Handout Statistics Software Training: Machine Learning Basics with Python

Novri Suhermi

1 Bahasa Pemrograman Python

1.1 Syntax Dasar Python

1.1.1 Hello Python!

In:

```
print ('Hello Python!')
```

Out:

```
Hello Python!
```

1.1.2 Operasi Aritmetika

In:

```
# Penjumlahan dan Pengurangan
print (4+10)
print (5-2)
```

Out:

```
14
3
```

In:

```
# Perkalian dan Pembagian
print (5*8)
print (9.0/2)
```

Out:

```
40
4.5
```

In:

```
# Operasi Pangkat
print (4**2)
```

Out:

```
16
```

In:

```
# Modulo
print (18%7)
```

Out:

```
4
```

1.2 Variable dan Type

1.2.1 Variable Assignment

In:

```
# Membuat variabel
savings = 100

# Mencetak variabel "saving"
print (savings)
```

Out:

```
100
```

1.2.2 Perhitungan dengan Variabel

In:

```
# Membuat variabel "factor"
factor = 1.10

# Menghitung "result"
result = savings * (factor**7)

# Mencetak "result"
print ('result = '+ str(result))
```

Out:

```
result = 194.87171000000012
```

1.2.3 Tipe variabel lain

In:

```
# Membuat variabel "desc"
desc = 'compound interest'

# Membuat variabel "profitable"
profitable = True
```

1.2.4 Operasi tipe variabel lain

In:

```
savings = 100
factor = 1.1
desc = "compound interest"

# Assign perkalian dari factor dan savings ke variabel year1
year1 = factor * savings

# Mencetak tipe variabel "year1"
print (type(year1))

# Assign penjumlahan "desc" dan "desc" ke variabel "doubledesc"
doubledesc = desc + desc

# Mencetak variabel "doubledesc"
print (doubledesc)
```

Out:

```
<class 'float'>
compound interestcompound interest
```

1.2.5 Konversi tipe variabel

In:

```
# Mendefinisikan variabel savings dan result
savings = 100
result = 100 * 1.10 ** 7

# Print out
print ("I started with $" + str(savings) + " and now have $" + str(result) + ". Awesome!")

# Mendefinisikan pi_string
pi_string = '3.1415926'

# Menkonversi pi_string ke dalam float
pi_float = float(pi_string)
```

Out:

```
I started with $100 and now have $194.87171000000012. Awesome!
```

1.3 List

1.3.1 Membuat sebuah list

In:

```
# variabel area (dalam m2)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Membuat list "areas"
areas = [hall, kit, liv, bed, bath]

# Mencetak "areas"
print(areas)
```

Out:

```
[11.25, 18.0, 20.0, 10.75, 9.5]
```

1.3.2 Membuat list dengan berbeda tipe variabel

In:

```
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# List "areas" baru
areas = ["hallway", hall, "kitchen", kit, "living room", liv, "bedroom", bed, "bathroom", bath]

# Mencetak areas
print(areas)
```

Out:

```
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0, 'bedroom', 10.75, 'bathroom', 9.5]
```

1.3.3 List of lists

In:

```
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# list of lists "house"
house = [["hallway", hall],
["kitchen", kit],
["living room", liv],
["bedroom", bed],
["bathroom", bath]]

# Mencetak "house"
print(house)

# Mencetak tipe "house"
print(type(house))
```

Out:

```
[['hallway', 11.25], ['kitchen', 18.0], ['living room', 20.0], ['bedroom', 10.75], ['bathroom',
  9.5]]
<class 'list'>
```

1.4 Subsetting Lists

1.4.1 Subset

In:

```
# list "areas"
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
         "bathroom", 9.50]

# Mencetak elemen kedua dari list areas
print(areas[1])

# Mencetak elemen terakhir dari list areas
print(areas[-1])

# Mencetak area dari "living room"
print(areas[5])
```

Out:

```
11.25
9.5
20.0
```

1.4.2 Subset dan operasi subset

In:

```
# list "areas"
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
         "bathroom", 9.50]

# Membuat variabel "eat_sleep_area"
eat_sleep_area = areas[3] + areas[7]

# Mencetak variabel "eat_sleep_area"
print(eat_sleep_area)
```

Out:

```
28.75
```

1.4.3 Slicing di dalam list

In:

```
# list areas
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
         "bathroom", 9.50]

# Menggunakan slicing untuk membuat list downstairs
downstairs = areas[0:6]

# Menggunakan slicing untuk membuat list upstairs
upstairs = areas[6:10]

# Mencetak downstairs dan upstairs
print (downstairs)
print (upstairs)
```

Out:

```
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0]
['bedroom', 10.75, 'bathroom', 9.5]
```

1.4.4 Slicing di dalam list (2)

In:

```
# list areas
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
"bathroom", 9.50]

# Alternative slicing
downstairs = areas[:6]
print (downstairs)

# Alternative slicing
upstairs = areas[6:]
print (upstairs)
```

Out:

```
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0]
['bedroom', 10.75, 'bathroom', 9.5]
```

1.5 Manipulasi list

1.5.1 Mengganti elemen di dalam list

In:

```
# list "areas"
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
         "bathroom", 9.50]

# Mengganti area bathroom
areas[-1] = 10.50

# Mengganti "living room" menjadi "chill zone"
areas[4] = "chill zone"

print (areas)
```

Out:

```
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bedroom', 10.75, 'bathroom', 10.5]
```

1.5.2 Ekstensi list

In:

```
# list area
areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,
"bedroom", 10.75, "bathroom", 10.50]

# ekstensi list baru: areas_1
areas_1 = areas + ["poolhouse", 24.5]

# ekstensi llist baru: areas_2
areas_2 = areas_1 + ["garage", 15.45]

print (areas)
print (areas_1)
print (areas_2)
```

Out:

```
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bedroom', 10.75, 'bathroom', 10.5]
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bedroom', 10.75, 'bathroom', 10.5, '
  poolhouse', 24.5]
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bedroom', 10.75, 'bathroom', 10.5, '
  poolhouse', 24.5, 'garage', 15.45]
```

1.5.3 Manipulasi lain

In:

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas_copy
areas_copy = areas[:]

# Change areas_copy
areas_copy[0] = 5.0

# Print areas
print (areas)

# Print areas_copy
print (areas_copy)
```

Out:

```
[11.25, 18.0, 20.0, 10.75, 9.5]
[5.0, 18.0, 20.0, 10.75, 9.5]
```

1.6 Dictionaries

1.6.1 Membuat dictionary

In:

```
# list countries dan capitals
countries = ['spain', 'france', 'germany', 'norway']
capitals = ['madrid', 'paris', 'berlin', 'oslo']

# membuat dictionary europe
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo'}

# Print europe
print(europe)
```

Out:

```
{'spain': 'madrid', 'france': 'paris', 'germany': 'berlin', 'norway': 'oslo'}
```

1.6.2 Akses dictionary

In:

```
# dictionary europe
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# mencetak keys dictionary europe
print(europe.keys())

# mencetak value dari keys 'norway'
print(europe['norway'])
```

Out:

```
dict_keys(['spain', 'france', 'germany', 'norway'])
oslo
```

1.6.3 Manipulasi dictionary

In:

```
# dictionary europe
europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# menambahkan italy ke dictionary europe
europe['italy'] = 'rome'

print('italy' in europe)

# mencetak value dari key 'italy'
print(europe['italy'])

# menambahkan poland ke dictionary europe
europe['poland'] = 'warsaw'

print(europe)
```

Out:

```
True
rome
{'spain': 'madrid', 'france': 'paris', 'germany': 'berlin', 'norway': 'oslo', 'italy': 'rome', '
  poland': 'warsaw'}
```

In:

```
# dictionary europe
europe = {'spain':'madrid', 'france':'paris', 'germany':'bonn',
'norway':'oslo', 'italy':'rome', 'poland':'warsaw',
'australia':'vienna' }

# Update capital of germany
europe.update({'germany':'berlin'})

# Remove australia
del(europe['australia'])

# Print europe
print(europe)
```

Out:

```
{'spain': 'madrid', 'france': 'paris', 'germany': 'berlin', 'norway': 'oslo', 'italy': 'rome', '
  poland': 'warsaw'}
```

1.6.4 Dictionaryception

In:

```
# Dictionary of dictionaries
europe = { 'spain': { 'capital':'madrid', 'population':46.77 },
'france': { 'capital':'paris', 'population':66.03 },
'germany': { 'capital':'berlin', 'population':80.62 },
'norway': { 'capital':'oslo', 'population':5.084 } }


# mencetak ibu kota France
print(europe['france']['capital'])

# membuat sub-dictionary data
data = {'capital':'rome', 'population':59.83}

# menambahkan data ke dictionary europe di bawah key 'italy'
europe.update({'italy':data})

# Print europe
print(europe)
```

Out:

```
paris
{'spain': {'capital': 'madrid', 'population': 46.77}, 'france': {'capital': 'paris', 'population
  ': 66.03}, 'germany': {'capital': 'berlin', 'population': 80.62}, 'norway': {'capital': 'oslo',
   'population': 5.084}, 'italy': {'capital': 'rome', 'population': 59.83}}
```

1.7 Fungsi (Function)

1.7.1 Beberapa contoh function

In:

```
# Membuat variabel "var1" dan "var2"
var1 = [1, 2, 3, 4]
var2 = True

# Mencetak tipe "var1" dengan fungsi "type"
print (type(var1))

# Mencetak panjang dari list "var1" dengan fungsi "len"
print (len(var1))

# Konversi "var2" menjadi tipe "int" menggunakan fungsi "int"
out2 = int(var2)

# Mencetak tipe dari "var2"
print (type(var2))
```

Out:

```
<class 'list'>
4
<class 'bool'>
```

1.7.2 Multiple arguments

In:

```
# Membuat list first dan second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]

# Menggabungkan list first dan second di dalam list full
full = first + second

# Mengurutkan list full dari tinggi ke rendah: full_sorted
full_sorted = sorted(full, reverse=True)

# Mencetak full_sorted
print (full_sorted)
```

Out:

```
[20.0, 18.0, 11.25, 10.75, 9.5]
```

1.7.3 Membuat Fungsi Sederhana

In:

```
# Define shout with the parameter, word
def shout(word):
    """Return a string with three exclamation marks"""
    # Concatenate the strings: shout_word
    shout_word = word + '!!!'
    # Replace print with return
    return(shout_word)
# Pass 'congratulations' to shout: yell
yell = shout('congratulations')
# Print yell
print(yell)
```

Out:

```
congratulations!!!
```

1.7.4 Fungsi dengan Multiple parameter

In:

```
# Define shout with parameters word1 and word2
def shout(word1, word2):
    """Concatenate strings with three exclamation marks"""
    # Concatenate word1 with '!!!': shout1
    shout1 = word1 + '!!!'

    # Concatenate word2 with '!!!': shout2
    shout2 = word2 + '!!!'

    # Concatenate shout1 with shout2: new_shout
    new_shout = shout1 + shout2

    # Return new_shout
    return new_shout

# Pass 'congratulations' and 'you' to shout(): yell
yell = shout('congratulations', 'you')

    # Print yell
print(yell)
```

Out:

```
congratulations!!!you!!!
```

1.8 Methods

1.8.1 Beberapa method dalam string

In:

```
# Membuat variabel string room
room = "poolhouse"

# menggunakan method "upper()"
room_up = room.upper()

# Mencetak room dan room_up
print (room)
print (room_up)

# Menggunakan  method "count" untuk menghitung banyaknya karakter "o" di dalam variabel room
print (room.count('o'))
```

Out:

```
poolhouse
POOLHOUSE
3
```

1.8.2 Methods di dalam list

In:

```
# list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Mencetak indeks dari element 20.0 dengan method "index()"
print (areas.index(20.0))

# Mencetak berapa kali element 14.5 muncul dengan method "count()"
print (areas.count(14.5))
```

Out:

```
2
0
```

1.8.3 Methods di dalam list (2)

In:

```
# list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Menggunakan method "append()"
areas.append(24.5)
areas.append(15.45)


# Mencetak areas
print (areas)

# Menggunakan method "reverse()"
areas.reverse()

# Mencetak areas
print (areas)
```

Out:

```
[11.25, 18.0, 20.0, 10.75, 9.5, 24.5, 15.45]
[15.45, 24.5, 9.5, 10.75, 20.0, 18.0, 11.25]
```

1.9 Packages

1.9.1 Import package

In:

```
# mendefinisikan variabel radius r
r = 0.43

# Import package "math"
import math

# Menghitung C
C = 2 * math.pi * r

# Menghitung A
A = math.pi * r**2

# Mencetak hasil
print ("Circumference: " + str(C))
print ("Area: " + str(A))
```

Out:

```
Circumference: 2.701769682087222
Area: 0.5808804816487527
```

1.9.2 Cara lain import package

In:

```
# variabel radius r
r = 192500

# Import fungsi "radians" dalam package math
from math import radians

# menghitung variabel "dist"
dist = r * radians(12)

# mencetak "dist"
print (dist)
```

Out:

```
40317.10572106901
```

1.10 Numpy

1.10.1 Numpy array

In:

```
# list baseball
baseball = [180, 215, 210, 210, 188, 176, 209, 200]

# import package numpy sebagai np
import numpy as np

# membuat numpy array dari baseball: np_baseball
np_baseball = np.array(baseball)

# mencetak tipe np_baseball
print (type(np_baseball))
```

Out:

```
<class 'numpy.ndarray'>
```

1.10.2 Operasi numpy

In:

```
# list height (inch)
height = [74, 74, 72, 72, 73, 69, 69, 71, 76, 71, 73, 73, 74, 74, 69]

# numpy array np_height
np_height = np.array(height)

# mencetak np_height
print(np_height)

# Konversi np_height (inch) ke np_height_m (meter)
np_height_m = np_height*0.0254

# Mencetak np_height_m
print (np_height_m)
```

Out:

```
[74 74 72 72 73 69 69 71 76 71 73 73 74 74 69]
[1.8796 1.8796 1.8288 1.8288 1.8542 1.7526 1.7526 1.8034 1.9304 1.8034
1.8542 1.8542 1.8796 1.8796 1.7526]
```

1.10.3 Operasi numpy (2)

In:

```
# list weight
weight = [180, 215, 210, 210, 188, 176, 209, 200, 231, 180, 188, 180, 185, 160, 180]

# numpy array np_height_m
np_height_m = np.array(height) * 0.0254

# numpy array np_weight_kg
np_weight_kg = np.array(weight) * 0.453592

# menghitung bmi (body mass index)
bmi = np_weight_kg / np_height_m **2

# mencetak bmi
print (bmi)
```

Out:

```
[23.11037639 27.60406069 28.48080465 28.48080465 24.80333518 25.99036864
30.86356276 27.89402921 28.11789135 25.10462629 24.80333518 23.7478741
23.75233129 20.54255679 26.58105883]
```

1.10.4 Operasi numpy (3)

In:

```
# menghitung bmi
np_height_m = np.array(height) * 0.0254
np_weight_kg = np.array(weight) * 0.453592
bmi = np_weight_kg / np_height_m ** 2

# mencari bmi < 21
light = bmi < 21

# mencetak light
print (light)
```

Out:

```
[False False False False False False False False False False False False
False  True False]
```

1.10.5 Subsetting Numpy Arrays

In:

```
# np_weight dan np_height
np_weight = np.array(weight)
np_height = np.array(height)

# mencetak np_weight pada index 5
print (np_weight[5])

# mencetak np_height dari index 5 sampai 10
print (np_height[8:11])
```

Out:

```
176
[76 71 73]
```

1.11 2D Numpy Arrays

1.11.1 Membuat numpy array 2 dimensi

In:

```
# list of list baseball
baseball = [[180, 78.4],
[215, 102.7],
[210, 98.5],
[188, 75.2]]

# Import numpy
import numpy as np

# 2D Numpy array np_baseball
np_baseball = np.array(baseball)

# mencetak tipe np_baseball
print (type(np_baseball))

# mencetak shape np_baseball
print (np_baseball.shape)
```

Out:

```
<class 'numpy.ndarray'>
(4, 2)
```

1.11.2 Contoh lain 2D numpy array

In:

```
# list of list baseball
baseball= [[73, 188], [73, 180], [74, 185], [74, 160], [69, 180], [70, 185], [73, 189], [75,
  185], [78, 219], [79, 230]]

# Import numpy
import numpy as np

# 2d numpy array np_baseball
np_baseball = np.array(baseball)

# mencetak shape np_baseball
print (np_baseball.shape)
```

Out:

```
(10, 2)
```

1.11.3 Subsetting 2D Numpy Arrays

In:

```
# mencetak baris ke 5 np_baseball
print (np_baseball[4,:])

# akses seluruh baris pada kolom ke 2 np_baseball: np_weight
np_weight = np_baseball[:,1]

# mencetak height pada index ke 8
print (np_baseball[8,0])
```

Out:

```
[ 69 180]
78
```

1.11.4 Aritmetika numpy

In:

```
# list of list baseball
baseball = [[74.0, 180.0, 22.99],
[74.0, 215.0, 34.69],
[72.0, 210.0, 30.78],
[72.0, 210.0, 35.43],
[73.0, 188.0, 35.71],
[69.0, 176.0, 29.39],
[69.0, 209.0, 30.77],
[71.0, 200.0, 35.07],
[76.0, 231.0, 30.19],
[73.0, 188.0, 23.88]]

# 2d numpy array update
update = np.array([[1.2303559, -11.16224898, 1.],
[1.02614252, 16.09732309, 1.],
[0.99484223, 8.14402711, 1.],
[0.99484223, 8.14402711, 1.],
[1.15442283, 5.08167641, 1.],
[1.09349925, 4.23890778, 1.],
[1.09349925, 4.23890778, 1.],
[1.09349925, 4.23890778, 1.],
[0.82285669, -17.78200035, 1.],
[0.99484223, 8.14402711, 1.]])
```

In:

```
# numpy array np_baseball
np_baseball = np.array(baseball)

# mencetak penjumalahan np_baseball + update
print (np_baseball + update)

# membuat numpy array conversion
conversion = np.array([0.0254, 0.453592, 1])

# mencetak perkalian np_baseball dan conversion
print (np_baseball * conversion)
```

Out:

```
[[ 75.2303559  168.83775102  23.99      ]
[ 75.02614252 231.09732309  35.69      ]
[ 72.99484223 218.14402711  31.78      ]
[ 72.99484223 218.14402711  36.43      ]
[ 74.15442283 193.08167641  36.71      ]
[ 70.09349925 180.23890778  30.39      ]
[ 70.09349925 213.23890778  31.77      ]
[ 72.09349925 204.23890778  36.07      ]
[ 76.82285669 213.21799965  31.19      ]
[ 73.99484223 196.14402711  24.88      ]]
[[  1.8796    81.64656   22.99    ]
[  1.8796    97.52228   34.69    ]
[  1.8288    95.25432   30.78    ]
[  1.8288    95.25432   35.43    ]
[  1.8542    85.275296  35.71    ]
[  1.7526    79.832192  29.39    ]
[  1.7526    94.800728  30.77    ]
[  1.8034    90.7184    35.07    ]
[  1.9304   104.779752  30.19    ]
[  1.8542    85.275296  23.88    ]]
```

1.12 Statistik dasar dengan numpy

1.12.1 mean dan median

In:

```
# np_height
np_height = np_baseball[:,0]

# mean np_height
print (np.mean(np_height))

# median np_height
print (np.median(np_height))
```

Out:

```
72.3
72.5
```

1.13 Statistik deskriptif numpy

In:

```
# mean height
avg = np.mean(np_baseball[:,0])
print ("Average: " + str(avg))
# median height
med = np.median(np_baseball[:,0])
print ("Median: " + str(med))
# standard deviasi
stddev = np.std(np_baseball[:,0])
print ("Standard Deviation: " + str(stddev))
# corelasi kolom 0 dan kolom 1
corr = np.corrcoef(np_baseball[:,0],np_baseball[:,1])
print ("Correlation: " + str(corr))
```

Out:

```
Average: 72.3
Median: 72.5
Standard Deviation: 2.0999999999999996
Correlation: [[1.         0.38329645]
[0.38329645 1.        ]]
```

1.13.1 Statistik deskriptif numpy (2)

In:

```
# heights & positions
heights = [191, 184, 185, 180, 181, 187, 170, 179, 183, 186, 185, 170, 187, 183, 173, 188, 183,
  180, 188, 175, 193]
positions = ['GK','M', 'A', 'D', 'M', 'D', 'M', 'M', 'M', 'A', 'M', 'M', 'A', 'A', 'A', 'M', 'D',
   'A', 'D', 'M', 'GK']

# numpy arrays np_positions & np_heights
np_positions = np.array(positions)
np_heights = np.array(heights)

# gk_heights
gk_heights = np_heights[np_positions=='GK']

# other_heights
other_heights = np_heights[np_positions!='GK']

# median height goalkeepers
print ("Median height of goalkeepers: " + str(np.median(gk_heights)))

# median height otherplayers
print ("Median height of other players: " + str(np.median(other_heights)))
```

Out:

```
Median height of goalkeepers: 192.0
Median height of other players: 183.0
```

1.14 Visualisasi dengan matplotlib

1.14.1 Line plot

In:

```
# List year
year = [1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964]

# List pop
pop = [2.57, 2.62, 2.67, 2.71, 2.76, 2.81, 2.86, 2.92, 2.97, 3.03, 3.08, 3.14, 3.2, 3.26]

# Import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

# membuat line plot: year pada x-axis, pop pada y-axis
plt.plot(year,pop)
plt.show()
```

Out:

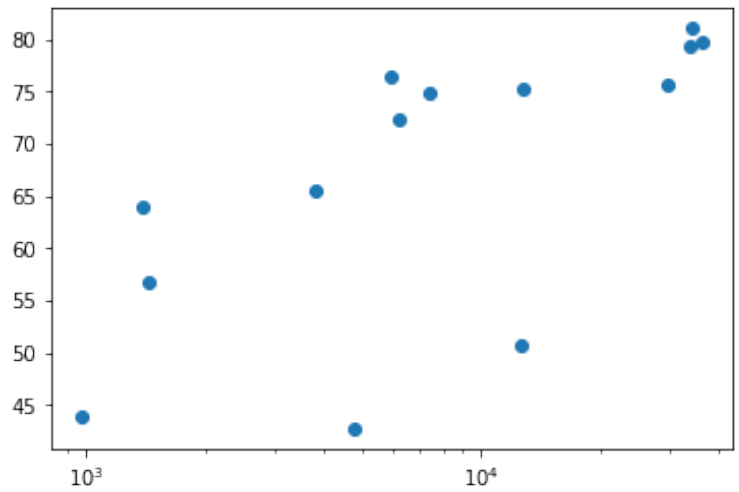
```
<matplotlib.figure.Figure at 0x248f049fef0>
```

1.14.2 Scatter Plot (1)

In:

```
# list gdp_cap
gdp_cap = [974.580, 5937.029, 6223.367, 4797.231, 12779.379, 34435.367, 36126.492, 29796.048,
  1391.253,
33692.605, 1441.284, 3822.137, 7446.298, 12569.851]

# list life_exp
life_exp = [43.828, 76.423, 72.301, 42.731, 75.319, 81.234, 79.828, 75.635, 64.061, 79.441,
  56.728, 65.554, 74.852, 50.728]

# membuat scatter plot
plt.scatter(gdp_cap, life_exp)

# scale x axis ke dalam skala logaritma
plt.xscale('log')

# menampilkan plot
plt.show()
```

Out:

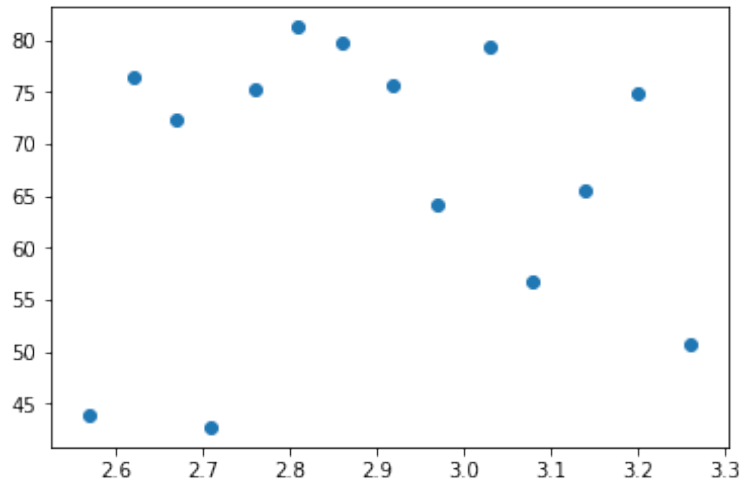


1.14.3 Scatter plot (2)

In:

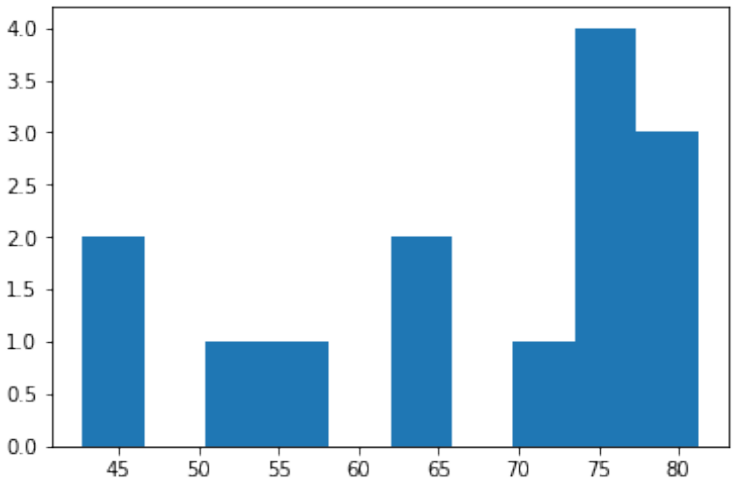
```
# contoh scatter plot lain
plt.scatter(pop, life_exp)

# menampilkan plot
plt.show()
```

Out:



1.14.4 histogram (1)

```
# membuat histogram dari data life_exp
plt.hist(life_exp)

# menampilkan histogram
plt.show()
```

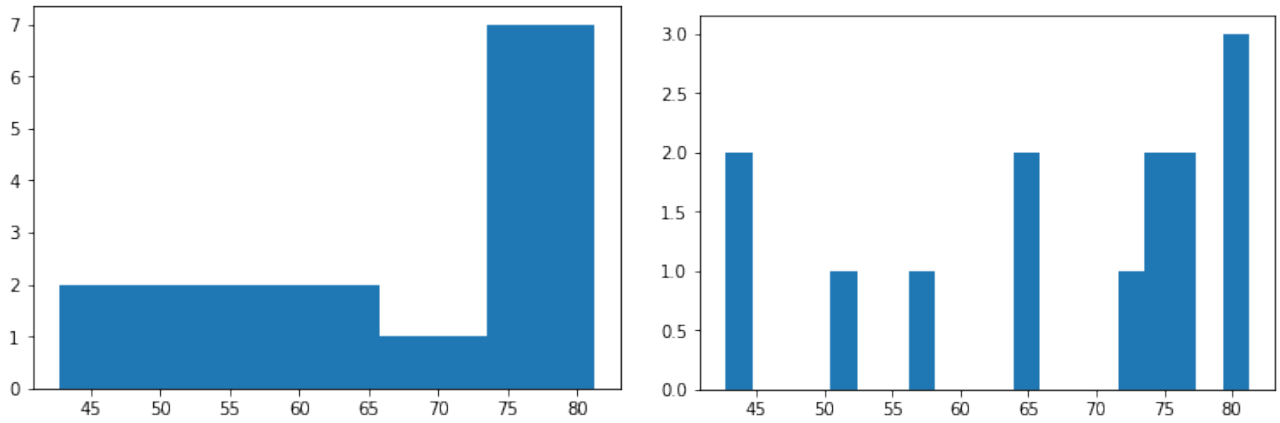


1.14.5 histogram (2)

In:

```
# membuat histogram dengan 5 bins
plt.hist(life_exp, bins=5)

# menampilkan dan membersihkan plot
plt.show()
plt.clf()

# membuat histogram dengan 20 bins
plt.hist(life_exp, bins=20)

# menampilkan histogram dan membersihkan (lagi)
plt.show()
plt.clf()
```

Out:



1.15 Logika Boolean, Conditional Statement, dan Looping

1.15.1 Equality

In:

```
# membandingkan 2 boolean
print (True == False)

# membandingkan integer
print (-5*15 != 75)

# membandingkan string
print ("pyscript" == "PyScript")

# membandingkan boolean dan integer
print (True == 1)
print (True == 0)
print (False == 0)
print (False == 1)
```

Out:

```
False
True
False
True
False
True
False
```

1.15.2 Greater dan less than

In:

```
# membandingkan 2 integer
x = -3 * 6
print (x >= -10)

# membandingkan 2 string
y = "test"
print ("test" <= y)

# membandingkan 2 boolean
print (True > False)
```

Out:

```
False
True
True
```

1.15.3 and, or, not

In:

```
# variabel my_kitchen dan your_kitchen
my_kitchen = 18.0
your_kitchen = 14.0

# my_kitchen>10 dan my_kitchen<18
print (my_kitchen>10 and my_kitchen<18)

# my_kitchen<14 atau my_kitchen>17
print (my_kitchen<14 or my_kitchen>17)

# 2 * my_kitchen < 3 * your_kitchen
print (my_kitchen*2<3*your_kitchen)
```

Out:

```
False
True
True
```

1.15.4 if

In:

```
# variabel room dan area
room = "kit"
area = 14.0

# statement if untuk room
if room == "kit":
print ("looking around in the kitchen.")

# statement if untuk area
if area > 15:
print ("big place!")
```

Out:

```
looking around in the kitchen.
```

1.15.5 if else

In:

```
# if-else untuk room
if room == "kit" :
print ("looking around in the kitchen.")
else :
print ("looking around elsewhere.")

# if-else untuk area
if area > 15 :
print ("big place!")
else:
print ("pretty small.")
```

Out:

```
looking around in the kitchen.
pretty small.
```

1.15.6 elif

In:

```
# if-elif-else untuk room
if room == "kit" :
print ("looking around in the kitchen.")
elif room == "bed":
print ("looking around in the bedroom.")
else :
print ("looking around elsewhere.")

# if-elif-else untuk area
if area > 15 :
print ("big place!")
elif area > 10:
print ("medium size, nice!")
else :
print ("pretty small.")
```

Out:

```
looking around in the kitchen.
medium size, nice!
```

1.15.7 while

In:

```
# variabel offset
offset = 4

# while loop
while offset !=0 :
print("correcting...")
offset = offset - 1
print(offset)
```

Out:

```
correcting...
3
correcting...
2
correcting...
1
correcting...
0
```

In:

```
# variabel offset
offset = -4

# while loop
while offset != 0 :
print("correcting...")
if offset > 0 :
offset = offset - 1
else :
offset = offset + 1
print(offset)
```

Out:

```
correcting...
-3
correcting...
-2
correcting...
-1
correcting...
0
```

1.15.8 for

In:

```
# list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# for loop
for area in areas:
print(area)
```

Out:

```
11.25
18.0
20.0
10.75
9.5
```

In:

```
# list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# for loop menggunakan enumerate()
for index,a in enumerate(areas) :
print("room "+str(index)+": "+str(a))
```

Out:

```
room 0: 11.25
room 1: 18.0
room 2: 20.0
room 3: 10.75
room 4: 9.5
```

In:

```
# lit areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# for loop
for index, area in enumerate(areas) :
print("room " + str(index+1) + ": " + str(area))
```

Out:

```
room 1: 11.25
room 2: 18.0
room 3: 20.0
room 4: 10.75
room 5: 9.5
```

In:

```
# list of list house
house = [["hallway", 11.25],
["kitchen", 18.0],
["living room", 20.0],
["bedroom", 10.75],
["bathroom", 9.50]]

# for loop
for x in house:
print("the "+str(x[0])+" is "+str(x[1])+" sqm")
```

Out:

```
the hallway is 11.25 sqm
the kitchen is 18.0 sqm
the living room is 20.0 sqm
the bedroom is 10.75 sqm
the bathroom is 9.5 sqm
```

In:

```
# dictionary europe
europe = {'spain':'madrid', 'france':'paris', 'germany':'bonn',
'norway':'oslo', 'italy':'rome', 'poland':'warsaw', 'australia':'vienna' }

# Iterasi europe
for country, city in europe.items():
print("the capital of "+ country + " is "+ city)
```

Out:

```
the capital of spain is madrid
the capital of france is paris
the capital of germany is bonn
the capital of norway is oslo
the capital of italy is rome
the capital of poland is warsaw
the capital of australia is vienna
```

1.16 Pandas

1.16.1 Dictionary to DataFrame

In:

```
# Pre-defined lists
names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
dr =  [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]

# Import pandas as pd
import pandas as pd

# membuat dictionary my_dict
my_dict = {'country':names, 'drives_right':dr, 'cars_per_cap':cpc}

# membuat DataFrame cars dari my_dict: cars
cars = pd.DataFrame(my_dict)

# Print cars
print(cars)
```

Out:

```
   cars_per_cap        country  drives_right
0           809  United States          True
1           731      Australia         False
2           588          Japan         False
3            18          India         False
4           200         Russia          True
5            70        Morocco          True
6            45          Egypt          True
```

In:

```
# row_labels
row_labels = ['US', 'AUS', 'JAP', 'IN', 'RU', 'MOR', 'EG']

# menjadikan row_labels sebagai index DataFrame cars
cars.index = row_labels

# Print cars
print(cars)
```

Out:

```
     cars_per_cap        country  drives_right
US            809  United States          True
AUS           731      Australia         False
JAP           588          Japan         False
IN             18          India         False
RU            200         Russia          True
MOR            70        Morocco          True
EG             45          Egypt          True
```

1.16.2 CSV to DataFrame (1)

In:

```
# Import pandas as pd
import pandas as pd

# Import data cars.csv
cars = pd.read_csv("cars.csv")

# mencetak cars
print (cars)
```

Out:

```
  Unnamed: 0  cars_per_cap        country  drives_right
0         US           809  United States          True
1        AUS           731      Australia         False
2        JAP           588          Japan         False
3         IN            18          India         False
4         RU           200         Russia          True
5        MOR            70        Morocco          True
6         EG            45          Egypt          True
```

1.16.3 CSV to DataFrame (2)

In:

```
# memperbaiki import csv dengan melibatkan index_col
cars = pd.read_csv('cars.csv', index_col=0)

# mencetak cars
print (cars)
```

Out:

```
     cars_per_cap        country  drives_right
US            809  United States          True
AUS           731      Australia         False
JAP           588          Japan         False
IN             18          India         False
RU            200         Russia          True
MOR            70        Morocco          True
EG             45          Egypt          True
```

1.16.4 Square brackets

In:

```
# Print out country column as Pandas Series
print(cars['country'])
```

Out:

```
US      United States
AUS         Australia
JAP             Japan
IN              India
RU             Russia
MOR            Morocco
EG               Egypt
Name: country, dtype: object
```

In:

```
# Print out country column as Pandas DataFrame
print(cars[['country']])
```

Out:

```
           country
US   United States
AUS      Australia
JAP          Japan
IN           India
RU          Russia
MOR         Morocco
EG            Egypt
```

In:

```
# Print out DataFrame with country and drives_right columns
print(cars[['country','drives_right']])
```

Out:

```
           country  drives_right
US   United States          True
AUS      Australia         False
JAP          Japan         False
IN           India         False
RU          Russia          True
MOR         Morocco          True
EG            Egypt          True
```

In:

```
# Print out first 3 observations
print(cars.iloc[:3])
```

Out:

```
     cars_per_cap        country  drives_right
US            809  United States          True
AUS           731      Australia         False
JAP           588          Japan         False
```

In:

```
# Print out fourth, fifth and sixth observation
print(cars.iloc[3:6])
```

Out:

```
     cars_per_cap  country  drives_right
IN             18    India         False
RU            200   Russia          True
MOR            70  Morocco          True
```

1.16.5 loc (1)

In:

```
# mencetak data cars dari Japan
print (cars.loc['JAP'])

# mencetak data cars dari Australia dan Egypt
print (cars.loc[['AUS', 'EG']])
```

Out:

```
ars_per_cap      588
country          Japan
drives_right     False
Name: JAP, dtype: object
     cars_per_cap    country  drives_right
AUS           731  Australia         False
EG             45      Egypt          True
```

1.16.6 loc (2)

In:

```
# Mencetak drives_right untuk negara Morocco
print (cars.loc['MOR', 'drives_right'])

# Mencetak sub-DataFrame
print (cars.loc[['RU','MOR'],['country', 'drives_right']])
```

Out:

```
True
     country  drives_right
RU    Russia          True
MOR  Morocco          True
```

1.16.7 loc (3)

In:

```
# mencetak kolom drives_right sebagai Series
print(cars.loc[:,'drives_right'])
```

Out:

```
US       True
AUS     False
JAP     False
IN      False
RU       True
MOR      True
EG       True
Name: drives_right, dtype: bool
```

In:

```
# mencetak kolom drives_right sebagai DataFrame
print(cars.loc[:,['drives_right']])
```

Out:

```
     drives_right
US           True
AUS         False
JAP         False
IN          False
RU           True
MOR          True
EG           True
```

In:

```
# mencetak kolom cars_per_cap dan drives_right sebagai DataFrame
print(cars.loc[:,['cars_per_cap','drives_right']])
```

Out:

```
     cars_per_cap  drives_right
US            809          True
AUS           731         False
JAP           588         False
IN             18         False
RU            200          True
MOR            70          True
EG             45          True
```

1.16.8 iloc

In:

```
# mencetak observasi indeks 2
print(cars.iloc[2])
```

Out:

```
cars_per_cap      588
country         Japan
drives_right    False
Name: JAP, dtype: object
```

1.16.9 loop

In:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Adapt for loop
for lab, row in cars.iterrows() :
print(lab +": "+ str(row['cars_per_cap']))
```

Out:

```
US: 809
AUS: 731
JAP: 588
IN: 18
RU: 200
MOR: 70
EG: 45
```

In:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Code for loop that adds COUNTRY column
for lab, row in cars.iterrows():
cars.loc[lab, "COUNTRY"] = row['country'].upper()


# Print cars
print(cars)
```

Out:

```
     cars_per_cap        country  drives_right        COUNTRY
US            809  United States          True  UNITED STATES
AUS           731      Australia         False      AUSTRALIA
JAP           588          Japan         False          JAPAN
IN             18          India         False          INDIA
RU            200         Russia          True         RUSSIA
MOR            70        Morocco          True        MOROCCO
EG             45          Egypt          True          EGYPT
```

In:

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Menggunakan .apply(str.upper)
cars["COUNTRY"] = cars['country'].apply(str.upper)
```

2 Implementasi Python untuk Klasifikasi

2.1 Import libraries

In:

```
%pylab inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, \
                             AdaBoostClassifier, GradientBoostingClassifier
pylab.rcParams['figure.figsize'] = (10, 7)
import warnings
warnings.filterwarnings('ignore')
```

2.2 Load dataset

In:

```
data = pd.read_csv('dataset/loan_dataset.csv', index_col='Loan_ID')
data.head()
```

In:

```
data.shape
```

Out:

```
(480, 12)
```

In:

```
data.info()
```

Out:

```
<class 'pandas.core.frame.DataFrame'>
Index: 480 entries, LP001003 to LP002990
Data columns (total 12 columns):
Gender               480 non-null object
Married              480 non-null object
Dependents           480 non-null object
Education            480 non-null object
Self_Employed        480 non-null object
ApplicantIncome      480 non-null int64
CoapplicantIncome    480 non-null float64
LoanAmount           480 non-null int64
Loan_Amount_Term     480 non-null int64
Credit_History       480 non-null int64
Property_Area        480 non-null object
Loan_Status          480 non-null object
dtypes: float64(1), int64(4), object(7)
memory usage: 48.8+ KB
```

2.3 Eksplorasi Data

2.3.1 Distribusi Kelas

In:

```
data['Loan_Status'].value_counts()
```

Out:

```
Y    332
N    148
Name: Loan_Status, dtype: int64
```

In:

```
data['Loan_Status'].value_counts(normalize=True)
```

Out:

```
Y    0.691667
N    0.308333
Name: Loan_Status, dtype: float64
```

In:

```
sns.countplot(x=data.Loan_Status)
plt.show()
```

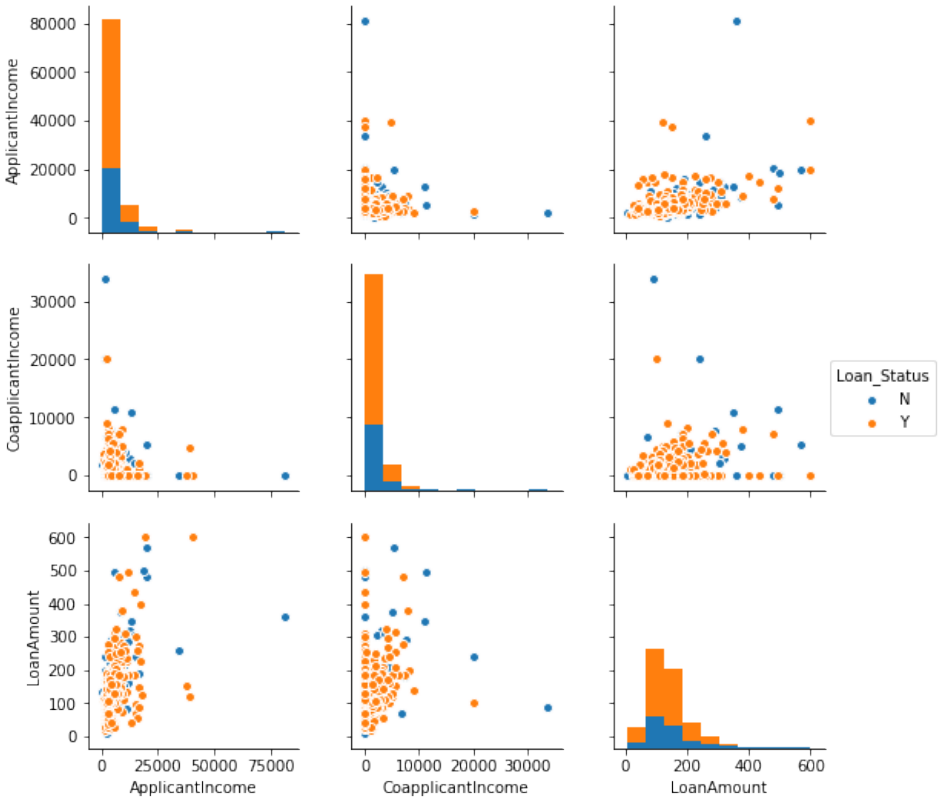
Out:

2.3.2 Matrix Plot

In:

```
sns.pairplot(data=data[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Status']],
  hue='Loan_Status')
plt.show()
```

Out:



2.4 Feature Engineering

In:

```
data['TotalIncome']=data['ApplicantIncome']+data['CoapplicantIncome']
data['Loan/Income']=data['LoanAmount']/data['TotalIncome']
data['Loan/Term']=data['LoanAmount']/data['Loan_Amount_Term']
data['RepaymentRatio']=(data['Loan/Term']*1000)/data['TotalIncome']
```

2.5 Encode Variabel Kategorik

In:

```
var_kategori = ['Gender','Married','Education','Self_Employed','Dependents','Credit_History','
  Loan_Status', 'Property_Area']
for feature in var_kategori:
    if feature in data.columns.values:
        data[feature] = LabelEncoder().fit_transform(data[feature])
```

2.6 Train-Test Split

In:

```
y = data['Loan_Status']
x = data.drop(['Loan_Status'], axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 123)
```

2.7 Training Model dan Prediksi

2.7.1 Decision Tree Classifier

In:

```
DecisionTree = DecisionTreeClassifier(random_state=123)
DecisionTree.fit(x_train, y_train)
```

Out:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=123,
            splitter='best')
```

In:

```
DecisionTree.score(x_train, y_train)
```

Out:

```
1.0
```

In:

```
DecisionTree.score(x_test, y_test)
```

Out:

```
0.6916666666666667
```

2.7.2 Bagging

In:

```
Bagging = BaggingClassifier(random_state=123)
Bagging.fit(x_train, y_train)
```

Out:

```
BaggingClassifier(base_estimator=None, bootstrap=True,
        bootstrap_features=False, max_features=1.0, max_samples=1.0,
        n_estimators=10, n_jobs=1, oob_score=False, random_state=123,
        verbose=0, warm_start=False)
```

In:

```
Bagging.score(x_train, y_train)
```

Out:

```
0.9888888888888889
```

In:

```
Bagging.score(x_test, y_test)
```

Out:

```
0.7666666666666667
```

2.7.3 Random Forest

In:

```
RF = RandomForestClassifier(random_state=123)
RF.fit(x_train, y_train)
```

Out:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=123, verbose=0, warm_start=False)
```

In:

```
RF.score(x_train, y_train)
```

Out:

```
0.9972222222222222
```

In:

```
RF.score(x_test, y_test)
```

Out:

```
0.7916666666666666
```

2.7.4 Gradient Boosting

In:

```
GB = GradientBoostingClassifier(random_state=123)
GB.fit(x_train, y_train)
```

Out:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                learning_rate=0.1, loss='deviance', max_depth=3,
                max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, n_estimators=100,
                presort='auto', random_state=123, subsample=1.0, verbose=0,
                warm_start=False)
```

In:

```
GB.score(x_train, y_train)
```

Out:

```
0.9416666666666667
```

In:

```
GB.score(x_test, y_test)
```

Out:

```
0.7833333333333333
```

2.7.5 Adaptive Boosting

In:

```
AB = AdaBoostClassifier(random_state=123)
AB.fit(x_train, y_train)
```

Out:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
        learning_rate=1.0, n_estimators=50, random_state=123)
```

In:

```
AB.score(x_train, y_train)
```

Out:

```
0.8861111111111111
```

In:

```
AB.score(x_test, y_test)
```

Out:

```
0.7583333333333333
```

2.8 Model Tuning

2.8.1 Decision Tree

In:

```
def GridSearch(x, y, model, parameters):
    clf = GridSearchCV(model, parameters, scoring='accuracy', n_jobs=-1, cv=5, verbose=1)
    clf.fit(x, y)
    print("Best Score: "+str(clf.best_score_))
    print("Best Params: "+str(clf.best_params_))
    return (clf)
```

In:

```
ListParams = {
    'criterion': ['gini','entropy'],
    'splitter': ['best', 'random'],
    'max_features': ['auto','sqrt','log2',None],
    'max_depth':[3,6,9],
    'class_weight':['balanced', None]
}

BestDecisionTree = GridSearch(x_train, y_train, DecisionTreeClassifier(random_state=123),
  ListParams)
```

Out:

```
Fitting 5 folds for each of 96 candidates, totalling 480 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.7s
Best Score: 0.8277777777777777
Best Params: {'class_weight': None, 'criterion': 'entropy', 'max_depth': 3, 'max_features': None,
   'splitter': 'random'}
[Parallel(n_jobs=-1)]: Done 480 out of 480 | elapsed:    5.1s finished
```

In:

```
BestDecisionTree.score(x_train, y_train)
```

Out:

```
0.8333333333333334
```

In:

```
BestDecisionTree.score(x_test, y_test)
```

Out:

```
0.7833333333333333
```

2.8.2 Random Forest

In:

```
ListParams = {
    'n_estimators': [50, 75, 100, 200],
    'max_depth':[1, 5, 10, 15, 20, 25, 30],
    'min_samples_leaf' : [1, 2, 4, 6, 8, 10],
    'max_features': [0.1, 'sqrt', 'log2', None]
}

BestRF = GridSearch(x_train, y_train, RandomForestClassifier(random_state=123), ListParams)
```

Out:

```
Fitting 5 folds for each of 672 candidates, totalling 3360 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    5.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   10.3s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   18.0s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   29.7s
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed:   46.4s
[Parallel(n_jobs=-1)]: Done 1784 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 2434 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 3184 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done 3360 out of 3360 | elapsed:  2.4min finished
Best Score: 0.8277777777777777
Best Params: {'max_depth': 5, 'max_features': None, 'min_samples_leaf': 4, 'n_estimators': 200}
```

In:

```
BestRF.score(x_train, y_train)
```

Out:

```
0.8444444444444444
```

In:

```
BestRF.score(x_test, y_test)
```

Out:

```
0.7833333333333333
```

2.8.3 Adapative Boosting

In:

```
ListParams = {
    'n_estimators': [50, 75, 100, 150, 200],
    'learning_rate': [0.001, 0.01, 0.1, 1],
    'algorithm' : ['SAMME', 'SAMME.R']
}

BestAB = GridSearch(x_train, y_train, AdaBoostClassifier(random_state=123), ListParams)
```

Out:

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   12.6s
Best Score: 0.8166666666666667
Best Params: {'algorithm': 'SAMME', 'learning_rate': 0.001, 'n_estimators': 50}
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:   13.7s finished
```

In:

```
BestAB.score(x_train, y_train)
```

Out:

```
0.8166666666666667
```

In:

```
BestAB.score(x_test, y_test)
```

Out:

```
0.7833333333333333
```

2.8.4 Gradient Boosting

In:

```
ListParams = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001, 0.01, 0.1],
    'n_estimators': [50, 75, 100, 200],
    'max_depth':[3, 5, 7],
    'subsample': [0.5, 0.75, 1],
    'max_features': [0.1, 'sqrt', 'log2', None]
}

BestGB = GridSearch(x_train, y_train, GradientBoostingClassifier(random_state=123), ListParams)
```

Out:

```
Fitting 5 folds for each of 864 candidates, totalling 4320 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    5.8s
[Parallel(n_jobs=-1)]: Done 282 tasks      | elapsed:    9.6s
[Parallel(n_jobs=-1)]: Done 673 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done 1023 tasks      | elapsed:   36.8s
[Parallel(n_jobs=-1)]: Done 1473 tasks      | elapsed:   57.3s
[Parallel(n_jobs=-1)]: Done 2266 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 3269 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 4320 out of 4320 | elapsed:  2.7min finished
Best Score: 0.8194444444444444
Best Params: {'learning_rate': 0.01, 'loss': 'deviance', 'max_depth': 3, 'max_features': None, '
  n_estimators': 200, 'subsample': 0.5}
```

In:

```
BestGB.score(x_train, y_train)
```

Out:

```
0.9138888888888889
```

In:

```
BestGB.score(x_test, y_test)
```

Out:

```
0.8
```