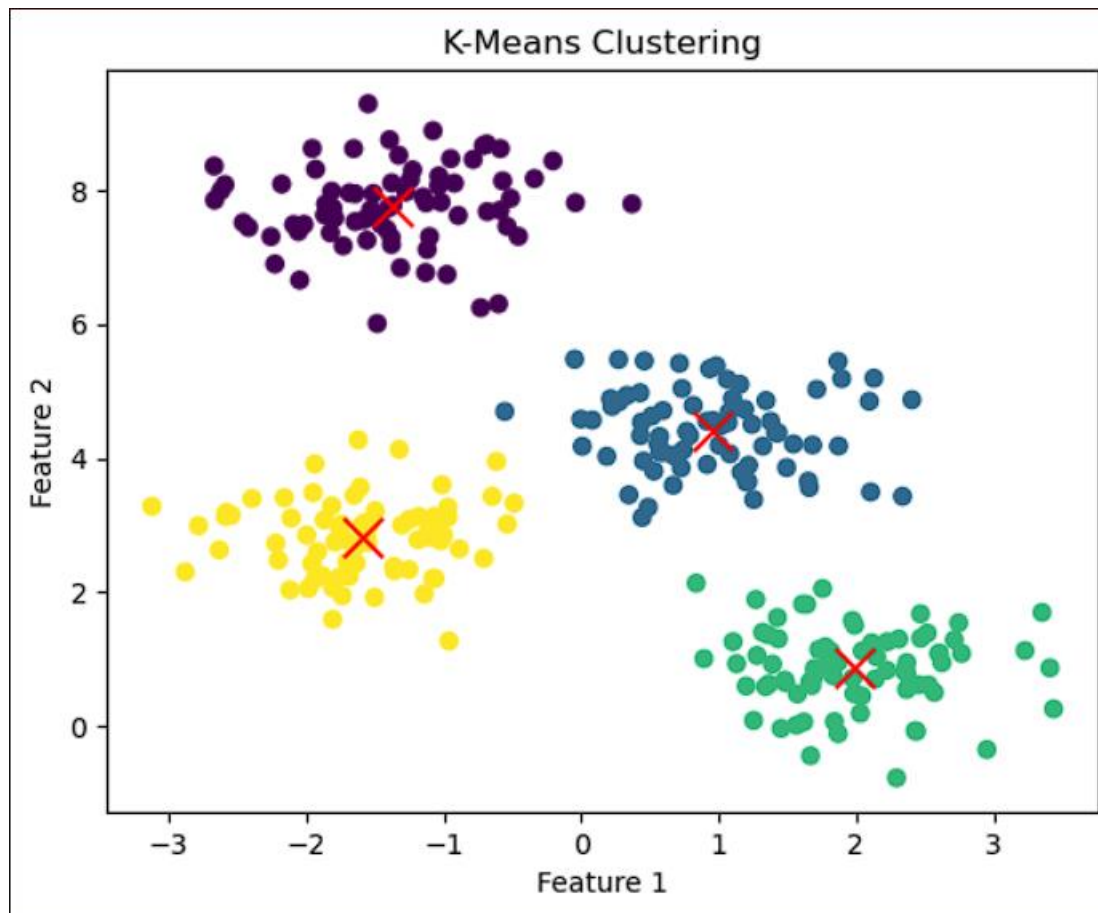
Cuối cùng, chúng ta vẽ biểu đồ scatter plot để hiển thị dữ liệu, các cluster centers (đánh dấu bằng dấu "x" đỏ), và màu sắc tương ứng với cluster của từng điểm dữ liệu.

Kết quả sẽ là một biểu đồ hiển thị các cluster và cluster centers trong dữ liệu mẫu

- Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Tạo dữ liệu mẫu
data, _ = make_blobs(n_samples=300, centers=4, random_state=0, cluster_std=0.60)

# Tạo mô hình K-Means với 4 clusters
kmeans = KMeans(n_clusters=4)
# Huấn luyện mô hình trên dữ liệu
kmeans.fit(data)
# Dự đoán nhãn của từng điểm dữ liệu
labels = kmeans.predict(data)
# Trích xuất các tọa độ của các cluster centers
centers = kmeans.cluster_centers_
# Vẽ biểu đồ dữ liệu và các cluster centers
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=200)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("K-Means Clustering")
plt.show()
```



## 2.13 Deploy machine

### Loading the Data

```
import numpy as np
import pandas as pd

df = pd.read_csv('diabetes.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   Pregnancies            768 non-null   int64  
1   Glucose                768 non-null   int64  
2   BloodPressure          768 non-null   int64  
3   SkinThickness          768 non-null   int64  
4   Insulin                768 non-null   int64  
5   BMI                    768 non-null   float64 
6   DiabetesPedigreeFunction 768 non-null   float64 
7   Age                   768 non-null   int64  
8   Outcome                768 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

## Cleaning the Data

```
print("Nulls")
print("=====")
print(df.isnull().sum())
```

```
Nulls
=====
Pregnancies           0
Glucose               0
BloodPressure         0
SkinThickness         0
Insulin              0
BMI                  0
DiabetesPedigreeFunction  0
Age                  0
Outcome              0
dtype: int64
```

```
print("0s")
print("==")
print(df.eq(0).sum())
```

```
0s
==
Pregnancies           111
Glucose                5
BloodPressure          35
SkinThickness         227
Insulin              374
BMI                   11
DiabetesPedigreeFunction  0
Age                   0
Outcome              500
dtype: int64
```

```
df[['Glucose', 'BloodPressure', 'SkinThickness',
      'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']] = \
df[['Glucose', 'BloodPressure', 'SkinThickness',
      'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']].replace(0, np.NaN)
```

```
df.fillna(df.mean(), inplace = True) # replace NaN with the mean
```

```
print(df.eq(0).sum())
```

```
Pregnancies           111
Glucose                0
BloodPressure          0
SkinThickness          0
Insulin                0
BMI                    0
DiabetesPedigreeFunction  0
Age                    0
Outcome              500
dtype: int64
```

## Examining the Correlation Between the Features

```
corr = df.corr()  
print(corr)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	\
Pregnancies	1.000000	0.127911	0.208522	0.082989	
Glucose	0.127911	1.000000	0.218367	0.192991	
BloodPressure	0.208522	0.218367	1.000000	0.192816	
SkinThickness	0.082989	0.192991	0.192816	1.000000	
Insulin	0.056027	0.420157	0.072517	0.158139	
BMI	0.021565	0.230941	0.281268	0.542398	
DiabetesPedigreeFunction	-0.033523	0.137060	-0.002763	0.100966	
Age	0.544341	0.266534	0.324595	0.127872	
Outcome	0.221898	0.492928	0.166074	0.215299	

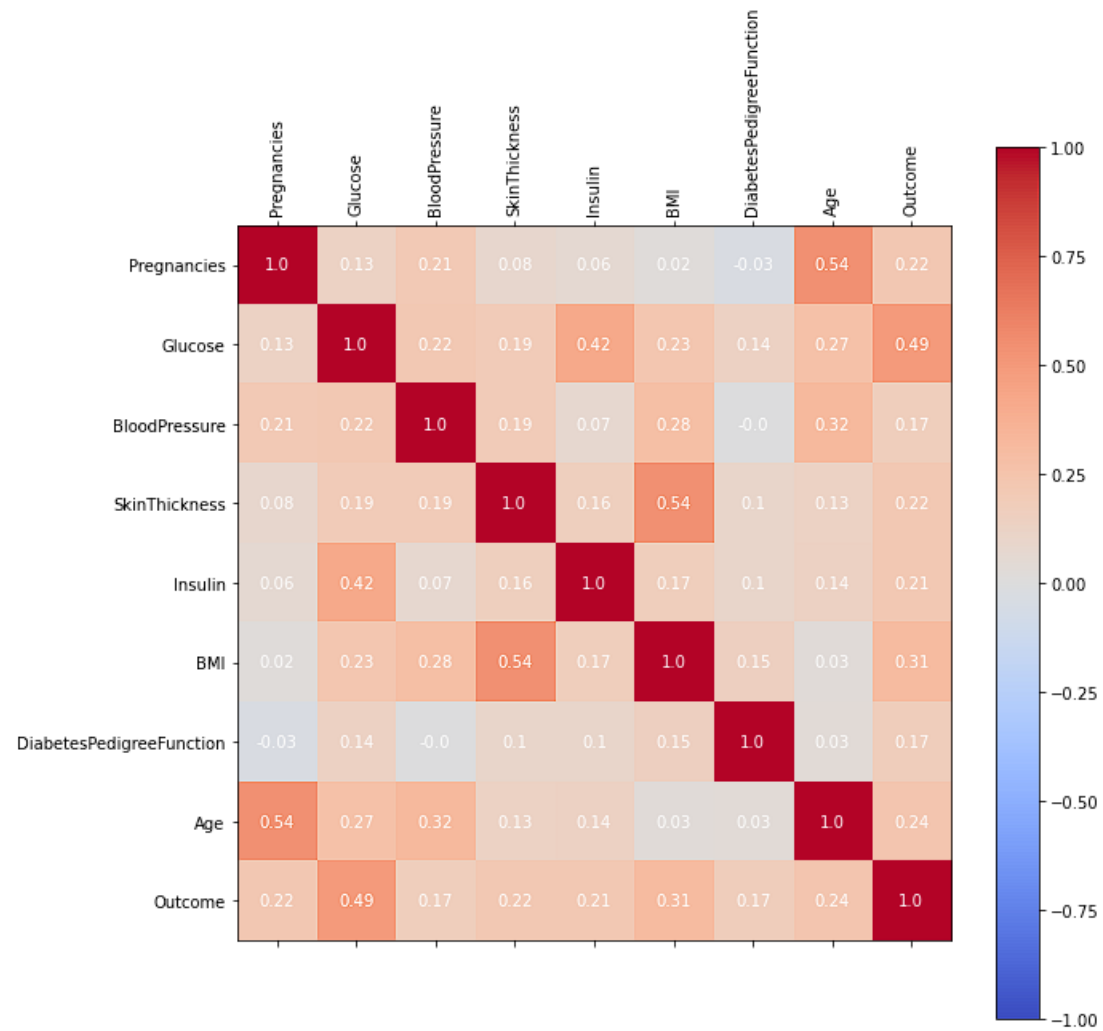
	Insulin	BMI	DiabetesPedigreeFunction	\
Pregnancies	0.056027	0.021565	-0.033523	
Glucose	0.420157	0.230941	0.137060	
BloodPressure	0.072517	0.281268	-0.002763	
SkinThickness	0.158139	0.542398	0.100966	
Insulin	1.000000	0.166586	0.098634	
BMI	0.166586	1.000000	0.153400	
DiabetesPedigreeFunction	0.098634	0.153400	1.000000	
Age	0.136734	0.025519	0.033561	
Outcome	0.214411	0.311924	0.173844	

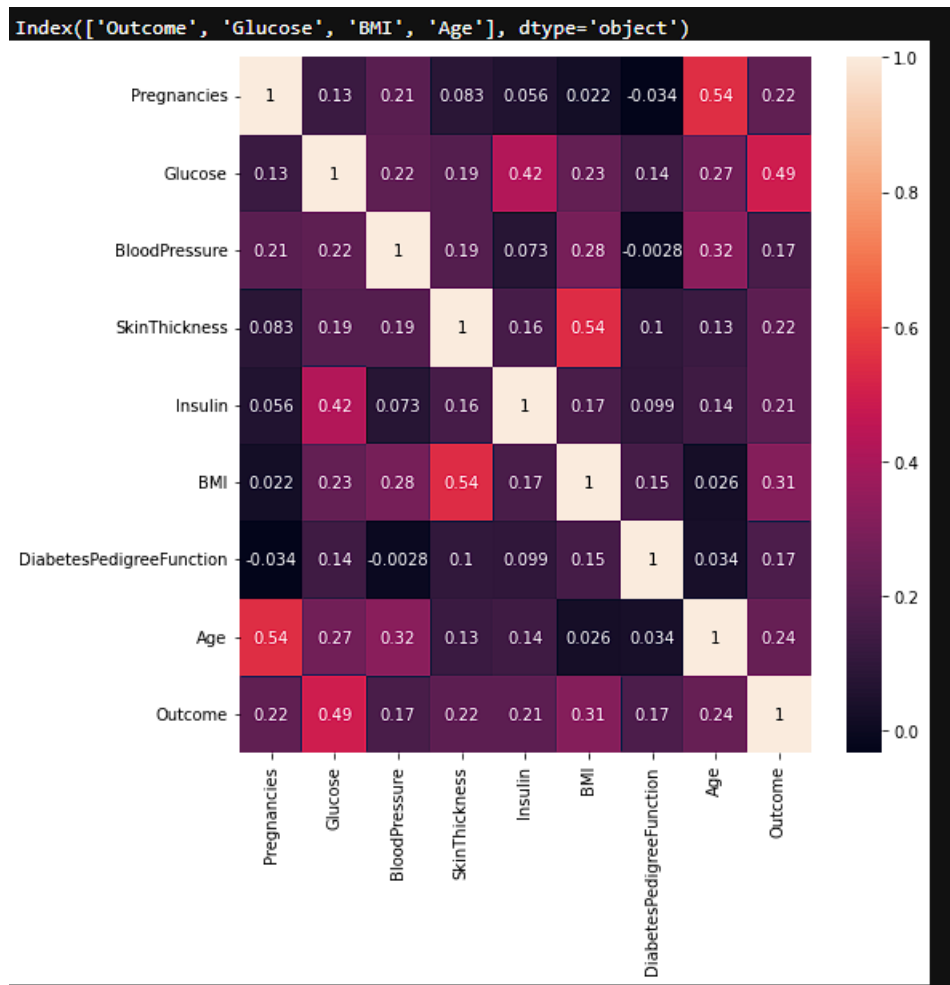
	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.266534	0.492928
BloodPressure	0.324595	0.166074
SkinThickness	0.127872	0.215299
Insulin	0.136734	0.214411
BMI	0.025519	0.311924
DiabetesPedigreeFunction	0.033561	0.173844
Age	1.000000	0.238356
Outcome	0.238356	1.000000

## Plotting the Correlation Between Features

```
%matplotlib inline  
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()
```



```
import seaborn as sns
sns.heatmap(df.corr(),annot=True)
#---get a reference to the current figure and set its size---
fig = plt.gcf()
fig.set_size_inches(8,8)
#---get the top four features that has the highest correlation---
print(df.corr().nlargest(4, 'Outcome').index)
```



```
#---print the top 4 correlation values---
print(df.corr().nlargest(4, 'Outcome').values[:,8])
[1.          0.49292767  0.31192439  0.23835598]
```

## Evaluating the Algorithms

### \* Logistic Regression

```
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)

0.7669856459330144

result = []
result.append(log_regress_score)
```

## \* K-Nearest Neighbors

```
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)
```

The optimal number of neighbors is 19  
0.7721462747778537

## \* Support Vector Machines

```
from sklearn import svm
linear_svm = svm.SVC(kernel='linear')
linear_svm_score = cross_val_score(linear_svm, X, y,
cv=10, scoring='accuracy').mean()
print(linear_svm_score)
result.append(linear_svm_score)
```

0.7656527682843473

```
rbf = svm.SVC(kernel='rbf')
rbf_score = cross_val_score(rbf, X, y, cv=10, scoring='accuracy').mean()
print(rbf_score)
result.append(rbf_score)
```

0.765704032809296

## Selecting the Best Performing Algorithm

```
algorithms = ["Logistic Regression", "K Nearest Neighbors", "SVM Linear Kernel", "SVM RBF Kernel"]
cv_mean = pd.DataFrame(result,index = algorithms)
cv_mean.columns=["Accuracy"]
cv_mean.sort_values(by="Accuracy",ascending=False)
```

	Accuracy
K Nearest Neighbors	0.772146
Logistic Regression	0.766986
SVM RBF Kernel	0.765704
SVM Linear Kernel	0.765653

## Training and Saving the Model

```
knn = KNeighborsClassifier(n_neighbors=19)
knn.fit(X, y)
```

```
KNeighborsClassifier(n_neighbors=19)
```

```
import pickle
#---save the model to disk---
filename = 'diabetes.sav'
#---write to the file using write and binary mode---
pickle.dump(knn, open(filename, 'wb'))
```

```
#---load the model from disk---
loaded_model = pickle.load(open(filename, 'rb'))
```

```
Glucose = 65
BMI = 70
Age = 50

prediction = loaded_model.predict([[Glucose, BMI, Age]])
print(prediction)
if (prediction[0]==0):
    print("Non-diabetic")
else:
    print("Diabetic")
```

```
[0]
Non-diabetic
```

```
proba = loaded_model.predict_proba([[Glucose, BMI, Age]])
print(proba)
print("Confidence: " + str(round(np.amax(proba[0]) * 100 ,2)) + "%")
```

```
[[0.94736842 0.05263158]]
Confidence: 94.74%
```



## Deploying the Model

```
pip install flask
```

```
import pickle
from flask import Flask, request, json, jsonify
import numpy as np

app = Flask(__name__)

#---the filename of the saved model---
filename = 'diabetes.sav'

#---load the saved model---
loaded_model = pickle.load(open(filename, 'rb'))

@app.route('/diabetes/v1/predict', methods=['POST'])
def predict():
    #---get the features to predict---
    features = request.json
    #---create the features list for prediction---
    features_list = [features["Glucose"],
                     features["BMI"],
                     features["Age"]]
    #---get the prediction class---
    prediction = loaded_model.predict([features_list])
    #---get the prediction probabilities---
    confidence = loaded_model.predict_proba([features_list])
    #---formulate the response to return to client---
    response = {}
    response['prediction'] = int(prediction[0])
    response['confidence'] = str(round(np.amax(confidence[0]) * 100 ,2))
    return jsonify(response)
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://10.0.0.94:5000/ (Press CTRL+C to quit)
```

## Testing the Model

```
(base) 03:49 ~ $ python REST_API.py
* Serving Flask app 'REST_API'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.0.0.94:5000
Press CTRL+C to quit
```

```
curl -H "Content-type: application/json" -X POST http://127.0.0.1:5000/diabetes/v1/predict -d '{"BMI":30, "Age":29,"Glucose":100 }'

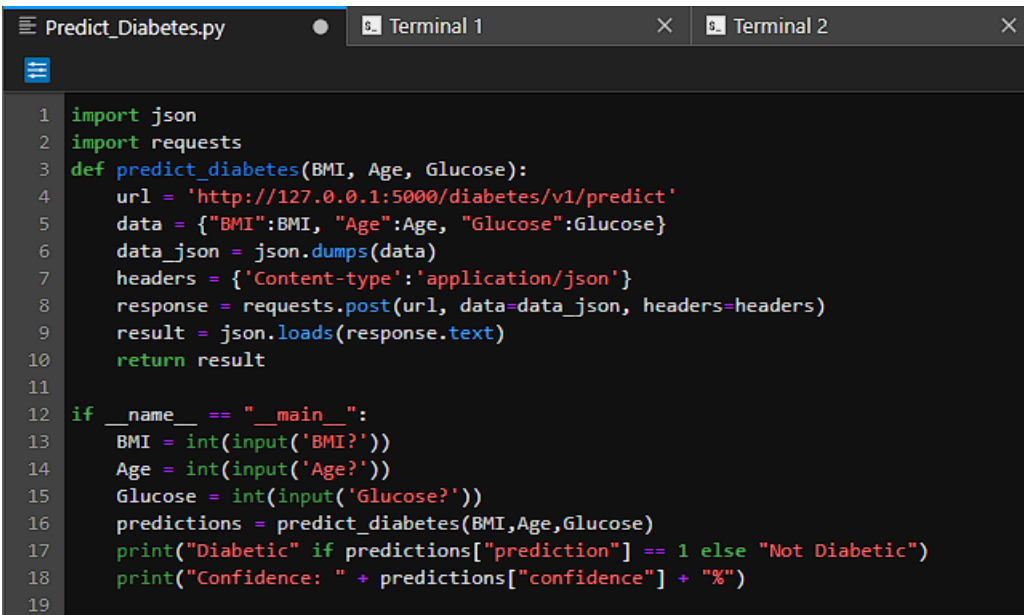
{"confidence": "78.95", "prediction": 0}
```

```
curl -H "Content-type: application/json" -X POST http://127.0.0.1:5000/diabetes/v1/predict -d '{"BMI":65, "Age":29,"Glucose":150 }'
{"confidence":"68.42","prediction":1}
```

## Creating the Client Application to Use the Model

```
import json
import requests
def predict_diabetes(BMI, Age, Glucose):
    url = 'http://127.0.0.1:5000/diabetes/v1/predict'
    data = {"BMI":BMI, "Age":Age, "Glucose":Glucose}
    data_json = json.dumps(data)
    headers = {'Content-type':'application/json'}
    response = requests.post(url, data=data_json, headers=headers)
    result = json.loads(response.text)
    return result
if __name__ == "__main__":
    predictions = predict_diabetes(30,40,100)
    print("Diabetic" if predictions["prediction"] == 1 else "Not Diabetic")
    print("Confidence: " + predictions["confidence"] + "%")
```

```
Not Diabetic
Confidence: 68.42%
```



```
Predict_Diabetes.py Terminal 1 Terminal 2
1 import json
2 import requests
3 def predict_diabetes(BMI, Age, Glucose):
4     url = 'http://127.0.0.1:5000/diabetes/v1/predict'
5     data = {"BMI":BMI, "Age":Age, "Glucose":Glucose}
6     data_json = json.dumps(data)
7     headers = {'Content-type':'application/json'}
8     response = requests.post(url, data=data_json, headers=headers)
9     result = json.loads(response.text)
10    return result
11
12 if __name__ == "__main__":
13     BMI = int(input('BMI?'))
14     Age = int(input('Age?'))
15     Glucose = int(input('Glucose?'))
16     predictions = predict_diabetes(BMI, Age, Glucose)
17     print("Diabetic" if predictions["prediction"] == 1 else "Not Diabetic")
18     print("Confidence: " + predictions["confidence"] + "%")
19
```

```
(base) 03:52 ~ $ python Predict_Diabetes.py
BMI?55
Age?29
Glucose?120
Not Diabetic
Confidence: 52.63%
```