 **Python intro notes**  (hopefully helpful for people coming from **R** )

[Use freely, cite me](https://creativecommons.org/licenses/by/4.0/)

Notes from [openHPI pythonjunior2020](https://open.hpi.de/courses/pythonjunior2020) , [Python for Digital Health](https://hpi.de/studium/im-studium/lehrveranstaltungen/digital-health-ma/lehrveranstaltung/wise-20-21-3110-introduction-data-science-and-python-for-digital-health.html) and personal learning. RefCards [search](https://www.ecosia.org/search?q=Python+Refcard)

By Berry Boessenkool, [berry-b@gmx.de](mailto:berry-b@gmx.de), [brry.github.io](https://brry.github.io/), Jan-Mar 2021 RefCard [example](https://michaelgoerz.net/refcards/python_refcard.pdf)

!!! means this will fry your brain if you're used to R. Especially subsetting with positions will be horrible.

Most python interpreters don't print unless excplicitely stated. print() is mostly left away here for brevity.

[Download & install](https://www.python.org/downloads/), hints for [Windows users](https://docs.python.org/3.9/using/windows.html). [tutorial](https://docs.python.org/3.9/tutorial/index.html), standard [libraries](https://docs.python.org/3/library/index.html), language [reference](https://docs.python.org/3/reference/index.html#reference-index), [documentation](https://docs.python.org/3.9/index.html).

**IDEs**

|  |  |
| --- | --- |
| [PyCharm](https://www.jetbrains.com/pycharm/download) | Good for scientific development, but slow in startup |
| [VScode](https://code.visualstudio.com/Download) | (Visual studio code) increasingly popular, supports multiple languages, e.g. R |
| IDLE | Installed by default, not suitable for large projects |
| More: | [www.programiz.com/python-programming/ide](http://www.programiz.com/python-programming/ide) , [colab.research.google.com](https://colab.research.google.com/notebooks/intro.ipynb), [Jupyter Manual](https://jupyter.brynmawr.edu/services/public/dblank/Jupyter%20Notebook%20Users%20Manual.ipynb) |

**Syntax**

function(arg, "txt", 'single quotes', 77.86) # comment

""" multi-line comment

with line breaks """

7\*6 ; 21+21 # semicolon possible, but not good practice.  *here for effective space use*

9 // 2 ; 20 % 7 ; 3 \*\* 2 # ≈ 9 %/% 2 ; 20 %% 7 ; 3^2 in R (int.div, modulo)

a = 5 ; a += 1 # short for a = a + 1 ; a \*= a+2 # short for a = a \* (a+2)

variable\_name = "value" # naming convention: lowercase, underscore

NameError: non-existing objects - [List of errors](https://docs.python.org/3/library/exceptions.html)

Variable names cannot start with numbers, Python is case sensitive

Reserved statements ([keywords](https://www.w3schools.com/python/python_ref_keywords.asp)) like else cannot be used as variable name

SyntaxError: often forgotten brackets or colons (e.g. in loops)

Method = function for an object class, e.g. listobject.append

Linter: program to analyze code style and determine structural problems (pointless lines of code, potentially overwriting variable names, etc)

**Collections (Arrays)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **type** | **example** | **changeable** | **ordered** | **indexed** | **notes** |
| list | [1,3] | yes | yes | yes | - |
| tuple | (1,2) | no | yes | yes | - |
| set | {1,4} | no, but add | no | no | no duplicates |
| dictionary | {"a":7, "b":3} | yes | no\* | by key | no duplicates |

**Data types**

integer, float, string, boolean (True/False), complex (2+1j) !!!

isinstance(7.5, int) # check for class

isinstance("Hello", (float, int, str, list, dict, tuple)) # one of types

value = input("Enter number: ") # interactive input ≈ readline("Enter num: ")

print("Type is: ", type(value)) ; type(int(value)) ; type(float(value))

value + 7 # ValueError if keyboard input was charstring

value = float(input("Give a number: ")) # read keyboard input and convert

**Lists**

list = [7, -4, 9, 1, 2, 3, 9] ; len(list)

list[0] # first element !!! ; list[1] # second element !!!

list[-1] # last element

list[-2] # second-to-last element list[5:-2] # range from left + right

list[2] = "newvalue" # overwrite third element, mixed data types possible !!!

list[2:5] # elements 3,4,5 exclusive at the right end !!!

list[4:] ; list[:6] # slicing: fifth till last element ; first till sixth

: is not an operator outside subsetting !!! list = [] # empty list

bar = list

list.append(66) # mutable object: changed even without re-assigning !!!

id(list) == id(bar) # both with 7,-4, "new value",1,2,3,9,66 !!!

list.pop() # remove last element (+ return it invisibly)

list.pop(index) # remove (+ return) selected element

del(list[index]) # only remove element at given index

list.insert(5, "new\_val") # insert at given position

list.index(9) # location of 9 ; list.remove(9) # remove first instance of 9

9 not in list # check for non-presence, returns a boolean ≈ ! 9 %in% vec in R

list.reverse() ; list.sort() ; list.sort(reverse=True)

list = [1,2,3,[31,32,33],4] # Nesting possible

list\_with\_charstrings[3][7] # eighth letter in fourth element

one\_list.extend(another\_list) # ≈ one\_list <- c(one\_list, another\_list) in R

one\_list + another\_list # ≈ c(one\_list, another\_list) in R

one\_list \* 2 # ≈ rep(one\_list, 2) in R

**Dictionaries**

\*: since python version 3.6/3.7, dictionaries *are* ordered

dict = {'name': "Berry", 'age': 31} # keys (name, age) must be unique

len(dict)

dict ['name'] = "new\_value" # key + value = 'pair' dict['new\_key'] = 42

f"Hi, {dict['name']}" # fstring double and single quotes cannot be mixed

print("Hello, {name}" .format(name=dict['name']) )

dict.get('NAME', "value\_if\_key\_not\_present")

del(dict['age']) # delete pair (entire entry). del d['k'] brackets optional!!

dict.keys() ; dict.values() ; dict.items()

list(dict.items()) # -> list with tuples -> very high memory usage!

the\_age = dict.pop('age') # KeyError: key no longer in dict

other\_dict = dict.copy() ; dict.clear() ; dict.update(another\_dictionary)

**Sets**

s1 = {1,2,3,4,5} ; s2 = {3,4,5,6,7,8} ; {} empty dict ; set() empty set

s1 | s2 ; s1 & s2 ; s1 - s2 # union ; intersection ; difference

**Charstrings**

"Hey" + "You there" # + operator to concatenate (chain) strings

3\*"Hi" # -> "HiHiHi" # \* operator to repeat strings

len("char string") # ≈ nchar in R. Not the same as len(some\_list) !!!

print("Hey", "You there", sep=" ", end="--\n—")

charstring = "Hi this is a text. with words"

"this" in charstring # ≈ grepl("this", charstring) in R

charstring[0] # ≈ substr(cs, 1,1) in R. Not the same as some\_list[0] !!!

~~charstring[1] = "b"~~ # not possible: unlike lists, strings are immutable

charstring[5:-2] # subset region

charstring[300] # IndexError: subsetting outside of existing range

charstring.split() # split at spaces. immutable - does not change object !!!

charstring.split(".") # split at periods. The default includes \n as space.

"\_".join(["list", "of", "words"]) # ≈ paste(wordvec, collapse="\_") in R

" char string ".strip() # strip white space (or given symbols) on both sides

"CharString".lower() # ≈ tolower("CharString") in R

"CharString".startswith("Ch") # ≈ startsWith("CharString", "Ch") in R

"CharString".count("r") ; max("CharString")

"CharString".find("tri") # gregexpr("tri", "CharString") in R

"Chars".replace("Ch", "K") # ≈ gsub("Ch", "K", "Chars", fixed=TRUE) in R

re module for regular expressions aka. wildcards (see section Packages):

import re ; re.sub('[xyz]', 'K', "abycd") # ≈ gsub("[xyz]", "K", "abycd")

F-string placeholder (since Python 3.6). Inline arithmetics posible:

person="Berry" ; f"{person} is a nice guy with {5+5} fingers"

print("%d %s cost $%.2f" % (6, "bananas", 1.74)) # -> 6 bananas cost $1.74

print("{0} {1} cost ${2}" .format(6, "bananas", 1.74))

**Packages**

[pip](https://pip.pypa.io/en/stable/) to install packages e.g from [pypi.org](https://pypi.org) (PYthon Package Index) ≈ CRAN for R. pip install pandas

[Anaconda](https://www.anaconda.com/products/individual) to install binary packages (also R) from their [cloud](https://anaconda.org/anaconda/repo). Anaconda Prompt: conda install pandas ; conda list

Popular packages: Data science: pandas, numpy, Machine learning: tensorflow, pytorch, Statistical analysis: scipy, Web application: django, Plotting: matplotlib, seaborn , package version management: virtualenv

ImportError: wrong library/module/script name, non-existing objects

from library import \* # all functions -> bad practice: object origin unclear

from library import function1, function2 # specific function(s)

import library then you can use library.function(…)

import library as lib then you can use lib.function(…)

from random import random, randint # random,math,etc come with python

random() # float between 0 (inclusive) and 1 (exclusive!!!), ≈ runif() in R

randint(1, 6) # int between start and end, including these

import os # os is a module in the standard library, no installation needed

print(os.getcwd()) # ≈ getwd() in base R ; os.chdir() # ≈ setwd()

from math import pi

**Read files**

If at os.getcwd(), there is *mydataset.py* with age = 45, we can use:

from mydataset import age # to then use age + 2

from mydataset import \* # to import all ≈ source("mydataset.py") in R

import mydataset # to then use mydataset.age + 2

print(dir(mydataset)) # list the objects in the module

import os, sys ; fname = os.path.join(sys.path[0],"file.txt") # for wd

with open(fname) as f: # with closes the connection (even in case of error)

content = f.read()#.splitlines() # ≈ readLines("file.txt") in R

**Logicals**

< ; <= ; > ; >= ; == ; != ; and ; or ; not # comparison / logical operators

7 < 8 ; "9" < "A" ; "A" < "B" ; "A" < "a" ; "a" < "b" !!order in R: "a" < "A"

7>1 & 6>1 in R, Python needs: (7>1) & (6>1), Py reads 7 > 1&6 > 1 and 1&6=0

|  |  |
| --- | --- |
| **Conditional code execution** | **Loops** |
| IndentationError: wrong number of spaces at the beginning of a line | |
| if cond:  do(1)  do(2)  else:  do(3)  if cond1:  do(1)  elif cond2:  do(2)  else:  do(3)  if cond1 and (cond2 or cond3):  print("stuff") | for number in (0,1,2,3): # or in range(4)  print(number) # range(8, 0, -2)  # range stop exclusive!!!  # convention for unused index variable:  for \_ in range(8): # or \_var  print("stuff")  for a,b in ( (1,4), (5,7), (6,9) ):  print(f"a={a}, b={b}, a+b={a+b}")  while cond:  run\_things()  if(cond2):  break  # continue ≈ next in R  enumerate("hello") ; iter # iterators |

result = []

for item in item\_list:

new\_item = do\_something\_with(item)

result.append(new\_item)

result = [do\_something\_with(item) for item in item\_list] # list comprehension

out = [] for word in charstring\_list if word[0] == "B": out.append(word)

out = [x for x in charstring\_list if x[0] == "B"]

≈ char\_vec[substr(char\_vec,1,1)=="B"] in R # not vectorizable in Python !!!

**Write custom functions**

def greet(name, time="morning"): # name+time are parameters

return f"Hello {name}! Good {time}." # return exits function execution

# explicit return is needed !!! else a function returns None  (≈ NULL in R)

greet("Berry") # Berry+evening are arguments

greet("Berry", "evening") # parameter=argument ≈ argument=value in R

def change\_object():

global ab

ab = 2

ab = 1 ; ab ; change\_object() ; ab # is now 2

multiply = lambda x,y: x\*y # single expression function on one line of code

multiply(7, 3)

**Multiple assignment**

def myfun(x, y): # related: swap two variables: a, b = b, a

return x\*2, y\*2

a, b = myfun(3, 4) # two int objects, each with a single value

c = myfun(3, 4) # tuple object with (6, 8)

list(map(len, ["abcdef","ab","abc"])) # sapply(c("abcdef","ab","abc"), nchar)

**Error management**

import traceback

try:

7 + "2" # code that might fail. int("seven") would give ValueError

except TypeError: # TypeError: wrong data type for operator or function

print("That mixed charstrings and numbers")

except Exception: # print instead of error

print("another error occured: ", traceback.format\_exc() )

else:

do("stuff")

**Write custom class**

class Person:

pass # Placeholder for future code. A class body may not be empty.

p1 = Person() # create object instance

p1.name = "Berry" ; p1.age = 31 # add attributes

class Person: # class attributes, generate w/ constructor

def \_\_init\_\_(self, name, age): # initialize (assign values) to data members

self.name = name # of the object when Person() is called

self.age = age

if name=="forbidden":

raise Exception("Name cannot be 'forbidden'") # ≈ stop("msg") in R)

def can\_watch\_movie(self): # class methods

if self.age >= 18:

return "Sure, watch it" # self represents object of class Person,

else: # always first arg to \_\_init\_\_

return "Too young, sorry"

p2 = Person("John", 25) ; p2.name ; p2.can\_watch\_movie() # instantiation

p2.\_\_dict\_\_ # dictionary of all given parameters and arguments

p2 = Person("forbidden", 25) ;

**turtle**

package to draw figures on plot range -200:200

forward(nsteps), right(degrees), goto(x,y), penup(), pendown(), shape("turtle"), register\_shape(), pencolor("yellow"), bgcolor(), fillcolor(), begin\_fill(), end\_fill()

**Count table**

colors = ['red', 'blue', 'blue', 'yellow', 'blue', 'red', 'green']

import collections

collections.Counter(colors).most\_common(6) ≈ sort(table(colors))[1:6] in R

**Numpy**: computationally efficient numerical arrays. pip install numpy

import numpy as np ;

np.array([1,2,3,4,5,6]) # 1D array. type: numpy.ndarray

ar = np.array([[1,2,3,4], [5,6,7,8]]) # 2D array from list of lists

np.random.randint(10, size=(3,4)) # random integers 0-9, 3 rows, 4 columns

Attributes: accessed without brackets (methods with brackets):

ar.ndim ; ar.shape ; ar.size # ≈ length(dim(ar)); dim(ar); length(ar) in R

ar.dtype # numpy-internal data type ar.itemsize ; ar.nbytes # in bytes

ar1 = np.arange(10) # sequential 1D array

ar1[4] ; ar1[-1] # fifth and last element of 1D array

ar1[start:stop:step] # general subsetting of 1D arrays

ar1[:5] ; ar1[4:] ; ar1[4:7] ; ar1[::2] / ar1[1::3] # every other element

ar1[::-1] # all elements, reversed. Works for lists & charstrings as well

ar[:2, :3] ; ar[:, 5] # 2 rows, 3 columns. all values in sixth column

ar[0] # first row, not first column !!!# as in R, not recommended!

ar[2,0] = 3.1415 # change single element at row three, column 1 of 2D array

# If array is integer, float is silently truncated to 3: Downcasting !!!

ar\_sub = ar[:2, :2] ; ar\_sub[0,0] = 99 # **changes both** ar\_sub and ar !!!

ar\_sub = ar[:2, :2].copy() ; ar\_sub[0,0] = 42 # does not change ar

ar = np.arange(1,10) ; grid = ar.reshape((3,3)) # 1D array to 2D array

ar[np.newaxis, :] # 2D, 1 row. Both do **not** change ar.

np.concatenate((ar1, [67,68,69]))

np.concatenate([grid, grid]) # ≈ rbind(grid, grid) in R

np.concatenate([grid, grid], axis=1) # ≈ cbind(grid, grid) in R

np.vstack(); np.hstack(); np.dstack() # the same, d for depth (3rd dimension)

s1,s2,s3 = np.split([1,2,3,4,5,6,7,8,9], [3,5]) ; np.hsplit; np.vsplit for 2D

Ufunc (Universal functions operating on full array)

%timeit compute\_long\_thing(big\_array). # %timeit by Ipython

Numpy enables very fast vectorized operations: 1.0/ar ; ar1/ar2 ; ar>=3.

ar + 5 # element-wise operation: broadcasting (≈ recycling in R)

np.ones((3,4)); np.zeros() # arrays full of 1 (or 0)

ar3x3 + ar3 -> ar3x3 # ar3 (1D)repeated for each row of ar3x3 (2D).

ar3 + ar3x1 -> ar3x3.

angles = np.linspace(0, np.pi, 3) # ≈ seq(0, pi, len=3) in R

np.sin(); np.exp(); np.log10(); np.power()

np.count\_nonzero(ar<6) or np.sum(ar<6) # ≈ sum(ar<6) in R

np.sum(ar<6, axis=1) # ≈ rowSums(ar<6) or apply(ar, 1, sum) in R

np.any() ; np.all() # can be called without np. as well

np.any(ar<6, axis=0) # columns ≈ apply(ar, 2, any) in R -> axis != MARGIN !!!

np.any(ar<6, axis=1) # rows ≈ apply(ar, 1, any) in R

ar[ar < 6] # reduces dimension (e.g. 2D to 1D)

ar.mean(); ar.mean(axis=0);ar.mean(axis=1) #≈ mean(ar); colMeans(ar);rowMeans

np. corrcoef(ar1, ar2) ; np.isfinite() ; np.isnan() ; np.asarray() ;

np.nanstd(ar) # ≈ sd(ar, na.rm=TRUE) in R ; np.nanmean(ar) ; np.nanmedian(ar)

np.sort(ar)

np.random.poisson(5, 100) # 100 random numbes from poisson distribution

np.random.normal(mu, sigma, 100)

**Pandas**: panel data analysis, builds on numpy. [API docs](https://pandas.pydata.org/pandas-docs/stable/reference/index.html). pip install pandas

Series (column) with axis labels and DataFrame of Series

import pandas as pd ;

pd.Series(data=list\_of\_vals, index=list\_of\_strings) # data can be numpy array

s1 = pd.Series(dictionary) ; s1.to\_list() ; s1.to\_dict() ; s1.size ;

s1 = pd.Series([1,2,3,4], index=["A","C","D","E"])

s2 = pd.Series([1,2,5,4], index=["A","B","C","E"]) ; s1["A"] # subset by name

s1 + s2 # returns: A:2, B:NaN, C:7, D:NaN, E:8 # Operations per index

df = pd.DataFrame(randn(5,4), index='A B C D E'.split(), # ≈ rownames in R

columns='W X Y Z'.split()) # ≈ colnames in R

df.shape; df.index ; df.columns ; df.values ; df.index.values ;

df.info() ≈ str(df) in R; df.dtypes

df.select\_dtypes(include='number') # see [dtypes](https://numpy.org/doc/stable/reference/arrays.scalars.html) # does not change df

|  |  |  |
| --- | --- | --- |
| Select columns | Select rows | Select elements |
| df["colname"]  df.colname  df[["col1","col2"]]  df.iloc[:, -1] # last C  df.iloc[:, 1:3] # C 2+3 | df.loc["rowname"]  df.iloc[2]  df.iloc[0:3] # first 3 r df.iloc[-1] # last row  df[ df.colname < 15 ] | df.iloc[2, 5]  df.loc["rname", "cname"]  df.colname[3:5]  iloc for index location |

Missing values

df1 = pd.DataFrame({'A':[1,2,np.nan],

'B':[5,None,np.nan],

'C':[1,2,3]})

df1.isna() # ≈ is.na(df) in R

df1.isna().sum() # ≈ apply(df, 2, is.na) in R # number of Nas per column

df1.isna().sum(axis=1) # number of Nas per row

df1[df1["B"].notna()] # ≈ df[!is.na(df$B), ] in R

df1.dropna() # ≈ na.omit(df) in R # see also df1.dropna(axis=1)

df1.dropna(thresh=2) # at least 2 finite numbers needed to be kept

df1.fillna(value='missing') # replace NA with "missing" # value=0 possible

df1.A.fillna(value=df1.A.mean()) # Replace with mean value of column

df1[df1.A.isna()].index.tolist() # ≈ rownames(df)[is.na(df$A)] in R

rows\_with\_nan = [index for index,row in df1.iterrows() if row.isna().any()]

df1.index[df1.isna().sum(axis=1) > 0].tolist() # the same, more readable

Combining dataframes

df.groupby('Age\_group').mean() # .min() ; .count() # mean only for numerics

pd.merge(df1, df2, on='key\_column', how="outer") # on=['key1','key2']

how: outer, inner, left, right # ≈ merge( all=T) all=F, all.x=T, all.y=T in R

df1.join(df2, how="outer") # cbind by rownames

pd.concat([df1, df2], axis=0) # outer by default. ≈ rbind in R, but expands

pd.concat([df1, df2], axis=1) # inner by default. ≈ cbind in R, but expands

Pandas misc

df.col.unique() # .nunique() ; df.col.value\_counts() # ≈ table(df$col) in R

df = df.assign(new\_col = lambda x: (x.col\*1000)) # df["new\_col"]= df.col\*1000

df.apply(lambda x : x/100, axis=1) ≈ apply(df, 1, function(x) x/100) in R

(df.colA > 6) & df.colB # &, |, !=, ==, ~ (not), >, <, >=, <=

df = df.sort\_values(by='colname') # ≈ df = df[order(df$colname)] in R

df.sort\_values(by='colname', inplace=True) # modify df directly

dfcp = df ; dfcp[5,2] = 42 # changes df as well, dfcp is only a pointer to df

dfcp = df.copy() # as usual :)

df.pivot\_table(index=['c1','c2'], columns=['c3'])

pd.crosstab(df.c1, df.c2); pd.crosstab(index=x, columns="Count") # ≈ table(x)

pd.read\_csv ; pd.read\_excel ; pd.read\_html ; df.to\_csv() ; pd.to\_exel()

df.describe() ≈ summary(df) in R ; df.head() ; df.tail()

**Statistics**

import pandas as pd ; import numpy as np ; import scipy ; import statistics

statistics.mean(x)

df.mean() # pandas.mean excludes nan by default ; df.median()

np.nanmedian() safer than statistics.median() with nans

statistics.quantiles(x, n=4) # in Python >3.8

df.quantile([0, 0.05, .25, .5, .75, .95, 1]) ; scipy.stats.iqr(x)

statistics.stdev(x) ; df.std() ; np.var(x, ddof=1) ; df.var()

statistics.mode(x) ; statistics.multimode(x) # Py>3.8 ; scipy.stats.mode(x)

scipy.stats.skew(x) ; df.skew(axis=0, skipna=True)

scipy.stats.kstest(x, 'norm') # Kolmogorov-Smirnov test for normality

scipy.stats.shapiro(x)[1] # Shapiro-Wilk test for normality

corcoef,pvalue = scipy.stats.pearsonr(x, y) ; df.corr(method="pearson")

scipy.stats.ttest\_1samp(x, popmean=182) # "is mean of x = 182?"

scipy.stats.ks\_2samp(x,y).pvalue # to answer "is x different from y?"

scipy.stats.ttest\_ind(x,y) # independent T-test "is x diff from y?"

scipy.stats.ttest\_rel(x,y) # [paired](https://en.wikipedia.org/wiki/Student%27s_t-test#Independent_(unpaired)_samples) T-test, when x and y related

scipy.stats.mannwhitneyu(x,y, alternative="greater") # one-sided Mann-Whitney-U Wilcoxon Rank test (≈ T-test for non-normal distribution shape)

scipy.stats.chi2\_contingency([x,y]) # categories. can take pd.crosstab output

scipy.stats.chisquare(f\_obs=observed, f\_exp=expected) # Goodness of fit test

scipy.stats.f\_oneway(x,y,z) # ANOVA "are x, y and z the same?"

mod=statsmodels.formula.api.ols('y ~ C(x1)+C(x2)+C(x1):C(x2)', data=df).fit()

statsmodels.api.stats.anova\_lm(mod, typ=2) # kind of ≈ lm(y~x1+x2+x1:x2) in R

**Data visualisation with matplotlib**

import matplotlib.pyplot as plt

%matplotlib inline # in notebook ; plt.show() in last line in other editors

plt.scatter(x,y) ; plt.hist(x) ; plt.boxplot(data, vert=True)

plt.plot(x, y, 'r--') # 'r--': red dashed line ; 'g\*-': green stars + line

plt.xlabel('X axis title') ; plt.title('Plot title') ;

plt.savefig("filename.png", dpi=200) # save to disc as png, pdf, svg, etc

fig = plt.figure() ; ax = fig.add\_subplot(1,1,1) # object-oriented API

ax.plot(x, x\*\*3, label="x\*\*3", linewidth=3, color="blue", alpha=0.5)

ax.plot(x, x\*\*2, label="x\*\*2", linestyle="-.", marker="s")

ax.legend(loc='lower right') (ax is an axes, i.e. a figure window)

Multipanel plots

plt.subplot(1,2,1) # 1 is figure number. ≈ par(mfrow=c(1,2)) in R

plt.plot(x, y) ; plt.subplot(1,2,2) ; plt.plot(y, x)

fig = plt.figure(figsize=(8,4), dpi=100) # nested plots

window = fig.add\_axes([0.1,0.1, 0.8,0.8]) # bottomleft + proportion of canvas

window.plot(x, y) ; window.set\_ylabel("ylab") ; window.set\_xlim([0,20])

fig,ax = plt.subplots(1,2) # ≈ par(mfrow=1:2, mar=c(3,2,1,0.5)) in R

ax[0].plot(x,y) ; ax[1].plot(x,y);ax[1].set\_ylabel('y') ; plt.tight\_layout()

**Data visualisation with seaborn** (builds on matplotlib, nice with pandas df)

import seaborn as sns # histogram with kernel density estimate:

sns.displot(data=df, x='column', kde=True, bins=30) # distributional summary

sns.catplot(data=df, kind="swarm", x="catcol", y="numcol", hue="catcolumn")

kind="box"; kind="bar" # categorical data; swarmplot, boxplot, barplot

sns.catplot(…, palette="Set2") # mypal={cat1:"g", cat2:"b"} [color palettes](https://seaborn.pydata.org/tutorial/color_palettes.html)

sns.pairplot(data=df, kind="kde", diag\_kind="hist", hue="category",

corner=True, diag\_kws=dict(fill=False), …)

sns.set\_theme() # ≈ par(…) in R

sns.relplot() # relationship scatterplots

**Machine learning** - classification, regression, clustering, dimensionality reduction, model selection

pip install scikit-learn ; import sklearn # note different names for install / import

Data prep

y = df.target ; x = df.drop('target', axis=1)

x\_train, x\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(

x, y, test\_size=0.3, random\_state=12) # 70% for training, seed for shuffle

scaler = sklearn.preprocessing.StandardScaler() # see also: minmax\_scale

x\_train\_norm = scaler.fit\_transform(x\_train.values)

x\_test\_norm = scaler.transform(x\_test.values)

Multivariate linear regression

logreg = sklearn.linear\_model.LogisticRegression(max\_iter=1000)

logreg.fit(x\_train, y\_train) ; logreg\_pred = logreg.predict(x\_test)

pd.crosstab(y\_test, logreg\_pred) ; logreg.score(x\_test, y\_test) # accuracy

logreg.predict\_proba(x\_test) ; logreg.coef\_ ;

k-Nearest-Neighbors Classification: predict outcome by majority at k most similar data points

knn\_5 = sklearn.neighbors.KNeighborsClassifier(n\_neighbors=5) # set the model

knn\_5.fit(x\_train\_norm, y\_train) # train the model

knn\_5.predict(x\_test\_norm) ; knn\_5.score(x\_test\_norm, y\_test)

Decision trees & Random forests

Hyperparameter: how high can tree depth be? (too high -> overfittting)

dt = sklearn.tree.DecisionTreeClassifier(random\_state=2, max\_depth=3)

dt.fit(x\_train, y\_train) ; dt.score(x\_test, y\_test)

sklearn.tree.plot\_tree(dt, feature\_names=x\_train.columns, filled=True)

rf = sklearn.ensemble.RandomForestClassifier(random\_state=2)

rf.fit(x\_train, y\_train) ; rf.score(x\_test, y\_test)

Evaluation

y\_pred = rf.predict(x\_test)

sklearn.metrics.confusion\_matrix(y\_test, y\_pred) ;

sklearn.metrics.plot\_confusion\_matrix(rf, x\_test, y\_test) # normalize='true'

sklearn.metrics.recall\_score(y\_test, y\_pred) # TP/(TP+FN)

sklearn.metrics.precision\_score(y\_test, y\_pred) # TP/(TP+FP), [WikiLink](https://en.wikipedia.org/wiki/Precision_and_recall)

sklearn.metrics.plot\_precision\_recall\_curve(rf, x\_test, y\_test) # [doc](https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html)

rf.score(x\_test, y\_test) # 'regular' accuracy, good when labels are balanced

sklearn.metrics.balanced\_accuracy\_score(y\_test, y\_pred)

Unsupervised learning: cluster analysis & PCA - No target variable, goal is not to predict something

kmeans = sklearn.cluster.KMeans(n\_clusters=2, init='random', random\_state=3)

kmeans.fit(x\_train\_norm) ; kmeans.cluster\_centers\_ # n dimensions = n columns

pred\_k\_means\_test = kmeans.predict(x\_test\_norm)

sklearn.metrics.accuracy\_score(y\_test, pred\_k\_means\_test)

pca = sklearn.decomposition.PCA(n\_components=2) # number of target dimensions

pd.DataFrame(np.vstack([x\_train.columns, # component contributions

pca.components\_.round(2)]).transpose()) # (feature effects)

pc = pca.fit\_transform(x\_test\_norm)

sns.scatterplot(x=pc[:,0], y=pc[:,1], hue=pred\_k\_means\_test, palette="Blues")

hierarc\_clust = sklearn.cluster. hierarchy.linkage(x\_test\_norm, method='ward')

sklearn.cluster. hierarchy.dendrogram(hierarc\_clust)

agg\_clustering = sklearn.cluster. AgglomerativeClustering(n\_clusters=2,

affinity='euclidean', linkage='ward')

pred\_agg\_test = agg\_clustering.fit\_predict(x\_test\_norm)

sklearn.metrics.accuracy\_score(y\_test, pred\_agg\_test) # and sns pred\_agg\_test

X\_agg\_values = scaler.inverse\_transform(x\_test\_norm)

X\_agg = pd.DataFrame(X\_agg\_values, index=x\_test.index,

columns=x\_test.columns) ; X\_agg['clust'] = pred\_agg\_test

sns.pairplot(X\_agg, hue='clust', palette='Blues')