1. If x and y are 2 lists, then x < y is true if x[0] < y[0] and x[1] < y[1] ... x and y do not need to be of equal size.

2. Sort in descending order: sorted(reverse = True)

3. rst = sorted(Alist, key = functools.cmp\_to\_key(diff)) where diff is a function of 2 arguments x and y, both of which are elements in Alist. Note that diff(x, y) does not return True or False!: but some arbitrary difference function x - y. Based on this difference function, sorted() will put x before y if x - y > 0.

4. Binary search: bisect.bisect\_left(a, x, lo=0, hi=len(a), \*, key=None) is the same as std::lower\_bound(). i = bisect.bisect\_left() returns the first index of a such that a[i] >= x. key is the comparison function, e.g. lambda u: u or lambda u: u - 10.. You understand it..

1. Since I am using anaconda, so to install a new package, go to ipython console, do things like !conda install packagename

2. when using numpy, always consider the functions in that package, like numpy.maximum(), don't try to use the in-build function max()

3. type() is like str() in R

254. Don't use Python [[0.0] \* 6] \* 6. \* here will just be a reference..

255. Python: when you append an object to a list, the last element IS a reference to that object, thus mutating the last element in the list WILL change the object.

264. pandas data frame stores data column-wise. Each column is named "series", and is acting like a 1-d np array.

265. Get column names of a pandas data frame: dat.columns

265. Subset pandas data frame by integer indices: dat.iloc[integerIndices]. If integerIndices is an integer scaler, the return is an array, otherwise the return is a data frame.

Subset pandas data frame by index labels: dat.loc[indexLabels]

Subset pandas data frame by boolean array: dat[boolArray]

265. Python pandas trap: dat['colname'][j] will give you the element indexed by j, not the j\_th element. Always remember if you should have dropped the row indices: dat.reset\_index(drop = True, inplace = True) .   
  
266. pandas dataframe.iloc[subset.array, i] where subset is a boolean series. But for pandas series, you should do series.loc[subset.array] .. Annoying!  
  
267. pandas rename series: ema.rename('tmp', inplace = True), pandas rename data frame: tmp.columns = ['Price', 'EMA', 'EMAindi']. pandas join (left): pricesNorm.join(emaNorm, lsuffix = 'x', rsuffix = 'y').join(emaIndi, lsuffix = 'x', rsuffix = 'y'). Always add lsuffix and rsuffix to avoid errors, and then just rename the dame dataframe.  
  
268. pandas plot all columns in dataframe: tmp.plot().get\_figure().savefig("EMA.png")  
  
269. pandas series to dataframe: rsi = rsi.to\_frame()  
  
270. numpy concatenate(())! The argument is a tuple!  
  
271. pandas get the column numpy array: dat['name'].array

278. Save multiple Python objects just like save() and load() in R: upvoted answer at <https://stackoverflow.com/questions/20716812/saving-and-loading-multiple-objects-in-pickle-file>

279. Python cannot find files to import, then:

import sys  
sys.path.append('c:/Users/i56087/Desktop/gt/cs7643/assignment2/part1-convnet')  
sys.path.append('c:/Users/i56087/Desktop/gt/cs7643/assignment2/part1-convnet/tests')  
  
280. Flatten numpy array except for the first dimension: .reshape(x.shape[0], -1)

281. np.choose(): for each row of a matrix, retrieve an element given its index.

284. Multidimensional array aggregation, np.sum(X, axis = 0). If X.shape = (2, 5), then the result should reduce the 0th dimension, and you would end up with an array of shape (,5) .

288. Look into python object structure: type() and dir() are particularly useful for inspecting object's type and attributes.

289. Note that, after "for i in range(0, 10): do something", i will stay at 9, not 10!

299. pandas.series.isin(values)

300. Python multiprocess computing and time measure.

tmpN = int(1e7)  
tmpNjob = 30  
from joblib import Parallel, delayed  
tmp = [i for i in range(tmpNjob)]  
def f(x):   
    for i in range(int(tmpN)):  
        x += random.uniform(0, 1)  
    return x  
  
import time  
starttime = time.time()

# According to its document, only tmp is copied to every process.  
res = Parallel(n\_jobs = 13)(  
    delayed(f)(x) for x in tmp  
)  
timecost = time.time() - starttime  
print(timecost)  
  
starttime = time.time()  
tmprst = []  
for x in tmp:  
    s = 0  
    for i in range(int(tmpN)):  
        s += random.uniform(0, 1)  
    tmprst.append(s)  
timecost = time.time() - starttime  
print(timecost)

301: since now Python and R are fully integrated via reticulate package and even ctrl + Enter can let you execute just segments of python code. Python + R + C++ trinity is really established! Notice that:

The py object provides a means for interacting with the Python main session directly from R. Python objects accessed through py are automatically converted into R objects, and can be used with any other R functions as needed.

If you don't access python objects via reticulate::py$nameOfObject, then that object in the Python environment won't be copied!

302. The task manager showing low GPU usage does not mean actual low GPU usage. Try os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1" to test the multi-CPU speed. For training the Tsunami U-net, Enabling GPU yields 5~7x speedup against multi-CPU training.

303. keras.model.fit will separate the last validatoin\_split examples aside **before** shuffling, and the validation set will not change during training!

304: python datatable package is great. It seems to read csv file faster than R's data.table. Use datatable.options.nthreads = 1 to change the number of threads being used. The most memory efficient way to read in pandas is probably a oneliner: dt.fread('path').to\_pandas().

305: Assume datatable X, a = X["name"].to\_numpy() returns a reference vector to the column! You can set a[0] = 1e9 and X will correspondingly be changed. However, doing a = b will not affect X since a becomes a new object. The right way would be X["name"] = datt.Frame(b). Python datatable also store a frame in separate columns.

306: Better way to use dictionary for counting occurrences: C = Counter(); C.update({key: value}) ...

307: D = defaultdict(list) means a dictionary with list value type. This way, one can directly do D[key].append(value) without

308: Always set "shell = True" in subprocess. Fucking stupid library wastes my goddame time.

309: cloudpickle can be used to save functions with all kinds of dependencies. But it is not fast at saving data of types like dictionaries...

310:  Function that works exactly like the `unlist` in R:

def unlist(x, recursive = True):  
  result = []  
  if recursive:  
      for u in x:  
          if type(u) is list: result.extend(unlist(u))  
          else: result.append(u)  
  else:  
      for u in x:  
          if type(u) is list:   
              for v in u: result.append(v)  
          else: result.append(u)  
  return result

320: R: readlines("py/loadSave.py") = Python: open("py/loadSave.py").read().splitlines()

R: writeLines("py/loadSave.py") = Python: open("py/loadSave.py", 'w').writelines(alist) where alist is a list of strings. Change 'w' to 'a' to append the new lines.

321: Clear everything in Python environment:

import sys; sys.modules[\_\_name\_\_].\_\_dict\_\_.clear()

or just exit() and then Python

322: **eval**() only evaluate expression that has no assignment, e.g., eval("sum([8, 16, 32])"). Use **exec**("x = sum([1, 2, 3])") for the other purpose.

323: Check the memory address of numpy array elements: x[2].data. Check object address: id(object).

324: pandas dataseries.to\_numpy() will return a numpy array without copying, or you could just df['x'].array: this refers to the underlying array. If the feature is numeric, then the pandas array is equivalent to numpy array, otherwise it is a pandas array of objects, e.g., string.

325: x.ravel() is faster than x.flatten() since the former will not make a copy.

326: import from sibling directory or from any place in general:

import sys

sys.path.append("../")

from sibDir.script import functionName

327. Get a list of function arguments: inspect.signature(functionName).

328: plt(block = 0) will prevent the figure window blocking python's current execution.

329: tmp = plt.hist(data, bins = 100); tmp = plt.scatter(data).. for quick visualization.

330: Python's numpy's random number generator gamma is about 10x faster than R due to no vectorization in the latter. Python's scipy.stats also computes densities slightly faster than R. Use Python in the future baby.

331: numpy.random.choice(int(1e9), int(1e7), replace = False) is much slower than R's sample(1e9, 1e7). For sample k = 1e7 from n = 1e9, and for R, when n > 1e7 and k \* 2 <= n, R uses hashing.

332. equivalent to R's aggregate:

x = pd.DataFrame({  
  'id': ['A','A','A','B','B','C','C','C','C'],  
  'groupId': [11,35,46,11,26,25,39,50,55],  
  'type': [1,1,1,1,1,2,2,2,2],

  'w': [1,2,3,4,1,2,3,5,6]

  }).groupby(['id','type'])

By this time, x is a groupby object, acting like a dictionary. One can then perform things like:

x['groupId'].apply(np.asarray).tolist() which will return a list of numpy arrays.

x['w'].apply('max') will return the max for each group.

x['w'].apply(lambda x: np.sqrt(np.sum(x \*\* 2))) will return the norm for each group.

333. Equivalent to R's unique:

np.unique(), but this will return a sorted array, and is slower than R's unique.

pd.unique() will return an array that respects the original order, and it uses a hash method.

334. Order by multiple columns in python: numpy.lexsort(), pandas.sort\_values().

335. pandas.read\_csv(, engine = "pyarrow") is only a little slower than datatable, but consumes 4x memory. Read with datatable, and then convert it into pandas, which will make a copy, but worth it. Then remove the datatable object.

For writing csv from pandas, first do datatable.Frame(pandasDataFrame). Interestingly, the result is a reference to pandasDataFrame..

336. Sadly, there is no good alternative to R's match() in Python. One could consider sort + numpy.searchSort

337. numpy.array.shape = (4,) guarantees no copy of the array while flattening or reshaping it.

338. All objects in the current environment are in a dictionary globals().

339. If a pandas data frame is created from a numpy array, like Y = pd.DataFrame(X), then Y is a reference to X. Now Y.to\_numply() will also output a reference to Y.

If Y's element is changed to another datatype, e.g., from int to float, then its whole column is converted to float. But other columns are still reference to the original numpy matrix.

340. You CAN compare two dictionaries!

341. list comprehension on a list of strings is faster than list comprehension on a numpy array of strings, and is faster than np.vectorize on numpy array, and is also faster than R's paste0().. interesting.

342: yield is best used when you have a function **that returns a sequence** and you want to iterate over that sequence, but you do not need to have every value in memory at once. yield is to create **generator function**

343: Y = datatable.Frame(X): X will be copied on modification of Y.

344: Python list append is 3.3x as fast as R's list: L[[i]] = i \* 1.1

345: Do not underestimate the speed of Python's list comprehension. Speeds for appending vs. preallocation:

*import time  
st = time.time()  
tmp = [0.0 for i in range(int(3e7))]  
for i in range(int(3e7)):  
  tmp[i] = i \* 1.1  
  
time.time() - st # 3.8s.* ***And the speed is the same for pre-allocating None or 0***

*import time*

*st = time.time()*

*tmp = []*

*for i in range(int(3e7)):*  
 *tmp.append(i \* 1.1)  
time.time() - st # 9.7s*

**Looping through a list is 1.5x faster than looping through a numpy array! This is because list[] is native, but numpy array [] is not.**

346. Python custom compare:  
def f(x, y): return x – y  
sorted(a, key = cmp\_to\_key(f))