Statistics

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

In [22]:
```
sb.set_palette('husl')
sb.set_style('darkgrid')
```

In [23]:
```
smartphones = pd.read_csv('c://smartphones.csv')
```

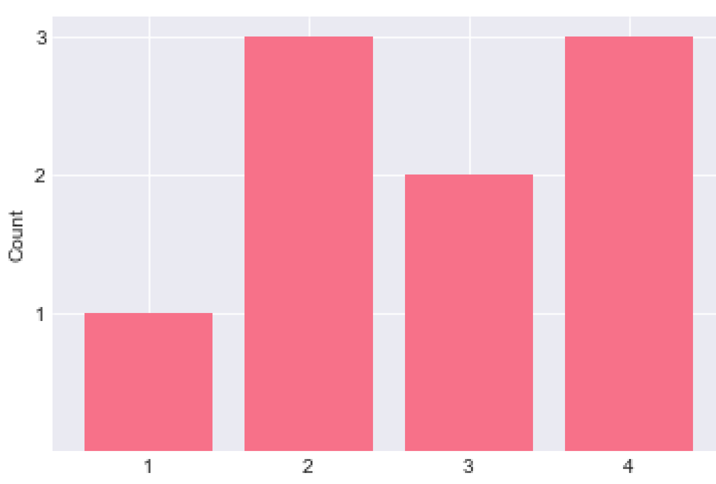
In [4]:
```
print(smartphones)
count = smartphones.Ram.value_counts()
```

```
         Name       OS  Capacity  Ram  Weight    Company  inch
0  Galaxy S8  Android        64    4   149.0    Samsung   5.8
1  Lumia 950  windows        32    3   150.0  Microsoft   5.2
2  Xpreia L1  Android        16    2   180.0       Sony   5.5
3   iphone 7      ios       128    2   138.0      Apple   4.7
4    U Ultra  Android        64    4   170.0        HTC   5.7
5  Galaxy S5  Android        16    2   145.0    Samsung   5.1
6  iphone 5s      ios        32    1   112.0      Apple   4.0
7    Moto G5  Android        16    3   144.5   Motorola   5.0
8      Pixel  Android       128    4   143.0     Google   5.0
```

BarPlot

In [7]:
```
category = count.index
```

In [24]:
```
plt.bar(category, count)
plt.xlabel('Ram')
plt.ylabel('Count')
plt.xticks([1, 2, 3, 4])
plt.yticks([1, 2, 3])
plt.show()
```



With bar charts, each column represents a group defined by a categorical variable; and with histograms, each column represents a group defined by a continuous, quantitative variable.

ECDF

> Empirical cumulative distribution function

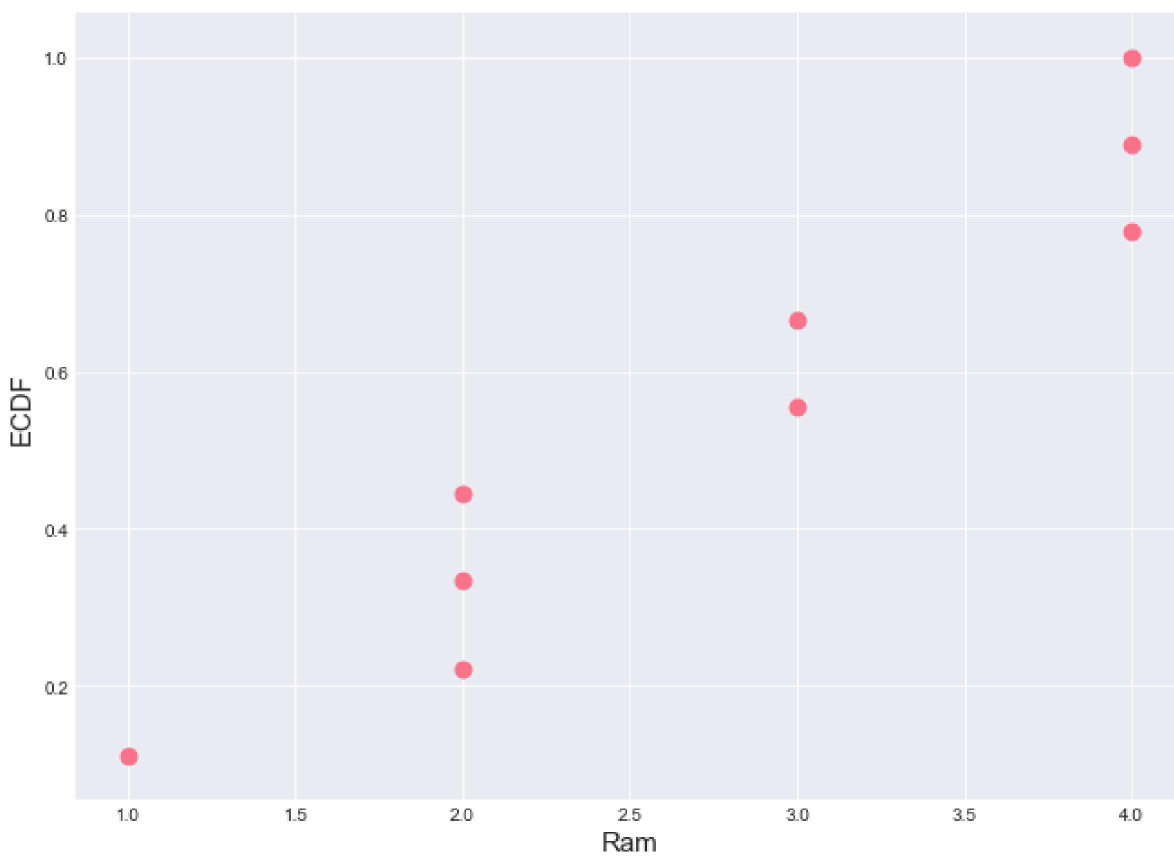
In [468]:
```
def ECDF(data):
    n = len(data)     # Number of data point
    x = np.sort(data)     # x-data for ECDF
    y= np.arange(1, n+1) / n     # y-data for ECDF
    return x, y
```

In [476]:
```
np.arange(1,10+1) / 10
```

Out[476]:
```
array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ])
```

In [10]:
```
x, y = ECDF(smartphones.Ram)
```

In [25]:
```
plt.figure(figsize=(11, 8))
plt.scatter(x, y, s=80)
plt.margins(0.05)
plt.xlabel('Ram', fontsize=15)
plt.ylabel('ECDF', fontsize=15)
plt.show()
```



Mean

In [141]:
```
np.mean(smartphones.inch)
```

Out[141]:
```
5.1111111111111107
```

Median

In [142]:
```
np.median(smartphones.inch)
```

Out[142]:
```
5.0999999999999996
```

Percentile

In [143]:
```
np.percentile(smartphones.inch,[25 ,50, 75])
```
Out[143]:
```
array([ 5. ,  5.1,  5.5])
```

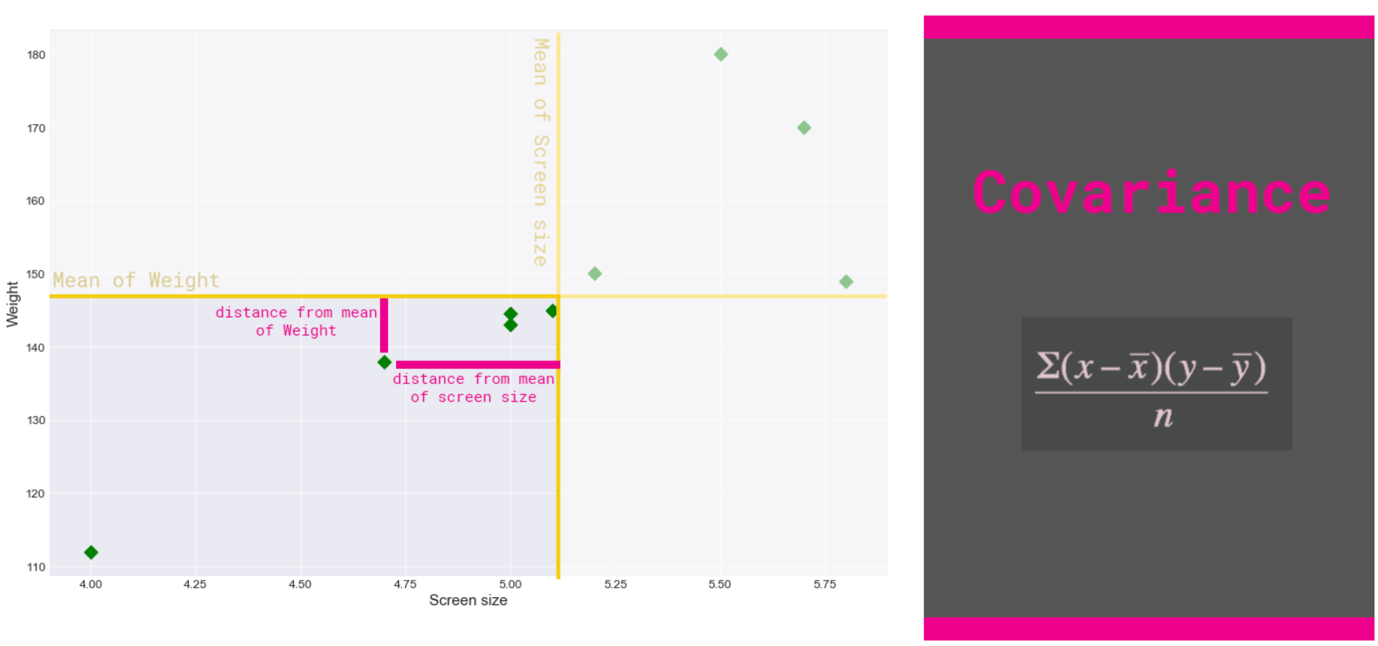
Variance and Standard Deviation

In [84]:
```
print('mean is {:.2f} and the variance is {:.2f}'.format(np.mean(smartphones.Ram), np.var(smartphones.Ram)))
```
```
mean is 2.78 and the variance is 1.06
```

In [85]:
```
diff = smartphones.Ram - np.mean(smartphones.Ram)
diff_sq = diff ** 2
var_exp = np.mean(diff_sq)
var_exp
```
Out[85]:
```
1.0617283950617284
```

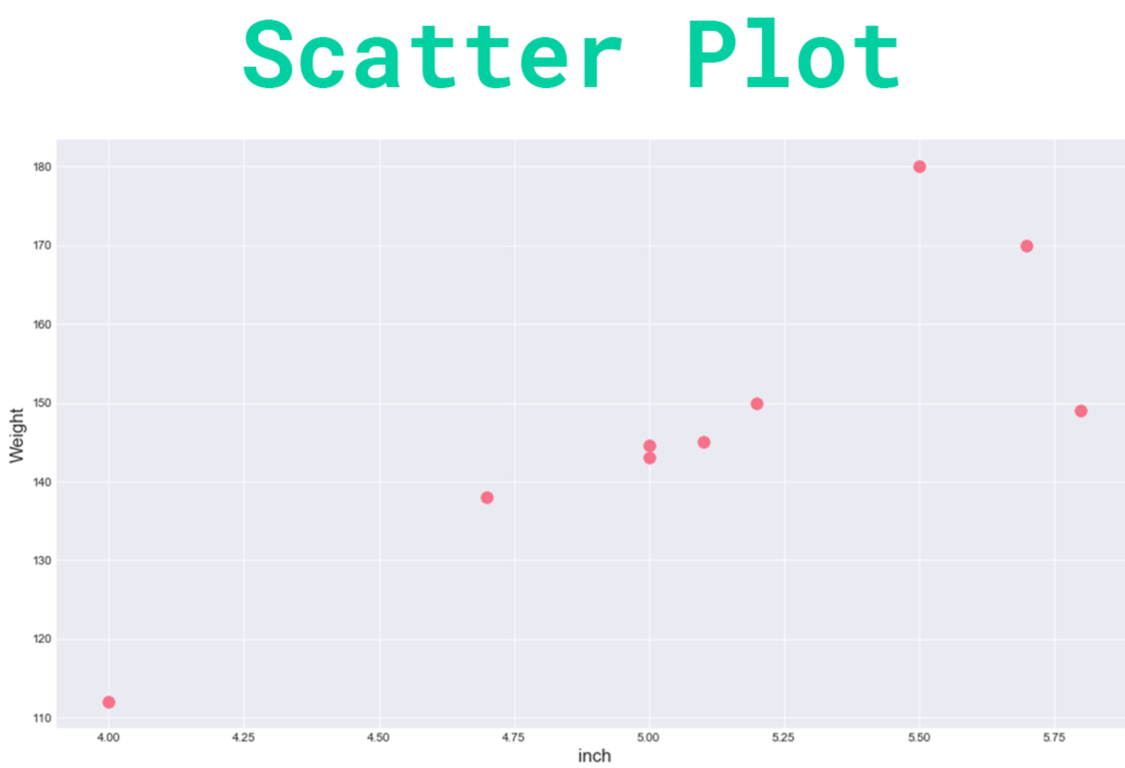
In [88]:
```
np.std(smartphones.Ram)
```
Out[88]:
```
1.0304020550550783
```

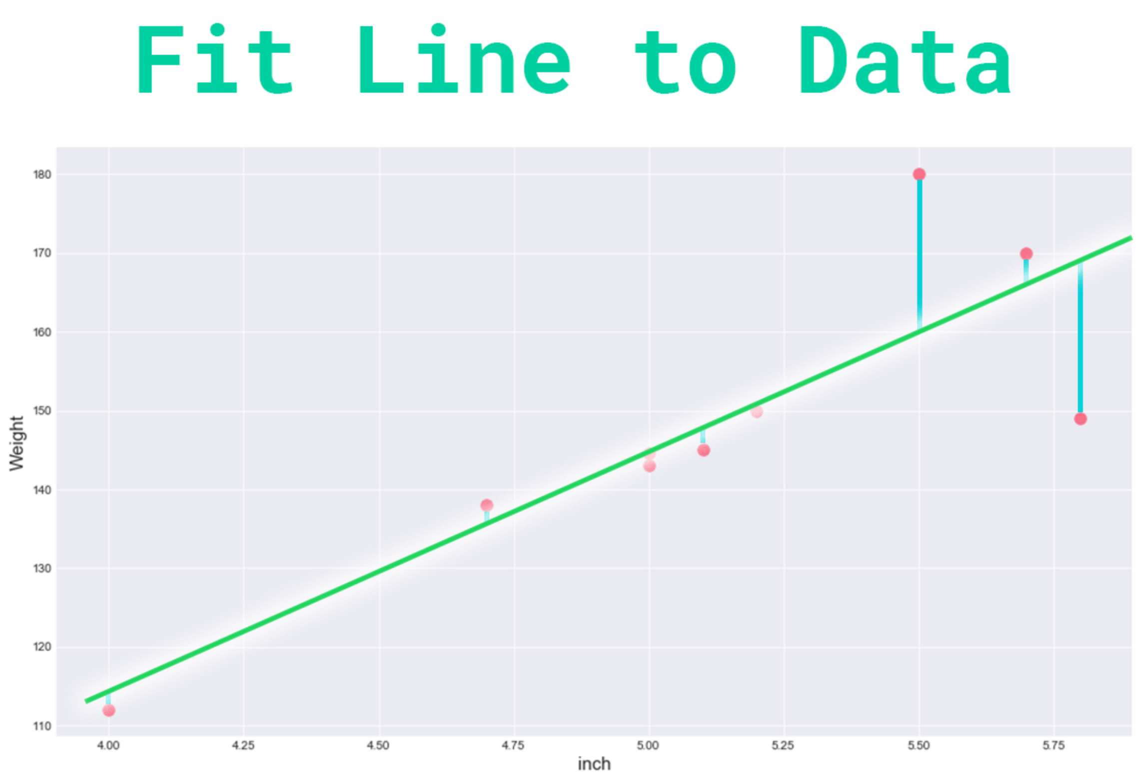
Covariance



```
In [29]: np.cov(smartphones.inch, smartphones.Weight)

Out[29]: array([[  3.01111111e-01,   8.91944444e+00],
                [  8.91944444e+00,   3.69402778e+02]])
```

Correlation





0 < Positive Correlation < 1

Zero Correlation = 0

-1 < Negative Correlation < 0

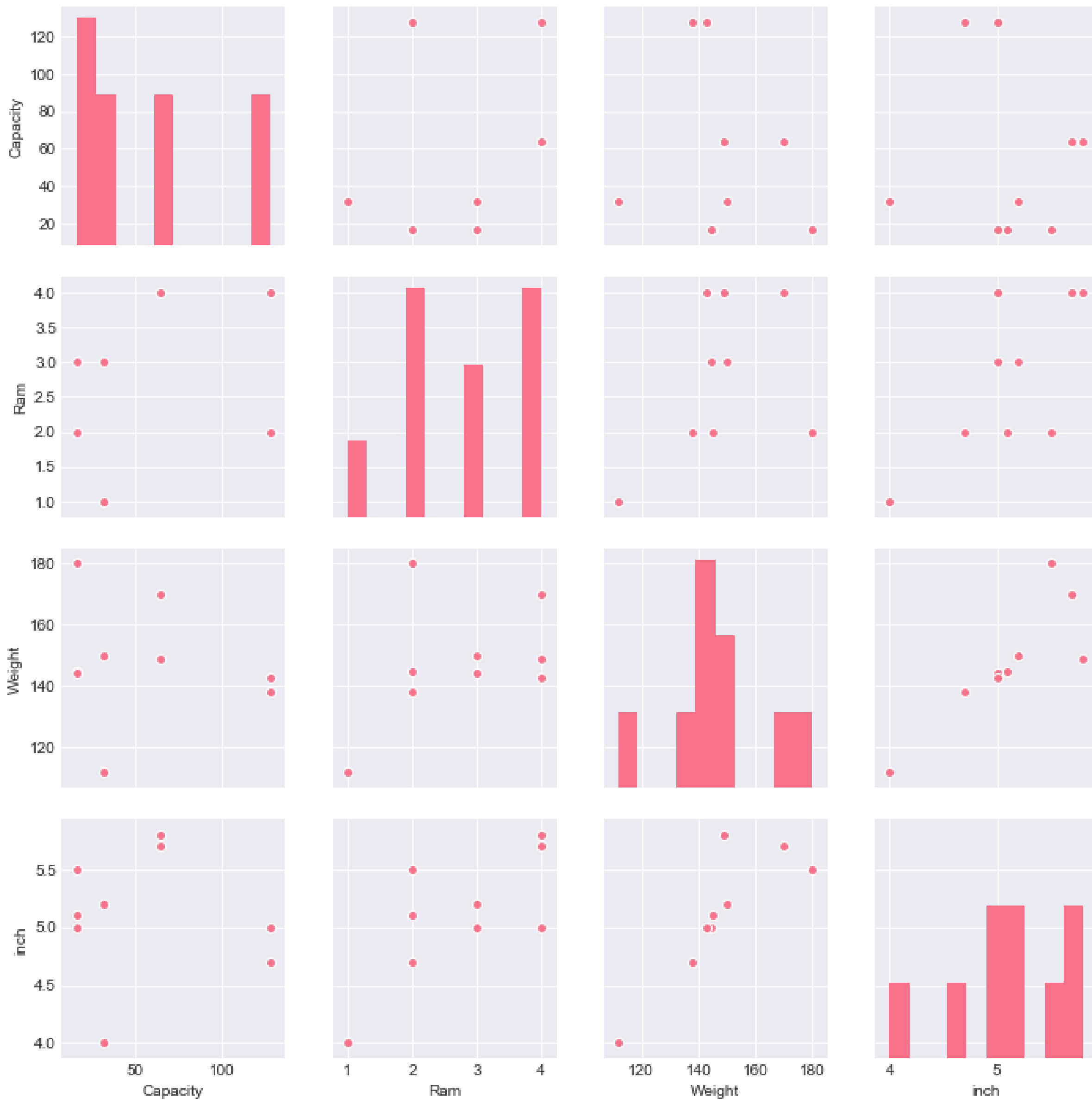
Quality of fit

Parametric Methods

Pearson Correlation

$$\text{Pearson Correlation} = \frac{\text{Covariance}}{(\text{std of } x)\,(\text{std of } y)}$$

In [139]:
```
sb.pairplot(smartphones)
plt.show()
```



Using scipy for calculate the pearson correlation coefficient

In [463]:
```
from scipy.stats.stats import pearsonr
pearson_coefficent, p_value = pearsonr(smartphones.Weight, smartphones.inch)
pearson_coefficent
```

Out[463]: 0.84571558837054206

Using pandas for calculate the pearson correlation coefficient

In [456]:
```
num_var = smartphones.drop(['Name','OS','Capacity', 'Ram', 'Company'], axis=1)
num_var
corr = num_var.corr()
corr
```

Out[456]:

	Weight	inch
0	149.0	5.8
1	150.0	5.2
2	180.0	5.5
3	138.0	4.7
4	170.0	5.7
5	145.0	5.1
6	112.0	4.0
7	144.5	5.0
8	143.0	5.0

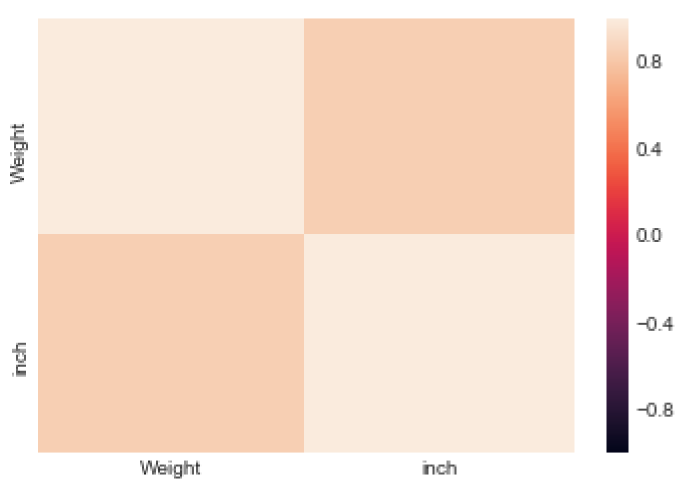
Out[456]:

	Weight	inch
Weight	1.000000	0.845716
inch	0.845716	1.000000

seaborn for visualize pearson correlation coefficient

In [455]:
```
sb.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, vmin=-1, vmax=1)
plt.show()
```

Out[455]: <matplotlib.axes._subplots.AxesSubplot at 0x23eccc43080>



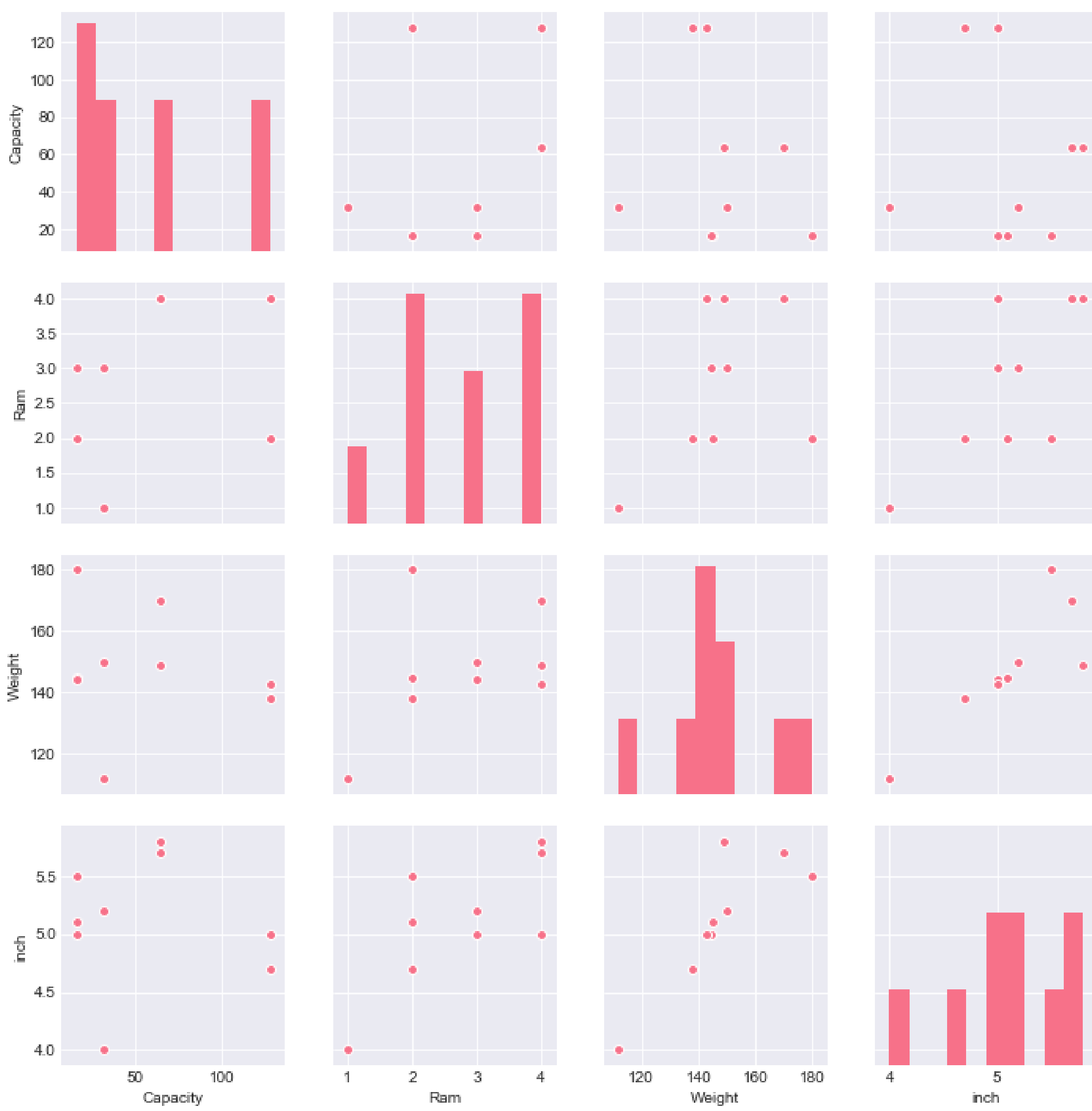
Nonparametric Methods ¶

Spearman's rank correlation

- Ordinal Variable (categorical-numeric)
- Non-normaly distributed
- Nonlinear related

In [458]:
```
sb.pairplot(smartphones)
plt.show()
```

Out[458]: <seaborn.axisgrid.PairGrid at 0x23ecbd414e0>



In [460]:
```
cat_var = smartphones.drop(['Name','OS', 'Weight', 'inch','Company'], axis=1)
```

In [369]:
```
from scipy.stats import spearmanr
```

In [459]:
```
spearmanr_coefficent, p_value = spearmanr(cat_var.Capacity, cat_var.Ram)
print('spearman rank correlation is ', spearmanr_coefficent)
```

```
spearman rank correlation is  0.441981903329
```

Chi-square test

```
null hypothesis : H0
alternative hypothesis : H1
```

P-value > 0.05

P-value < 0.05

Total = 50

Observed					
	16GB	32GB	64GB	128GB	
Android	4	7	9	6	26
ios	5	5	4	2	16
windows	3	3	2	0	8
	12	15	15	8	

Expected				
	16GB	32GB	64GB	128GB
Android				
ios				
windows				

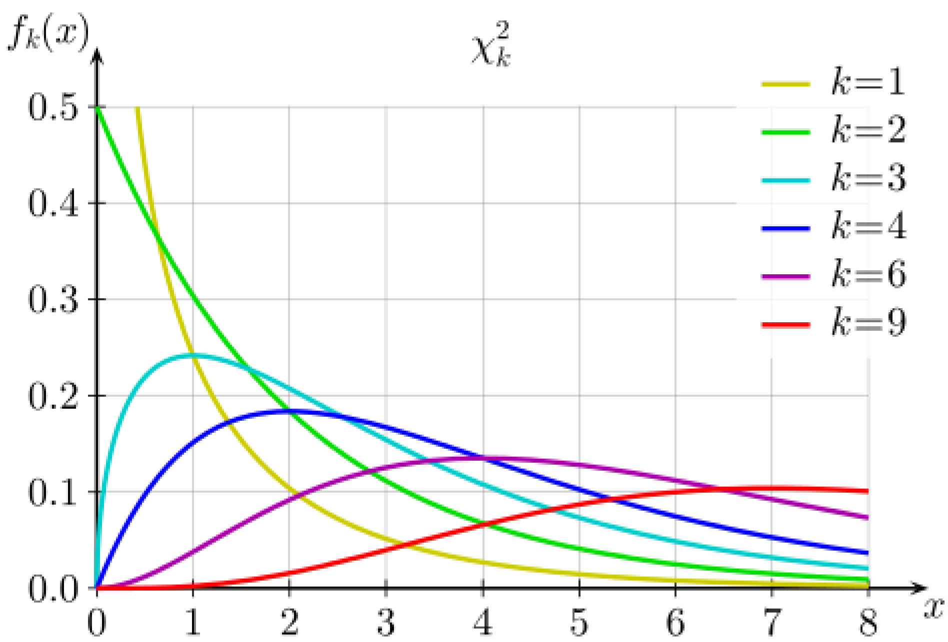
$$\mathbf{E}\,(\text{Android},\ 16\text{GB}) = \frac{(\text{Androids}).(16\text{GB})}{\text{Total}}$$

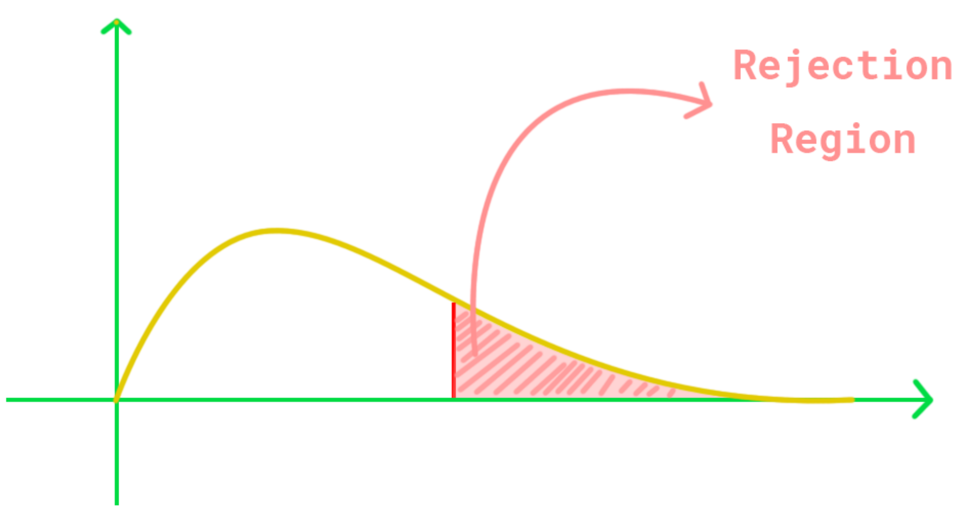
Expected				
	16GB	32GB	64GB	128GB
Android	6.24			
ios				
windows				

$$\mathbf{E}\,(\text{Android},\ 16\text{GB}) = \frac{(\text{Androids}).(16\text{GB})}{\text{Total}} = \frac{26 * 12}{50} = 6.24$$

Expected				
	16GB	32GB	64GB	128GB
Android	6.24	7.8	7.8	4.16
ios	3.84	4.8	4.8	2.56
windows	1.92	2.4	2.4	1.28

$$E\,(\text{Android, 16GB}) = \frac{(\text{Androids}).(\text{16GB})}{\text{Total}} = \frac{26 * 12}{50} = 6.24$$



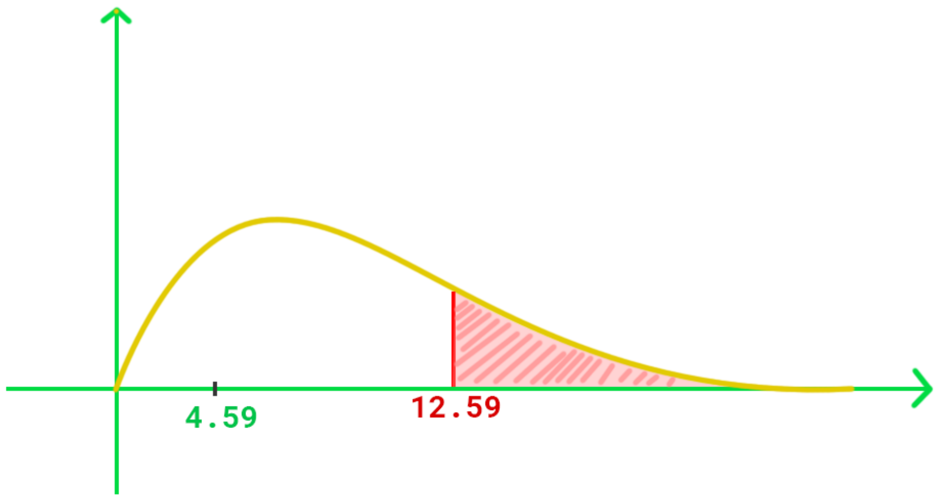


Degrees of freedom (df)	χ^2 value[19]									
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.63	10.83
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.60	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.82	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88
10	3.94	4.87	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59
P value (Probability)	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001

$$\chi^2 = \mathrm{SUM}\left[\frac{(O - E)^2}{E}\right]$$

$$\frac{(4-6.24)^2}{6.24} + \frac{(7-7.8)^2}{7.8} + \frac{(9-7.8)^2}{7.8} + \frac{(6-4.16)^2}{4.16} + \frac{(5-3.84)^2}{3.84} + \frac{(5-4.8)^2}{4.8} + \frac{(4-4.8)^2}{4.8} + \frac{(2-2.56)^2}{2.56} + \frac{(3-1.92)^2}{1.92} + \frac{(3-2.4)^2}{2.4} + \frac{(2-2.4)^2}{2.4} + \frac{(0-1.28)^2}{1.28}$$

$$= 4.59$$



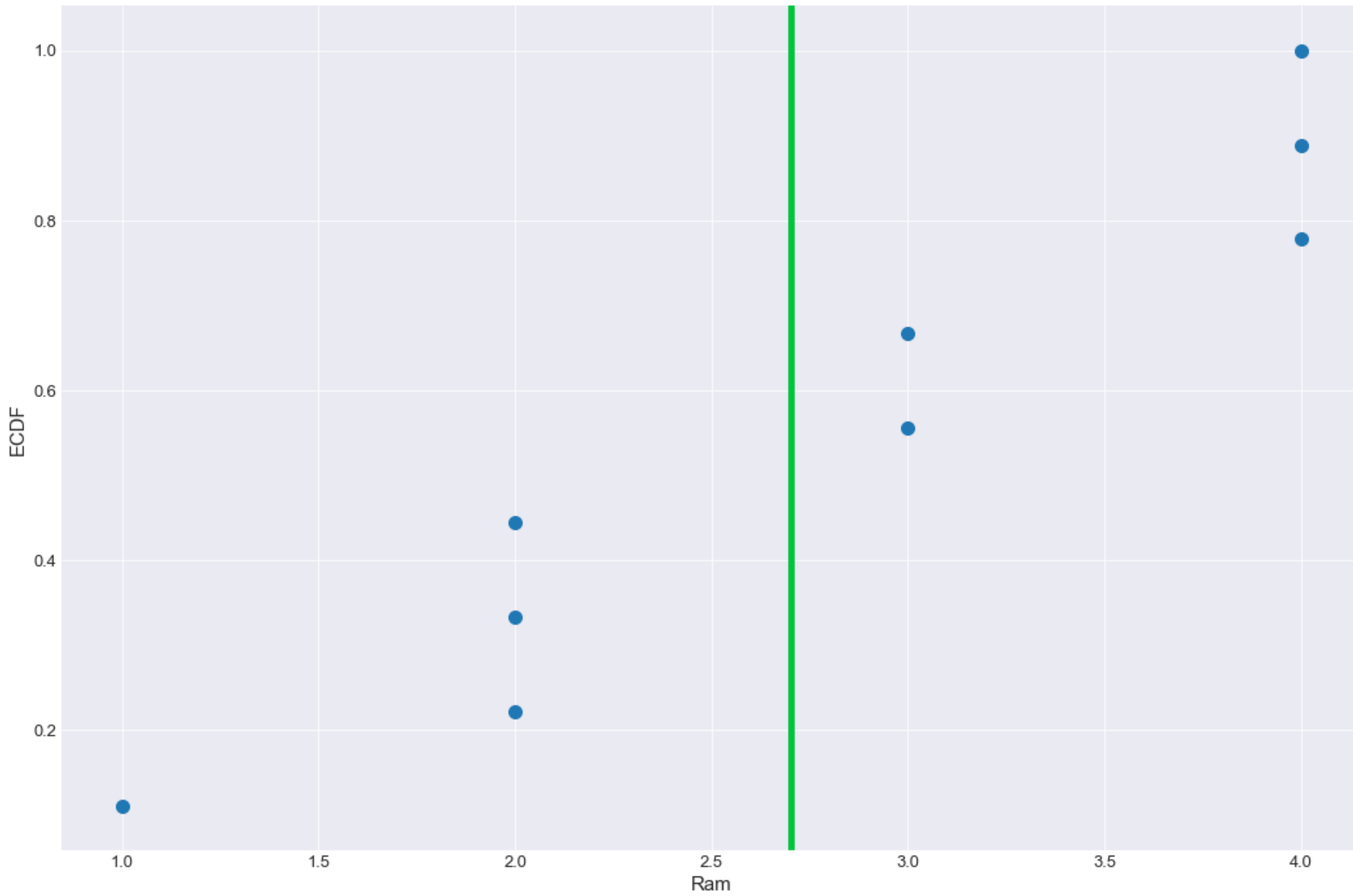

```
In [477]: from scipy.stats import chi2_contingency
```

```
In [489]: table = pd.crosstab(cat_var.Capacity, cat_var.Ram)
          chi2, p_value, dof, expected = chi2_contingency(table.values)
          chi2
          p_value
          dof
          expected
```

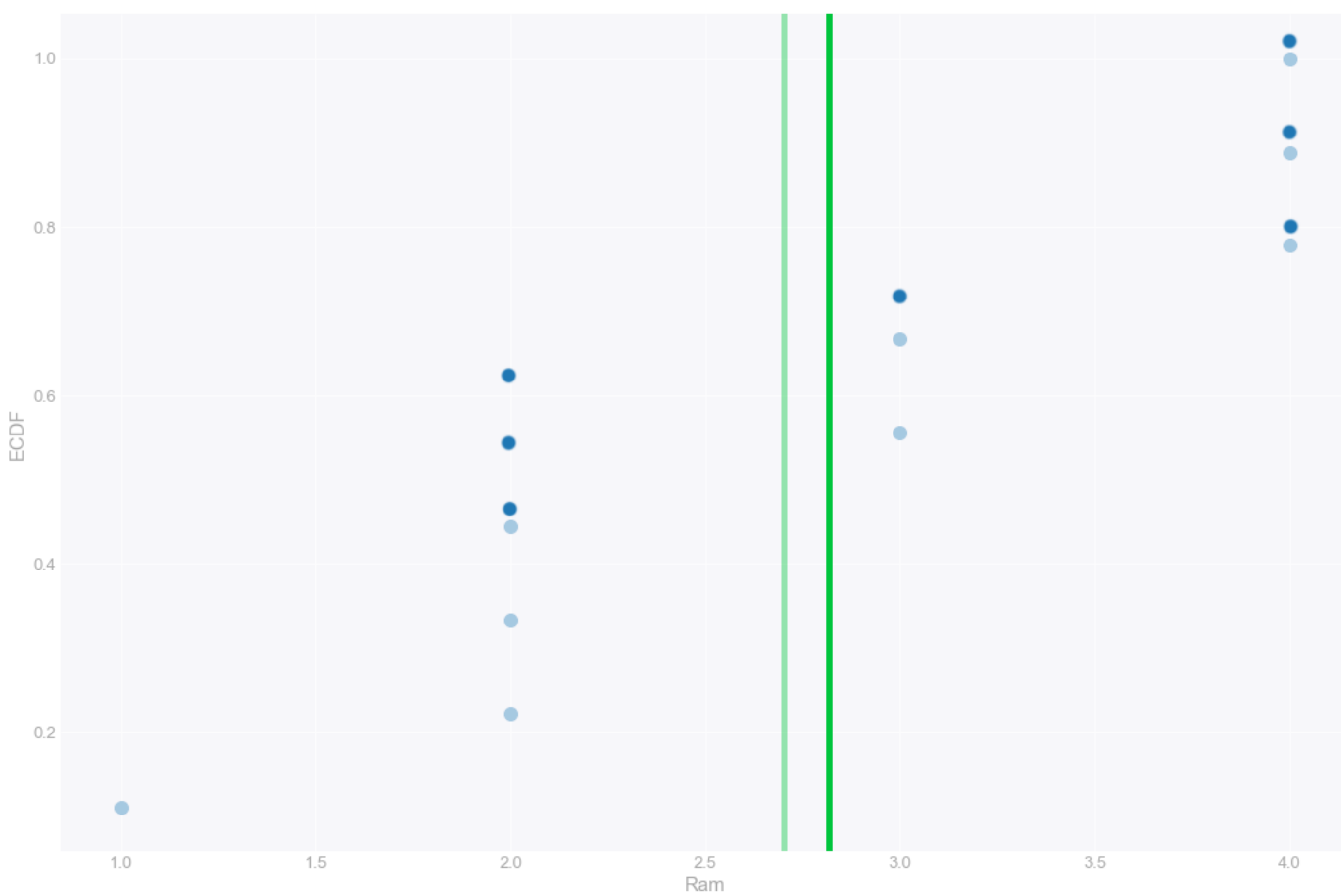
```
Out[489]: 12.25

Out[489]: 0.19957963261092318

Out[489]: 9

Out[489]: array([[ 0.33333333,  1.        ,  0.66666667,  1.        ],
                 [ 0.22222222,  0.66666667,  0.44444444,  0.66666667],
                 [ 0.22222222,  0.66666667,  0.44444444,  0.66666667],
                 [ 0.22222222,  0.66666667,  0.44444444,  0.66666667]])
```

Probability

Mean for 9 Smartphones

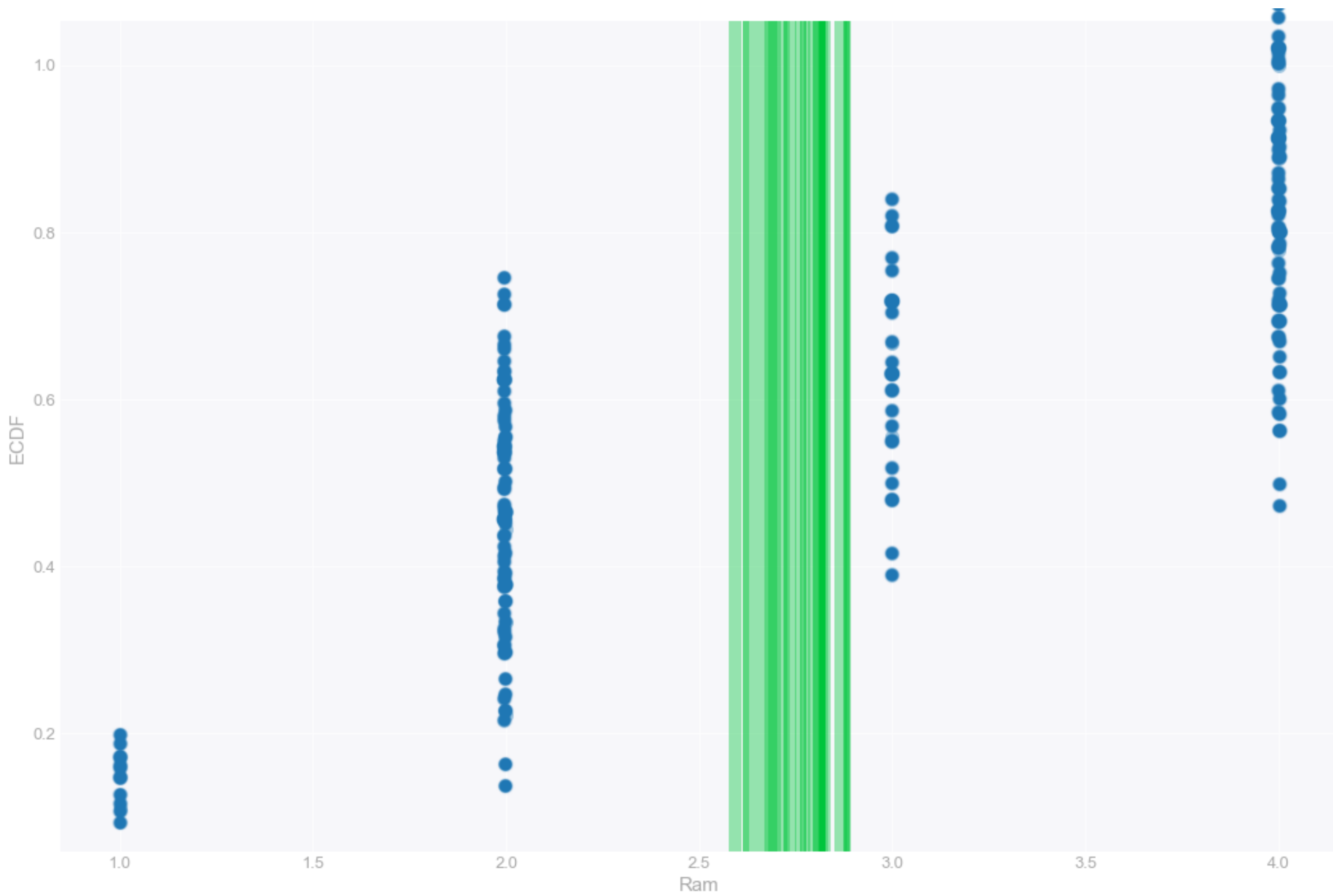


mean for another dataset



now, after analysis some dataset , we have a range that help us predict, mean

your data speaks with probabilty language



Pseudo Random number Generator

seed: seed() = random generator uses of formula that use seed to generate a random number.this is why we say psudo random number generator. sets the random seed, so that your results are the reproducible between simulations. As an argument, it takes an integer of your choosing. If you call the function, no output will be generated.
random float : rand() = : if you don't specify any arguments, it generates a random float between zero and one.
random integer : randint(start,stop-not included) create randome integer

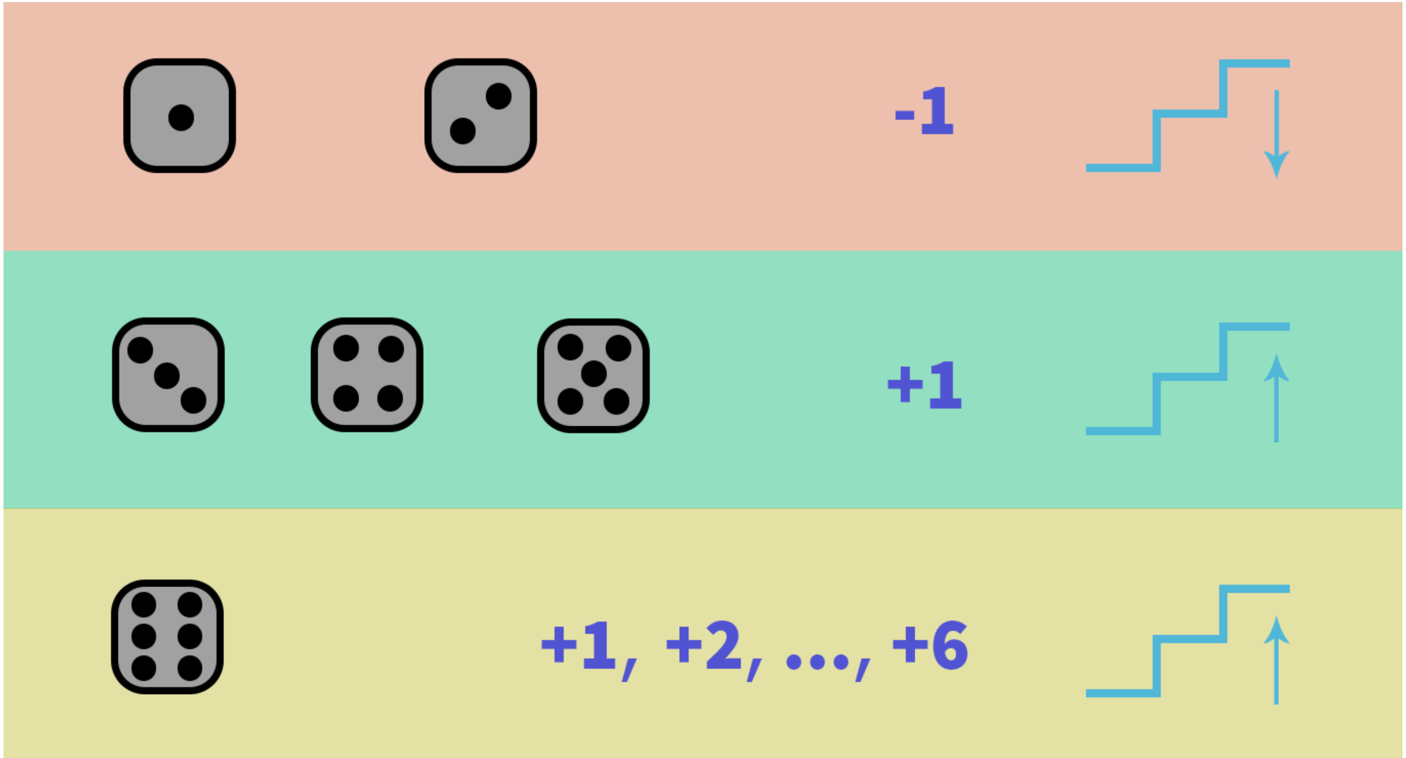
In [68]:
```
np.random.seed(42)
rand_num = np.random.random(3)     #randint(1,10)
win = ran_num < 0.5
print(win)
print(rand_num[win])
```

```
[ True False False]
[ 0.37454012]
```

In [102]:
```
num_trial = 1000
rand_num = np.random.random(size=num_trial)
win = rand_num > 0.5
num_win = np.sum(win)

print(num_win/num_trial)
```

```
0.518
```

** A GAME **



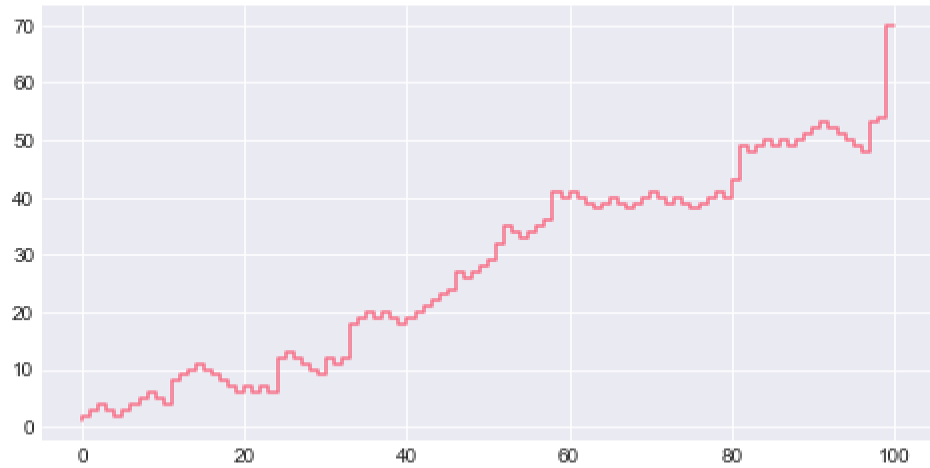
In [490]:
```
step = 0

dice = np.random.randint(1,7)

if dice <3 :
    step = max(0, step - 1)
elif dice<=5 :
    step = step + 1
else :
    num = np.random.randint(1,7)
    step = step + num

print('dice is {} and you are in step {}'.format(dice,step))
```

```
dice is 4 and you are in step 1
```

Random Walk

In [ ]:
```
step = 0
rand_walk = np.empty(0)

for i in range (100):
    dice = np.random.randint(1,7)

    if dice <3 :
        step = max(0, step - 1)
    elif dice<=5 :
        step = step + 1
    else :
        num = np.random.randint(1,7)
        step = step + num
    rand_walk = np.append(rnd_walk, step)
    print('dice is {} and you are in step {}'.format(dice,step))
```

In [121]:
```
plt.figure(figsize=(8,4))
plt.step(np.arange(101),rand_walk)
plt.show()
```



Distribution

(one time trial-->n time trial(random walk)-->m category of n time trial(distribution)
) what is the chance of reach 60 step or upper in 100 itteration?

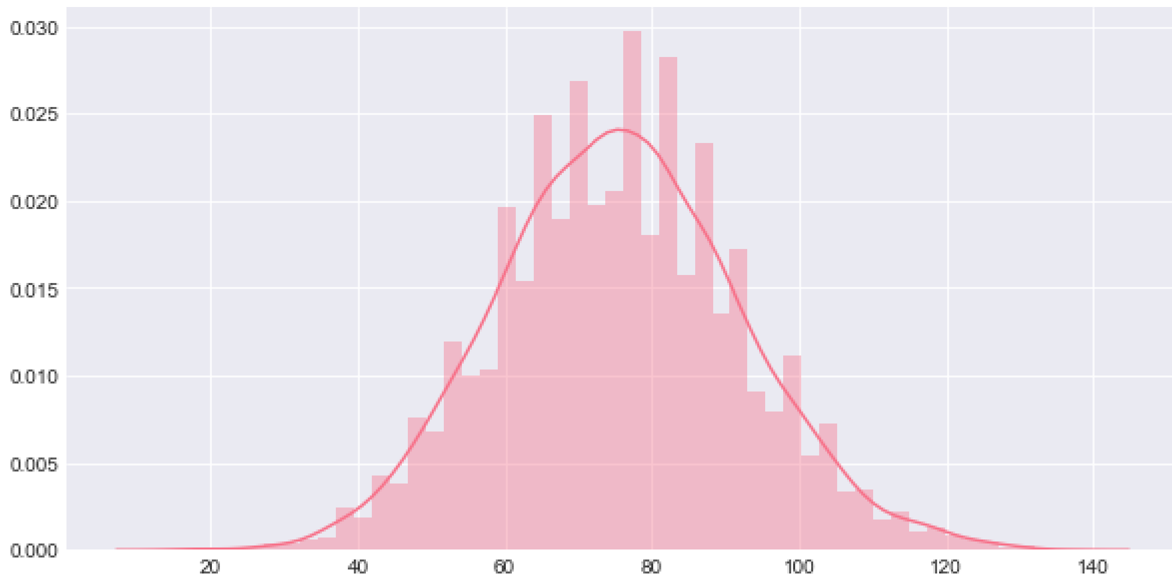
In [134]:
```
dis = np.empty(0)
np.random.seed(20)
for n_rndWalk in range(10000) :
    step = 0
    rnd_walk = np.empty(0)

    for n_dice in range (100):
        dice = np.random.randint(1,7)

        if dice <3 :
            step = max(0, step - 1)
        elif dice<=5 :
            step = step + 1
        else :
            num = np.random.randint(1,7)
            step = step + num
    dis = np.append(dis, step)

dis.size
```

Out[134]: 10000

```
In [178]: plt.figure(figsize=(10,5))
          sb.distplot(dis)
          plt.show()
```

```
Out[178]: <matplotlib.figure.Figure at 0x23ebf903eb8>
```

```
Out[178]: <matplotlib.axes._subplots.AxesSubplot at 0x23ebf426f98>
```



```
In [136]: np.mean(dis>60)
```

```
Out[136]: 0.81799999999999995
```

Normal Distribution

```
In [ ]: samples = np.random.normal(20,1,size=100000)
```

```
In [177]: sb.distplot(samples)
          plt.show()
```

```
Out[177]: <matplotlib.axes._subplots.AxesSubplot at 0x23ebf27d3c8>
```