Python Mandatory

February 24, 2022

Q1) Given two matrices please print the product of those two matrices.

Example 1: $A = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 5 & 7 \\ 5 & 9 & 6 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $A \times B = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 5 & 7 \\ 5 & 9 & 6 \end{bmatrix}$

Example 2: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$, $A \times B = \begin{bmatrix} 11 & 14 & 17 & 20 & 23 \\ 23 & 30 & 37 & 44 & 51 \end{bmatrix}$

Example 3: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 4 \\ 5 & 6 \\ 7 & 8 \\ 9 & 6 \end{bmatrix}$, $A \times B =$ Not possible

[1]:
```python
class MatrixMultiplication(object):
    """
    This class is for matrix multiplication.
    """

    def __init__(self):
        pass

    def matrix_test(self, mat) -> bool:
        """
        This method tests the matrix.
        """
        rows = len(mat)
        if rows == 0:
            return False
        else:
            cols = [len(row) for row in mat if isinstance(row, list)]
            if sum(cols) == 0:
                return False
            elif len(set(cols)) > 1:
                return False
            else:
                return sum(cols) == rows * cols[0]

    def matrix_shape(self, mat) -> tuple:
```

```
        """
        This method calculates the shape of the matrix.
        """
        if self.matrix_test(mat=mat):
            rows = len(mat)
            cols = len(mat[0])
            return (rows, cols)
        else:
            return (None, None)

    def mat_mul_rule(self, mat1, mat2) -> bool:
        """
        This method checks the matrix multilication rule.
        """
        mat1_shape = self.matrix_shape(mat=mat1)
        mat2_shape = self.matrix_shape(mat=mat2)
        if (isinstance(mat1_shape[1], int) and
            isinstance(mat2_shape[0], int)):
            return mat1_shape[1] == mat2_shape[0]
        else:
            return False

    def transpose(self, mat) -> list:
        """
        This method transposes the given matrix.
        """
        row_len, col_len = self.matrix_shape(mat=mat)
        if isinstance(row_len, int) and isinstance(col_len, int):
            max_len = row_len if row_len >= col_len else col_len
            return [[row[i] for row in mat] for i in range(max_len)]
        else:
            return None

    def dot_product(self, r, c) -> int:
        """
        This method dots the given row and col.
        r = row and c = col.
        """
        dot = 0
        for i in range(len(r)):
            dot += (r[i] * c[i])
        return dot

    def matrix_mul(self, mat1, mat2) -> list:
        """
        This method accepts two matrices and multiplies them.
        """
```

```
        if self.mat_mul_rule(mat1=mat1, mat2=mat2):
            mat2_T = self.transpose(mat=mat2)
            output = [[self.dot_product(r=r, c=c) for c in mat2_T]
                      for r in mat1]
        else:
            output = "Not possible!"
        return output
```

[2]:
```
A = [[1, 2], [3], []]
B1 = [[1, 2], [3], [5]]
B2 = [[1, 2], [3], [5, 6]]
B3 = [[1, 2], [3], [5, 6, 7]]
B4 = [[1, 2, 3], [3], [1, 2, 4]]
C = [[]]
D = [1, 2, 3]
E = [[1], [2], [3]]
F = [[1, 2, 3]]
G = [[1, 2, 3], [1, 2, 3]]
```

[3]:
```
mat_mul = MatrixMultiplication()
for mat in [A, B1, B2, B3, B4, C, D, E, F, G]:
    print(mat_mul.matrix_test(mat=mat))
    print(mat_mul.matrix_shape(mat=mat))
```

```
False
(None, None)
False
(None, None)
False
(None, None)
False
(None, None)
False
(None, None)
False
(None, None)
False
(None, None)
True
(3, 1)
True
(1, 3)
True
(2, 3)
```

[4]:
```
mat1 = [[1, 2], [3, 4]]
mat2 = [[1, 2, 3, 4, 5], [5, 6, 7, 8, 9]]
```

[5]:
```
mat_mul = MatrixMultiplication()
print(mat_mul.matrix_mul(mat1=mat1, mat2=mat2))
```

```
[[11, 14, 17, 20, 23], [23, 30, 37, 44, 51]]
```

[6]:
```
mat1 = [[1], [2], [3]]
mat2 = [[1, 2, 3]]
```

[7]:
```
mat_mul = MatrixMultiplication()
print(mat_mul.matrix_mul(mat1=mat1, mat2=mat2))
```

```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

[8]:
```
mat1 = [[1, 2], [3, 4]]
mat2 = [[1, 4], [5, 6], [7, 8], [9, 6]]
```

[9]:
```
mat_mul = MatrixMultiplication()
print(mat_mul.matrix_mul(mat1=mat1, mat2=mat2))
```

```
Not possible!
```

Q2) Proportional Sampling – Select a number randomly with probability proportional to its magnitude from the given array of n elements.

Consider an experiment, selecting an element from the list A randomly with probability proportional to its magnitude. Assume we are doing the same experiment for 100 times with replacement, in each experiment you will print a number that is selected randomly from A.

Example 1: $A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]$

Let $f(x)$ denote the number of times x getting selected in 100 experiments.

$f(100) > f(79) > f(45) > f(28) > f(27) > f(13) > f(10) > f(6) > f(5) > f(0)$

Further learning on `random` module: here.

[10]:
```
class ProportionalSampling(object):
    """
    This class is for proportional sampling.
    """

    def __init__(self, data):
        self.data = data
        self.experiment = None

    def sum_of_data(self) -> int:
        """
        This method returns the sum of the data points.
        """
        return sum(self.data)
```

```
    def probability(self) -> list:
        """
        This method calculates probability of each data point.
        """
        return [(d / self.sum_of_data()) for d in self.data]

    def do_experiment(self, ktimes=100):
        """
        This method does the statistical experiment.
        """
        import random
        return random.choices(population=self.data,
                              weights=self.probability(),
                              k=ktimes)

    def hist_of_experiment(self) -> dict:
        """
        This method obtains the frequency of the experiment.
        """
        hist = dict()
        for obs in self.do_experiment():
            hist.update({obs: hist.get(obs, 0) + 1})
        hist = dict(sorted(hist.items(),
                           key=lambda tup: tup[1],
                           reverse=True))
        return hist
```

[11]:
```
data = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
f_x = ProportionalSampling(data=data)
print(f_x.hist_of_experiment())
```

```
{100: 30, 79: 23, 45: 19, 28: 14, 27: 7, 10: 5, 13: 2}
```

Q3) Replace the digits in the string with #.

Consider a string that will have digits in that, we need to remove all the not digits and replace the digits with #.

- Example 1: A = '234' and Output: '###'
- Example 2: A = 'a2b3c4' and Output: '###'
- Example 3: A = 'abc' and Output: ''
- Example 5: A = '#2a$#b%c%561#' and Output: '####'

[12]:
```
def check_digit_replace(string) -> str:
    """
    This function replaces digits with '#'.
    """
```

```
    output = str()
    for char in string:
        if char.isdigit():
            output += '#'
    return output
```

[13]:
```
strings = ['234', 'a2b3c4', 'abc', '#2a$#b%c%561#']
for s_no, string in enumerate(strings):
    print("{} --> {}".format(s_no+1, check_digit_replace(string=string)))
```

```
1 --> ###
2 --> ###
3 -->
4 --> ####
```

Q4) Students marks dashboard.

Consider the marks list of class students given two lists –

Students = ['student1', 'student2', 'student3', 'student4', 'student5', 'student6', 'student7', 'student8', 'student9', 'student10']

Marks = $[45, 78, 12, 14, 48, 43, 45, 98, 22, 80]$

From the above two lists the Student[0] got Marks[0], Student[1] got Marks[1] and so on

Your task is to print the name of students

- Who got top 5 ranks, in the descending order of marks.
- Who got least 5 ranks, in the increasing order of marks.
- Who got marks between $> 25^{th}$ percentile $< 75^{th}$ percentile, in the increasing order of marks.

Further learning on percentile formula: here.

Percentiles

$$l_p = \left[(n-1)\frac{p}{100}\right] + 1$$

l_p – location of p^{th}percentile.

$$p_v = X[\text{prev}] + \left(\text{floating part of } l_p \times (X[\text{curr}] - X[\text{prev}])\right)$$

p_v – percentile value.

[14]:
```
class MarksDashboard(object):
    """
    This class generates the dashboard.
    """

    def __init__(self, students, marks):
        self.students = students
```

```
        self.marks = marks

    def process_data(self) -> dict:
        """
        This method processes the data.
        """
        data = {k: v for (k, v) in zip(self.students, self.marks)}
        data = dict(sorted(data.items(),
                           key=lambda tup: tup[1]))
        return data

    def top_5(self) -> dict:
        """
        This method lists top 5 students.
        """
        data = self.process_data()
        top_5_marks = list(data.values())[-5:]
        print("Top 5 ranks in the class:")
        top_5_ranks = {k: v for (k, v) in data.items()
                       if v in top_5_marks}
        return dict(sorted(top_5_ranks.items(),
                           key=lambda tup: tup[1],
                           reverse=True))

    def least_5(self) -> dict:
        """
        This method lists least 5 students.
        """
        data = self.process_data()
        least_5_marks = list(data.values())[:5]
        print("Least 5 ranks in the class:")
        return {k: v for (k, v) in data.items()
                if v in least_5_marks}

    def percentile(self, p, data) -> int:
        """
        This method computes the pth percentile.
        """
        lp = ((len(data) - 1) * (p / 100)) + 1
        curr = int(lp)
        floatlp = lp - curr
        prev = curr - 1
        return data[prev] + (floatlp * (data[curr] - data[prev]))

    def IQR(self) -> dict:
        """
        This method computes the IQR.
```

```
        """
        data = self.process_data()
        p_25 = self.percentile(p=25, data=list(data.values()))
        p_75 = self.percentile(p=75, data=list(data.values()))
        print("IQR")
        print("25th Percentile = {}".format(p_25))
        print("75th Percentile = {}".format(p_75))
        marks_iqr = [mark
                     for mark in list(data.values())
                     if mark >= p_25 and mark < p_75]
        return {k: v for (k, v) in data.items() if v in marks_iqr}

    def display_dashboard(self) -> None:
        """
        This method displays the dashboard.
        """
        print(self.top_5(), "\n")
        print(self.least_5(), "\n")
        print(self.IQR())
        return None
```

[15]:
```
students = ['Student1', 'Student2', 'Student3', 'Student4', 'Student5',
            'Student6', 'Student7', 'Student8', 'Student9', 'Student10']
marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
```

[16]:
```
mark_dash = MarksDashboard(students=students, marks=marks)
mark_dash.display_dashboard()
```

```
Top 5 ranks in the class:
{'Student8': 98, 'Student10': 80, 'Student2': 78, 'Student5': 48, 'Student7':
47}

Least 5 ranks in the class:
{'Student3': 12, 'Student4': 14, 'Student9': 35, 'Student6': 43, 'Student1': 45}

IQR
25th Percentile = 37.0
75th Percentile = 70.5
{'Student6': 43, 'Student1': 45, 'Student7': 47, 'Student5': 48}
```

Q5) Find the closest points.

Consider you have been given n data points in the form of list of tuples like,

$S = [(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), \ldots, (x_n, y_n)]$

and a point $P = (p, q)$.

Your task is to find 5 closest points (based on cosine distance) in S from P.

Cosine distance between two points X (x, y) and P (p, q) is defind as $-$

$$\overline{XP} = cos^{-1}\left(\frac{xp + yq}{\sqrt{x^2 + y^2}\sqrt{p^2 + q^2}}\right)$$

$S = [(1, 2), (3, 4), (-1, 1), (6, -7), (0, 6), (-5, -8), (-1, -1), (6, 0), (1, -1)]$

$P = (3, -4)$

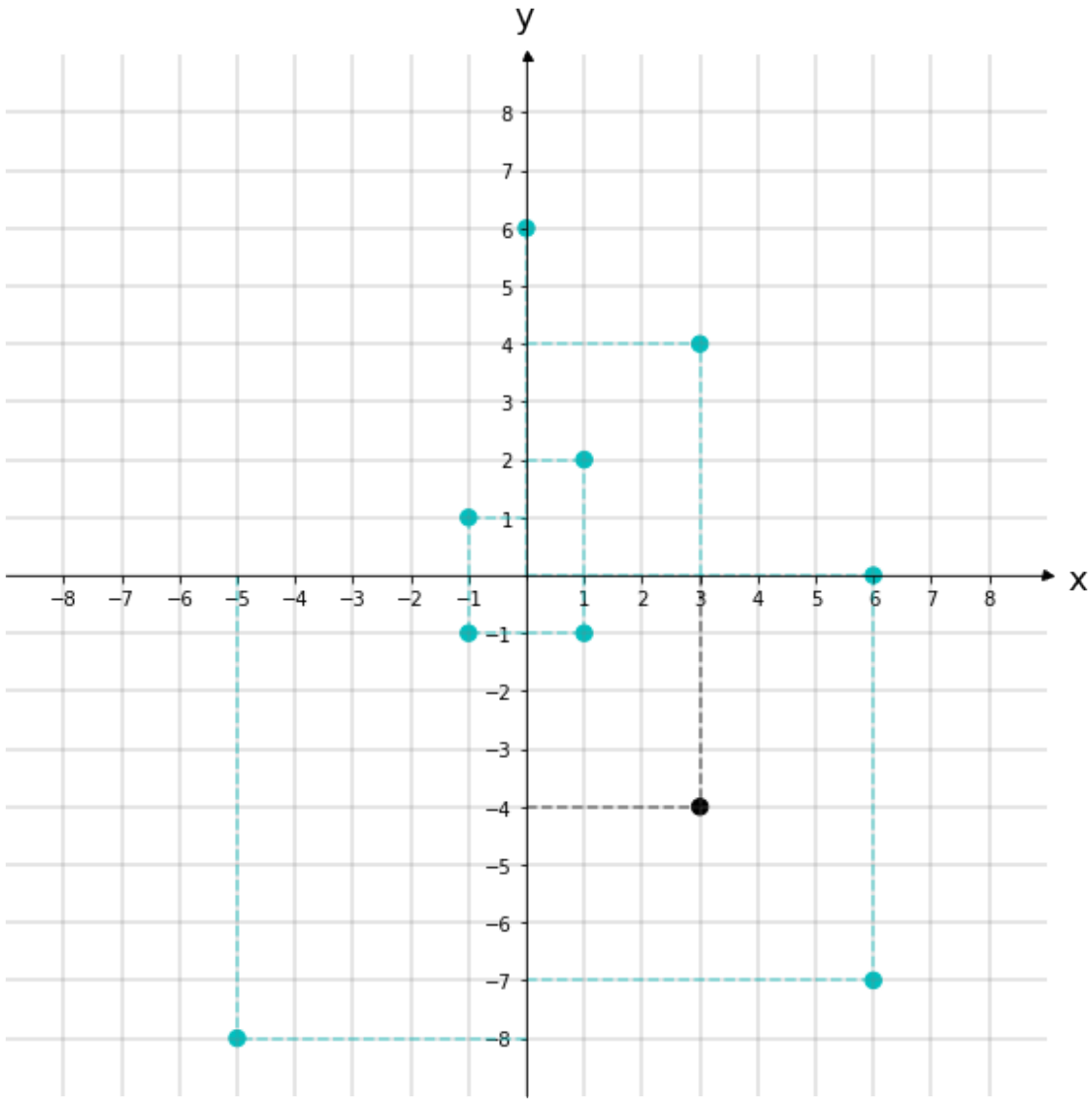

Image source: here.

[17]:
```
class CosineDistance(object):
    """
    This class finds closest points using cosine distance.
    """
```

```
    def __init__(self, points, P):
        self.points = points
        self.P = P

    def cosine_distance(self, P1, P2) -> float:
        """
        This method computes cosine distance between two points.
        """
        from math import acos
        from math import sqrt
        x, y = P1
        p, q = P2
        numerator = x*p + y*q
        denominator = sqrt(x**2 + y**2) * sqrt(p**2 + q**2)
        d = acos(numerator / denominator)
        return round(d, 3)

    def all_distances(self) -> None:
        """
        This method finds cosine distance between points and query point.
        """
        ds = [self.cosine_distance(P1=P1, P2=self.P)
              for P1 in self.points]
        pds = {k: v for (k, v) in zip(self.points, ds)}
        pds = dict(sorted(pds.items(), key=lambda tup: tup[1]))
        ds = sorted(ds)
        cds = ds[:5]
        cps = {k: v for (k, v) in pds.items() if v in cds}
        print("Closest points:")
        print(cps, "\n")
        fds = ds[5:]
        fps = {k: v for (k, v) in pds.items() if v in fds}
        print("Farthest points:")
        print(fps)
        return None
```

[18]:
```
points = [(1, 2), (3, 4), (-1, 1), (6, -7), (0, 6),
          (-5, -8), (-1, -1), (6, 0), (1, -1)]
P = (3, -4)
```

[19]:
```
c_d = CosineDistance(points=points, P=P)
c_d.all_distances()
```

```
Closest points:
{(6, -7): 0.065, (1, -1): 0.142, (6, 0): 0.927, (-5, -8): 1.202, (-1, -1):
1.429}

Farthest points:
```

```
{(3, 4): 1.855, (1, 2): 2.034, (0, 6): 2.498, (-1, 1): 3.0}
```

Q6) Find which line separates red balls and blue balls.

Consider you have given two set of data points in the form of list of tuples and set of line equations

- Red = $[(R_{11}, R_{12}), (R_{21}, R_{22}), (R_{31}, R_{32}), (R_{41}, R_{42}), (R_{51}, R_{52}), \ldots, (R_{n1}, R_{n2})]$
- Blue = $[(B_{11}, B_{12}), (B_{21}, B_{22}), (B_{31}, B_{32}), (B_{41}, B_{42}), (B_{51}, B_{52}), \ldots, (B_{m1}, B_{m2})]$
- Lines = $[a_1x + b_1y + c_1, a_2x + b_2y + c_2, a_3x + b_3y + c_3, a_4x + b_4y + c_4, \ldots, k$ lines].

Your task is for each line, print `True`, if all the red balls are one side of the line and blue balls are on the other side of the line, otherwise print `False`.

Example:

- Red = $[(1, 1), (2, 1), (4, 2), (2, 4), (-1, 4)]$
- Blue = $[(-2, -1), (-1, -2), (-3, -2), (-3, -1), (1, -3)]$
- Lines = ['1x+1y+0', '1x-1y+0', '1x+0y-3', '0x+1y-0.5']

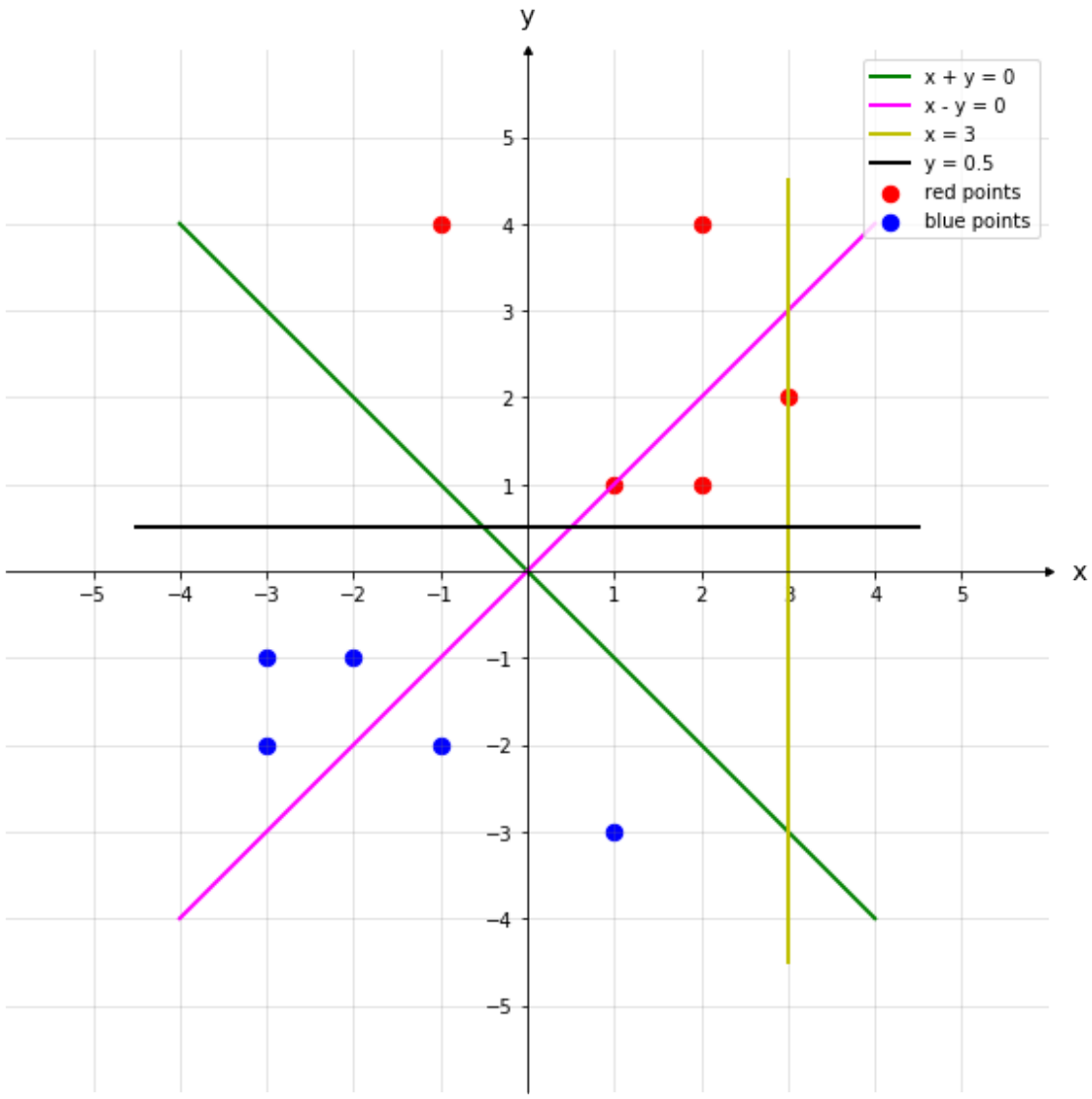

Image source: here.

Formula:

The distance d between a point $P = (x_0, y_0)$ and a line $L : ax + by + c = 0$ is

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

here, in the formula we force d to be positive, because distance is always positive. But for this exercise, we have to find the lines that separate the red balls and blue balls. So, for this particular case, we will not use the absolute distance. If the distance is positive, then the balls are in one side else if the distance is negative, then the balls are in other side of the line.

Further learning: here from brilliant.org.

[20]:
```
class LineSeparation(object):
    """
    This class finds which line separates the red and the blue balls.
    """

    def __init__(self, reds, blues, lines):
        self.reds = reds
        self.blues = blues
        self.lines = lines

    def pl_d(self, p, l) -> float:
        """
        This method computes the distance between a point and a line.
        """
        from math import sqrt
        import re
        matches = re.findall(pattern='[\d?.-]+', string=l)
        a, b, c = list(map(float, matches))
        x0, y0 = p
        return (a*x0 + b*y0 + c) / (sqrt(a**2 + b**2))

    def is_separating(self) -> dict:
        """
        This method validates whether the line separates the points or not.
        """
        line_dict = dict.fromkeys(self.lines, None)
        for l in self.lines:
            red_ds = [True if self.pl_d(p=rp, l=l) > 0 else False
                      for rp in self.reds]
            blue_ds = [True if self.pl_d(p=bp, l=l) < 0 else False
                       for bp in self.blues]
            line_dict[l] = all(red_ds) and all(blue_ds)
        return line_dict
```

[21]:
```
reds = [(1, 1), (2, 1), (4, 2), (2, 4), (-1, 4)]
blues = [(-2, -1), (-1, -2), (-3, -2), (-3, -1), (1,-3)]
lines = ['1x+1y+0','1x-1y+0','1x+0y-3','0x+1y-0.5']
```

[22]:
```
ls = LineSeparation(reds=reds, blues=blues, lines=lines)
print(ls.is_separating())
```

```
{'1x+1y+0': True, '1x-1y+0': False, '1x+0y-3': False, '0x+1y-0.5': True}
```

Q7) Filling the missing values in the specified format.

Your program reads a string for example "_,_,x,_,_,_" and returns the filled sequence.

I: _,_,_,24

```
O: 6,6,6,6
I: 40,_,_,_,60
O: 20,20,20,20,20
I: 80,_,_,_,_
O: 16,16,16,16,16
I: _,_,30,_,_,_,50,_,_
O: 10,10,12,12,12,12,4,4,4
```

[23]:
```python
class SmoothHepler(object):
    """
    This class is a helper for smoothing problem.
    """

    def __init__(self, string):
        self.string = string.split(',')
        self.output = []

    def get_proper_input(self) -> list:
        """
        This method obtains a proper input list for special case.
        """
        for i, char in enumerate(self.string):
            if char.isdigit():
                self.output.append(','.join(self.string[:i+1]))
                self.string = self.string[i+1:]
                return self.get_proper_input()
            else:
                self.output.append(','.join(self.string[:i+1]))
        remove_ = lambda x: x[-1].isdigit()
        if self.output[-1][-1].isdigit():
            self.output = list(filter(remove_, self.output))
        else:
            B = [self.output[-1]]
            A = list(filter(remove_, self.output[:-1]))
            self.output = A + B
        if self.output[0].isdigit():
            self.output[0] = self.output.pop(0) + ',' + self.output[0]
        return self.output
```

[24]:
```python
strings = ["_,_,30,_,_,_,50,_,_","_,_,30,_,_,_,50", "30,_,_,_,50,_,_"]
```

[25]:
```python
for string in strings:
    s_helper = SmoothHepler(string=string)
    print(s_helper.get_proper_input())
```

```
['_,_,30', '_,_,_,50', '_,_']
['_,_,30', '_,_,_,50']
['30,_,_,_,50', '_,_']
```

I will use the `SmoothHepler` class inside `SmoothCurve` class, to solve the special case of the problem. Honestly this question is so interesting, it took me some hours to solve it.

[26]:
```python
class SmoothCurve(object):
    """
    This class fills the blanks.
    """

    def __init__(self):
        self.solution = list()

    def pick_numbers(self, s)-> list:
        """
        This method picks the numbers in the string.
        """
        import re
        return list(map(int, re.findall(pattern='[\d]+', string=s)))

    def batches(self, s) -> list:
        """
        This method gets the set of blanks in the string.
        """
        import re
        return re.findall(pattern='[_,]+', string=string)

    def smoothing(self, string) -> str:
        """
        This method smooths the curve, by filling the blank spaces.
        """
        batches = self.batches(s=string)
        if len(batches) == 1:
            sol = self.ordinary(string=string)
            return ','.join(sol)
        elif len(batches) > 1:
            s_helper = SmoothHepler(string=string)
            p_input = s_helper.get_proper_input()
            sol = self.special(p_input=p_input)
            return ','.join(sol)
        else:
            return None

    def ordinary(self, string) -> list:
        """
        This method smooths ordinary set of problems.
        """
        size = len(string.split(','))
        nums = self.pick_numbers(s=string)
```

```
        total = sum(nums)
        return list(map(str, [total // size] * size))

    def special_helper(self, string) -> tuple:
        """
        This method is helps self.special(string).
        """
        ord_sol = self.ordinary(string=string)
        return ord_sol.pop(), ord_sol

    def special(self, p_input) -> list:
        """
        This method smooths special set of problems.
        """
        if len(p_input) == 1:
            ord_sol = self.ordinary(p_input[0])
            self.solution.append(','.join(ord_sol))
        else:
            p_str = p_input.pop(0)
            l_ord, ord_sol = self.special_helper(string=p_str)
            self.solution.append(','.join(ord_sol))
            p_input[0] = l_ord + ',' + p_input[0]
            self.special(p_input=p_input)
        return self.solution
```

[27]:
```
strings = ["_,_,_,24", "40,_,_,_,60", "80,_,_,_,_", "_,_,30,_,_,_,50,_,_"]
```

[28]:
```
smooth_it = SmoothCurve()

for string in strings:
    print("I: {}".format(string))
    print("O: {}".format(smooth_it.smoothing(string=string)))
```

```
I: _,_,_,24
O: 6,6,6,6
I: 40,_,_,_,60
O: 20,20,20,20,20
I: 80,_,_,_,_
O: 16,16,16,16,16
I: _,_,30,_,_,_,50,_,_
O: 10,10,12,12,12,12,4,4,4
```

Q8) Compute conditional probabilities.

You will be given a list of lists, each sublist will be of length 2 i.e. `[[x, y], [p, q], [l, m], ..., [r, s]]` consider its like a martrix of n rows and 2 columns.

- The 1st column F will contain only 5 uniques values `(F1, F2, F3, F4, F5)`.

- The 2nd column S will contain only 3 uniques values (`S1, S2, S3`).

```
I: [[F1, S1], [F2, S2], [F3, S3], [F1, S2], [F2, S3],
    [F3, S2], [F2, S1], [F4, S1], [F4, S3], [F5, S1]]
O:
a. P(F=F1|S==S1) = 1/4, P(F=F1|S==S2) = 1/3, P(F=F1|S==S3) = 0/3
b. P(F=F2|S==S1) = 1/4, P(F=F2|S==S2) = 1/3, P(F=F2|S==S3) = 1/3
c. P(F=F3|S==S1) = 0/4, P(F=F3|S==S2) = 1/3, P(F=F3|S==S3) = 1/3
d. P(F=F4|S==S1) = 1/4, P(F=F4|S==S2) = 0/3, P(F=F4|S==S3) = 1/3
e. P(F=F5|S==S1) = 1/4, P(F=F5|S==S2) = 0/3, P(F=F5|S==S3) = 0/3
```

[29]:
```python
class ConditionalProbability(object):
    """
    This class finds the conditional probabilities.
    """

    def __init__(self, AB_space):
        self.AB_space = AB_space
        self.A_space = set(map(lambda x: x[0], self.AB_space))
        self.B_space = set(map(lambda x: x[1], self.AB_space))
        print("A: {}".format(self.A_space))
        print("B: {}".format(self.B_space))

    def total_Bi(self):
        """
        This method gets the total Bi's.
        """
        all_Bis = list(map(lambda x: x[1], self.AB_space))
        B_hist = dict()
        for b in all_Bis:
            B_hist.update({b: B_hist.get(b, 0) + 1})
        return B_hist

    def do_conditional_probability(self):
        """
        This method computes conditional probability.
        """
        solution = list()
        for Ai in self.A_space:
            Ai_Bi = list()
            for Bi in self.B_space:
                num = self.AB_space.count([Ai, Bi])
                den = self.total_Bi()[Bi]
                Ai_Bi.append(f'P({Ai}|{Bi}) = {num}/{den}')
            solution.append(Ai_Bi)
        return solution
```

[30]:
```
AB_space = [['F1', 'S1'], ['F2', 'S2'], ['F3', 'S3'], ['F1', 'S2'], ['F2',␣
↪'S3'],
           ['F3', 'S2'], ['F2', 'S1'], ['F4', 'S1'], ['F4', 'S3'], ['F5',␣
↪'S1']]
```

[31]:
```
cp = ConditionalProbability(AB_space=AB_space)
solution = cp.do_conditional_probability()
for sol in solution:
    print(sol)
```

```
A: {'F4', 'F3', 'F5', 'F2', 'F1'}
B: {'S1', 'S3', 'S2'}
['P(F4|S1) = 1/4', 'P(F4|S3) = 1/3', 'P(F4|S2) = 0/3']
['P(F3|S1) = 0/4', 'P(F3|S3) = 1/3', 'P(F3|S2) = 1/3']
['P(F5|S1) = 1/4', 'P(F5|S3) = 0/3', 'P(F5|S2) = 0/3']
['P(F2|S1) = 1/4', 'P(F2|S3) = 1/3', 'P(F2|S2) = 1/3']
['P(F1|S1) = 1/4', 'P(F1|S3) = 0/3', 'P(F1|S2) = 1/3']
```

Q9) Given two sentances $S1$, $S2$ your task is to find common words.

Find:

- Number of common words between $S1$, $S2$.
- Words in $S1$ but not in $S2$.
- Words in $S2$ but not in $S1$.

[32]:
```
class Commonality(object):
    """
    This class finds the common friends.
    """

    def __init__(self, S1, S2):
        self.S1 = set(list(S1.split()))
        self.S2 = set(list(S2.split()))

    def solution(self) -> None:
        """
        This methods provides the solution to problem 9.
        """
        print("a. {}".format(len(self.S1.intersection(self.S2))))
        print("b. {}".format(list(self.S1.difference(self.S2))))
        print("c. {}".format(list(self.S2.difference(self.S1))))
        return None
```

[33]:
```
S1 = "The first column F will contain only 5 unique values"
S2 = "The second column S will contain only 3 unique values"
```

[34]:
```
comm = Commonality(S1=S1, S2=S2)
comm.solution()
```

```
a. 7
b. ['F', '5', 'first']
c. ['3', 'S', 'second']
```

Q10) Compute log **loss.**

Given $Y = (y_1, y_2, y_3, \ldots, y_n)$ and $Y' = (y'_1, y'_2, y'_3, \ldots, y'_n)$, then

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(y'_i) + (1 - y_i) \log(1 - y'_i)$$

LogLoss is also called as Binary Cross Entropy.

[35]:
```
class LogLoss(object):
    """
    This class computes the LogLoss function.
    """

    def __init__(self, Y_Yhat):
        self.Y_Yhat = Y_Yhat
        self.n = len(Y_Yhat)

    def compute_logloss(self) -> float:
        """
        This method computes LogLoss function.
        """
        from math import log10
        logloss = 0
        for y, yhat in self.Y_Yhat:
            logloss += (y * log10(yhat)) + ((1-y) * log10(1-yhat))
        return round(((-logloss) / self.n), 7)
```

[36]:
```
Y_Yhat = [[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3],
          [0, 0.6], [1, 0.1], [1, 0.9], [1, 0.8]]
```

[37]:
```
logloss = LogLoss(Y_Yhat=Y_Yhat)
print(logloss.compute_logloss())
```

```
0.4243099
```

End of the file.