In [2]:

```python
# Ex 1a Calculating the simple probabilities
import numpy as np
import pandas as pd

# Import the dataset
df = pd.read_csv("Book1.csv")
print(df)

#Extract a Paticular Column
pen = df['Book'].tolist()
print(pen)

#Average
#print('Average:',np.average(pen))

#Mean
print('Mean:',np.mean(pen))

# Variance
print('Variance:',np.var(pen))

#Standard Deviation
print('Standard Deviation:',np.std(pen))
```

```
    Months   Pen  Book  Marker  Chair  Table  Pen Stand  Total Units  \
0        1  2500  1500    5200   9200   1200       1500        21100   
1        2  2630  1200    5100   6100   2100       1200        18330   
2        3  2140  1340    4550   9550   3550       1340        22470   
3        4  3400  1130    5870   8870   1870       1130        22270   
4        5  3600  1740    4560   7760   1560       1740        20960   
5        6  2760  1555    4890   7490   1890       1555        20140   
6        7  2980  1120    4780   8980   1780       1120        20760   
7        8  3700  1400    5860   9960   2860       1400        25180   
8        9  3540  1780    6100   8100   2100       1780        23400   
9       10  1990  1890    8300  10300   2300       1890        26670   
10      11  2340  2100    7300  13300   2400       2100        41280   
11      12  2900  1760    7400  14400   1800       1760        30020   

    TotalProfit  
0        211000  
1        183300  
2        224700  
3        222700  
4        209600  
5        201400  
6        207600  
7        251800  
8        234000  
9        266700  
10       412800  
11       300200  
[1500, 1200, 1340, 1130, 1740, 1555, 1120, 1400, 1780, 1890, 2100, 1760]
Mean: 1542.9166666666667
Variance: 91960.24305555555
Standard Deviation: 303.2494732980678
```

In [17]:

```
# Ex 2 Probability distributions- Binomial Distribution

from scipy.stats import binom
# setting the values
# of n and p
n = 6
p = 0.6
# defining the list of r values
r_values = list(range(n + 1))
# obtaining the mean and variance
```

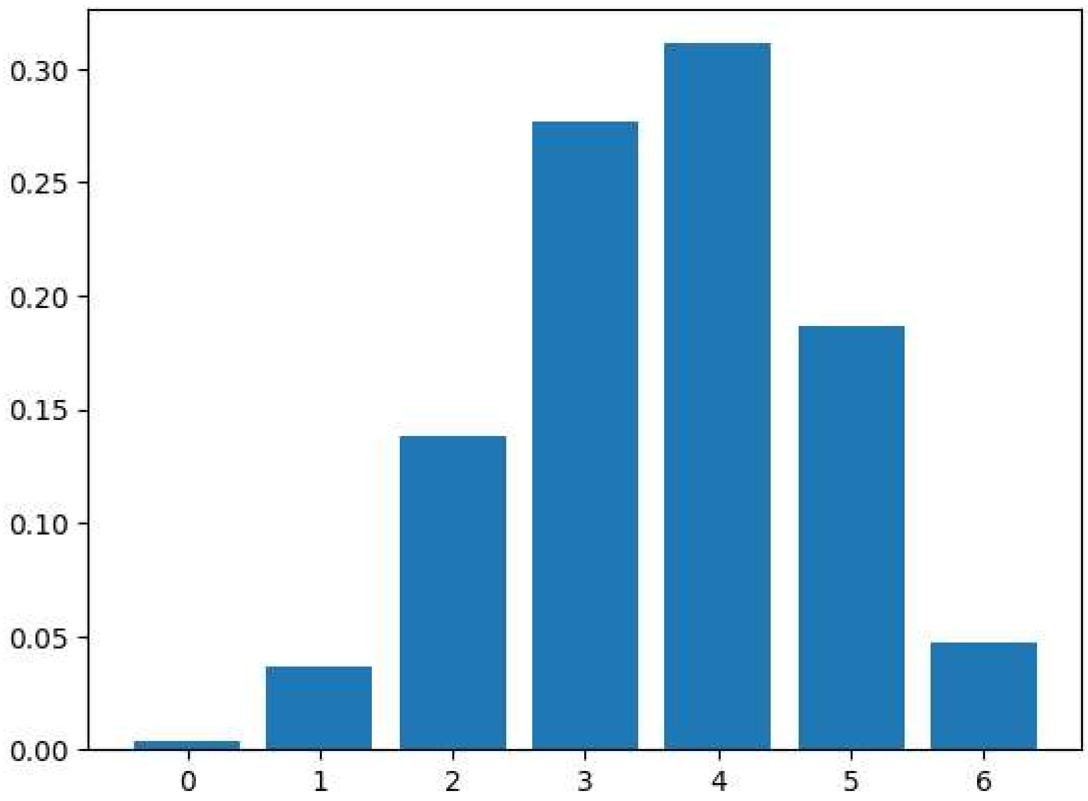
```
mean, var = binom.stats(n, p)
# list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values ]
# printing the table
print("r\tp(r)")
for i in range(n + 1):
    print(str(r_values[i]) + "\t" + str(dist[i]))
# printing mean and variance
print("mean = "+str(mean))
print("variance = "+str(var))
```

```
r       p(r)
0       0.004096000000000002
1       0.03686400000000002
2       0.13824000000000009
3       0.27648000000000017
4       0.3110400000000001
5       0.1866240000000001
6       0.046655999999999996
mean = 3.5999999999999996
variance = 1.44
```

In [18]:

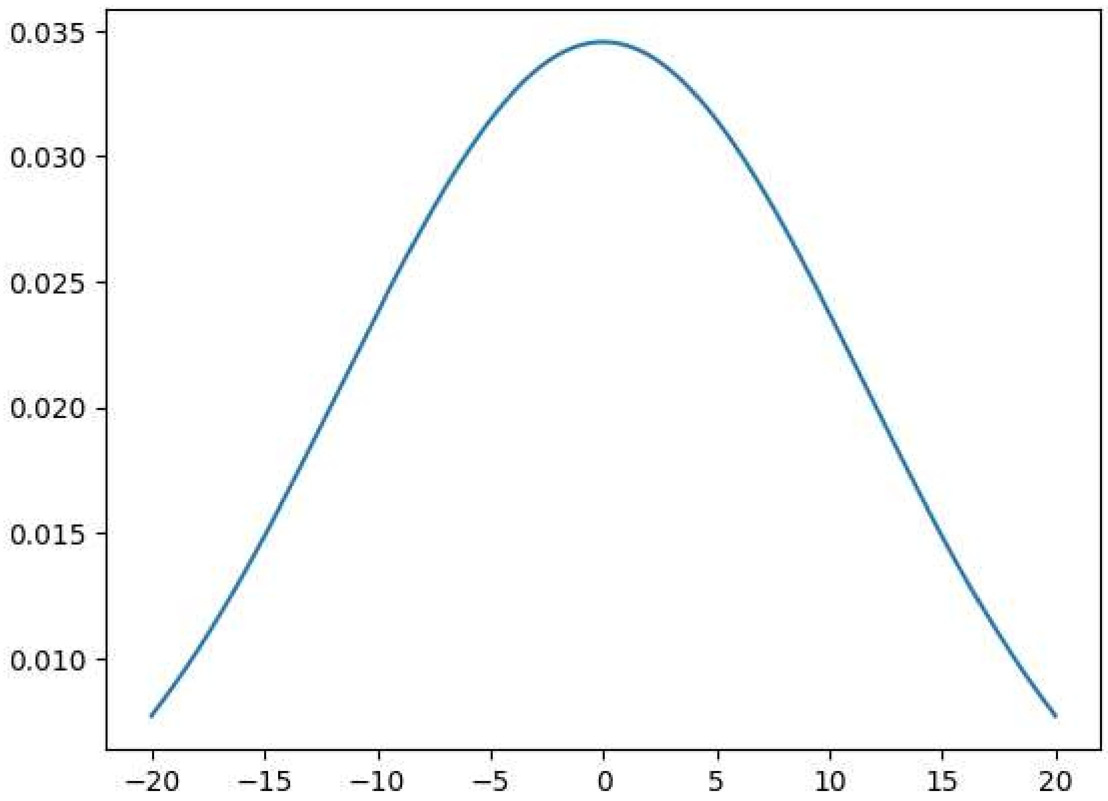
```
# Ex 2 Probability distributions- Binomial Distribution

from scipy.stats import binom
import matplotlib.pyplot as plt
# setting the values
# of n and p
n = 6
p = 0.6
# defining list of r values
r_values = list(range(n + 1))
print (r_values)
# list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values ]
# plotting the graph
plt.bar(r_values, dist)
plt.show()
```

```
[0, 1, 2, 3, 4, 5, 6]
```



In [19]:

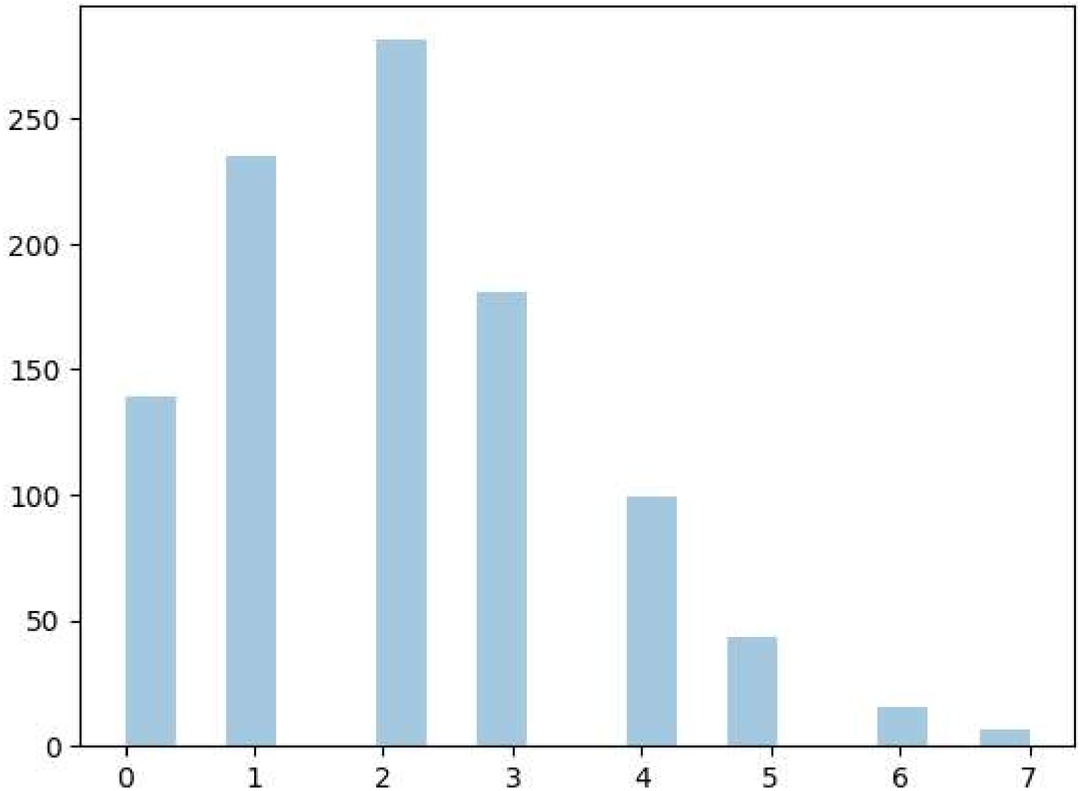
```
# Ex 2 Probability distributions- Normal Distribution

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import statistics

# Plot between -10 and 10 with .001 steps.
x_axis = np.arange(-20, 20, 0.01)

# Calculating mean and standard deviation
mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)

plt.plot(x_axis, norm.pdf(x_axis, mean, sd))
plt.show()
```



In [21]:

```
# Ex 2 Probability distributions- Poisson Distribution

from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.poisson(lam=2, size=1000), kde=False)

plt.show()
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_8700\476171710.py:7: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(random.poisson(lam=2, size=1000), kde=False)
```

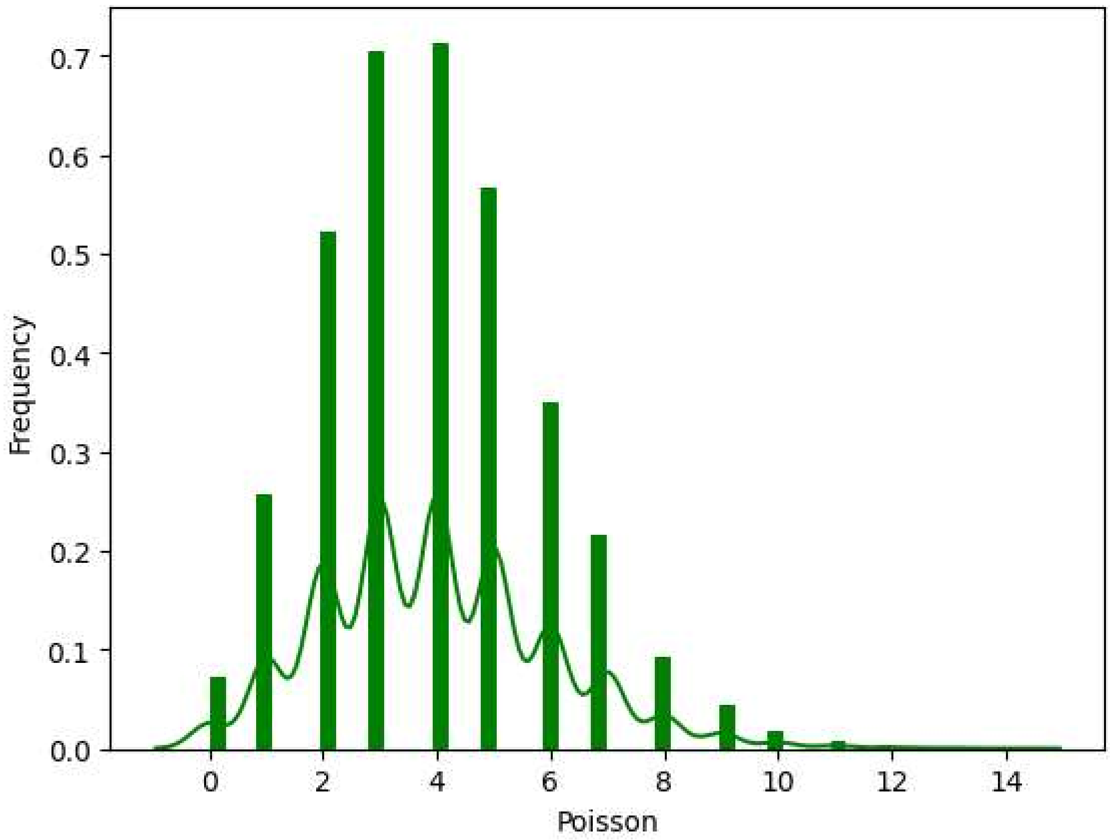


In [22]:
```
# Ex 2 Probability distributions- Poisson Distribution
from scipy.stats import poisson
import seaborn as sb

data_binom = poisson.rvs(mu=4, size=10000)
```

```python
ax = sb.distplot(data_binom,
                  kde=True,
                  color='green',
                  hist_kws={"linewidth": 25,'alpha':1})
ax.set(xlabel='Poisson', ylabel='Frequency')
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_8700\1628879657.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sb.distplot(data_binom,
```

Out[22]:
```
[Text(0.5, 0, 'Poisson'), Text(0, 0.5, 'Frequency')]
```



In [4]:

```
# Ex 2 Test of Significance
# Python program to demonstrate how to
# perform one sample T-test

# import packages
import scipy.stats as stats
import pandas as pd

# loading the csv file
data = pd.read_csv('areas.csv')

# perform one sample t-test
t_statistic, p_value = stats.ttest_1samp(a=data, popmean=5000)
print(t_statistic , p_value)
```

```
[-26.7969394] [9.22302761e-13]
```

In [2]:

```python
# Ex 2 Test of Significance
# Python program to demonstrate how to
# perform two sample T-test

# Import the library
import scipy.stats as stats
import numpy as np

# Creating data groups
data_group1 = np.array([14, 15, 15, 16, 13, 8, 14,
                        17, 16, 14, 19, 20, 21, 15,
                        15, 16, 16, 13, 14, 12])

data_group2 = np.array([15, 17, 14, 17, 14, 8, 12,
                        19, 19, 14, 17, 22, 24, 16,
                        13, 16, 13, 18, 15, 13])

# Perform the two sample t-test with equal variances
stats.ttest_ind(a=data_group1, b=data_group2, equal_var=True)
```

Out[2]:

```
TtestResult(statistic=-0.6337397070250238, pvalue=0.5300471010405257, df=38.0)
```

In [ ]:

```python
# Ex 2 Test of Significance
# Python program to demonstrate how to
# ANOVA

# Importing library
from scipy.stats import f_oneway

# Performance when each of the engine
# oil is applied
performance1 = [89, 89, 88, 78, 79]
performance2 = [93, 92, 94, 89, 88]
performance3 = [89, 88, 89, 93, 90]
performance4 = [81, 78, 81, 92, 82]

# Conduct the one-way ANOVA
f_oneway(performance1, performance2, performance3, performance4)
```

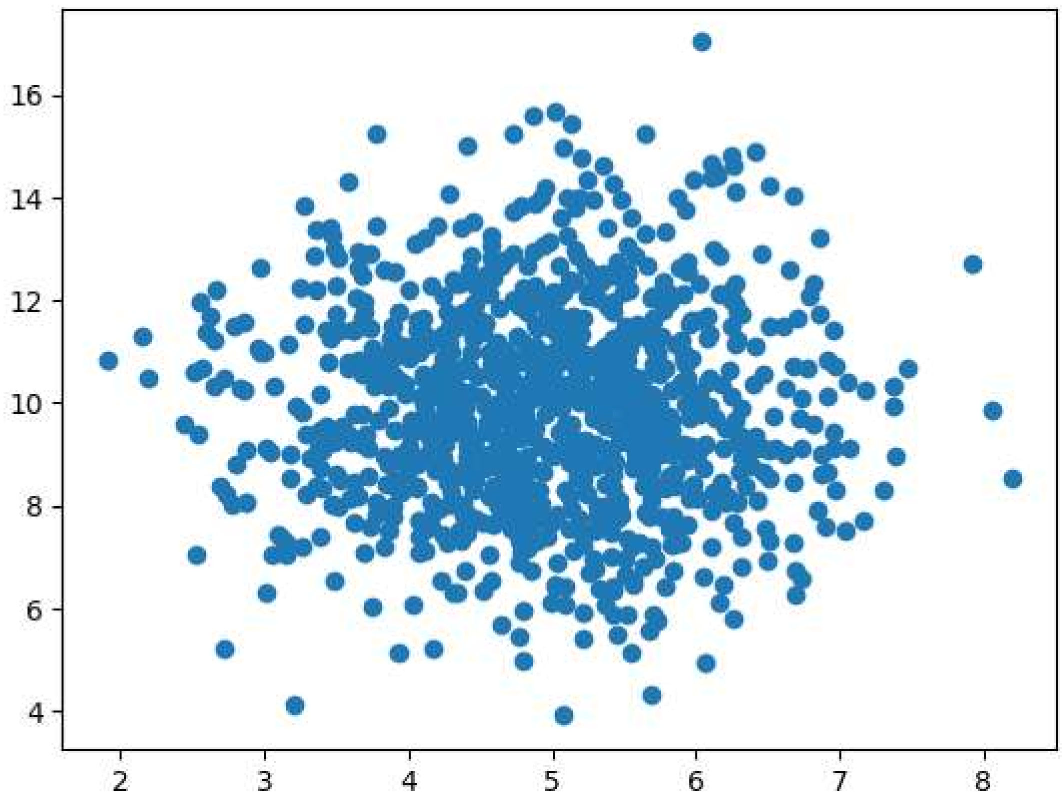
In [1]:

```python
# Ex 2 Test of Significance
# Python program to demonstrate how to
# Chi-Square Test

from scipy.stats import chi2_contingency

# defining the table
data = [[207, 282, 241], [234, 242, 232]]
stat, p, dof, expected = chi2_contingency(data)

# interpret p-value
alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
    print('Dependent (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
p value is 0.10319714047309392
Independent (H0 holds true)
```

In [2]:

```python
# Ex 3 Correlation and Regression analysis
# Scattered diagram
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

plt.scatter(x, y)
plt.show()
```



In [4]:

```
# Ex 3 Correlation and Regression analysis
# Python Program to find correlation coefficient.
import math

# function that returns correlation coefficient.
def correlationCoefficient(X, Y, n) :
    sum_X = 0
    sum_Y = 0
    sum_XY = 0
    squareSum_X = 0
    squareSum_Y = 0


    i = 0
    while i < n :
        # sum of elements of array X.
```

```
        sum_X = sum_X + X[i]

        # sum of elements of array Y.
        sum_Y = sum_Y + Y[i]

        # sum of X[i] * Y[i].
        sum_XY = sum_XY + X[i] * Y[i]

        # sum of square of array elements.
        squareSum_X = squareSum_X + X[i] * X[i]
        squareSum_Y = squareSum_Y + Y[i] * Y[i]

        i = i + 1

    # use formula for calculating correlation
    # coefficient.
    corr = (float)(n * sum_XY - sum_X * sum_Y)/(float)(math.sqrt((n * squareSum_X -sum_X * sum_X)* (n * squareSum_Y -
    return corr

# Driver function
X = [15, 18, 21, 24, 27]
Y = [25, 25, 27, 31, 32]

# Find the size of array.
n = len(X)

# Function call to correlationCoefficient.
print ('{0:.6f}'.format(correlationCoefficient(X, Y, n)))
```

```
0.953463
```

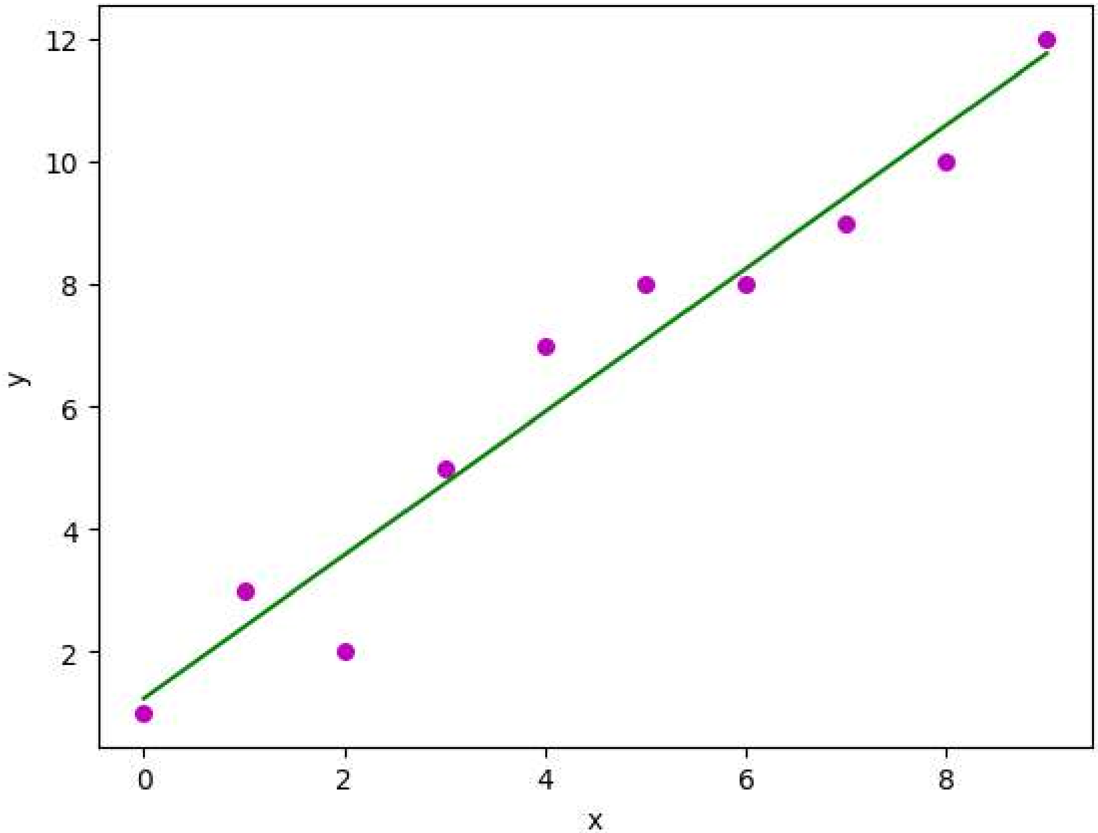
In [5]:

```
# Ex 3 Correlation and Regression analysis
# Linear Reggresion
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
```

```
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)
```

```python
if __name__ == "__main__":
    main()
```

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```




```
In [ ]:
```

In [9]:
```
# Ex 3 Correlation and Regression analysis
# Logistic Reggresion
```

In [14]:
```
# Read and Explore the data
import pandas as pd
dataset = pd.read_csv("User_Data1.csv")

# input
x = dataset.iloc[:, [2, 3]].values

# output
y = dataset.iloc[:, 4].values

print(dataset)
```

```
     User ID  Gender  Age  Estimated Salary  Purchased
0   15810944    Male   35             20000          0
1   15668575  Female   26             43000          0
2   15603246  Female   27             57000          0
3   15804002    Male   19             76000          0
4   15728773    Male   27             58000          0
5   15598044  Female   27             84000          0
6   15694829  Female   32            150000          1
7   15600575    Male   25             33000          0
8   15727311  Female   35             65000          0
9   15570769  Female   26             80000          0
10  15606274  Female   26             52000          0
11  15746139    Male   20             86000          0
12  15704987    Male   32             18000          0
13  15628972    Male   18             82000          0
14  15697686    Male   29             80000          0
15  15733883    Male   47             25000          1
16  15617482    Male   45             26000          1
17  15704583    Male   46             28000          1
```

In [3]:
```
#Splitting The Dataset: Train and Test dataset
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    x, y, test_size=0.25, random_state=0)
```

In [4]:
```
#Splitting The Dataset: Train and Test dataset
from sklearn.preprocessing import StandardScaler

sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)

print (xtrain[0:10, :])
```

```
[[-0.33994485  0.23651273]
 [-0.33994485  0.1985539 ]
 [ 1.50998758 -0.97816994]
 [ 1.61276161 -0.90225228]
 [-0.44271887  1.07160707]
 [-0.5454929  -0.71245811]
 [-1.26491106  1.14752474]
 [-1.05936302  1.29936007]
 [-1.16213704  0.91977174]
 [ 0.48224734 -1.20592294]]
```

In [5]:
```
#Train The Model
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
```

Out[5]:
```
▾ LogisticRegression
LogisticRegression(random_state=0)
```

In [6]:
```
y_pred = classifier.predict(xtest)
```

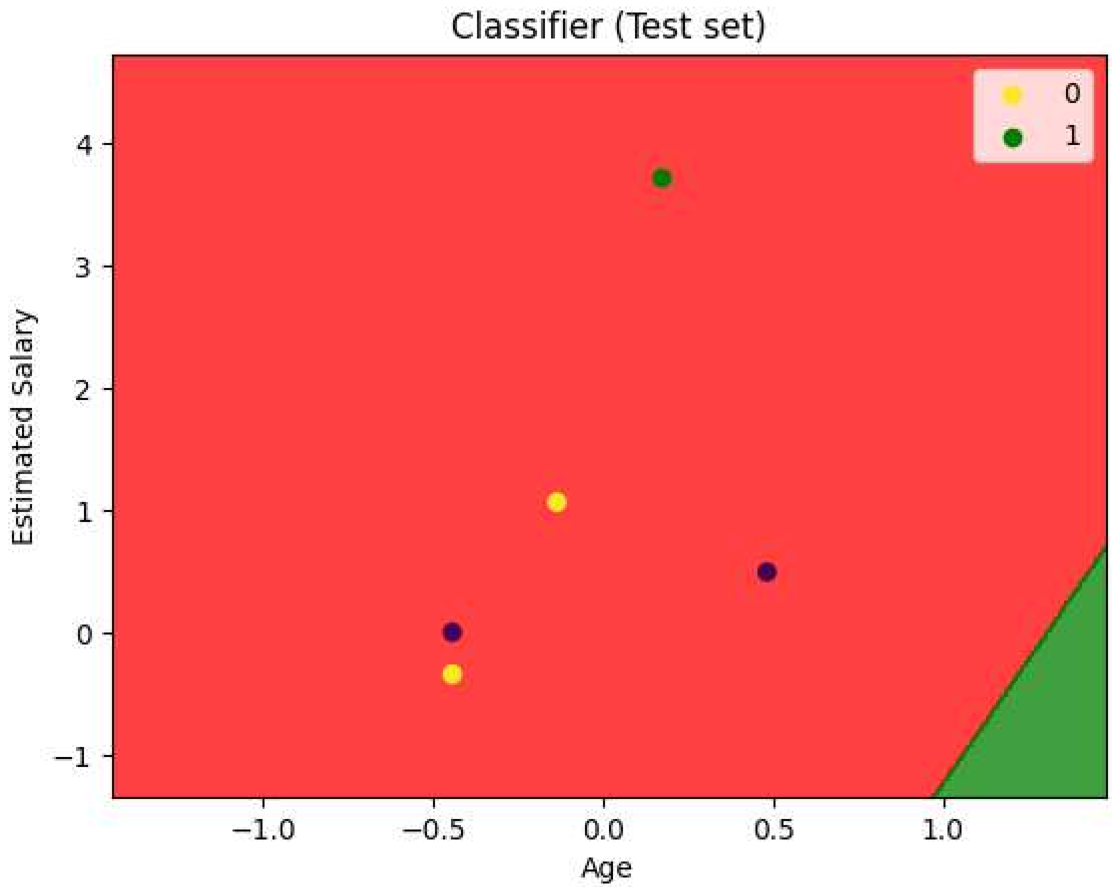
In [7]:
```
#Evaluation Metrics

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(ytest, y_pred)
print ("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[4 0]
 [1 0]]
```

In [8]:

```python
#Visualizing the performance
from matplotlib.colors import ListedColormap
import numpy as np
from matplotlib import pyplot as plt

X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(
             np.array([X1.ravel(), X2.ravel()]).T).reshape(
             X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_7924\4261614875.py:20: UserWarning: *c* argument looks like a single numer
ic RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches wit
h *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specif
y the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```


Classifier (Test set)
0
1
Estimated Salary
4
3
2
1
0
−1
−1.0
−0.5
0.0
0.5
1.0
Age

In [ ]: