lazy data base API

Dependency:

csv build-in

os build-in

matplotlib scipy-package

mpl\_interaction github

Help Function:

str

int

None True False

dict

float

list

inilist(n, s): make a new list with n elements of value s return list of str

inilist(n, s): make a new list with n elements of value s return list of int

inilist(int, s): make a new list with n elements of value s return list of bool/None

pdict(d): print d in a pretty format

slist(s, sep=’,’): split s by sep return list of str

delete space arount sep

isnum(s): if s is int or float return True

else return False

valueof(s): if s is s of int/float/bool/None return eval(s)

elif s is "" return None

else return s

decomp1(s): split s by “=” then split by “,” return 2D-list of str

decomp2(s): split s by “,” then by “=” return 2D-list of str

decomp(s): auto split by “=” and “,” return 2D-list of str

Ex:

decomp(“a 0 , b 0 , c 0 = a 1 , b 1 , c 1”)= [['a 0', 'b 0', 'c 0'], ['a 1', 'b 1', 'c 1']]

decomp(“a 0 = a 1 , b 0 = b 1, c 0 = c1”)= [['a 0', 'b 0', 'c 0'], ['a 1', 'b 1', 'c 1']]

###############################################################

Table:

#class variable

coding = "utf-8" encoding type

spacing = None position of value

building-method:

next(self) return row in table

for row in self traverse the table row by row

len(self) return the length of the table

str(self) return a string representation of table

self represention of the table

self[i] get the i-th row

self[i][j] get the i-th row, j-th value

del self[i] delete the i-th row (i!=0)

del self[i][j] set the i-th row, j-th col value to None

self[i]= r set the i-th row to a new list (r is a list)

self[i][j]= v set the i-th row, j-th col value to v

r in self return True if r in self, else return False

reverse() return a back ward iterator

del self delete the self object

def \_\_init\_\_(self, table=None, name=None):

self.index = 0

self.name = name

self.colmap = {}

"need to do!"

# colmap should be used in saving!!!, and repr and str!!!

self.rowmap = {}

if table is None:

self.table = []

self.lenmap = []

elif isinstance(table, list):

self.table = []

for row in table:

self.table.append(Row(row, self))

self.lenmap = getlenmap(table)

# set colmap

top = table[0]

for i in range(len(top)):

self.colmap[top[i]] = i

# set rowmap

"need to do!"

self.rowmap = {}

else:

raise Exception("table should be 2D-list")

shift() make the v shift center to left or left to center

def p(self):

for v in self.table:

s = self.row2str(v)

command = input(s)

if command == "exit":

break

print(str(len(self.table) - 1) + " row(s)", end="")

def get(self, index, name):

# could remove

return self.table[index][self.colmap[name]]

def row(self, index):

return self.table[index]

def col(self, name):

return [self.row(i)[self.colmap[name]] for i in range(1, len(self.table))]

def read(name):

result = Table(name=name)

fname = name + ".csv"

file = open(fname, "r", encoding=Table.coding, newline='')

lines = csv.reader(file, delimiter=',', quotechar='"')

G = (row for row in lines)

A = next(G)

A = [v.strip() for v in A]

A = Row(A, result)

result.table.append(A)

result.lenmap = [len(v) for v in A]

for i in range(len(A)):

result.colmap[A[i]] = i

for row in G:

row = [v.strip() for v in row]

for i in range(len(row)):

result.lenmap[i] = max(result.lenmap[i], len(row[i]))

row[i] = valueof(row[i])

result.table.append(Row(row, result))

file.close()

print("READ <{}> FROM {}".format(result.name, fname))

return result

def save(self, name=None):

# ask for ensure!!!

if self.name is None and name is None:

raise Exception("give a name for the table to save")

elif self.name is None and isinstance(name, str):

self.name = name

fname = self.name + ".csv"

file = open(fname, "w", encoding=Table.coding, newline='')

lines = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE\_MINIMAL)

for row in self.table:

lines.writerow(row)

file.close()

print("SAVE <{}> TO {}".format(self.name, fname))

def apply(self, name, f):

l = []

for i in range(1, len(self.table)):

if f(self.get(i, name)):

l.append(self.table[i])

return Table(l)

def select(self, s):

# using string to input all fix!!!

l = slist(s)

result = [[]]

for v in l:

if v is None:

pass

elif not v in self.colmap:

print("No Attribute {} Auto Pass".format(v))

else:

result[0].append(v)

for i in range(1, len(self.table)):

sub = [self.get(i, key) for key in result[0]]

result.append(sub)

print("SELECT {} from <{}>".format(",".join(result[0]), self.name))

return Table(result)

def rename(self, s):

s = decomp(s)

self.\_checklist(s[0])

for i in range(len(s[0])):

self.colmap[s[1][i]] = self.colmap[s[0][i]]

del self.colmap[s[0][i]]

self.table[0][self.colmap[s[1][i]]] = s[1][i]

self.lenmap[self.colmap[s[1][i]]] = max(self.lenmap[self.colmap[s[1][i]]], len(s[1][i]))

print("RENAME {} TO {}".format(s[0][i], s[1][i]))

def \_\_del\_\_(self):

fname = ".csv"

if isinstance(self.name, str):

fname = self.name + fname

print(fname)

else:

raise Exception("Table do not have a name to delete")

print(fname)

if os.path.isfile(fname):

v = input("Are you sure to delete {}.csv ? Enter y for delete\n".format(self.name))

if v == "y":

print("start to remove")

os.remove(fname)

del self.table

del self.lenmap

del self.rowmap

# delete every attributes!!

print("DELETE <{}>".format(self.name))

else:

print("DELETE CANCELED")

else:

raise Exception("NO {} FILE" % fname)

def \_join\_on(self, other, on):

array = decomp(on)

A1 = array[0]

A2 = array[1]

B = []

C = []

for v1 in A1:

if not self.\_check(v1):

raise Exception("{} has no attribute {}".format(self.name, v1))

for v2 in A2:

if not other.\_check(v2):

raise Exception("{} has no attribute {}".format(other.name, v2))

for key1 in self.colmap:

if key1 not in A1:

B.append(key1)

for key2 in other.colmap:

if key2 not in A2:

C.append(key2)

rd = {}

result = [A1.copy()]

result[0].extend(B)

result[0].extend(C)

for i in range(1, len(self.table)):

key = [self.get(i, name) for name in A1]

key = tuple(key)

value = [self.get(i, name) for name in B]

if key not in rd:

rd[key] = [value]

else:

rd[key].append(value)

for i in range(1, len(other.table)):

key = [other.get(i, name) for name in A2]

key = tuple(key)

value = [other.get(i, name) for name in C]

if key in rd:

for array in rd[key]:

sub = list(key)

sub.extend(array.copy())

sub.extend(value.copy())

result.append(sub)

print("JOIN <{}>, <{}> ON {}".format(self.name, other.name, on))

return Table(result)

# bug issues switch order result change dictionary value should be list of list

def \_join(self, other):

newd = {}

for key in self.colmap:

newd[key] = 1

for key in other.colmap:

if key in newd:

newd[key] = 0

else:

newd[key] = -1

A = []

B = []

C = []

for key in newd:

if newd[key] == 0:

A.append(key)

elif newd[key] == 1:

B.append(key)

elif newd[key] == -1:

C.append(key)

rd = {}

result = [A.copy()]

result[0].extend(B)

result[0].extend(C)

for i in range(1, len(self.table)):

key = [self.get(i, name) for name in A]

key = tuple(key)

value = [self.get(i, name) for name in B]

if key not in rd:

rd[key] = [value]

else:

rd[key].append(value)

for i in range(1, len(other.table)):

key = [other.get(i, name) for name in A]

key = tuple(key)

value = [other.get(i, name) for name in C]

if key in rd:

for array in rd[key]:

sub = list(key)

sub.extend(array.copy())

sub.extend(value.copy())

result.append(sub)

print("JOIN <{}>, <{}> ON {}".format(self.name, other.name, ",".join(A)))

return Table(result)

def \_\_mul\_\_(self, other):

return self.join(other)

def join(self, other, on=None):

if on is None:

return self.\_join(other)

elif isinstance(on, str):

return self.\_join\_on(other, on)

def orderby(self, s):

# this should be done at origin vaiable not output another new

s = slist(s)

for v in s:

if not self.\_check(v):

raise Exception("no attribute {} in {}".format(v, self.name))

else:

array = []

for i in range(1, len(self.table)):

array.append([[self.get(i, v) for v in s], i])

array.sort()

result = [self.table[0].copy()]

for v in array:

result.append(self.table[v[1]].copy())

return Table(result)

pass

def setlib():

s = """

try:

import matplotlib.pyplot as plot

global plt

plt = plot

except Exception:

print("can not import matplotlib!")

try:

from mpl\_interaction import PanAndZoom

global PAZ

PAZ = PanAndZoom

except:

print("please download mpl\_interaction.py")

"""

return exec(s, globals(), locals())

def addrow(self, array):

if (len(array) == len(self.table[0])):

self.table.append(Row(array.copy(), self))

self.lenmap = [max(self.lenmap[i], len(str(array[i]))) for i in range(len(array))]

else:

raise Exception("length not match")

def addcol(self, name, array):

name = name.strip()

if name not in self.colmap:

try:

n = len(self.table[0])

self.table[0].append(name)

except:

n = 0

self.table.append([name])

self.colmap[name] = n

self.lenmap.append(len(str(name)))

for i in range(len(array)):

v = array[i]

if n == 0:

self.table.append([v])

else:

self.table[i + 1].append(v)

# print(self.lenmap,n,str(v))

self.lenmap[n] = max(self.lenmap[n], len(str(v)))

def \_\_matmul\_\_(self, other):

return self.cross(other)

def cross(self, other):

sa = newlist(len(self.table[0]), "None")

so = newlist(len(other.table[0]), "None")

for key in self.colmap:

sa[self.colmap[key]] = key

for key in other.colmap:

so[other.colmap[key]] = key

saso = []

saso.extend(sa)

saso.extend(so)

# print(saso)

result = Table([saso])

# print("good")

for r1 in range(1, len(self.table)):

for r2 in range(1, len(other.table)):

sub = []

sub.extend(self.table[r1])

sub.extend(other.table[r2])

result.addrow(sub)

return result

def \_\_len\_\_(self):

return len(self.table)

def bar(self, label, value):

Table.setlib()

self.orderby(label)

fig, ax = plt.subplots()

ax.bar([str(v) for v in self.col(label)], self.col(value))

plt.xlabel(label)

plt.ylabel(value)

pan\_zoom = PAZ(fig)

plt.show()

def pie(self, label, value):

Table.setlib()

fig, ax = plt.subplots()

ax.pie(self.col(value), labels=self.col(label), autopct='%1.1f%%')

pan\_zoom = PAZ(fig)

plt.show()

def hist(self, value, low, up, num):

Table.setlib()

fig, ax = plt.subplots()

amount = up - low

block = amount / num

bins = [low + i \* block for i in range(num + 2)]

ax.hist(self.col(value), bins, facecolor='green', edgecolor="yellowgreen")

plt.xlabel(value)

plt.ylabel("number")

pan\_zoom = PAZ(fig)

plt.show()

def plot(self, x, y, line="."):

# add polar coord

# add multiple and lines

# add spline

Table.setlib()

fig, ax = plt.subplots()

ax.plot(self.col(x), self.col(y), line)

plt.xlabel(x)

plt.ylabel(y)

pan\_zoom = PAZ(fig)

plt.show()

def \_\_getitem\_\_(self, key):

return self.table[key]

def \_\_setitem\_\_(self, key, value):

# add check

self.table[key] = value

self.lenmap = [max(self.lenmap[i], len(str(value[i]))) for i in range(len(value))]

def \_\_delitem\_\_(self, key):

# add check and primary key setting

self.table.pop(key)

pass

def \_\_contains\_\_(self, item):

# check primary keys

pass

def union(self, other):

# if one have more attributes include less one with None

pass

def intersect(self, other):

pass

def minus(self, other):

pass

def radar(self):

# multiple in one graph

pass

def polar\_radar(self):

pass

def hist2d(self):

pass

def bar2d(self):

pass

def scatter(self):

pass

def classify(self):

# classify the value

pass

def groupby(self, s):

# group table by attributes in s

# form a subtable in groupby dictionary

pass

class Row(list):

# behave all like a list and return list except get,set, del

def \_\_init\_\_(self, row, table=None):

list.\_\_init\_\_(self, row)

self.table = table

def getindex(self, key):

if isinstance(key, int) and key < len(self):

return key

elif isinstance(key, str) and self.table.\_check(key):

return self.table.colmap[key]

else:

raise KeyError("only int or str is supported")

def \_\_getitem\_\_(self, key):

index = self.getindex(key)

return super().\_\_getitem\_\_(index)

def \_set(self, key, value):

i = self.getindex(key)

# print(i)

if self == self.table.table[0]:

oldvalue = self[i]

indexcol = self.table.colmap[oldvalue]

del self.table.colmap[oldvalue]

self.table.colmap[value] = indexcol

super().\_\_setitem\_\_(i, value)

t = self.table

vlen = len(str(value))

if vlen > t.lenmap[i]:

t.lenmap[i] = vlen

elif vlen < t.lenmap[i]:

t.lenmap[i] = max([len(str(t.table[k][i])) for k in range(len(t.table))])

def \_\_setitem\_\_(self, key, value):

self.\_set(key, value)

def \_\_delitem\_\_(self, key):

self.\_set(key, None)

attributes

Building Function

Graph Display

Core function:

join

group

再多个entry内使用同样的参数，来达到group的目的，而在python内部用group dictionary来保存group信息，方便查看。

如果是竖向group， 名称放在第一栏，其他没有名字显视，

如果横向group，删掉分割符|， 按照设定居中或左对齐。（如果None的话可以选择不显示，但之前肯定会有|来区分group单元和None值）

被纵向group 过的话，group就可以做为一个key，通过[“someting”]就可以返回所有列，并且是一个table。考虑多个参数分几列group的情况。形成递归结构。被group的结构不能被下一轮group拆开，除非手动拆开group