Learning_Pandas_Part_103_ProblemSolving

June 21, 2021

[2]:
```
# To get multiple outputs in the same cell

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[3]:
```
# Import the required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

0.1 0. Data

[3]:
```
df1 = pd.DataFrame({'id': [1,2],
                   'name': ['a','b'],
                   'prem1' : [100,280],
                   'prem2' : [np.NaN,180],
                   'prem3' : [300,np.NaN],
                   'disc1' : [20,40],
                   'disc2' : [np.NaN,30],
                   'disc3' : [50,np.NaN],})
df1
```

[3]:
```
   id name  prem1  prem2  prem3  disc1  disc2  disc3
0   1    a    100    NaN  300.0     20    NaN   50.0
1   2    b    280  180.0    NaN     40   30.0    NaN
```

[4]:
```
df1.iloc[::-1,::-1]
```

[4]:
```
   disc3  disc2  disc1  prem3  prem2  prem1 name  id
1    NaN   30.0     40    NaN  180.0    280    b   2
0   50.0    NaN     20  300.0    NaN    100    a   1
```

[5]:
```
d = {"salesperson":["Nico", "Carlos", "Juan", "Nico", "Nico", "Juan", "Maria",␣
 ↪"Carlos"], "beer_item":[10, 120, 130, 200, 300, 550, 12.3, 200],
    "wine_item":[10, 120, 130, 200, 300, 550, 12.3, 200], "spirit_item":[10,␣
↪120, 130, 200, 300, 550, 12.3, 200]}
df = pd.DataFrame(d)
df

drink = 'wine'
#drink = ['wine','beer']
df[[f"salesperson",f"{drink}_item"]]
```

[5]:
```
  salesperson  beer_item  wine_item  spirit_item
0        Nico       10.0       10.0         10.0
1      Carlos      120.0      120.0        120.0
2        Juan      130.0      130.0        130.0
3        Nico      200.0      200.0        200.0
4        Nico      300.0      300.0        300.0
5        Juan      550.0      550.0        550.0
6       Maria       12.3       12.3         12.3
7      Carlos      200.0      200.0        200.0
```

[5]:
```
  salesperson  wine_item
0        Nico       10.0
1      Carlos      120.0
2        Juan      130.0
3        Nico      200.0
4        Nico      300.0
5        Juan      550.0
6       Maria       12.3
7      Carlos      200.0
```

0.2 1. Validate EMAIL ID

Valid email ID Description Consider that email IDs are supposed to be for the following format: username@website.extension. Here, there are three conditions to keep in mind: 1. The username can only contain characters 0-9, a-z and A-Z. 2. The website name can contain only characters a-z 3. The extension can have 2 or 3 alphabets(a-z).

Given an email ID, you have to determine if it is valid or not.

Sample Input: prerna@upgrad.com

Sample Output: valid

[6]:
```
import re

def checkmail(email):
    #complete the function
    #the function should return the strings "invalid" or "valid" based on the␣
↪email ID entered
    mo = re.search(r'\A[a-zA-Z0-9]+@[a-z]+\.[a-z]{2,3}$',email)
    if mo == None:
        return 'invalid'
    else:
        return 'valid'



email='hi*gail.com'
print(checkmail(email))
```

```
invalid
```

0.3 2. Flatten a list

Flatten a list Description Given a nested list, write python code to flatten the list. Note: The input list will strictly have 2 levels, i.e. the list will be of the form [[1,2],[3,4]]. Inputs like [1,[2,3]] and [[1,[2,3],4],5] are not applicable.

For example: If the input list is : [[1,2,3],[4,5],[6,7,8,9]] Then the output should be: [1,2,3,4,5,6,7,8,9]

[7]:
```
lst = [[1,2,3],[4,5],[6,7,8,9]]

fl = [y for x in lst for y in x]
fl
```

[7]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

0.4 3. Calculate squares conditionally

Description Given a list of positive integers, you have to find numbers divisible by 3 and replace them with their squares. For example, consider the list below: Input: [1,2,3,4,5,6] The output for the above list would be: [1,2,9,4,5,36]. Because 3 and 6 were divisible by 3, these numbers were replaced with their squares.

[8]:
```
lst =  [1,2,3,4,5,6]
lst
```

[8]:
```
[1, 2, 3, 4, 5, 6]
```

[9]:
```
sq_lst = [x**2 if x % 3 == 0 else x for x in lst]
sq_lst
```

[9]:
```
[1, 2, 9, 4, 5, 36]
```

0.5 4. A weird sum

Description

```
Write a program that computes the value of n+nn+nnn+nnnn with a given digit as the value of n.
```

For example, if n=9 , then you have to find the value of 9+99+999+9999.

[10]:
```
inp=input()

sums = int()
for i in range(1,5):
    s = ''
    for j in range(i):
        s += inp
    sums += int(s)
    print(s)
print(sums)
```

```
1
1
11
111
1111
1234
```

[11]:
```
n=input()
n1 = int( "%s" % n)
n2 = int( "%s%s" % (n,n) )
n3 = int( "%s%s%s" % (n,n,n) )
n4 = int( "%s%s%s%s" % (n,n,n,n) )

print (n1+n2+n3+n4)
```

```
1
1234
```

0.6 5. Frequent Letters

Description

Given a string, you have to find the first n most frequent characters in it.

You have to print these n letters in alphabetically sorted order.

The input will contain two lines, the first line will contain a string and the second line wil

The output should be a list of n most frequent letters in alphabetically sorted order.

Note: If there are two letters with the same frequency, then the alphabetically preceding alpha

Sample Input:

ddddaacccb

```
3

Sample Output:

['a', 'c', 'd']
```

[12]:
```python
string=input()
n=int(input())
#write your code here

#''.join(sorted(test_str))
uniq_char = sorted(list(set(string)), reverse=True)
#uniq_char
#type(uniq_char)

d = {}
#type(d)

for c in uniq_char:
    counter = string.count(c)
    #c
    #counter
    d[counter] = c
#d

sorted_dict = {r: d[r] for r in sorted(d, reverse=False)}
sorted_dict

ls = list(sorted_dict.values())
print(ls[-n:])
```

```
1
1
```

[12]:
```
{1: '1'}
```

```
['1']
```

[13]:
```python
string=input()
n=int(input())
import collections
out=[collections.Counter(string).most_common(i+1)[i][0] for i in range(n)]
out.sort()
print(out)
```

```
1
1
['1']
```

0.7 6. 2D array

Description

```
Write Python code which takes 2 numbers x and y as input and generates a 2-dimensional numpy a

Note: i=0,1,...x-1 and j=0,1....,y-1
    The input will have two lines with x and y respectively.
    The output should be a 2D numpy array.
```

Sample Input:

3

4

Sample Output:

[[0. 0.5 1. 1.5] [0.5 1. 1.5 2.] [1. 1.5 2. 2.5]]

[14]:
```
x=3
y=4
arr = np.ones([x,y])
arr
```

[14]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[15]:
```
arr1 = np.empty([x,y])
arr1
```

[15]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[16]:
```
for i in range(x):
    for j in range(y):
        arr[i,j] = (i+j)/2
arr
```

[16]:
```
array([[0. , 0.5, 1. , 1.5],
       [0.5, 1. , 1.5, 2. ],
       [1. , 1.5, 2. , 2.5]])
```

[17]:
```
x=int(input())
y=int(input())
from numpy import zeros
a = zeros([x,y])
for row in range(x):
    for col in range(y):
        a[row][col]= (row+col)/2
print(a)
```

```
1
1
[[0.]]
```

0.8 7. 8th Highest Purchase

- One complex sql query- 2 table are there, Table1(cust_id,Name) Table2(cust_id,Transaction_amt)
- Write a query to return the name of customers with 8th highest lifetime purchase.

[4]:
```
rows = [['1','Abhishek'],
        ['2','Anusha'],
        ['3','BalaJi']]

columns = ['id','name']

cust = pd.DataFrame(rows, columns=columns)
cust
```

[4]:
```
  id      name
0  1  Abhishek
1  2    Anusha
2  3    BalaJi
```

[5]:
```
data_dict = {'id' : ['1','1','2','3','1','2','3','1','3','3','3','2'],
             'amt' : [100,50,200,300,400,500,20,10,100,180,30,600]}

trans = pd.DataFrame(data_dict)
trans
```

[5]:
```
   id  amt
0   1  100
1   1   50
2   2  200
3   3  300
4   1  400
5   2  500
6   3   20
7   1   10
8   3  100
9   3  180
10  3   30
11  2  600
```

[12]:
```
# create a list for column 1
id = ['1','1','2','3','1','2','3','1','3','3','3','2']

# create a list for column 2
amt = [100,50,200,300,400,500,20,10,100,180,30,600]

list_of_tuples = list(zip(id, amt))
list_of_tuples

transtest = pd.DataFrame(list_of_tuples, columns = ['id','amt'])
transtest
```

[12]:
```
[('1', 100),
 ('1', 50),
 ('2', 200),
 ('3', 300),
 ('1', 400),
 ('2', 500),
 ('3', 20),
 ('1', 10),
 ('3', 100),
 ('3', 180),
 ('3', 30),
 ('2', 600)]
```

[12]:
```
   id  amt
0   1  100
1   1   50
2   2  200
3   3  300
4   1  400
5   2  500
6   3   20
7   1   10
8   3  100
9   3  180
10  3   30
11  2  600
```

[21]:
```
# Method 1 : Creating an Aggregate table and renaming columns
trans_agg = trans.groupby('id', as_index=False)['amt'].sum().rename(columns =␣
 ↪{'amt': 'TotAmt'})
trans_agg
```

[21]:
```
  id  TotAmt
0  1     560
1  2    1300
```

```
2  3     630
```

[20]:
```
# Method 2 : Creating an Aggregate table with NamedAggregate
trans.groupby('id', as_index=False).agg(TotAmount = pd.NamedAgg('amt','sum'))
```

[20]:
```
   id  TotAmount
0   1        560
1   2       1300
2   3        630
```

[23]:
```
all = pd.merge(cust,trans_agg, on='id')
all
```

[23]:
```
   id      name  TotAmt
0   1  Abhishek     560
1   2    Anusha    1300
2   3    BalaJi     630
```

[27]:
```
all.sort_values('TotAmt', ascending=False)
```

[27]:
```
   id      name  TotAmt
1   2    Anusha    1300
2   3    BalaJi     630
0   1  Abhishek     560
```

[31]:
```
# Here 2nd Highest is found, by passing row index = 1 on the sorted data
all.sort_values('TotAmt', ascending=False).reset_index().drop('index', axis=1).
 ↪iloc[[1]]
```

[31]:
```
   id    name  TotAmt
1   3  BalaJi     630
```

[37]:
```
# Here 2nd highest is found, by creating a Rank on Salary and then filtering it
all['rnk'] = all['TotAmt'].rank()
all[all.rnk==2]
```

[37]:
```
   id    name  TotAmt  rnk
2   3  BalaJi     630  2.0
```

[38]:
```
all['TotAmt'].nlargest(2)
```

[38]:
```
1    1300
2     630
Name: TotAmt, dtype: int64
```

[39]:
```
all.nlargest(2,['TotAmt'])
```

```
[39]:    id    name  TotAmt  rnk
       1  2  Anusha    1300  3.0
       2  3  BalaJi     630  2.0
```

0.9 9. To replace values greater than 1000 with Null for all the columns or numeric columns in dataframe

```
[42]: # Data Preparation
      id = [1,2,3,4,5]
      prod1 = [100,np.NaN,250,225,300]
      prod2 = [400,100,250,np.NaN,60]

      list_of_tuples = list(zip(id, prod1,prod2))
      list_of_tuples

      data = pd.DataFrame(list_of_tuples, columns = ['id','prod1','prod2'])
      data
```

```
[42]: [(1, 100, 400), (2, nan, 100), (3, 250, 250), (4, 225, nan), (5, 300, 60)]
```

```
[42]:     id  prod1  prod2
       0   1  100.0  400.0
       1   2    NaN  100.0
       2   3  250.0  250.0
       3   4  225.0    NaN
       4   5  300.0   60.0
```

```
[46]: data['prod11'] = data['prod1'].apply(lambda x : 0 if x >= 250 else x)
      data
```

```
[46]:     id  prod1  prod2  prod11
       0   1  100.0  400.0   100.0
       1   2    NaN  100.0     NaN
       2   3  250.0  250.0     0.0
       3   4  225.0    NaN   225.0
       4   5  300.0   60.0     0.0
```

```
[47]: data.applymap(lambda x : 0 if x >= 250 else x)
```

```
[47]:     id  prod1  prod2  prod11
       0   1  100.0    0.0   100.0
       1   2    NaN  100.0     NaN
       2   3    0.0    0.0     0.0
       3   4  225.0    NaN   225.0
       4   5    0.0   60.0     0.0
```

```
[ ]:
```