Problem Set 1

CED18I039 - Paleti Krishnasai

(1)

Given the following setup {Class, Tally score, Frequency}, develop an application that generates the table shown; (you can populate the relevant data; minimum data size :50 records). The table is only an illustration for a data of color scores, you are free to test the application over any data set with the application generating the tally and frequency scores.

In [1]:
```
import random
```

In [2]:
```
def tally(value):
    five = '||||\ '
    tally = five * int(value/5) + '|' * (value%5)
    return tally
```

In [3]:
```
def countdata(data):
    frequncy={}
    frequncy['1']=0
    frequncy['2']=0
    frequncy['3']=0
    frequncy['other']=0
    for i in data:
        if i in [1,2,3]:
            frequncy[str(i)] += 1
        else:
            frequncy['other'] += 1
    return frequncy
```

In [4]:
```
data=[]
tot=50
for i in range(tot):
    data.append(random.randint(1,6))
colors=['1','2','3','other']
frequency = countdata(data)

for j in colors:
    print(j + "\t"+tally(frequency[j])+"\t\t"+str(frequency[j]) )
```

```
1       ||||            4
2       ||||\ ||||\             10
3       ||||\ ||||\             10
other   ||||\ ||||\ ||||\ ||||\ ||||\ |         26
```

---

(2)

In a class of 18 students, assume marks distribution in an exam are as follows. Let the roll numbers start with CSE20D01 and all the odd roll numbers secure marks as follows: 25+((i+7)%10) and even roll numbers : 25+((i+8)%10). Develop an application that sets up the data and calculate the mean and median for the marks obtained using the platform support.

In [5]:
```
def generatedata(students,rolltemplate):
    marks=[]
    rollno = []
    for i in range(1,students+1):
        zero = '0' * (len(str(students))-len(str(i)))
        rollno_temp = rolltemplate + zero + str(i)
        mark_temp = 0
        if i%2 == 0:
            mark_temp = 25 +((i+8)%10)
        else:
            mark_temp = 25 +((i+7)%10)
        marks.append(mark_temp)
        rollno.append(rollno_temp)
    return marks,rollno
```

In [6]:
```
def mean(marks):
    sum = 0
    for mark in marks:
        sum += mark
    return sum/len(marks)

def median(marks):
    sorted_marks = sorted(marks)
    if len(sorted_marks)%2==1:
        return sorted_marks[int((len(sorted_marks)-1)/2)]
    else:
        return (sorted_marks[int(len(sorted_marks)/2)] + sorted_marks[int(ler
```

In [7]:
```
students = int(input("Enter no of students: "))
rolltemplate = 'CSE20D'
marks,rollno = generatedata(students,rolltemplate)
print("DATA")
a=0
for mark,rollno in zip(marks,rollno):
    print(str(a+1)+":"+rollno+"--",mark)
    a+=1
print("MEAN: ",mean(marks))
print("MEDIAN: ",median(marks))
```

```
Enter no of students: 25
DATA
1:CSE20D01-- 33
2:CSE20D02-- 25
3:CSE20D03-- 25
4:CSE20D04-- 27
5:CSE20D05-- 27
6:CSE20D06-- 29
7:CSE20D07-- 29
8:CSE20D08-- 31
9:CSE20D09-- 31
10:CSE20D10-- 33
11:CSE20D11-- 33
12:CSE20D12-- 25
```

```
13:CSE20D13-- 25
14:CSE20D14-- 27
15:CSE20D15-- 27
16:CSE20D16-- 29
17:CSE20D17-- 29
18:CSE20D18-- 31
19:CSE20D19-- 31
20:CSE20D20-- 33
21:CSE20D21-- 33
22:CSE20D22-- 25
23:CSE20D23-- 25
24:CSE20D24-- 27
25:CSE20D25-- 27
MEAN:  28.68
MEDIAN:  29
```

---

(3)

For a sample space of 20 elements, the values are fitted to the line Y=2X+3, X>5. Develop an application that sets up the data and computes the standard deviation of this sample space. (use random number generator supported in your development platform to generate values of X).

In [8]:

```
def GenerateData(data, MaxVal):
    X = []
    Y = []
    for i in range(data):
        x = random.randint(6, Maxvalue)
        y = (2 * x) + 3
        X.append(x)
        Y.append(y)
    return X, Y

def Mean(X):
    sum = 0
    for x in X:
        sum += x
    return sum / len(X)

def STDDEV(X):
    SD = 0.0
    mean = Mean(X)
    numerator = 0.0
    for x in X:
        numerator += (x - mean) ** 2
    SD = (numerator / len(X)) ** (0.5)
    return SD
```

In [9]:

```
data = int(input("Enter no of students: "))
Maxvalue = 100
X,Y = GenerateData(data,Maxvalue)
print("DATA")
a=0
for x,y in zip(X,Y):
    print(str(a+1)+":(",x,",",y,")")
    a+=1
print("STDDEV of X: ",STDDEV(X))
print("STDDEV of Y: ",STDDEV(Y))
```

```
Enter no of students: 40
DATA
1:( 67 , 137 )
2:( 77 , 157 )
3:( 54 , 111 )
4:( 59 , 121 )
5:( 98 , 199 )
6:( 66 , 135 )
7:( 78 , 159 )
8:( 84 , 171 )
9:( 78 , 159 )
10:( 23 , 49 )
11:( 13 , 29 )
12:( 28 , 59 )
13:( 23 , 49 )
14:( 82 , 167 )
15:( 51 , 105 )
16:( 45 , 93 )
17:( 91 , 185 )
18:( 83 , 169 )
19:( 18 , 39 )
20:( 16 , 35 )
21:( 21 , 45 )
22:( 88 , 179 )
23:( 8 , 19 )
24:( 29 , 61 )
25:( 39 , 81 )
26:( 64 , 131 )
27:( 14 , 31 )
28:( 60 , 123 )
29:( 77 , 157 )
30:( 49 , 101 )
31:( 19 , 41 )
32:( 43 , 89 )
33:( 44 , 91 )
34:( 84 , 171 )
35:( 94 , 191 )
36:( 11 , 25 )
37:( 78 , 159 )
38:( 41 , 85 )
39:( 66 , 135 )
40:( 63 , 129 )
STDDEV of X:  27.016245112894573
STDDEV of Y:  54.03249022578915
```

---

(4)

For a given data of heights of a class, theheights of 15 students are recorded as 167.65, 167, 172, 175, 165, 167, 168, 167, 167.3, 170, 167.5, 170, 167, 169, and 172. Develop an application that computes; explore if there are any packages supported in your platform that depicts these measures / their calculations of central tendency in a visual form for ease of understanding.

- a. Mean height of the student
- b. Median and Mode of the sample space
- c. Standard deviation
- d. Measure of skewness. [(Mean-Mode)/standard deviation]

In [10]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [11]:
```
X_list = list(range(1,16))
Y_list = [167.65, 167, 172, 175, 165, 167, 168, 167, 167.3, 170, 167.5, 170,
```

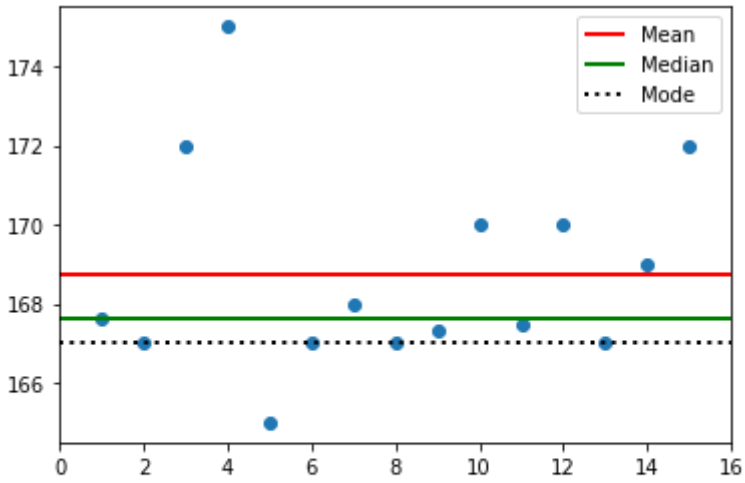
In [12]:
```
dataset = pd.DataFrame({"X":X_list,"Y":Y_list})
dataset
```

Out[12]:

	X	Y
0	1	167.65
1	2	167.00
2	3	172.00
3	4	175.00
4	5	165.00
5	6	167.00
6	7	168.00
7	8	167.00
8	9	167.30
9	10	170.00
10	11	167.50
11	12	170.00
12	13	167.00
13	14	169.00
14	15	172.00

In [13]:
```
plt.xlim(0,16)
plt.scatter(dataset['X'],dataset['Y'])
mean=np.mean(Y_list)
median=np.median(Y_list)
mode=dataset['Y'].mode()[0] # returns a series of descending values, hence ch
stddev=dataset['Y'].std()
skew= (mean-mode)/stddev

plt.hlines(mean,0,16,color='red',linestyle='solid',label='Mean',linewidth=2)
plt.hlines(median,0,16,color='green',linestyle='solid',label='Median',linewic
plt.hlines(mode,0,16,color='black',linestyle='dotted',label='Mode',linewidth=
plt.text(0,160,"Mean = {:.2f} \nMedian = {:.2f} \nMode = {:.2f} \nStandard De
plt.legend()
plt.show()
#single cell produces overlapped lines along with scatter, multiple cells pro
```



Mean = 168.76
Median = 167.65
Mode = 167.00
Standard Deviation = 2.61
Measure of Skewness = 0.68

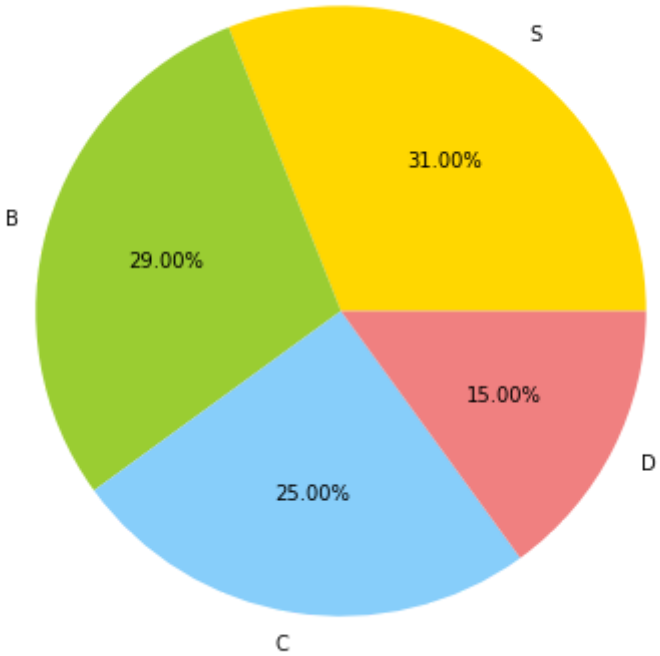
---

(5)

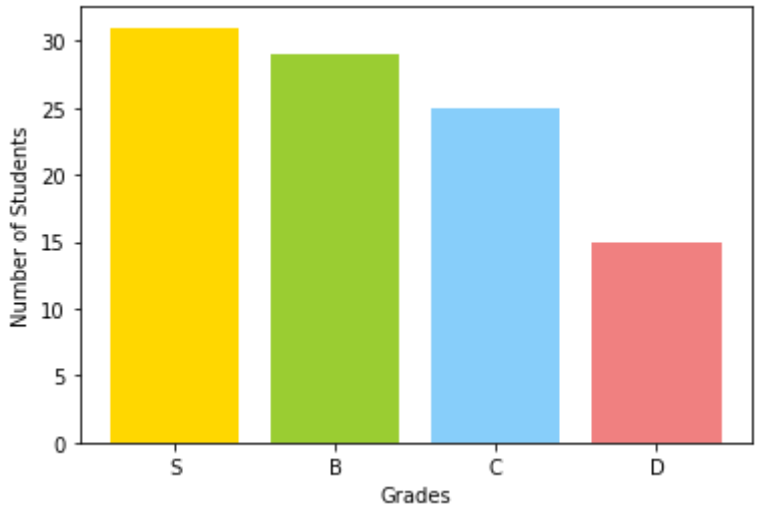
In Analytics and Systems of Bigdata course, for a class of 100 students, around 31 students secured ‘S’ grade, 29 secured ‘B’ grade, 25 ‘C’ grades, and rest of them secured ‘D’ grades. If the range of each grade is 15 marks. (S for 85 to 100 marks, A for 70 to 85 …). Develop an application that represents the above data: using Pie and Bar graphs.

In [14]:

```
grades = ["S","B","C","D"]
students = [31,29,25,15]
fig = plt.figure(figsize =(10, 7))
plt.pie(students,labels=grades,colors=["gold","yellowgreen","lightskyblue","l
plt.show()
```



In [15]:
```
plt.bar(grades,students,color=["gold","yellowgreen","lightskyblue","lightcora
plt.xlabel("Grades")
plt.ylabel("Number of Students")
plt.show()
```

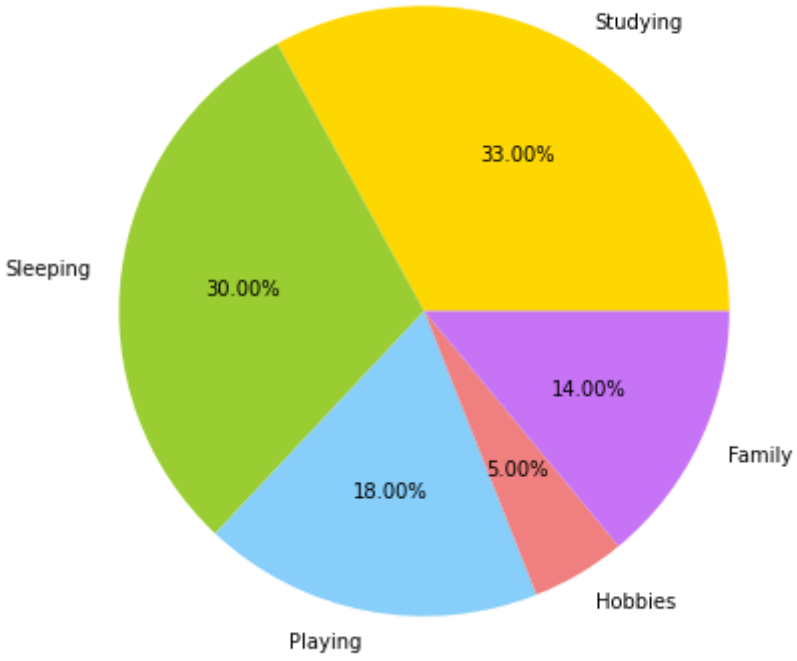


---

(6)

On a given day (average basis), a student is observed to spend 33% of time in studying, 30% in sleeping, 18% in playing, 5% for hobby activities, and rest for spending with friends and family. Plot a pie chart showing his daily activities.

In [16]:
```
Schedule = ["Studying","Sleeping","Playing","Hobbies","Family"]
Timeshare = [33,30,18,5,14]
fig = plt.figure(figsize =(10, 7))
plt.pie(Timeshare,labels=Schedule,colors=["#FFD700","#9acd32","#87cefa","#f08
plt.show()
```



---

(7)

Develop an application (absolute grader) that accepts marks scored by 20 students in ASBD course (as a split up of three: Mid Sem (30), End Sem (50) and Assignments(20). Compute the total and use it to grade the students following absolute grading: >=90 – S ; >=80 – A and so on till D. Compute the Class average for total marks in the course and 50% of class average would be fixed as the cut off for E. Generate a frequency table for the grades as well (Table displaying the grades and counts of them).

- Maroon shows failure (" U grade "); similar to a heatmap...cold to warm.
- The color scheme is automatic and the least grade ends up with red-maroon shade

In [52]:
```
Midterm = [random.randint(0,30) for i in range(20)]
Endterm = [random.randint(0,50) for i in range(20)]
Assignment = [random.randint(0,20) for i in range(20)]
Marks = np.array([Midterm[i]+Endterm[i]+Assignment[i] for i in range(20)])
lettergrade = ['S','A','B','C','D','E','U']

def AbsGrader(mark,avg):
    if(mark>=90):
        return('S')
    elif(mark>=80):
        return('A')
    elif(mark>=70):
        return('B')
    elif(mark>=60):
        return('C')
    elif(mark>=50):
        return('D')
    elif(mark>=(avg/2)):
        return('E')
    else:
        return('U')

Grades=[]
for i in range(20):
    Grades.append(AbsGrader(Marks[i],Marks.mean()))

frequency={}
frequency['S']=0
frequency['A']=0
frequency['B']=0
frequency['C']=0
frequency['D']=0
frequency['E']=0
frequency['U']=0

for i in Grades:
    if i in lettergrade:
        frequency[str(i)] += 1
```

In [53]:
```
frequency #testing purpose
```

Out[53]:
```
{'S': 0, 'A': 1, 'B': 3, 'C': 5, 'D': 2, 'E': 6, 'U': 3}
```

In [54]:

```
dataset = pd.DataFrame(frequency.items(),columns=['Grade', 'Frequency'])
dataset
```

Out[54]:

	Grade	Frequency
0	S	0
1	A	1
2	B	3
3	C	5
4	D	2
5	E	6
6	U	3

In [55]:

```
rolllist = pd.DataFrame({"Marks":Marks,"Grade":Grades})
rolllist.style.background_gradient(cmap="Spectral")
```

Out[55]:

	Marks	Grade
0	67	C
1	48	E
2	24	U
3	63	C
4	71	B
5	26	U
6	65	C
7	51	D
8	75	B
9	84	A
10	43	E
11	68	C
12	24	U
13	65	C
14	75	B
15	44	E
16	55	D
17	35	E
18	45	E
19	44	E

---

(8)

Extend the application developed in (7) to support relative grading which uses the class average (mean) and standard deviation to compute the cutoffs for various grades as opposed to fixing

them statically; you can refer the sample grader (excel sheet) attached to understand the formulas for fixing the cutoffs; the grader would involve, mean, standard deviation, max mark, passed students data mean, etc. Understand the excel grader thoroughly before you try mimicking such an application in your development platform.

Formulas Required for Relative Grading:

- Passing Minimum: 50% of class average. (Minimum marks for passing)
- X= Passing Students' Mean- Passing Minimum.
- S_cutoff = Max_Mark – 0.1 *(Max_Mark-Passing Students Mean)
- Y= S_cutoff – Passing Students Mean
- A_cutoff = Passing Students Mean + Y * (5/8)
- B_cutoff = Passing Students Mean + Y * (2/8)
- C_cutoff = Passing Students Mean - X * (2/8)
- D_cutoff = Passing Students Mean - X * (5/8)
- E_cutoff = Passing Minimum

- Maroon shows failure (" U grade "); similar to a heatmap...cold to warm.
- The color scheme is automatic and the least grade ends up with red-maroon shade

In [56]:
```
Mean = Marks.mean()
PassingMin = Mean/2
PassingMarks = Marks[Marks>PassingMin]
PassingMean = PassingMarks.mean()
MaxMark = Marks.max()
```

In [57]:
```
X = PassingMean - PassingMin
S_cutoff = MaxMark - 0.1*(MaxMark-PassingMean)
Y = S_cutoff - PassingMean
A_cutoff = PassingMean + Y*(5/8)
B_cutoff = PassingMean + Y*(2/8)
C_cutoff = PassingMean + Y*(2/8)
D_cutoff = PassingMean + Y*(5/8)
E_cutoff = PassingMin
```

In [58]:
```
#Relative Grader
def RelGrader(mark):
    if(mark>=S_cutoff):
        return('S')
    elif(mark>=A_cutoff):
        return('A')
    elif(mark>=B_cutoff):
        return('B')
    elif(mark>=C_cutoff):
        return('C')
    elif(mark>=D_cutoff):
        return('D')
    elif(mark>=E_cutoff):
        return('E')
    else:
        return('U')

Grades_rel=[]
for i in range(20):
    Grades_rel.append(RelGrader(Marks[i]))
```

In [59]:
```
frequency_rel={}
frequency_rel['S']=0
frequency_rel['A']=0
frequency_rel['B']=0
frequency_rel['C']=0
frequency_rel['D']=0
frequency_rel['E']=0
frequency_rel['U']=0

for i in Grades_rel:
    if i in lettergrade:
        frequency_rel[str(i)] += 1
```

In [60]:
```
frequency_rel #testing purpose
```

Out[60]:
```
{'S': 1, 'A': 2, 'B': 5, 'C': 0, 'D': 0, 'E': 9, 'U': 3}
```

In [61]:
```
dataset1 = pd.DataFrame(frequency_rel.items(),columns=['Grade_rel', 'Frequenc
dataset1
```

Out[61]:

	Grade_rel	Frequency
0	S	1
1	A	2
2	B	5
3	C	0
4	D	0
5	E	9
6	U	3

In [62]:
```
rolllist1 = pd.DataFrame({"Marks":Marks,"Grade":Grades_rel})
rolllist1.style.background_gradient(cmap="Spectral")
```

Out[62]:

	Marks	Grade
0	67	B
1	48	E
2	24	U
3	63	E
4	71	B
5	26	U
6	65	B
7	51	E
8	75	A
9	84	S
10	43	E
11	68	B

	Marks	Grade
12	24	U
13	65	B
14	75	A
15	44	E
16	55	E
17	35	E
18	45	E
19	44	E

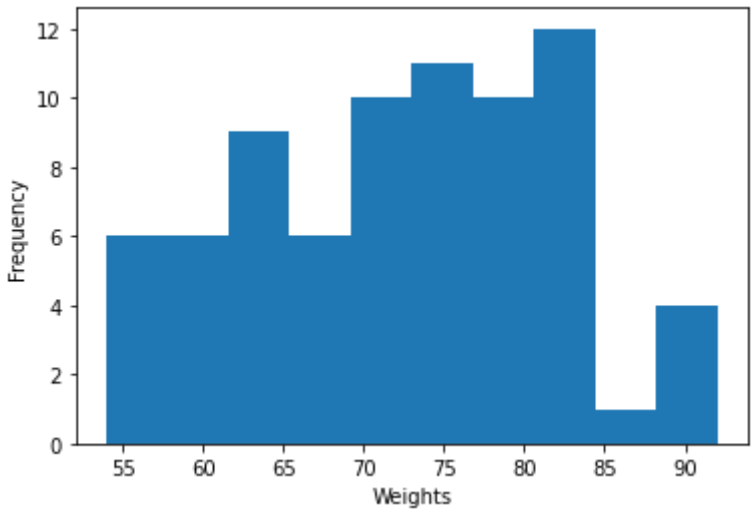
---

(9)

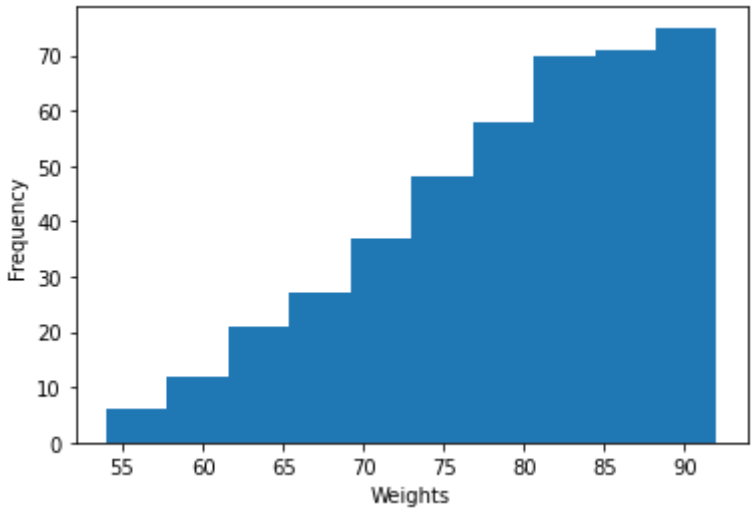
Consider the following sample of weights for 45 individuals: 79 71 89 57 76 64 82 82 67 80 81 65 73 79 79 60 58 83 74 68 78 80 78 81 76 65 70 76 58 82 59 73 72 79 87 63 74 90 69 70 83 76 61 66 71 60 57 81 57 65 81 78 77 81 81 63 71 66 56 62 75 64 74 74 70 71 56 69 63 72 81 54 72 91 92. For the above data generates histograms and depict them using packages in your platform. Explore the different types of histograms available and test drive the types supported in your platform

In [46]:
```
weights = [79,71,89,57,76,64,82,82,67,80 ,81, 65, 73, 79, 79, 60, 58, 83, 74,
```

In [47]:
```
# default histogram
plt.hist(weights)
plt.xlabel("Weights")
plt.ylabel("Frequency")
plt.show()
```

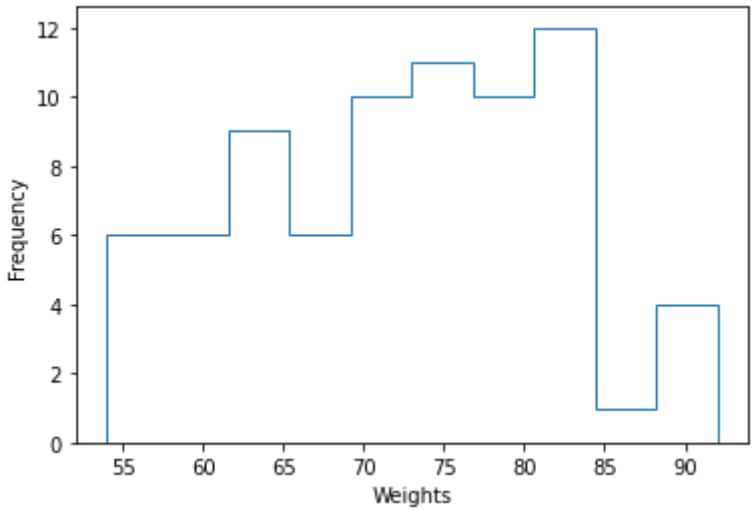


In [48]:
```
# cumulative Histogram
plt.hist(weights,cumulative=True)
plt.xlabel("Weights")
plt.ylabel("Frequency")
plt.show()
```



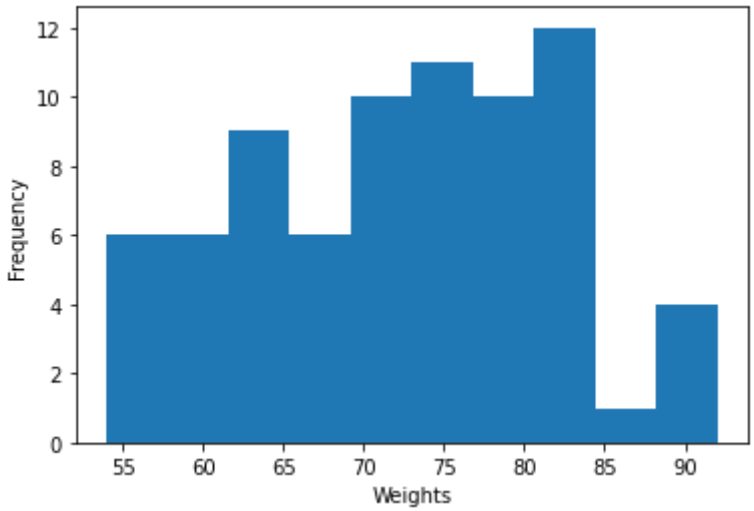
In [49]:

```
# step histogram
plt.hist(weights,histtype="step")
plt.xlabel("Weights")
plt.ylabel("Frequency")
plt.show()
```



In [50]:

```
# barstacked histogram ( same as default in this case )
plt.hist(weights,histtype="barstacked")
plt.xlabel("Weights")
plt.ylabel("Frequency")
plt.show()
```



In [51]:

```
# stepfilled horizontal(orientation can be changed for all the above histogra
plt.hist(weights,histtype="stepfilled",orientation='horizontal')
plt.xlabel("Weights")
plt.ylabel("Frequency")
plt.show()
```