Python Cheatsheed

Contents

Foundamentals

Data types

Name	Data type	Description
string	'str'	sequence of characters
integer	1	integers numbers
float	0.1	decimals numbers
complex	1j	imaginary numbers
boolean	True/False	logical values
list	[a, b, ...]	mutable ordered sequence
tuple	(a, b, ...)	immutable ordered sequence
range	range(init, fin, step)	immutable ordered sequence
dictionary	{key=value, ...}	mutable unordered mapping
set	{x, ...}	mutable unordered set
frozenset	frozenset({x, ...})	immutable unordered set
byte		
bytearray		
memoryview		

Data conversion

	convert in string data type
`int()`	convert in integer data type
`float()`	convert in float data type
`bool()`	convert in bool data type
`list()`	convert in list data type
`dict()`	convert in dict data type
`set()`	convert in set data type

Generic operations

`len(x)`	variable length
`min(x), max(x)`	min/max value
`sorted(x)`	sort a list
`enumerate(x, ...)`	iterator
`zip(x)`	iterator on tuple
`all(x)`	true if all are true
`any(x)`	true if at least one is true

Sting operations

`s1 + s2`	concatenation
`s * int`	multiplication
`s[i]`	indexing
`s.index()`	??
`s.find()`	??
`s.strip()`	remove white spaces
`s.lower()`	lowercase
`s.upper()`	uppercase
`s.split([sep])`	creates string array of words
`s.join(seq)`	creates string array of words
`s.find('string')`	match string in string
`s.replace('old', 'new')`	replace string

List operations

`l.append(x)`	append a element
`l.extend(x)`	append as sequence of elements
`l.insert(i,x)`	insert variable at i position
`l.remove(x)`	remove value x
`l.pop([i])`	remove and return item at index i
`l.sort()`	sort
`l.revert()`	reverse sort

Dictionaries operations

`d[key] = value`	assign value to a key
`d.clear()`	clear dictionary
`d.update(d2)`	??
`d.key()`	??
`d.values()`	??
`d.items()`	??
`d.pop(key[default])`	??
`d.popitem()`	??
`d.get(key[default])`	??
`d.setdefault(key[default])`	??

Sets operations

`s.update(s2)`	??
`s.copy()`	??
`s.add(key)`	??
`s.remove(key)`	??
`s.discard(key)`	??
`s.clear()`	??
`s.pop()`	??

Operators

Arithmetic operators

`+, -, *, /`	basic operations
`%`	module
`**`	exponential
`//`	floor

Logic operators

`==`	equal
`!=`	not equal
`<,>`	strict inequality
`<=,>=`	inequality

Boolean operators

`x and y`	and operator
`x or y`	or operator
`not x`	not operator

Assignment operators

`=`	value assignment
`+=, -=, *=, /=, **=, //=`	arithmetic assignment

Other operators

`x is y`	true if x is y
`x is not y`	true if x is not y
`x in y`	true x is in y sequence
`x not in y`	true x is not in y sequence

Indexing

`x[i]`	list i-th element
`x[init_val:fin_val]`	list slicing

Loops/Conditions

if/elif/else

```
if condition:
    ...
elif condition:
    ...
else:
    default
```

while

```
while condition:
    ...
```

for

```
\for condition:
    ...
```

Keywords:	`break`	end the loop
	`continue`	continue the loop
	`\t`	tab

Functions

```
# function definition
def fun_name(param, *args, param=default, **kwargs):
    # function body
    ...
    return x    # value to return

# function call
a = fun_name(arg_1, ..., arg_n))
```

Input/Output

```
print(x)                  print function
x = input("string: ")     input function
```

Keywords:

\n	newline
\s	space
\t	tab

Filesystem

```
f = open("file","mode",encoding="enc")   open file
f.write(text)                            writing
f.writelines(list of lines)              writing
f.read()                                 reading chars
f.readlines()                            reading lines
f.readline()                             next line
f.close()                                close file
```

mode:
- r: read
- a: append
- w: write
- x: create

enc:
- utf8
- ascii
- latin1

```
# open file
with open("file", "mode") as f:
    for line in f:
        ...
```

Error Handling

```
raise ExcClass()
try:
    <code>
except Exception as exc:
    <error>
```

Common Modules

Math

```
import math
```

	absolute value
	round value
()	sine
cos()	cosine
tan()	tangent
sqrt()	square root
log()	logarithm
ceil()	ceil
floor()	floor

Argumens Parser

```
import argparse

# create object
arg = argparse.ArgumentParser()

# add argument
arg.add_argument('-strink_arg_name', 'full_arg_name',
↪  required=True, help='description')

args = vars(arg.parse_args())
```

Filesystem

```
from pathlib import Path

# creating path
address = Path('home', '...', 'file')
actual_path = Path.cwd()
home_path = Path.home()

# absolute path
address.is_absolute()

# get path parts
address.parent      # previous folder
address.name        # final position
address.stem        # home
address.drive
```

File Management

csv

```
import csv

# write
f = open('file.csv', 'w', newline = '')
f_csv = csv.writer(f, delimiter='\t',
↪  lineterminator='\n\n')
f_csv.writerow(['elem_0', ..., 'elem_n'])

# write as dictionary
f = open('file.csv', 'w', newline = '')
f_csv = csv.DictWriter(f, ['key_0', ..., 'key_n'])
f_csv.writeheader()
f_csv.writerow({'key_0': value_0, ..., 'key_n':
↪  value_n})

# read
f = open('file.csv', 'r')
f_csv = csv.reader(f)
for row in f_csv:
    print('Row #' + str(f_csv.line_num) + ' ' +
    ↪  str(row))

# read dictionary
f = open('file.csv')

f_csv = csv.DictReader(f, ['key_0', ..., 'key_n'])
for row in f_csv:
    print(row['key_0'], ..., row['key_n'])

# close
f.close()
```

json

```
import json

# write
json_write = {'key_0': value_0, ..., 'key_n':
↪  value_n}
stringOfJsonData = json.dumps(json_write)

# read
json_read = '{'key_0': value_0, ..., 'key_n':
↪  value_n}'
jsonDataAsPythonValue = json.loads(json_read)
print(jsonDataAsPythonValue)
```

Regular Expression

```
# object
regex_obj = re.compile(r'string_to_match')

# find the first result
match = regex_obj.search(Text)
regex_obj.search.group()
regex_obj.search.groups()

# find all results and replace
match = regex_obj.findall(Text)

# censoring data
Censored_Text = regex_obj.sub(r'CENSORED', Text)
```

string_to_match:

string	specific string
(\d\D{1})(\w{2}\W)	
(...)	grouping
[]	create a character class
+	match one or more characters
*	match zero or more characters
?	match zero or one character
.	match any character except for newline
^string	start with
string&	end with

Green thread process

Multi-threading (aka "Green thread process") is used in IO applications.

```
import threading

def fun_name(...):
    ...

# thread 1
t1 = threading.Thread(target=fun_name)
t1.deamon = True
t1.start()

# thread 2
t2 = threading.Thread(target=fun_name)
t2.deamon = True
t2.start()
```

Multiprocessing

This method is the proper multi-threading.

```
import numpy as np
from numba import njit
from concurrent.futures import ThreadPoolExecutor

# function definition
def fun_name(...):
    ...

# multi-process function
multi_fun = njit(fun_name, nogil=True)


# process 1
with ThreadPoolExecutor(8) as ex:
    ex.map(multi_fun, range(100))

# process 2
with ThreadPoolExecutor(8) as ex:
    ex.map(multi_fun, range(10))
```

Modules Management

Virtual Environment

`python3 -m venv ws_name`	create virtual environment
`source ws_name/bin/activate`	activate virtual environment
`deactivate`	deactivate virtual environment

Package Installer

Basic commands

`pip3 list`	list of installed packages
`pip3 install`	install packages
`pip3 unistall`	unistall packages
`pip3 show`	info about package
`pip3 check`	verify dependencies
`pip3 search`	search PyPI for packages

Python dependancies

`pip3 freeze > deps.txt`	create list of dependencies
`pip3 install -r deps.txt`	install dependencies from list
`pip3 uninstall -r deps.txt`	unistall dependencies from list

Object Oriented Programming

Introduction

Principles

- Abstraction
- Encapsulation
- Inheritance
- Polimorphism

Rules

- S
- O
- L
- I
- D

Class

```
class ClassName:

    # Constructor
    def __init__(self, args):
        self.field = value

    # Method definition
    def method(self, args):
        ...


# Child class (Inheritance)
class ChildClassName(ClassName):

    # Static attribute (class attributes definition)
    class_attribute = value

    def __init__(self, args):
        super().__init__(self, args)
        self.field = value

        # modify class attribute
        ClassName.class_attribute = value

        # call class method
        ClassName.classmethod()

    # Static method (class method definition)
    @classmethod
    def class_method(cls):
        cls.class_attribute = value
```

Data Plotting

```
import matplotlib.pyplot as plt
```

Plot

```
fig, axes = plt.subplots(figsize=(12,6))

axes.plot(
    x, y,
    color='value',
    linewidth=value,
    marker='value',
    markersize=value,
    label='string'
)

plt.show()
```

Subplot

```
plot_obj = plt.subplots(nrows=nx, ncols=ny, figsize=(12,6))
fig, (axes_0, ..., axes_n) = plot_obj

# subplot 0
axes_0.plot(
    x0, y0, ...
)

...

# subplot n
axes_n.plot(
    xn, yn, ...
)

plt.show()
```

Scatter

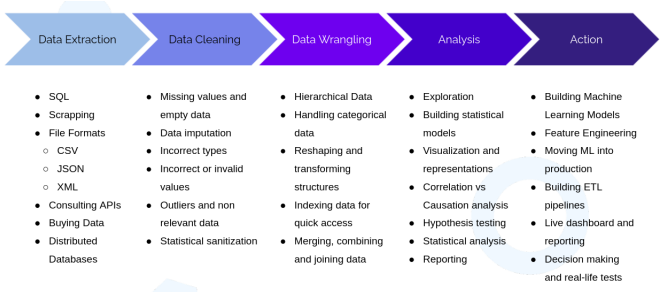
Bar

Histogram

Kernel density estimator

Data Analysis

Introduction



Useful libraries:

- pandas
- numpy
- matplotlib
- scipy

Read/Write files

```
df = pd.read_csv(csv_file_name)
df.to_csv('file_name.csv')
```

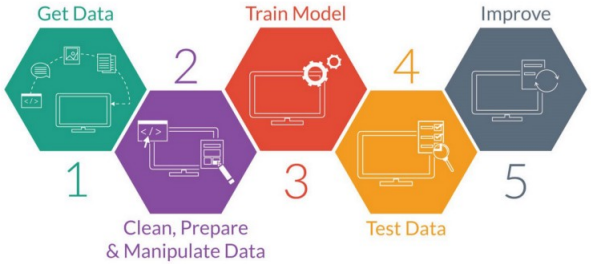
DataFrame

```
# Create dataframe
df = pd.DataFrame({
    'col_1': value.index,
    'col_2': value,
    ...
})

# Analysis
df.info()            # info about dataset
df.description()     # statistical info about dataset
df.columns()         # dataframe columns

# Indexing
x = df.column_title                 # single column tar
x = df[['column_title_1'], ...]     # multiple columns
```

Machine Learning

Introduction

Machine Learning (ML) is the set of practices that allow to obtain a prediction model from a set of input/output (dataset), in a process called training. Dataset is a collection of entries with a number of fields (features) that describe the data. The dataset is often divided in two pieces: training dataset and validation dataset.



Steps

1. Formulating the problem
2. Framing the problem: identify which type of ML approaches (one or more) fits best the problem
3. Design the dataset
4. Training
5. Validating the model

Problem Framing

- Supervised Learning: if data are labeled, so the output is a feature of the dataset; the goal is finding the relation between the features and the labels.
  - Classification: pick one of the N labels (binary, multi-class)
    - Logistic Regression
    - SVM (Support Vector Machine)
    - ANN
    - Decision Trees
    - Random Forest
  - Regression: predict numerical values
    - Linear Regression
    - Polynomial Regression
  - Deep Learning:
    - CNN
    - RNN
- Unsupervised Learning: if data are not labeled, so the output is not in the dataset; the goal is to identify meaningful patterns in the data and build its own labels (clusters).
  - Clustering: group similar data (K-Means, SVD, PCA)
  - Association: infer likely association patterns in data
  - Dimensionality Reduction
  - Anomaly Detection
- Reinforcement Learning: the model (agent) is set up and receives reward (reward function) when it performs the specified task.
  - Dynamic Programming
  - Monte Carlo methods

Overfiting/Underfitting

Overfitting occurs when features are too much and the model obtained track very well the training data, which can be affected by noise, but does not pass validation. Underfitting occurs when features are not enough to fit a proper model, then is poor both in traingn and in validation.

Validation

Computer Vision

OpenCV

Input/Output

```
import cv2

# Read
image_obj = cv2.imread(image_file.format, options)
'''
options:
- cv2.IMREAD_COLOR: read colors
- cv2.IMREAD_GRAYSCALE: read gray-scale
'''
h, w = image.shape[:2]    # image height and width

# Show
cv2.imshow('image_out_name', image_obj)
cv2.waitKey(option)
'''
options: if empty it close at key pressing, if not
↪  indicate the value in ms after which the picture
↪  is closed
'''

# Write
cv2.imwrite('image_file.format', image_obj)
```

Manipulation

```
# Set colors
            # set color
            imag_obj[i, j] = color

            # read color
            (b, g, r) = image_obj[i, j]


# Cut
cut_image = image_obj[i, j]


# Resize
            # resize by width
            new_w = 300
            resize_ratio = new_w/w
            dim = (new_w, int(h*resize_ratio))
            resized_image = cv2.resize(image_obg,
            ↪  dim, interpolation=cv2.INTER_AREA)

            # resize by percentage
            resized_image = cv2.resize(image, (0, 0),
            ↪  fx=perc_x, fy=perc_y,
            ↪  interpolation=cv2.INTER_AREA)


# Reflection
flipped_x = cv2.flip(image_obj)
'''
option:
- 0: reflect around vertical axis
- 1: reflect around horizontal axis
'''


# Pudding
pud_image = cv2.copyMakeBorder(image_obj, top_aug,
↪  bottom_aug, sx_aug, dx_aug, option)
'''
- cv2.BORDER_REPLICATE: the pixel at borders are
↪  replicated
- cv2.BORDER_REFLEECT: the image is reflected at
↪  borders
- cv2.REFLECT_101: the image is reflected at borders
↪  without borders
- cv2.BORDER_WRAP: replicate image at borders
- cv2.BORDER_CONSTANT, value=color: fill with color
'''
```

Drawing

```
# Creating
    # gray-scale image
    image_obj = np.zeros((x_size, y_size),
    ↪  dtype='data_type')

    # color image
    image_obj = np.zeros((x_size, y_size, 3),
    ↪  dtype='data_type')


# Figures
    # Line
    cv2.line(image_obj, init_point, fin_point, color,
    ↪  thickness=thick_value,
    ↪  lineType=cv2.line_type)

    # Rectangle
    cv2.rectangle(image_obj, init_point, fin_point,
    ↪  color, thickness=thick_value,
    ↪  lineType=cv2.line_type)

    # Circle
    cv2.circle(image_obj, (center_x, center_y),
    ↪  radius, color, thickness=thick_value,
    ↪  lineType=cv2.line_type)
'''
options:
- thick\_value: value of thickness, if set to -1 it
↪  means solid
- line\_type:
- cv2.LINE_AA: line AA
'''
```

Transformations

```
# Translation
T = np.float32([[1, 0, x], [0, 1, y]])
image_translated = cv2.warpAffine(image_obj, T, (w,
↪  h))

# Rotation
# custom rotation
R = cv2.getRotationMatrix2D((x_center, y_center),
↪  angle, 1.0)
image_rotated = cv2.warpAffine(image_obj, R, (w, h))

# quarter rotation
image_rotated = cv2.rotate(image_obj, option)
'''
option:
- cv2.ROTATE_90_CLOCKWISE: -90°
- cv2.ROTATE_90_COUNTERCLOCKWISE: +90°
- cv2.ROTATE_180_CLOCKWISE: -180°
- cv2.ROTATE_90_COUNTERCLOCKWISE: +180°
'''
```