Given a dataframe with numeric values, how do you get the row-wise and column-wise average values? For instance, given a dataframe with 4 columns and 10 rows, how to get the average value per column/row (For rows, 10 average values i.e., per row; For columns, 4 average values i.e., per column)?

In [145…
```
df_num = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'B': [10, 9, 8, 7, 6, 5, 4, 3, 2,3], 'C': [1, 4, 9, 16, 25, 36, 49, 64, 81, 100],
                      'D': [100, 81, 64, 49, 36, 25, 16, 9, 4, 1]})
```

In [146…
```
df_num
```

Out[146]:

	A	B	C	D
0	1	10	1	100
1	2	9	4	81
2	3	8	9	64
3	4	7	16	49
4	5	6	25	36
5	6	5	36	25
6	7	4	49	16
7	8	3	64	9
8	9	2	81	4
9	10	3	100	1

In [147…
```
row_wise_average = df.mean(axis=1)
print("Row-wise average values:")
print(row_wise_average)
```

```
Row-wise average values:
0    28.0
1    24.0
2    21.0
3    19.0
4    18.0
5    18.0
6    19.0
7    21.0
8    24.0
9    28.5
dtype: float64
```

In [148…
```
column_wise_average = df.mean(axis=0)
print("\nColumn-wise average values:")
print(column_wise_average)
```

```
Column-wise average values:
A     5.5
B     5.7
C    38.5
D    38.5
dtype: float64
```

Given a dataframe with the following columns: FirstName, LastName, Gender; Create another column called, “FullName”, that contains concatenated derived values of the columns in the form of “<Salutation> <FirstName> <LastName>”, where “<Salutation>” is either “Mr.” or “Ms.” depending on the Gender column values (Values for Gender are only “Male” and “Female”). For instance, for a row with the values, {FirstName= ‘John’, LastName=’Smith’, Gender=’Male’}, the FullName will be “Mr. John Smith”.

In [140…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [32]:
```
dfs=pd.DataFrame({'Name':['Prema','Sona','Soma'],'Last':['Maity','Sharma','kumar'],'Gender':['Female','Male','Female']})
```

In [33]:
```
dfs
```

Out[33]:

	Name	Last	Gender
0	Prema	Maity	Female
1	Sona	Sharma	Male
2	Soma	kumar	Female

In [36]:
```
def new(gender):
    if gender=='Female':
        return "miss"
    else:
        return 'Mr'
```

In [41]:
```
dfs['Full Name']=dfs.Gender.apply(new)+" "+dfs.Name+" "+dfs.Last
```

In [42]:
```
dfs
```

Out[42]:

	Name	Last	Gender	Full Name
0	Prema	Maity	Female	miss Prema Maity
1	Sona	Sharma	Male	Mr Sona Sharma
2	Soma	kumar	Female	miss Soma kumar

Given a dataframe with 2 string-based columns with duplicate occurrences, find out the frequency of the occurrences of each value combination of the 2 columns. For instance, if the dataframe were represented as {‘column1’: [‘a’, ‘a’, ‘b’, ‘b’, ‘a’, ‘a’], ‘column2’: [‘c’, ‘c’, ‘c’, ‘d’, ‘d’, ‘d’, ‘e’]} then the result could be represented as [(‘a’, ‘c’) = 2, (‘a’, ‘d’) = 1, (‘a’, ‘e’) = 1, (‘b’, ‘c’) = 1, (‘b’, ‘d’) = 1].

In [43]:
```
df = pd.DataFrame({'column1': ['a', 'a', 'b', 'b', 'a', 'a'],'column2': ['c', 'c', 'c', 'd', 'd', 'd']})
```

In [44]:
```
df
```

Out[44]:

	column1	column2
0	a	c
1	a	c
2	b	c
3	b	d
4	a	d
5	a	d

In [57]:
```
result=df.groupby(['column1', 'column2']).size().reset_index(name='counts')
```

In [58]:
```
result
```

Out[58]:

	column1	column2	counts
0	a	c	2
1	a	d	2
2	b	c	1
3	b	d	1

In [59]:
```
# advance
```

Given a dataframe with the following columns: LocationID, XValue, YValue; Add another column, “NearestLocationID”, that contains the LocationID that is closest to the respective LocationIDs. For instance, given 3 rows [(1, 50, 100), (2, 20, 100), (3, 50, 150)], the resultant column will be [2, 1, 1].

This solution I took help

In [60]:
```
df_location = pd.DataFrame({'LocationId':[1,20,100],'XValue':[2,20,100],'YValue':[3,50,150]})
```

In [61]:
```
df_location
```

Out[61]:

	LocationId	XValue	YValue
0	1	2	3
1	20	20	50
2	100	100	150

In [63]:
```
def calc_distance(row, df_location):
    x_diff = df_location['XValue'] - row['XValue']
    y_diff =  df_location['YValue'] - row['YValue']
    distance = np.sqrt(x_diff**2 + y_diff**2)
    return distance
```

In [70]:
```
df_location
```

Out[70]:

	LocationId	XValue	YValue	NearestLocationID
0	1	2	3	1
1	20	20	50	20
2	100	100	150	100

In [71]:
```
df = pd.DataFrame({'LocationID': [1, 2, 3],
                   'XValue': [50, 20, 50],
                   'YValue': [100, 100, 150]})

# calculate the distance between each pair of locations
def calc_distance(row, df):
    x_diff = df['XValue'] - row['XValue']
    y_diff = df['YValue'] - row['YValue']
    distance = np.sqrt(x_diff**2 + y_diff**2)
    return distance

# create the NearestLocationID column
df['NearestLocationID'] = df.apply(lambda row: df.loc[calc_distance(row, df).idxmin(), 'LocationID'], axis=1)
```

In [72]:
```
df
```

Out[72]:

	LocationID	XValue	YValue	NearestLocationID
0	1	50	100	1
1	2	20	100	2
2	3	50	150	3

Given a dataframe with the following columns: Race100m, Race200m, Race500m, which are indicative of the race distance in meters; And the rows are indicative of records of each individual athlete, while the values are time taken (Seconds) by corresponding athletes to finish the respective races. Find out the average speed (In m/sec i.e., meters per second) of each athlete, considering that the athletes run at the same own pace in each race.

In [73]:
```
# example dataframe
df = pd.DataFrame({'Race100m': [10.0, 9.5, 10.2], 'Race200m': [20.0, 19.0, 20.1], 'Race500m': [50.0, 48.0, 49.5]})
# add columns for distance and speed for each race
df['Speed100m'] = 100 / df['Race100m']
df['Speed200m'] = 200 / df['Race200m']
df['Speed500m'] = 500 / df['Race500m']
# calculate the average speed across all races
df['AverageSpeed'] = df[['Speed100m', 'Speed200m', 'Speed500m']].mean(axis=1)
```

In [74]:
```
df
```

Out[74]:

	Race100m	Race200m	Race500m	Speed100m	Speed200m	Speed500m	AverageSpeed
0	10.0	20.0	50.0	10.000000	10.000000	10.000000	10.000000
1	9.5	19.0	48.0	10.526316	10.526316	10.416667	10.489766
2	10.2	20.1	49.5	9.803922	9.950249	10.101010	9.951727

Given an AlphaNumeric string, return the string with all the numbers removed.

In [87]:
```
a='prema123'
```

In [92]:
```
a_new = "".join((x for x in a if not x.isdigit()))
```

In [93]:
```
a_new
```

Out[93]:
```
'prema'
```

Given a list of integers (both positive and negative), find the weighted average based on the following 2 conditions: The numbers have integer-ranged weights in the descending order (ordering includes zeroes), and the average value is calculated while ignoring the 0 values. For instance, given [8, 1, -9, 0, 4, 0, 0, -15] the average would be (8*8 + 7*4 + 6*1 + 2*-9 +1*-15)/5. To simplify, the sorted list would be [8, 4, 1, 0, 0, 0, -9, -15] with the corresponding weights as [8, 7, 6, 0, 0, 0, 2, 1], and the valid denominator count is 5 (Total non-zero numbers).

In [137…
```
def weighted_average(lst):
    sorted_lst = sorted(lst, reverse=True)
    weighted_sum = 0
    weight = len(lst)
    count = 0
    for i in sorted_lst:
        if i == 0:
            weight -= 1
        else:
            weighted_sum += weight * i
            count =count+1
    return weighted_sum / count if count > 0 else 0
```

In [138…
```
lst=[8, 7, 6, 0,0, 0, 2, 1]
```

In [139…
```
weighted_average(lst)
```

Out[139]:
```
38.4
```

Given an integer, find the ratio of the reversed integer (Position of digits are reversed, i.e., 123 becomes 321) to the sum of all the digits in the integer.

In [143…
```
def reverse_ratios(x):
    reverse= int(str(x)[::-1])
    suma=sum([int(i) for i in str(x)])
    return reverse/suma
```

In [144…
```
reverse_ratios(123)
```

Out[144]:
```
53.5
```

1. Given 2 integers, find out the resulting integer that would be the exponent relating to the given2 numbers. By definition: a^n = b, where a, b are provided as input, while n needs to be computed. For instance, given 5 and 625, the result would be 4.

In [145…
```
import math

def find_exponent(a, b):
    return int(math.log(b) / math.log(a))
```

In [146…
```
find_exponent(5,625)
```

Out[146]:
```
4
```

6. Write star pattern

*

- *

- *

In [162…
```
def triangle_pattern(n):
    for i in range(1, n + 1):
        for j in range(i):
            print("* ", end="")
        print("")
    for i in range(n - 1, 0, -1):
        for j in range(i):
            print("* ", end="")
        print("")
```

In [163…
```
triangle_pattern(6)
```

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

7 7. S = "asdfg12345!_()*&#$%dsfjjhf" a. Separate similar datatypes into different lists b. Remove numbers and return string c. Return no of vowels and numbers present.

In [8]:
```
def separate_lists(s):
    letters = []
    numbers = []
    special_characters = []
    vowels = "aeiouAEIOU"
    for char in s:
        if char.isalpha():
            letters.append(char)
        elif char.isdigit():
            numbers.append(char)
        else:
            special_characters.append(char)
    return letters, numbers, special_characters

def remove_numbers(s):
    return ''.join([char for char in s if not char.isdigit()])

def count_vowels_and_numbers(s):
    vowels = "aeiouAEIOU"
    vowels_count = 0
    numbers_count = 0
    for char in s:
        if char.isdigit():
            numbers_count += 1
        elif char in vowels:
            vowels_count += 1
    return vowels_count, numbers_count
```

In [7]:
```
S="&quot;asdfg12345!_()*&amp;#$%dsfjjhf&quot"
letters, numbers, special_characters = separate_lists(S)
print("Letters:", letters)
print("Numbers:", numbers)
print("Special Characters:", special_characters)

without_numbers = remove_numbers(S)
print("String without numbers:", without_numbers)

vowels_count, numbers_count = count_vowels_and_numbers(S)
print("Vowels count:", vowels_count)
print("Numbers count:", numbers_count)
```

```
Letters: ['q', 'u', 'o', 't', 'a', 's', 'd', 'f', 'g', 'a', 'm', 'p', 'd', 's', 'f', 'j', 'j', 'h', 'f', 'q', 'u', 'o', 't']
Numbers: ['1', '2', '3', '4', '5']
Special Characters: ['&', ';', '!', '_', '(', ')', '*', '&', ';', '#', '$', '%', '&']
String without numbers: &quot;asdfg!_()*&amp;#$%dsfjjhf&quot
Vowels count: 6
Numbers count: 5
```

Intermediate

In [135…
```
#1. already done above
```

1. a a. Given a list of strings, segregate the strings (all characters, no digits) into 2 lists (Homogeneous and Heterogeneous). Definition of Homogeneous in this exercise is if the characters are either all upper-case or all lower-case. Anything else would be classified as Heterogeneous.

In [46]:
```
def homo_hetero(lis):
    hetero=[]
    homo=[]
    for i in range(0,len(lis)):
        if (lis[i]==lis[i].upper() or lis[i]==lis[i].lower()):
            hetero.append(lis[i])
        else:
            homo.append(lis[i])

    print("HOMO :",homo)
    print("Hetetro :",hetero)
```

In [47]:
```
homo_hetero(lis)
```

```
HOMO : ['AbhishekRoy']
Hetetro : ['prema', 'SOMA']
```

In [132…
```
# 2. b already done above
```

1. #Given a string in the following format: “CharIntCharInt....” i.e. a character (a-z, A- Z) followed by an integer (Between 1 and 2147483647), and the string could be as big as needed. Decompose the string to avail a list comprising the characters amounting according to the characters’ succeeding integers. For instance, given an input string, “x11y3”, the expected output is [x, x, x, x, x, x, x, x, x, x, x, y, y, y].

In [99]:
```
def char_digit_split(input_string):
    result = []
    i = 0
    while i < len(input_string):
        char = input_string[i]
        i += 1
        count = ""
        while i < len(input_string) and input_string[i].isdigit():
            count += input_string[i]
            i += 1
        result.extend([char] * int(count))
    return result
```

In [100…
```
char_digit_split('x11y3')
```

Out[100]:
```
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'y', 'y', 'y']
```

Given a string containing a simple arithmetic expression (Arithmetic expressions

considered include only addition [“+”], subtraction [“-”], multiplication [“*”] and division [“/”]), decompose the string to find the result of the expression. For instance, given a sample string, “4+5”, the expected result is 9 (Datatype is up to the candidate).

In [131…
```
# solution required
```

In [ ]: