Chapter 1. Dasar Python untuk Deep Learning

Pada chapter ini dijelaskan dasar-dasar bahasa pemrograman python dan berbagai library pendukung yang sering digunakan dalam pengembangan model Deep Learning (DL). Python sendiri merupakan salah satu bahasa pemrograman popular yang banyak didukung oleh library-library dari pihak ketiga sehingga sangat powerfull untuk berbagai kegiatan analisis data, terutama data citra, video, dan teks. Adapun library yang digunakan dalam chapter ini meliputi Numpy, Pandas, Tensorflow, dan Matplotlib.

A. Dasar Python

Python termasuk ke dalam bahasa pemrograman tingkat tinggi. Kode python hampir mirip dengan pseudocode, sehingga kita dapat membuat ekspresi yang super singkat untuk mengekspresikan sebuah algoritma. Meskipun demikian, kode-kode bahasa python ini tetap mudah untuk dipahami oleh manusia. Sebagai contoh, pada cuplikan kode disajikan implementasi algoritma quicksort dalam bahasa Python.

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([3,6,8,10,1,2,1]))
# Print "[1, 1, 2, 3, 6, 8, 10]"
```

Terdapat dua versi berbeda untuk interpreter Python ini, yaitu 2.. dan 3.., dimana keduanya memiliki perbedaan yang cukup signifikan, sehingga kode-kode pada versi 2 sebagian besar tidak dapat diinterpretasikan oleh versi 3, begitupun sebaliknya. Dalam chapter ini digunakan interpreter versi 3.7. Untuk melakukan pengecekan versi python yang digunakan dapat menggunakan perintah berikut.

```
python --version
```

1. Tipe Data Dasar

Seperti pada bahasa-bahasa pemrograman lainnya, python memiliki tipe data dasar yang meliputi integer, float, Boolean, dan string. Namun, pada python, setiap tipe data tidak harus didefinisikan secara tertulis, setiap variable akan otomatis bertipe data sesuai dengan nilai yang diberikan kepadanya.

a. Bilangan Bulat dan Pecahan

Pada tipe data bilangan di python tidak terdapat operator unary increment (x++) dan decrement (x--), sehingga proses ini harus kita implementasikan sendiri. Adapun operasi lainnya sama seperti pada bahasa pemrograman lainnya. Contoh penggunaan tipe data bilangan adalah sebagai berikut.

```
x = 3
print(type(x)) # Print "<class 'int'>"
print(x)       # Print "3"
print(x + 1)   # Penjumlahan; print "4"
print(x - 1)   # Pengurangan; print "2"
print(x * 2)   # Perkalian; print "6"
print(x ** 2)  # Perpangkatan; print "9"
x += 1
print(x)       # Print "4"
x *= 2
print(x)       # Print "8"
y = 2.5
print(type(y)) # Print "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Print "2.5 3.5 5.0 6.25"
```

b. Boolean

Python mengimplementasikan semua operator Boolean seperti pada bahasa pemrograman lainnya, namun setiap operator menggunakan bahas inggris, tidak menggunakan symbol (&& , ||). Contoh penggunaannya adalah sebagai berikut.

```
t = True
f = False
print(type(t)) # Print "<class 'bool'>"
print(t and f) # Logical AND; print "False"
print(t or f)  # Logical OR; print "True"
print(not t)   # Logical NOT; print "False"
print(t != f)  # Logical XOR; print "True"
```

c. String

Python sangat banyak mendukung pengolahan string. Terdapat banyak sekali fungsi-fungsi yang dapat digunakan untuk memanipulasi string. Adapun contoh penggunaannya adalah sebagai berikut.

```
hello = 'hello'           # String dapat menggunakan single quotes
world = "world"           # maupun double quotes;
print(hello)              # Print "hello"
print(len(hello))         # Panjang String; print "5"
hw = hello + ' ' + world  # Penggabungan String (concatenation)
print(hw)                 # print "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)               # print "hello world 12"
```

```
s = 'hello'
print(s.capitalize()) # Setiap awal kata menjadi huruf kapital; print "Hello"
print(s.upper())      # Setiap huruf menjadi kapital; print "HELLO"
print(s.rjust(7))     # Rata kanan, padding dg spasi; print "  hello"
print(s.center(7))    # Rata tengah, padding dg spasi; print " hello "
print(s.replace('l','(ell)')) #Mereplace string, print "he(ell)(ell)o"
print('  world '.strip())      # hapus spasi di awal dan akhir; print "world"
```

2. Collection

Collection merupakan tipe data yang digunakan untuk menampung sekumpulan data sekaligus. Python memiliki tipe data collection yang meliputi: List, Dictionaries, Sets, dan Tuples.

a. List

List setara dengan array, namun ukurannya fleksibel atau bisa diubah sesuai kebutuhan serta dapat menampung data yang berbeda tipe data nya. Adapun contoh penggunaannya adalah sebagai berikut.

```
xs = [3, 1, 2]    # membuat list
print(xs, xs[2])  # Print "[3, 1, 2] 2"
print(xs[-1])     # Indeks negative diurut dari data terakhir; print "2"
xs[2] = 'foo'     # List dapat mengandung nilai yang berbeda tipe data
print(xs)         # Print "[3, 1, 'foo']"
xs.append('bar')  # Menambahkan data pada ujunglist
print(xs)         # Print "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Menghapus & mengembalikan nilai elemen terakhir dari list
print(x, xs)      # Print "bar [3, 1, 'foo']"
```

Pada list kita dimungkinkan untuk mengakses sebagaian elemen list (sub-list) secara langsung, proses ini lebih dikenal dengan istilah slicing, dimana contoh penggunaanya adalah sebagai berikut.

```
nums = list(range(5))# range digunakan utk membuat list integer
print(nums)          # Print "[0, 1, 2, 3, 4]"
print(nums[2:4])     # Sublist index 2 s.d. 4 (exclusive); print "[2, 3]"
print(nums[2:])      # Sublist index 2 s.d. akhir; print "[2, 3, 4]"
print(nums[:2])      # Sublist awal s.d. index 2 (exclusive); print "[0, 1]"
print(nums[:])       # Sublist semua elemen; print "[0, 1, 2, 3, 4]"
print(nums[:-1])     # Slice dg index negative; print "[0, 1, 2, 3]"
nums[2:4] = [8, 9]   # Mengganti nilai list pada index 2 s.d. 4
print(nums)          # Print "[0, 1, 8, 9, 4]"
```

Untuk mengakses setiap elemen pada list dapat digunakan looping. Beberapa cara looping yang dapat digunakan adalah seagai berikut.

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
    # Print "cat", "dog", "monkey"

for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
    # Print "#1: cat", "#2: dog", "#3: monkey"
```

Beberapa proses pada looping dapat disederhakan pengkodeannya, sehingga dapat mempersingkat kode kita. Hal ini sebagaimana pada contoh kode berikut.

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Print [0, 1, 4, 9, 16]

# Penyederhanaanya dapat ditulis sbb:
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Print [0, 1, 4, 9, 16]

#Dapat pula dikombinasikan dengan if/kondisi tertentu
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Print "[0, 4, 16]"
```

b. Dictionaries

Dictionaries mirip seperti array, namun pada dictionaries index yang digunakan dapat berupa string, maka setiap kali menyimpan data kita membutuhkan sepasang index/key dan nilainya (key,value). Tipe ini sepadan dengan Map pada bahasa Java. Adapun contoh penggunaanya sebagaimana tersaji berikut ini.

```
d = {'cat': 'cute', 'dog': 'furry'}   # Membuat dictionaries
print(d['cat'])                       # Mengakses nilai dictionaries ; print␣
 ↪"cute"
print('cat' in d)                     # Mengecek ketersediaan key dictionaries;␣
 ↪prints "True"
d['fish'] = 'wet'                     # Mengeset nilai pada dictionaries
print(d['fish'])                      # Print "wet"
#print(d['monkey'])                    # KeyError: 'monkey', key tidak terdapat␣
 ↪di d
print(d.get('monkey', 'N/A'))         # akses nilai dg index default; print "N/
 ↪A"
```

```
print(d.get('fish', 'N/A'))           # akses nilai dg index default; print␣
 ↪"wet"
del d['fish']                         # menghapus elemen pada dictionaries
print(d.get('fish', 'N/A'))           # "fish" tidak tersedia lagi; print "N/A"
```

Seperti halnya pada list, looping juga dapat diterapkan pada dictionaries untuk pengaksesan elemennya. Berikut pada cuplikan kode berikut disajikan contoh-contoh penggunaannya.

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Print "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"

for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Print "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"

#membuat dictionaries dengan inline looping
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  # Print "{0: 0, 2: 4, 4: 16}"
```

c. Sets

Set merupakan tipe data himpunan, dimana tidak ada duplikasi nilai dari elemen-elemennya. Berbeda dengan collection lainnya, set merupakan tipe collection yang tidak terurut. Contoh penggunaannya tersaji pada cuplikan kode berikut.

```
animals = {'cat', 'dog'}
print('cat' in animals)   # cek ketersediaan elemen; print "True"
print('fish' in animals)  # print "False"
animals.add('fish')       # Menambahkan elemen ke dalam set
print('fish' in animals)  # Print "True"
print(len(animals))       # menghitung jml elemen set; print "3"
animals.add('cat')        # Menambah data yg sudah tersedia, tidak berefek
print(len(animals))       # Print "3"
animals.remove('cat')     # Menghapus elemen dari set
print(len(animals))       # Print "2"
```

Mengingat Set merupakan collection yang tidak terurut, maka kita tidak dapat mengaksesnya secara terurut dengan menggunakan index seperti halnya pada list. Penggunaan looping pada set disajikan pada cuplikan kode berikut.

```
from math import sqrt

animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
```

```
    print('#%d: %s' % (idx + 1, animal))
# Print "#1: fish", "#2: dog", "#3: cat"

nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Print "{0, 1, 2, 3, 4, 5}"
```

d. Tuples

Tuple merupakan list yang mengandung serangkaian nilai terurut. Dalam banyak hal tuples penggunaanya sama seperti list, namun tuple dapat digunakan sebagai key dalam dictionaries maupun sebagai elemen dalam set, sedangkan list tidak. Contoh penggunan tuples adalah sebagai berikut.

```
d = {(x, x + 1): x for x in range(10)}  # membuat dictionary dg key tuples
t = (5, 6)        # membuat tuple
print(type(t))    # Print "<class 'tuple'>"
print(d[t])       # Print "5"
print(d[(1, 2)])  # Print "1"
```

3. Function

Untuk mendefinisikan fungsi dalam python dapat menggunakan kata tercadang def. Kode yang merupakan body dari fungsi tersebut dibuat menjorok ke dalam. Adapun contoh fungsi disajikan pada cuplikan kode berikut.

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Print "negative", "zero", "positive"
```

Seperti halnya pada bahasa pemrograman lainnya, kita dapat menambahkan optional argument pada fungsi dengan memberi nilai inisial. Adapun penggunaanya sebagaimana terlihat pada cuplikan kode berikut.

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)
```

```
hello('Bob') # Print "Hello, Bob"
hello('Fred', loud=True)  # Prints"HELLO, FRED!"
```

4. Class

Dalam paradigma OOP, untuk implementasinya dibutuhkan definisi kelas. Dalam python pendefinisian sebuah kelas cukup sederhana sebagaimana terlihat pada cuplikan kode berikut.

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # mendefinisikan property kelas

    # Contoh method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred')  # Instansiasi Greeter
g.greet()            # memanggil method; print "Hello, Fred"
g.greet(loud=True)   # memanggil method dg parameter; print "HELLO, FRED!"
```

B. Numpy

Numpy merupakan library python yang banyak digunakan untuk pengolahan data numerik dan data scientific. Library ini menyediakan algoritma-algoritma pengolahan array multi dimensi dengan performa yang sangat baik, seperti halnya pada MATLAB.

1. Array

Sebuah Numpy Array merupakan kumpulan nilai dengan tipe data yang sama dalam bentuk Grid dan berindekskan tuple integer non-negatif. Jumlah dimensi dari array tsb. disebut Rank, dan ukuran array setiap dimensinya disebut shape yang dinyatakan dalam sebuah tuple.

Kita dapat menginisialisasi Numpy Array dengan sebuah list, dan mengakses setiap indexnya dengan menggunakan kurung siku sebagaimana terlihat pada cuplikan berikut.

```
import numpy as np

a = np.array([1, 2, 3])   # membuat Numpy array rank 1
print(type(a))            # Print "<class 'numpy.ndarray'>"
```

```
print(a.shape)             # Print "(3,)"
print(a[0], a[1], a[2])    # Print "1 2 3"
a[0] = 5                   # Merubah nilai sebuah elemen array
print(a)                   # Print "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # membuat Numpy array rank 2
print(b.shape)                     # Print "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Print "1 2 4"
```

Selain itu, Numpy juga menyediakan banyak fungsi untuk membuat dan menginisialisasi Numpy array. Contoh pemanfaatannya dapat adalah sebagai berikut.

```
import numpy as np

a = np.zeros((2,2))   # Membuat array dengan nilai nol pada setiap elemennya
print(a)              # Print  "[[ 0.  0.]
                      #         [ 0.  0.]]"

b = np.ones((1,2))    # Membuat array dengan nilai satu pada setiap elemennya
print(b)              # Print "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Membuat array dengan nilai tertentu
print(c)               # Print  "[[ 7.  7.]
                       #         [ 7.  7.]]"

d = np.eye(2)         # Memebuat matriks identitas 2x2
print(d)              # Print  "[[ 1.  0.]
                      #         [ 0.  1.]]"

e = np.random.random((2,2))  # Membuat array dg nilai random
print(e)                     # print   "[[ 0.91940167  0.08143941]
                             #          [ 0.68744134  0.87236687]]"
```

2. Array Indexing

Pada Numpy terdapat beberapa cara untuk melakukan pengaksesan index dari sebuah array, meliputi slicing, integer indexing, dan Boolean indexing.

a. Slicing

Sama seperti halnya pada python core nya, Numpy mendukung proses slicing atau pengaksesan sublist/sub-array. Namun karena dimensi Numpy Array memiliki multi dimensi, maka kita harus menentukan index slice untuk masing-masing dimensinya. Contoh penggunaanya adalah sebagai berikut.

```
import numpy as np

```

```
# membuat array rank 2 dg shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Menggunakan slicing untuk mendapatkan subarray yg terdiri dari 2 baris
# pertama dan kolom 1-2, sehingga b mengandung nilai:
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# Subarray/Slice mengacu pada data/alamat memori yang sama, sehingga
# ketika merubah sub array, artinya merubah isi array utamanya
print(a[0, 1])   # Print "2"
b[0, 0] = 77     # b[0, 0] merujuk pada lokasi yg sama dg a[0, 1]
print(a[0, 1])   # Print "77"
```

Slicing juga dapat dikombinasikan dengan integer indexing, namun ketika hal ini dilakukan, maka hasilnya rank array hasil akan lebih rendah dibanding array aslinya, sebagaimana terlihat pada cuplikan kode berikut.

```
import numpy as np
# membuat array dg shape (3, 4), rank 2
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

row_r1 = a[1, :]    # Rank 1, menampilkan baris kedua dari a
row_r2 = a[1:2, :]  # Rank 2, menampilkan baris kedua dari a
print(row_r1, row_r1.shape)  # Print "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Print "[[5 6 7 8]] (1, 4)"

# Begitupun ketika mengakses kolom
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # Print "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # Print "[[ 2]
                             #         [ 6]
                             #         [10]] (3, 1)"
```

b. Integer Indexing

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# Contoh, array hasil akan memiliki shape (3,)
print(a[[0, 1, 2], [0, 1, 0]])                # Print "[1 4 5]"
```

```
# Operasi di atas, sama dengan:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Print "[1 4 5]"

print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Operasi di atas, sama dengan:
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

C. Pandas

Pandas merupakan library open source yang banyak digunakan dalam melakukan analisis, manipulasi, dan pengelolaan data. Library ini menyediakan fungsi-fungsi yang powerfull dan fleksibel serta mudah digunakan terutama dalam mempersiapkan data agar dapat digunakan atau menjadi input untuk model machine learning.

Adapun dokumentasi selengkapnya dapat diakses pada tautan berikut:Dokumentasi Pandas

1. Tipe Data

Untuk dapat menggunakan Pandas, kita perlu meng-import library-nya sebagai berikut.

```
import pandas as pd
```

Pandas memiliki dua tipe data utama, yaitu Series dan DataFrame.

a. Series

Series merupakan tipe data yang dapat digunakan untuk menyimpan koleksi data satu dimensi seperti halnya pada Array satu dimensi. Series hanya dapat menyimpan koleksi data dengan tipe sama dan dapat diperlakukan sebagaimana pada numpy array. Berikut contoh pembuatan series pada Pandas.

```
students = pd.Series(["Nono","Toto","Budi"]) # Membuat dan menginisialisasi␣
 ↪Series
print(students)                              # Menampilkan Series
```

```
0    Nono
1    Toto
2    Budi
dtype: object
```

```
# numbers = pd.Series("one",2,3.0) # jika dijalankan akan terjadi error
                                   # karena tipe data nilai dalam koleksi
                                   # berbeda-beda
```

Series juga dapat dibuat dari data koleksi yang sudah ada, seperti dictionary, dan lainnya, sehingga key nya tidak harus selalu berupa tipe integer. Adapun contohnya sebagaimana terlihat pada cuplikan kode berikut.

```
students_dict = {"a":"Andi","b":"Budi","c":"Cintya"} # dict sumber data
students2 = pd.Series(students_dict)                 # membuat series dari dict
print(students2)                                     # menampilkan students2
```

```
a      Andi
b      Budi
c    Cintya
dtype: object
```

Data pada series dapat diakses dengan menggunakan index/key nya, seperti pada cuplikan kode berikut.

```
print(students[1])    # mengakses data dg idx 1 dari students, print: Toto
print(students2["a"]) # mengakses data dg key a dari students2, print: Andi
```

```
Toto
Andi
```

b. DataFrame

DataFrame merupakan tipe data yang dapat digunakan untuk menyimpan koleksi data dua dimensi seperti tabel dan matriks. Selain itu, tipe data ini terdiri dari baris dan kolom seperti pada tabel SQL. Antara satu kolom dengan kolom lainnya dapat memiliki tipe data yang berbeda, namun tetap pada satu kolom yang sama hanya dapat menampung nilai dengan tipe data yang sama. Adapun anatomi sebuah DataFrame dapat dilihat pada gambar berikut.

Seperti halnya pada Series, untuk membuat DataFrame terdapat beberapa cara. Hal ini dapat dilakukan dengan menggunakan koleksi data yang sudah ada, bahkan menggunakan Series yang telah dibuat sebelumnya. Namun pastikan ketika membuat DataFrame jumlah data pada collection ataupun pada series setiap kolom haruslah sama, karena data tersebut akan menjadi cell untuk setiap baris data. Adapun contohnya sebagaimana terlihat pada cuplikan kode berikut.

```
#DataFrame dibuat dari series
students = pd.Series(["Nono","Toto","Budi","Lena"])
age = pd.Series([16,21,18,16])
gpa = pd.Series([3.75,3.82,3.05,3.78])
#Pastikan jml data students dan age sama
df_students = pd.DataFrame({
    "name":students,
    "age": age,
    "gpa": gpa
})
print(df_students)
```

```
   name  age   gpa
0  Nono   16  3.75
```

```
1  Toto   21  3.82
2  Budi   18  3.05
3  Lena   16  3.78
```

Pada contoh tersebut terlihat bahwa key dari dictionary menjadi header untuk kolom pada DataFrame. Adapun contoh pembuatan DataFrame dari dictionary adalah seperti terlihat pada cuplikan kode berikut.

```
dict_students = {
    "name":["Nono","Toto","Budi","Lena"],
    "age":[16,21,18,16],
    "gpa":[3.75,3.82,3.05,3.78]
    }
df_students2 = pd.DataFrame(dict_students)
print(df_students2)
```

```
   name  age   gpa
0  Nono   16  3.75
1  Toto   21  3.82
2  Budi   18  3.05
3  Lena   16  3.78
```

Untuk mengakses data pada DataFrame dapat dilakukan dengan berbagai cara. Salah satunya dengan menggunakan kurung siku([]). Adapun contoh penggunaannya sebagaimana terlihat pada cuplikan kode berikut.

```
# Mengakses kolom name
print(df_students["name"]) #bisa juga df_students.name
```

```
0    Nono
1    Toto
2    Budi
3    Lena
Name: name, dtype: object
```

```
#Mengakses baris ke-1 s.d. ke-2 (ekslusif, jadi 3 tidak masuk)
print(df_students[1:3])
```

```
   name  age   gpa
1  Toto   21  3.82
2  Budi   18  3.05
```

```
#Menampilkan 3 baris pertama
print(df_students[:3])
```

```
   name  age   gpa
0  Nono   16  3.75
1  Toto   21  3.82
2  Budi   18  3.05
```

```
#Menampilkan 2 baris terakhir
print(df_students[-2:])
```

```
   name  age   gpa
2  Budi   18  3.05
3  Lena   16  3.78
```

Selain itu, data dapat juga diakses menggunakan labelnya melalui atribut loc. Pada loc index pertama menyatakan label barisnya, sedangkan index kedua menyatakan index kolomnya.

```
#Mengakses semua baris, pada kolom age
print(df_students.loc[:,["age"]])
```

```
   age
0   16
1   21
2   18
3   16
```

```
#Mengakses baris 1 s.d. 2, pada kolom name dan age
print(df_students.loc[1:2,["name","age"]]) #inclusive
```

```
   name  age
1  Toto   21
2  Budi   18
```

```
#Mengakses baris 1 dan 3, pada kolom name dan gpa
print(df_students.loc[[1,3],["name","gpa"]])
```

```
   name   gpa
1  Toto  3.82
3  Lena  3.78
```

```
#Mengkases satu baris data pada idx ke 0
print(df_students.loc[0])
```

```
name    Nono
age       16
Name: 0, dtype: object
```

Selain loc, data dapat juga diakses menggunakan posisinya melalui atribut iloc. Sama seperti pada loc, pada iloc juga index pertama menyatakan label barisnya, dan index kedua menyatakan index kolomnya.

```
#Mengakses semua baris, pada kolom age
print(df_students.iloc[:,[1]])
```

```
   age
0   16
```

```
1   21
2   18
3   16
```

```
#Mengakses baris 1 s.d. 2, pada kolom name dan age
print(df_students.iloc[1:2,[0,1]]) #inclusive
```

```
   name  age
1  Toto   21
```

```
#Mengakses baris 1 dan 3, pada kolom name dan gpa
print(df_students.iloc[[1,3],[0,2]])
```

```
   name   gpa
1  Toto  3.82
3  Lena  3.78
```

```
#Mengkases satu baris data pada idx ke 0
print(df_students.iloc[0])
```

```
name    Nono
age       16
gpa     3.75
Name: 0, dtype: object
```

```
```

2. Melakukan Import Data

Pada kegiatan analisis data, biasanya data didapat dari sumber lain yang telah tersedia, misalnya dari dataset publik. Oleh karena itu, proses pembuatan DataFrame tidak lagi manual, melainkan dapat dilakukan melalui proses import. Sebagai contoh kita akan mengimport file csv dari link berikut: https://drive.google.com/uc?export=view&id=1nryxZ7nFeVE04LlTGjqD6HUJu3ZlboAk, maka dapat dilakukan dengan menggunakan metode read_csv.

```
url = "https://drive.google.com/uc?
 ↪export=view&id=1nryxZ7nFeVE04LlTGjqD6HUJu3ZlboAk"
df = pd.read_csv(url)
df
```

```
      Make Colour  Odometer (KM)  Doors    Price
0    Honda  White        35431.0    4.0  15323.0
1      BMW   Blue       192714.0    5.0  19943.0
2    Honda  White        84714.0    4.0  28343.0
3   Toyota  White       154365.0    4.0  13434.0
```

```
4    Nissan   Blue       181577.0    3.0  14043.0
..      ...    ...            ...    ...      ...
995  Toyota  Black        35820.0    4.0  32042.0
996     NaN  White       155144.0    3.0   5716.0
997  Nissan   Blue        66604.0    4.0  31570.0
998   Honda  White       215883.0    4.0   4001.0
999  Toyota   Blue       248360.0    4.0  12732.0

[1000 rows x 5 columns]
```

3. Eksplorasi Data

Sebelum melakukan analisis data, langkah yang umum dilakukan pertama kali adalah melakukan eksplorasi dan melihat keadaan data yang kita miliki. Terdapat beberapa perintah dalam pandas yang dapat mempermudah proses eksplorasi data ini, yaitu sebagai berikut:

a. Mendeskripsikan Data

Untuk mengetahui keadaan data yang kita miliki, dapat menggunakan methods dan atribut berikut:

.dtypes: Menampilkan tipe data utk setiap kolom.

```
df.dtypes
```

```
Make              object
Colour            object
Odometer (KM)    float64
Doors            float64
Price            float64
dtype: object
```

.describe(): Menampilkan statistik data

```
df.describe()
```

```
       Odometer (KM)       Doors         Price
count     950.000000  950.000000    950.000000
mean   131253.237895    4.011579  16042.814737
std     69094.857187    0.382539   8581.695036
min     10148.000000    3.000000   2796.000000
25%     70391.250000    4.000000   9529.250000
50%    131821.000000    4.000000  14297.000000
75%    192668.500000    4.000000  20806.250000
max    249860.000000    5.000000  52458.000000
```

.info(): Menampilkan informasi jumlah data, jml data non-null, serta tipe data untuk setiap kolom

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Make           951 non-null    object
 1   Colour         950 non-null    object
 2   Odometer (KM)  950 non-null    float64
 3   Doors          950 non-null    float64
 4   Price          950 non-null    float64
dtypes: float64(3), object(2)
memory usage: 39.2+ KB
```

.columns: Menampilkan daftar kolom

```
df.columns
```

```
Index(['Make', 'Colour', 'Odometer (KM)', 'Doors', 'Price'], dtype='object')
```

Nama kolom bersifat case-sensitive

.namaKolom.unique(): Menampilkan nilai unik dari suatu kolom

```
df.Colour.unique()
```

```
array(['White', 'Blue', 'Red', 'Green', nan, 'Black'], dtype=object)
```

b. Menampilkan Cuplikan Data

Selain mendeskripsikan data, kadang kita juga perlu melihat cuplikan data yang kita miliki untuk mendapatkan ide dalam melakukan proses analisis data. Beberapa methods dan atribut DataFrame yang berguna dalam menampilkan cuplikan data yaitu sebagai berikut.

.head(n): Menampilkan n baris pertama data, default n=5

```
df.head(6)
```

```
     Make Colour  Odometer (KM)  Doors    Price
0   Honda  White        35431.0    4.0  15323.0
1     BMW   Blue       192714.0    5.0  19943.0
2   Honda  White        84714.0    4.0  28343.0
3  Toyota  White       154365.0    4.0  13434.0
4  Nissan   Blue       181577.0    3.0  14043.0
5   Honda    Red        42652.0    4.0  23883.0
```

.tail(n): Menampilkan n baris terakhir data, default n=5

```
df.tail(6)
```

```
       Make Colour  Odometer (KM)  Doors    Price
994     BMW   Blue       163322.0    3.0  31666.0
995  Toyota  Black        35820.0    4.0  32042.0
996     NaN  White       155144.0    3.0   5716.0
997  Nissan   Blue        66604.0    4.0  31570.0
998   Honda  White       215883.0    4.0   4001.0
999  Toyota   Blue       248360.0    4.0  12732.0
```

.loc[kriteria]: Memilih baris sesuai kriteria

```
#Menampilkan data dengan nilai Doors < 4
df.loc[df.Doors < 4]
```

```
       Make Colour  Odometer (KM)  Doors    Price
4    Nissan   Blue       181577.0    3.0  14043.0
18   Nissan  White        67991.0    3.0   9109.0
33   Nissan  Green       153554.0    3.0   9780.0
38   Nissan   Blue       146430.0    3.0   9224.0
48   Nissan  White       107096.0    3.0   6075.0
..      ...    ...            ...    ...      ...
949  Nissan   Blue        57558.0    3.0  16622.0
957  Nissan    NaN        39639.0    3.0  12315.0
971     BMW  Black       178164.0    3.0  24891.0
994     BMW   Blue       163322.0    3.0  31666.0
996     NaN  White       155144.0    3.0   5716.0

[64 rows x 5 columns]
```

```
#Mendeskripsikan data dengan nilai Doors < 4 dan Colour='Blue'
df.loc[(df.Doors < 4) & (df.Colour=='Blue')].describe()
```

```
       Odometer (KM)  Doors         Price
count      26.000000   28.0     28.000000
mean   124932.807692    3.0  18030.142857
std     63014.379499    0.0   9628.013709
min     18044.000000    3.0   4863.000000
25%     68820.750000    3.0  11648.750000
50%    136503.000000    3.0  15102.500000
75%    178722.000000    3.0  24171.250000
max    234146.000000    3.0  39021.000000
```

c. Agregasi Data

Terdapat banyak cara guna menampilkan agregasi data pada DataFrame. Berikut beberapa contoh pemanfataannya.

.sum(): Menampilkan jumlah/hasil penjumlahan data

```
df.Doors.sum()
```

```
3811.0
```

.mean(): Menampilkan data *rataan*

```
df.Price.mean()
```

```
16042.814736842105
```

.isnull().sum(): Menampilkan data kosong

```
df.isnull().sum()
```

```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

.crosstab(kolom1,kolom2): Menampilkan jml data kombinasi nilai dari 2 kolom berbeda

```
pd.crosstab(df["Colour"],df["Doors"])
```

```
Doors   3.0  4.0  5.0
Colour
Black     3   81    5
Blue     28  238   20
Green     5   61    6
Red       4   65   12
White    22  326   28
```

.groupby(): Mengelompokkan Data

```
df.groupby(["Colour","Doors"]).sum()
```

```
              Odometer (KM)      Price
Colour Doors
Black  3.0         390403.0    61096.0
       4.0       10546525.0  1190949.0
       5.0         791114.0   136552.0
Blue   3.0        3248253.0   504844.0
       4.0       31300457.0  3456125.0
       5.0        2320941.0   560981.0
```

```
Green  3.0         677112.0    71197.0
       4.0        7819386.0   833588.0
       5.0         780625.0   108772.0
Red    3.0         644422.0    60281.0
       4.0        7835917.0   867177.0
       5.0        1231900.0   211318.0
White  3.0        2967363.0   335700.0
       4.0       38146682.0  4718099.0
       5.0        3126449.0   776255.0
```

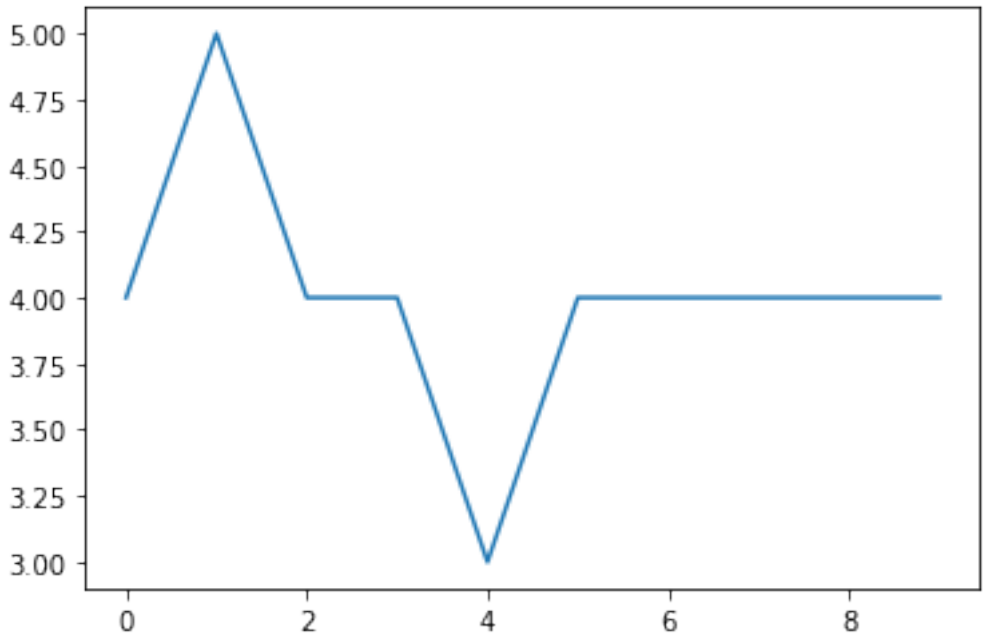
d. Visualisasi Data

Selain menampilkan data dalam bentuk tabel dan statistik, terkadang untuk melihat lebih jauh kondisi data kita perlu melakukan visualisasi dalam bentuk grafik, berikut beberapa contoh visualisasi yang dapat dilakukan pada pandas.

.plot(): Membuat grafik garis

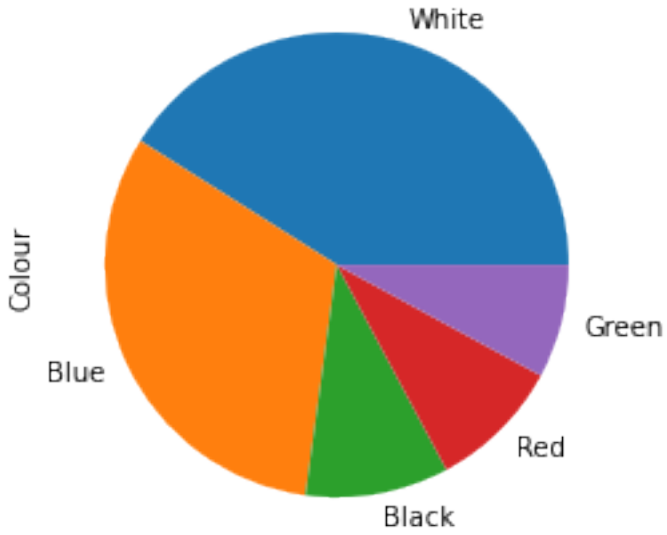
```
df.Doors[:10].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0127695210>
```




```
df.Colour.value_counts().plot(kind="pie")
```

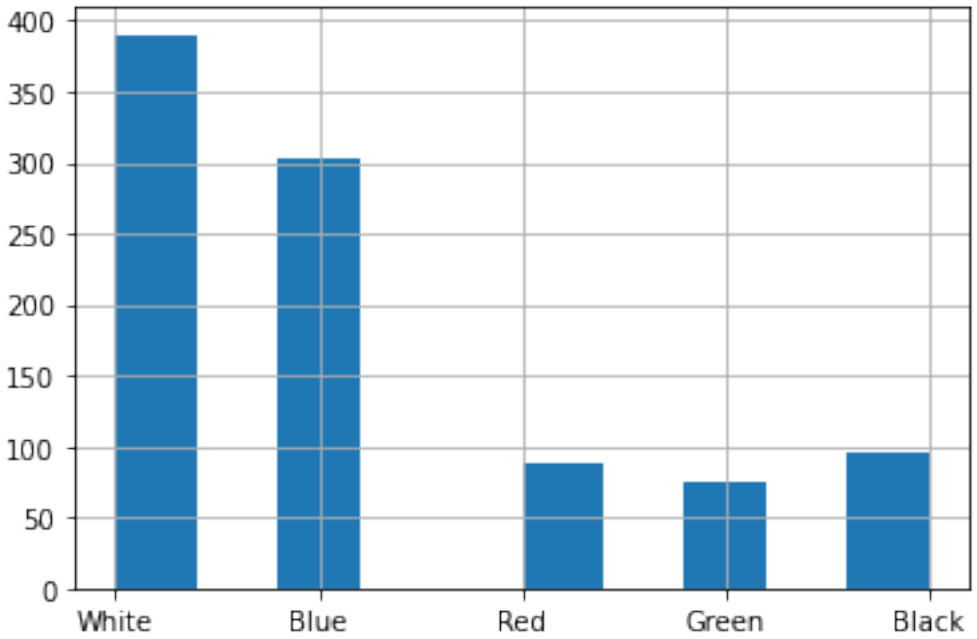
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f01278895d0>
```



.hist(): Menamplikan histogram distribusi data

```
df.Colour.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f012774f5d0>
```



4. Manipulasi Data

Selain melakukan eksplorasi data, terkadang kita perlu melakukan manipulasi data dalam rangka praprocessing. Hal ini dilakukan guna memperbaiki kualitas data yang kita miliki, seperti menghandel data kosong, merubah huruf besar menjadi huruf kecil dan lain-lain. Adapun beberapa methods yang sering digunakan dapat dilihat pada cuplikan kode berikut.

.str.lower(): Merubah semua huruf pada tipe data string menjadi huruf kecil

```
# Mendapatkan data colour dan merubahnya menjadi huruf kecil
series_huruf_kecil = df.Colour.str.lower()
# Mereplace series Colour pada df dengan hasil perubahan
df.Colour = series_huruf_kecil
df.head()
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	white	35431.0	4.0	15323.0
1	BMW	blue	192714.0	5.0	19943.0
2	Honda	white	84714.0	4.0	28343.0
3	Toyota	white	154365.0	4.0	13434.0
4	Nissan	blue	181577.0	3.0	14043.0

.fillna(): Mengisi data yang kosong pada DataFrame

```
#Terdapat missing value (NaN) pada kolom Doors
df.loc[df.Doors.isnull()].head()
```

	Make	Colour	Odometer (KM)	Doors	Price
31	Toyota	white	108569.0	NaN	6866.0
47	Toyota	blue	243969.0	NaN	16138.0
51	Honda	white	161068.0	NaN	13698.0
75	Toyota	blue	156478.0	NaN	20424.0
88	Nissan	red	61892.0	NaN	18160.0

```
#Mising value dapat diisi dengan nilai tertentu sesuai kebutuhan
#Misal: mean, modus, median, atau nilai tertentu
df.Doors.fillna(df.Doors.mode,
                inplace=True) #default: false, autoreplace/tidak
```

```
df.loc[df.Doors.isnull()].head()
#Sudah tidak ada missing values
```

```
Empty DataFrame
Columns: [Make, Colour, Odometer (KM), Doors, Price]
Index: []
```

```
df.isnull().sum()
```

```
Make             49
Colour           50
Odometer (KM)    50
Doors             0
Price            50
dtype: int64
```

.dropna(): Menghapus baris yang mengandung missingvalue

```
#Data sebelum dihapus missing-valuenya
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Make           951 non-null    object 
 1   Colour         950 non-null    object 
 2   Odometer (KM)  950 non-null    float64
 3   Doors          1000 non-null   object 
 4   Price          950 non-null    float64
dtypes: float64(2), object(3)
memory usage: 39.2+ KB
```

```
#Menghapus baris yg mengandung missing value
df.dropna(inplace=True)
#Data setelah dihapus missing-valuenya
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 815 entries, 0 to 999
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Make           815 non-null    object 
 1   Colour         815 non-null    object 
 2   Odometer (KM)  815 non-null    float64
 3   Doors          815 non-null    object 
 4   Price          815 non-null    float64
dtypes: float64(2), object(3)
memory usage: 38.2+ KB
```

.drop(): Menghapus kolom atau baris

```
#Menghapus kolom Colour
df.drop("Colour",axis=1,inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 815 entries, 0 to 999
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Make            815 non-null    object 
 1   Odometer (KM)   815 non-null    float64
 2   Doors           815 non-null    object 
 3   Price           815 non-null    float64
dtypes: float64(2), object(2)
memory usage: 31.8+ KB
```

```
#Data sebelum baris kedua dihapus
df.head(5)
```

	Make	Odometer (KM)	Doors	Price
0	Honda	35431.0	4.0	15323.0
1	BMW	192714.0	5.0	19943.0
2	Honda	84714.0	4.0	28343.0
3	Toyota	154365.0	4.0	13434.0
4	Nissan	181577.0	3.0	14043.0

```
#Menghapus data baris kedua
df.drop(1,axis=0,inplace=True)
#Data sebelum baris kedua dihapus
df.head(5)
```

	Make	Odometer (KM)	Doors	Price
0	Honda	35431.0	4.0	15323.0
2	Honda	84714.0	4.0	28343.0
3	Toyota	154365.0	4.0	13434.0
4	Nissan	181577.0	3.0	14043.0
5	Honda	42652.0	4.0	23883.0

5. Melakukan Export Data

Setelah melakukan praprocessing, data yang sudah siap digunakan sebaiknya disimpan ke dalam file yang terpisah, sehingga praprocess yang sudah dilakukan tidak perlu diualang kembali di kemudian hari. Untuk menyimpan dataframe ke dalam sebuah file dapat digunakan method to_csv(url) sebagaimana terlihat pada cuplikan kode berikut.

```
df.to_csv("data_bersih.csv")
```

E. Matplotlib

D. Tensorflow