HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

-----📖-----

TIỂU LUẬN

PHÁT TRIỂN CÁC HỆ THỐNG THÔNG MINH

Sinh viên: Trần Quang Hưng

MSV: B19DCCN333

Nhóm học phần: 01

Giảng viên : Trần Đình Quế

Hà Nội 2022

Mục lục

PHẦN I: MACHINE LEARNING CƠ BẢN

CHƯƠNG 1: XỬ LÝ DỮ LIỆU

1.1. Các thư viện hỗ trợ

1.1.1. Numpy

1.1.1.1. Giới thiệu

Trong Python, `list` là cấu trúc dữ liệu để lưu trữ dữ liệu dưới dạng mảng. Không giống với một số ngôn ngữ phổ biến khác, các phần tử trong một `list` của Python không cần cùng phải có cùng kiểu dữ liệu. Điều này thuận tiện cho ta trong lập trình nhưng lại gây nên một số vấn đề về hiệu năng (thường là về bộ nhớ).

Để cải thiện điều đó, thư viện `numpy` đã hỗ trợ những cấu trúc dữ liệu để lưu trữ các phần tử cũng dưới dạng mảng, nhưng sẽ cùng một kiểu dữ liệu.

1.1.1.2. Sử dụng

Import package:

In [354…
```
import numpy as np
```

Tạo một mảng 1 chiều có 10 số tự nhiên đầu tiên:

In [355…
```
arr = np.arange(10)
arr
```

Out[355]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Số phần tử trong từng chiều:

In [356…
```
arr.shape
```
Out[356]:
```
(10,)
```

Ta có thể tạo mảng tùy theo quy tắc:

In [357…
```
np.arange(0, 10, 2)
```
Out[357]:
```
array([0, 2, 4, 6, 8])
```

Tạo mảng 5 phần tử chỉ chứa số 0:

In [358…
```
np.zeros(5)
```
Out[358]:
```
array([0., 0., 0., 0., 0.])
```

Ta có thể tạo mảng 2 chiều:

In [359…
```
np.zeros((2, 3))
```
Out[359]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Tạo mảng nhiều chiều với giá trị khác 0:

In [360…
```
np.full((2, 3, 4), 10)
```
Out[360]:
```
array([[[10, 10, 10, 10],
        [10, 10, 10, 10],
        [10, 10, 10, 10]],

       [[10, 10, 10, 10],
        [10, 10, 10, 10],
        [10, 10, 10, 10]]])
```

Tạo ma trận đơn vị:

In [361…
```
np.eye(3)
```
Out[361]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Ta có thể tạo ra một mảng 2 chiều với các giá trị ngẫu nhiên:

In [362…
```
np.random.random((2, 4))
```
Out[362]:
```
array([[0.47216029, 0.31082282, 0.22599143, 0.77937244],
       [0.96448981, 0.81359256, 0.76946368, 0.28642487]])
```

Ta có thể tạo ra mảng `numpy` từ `list` của Python:

In [363…
```
np.array([1, 2, 3, 4, 5])
```
Out[363]:
```
array([1, 2, 3, 4, 5])
```

Truy cập phần tử trong mảng:

In [364…
```
print(arr[0], arr[1])
```

```
0 1
```

In [365…
```
arr2 = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])
print(arr2[0, 0], arr2[1, 1], arr2[2, 2])
```

```
1 5 9
```

Ta còn có thể lấy được dãy con từ mảng:

In [366…
```
arr3 = np.array([2, 3, 0, 9, 2, 0, 0, 1])
arr3[[3,4]]
```

Out[366]:
```
array([9, 2])
```

So sánh trong `numpy`:

In [367…
```
arr > 5
```

Out[367]:
```
array([False, False, False, False, False, False,  True,  True,  True,
        True])
```

Lấy mảng con theo điều kiện:

In [368…
```
arr[arr > 5]
```

Out[368]:
```
array([6, 7, 8, 9])
```

In [369…
```
arr[arr % 2 == 1]
```

Out[369]:
```
array([1, 3, 5, 7, 9])
```

Truy cập theo vùng:

In [370…
```
arr = np.array([[1,2,3,4,5],
                [4,5,6,7,8],
                [9,8,7,6,5]])
```

In [371…
```
arr[1:3, :3] # Chọn các phần tử hàng 1 tới 3 (không bao gồm) và trong 3 cột đầu
```

Out[371]:
```
array([[4, 5, 6],
       [9, 8, 7]])
```

In [372…
```
arr[-2:, -2:] # Chọn các phần tử 2 hàng cuối, 2 cột cuối
```

Out[372]:
```
array([[7, 8],
       [6, 5]])
```

Note: Đây là tham chiếu, không phải copy giá trị và tạo một mảng mới

Thay đổi không gian chứa với `reshape()`. Giả sử với không gian 2D, ta có `reshape(x, y)` trong đó, x là số hàng, y là số cột sau khi thay đổi, tất nhiên là tích của chúng sẽ phải bằng số lượng phần tử ban đầu (Nếu mang giá trị `-1`, nó sẽ tự cân bằng tương ứng với các chiều còn lại):

In [373…
```
arr.reshape(-1, 3)
```

Out[373]:
```
array([[1, 2, 3],
       [4, 5, 4],
       [5, 6, 7],
```

```
        [8, 9, 8],
        [7, 6, 5]])
```

Xử lý toán học:

In [374…
```
x1 = np.array([[1,2,3],[4,5,6]])
y1 = np.array([[7,8,9],[2,3,4]])
print(x1 + y1, x1 - y1, x1 * y1, sep='\n')
```

```
[[ 8 10 12]
 [ 6  8 10]]
[[-6 -6 -6]
 [ 2  2  2]]
[[ 7 16 27]
 [ 8 15 24]]
```

Kết quả giống như khi ta tính toán 2 vector chung gốc.

Áp dụng thực tế để tính toán chỉ số BMI:

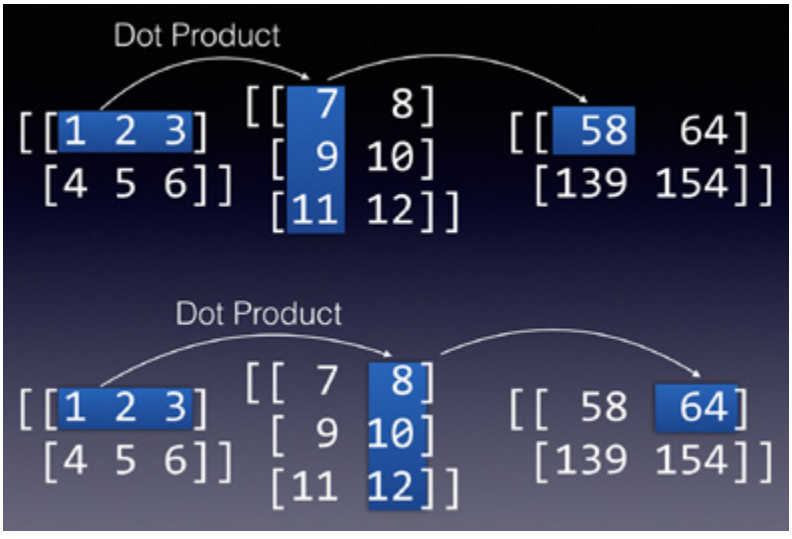
In [375…
```
names = np.array(['Ann','Joe','Mark'])
heights = np.array([1.5, 1.78, 1.6])
weights = np.array([65, 46, 59])
bmi = weights/heights **2
print(bmi)
print("Overweight: " , names[bmi>25])
print("Underweight: " , names[bmi<18.5])
print("Healthy: " , names[(bmi>=18.5) & (bmi<=25)])
```

```
[28.88888889 14.51836889 23.046875  ]
Overweight:  ['Ann']
Underweight:  ['Joe']
Healthy:  ['Mark']
```

Tính tích vô hướng bằng `numpy` :

In [376…
```
x2 = np.array([2,3])
y2 = np.array([4,2])
print(np.dot(x2, y2)) # 2x4 + 3x2 = 14
x2 = np.array([[1,2,3],[4,5,6]])
y2 = np.array([[7,8],[9,10], [11,12]])
print(np.dot(x2, y2))
```

```
14
[[ 58  64]
 [139 154]]
```



Ma trận toán học trong `numpy` sử dụng `matrix`:

In [377…
```
x1 = np.array([[1,2],[4,5]])
y1 = np.array([[7,8],[2,3]])
print(x1 * y1) # element-by-element multiplication
x2 = np.matrix([[1,2],[4,5]])
y2 = np.matrix([[7,8],[2,3]])
print(x2 * y2) # dot product; same as np.dot()
```

```
[[ 7 16]
 [ 8 15]]
[[11 14]
 [38 47]]
```

Tổng tích lũy:

In [378…
```
print(np.array([(1,2,3),(4,5,6), (7,8,9)]).cumsum())
print(np.array([(1,2,3),(4,5,6), (7,8,9)]).cumsum(axis=0))  # sum over rows for each of
print(np.array([(1,2,3),(4,5,6), (7,8,9)]).cumsum(axis=1))  # sum over columns for each
```

```
[ 1  3  6 10 15 21 28 36 45]
[[ 1  2  3]
 [ 5  7  9]
 [12 15 18]]
[[ 1  3  6]
 [ 4  9 15]
 [ 7 15 24]]
```

Sắp xếp:

In [379…
```
ages = np.array([34,12,37,5,13])
np.sort(ages)
```

Out[379]:
```
array([ 5, 12, 13, 34, 37])
```

Lấy thứ tự giá trị của từng phần tử tại từng vị trí:

In [380…
```
ages = np.array([34,12,37,5,13])
ages.argsort()
```

Out[380]:
```
array([3, 1, 4, 0, 2])
```

Ứng dụng thực tế trong trường hợp ta muốn sắp xếp các trường dữ liệu phụ thuộc vào một trường cá biệt nào đó:

In [381…
```
persons = np.array(['Johnny','Mary','Peter','Will','Joe'])
ages = np.array([34,12,37,5,13])
heights = np.array([1.76,1.2,1.68,0.5,1.25])
sort_indices = np.argsort(ages) # Sắp xếp theo tuổi
print(persons[sort_indices])
print(ages[sort_indices])
print(heights[sort_indices])
```

```
['Will' 'Mary' 'Joe' 'Johnny' 'Peter']
[ 5 12 13 34 37]
[0.5  1.2  1.25 1.76 1.68]
```

1.1.2. Pandas

1.1.2.1. Giới thiệu

Mặc dù `numpy` đã hỗ trợ rất nhiều trong việc lưu trữ và xử lý dữ liệu trong Python, nhưng để xử lý dữ liệu tốt hơn, ta cần sử dụng đến `pandas`. Thư viện này song song với `numpy`, đều là các thư viện giúp xử lý dữ liệu, đồng thời `pandas` có thể thực hiện cả đọc/ghi dữ liệu.

1.1.2.2. Sử dụng

Import package:

In [382…
```
import pandas as pd
```

Đánh số các bản ghi trong `pandas` (bắt đầu từ 0):

In [383…
```
series = pd.Series([2, "4", '5', 0])
print(series)
```

```
0    2
1    4
2    5
3    0
dtype: object
```

Ta có thể tùy biến cách đánh `series`:

In [384…
```
series = pd.Series([1,2,3,4,5], index=['a','b','c','d','c'])
print(series)
```

```
a    1
b    2
c    3
d    4
c    5
dtype: int64
```

Truy cập các bản ghi theo vị trí:

In [385…
```
print(series[2])
print(series.iloc[2])
```

```
3
3
```

Truy cập các bản ghi theo serial key:

In [386…
```
print(series['d'])
print(series.loc['d'])
print(series['c']) # Có nhiều hơn 2 bản có cùng serial key là 'c'
```

```
4
4
c    3
c    5
dtype: int64
```

Lấy dãy bản ghi con:

In [387…
```
print(series[2:])
print(series.iloc[2:])
```

```
c    3
d    4
c    5
dtype: int64
```

```
c    3
d    4
c    5
dtype: int64
```

Sử dụng `DataFrame` để lưu trữ và xử lý dữ liệu dạng bảng:

In [388…
```
df = pd.DataFrame(np.random.randn(10,4), columns=list('ABCD'))
df
```

Out[388]:

	A	B	C	D
0	0.806266	0.392608	0.556294	0.034047
1	-0.158074	0.247951	0.537748	1.012473
2	0.065018	0.285939	-0.494612	1.201865
3	1.287152	0.909164	-0.475680	1.117565
4	-2.174226	-1.258408	1.643840	0.171294
5	0.408650	0.570175	0.367287	-1.760007
6	-0.905922	1.046675	0.590665	-0.093675
7	0.261140	-0.497732	2.121964	1.688859
8	0.744305	0.830586	0.807131	0.663170
9	-0.598963	-0.300855	1.074769	-0.056793

Thông thường, ta sẽ đọc dữ liệu từ trong một tệp `csv`, khi đó, ta sử dụng `pd.read_csv()` để lấy được `DataFrame` chứa dữ liệu đó.

Ta có thể lấy thống kê cơ bản của dữ liệu trong `DataFrame` bằng phương thức `describe`:

In [389…
```
df.describe()
```

Out[389]:

	A	B	C	D
count	10.000000	10.000000	10.000000	10.000000
mean	-0.026465	0.222610	0.672941	0.397880
std	1.001384	0.719626	0.818968	0.975683
min	-2.174226	-1.258408	-0.494612	-1.760007
25%	-0.488741	-0.163653	0.409902	-0.034083
50%	0.163079	0.339274	0.573480	0.417232
75%	0.660391	0.765483	1.007860	1.091292
max	1.287152	1.046675	2.121964	1.688859

Truy cập các bản ghi của `DataFrame`:

- Các dòng đầu/cuối:

In [390…
```
print(df.head(3)) # 3 dòng đầu
print(df.tail(3)) # 3 dòng cuối
```

```
          A         B         C         D
0  0.806266  0.392608  0.556294  0.034047
1 -0.158074  0.247951  0.537748  1.012473
2  0.065018  0.285939 -0.494612  1.201865
          A         B         C         D
7  0.261140 -0.497732  2.121964  1.688859
8  0.744305  0.830586  0.807131  0.663170
9 -0.598963 -0.300855  1.074769 -0.056793
```

- Theo cột:

In [391…
```
print(df['A'])
print(df.A)
print(df[['A', 'B']])
```

```
0    0.806266
1   -0.158074
2    0.065018
3    1.287152
4   -2.174226
5    0.408650
6   -0.905922
7    0.261140
8    0.744305
9   -0.598963
Name: A, dtype: float64
0    0.806266
1   -0.158074
2    0.065018
3    1.287152
4   -2.174226
5    0.408650
6   -0.905922
7    0.261140
8    0.744305
9   -0.598963
Name: A, dtype: float64
          A         B
0  0.806266  0.392608
1 -0.158074  0.247951
2  0.065018  0.285939
3  1.287152  0.909164
4 -2.174226 -1.258408
5  0.408650  0.570175
6 -0.905922  1.046675
7  0.261140 -0.497732
8  0.744305  0.830586
9 -0.598963 -0.300855
```

- Theo dải:

In [392…
```
print(df[2:4])
print(df.iloc[2:4])
print(df.iloc[[2,4]]) # Truy cập chính xác hàng nào đó
```

```
          A         B         C         D
2  0.065018  0.285939 -0.494612  1.201865
3  1.287152  0.909164 -0.475680  1.117565
          A         B         C         D
2  0.065018  0.285939 -0.494612  1.201865
3  1.287152  0.909164 -0.475680  1.117565
          A         B         C         D
```

```
2  0.065018  0.285939 -0.494612  1.201865
4 -2.174226 -1.258408  1.643840  0.171294
```

In [393…
```
print(df.iloc[2:4, 1:4])
print(df.iloc[[2,4], [1,3]])
```

```
          B         C         D
2  0.285939 -0.494612  1.201865
3  0.909164 -0.475680  1.117565
          B         D
2  0.285939  1.201865
4 -1.258408  0.171294
```

- Theo nhãn:

In [394…
```
df.index = pd.date_range('20190525', periods=10)
print(df['20190601':'20190603'])
print(df.loc['20190601':'20190603'])
print(df.loc['20190601':'20190603', 'A':'C'])
```

```
                   A         B         C         D
2019-06-01  0.261140 -0.497732  2.121964  1.688859
2019-06-02  0.744305  0.830586  0.807131  0.663170
2019-06-03 -0.598963 -0.300855  1.074769 -0.056793
                   A         B         C         D
2019-06-01  0.261140 -0.497732  2.121964  1.688859
2019-06-02  0.744305  0.830586  0.807131  0.663170
2019-06-03 -0.598963 -0.300855  1.074769 -0.056793
                   A         B         C
2019-06-01  0.261140 -0.497732  2.121964
2019-06-02  0.744305  0.830586  0.807131
2019-06-03 -0.598963 -0.300855  1.074769
```

- Theo quy tắc tùy chỉnh:

In [395…
```
print(df[(df.A > 0) & (df.B>0)])
```

```
                   A         B         C         D
2019-05-25  0.806266  0.392608  0.556294  0.034047
2019-05-27  0.065018  0.285939 -0.494612  1.201865
2019-05-28  1.287152  0.909164 -0.475680  1.117565
2019-05-30  0.408650  0.570175  0.367287 -1.760007
2019-06-02  0.744305  0.830586  0.807131  0.663170
```

Lấy ma trận trực giao của `DataFrame`:

In [396…
```
print(df.transpose())
print(df.T)
```

```
   2019-05-25  2019-05-26  2019-05-27  2019-05-28  2019-05-29  2019-05-30  \
A    0.806266   -0.158074    0.065018    1.287152   -2.174226    0.408650
B    0.392608    0.247951    0.285939    0.909164   -1.258408    0.570175
C    0.556294    0.537748   -0.494612   -0.475680    1.643840    0.367287
D    0.034047    1.012473    1.201865    1.117565    0.171294   -1.760007

   2019-05-31  2019-06-01  2019-06-02  2019-06-03
A   -0.905922    0.261140    0.744305   -0.598963
B    1.046675   -0.497732    0.830586   -0.300855
C    0.590665    2.121964    0.807131    1.074769
D   -0.093675    1.688859    0.663170   -0.056793
   2019-05-25  2019-05-26  2019-05-27  2019-05-28  2019-05-29  2019-05-30  \
A    0.806266   -0.158074    0.065018    1.287152   -2.174226    0.408650
B    0.392608    0.247951    0.285939    0.909164   -1.258408    0.570175
C    0.556294    0.537748   -0.494612   -0.475680    1.643840    0.367287
```

```
D    0.034047    1.012473    1.201865    1.117565    0.171294   -1.760007

   2019-05-31  2019-06-01  2019-06-02  2019-06-03
A   -0.905922    0.261140    0.744305   -0.598963
B    1.046675   -0.497732    0.830586   -0.300855
C    0.590665    2.121964    0.807131    1.074769
D   -0.093675    1.688859    0.663170   -0.056793
```

Sắp xếp dữ liệu trong `DataFrame`:

- Theo nhãn, sử dụng `sort_index()`:

In [397…
```
print(df.sort_index(axis=0, ascending=False)) # Sắp xếp theo hàng, giảm dần
print(df.sort_index(axis=1, ascending=False)) # Sắp xếp theo cột, giảm dần
```

```
                   A         B         C         D
2019-06-03 -0.598963 -0.300855  1.074769 -0.056793
2019-06-02  0.744305  0.830586  0.807131  0.663170
2019-06-01  0.261140 -0.497732  2.121964  1.688859
2019-05-31 -0.905922  1.046675  0.590665 -0.093675
2019-05-30  0.408650  0.570175  0.367287 -1.760007
2019-05-29 -2.174226 -1.258408  1.643840  0.171294
2019-05-28  1.287152  0.909164 -0.475680  1.117565
2019-05-27  0.065018  0.285939 -0.494612  1.201865
2019-05-26 -0.158074  0.247951  0.537748  1.012473
2019-05-25  0.806266  0.392608  0.556294  0.034047
                   D         C         B         A
2019-05-25  0.034047  0.556294  0.392608  0.806266
2019-05-26  1.012473  0.537748  0.247951 -0.158074
2019-05-27  1.201865 -0.494612  0.285939  0.065018
2019-05-28  1.117565 -0.475680  0.909164  1.287152
2019-05-29  0.171294  1.643840 -1.258408 -2.174226
2019-05-30 -1.760007  0.367287  0.570175  0.408650
2019-05-31 -0.093675  0.590665  1.046675 -0.905922
2019-06-01  1.688859  2.121964 -0.497732  0.261140
2019-06-02  0.663170  0.807131  0.830586  0.744305
2019-06-03 -0.056793  1.074769 -0.300855 -0.598963
```

- Theo giá trị, sử dụng `sort_values()`:

In [398…
```
print(df.sort_values('A', axis=0))
print(df.sort_values('20190601', axis=1))
```

```
                   A         B         C         D
2019-05-29 -2.174226 -1.258408  1.643840  0.171294
2019-05-31 -0.905922  1.046675  0.590665 -0.093675
2019-06-03 -0.598963 -0.300855  1.074769 -0.056793
2019-05-26 -0.158074  0.247951  0.537748  1.012473
2019-05-27  0.065018  0.285939 -0.494612  1.201865
2019-06-01  0.261140 -0.497732  2.121964  1.688859
2019-05-30  0.408650  0.570175  0.367287 -1.760007
2019-06-02  0.744305  0.830586  0.807131  0.663170
2019-05-25  0.806266  0.392608  0.556294  0.034047
2019-05-28  1.287152  0.909164 -0.475680  1.117565
                   B         A         D         C
2019-05-25  0.392608  0.806266  0.034047  0.556294
2019-05-26  0.247951 -0.158074  1.012473  0.537748
2019-05-27  0.285939  0.065018  1.201865 -0.494612
2019-05-28  0.909164  1.287152  1.117565 -0.475680
2019-05-29 -1.258408 -2.174226  0.171294  1.643840
2019-05-30  0.570175  0.408650 -1.760007  0.367287
2019-05-31  1.046675 -0.905922 -0.093675  0.590665
2019-06-01 -0.497732  0.261140  1.688859  2.121964
```

```
2019-06-02  0.830586  0.744305  0.663170  0.807131
2019-06-03 -0.300855 -0.598963 -0.056793  1.074769
```

Ta có thể ánh xạ các phần tử trong `DataFrame` thành các ảnh bằng phương thức `apply`:

In [399…
```
import math
sq_root = lambda x: math.sqrt(x) if x > 0 else x
sq = lambda x: x**2
print(df.B.apply(sq_root))
print(df.B.apply(sq))
```

```
2019-05-25    0.626584
2019-05-26    0.497946
2019-05-27    0.534733
2019-05-28    0.953501
2019-05-29   -1.258408
2019-05-30    0.755099
2019-05-31    1.023072
2019-06-01   -0.497732
2019-06-02    0.911365
2019-06-03   -0.300855
Freq: D, Name: B, dtype: float64
2019-05-25    0.154141
2019-05-26    0.061480
2019-05-27    0.081761
2019-05-28    0.826579
2019-05-29    1.583590
2019-05-30    0.325100
2019-05-31    1.095530
2019-06-01    0.247737
2019-06-02    0.689872
2019-06-03    0.090513
Freq: D, Name: B, dtype: float64
```

Thêm và xóa hàng/cột trong `DataFrame`:

In [400…
```
data = {
    'name': ['Janet', 'Nad', 'Timothy', 'June', 'Amy'],
    'year': [2012, 2012, 2013, 2014, 2014],
    'reports': [6, 13, 14, 1, 7]
    }
df = pd.DataFrame(data, index = ['Singapore', 'China', 'Japan', 'Sweden', 'Norway'])
print(df)
```

```
             name  year  reports
Singapore    Janet  2012        6
China          Nad  2012       13
Japan      Timothy  2013       14
Sweden        June  2014        1
Norway         Amy  2014        7
```

- Thêm:

In [401…
```
import numpy as np
schools = np.array(["Cambridge","Oxford","Oxford","Cambridge","Oxford"])
df["school"] = schools
print(df)
```

```
             name  year  reports     school
Singapore    Janet  2012        6  Cambridge
China          Nad  2012       13     Oxford
Japan      Timothy  2013       14     Oxford
Sweden        June  2014        1  Cambridge
Norway         Amy  2014        7     Oxford
```

- Xóa:

In [402…
```
print(df.drop(['China', 'Japan']))
print(df[df.name != 'Nad'])
print(df.drop(df.index[1]))
print(df.drop(df.index[[1,2]]))
print(df.drop('reports', axis=1))
print(df.drop(df.columns[1], axis=1))
print(df.drop(df.columns[[1,3]], axis=1))
```

```
            name  year  reports     school
Singapore  Janet  2012        6  Cambridge
Sweden      June  2014        1  Cambridge
Norway       Amy  2014        7     Oxford
              name  year  reports     school
Singapore    Janet  2012        6  Cambridge
Japan      Timothy  2013       14     Oxford
Sweden        June  2014        1  Cambridge
Norway         Amy  2014        7     Oxford
              name  year  reports     school
Singapore    Janet  2012        6  Cambridge
Japan      Timothy  2013       14     Oxford
Sweden        June  2014        1  Cambridge
Norway         Amy  2014        7     Oxford
            name  year  reports     school
Singapore  Janet  2012        6  Cambridge
Sweden      June  2014        1  Cambridge
Norway       Amy  2014        7     Oxford
              name  year     school
Singapore    Janet  2012  Cambridge
China          Nad  2012     Oxford
Japan      Timothy  2013     Oxford
Sweden        June  2014  Cambridge
Norway         Amy  2014     Oxford
              name  reports     school
Singapore    Janet        6  Cambridge
China          Nad       13     Oxford
Japan      Timothy       14     Oxford
Sweden        June        1  Cambridge
Norway         Amy        7     Oxford
              name  reports
Singapore    Janet        6
China          Nad       13
Japan      Timothy       14
Sweden        June        1
Norway         Amy        7
```

1.1.3. Mathplotlib

1.1.3.1. Giới thiệu

Thư viện này là thư viện chính giúp trực quan hóa dữ liệu bằng cách biểu diễn chúng trên đồ thị, biểu đồ, ...

1.1.3.2. Sử dụng

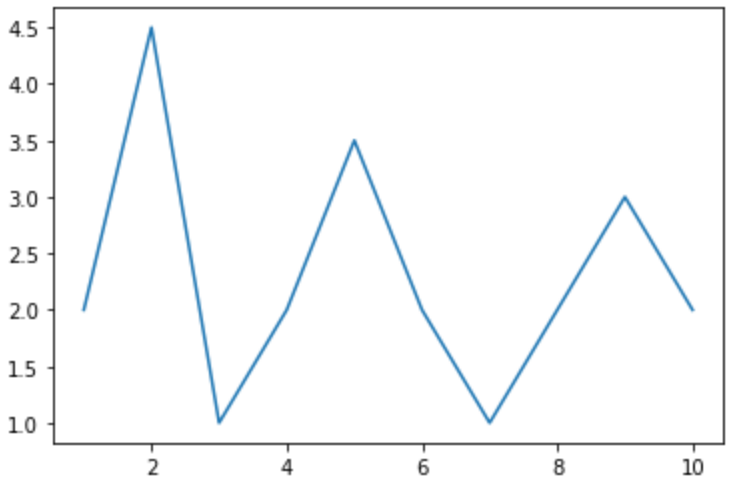
Import package:

In [216…
```
%matplotlib inline
import matplotlib.pyplot as plt
```

Vẽ biểu đồ:

In [217…
```
plt.plot(
  [1,2,3,4,5,6,7,8,9,10],
  [2,4.5,1,2,3.5,2,1,2,3,2]
)
```

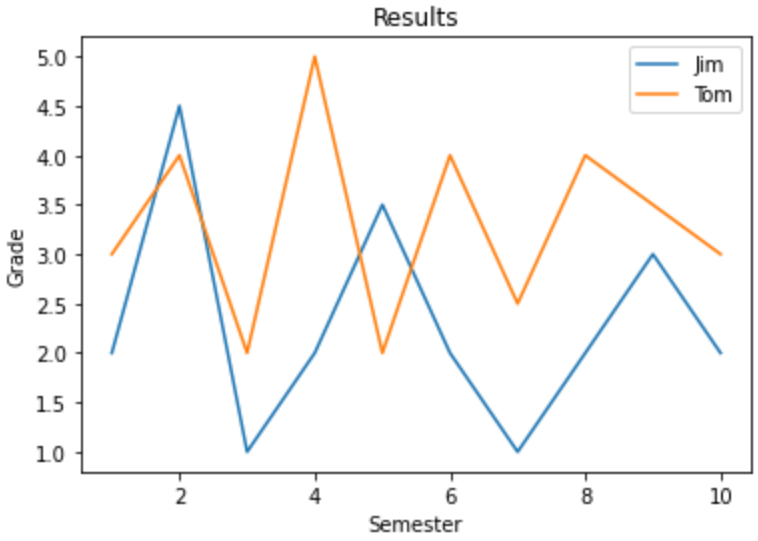
Out[217]:
```
[<matplotlib.lines.Line2D at 0x7f178aca6590>]
```



Thêm tiêu đề và nhãn:

In [218…
```
plt.plot(
[1,2,3,4,5,6,7,8,9,10],
[2,4.5,1,2,3.5,2,1,2,3,2],
label="Jim"
)
plt.plot(
[1,2,3,4,5,6,7,8,9,10],
[3,4,2,5,2,4,2.5,4,3.5,3],
label="Tom"
)
plt.title("Results")
plt.xlabel("Semester")
plt.ylabel("Grade")
plt.legend()
```

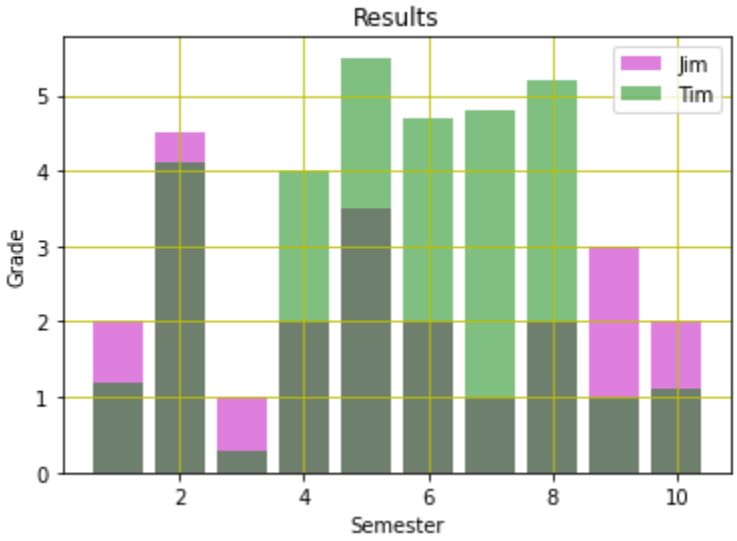
Out[218]:
```
<matplotlib.legend.Legend at 0x7f1789dac610>
```



Vẽ biểu đồ dạng cột:

In [219…
```
plt.bar(
  [1,2,3,4,5,6,7,8,9,10],
```

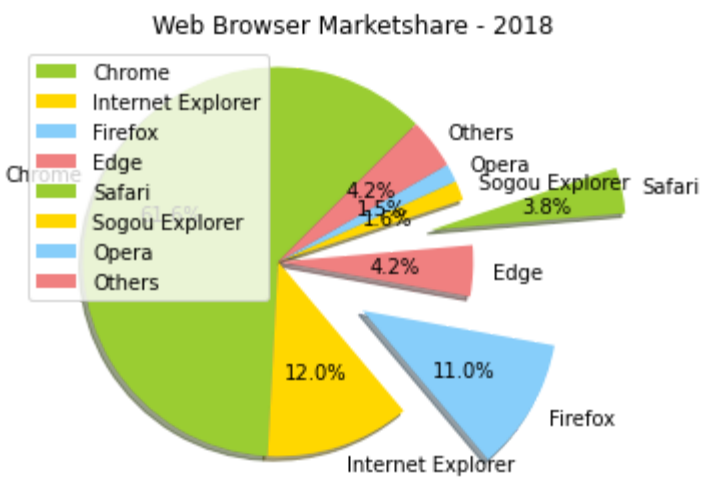
```
  [2,4.5,1,2,3.5,2,1,2,3,2],
  label = "Jim",
  color = "m",
  align = "center",
  alpha = 0.5
)

plt.bar(
  [1,2,3,4,5,6,7,8,9,10],
  [1.2,4.1,0.3,4,5.5,4.7,4.8,5.2,1,1.1],
  label = "Tim",
  color = "g",
  align = "center",
  alpha = 0.5
)

plt.title("Results")
plt.xlabel("Semester")
plt.ylabel("Grade")
plt.legend()
plt.grid(True, color="y")
```



Vẽ biểu đồ hình quạt:

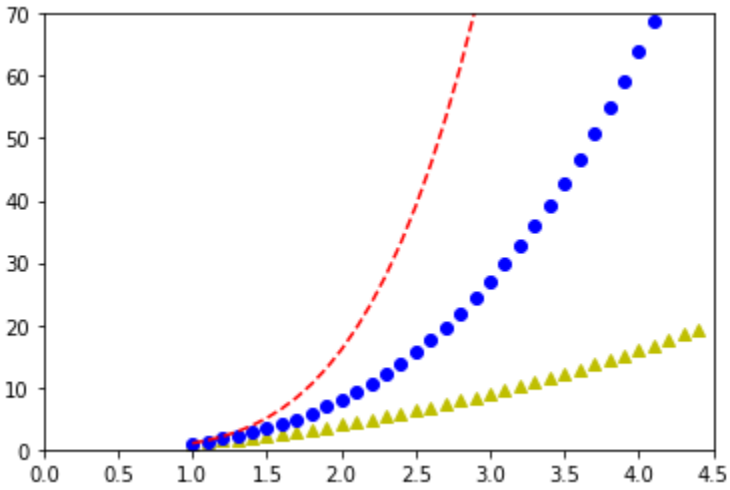
In [220…

```
labels = ["Chrome", "Internet Explorer",
"Firefox", "Edge","Safari",
"Sogou Explorer","Opera","Others"]
marketshare = [61.64, 11.98, 11.02, 4.23, 3.79, 1.63, 1.52, 4.19]
explode = (0,0,0.5,0,0.8,0,0,0)
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']

pie = plt.pie(
    marketshare,
    explode = explode, # fraction of the radius with which to offset each wedge
    labels = labels,
    colors = colors,
    autopct="%.1f%%", # string or function used to label the wedges with their numeric v
    shadow=True,
    startangle=45 # rotates the start of the pie chart by angle degrees counterclockwise
)
plt.axis("equal") # turns off the axis lines and labels
plt.title("Web Browser Marketshare - 2018")
plt.legend(pie[0], labels, loc="best")
plt.show()
```



Vẽ đồ thị phân tán:

In [221…
```
a = np.arange(1,4.5,0.1) # 1.0, 1.1, 1.2, 1.3...4.4
plt.plot(a, a**2, 'y^', # yellow triangle_up marker
         a, a**3, 'bo', # blue circle
        a, a**4, 'r--',) # red dashed line
plt.axis([0, 4.5, 0, 70]) # xmin, xmax, ymin, ymax
plt.show()
```



1.1.4. Scikit-learn

1.1.4.1. Giới thiệu

Là một thư viện chứa rất nhiều mẫu dữ liệu cũng như các thuật toán Machine learning.

1.1.4.2. Sử dụng

1.1.4.2.1. Dataset

Ví dụ về load dataset:

In [403…
```
from sklearn import datasets
iris = datasets.load_iris()
iris
```

Out[403]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
```

```
[5.4, 3.9, 1.7, 0.4],
[4.6, 3.4, 1.4, 0.3],
[5. , 3.4, 1.5, 0.2],
[4.4, 2.9, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.1],
[5.4, 3.7, 1.5, 0.2],
[4.8, 3.4, 1.6, 0.2],
[4.8, 3. , 1.4, 0.1],
[4.3, 3. , 1.1, 0.1],
[5.8, 4. , 1.2, 0.2],
[5.7, 4.4, 1.5, 0.4],
[5.4, 3.9, 1.3, 0.4],
[5.1, 3.5, 1.4, 0.3],
[5.7, 3.8, 1.7, 0.3],
[5.1, 3.8, 1.5, 0.3],
[5.4, 3.4, 1.7, 0.2],
[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
```

```
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
```

```
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set
Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :
Number of Attributes: 4 numeric, predictive attributes and the class\n    :Attribute Inf
ormation:\n        - sepal length in cm\n        - sepal width in cm\n        - petal le
ngth in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa
\n                - Iris-Versicolour\n                - Iris-Virginica\n
\n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ==============
=====\n                    Min  Max   Mean    SD   Class Correlation\n    ==============
==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83
    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.
0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76
0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\n\n
:Missing Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n
    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n
    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The da
taset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in th
e UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps
the best known database to be found in the\npattern recognition literature.  Fisher\'s p
aper is a classic in the field and\nis referenced frequently to this day.  (See Duda & H
art, for example.)  The\ndata set contains 3 classes of 50 instances each, where each cl
ass refers to a\ntype of iris plant.  One class is linearly separable from the other 2;
the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   -
Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugen
ics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistic
s" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification
and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 2
18.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Str
ucture and Classification Rule for Recognition in Partially Exposed\n     Environments".
  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No.
1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactio
ns\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings,
54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classe
s in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

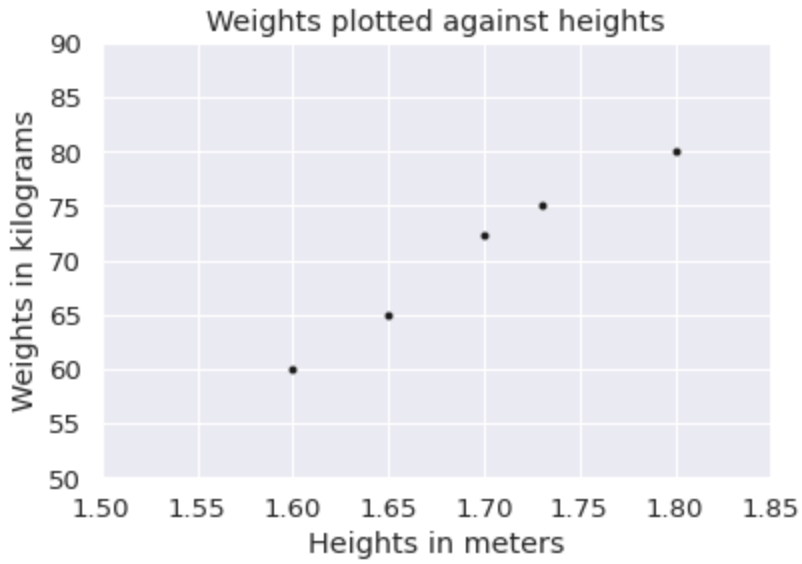

1.1.4.2.2. UCI Machine Learning Repository

Linear regression - Hồi quy tuyến tính

Là một phương pháp thống kê để hồi quy dữ liệu với biến phụ thuộc có giá trị liên tục trong khi các biến độc lập có thể có một trong hai giá trị liên tục hoặc là giá trị phân loại.

Với dữ liệu:

In [404…
```
# represents the heights of a group of people in meters
heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]
# represents the weights of a group of people in kgs
weights = [[60], [65], [72.3], [75], [80]]
plt.title('Weights plotted against heights')
plt.xlabel('Heights in meters')
plt.ylabel('Weights in kilograms')
plt.plot(heights, weights, 'k.')
# axis range for x and y
plt.axis([1.5, 1.85, 50, 90])
plt.grid(True)
```



Sử dụng `LinearRegression` trong `sklearn.linear_model` để áp dụng thuật toán:

In [405…
```
from sklearn.linear_model import LinearRegression
# Create and fit the model
model = LinearRegression()
model.fit(X=heights, y=weights)
```

Out[405]:
```
LinearRegression()
```

Thử dự đoán kết quả thông qua thông tin có sẵn:

In [406…
```
weight = model.predict([[1.75]])[0][0]
round(weight,2)
```

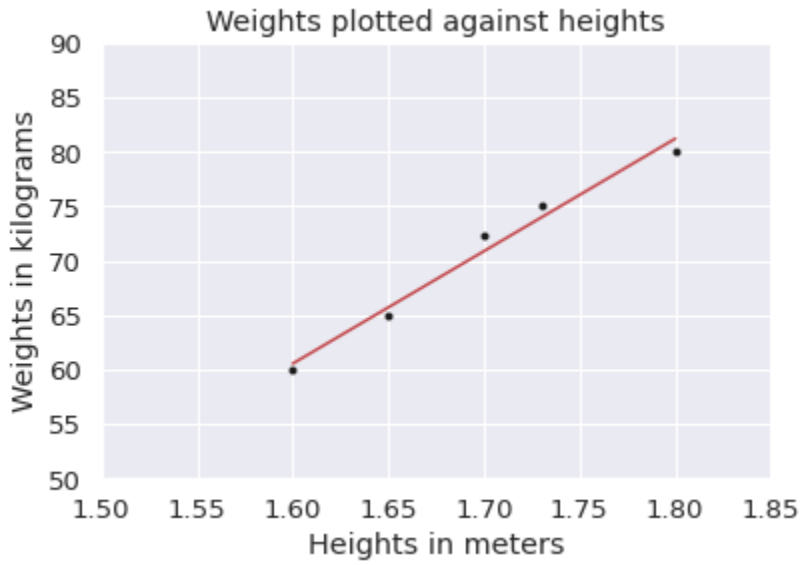
Out[406]:
```
76.04
```

Trực quan hóa đường thẳng hồi quy:

In [407…
```
heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]
weights = [[60], [65], [72.3], [75], [80]]
plt.title('Weights plotted against heights')
plt.xlabel('Heights in meters')
plt.ylabel('Weights in kilograms')
plt.plot(heights, weights, 'k.')
plt.axis([1.5, 1.85, 50, 90])
```

```
plt.grid(True)
# plot the regression line
plt.plot(heights, model.predict(heights), color='r')
```

Out[407]:
```
[<matplotlib.lines.Line2D at 0x7f1784695dd0>]
```



Đánh giá `model`:

In [408…
```
# test data
heights_test = [[1.58], [1.62], [1.69], [1.76], [1.82]]
weights_test = [[58], [63], [72], [73], [85]]

# Total Sum of Squares (TSS)
weights_test_mean = np.mean(np.ravel(weights_test))
TSS = np.sum((np.ravel(weights_test) -
weights_test_mean) ** 2)
print("TSS: %.2f" % TSS)
# Residual Sum of Squares (RSS)
RSS = np.sum((np.ravel(weights_test) -
np.ravel(model.predict(heights_test)))
** 2)
print("RSS: %.2f" % RSS)
# R_squared
R_squared = 1 - (RSS / TSS)
print("R-squared: %.2f" % R_squared)
```

```
TSS: 430.80
RSS: 24.62
R-squared: 0.94
```

=> Tỉ lệ chính xác là 94%.

1.2. Data Cleaning

1.2.1. Giới thiệu

Là quá trình thay đổi hoặc loại bỏ dữ liệu không chính xác, trùng lặp, bị hỏng hoặc không đầy đủ. Nếu dữ liệu không chính xác, các thuật toán và kết quả cho ra không đáng tin cậy.

Các thao tác thường thấy:

- Loại bỏ dữ liệu.
- 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.

Yêu cầu về chất lượng của dữ liệu:

- **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.

1.2.2. Áp dụng

Với tập dữ liệu: https://www.kaggle.com/c/sberbank-russian-housing-market/data?select=train.csv.zip

Ta có các bước để làm sạch dữ liệu:

1.2.2.0. Init - Khởi tạo và chuẩn bị

In [436…
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('housing_data.csv')
df.shape
```

Out[436]:
```
(30471, 292)
```

Tách các cột số và các cột còn lại ra làm 2 phần:

In [437…
```
# select numerical columns
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
# select non-numeric columns
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
```

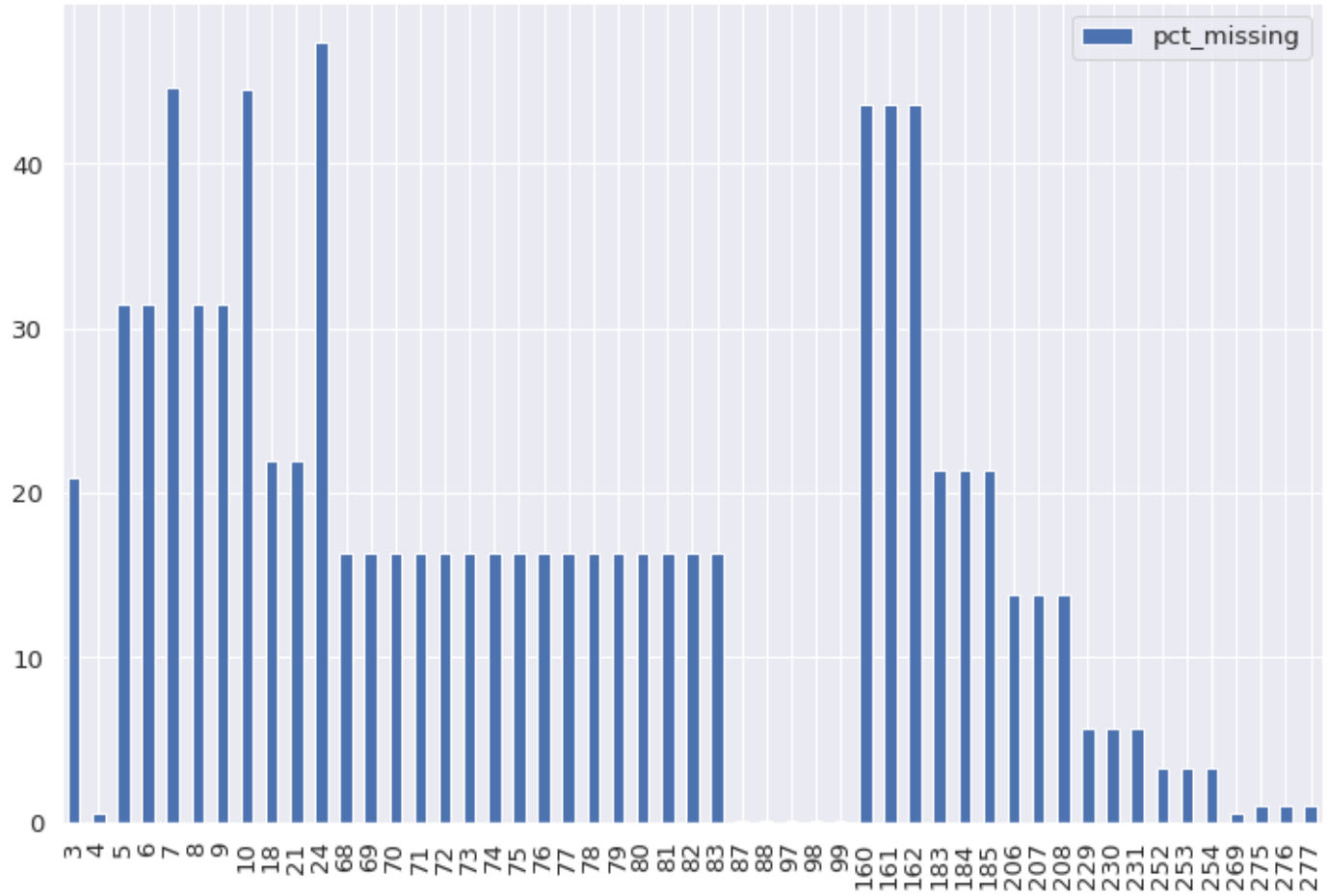
1.2.2.1. Missing values - Xử lý thiếu giá trị

Bắt đầu bằng cách tính toán tỷ lệ phần trăm giá trị bị thiếu trong mỗi cột, và sau đó lưu trữ thông tin này trong một `DataFrame`.

In [438…
```
# % of values missing in each column
values_list = list()
cols_list = list()
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())*100
    cols_list.append(col)
    values_list.append(pct_missing)
pct_missing_df = pd.DataFrame()
pct_missing_df['col'] = cols_list
pct_missing_df['pct_missing'] = values_list
```

`pct_missing_df` hiện giờ sẽ chứa phần trăm giá trị bị thiếu trong mỗi cột cùng với tên cột:

In [439…
```
pct_missing_df.loc [pct_missing_df.pct_missing> 0] .plot (kind = 'bar', figsize = (12,8)
plt.show ()
```



Các bước xử lý:

Drop observations

Loại bỏ theo dõi các cột không chứa hoặc chứa ít giá trị null (không quá 0.5%):

In [440…
```
less_missing_values_cols_list = list(pct_missing_df.loc[(pct_missing_df.pct_missing < 0.
df.dropna(subset=less_missing_values_cols_list, inplace=True)
```

Remove features

Xóa bỏ những feature chứa quá nhiều bản ghi có giá trị null (nhiều hơn 40%):

In [441…
```
# dropping columns with more than 40% null values
_40_pct_missing_cols_list = list(pct_missing_df.loc[pct_missing_df.pct_missing > 40, 'co
df.drop(columns=_40_pct_missing_cols_list, inplace=True)
```

Impute missing values

Những giá trị bị thiếu còn lại sẽ được thay thế bằng giá trị trung bình của cột:

In [442…
```
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
for col in numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:  # impute values only for columns that have missing values
```

```
        med = df[col].median() # impute with the median
        df[col] = df[col].fillna(med)
```

Với trường hợp dành cho các cột không có kiểu dữ liệu số, thay thế bằng giá trị xuất hiện nhiều nhất trong cột:

In [443…
```
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
for col in non_numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:  # impute values only for columns that have missing values
        mod = df[col].describe()['top'] # impute with the most frequently occuring value
        df[col] = df[col].fillna(mod)
```

Kiểm tra:

In [444…
```
df.isnull().sum().sum()
```

Out[444]:
```
0
```

1.2.2.2. Outliers - Xử lý dữ liệu ngoại lai

Dữ liệu ngoại lai là những dữ liệu bất thường, có giá trị khác xa với giá trị trung bình của cùng feature.

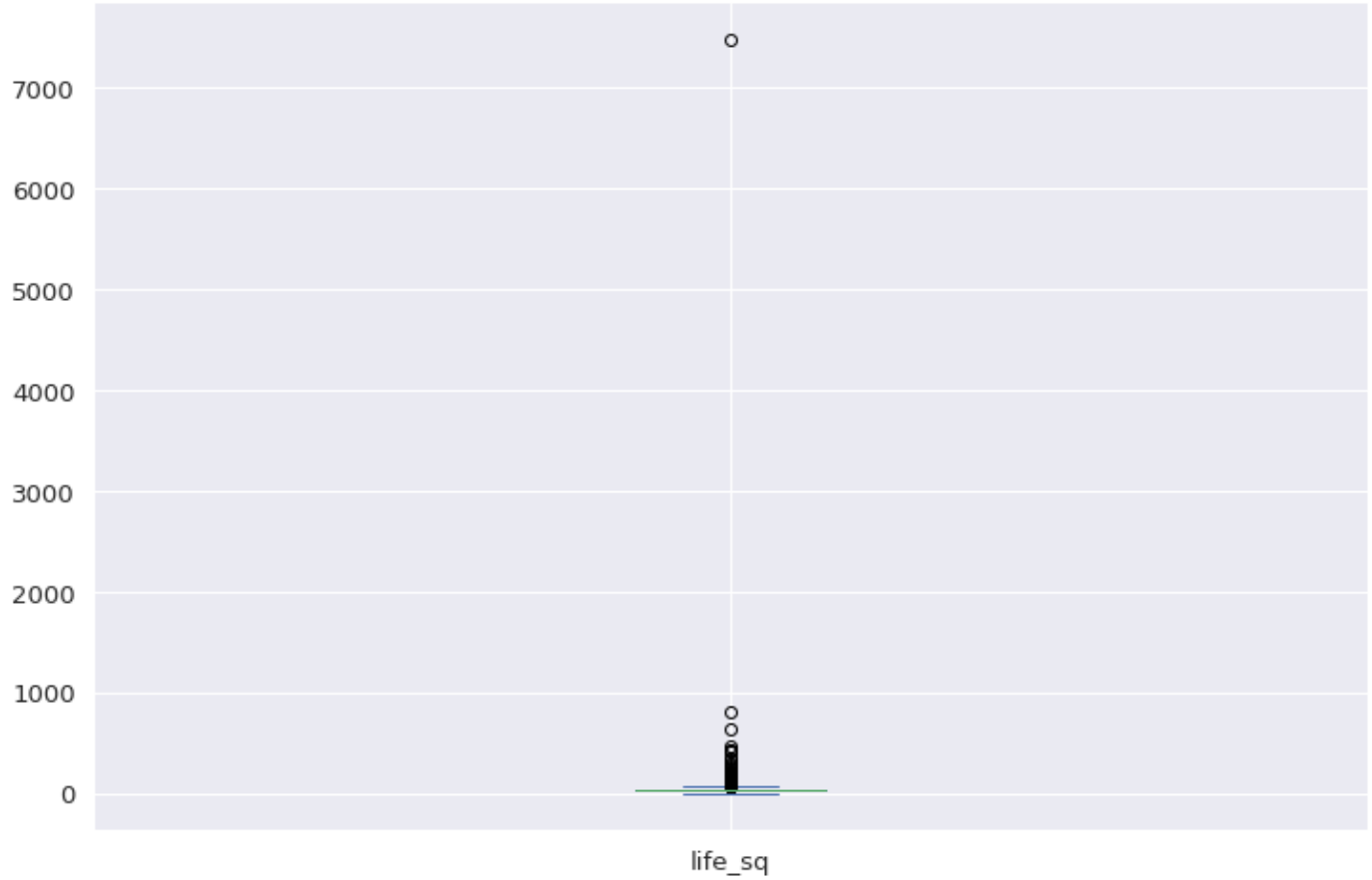
Ví dụ với cột `life_sq`, ta sẽ xem thống kê:

In [445…
```
df.life_sq.describe ()
```

Out[445]:
```
count    30446.000000
mean        33.482658
std         46.538609
min          0.000000
25%         22.000000
50%         30.000000
75%         38.000000
max       7478.000000
Name: life_sq, dtype: float64
```

Ta thấy, giá trị tối đa rất lớn so với giá trị trung bình và giá trị trung vị của feature, ta có thể thử quan sát qua biểu đồ:

In [446…
```
df.life_sq.plot(kind = 'box', figsize = (12, 8))
plt.show()
```



Để xử lý, ta có thể thay thế nó bằng giá trị phù hợp hoặc đơn giản là loại bỏ bản ghi này:

In [447…
```
# removing the outlier value in life_sq column
df = df.loc[df.life_sq < 7478]
```

1.2.2.3. Duplicate records - Xử lý các bản ghi trùng lặp

Sẽ có thể có những bản ghi dữ liệu bị trùng lặp.

Trong tập dữ liệu này, ID là cột duy nhất giúp nhận dạng và phân biệt các bản ghi, ta sẽ thực hiện loại bỏ các bản ghi trùng lặp dữ liệu cột ID.

In [448…
```
# dropping duplicates by considering all columns other than ID
cols_other_than_id = list(df.columns)[1:]
df.drop_duplicates(subset=cols_other_than_id, inplace=True)
```

1.2.2.4. Fixing data type - Xử lý sai kiểu dữ liệu

Trong tập dữ liệu sẽ có thể có những giá trị không được lưu trữ theo kiểu dữ liệu chính xác.

Ví dụ với loại lỗi dữ liệu phổ biến là ngày tháng, ta sẽ kiểm tra cột `timestamp`:

In [449…
```
df.timestamp.dtype
```

Out[449]:
```
dtype('O')
```

Kết quả trả về kiểu dữ liệu là "Object", ta hiểu rằng dữ liệu trong cột không đồng bộ và không đúng kiểu dữ liệu ta mong muốn, do đó, sửa kiểu dữ liệu thành `pandas.DateTime`:

In [450…
```
# converting timestamp to datetime format
df['timestamp'] = pd.to_datetime(df.timestamp, format='%Y-%m-%d')
```

Tương tự cho các cột còn lại.

CHƯƠNG 2: HỌC CÓ GIÁM SÁT

2.1. Linear Regression

Ở chương 1, ta đã đề cập tới thuật toán **Linear Regression** thông qua thư viện `sklearn` trong Python. Ở chương này, ta sẽ nghiên cứu tập trung vào dạng nâng cao hơn, gọi là **Multiple Linear Regression**.

Để nghiên cứu thuật toán, ta sẽ sử dụng tập dữ liệu Boston trong `sklearn.datasets`.

In [451…

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
dataset = load_boston()
df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
unction load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be remove
d in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Mô tả theo từng feature của tập dữ liệu:

In [452…
```
print(dataset.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 1
4) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon U
niversity.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address r
egression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and So
urces of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceeding
s on the Tenth International Conference of Machine Learning, 236-243, University of Mass
achusetts, Amherst. Morgan Kaufmann.
```

Thêm cột `MEDV` (là thông tin cần tìm đối với mỗi bản ghi):

In [453…
```
df['MEDV'] = dataset.target
df.head()
```

Out[453]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Data cleaning

In [454…

```
print(df.info())
print(df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
None
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

=> Tập dữ liệu tốt.

Feature selection:

Bước đầu ta cần kiểm tra mối tương quan:

In [455…

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

```
             CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM     1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  0.352734
ZN      -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
INDUS    0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
CHAS    -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
NOX      0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
RM      -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
AGE      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
DIS     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
RAD      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
TAX      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
PTRATIO  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
B       -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
LSTAT    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
MEDV    -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955

               DIS       RAD       TAX   PTRATIO         B     LSTAT      MEDV
CRIM    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621 -0.388305
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
MEDV     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

In [456…

```
print(df.corr().abs().nlargest(3, 'MEDV').index)
print(df.corr().abs().nlargest(3, 'MEDV').values[:,13])
```

```
Index(['MEDV', 'LSTAT', 'RM'], dtype='object')
[1.         0.73766273 0.69535995]
```

=> Ta thấy `LSTAT` và `RM` là 2 feature có tính tương quan cao nhất so với `MEDV`, ta chọn 2 feature này để đào tạo model.

2.1.1. Multiple Regression

Thay vì đào tạo model với 1 nhãn như bình thường, **Multiple Regression** sẽ đào tạo model với nhiều nhãn cùng lúc.

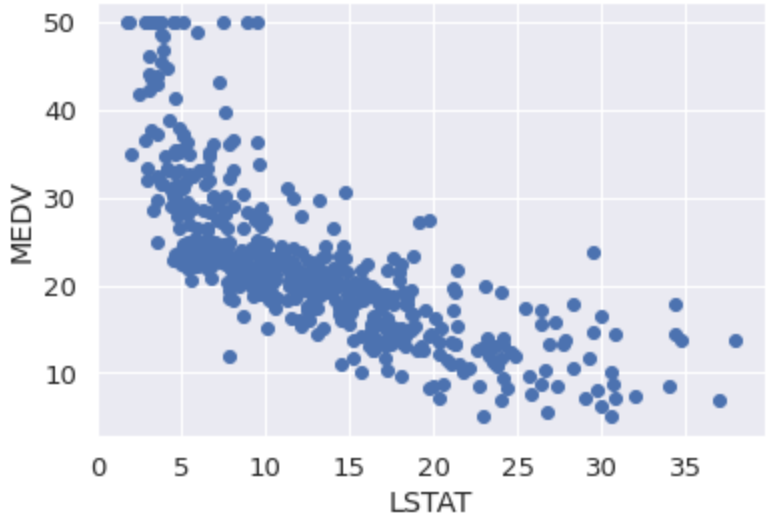
Các đồ thị biểu diễn kết quả dựa theo từng và tất cả feature sử dụng:

In [457…

```
plt.scatter(df['LSTAT'], df['MEDV'], marker='o')
plt.xlabel('LSTAT')
plt.ylabel('MEDV')
```

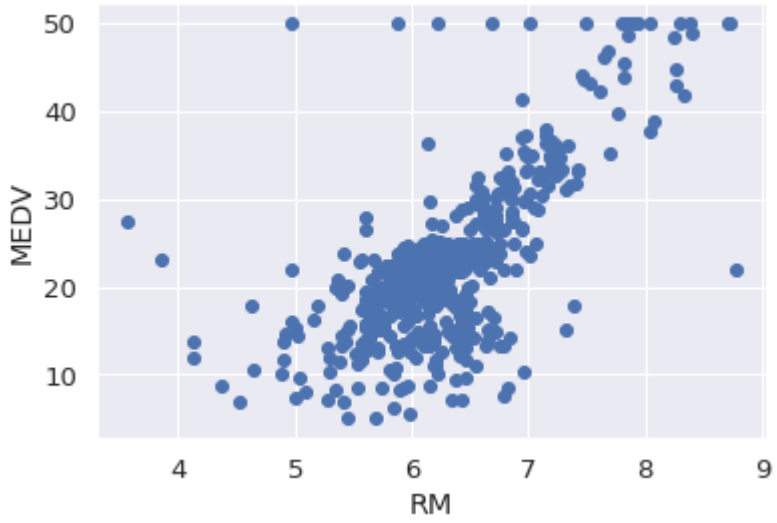
Out[457]:

```
Text(0, 0.5, 'MEDV')
```

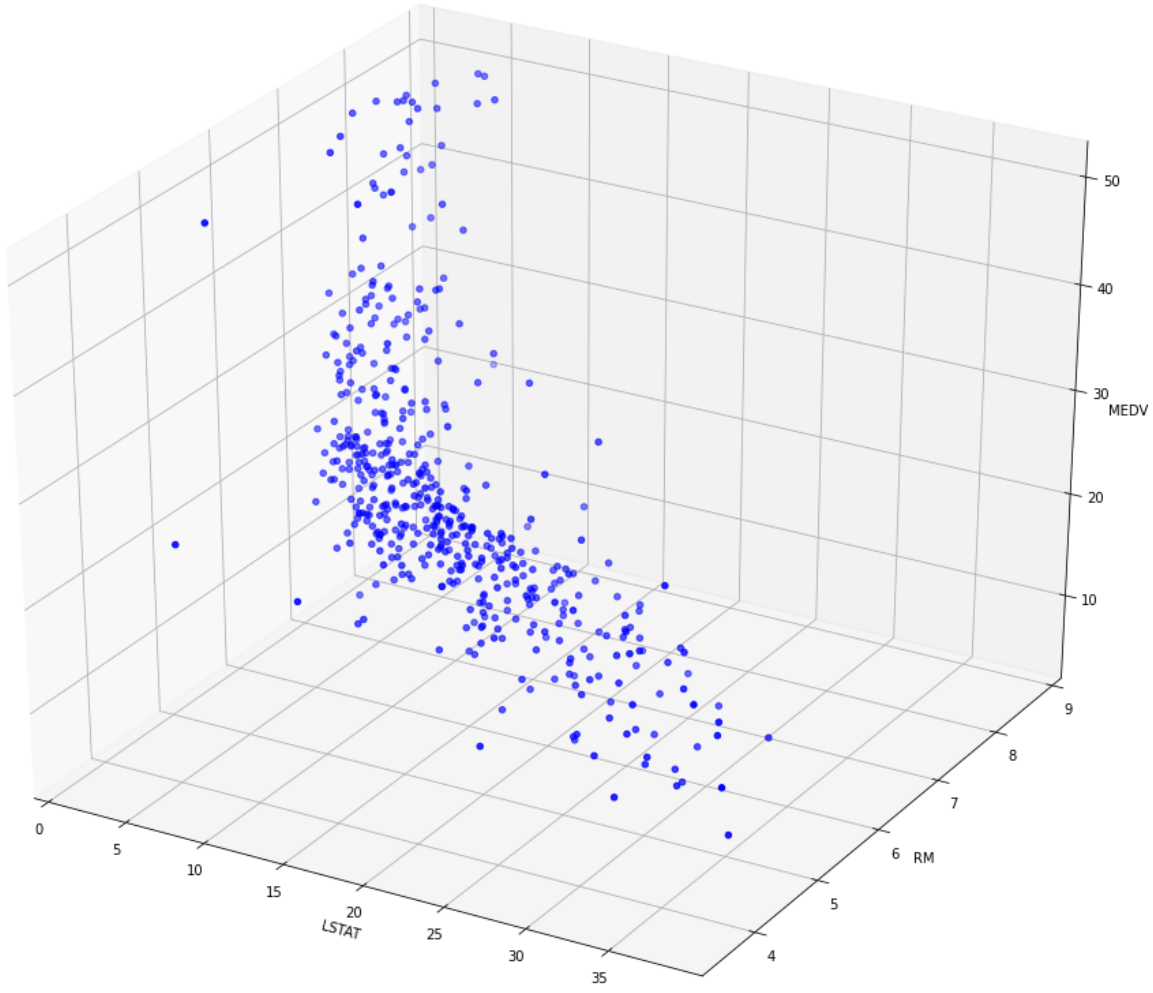


In [458…
```
plt.scatter(df['RM'], df['MEDV'], marker='o')
plt.xlabel('RM')
plt.ylabel('MEDV')
```

Out[458]:
```
Text(0, 0.5, 'MEDV')
```



In [309…
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['LSTAT'],
df['RM'],
df['MEDV'],
c='b')
ax.set_xlabel("LSTAT")
ax.set_ylabel("RM")
ax.set_zlabel("MEDV")
plt.show()
```



Training model:

- Chia các tập train và test:

In [311…
```
x = pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns = ['LSTAT','RM'])
Y = df['MEDV']

from sklearn.model_selection import train_test_split
x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size = 0.3, random_state=
```

- Sử dụng `LinearRegression` của `sklearn`:

In [312…
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, Y_train)
```

Out[312]:
```
LinearRegression()
```

- Kiểm tra model (kết quả càng gần 1 càng tốt):

In [314…
```
price_pred = model.predict(x_test)
print('R-Squared: %.4f' % model.score(x_test, Y_test))
```

```
R-Squared: 0.6162
```

- Trực quan hóa kết quả đã có và kết quả thu được:

In [315…
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(Y_test, price_pred)
print(mse)
plt.scatter(Y_test, price_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted prices")
plt.title("Actual prices vs Predicted prices")
```

```
36.49422110915324
```
Out[315]:
```
Text(0.5, 1.0, 'Actual prices vs Predicted prices')
```

Vì ta sử dụng 2 feature để đào tạo model, nên công thức hồi quy sẽ có dạng $A + Bx_1 + Cx_2$.

In [316…
```
print(model.intercept_)
print(model.coef_)
```

```
0.38437936780346504
[-0.65957972  4.83197581]
```

=>

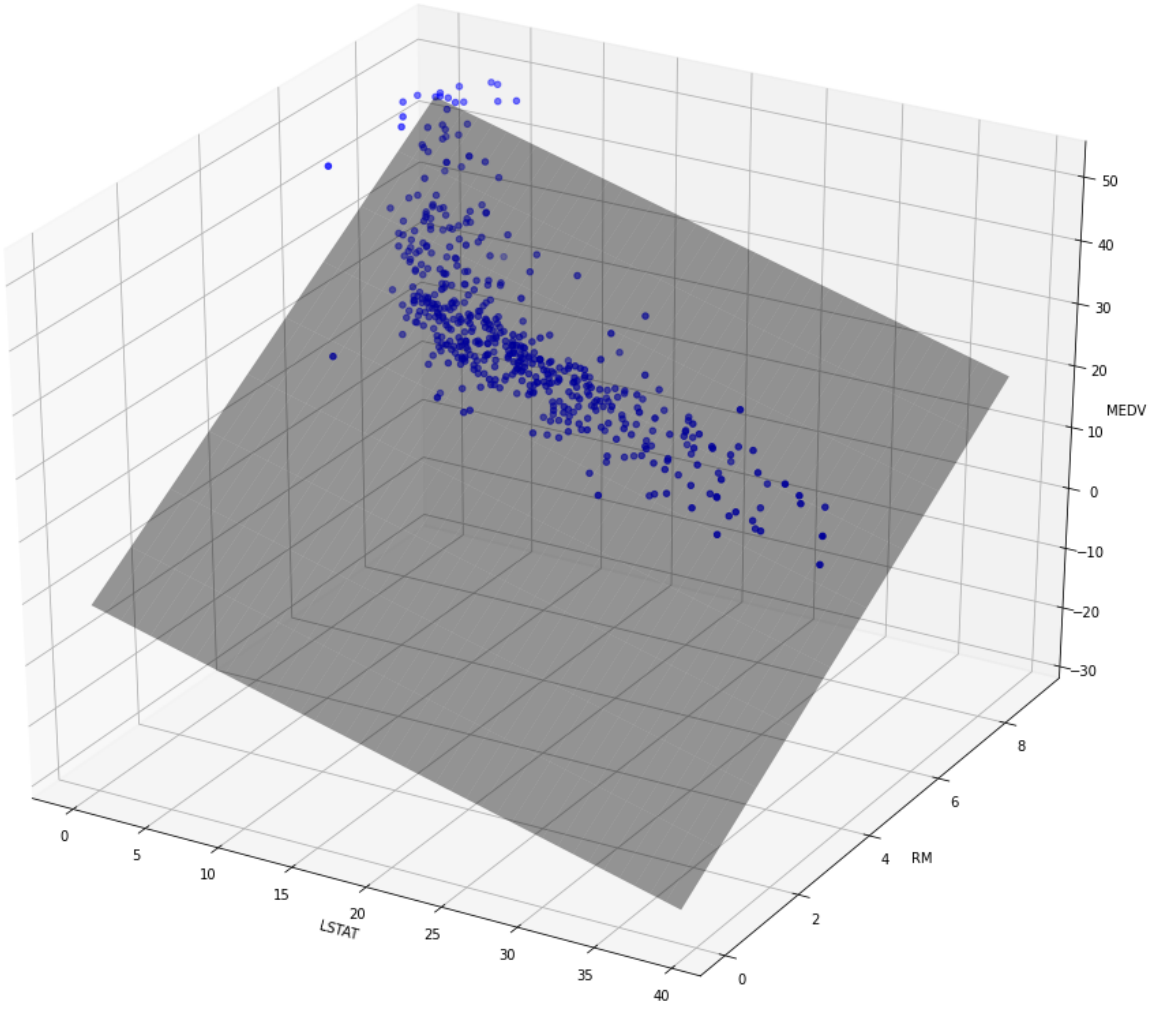
- A = 0.38437936780346504
- B = -0.65957972
- C = -0.65957972

Trực quan hóa trong không gian 3 chiều:

In [318…
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import load_boston
dataset = load_boston()
df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
df['MEDV'] = dataset.target

x = pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns = ['LSTAT','RM'])
Y = df['MEDV']

fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')
```

```
ax.scatter(x['LSTAT'],
           x['RM'],
           Y,
           c='b')

ax.set_xlabel("LSTAT")
ax.set_ylabel("RM")
ax.set_zlabel("MEDV")

#---create a meshgrid of all the values for LSTAT and RM---
x_surf = np.arange(0, 40, 1) #---for LSTAT---
y_surf = np.arange(0, 10, 1) #---for RM---
x_surf, y_surf = np.meshgrid(x_surf, y_surf)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x, Y)

#---calculate z(MEDC) based on the model---
z = lambda x,y: (model.intercept_ + model.coef_[0] * x + model.coef_[1] * y)
ax.plot_surface(x_surf, y_surf, z(x_surf,y_surf),
                rstride=1,
                cstride=1,
                color='None',
                alpha = 0.4)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
unction load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be remove
d in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```



2.1.2. Polynomial Regression

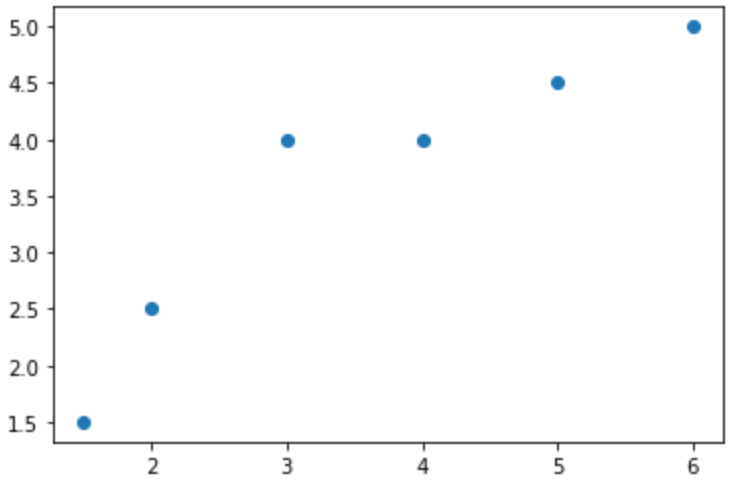
Ở phần vừa rồi, tuy rằng kết quả có thể chấp nhận được, nhưng độ chính xác như thế chưa đủ thuyết phục. Đó là do đôi khi một đường hồi quy tuyến tính không phải giải pháp tốt nhất để xác định mối quan hệ giữa các feature cùng nhãn kết quả một cách chính xác. Vì vậy, một số trường hợp, ta nên sử dụng đường cong.

Tạo tập dữ liệu:

In [319…
```
with open('polynomial.csv', 'w') as f:
    f.write(
        '\n'.join([
            "x,y",
            "1.5,1.5",
            "2,2.5",
            "3,4",
            "4,4",
            "5,4.5",
            "6,5"
            ]
        )
    )
```

In [320…
```
df = pd.read_csv('polynomial.csv')
plt.scatter(df.x,df.y)
```

Out[320]:
```
<matplotlib.collections.PathCollection at 0x7f17848b1e50>
```

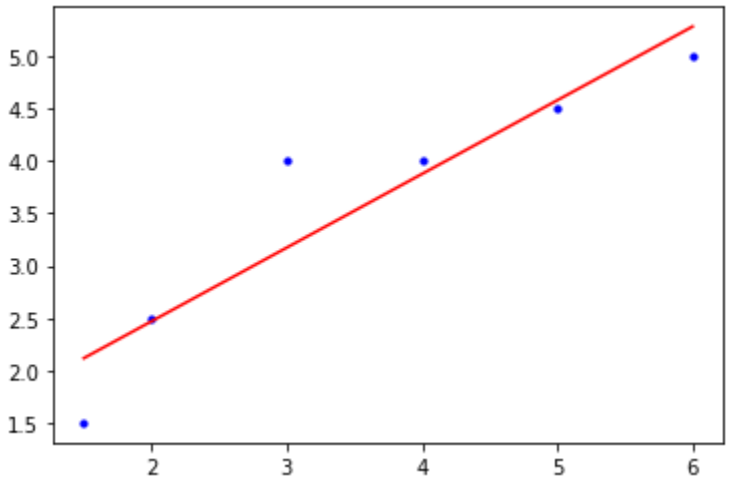


Ta có thể sử dụng `LinearRegression` trong `sklearn` để thử đánh giá:

In [321…

```
model = LinearRegression()
x = df.x[0:6, np.newaxis] #---convert to 2D array---
y = df.y[0:6, np.newaxis] #---convert to 2D array---
model.fit(x,y)
#---perform prediction---
y_pred = model.predict(x)
#---plot the training points---
plt.scatter(x, y, s=10, color='b')
#---plot the straight line---
plt.plot(x, y_pred, color='r')
plt.show()
#---calculate R-Squared---
print('R-Squared for training set: %.4f' % model.score(x,y))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: Support f
or multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in
a future version.  Convert to a numpy array before indexing instead.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: Support f
or multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in
a future version.  Convert to a numpy array before indexing instead.
  This is separate from the ipykernel package so we can avoid doing imports until
```




```
R-Squared for training set: 0.8658
```

=> Ta cần độ chính xác cao hơn <=> đường hồi quy phải cong, khi đó, công thức đường hồi quy sẽ có dạng:

$$Y = A_0 + A_1 x + A_2 x^2 + \ldots + A_n x^n$$

Trong `sklearn` có class `PolynomialFeatures` có thể giải hàm hồi quy đa thức:

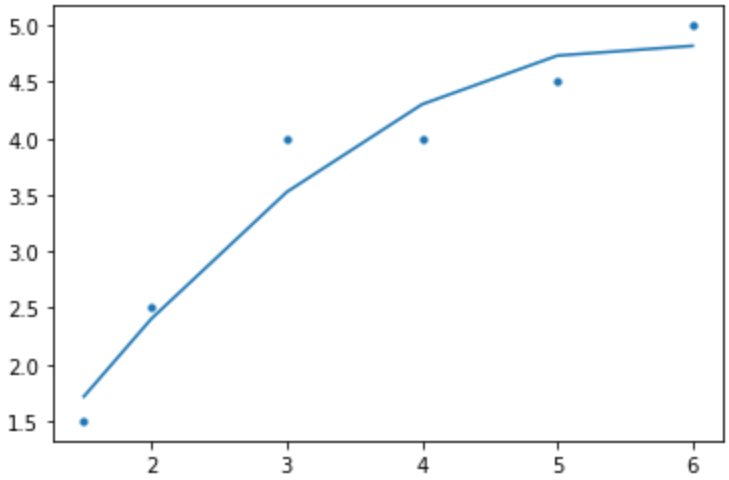
In [324…
```
from sklearn.preprocessing import PolynomialFeatures
degree = 2
polynomial_features = PolynomialFeatures(degree = degree)
x_poly = polynomial_features.fit_transform(x)
print(x_poly)
print(polynomial_features.get_feature_names('x'))
```

```
[[ 1.    1.5   2.25]
 [ 1.    2.    4.  ]
 [ 1.    3.    9.  ]
 [ 1.    4.   16.  ]
 [ 1.    5.   25.  ]
 [ 1.    6.   36.  ]]
['1', 'x', 'x^2']
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
unction get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Giờ ta sử dụng `LinearRegression` để train model:

In [326…
```
model = LinearRegression()
model.fit(x_poly, y)
y_poly_pred = model.predict(x_poly)
#---plot the points---
plt.scatter(x, y, s=10)
#---plot the regression line---
plt.plot(x, y_poly_pred)
plt.show()
print(model.intercept_)
print(model.coef_)
print('R-Squared for training set: %.4f' % model.score(x_poly,y))
```




```
[-0.87153912]
[[ 0.          1.98293207 -0.17239897]]
R-Squared for training set: 0.9474
```

=> Kết quả đã chính xác hơn nhiều. Nếu sử dụng bậc cao hơn, kết quả sẽ chính xác hơn.

2.1.3. Polynomial Multiple Regression

Sau khi tìm hiểu 2 luồng kiến thức, ta sẽ áp dụng nó vào tập dữ liệu Boston:

In [327…
```
import matplotlib.pyplot as plt
import pandas as pd
```

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
dataset = load_boston()
df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
df['MEDV'] = dataset.target
x = pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns = ['LSTAT','RM'])
Y = df['MEDV']
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
unction load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be remove
d in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Chia tập train và test:

In [328…
```
from sklearn.model_selection import train_test_split
x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size = 0.3, random_state=
```

Sử dụng `PolynomialFeatures` với bậc là 2:

In [329…
```
degree = 2
polynomial_features= PolynomialFeatures(degree = degree)
x_train_poly = polynomial_features.fit_transform(x_train)
```

Train model:

In [331…
```
model = LinearRegression()
```

```
model.fit(x_train_poly, Y_train)
```

Out[331]:
```
LinearRegression()
```

Đánh giá:

In [332…
```
x_test_poly = polynomial_features.fit_transform(x_test)
print('R-Squared: %.4f' % model.score(x_test_poly, Y_test))
```

```
R-Squared: 0.7340
```

=> Đã chính xác hơn.

2.2. Classification Using Logistic Regression

Không giống như hồi quy tuyến tính, hồi quy logistic không cố gắng dự đoán giá trị đầu ra sau khi cho trước một tập hợp các đầu vào. Thay vào đó, đầu ra của hồi quy logistic là xác suất của một điểm đầu vào nhất định thuộc một lớp cụ thể. Đầu ra của hồi quy logistic luôn nằm trong [0,1].

2.2.1. Tỉ lệ cược

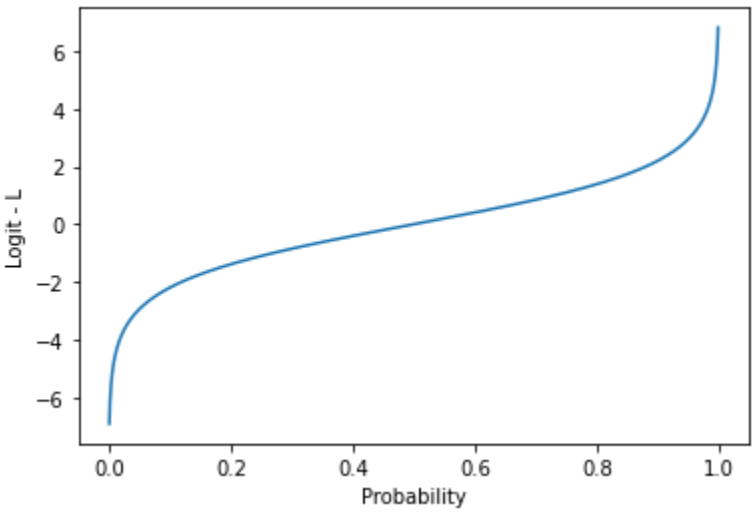
Tỉ lệ cược là tỉ lệ giữa xác suất thành công và xác suất thất bại: $\frac{P}{1-P}$ với P là xác suất thành công.

2.2.2. Hàm logit

$L = ln\frac{P}{1-P}$

In [334…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
def logit(x):
    return np.log( x / (1 - x) )
x = np.arange(0.001,0.999, 0.0001)
y = [logit(n) for n in x]
plt.plot(x,y)
plt.xlabel("Probability")
plt.ylabel("Logit - L")
```

Out[334]:
```
Text(0, 0.5, 'Logit - L')
```



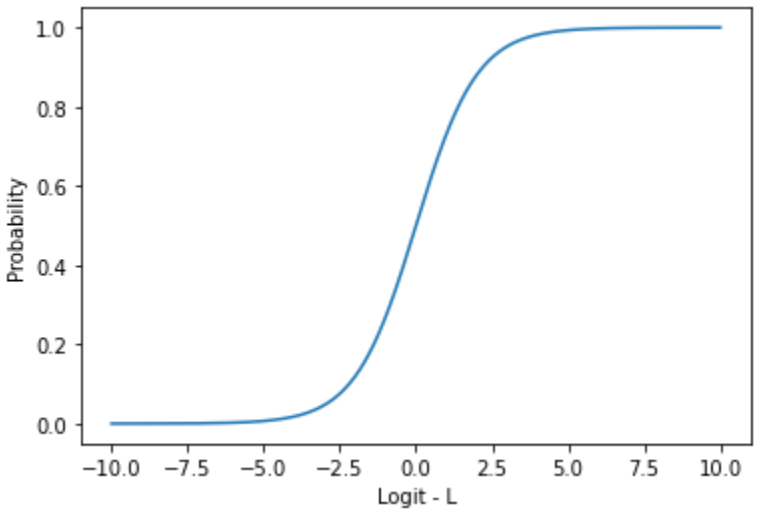
2.2.3. Đường cong sigmoid

Đối với đường cong logit, trục x là xác suất và trục y là phạm vi số thực. Đối với hồi quy logistic, những gì ta cần là một hàm ánh xạ các số trên hệ thống số thực với các xác suất, vì vậy ta cần lật các trục của hàm logit.

$$P = \frac{1}{1-e^{-L}}$$

In [335…
```
def sigmoid(x):
    return (1 / (1 + np.exp(-x)))
x = np.arange(-10, 10, 0.0001)
y = [sigmoid(n) for n in x]
plt.plot(x,y)
plt.xlabel("Logit - L")
plt.ylabel("Probability")
```

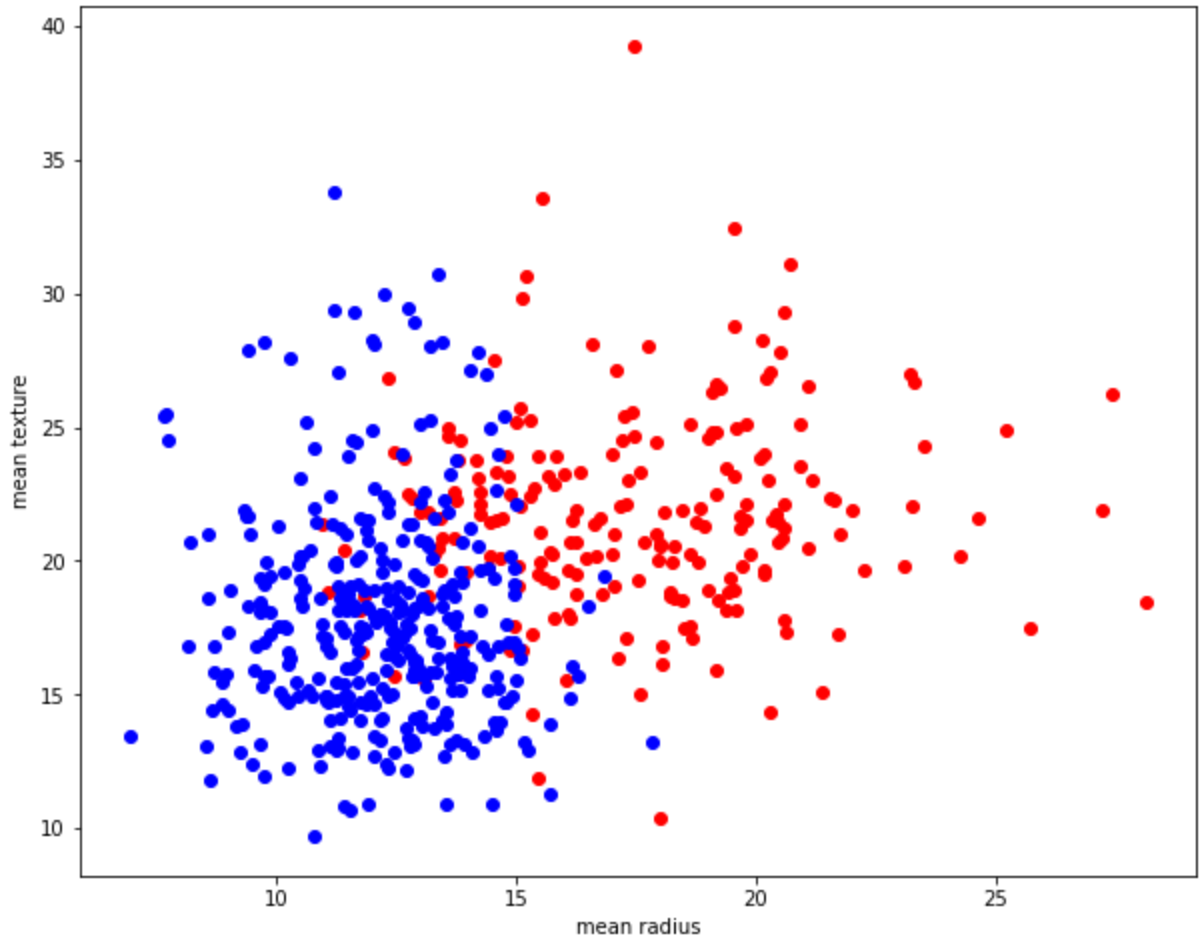
Out[335]:
```
Text(0, 0.5, 'Probability')
```



2.2.4. Triển khai

Để triển khai thuật toán, ta sẽ sử dụng tập dữ liệu "Breast Cancer Wisconsin (Diagnostic)" có trong `sklearn` :

In [336…
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

#---copy from dataset into a 2-d list---
X = []
for target in range(2):
    X.append([[], []])
    for i in range(len(cancer.data)): # target is 0 or 1
        if cancer.target[i] == target:
            X[target][0].append(cancer.data[i][0]) # first feature - mean radius
            X[target][1].append(cancer.data[i][1]) # second feature — mean texture
colours = ("r", "b") # r: malignant, b: benign
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111)
for target in range(2):
    ax.scatter(X[target][0],
               X[target][1],
               c=colours[target])
ax.set_xlabel("mean radius")
ax.set_ylabel("mean texture")
plt.show()
```



2.2.4.1. Huấn luyện sử dụng 1 feature

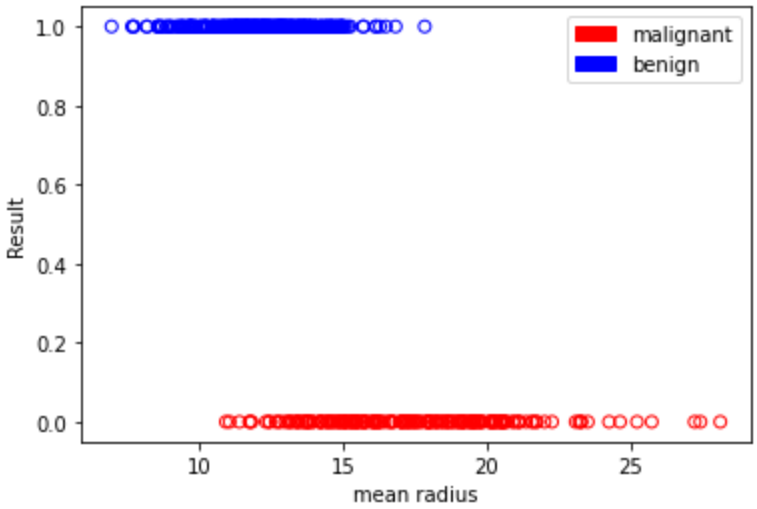
Đầu tiên chúng ta sẽ sử dụng `mean radius` để chẩn đoán xem một khối u có phải là ung thư hay không:

In [345…

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer() # Load dataset
x = cancer.data[:,0] # mean radius
y = cancer.target # 0: malignant, 1: benign
colors = {0:'red', 1:'blue'} # 0: malignant, 1: benign
plt.scatter(x,y,
        facecolors='none',
        edgecolors=pd.DataFrame(cancer.target)[0].apply(lambda x:colors[x]),
        cmap=colors)
plt.xlabel("mean radius")
plt.ylabel("Result")
red = mpatches.Patch(color='red', label='malignant')
blue = mpatches.Patch(color='blue', label='benign')
plt.legend(handles=[red, blue], loc=1)
```

Out[345]:

```
<matplotlib.legend.Legend at 0x7f178abcbdd0>
```



Sử dụng `LogisticRegression` có trong `sklearn` để áp dụng thuật toán:

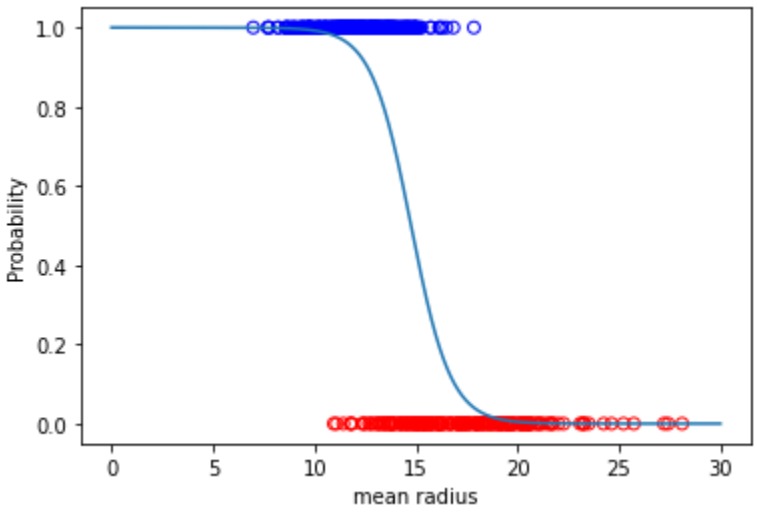
In [346…
```
from sklearn import linear_model
import numpy as np
log_regress = linear_model.LogisticRegression()
#---train the model---
log_regress.fit(X = np.array(x).reshape(len(x),1), y = y)
```

Out[346]:
```
LogisticRegression()
```

Trực quan hóa:

In [347…
```
def sigmoid(x):
  return (1 / (1 + np.exp(-(log_regress.intercept_[0] + (log_regress.coef_[0][0] * x)))))
x1 = np.arange(0, 30, 0.01)
y1 = [sigmoid(n) for n in x1]
plt.scatter(x,y,
      facecolors='none',
      edgecolors=pd.DataFrame(cancer.target)[0].apply(lambda x:
      colors[x]),
      cmap=colors)
plt.plot(x1,y1)
plt.xlabel("mean radius")
plt.ylabel("Probability")
```

Out[347]:
```
Text(0, 0.5, 'Probability')
```



2.2.4.2. Huấn luyện sử dụng tất cả các feature

In [340…
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer() # Load dataset
```

Chia tập train, test theo tỉ lệ 3/1:

In [341…
```
from sklearn.model_selection import train_test_split
train_set, test_set, train_labels, test_labels = train_test_split(
              cancer.data, # features
              cancer.target, # labels
              test_size = 0.25, # split ratio
              random_state = 1, # set random seed
              stratify = cancer.target) # randomize based on labels
```

Train model:

In [342…
```
from sklearn import linear_model
x = train_set[:,0:30] # mean radius
y = train_labels # 0: malignant, 1: benign
log_regress = linear_model.LogisticRegression()
log_regress.fit(X = x, y = y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Out[342]:
```
LogisticRegression()
```

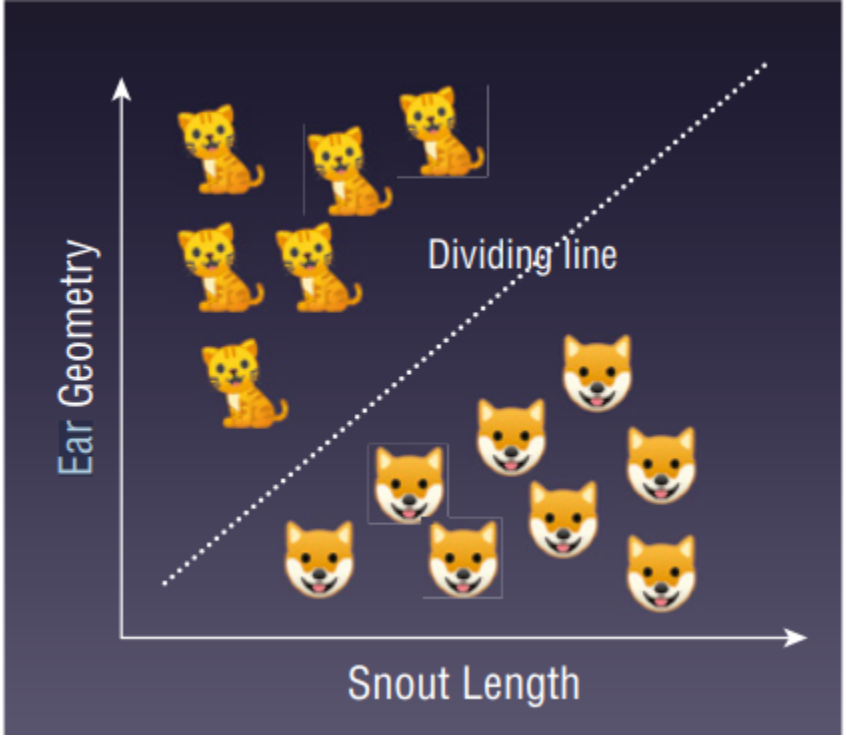
Tỉ lệ chính xác của model:

In [343…
```
print(log_regress.score(X = test_set , y = test_labels))
```

```
0.9370629370629371
```

2.3. Classification Using Support Vector Machines

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

phân chia chúng. 

Để triển khai thuật toán này, ta sẽ sử dụng tập dữ liệu tự tạo như sau:

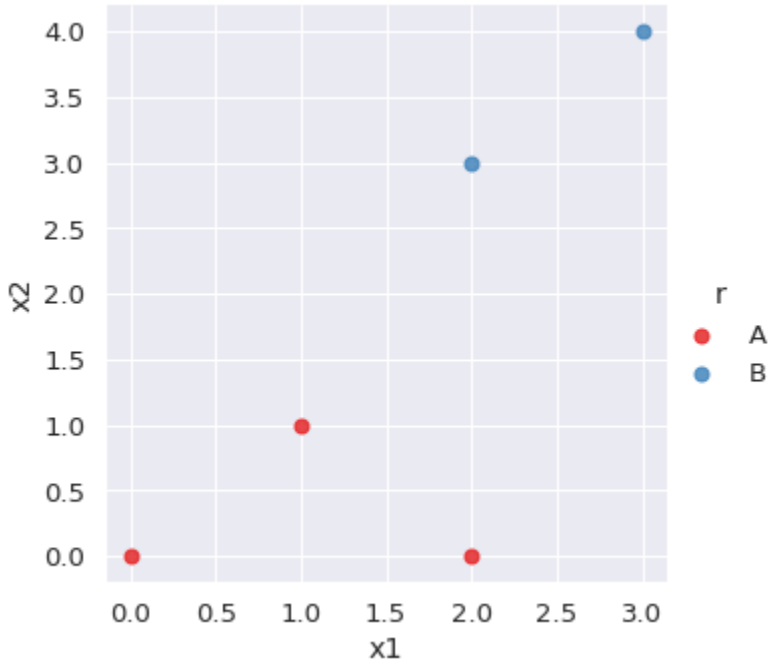
In [348…
```
with open('svm.csv', 'w') as f:
    f.write(
        '\n'.join([
            "x1,x2,r",
            "0,0,A",
            "1,1,A",
            "2,3,B",
            "2,0,A",
            "3,4,B"
            ]
        )
    )
```

Đọc và biểu thị dữ liệu:

In [349…
```
import pandas as pd
import numpy as np
import seaborn as sns; sns.set(font_scale=1.2)
import matplotlib.pyplot as plt
data = pd.read_csv('svm.csv')
sns.lmplot('x1', 'x2',
        data=data,
        hue='r',
        palette='Set1',
        fit_reg=False,
        scatter_kws={"s": 50})
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
e following variables as keyword args: x, y. From version 0.12, the only valid positiona
l argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  FutureWarning
```

Out[349]:
```
<seaborn.axisgrid.FacetGrid at 0x7f17852d4150>
```



Sử dụng class `SVC` trong `sklearn`:

In [350…

```
from sklearn import svm
#---Converting the Columns as Matrices---
points = data[['x1','x2']].values
result = data['r']
clf = svm.SVC(kernel = 'linear')
clf.fit(points, result)
print('Vector of weights (w) = ',clf.coef_[0])
print('b = ',clf.intercept_[0])
print('Indices of support vectors = ', clf.support_)
print('Support vectors = ', clf.support_vectors_)
print('Number of support vectors for each class = ', clf.n_support_)
print('Coefficients of the support vector in the decision function = ', np.abs(clf.dual_
```

```
Vector of weights (w) =  [0.4 0.8]
b =  -2.2
Indices of support vectors =  [1 2]
Support vectors =  [[1. 1.]
 [2. 3.]]
Number of support vectors for each class =  [1 1]
Coefficients of the support vector in the decision function =  [[0.4 0.4]]
```

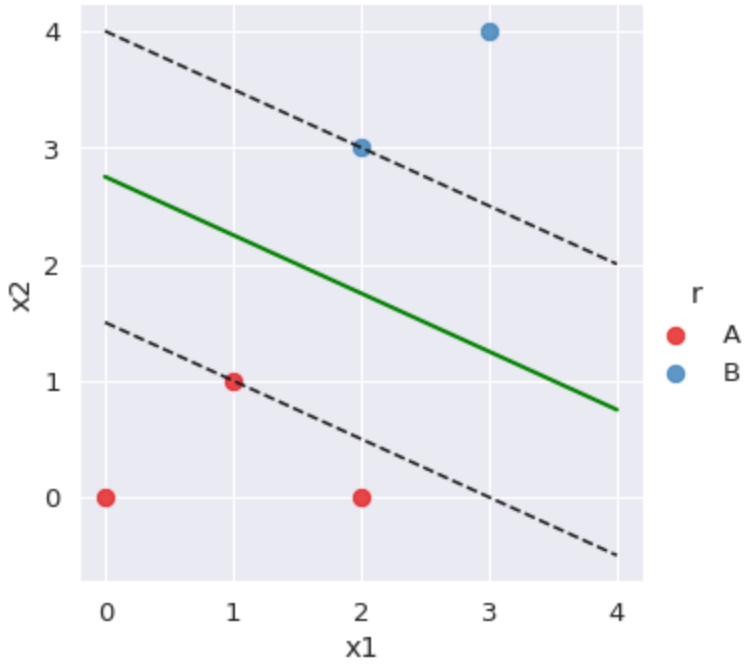
In [351…

```
#---w is the vector of weights---
w = clf.coef_[0]
#---find the slope of the hyperplane---
slope = -w[0] / w[1]
b = clf.intercept_[0]
#---find the coordinates for the hyperplane---
xx = np.linspace(0, 4)
yy = slope * xx - (b / w[1])
#---plot the margins---
s = clf.support_vectors_[0] #---first support vector---
yy_down = slope * xx + (s[1] - slope * s[0])
s = clf.support_vectors_[-1] #---last support vector---
yy_up = slope * xx + (s[1] - slope * s[0])
#---plot the points---
sns.lmplot('x1', 'x2', data=data, hue='r', palette='Set1',
fit_reg=False, scatter_kws={"s": 70})
#---plot the hyperplane---
plt.plot(xx, yy, linewidth=2, color='green');
#---plot the 2 margins---
```

```
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
e following variables as keyword args: x, y. From version 0.12, the only valid positiona
l argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  FutureWarning
```

Out[351]:
```
[<matplotlib.lines.Line2D at 0x7f1784e1b2d0>]
```



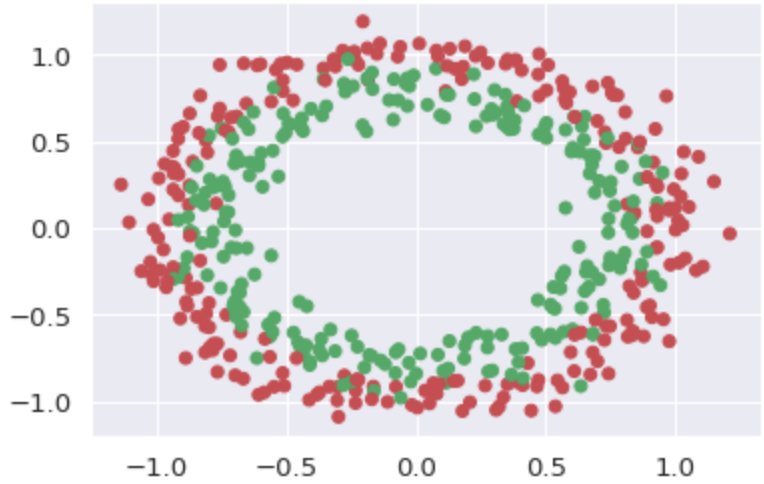
In [352…
```
print(clf.predict([[3,3]])[0])
print(clf.predict([[4,0]])[0])
print(clf.predict([[2,2]])[0])
print(clf.predict([[1,2]])[0])
```

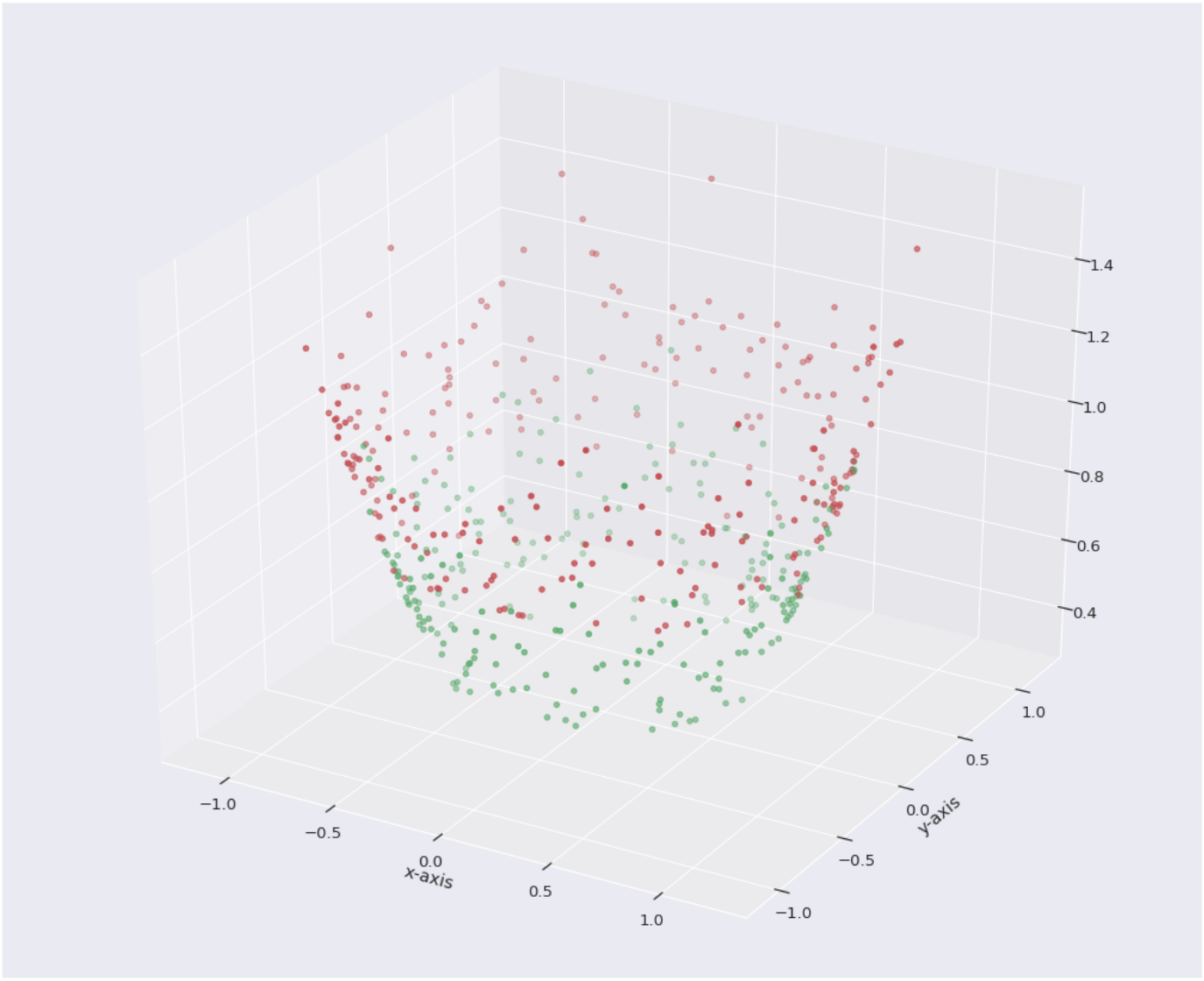
```
B
A
B
A
```

Kernel Trick:

Đôi khi, các điểm trong tập dữ liệu không thể phân chia theo một đường thẳng. Khi đó, ta có thể mở rộng thêm chiều không gian mới và xác định mặt chia thông qua đó.

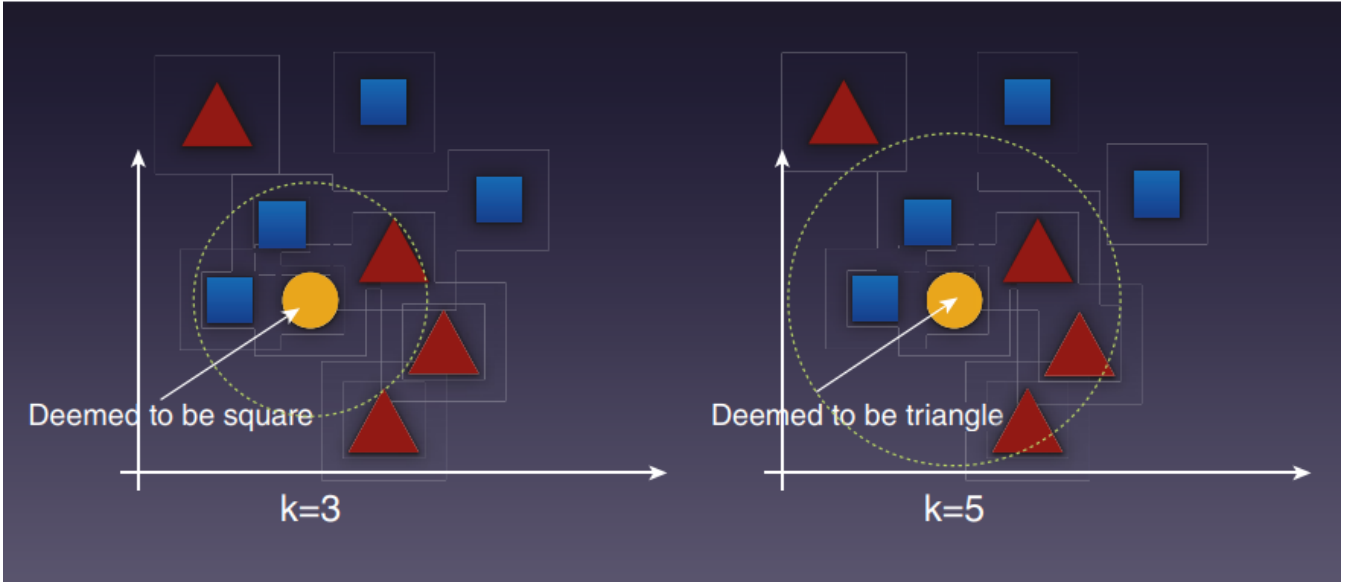
In [353…
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_circles
#---X is features and c is the class labels---
X, c = make_circles(n_samples=500, noise=0.09)
rgb = np.array(['r', 'g'])
plt.scatter(X[:, 0], X[:, 1], color=rgb[c])
plt.show()
fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')
z = X[:,0]**2 + X[:,1]**2
ax.scatter(X[:, 0], X[:, 1], z, color=rgb[c])
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()
```





2.4. Classification Using K-Nearest Neighbors (KNN)

Hoạt động bằng cách so sánh với K mẫu láng giềng gần nhất. Dự đoán phân loại cho cá thể ban đầu sẽ là mẫu có xuất hiện nhiều nhất trong K mẫu láng giềng gần nhất.



2.4.1. Triển khai trong Python

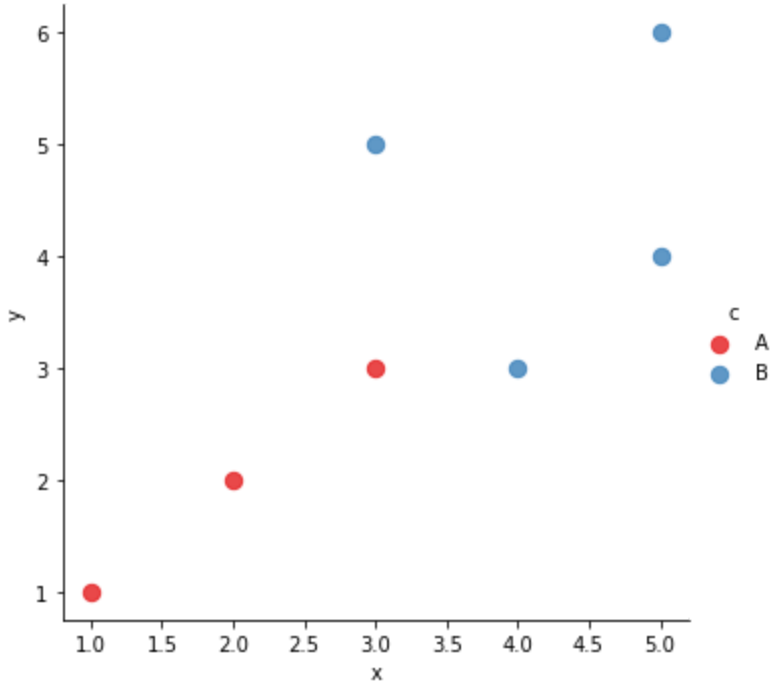
Tạo một tập dữ liệu ví dụ:

In [280…
```
with open('knn.csv', 'w') as f:
    f.write(
        '\n'.join([
            "x,y,c",
            "1,1,A",
            "2,2,A",
            "4,3,B",
            "3,3,A",
            "3,5,B",
            "5,6,B",
            "5,4,B"
            ]
        )
    )
```

Load data:

In [281…
```
import pandas as pd
import numpy as np
import operator
import seaborn as sns
import matplotlib.pyplot as plt

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
e following variables as keyword args: x, y. From version 0.12, the only valid positiona
l argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  FutureWarning
```



Tạo hàm tính khoảng cách Euclidean giữa 2 điểm:

In [282…

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

Thuật toán:

In [283…

```
def knn(training_points, test_point, k):
    distances = {}

    # Chiều cần xử lý
    dimension = test_point.shape[1]

    # Tính khoảng cách giữa mỗi cặp điểm trong tập train và tập test
    for x in range(len(training_points)):
        dist = euclidean_distance(test_point, training_points.iloc[x], dimension)
        # Lưu lại khoảng cách tính được
        distances[x] = dist[0]

    # Sắp xếp các khoảng cách
    sorted_d = sorted(distances.items(), key=operator.itemgetter(1))
    # Tạo mảng lưu các láng giềng
    neighbors = []
    # Tìm k láng giềng gần nhất
    for x in range(k):
        neighbors.append(sorted_d[x][0])
        # Với mỗi láng giềng tìm thấy, tìm class của nó
    class_counter = {}
    for x in range(len(neighbors)):
    # Tìm ra class cho điểm cụ thể của nó
        cls = training_points.iloc[neighbors[x]][-1]
        if cls in class_counter:
            class_counter[cls] += 1
        else:
            class_counter[cls] = 1
    # Sắp xếp lại dict đếm mẫu trong class theo thứ tự giảm dần
    sorted_counter = sorted(class_counter.items(), key=operator.itemgetter(1), reverse=T
```

```
    # Trả về class có nhiều mẫu nhất
    return(sorted_counter[0][0], neighbors)
```

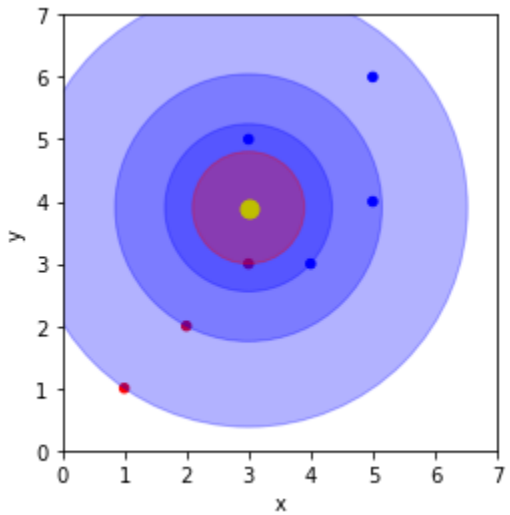
Thử đưa ra dự đoán:

In [284…
```
test_set = [[3,3.9]]
test = pd.DataFrame(test_set)
cls,neighbors = knn(data, test, 5)
print("Predicted Class: " + cls)
```

```
Predicted Class: B
```

Để hình dung rõ hơn khi k thay đổi, kết quả sẽ thay đổi như thế nào, ta sẽ vẽ sơ đồ trực quan hóa:

In [286…
```
#---generate the color map for the scatter plot if column 'c' is A, then use Red, else u
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')
for k in range(7,0,-2):
    cls,neighbors = knn(data, test, k)
    furthest_point = data.iloc[neighbors].tail(1)
    #---draw a circle connecting the test point and the furthest point---
    radius = euclidean_distance(test, 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()
```

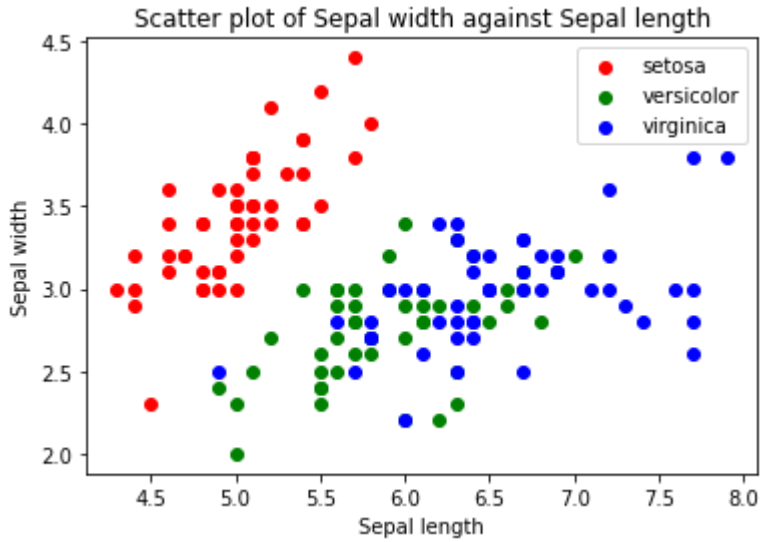


2.4.2. Sử dụng `KNeighborsClassifier` trong thư viện `sklearn`

Load data:

In [287…
```
import pandas as pd
import numpy as np
import matplotlib.patches as mpatches
from sklearn import svm, datasets
import matplotlib.pyplot as plt
iris = datasets.load_iris()
X = iris.data[:, :2] # take the first two features
y = iris.target
```

```
#---plot the points---
colors = ['red', 'green', 'blue']
for color, i, target in zip(colors, [0, 1, 2], iris.target_names):
    plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('Scatter plot of Sepal width against Sepal length')
plt.show()
```

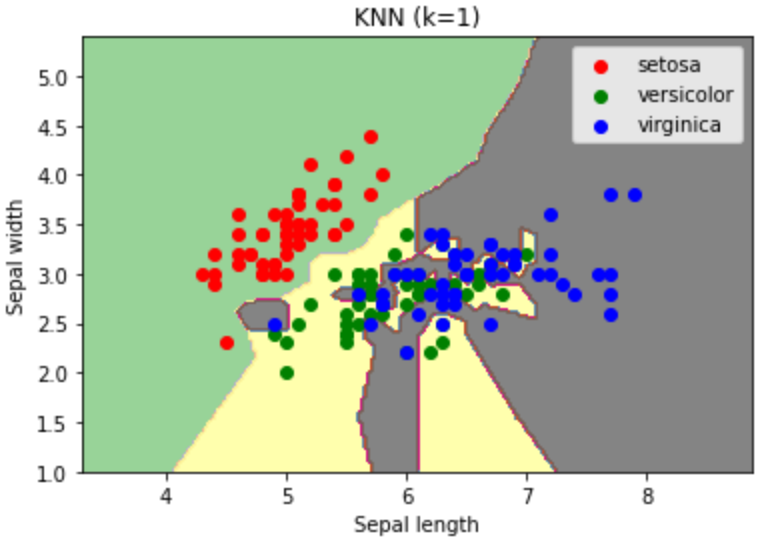


Triển khai:

In [288…

```
from sklearn.neighbors import KNeighborsClassifier
k = 1
# Khởi tạo mô hình
knn = KNeighborsClassifier(n_neighbors=k)
# Đào tạo
knn.fit(X, y)

#---min and max for the first feature---
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
#---min and max for the second feature---
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
#---step size in the mesh---
h = (x_max / x_min)/100
#---make predictions for each of the points in xx,yy---
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])

#---draw the result using a color plot---
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Accent, alpha=0.8)
#---plot the training points---
colors = ['red', 'green', 'blue']
for color, i, target in zip(colors, [0, 1, 2], iris.target_names):
    plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title(f'KNN (k={k})')
plt.legend(loc='best', shadow=False, scatterpoints=1)
predictions = knn.predict(X)
#--classifications based on predictions---
print(np.unique(predictions, return_counts=True))
```

```
(array([0, 1, 2]), array([50, 54, 46]))
```



2.4.3. Cross-Validation - Kiểm chứng chéo

Thử kiểm chứng chéo 10 lần với mỗi giá trị của k:

In [290…

```
from sklearn.model_selection import cross_val_score
#---holds the cv (cross-validates) scores---
cv_scores = []
#---use all features---
X = iris.data[:, :4]
y = iris.target
#---number of folds---
folds = 10
#---creating odd list of K for KNN---
ks = list(range(1,int(len(X) * ((folds - 1)/folds))))
#---remove all multiples of 3---
ks = [k for k in ks if k % 3 != 0]
#---perform k-fold cross validation---
for k in ks:
    knn = KNeighborsClassifier(n_neighbors=k)
    #---performs cross-validation and returns the average accuracy---
    scores = cross_val_score(knn, X, y, cv=folds, scoring='accuracy')
    mean = scores.mean()
    cv_scores.append(mean)
```

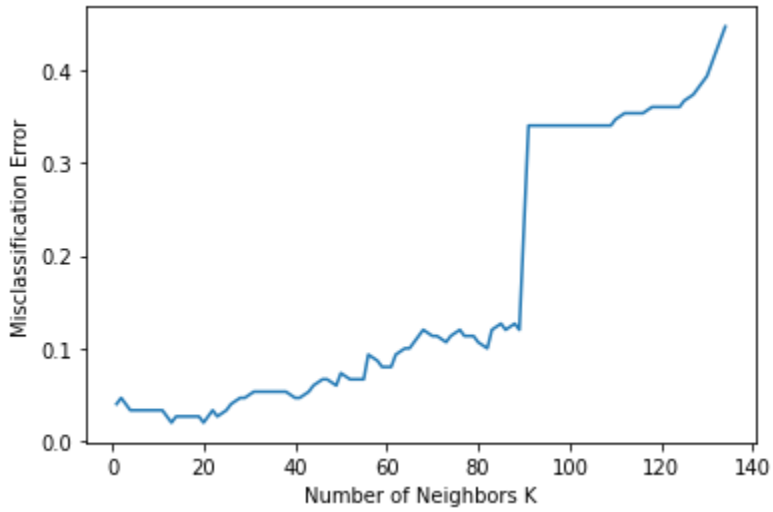
Sử dụng K tối ưu và vẽ biểu đồ trực quan:

In [294…

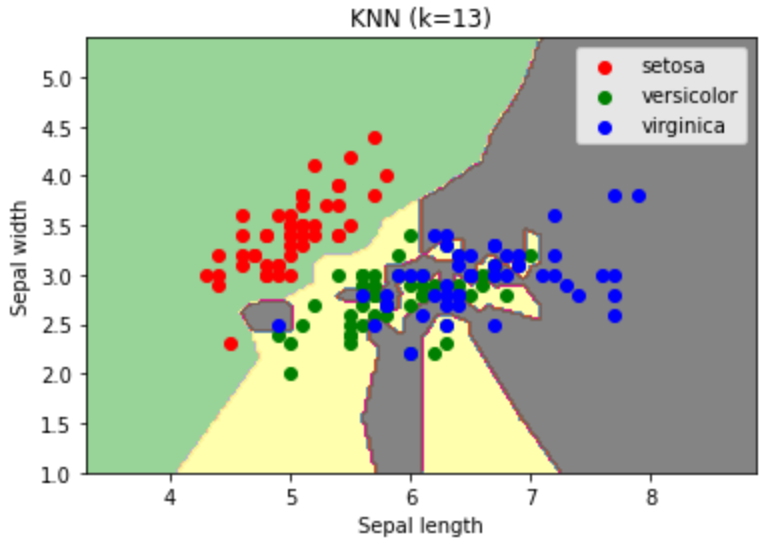
```
#---calculate misclassification error for each k---
MSE = [1 - x for x in cv_scores]
#---determining best k (min. MSE)---
optimal_k = ks[MSE.index(min(MSE))]
print(f"The optimal number of neighbors is {optimal_k}")
#---plot misclassification error vs k---
plt.plot(ks, MSE)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Accent, alpha=0.8)
#---plot the training points---
colors = ['red', 'green', 'blue']
for color, i, target in zip(colors, [0, 1, 2], iris.target_names):
    plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
```

```
plt.title(f'KNN (k={optimal_k})')
plt.legend(loc='best', shadow=False, scatterpoints=1)
```

```
The optimal number of neighbors is 13
```



Out[294]:
```
<matplotlib.legend.Legend at 0x7f178528bf50>
```



CHƯƠNG 3: HỌC KHÔNG GIÁM SÁT

Clustering Using K-Means

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

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

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

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

Triển khai:

- Import libraries và load data

In [243…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from copy import deepcopy

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

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-243-a30be125768a> in <module>
      4 from copy import deepcopy
      5
----> 6 dataFrame = pd.read_csv("kmeans.csv")
      7 plt.scatter(
      8     dataFrame['x'],

/usr/local/lib/python3.7/dist-packages/pandas/util/_decorators.py in wrapper(*args, **kw
args)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

/usr/local/lib/python3.7/dist-packages/pandas/io/parsers/readers.py in read_csv(filepath
_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_d
upe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skipro
ws, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines,
 parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,
 iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quotin
g, doublequote, escapechar, comment, encoding, encoding_errors, dialect, error_bad_line
s, warn_bad_lines, on_bad_lines, delim_whitespace, low_memory, memory_map, float_precisi
on, storage_options)
    584     kwds.update(kwds_defaults)
    585
--> 586     return _read(filepath_or_buffer, kwds)
    587
    588

/usr/local/lib/python3.7/dist-packages/pandas/io/parsers/readers.py in _read(filepath_or
_buffer, kwds)
    480
    481     # Create the parser.
--> 482     parser = TextFileReader(filepath_or_buffer, **kwds)
    483
    484     if chunksize or iterator:

/usr/local/lib/python3.7/dist-packages/pandas/io/parsers/readers.py in __init__(self, f,
 engine, **kwds)
    809             self.options["has_index_names"] = kwds["has_index_names"]
    810
--> 811         self._engine = self._make_engine(self.engine)
    812
    813     def close(self):

/usr/local/lib/python3.7/dist-packages/pandas/io/parsers/readers.py in _make_engine(sel
f, engine)
   1038             )
   1039         # error: Too many arguments for "ParserBase"
```

```
-> 1040         return mapping[engine](self.f, **self.options)  # type: ignore[call-arg]
   1041
   1042     def _failover_to_python(self):

/usr/local/lib/python3.7/dist-packages/pandas/io/parsers/c_parser_wrapper.py in __init__
(self, src, **kwds)
     49
     50         # open handles
---> 51         self._open_handles(src, kwds)
     52         assert self.handles is not None
     53

/usr/local/lib/python3.7/dist-packages/pandas/io/parsers/base_parser.py in _open_handles
(self, src, kwds)
    227             memory_map=kwds.get("memory_map", False),
    228             storage_options=kwds.get("storage_options", None),
--> 229             errors=kwds.get("encoding_errors", "strict"),
    230         )
    231

/usr/local/lib/python3.7/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mo
de, encoding, compression, memory_map, is_text, errors, storage_options)
    705                 encoding=ioargs.encoding,
    706                 errors=errors,
--> 707                 newline="",
    708             )
    709         else:

FileNotFoundError: [Errno 2] No such file or directory: 'kmeans.csv'
```

- Create centroids - Tạo các điểm trọng tâm

Giả sử k = 3, ta sẽ cần tạo ra 3 điểm trọng tâm của 3 nhóm ở vị trí bất kỳ:

In [ ]:
```
k = 3
```

- Tạo một ma trận chứa tất cả các điểm:

In [ ]:
```
X = np.array(list(
      zip(
          dataFrame['x'],
          dataFrame['y']
      )
    ))
```

- Sinh ra k trọng tâm ở các vị trí bất kỳ:

In [ ]:
```
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)
```

- Biểu diễn k trọng tâm dưới dạng ma trận:

In [ ]:
```
C = np.array(
        list(zip(Cx, Cy)),
        dtype=np.float64
    )
C
```

- Vẽ đồ thị tất cả các điểm:

In [ ]:
```
plt.scatter(dataFrame['x'], dataFrame['y'], c='r', s=8)
plt.scatter(Cx, Cy, marker='*', c='g', s=160)
plt.xlabel("x")
plt.ylabel("y")
```

- Định nghĩa một function tính khoảng cách Euclid giữa điểm `point` và các điểm `others`:

In [ ]:
```
def euclidean_distance(point, others, ax=1):
    return np.linalg.norm(point - others, axis=ax)
```

- Tạo ma trận lưu C cũ để có thể tính độ lệch sau mỗi bước lần lặp thuật toán:

In [ ]:
```
C_prev = np.zeros(C.shape)
```

- Tạo list lưu nhóm mà các điểm thuộc về:

In [ ]:
```
clusters = np.zeros(len(X))
```

- Đo khoảng cách của các trọng tâm giữa vị trí hiện tại và vị trí cũ:

In [ ]:
```
distance_differences = euclidean_distance(C, C_prev)
```

- Lặp cho tới khi các điểm trọng tâm không còn thay đổi vị trí nữa:

In [ ]:
```
while distance_differences.any() != 0:
    # Gán nhóm cho từng điểm:
    for i in range(len(X)):
        distances = euclidean_distance(X[i], C)
        # Trả về chỉ số của giá trị khoảng cách nhỏ nhất trong dãy giá trị tính được:
        cluster = np.argmin(distances)
        clusters[i] = cluster
    # Sau khi kết thúc, lưu trữ vị trí của các điểm trong C:
    C_prev = deepcopy(C)
    # Tìm các vị trí trọng tâm mới:
    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        if len(points) != 0:
            C[i] = np.mean(points, axis=0)
    distance_differences = euclidean_distance(C, C_prev)
```

- Vẽ biểu đồ phân tán:

In [244…
```
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: # Nhóm này không chứa điểm nào
        print("Plesae 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')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-244-51249bba073c> in <module>
      1 colors = ['b', 'r', 'y', 'g', 'c', 'm']
----> 2 for i in range(k):
      3     points = np.array([X[j] for j in range(len(X)) if clusters[j] == i])
      4     if len(points) > 0:
      5         plt.scatter(points[:, 0], points[:, 1], s=10, c=colors[i])

NameError: name 'k' is not defined
```

Sử dụng K-means trong thư viện `sklearn`:

In [ ]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=k)
kmeans = kmeans.fit(X)
labels = kmeans.predict(X)
centroids = kmeans.cluster_centers_
c = ['b','r','y','g','c','m']
colors = [c[i] for i in labels]
plt.scatter(dataFrame['x'], dataFrame['y'], c=colors, s=18)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=100, c='black')
```

CHƯƠNG 4: CASE STUDY & DEPLOY

Ở chương này, ta sẽ thực hiện deploy một mô hình Machine Learning thông qua REST API.

Với case study chẩn đoán khả năng mắc bệnh tiểu đường, ta sẽ sử dụng tập dữ liệu tại: https://www.kaggle.com/uciml/pima-indians-diabetes-database.

Các feature có trong tập dữ liệu:

- **Pregnancies**: Số lần mang thai.
- **Glucose**: Nồng độ glucose trong huyết tương sau 2 giờ trong xét nghiệm dung nạp glucose qua đường uống.
- **BloodPressure**: Huyết áp tâm trương (mm Hg).
- **SkinThickness**: Độ dày da (mm).
- **Insulin**: Huyết thanh trong 2 giờ (mu U/ml)
- **BMI**: Chỉ số khối cơ thể (khối lượng là kg/(chiều cao là m)^2).
- **DiabetesPedigreeFunction**: Chức năng phả hệ bệnh tiểu đường.
- **Age**: Tuổi (theo năm).
- **Outcome**: 0 (âm tính) hoặc 1 (dương tính).

In [246…
```
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
```

4.1. Cleaning Data

Kiểm tra và loại bỏ các giá trị `null`:

In [247…
```
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
```

Kiểm tra và loại bỏ các giá trị `0` sai ý nghĩa (các giá trị `0` ở cột bất kỳ ngoại trừ `Pregnancies` và `Outcome`):

In [248…
```
print("0s")
print("==")
print(df.eq(0).sum())
df[['Glucose','BloodPressure','SkinThickness', 'Insulin','BMI','DiabetesPedigreeFunction
df[['Glucose','BloodPressure','SkinThickness', 'Insulin','BMI','DiabetesPedigreeFunction
```

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

Thay thế chúng bằng giá trị trung bình của cột:

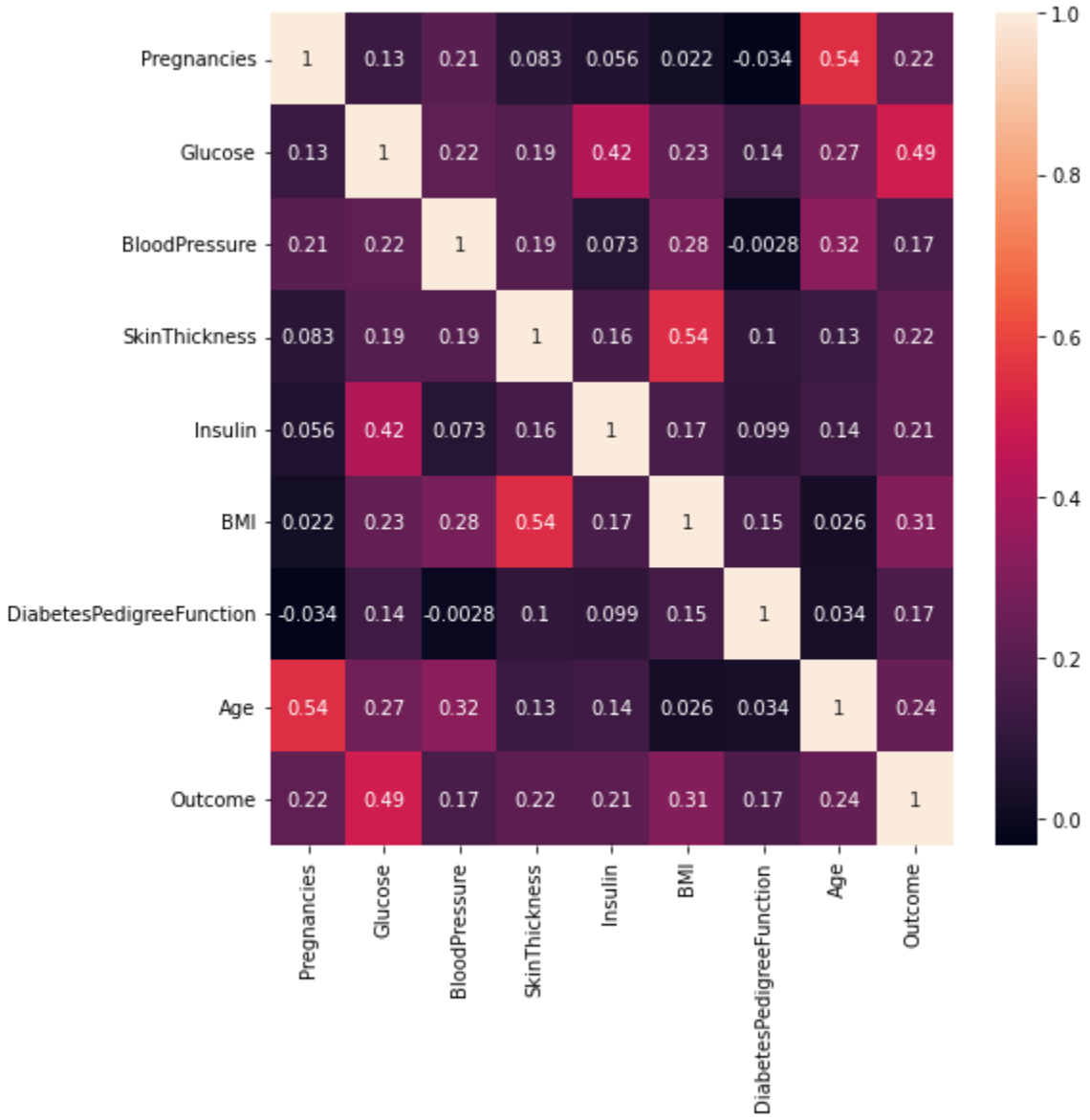
In [249…
```
df.fillna(df.mean(), inplace = True)
```

4.2. Kiểm tra tính tương quan

Ta cần kiểm tra tính tương quan để xem các yếu tố ảnh hưởng tới kết quả như thế nào:

In [250…
```
import seaborn as sns
sns.heatmap(df.corr(),annot=True)
```

```
fig = plt.gcf()
fig.set_size_inches(8,8)
```



Để loại bỏ những yếu tố không/ít ảnh hưởng, ta sẽ chọn ra 3 yếu tố ảnh hưởng nhiều nhất để huấn luyện mô hình:

In [253…
```
print(df.corr().nlargest(4, 'Outcome').index)
print(df.corr().nlargest(4, 'Outcome').values[:,8])
```

```
Index(['Outcome', 'Glucose', 'BMI', 'Age'], dtype='object')
[1.         0.49292767 0.31192439 0.23835598]
```

=> Đó là `Glucose`, `BMI`, và `Age`.

4.3. Đánh giá các thuật toán

Tạo một `list` lưu các kết quả đánh giá:

In [261…
```
result = []
```

4.3.1. Logistic Regression

Thay vì chia nhỏ tập dữ liệu thành các tập huấn luyện và thử nghiệm, ta sẽ sử dụng kiểm chứng chéo 10 lần để thu được điểm trung bình của thuật toán được sử dụng:

In [262…
```
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()
log_regress_score
```

Out[262]:
```
0.7669856459330144
```

Thêm vào kết quả đánh giá có được:

In [263…
```
result.append(log_regress_score)
```

4.3.2. K-Nearest Neighbors

Ngoài việc sử dụng kiểm chứng chéo 10 lần để có được điểm trung bình của thuật toán, chúng ta cũng cần thử các giá trị khác nhau của k để có được giá trị k tối ưu có độ chính xác tốt nhất.

In [264…
```
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
```

4.3.3. Support Vector Machines

4.3.3.1. Linear Kernel

In [265…
```
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
```

4.3.3.2. RBF (Radial Basic Function) Kernel

In [266…
```
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
```

4.4. Chọn lựa thuật toán sử dụng

In [267…
```
algorithms = ["Logistic Regression", "K Nearest Neighbors", "SVM Linear Kernel", "SVM RB
cv_mean = pd.DataFrame(result, index = algorithms)
cv_mean.columns=["Accuracy"]
cv_mean.sort_values(by="Accuracy",ascending=False)
```

Out[267]:

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

=> Ta chọn thuật toán "K Nearest Neighbors".

4.5. Huấn luyện và lưu Model

Vì thuật toán đã hoạt động và tìm ra k tối ưu là 19, ta có thể huấn luyện nó như sau:

In [269…
```
knn = KNeighborsClassifier(n_neighbors=19)
knn.fit(X, y)
```

Out[269]:
```
KNeighborsClassifier(n_neighbors=19)
```

Lưu model:

In [270…
```
import pickle
filename = 'diabetes.sav'
pickle.dump(knn, open(filename, 'wb'))
```

4.6. Deploying

4.6.1. Server

Ở phía server, ta sẽ deploy machine learning model qua REST API, sử dụng framework `flask` :

In [271…
```
import pickle
from flask import Flask, request, json, jsonify

app = Flask(__name__)
```

Load model:

In [272…
```
#---the filename of the saved model---
filename = 'diabetes.sav'
```

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

Dựng hàm chuẩn đoán kết quả và phản hồi lại client khi client sử dụng method POST:

In [273…
```
@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)
```

Chạy ứng dụng:

In [277…
```
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
INFO:werkzeug: * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

4.6.2. Client

Ở phía client, sau khi chạy server, ta sẽ thử kiểm tra:

In [275…
```
import json
import requests
```

Hàm kết nối tới server, POST dữ liệu thông qua API và nhận phản hồi:

In [276…
```
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
```

Thử chẩn đoán với các chỉ số `BMI` , `Age` , `Glucose` lần lượt là 30, 40 và 100:

In [ ]:
```
if __name__ == "__main__":
    predictions = predict_diabetes(30,40,100)
    print("Diabetic" if predictions["prediction"] == 1 else "Not Diabetic")
    print("Confidence: " + predictions["confidence"] + "%")
```

Kết quả sẽ là:

```
Not Diabetic
Confidence: 52.63%
```