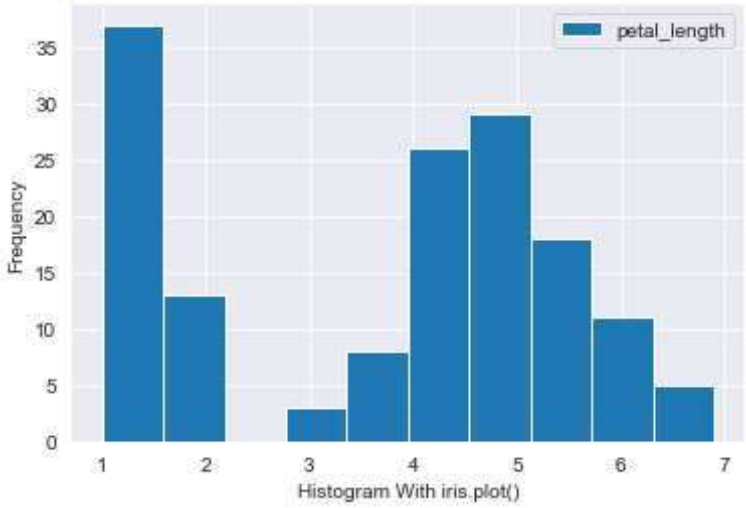
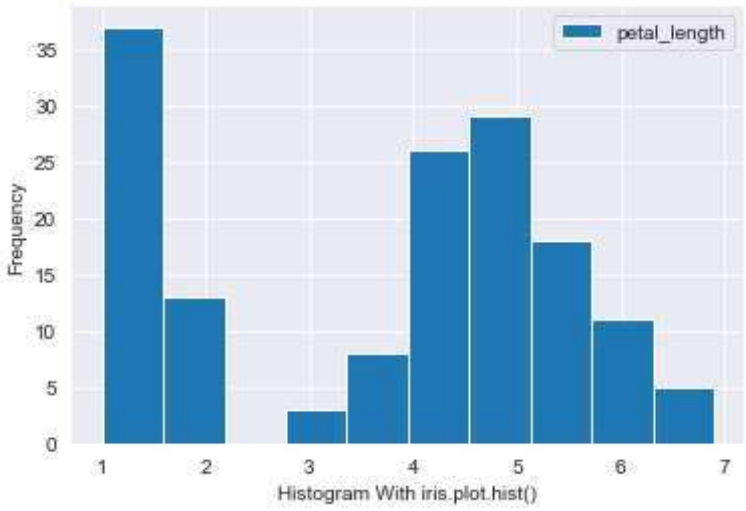
For example, the histogram of the Petal_Length column in the iris dataset will be drawn. We can do this using one of the following methods. While the result of the first two methods was similar, the third method drew a slightly different graph. These three methods are also valid for other drawing types other than histograms.

In [3]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset('iris')
iris.plot(y='petal_length',kind ='hist')
plt.xlabel('Histogram With iris.plot()')
plt.show()
```

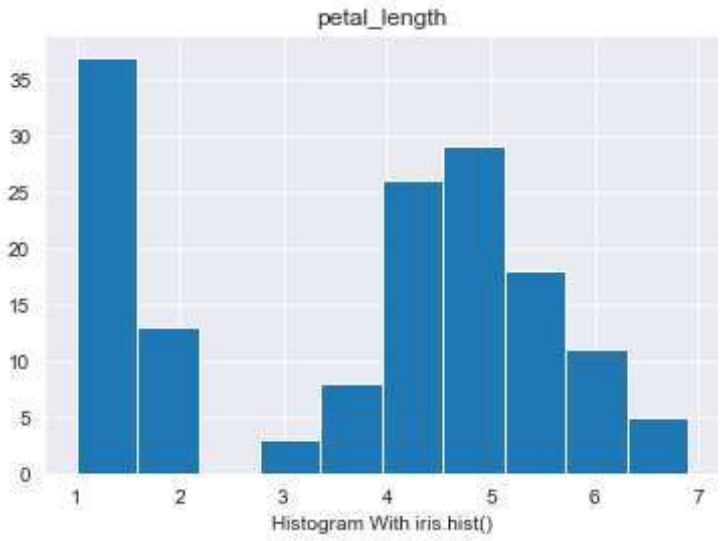


In [4]:
```
iris.plot.hist(y='petal_length')
plt.xlabel('Histogram With iris.plot.hist()')
plt.show()
```



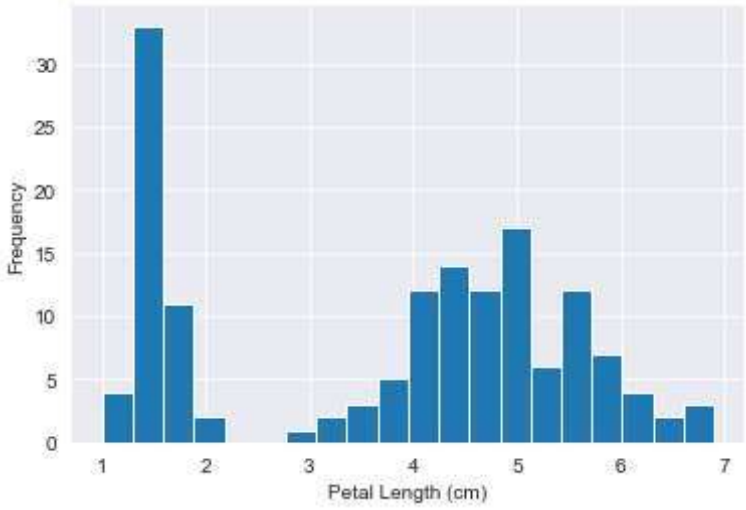
In [5]:
```
iris.hist(column='petal_length')
plt.xlabel('Histogram With iris.hist()')
```

```
plt.show()
```



We can change the number of bins with the bins argument.

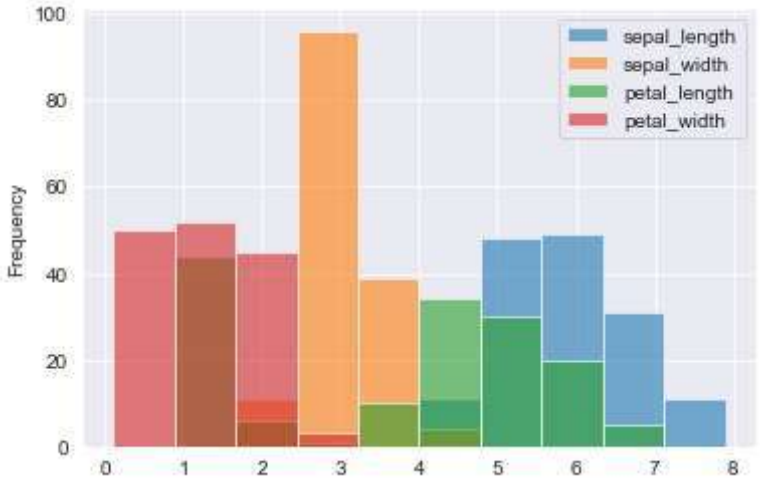
In [6]:
```
plt.hist(iris['petal_length'],bins=20)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Frequency')
plt.show()
```



Let's see the ratio of the total number of data on the y-axis instead of the number of data. For this, we will set 'normed=True'.

What happens if we don't write the column name? Let's see this with the code below.

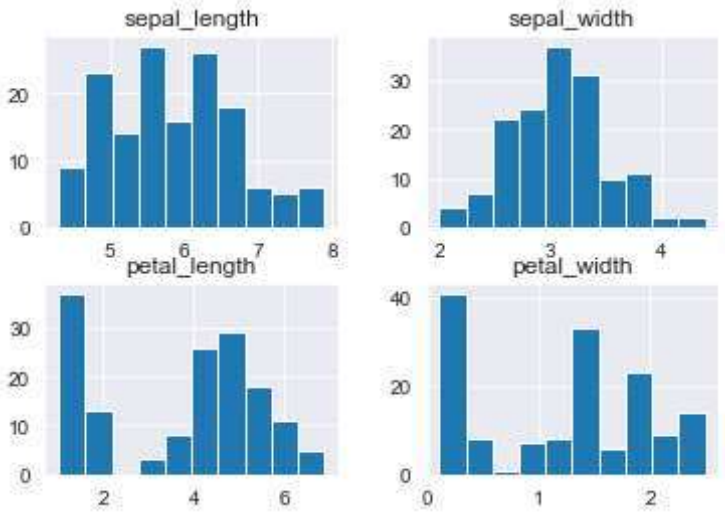
In [14]:
```
iris.plot(kind='hist',alpha=0.6)
plt.show()
```



In the example above we saw the histogram of the different columns together.

Let's look at another usage without inputting a column name.

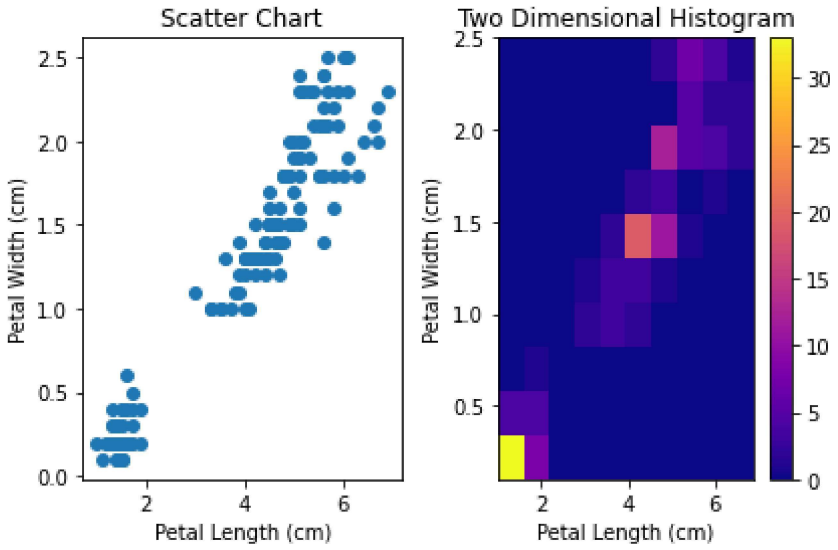
In [17]:
```
iris.hist()
plt.show()
```



This usage plotted Histogram graph of each column separately as seen.

We can plot histograms of data consisting of two variables. These graphs are also called heat maps. Let's examine this situation in the example below.

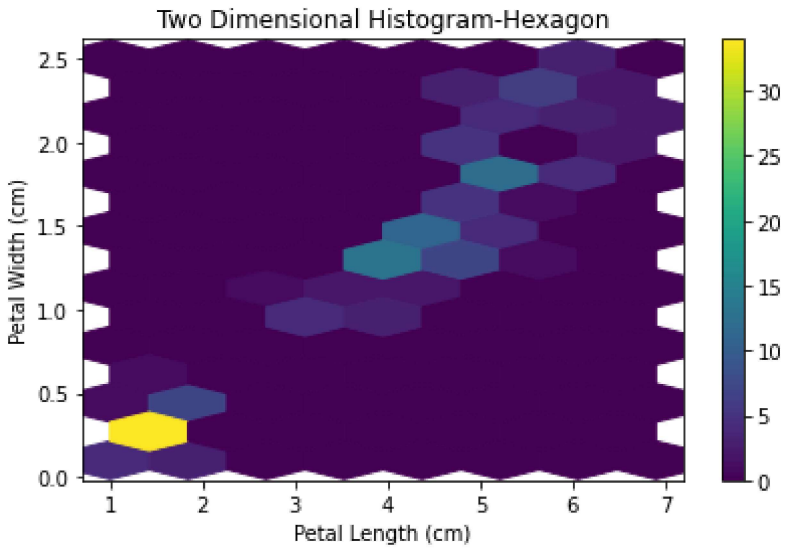
In [1]:
```
import matplotlib.pyplot as plt
#to load the dataset.
from sklearn import datasets
iris = datasets.load_iris()

x = iris.data[:,2]
y = iris.data[:,3]

plt.subplot(1,2,1)
plt.scatter(x,y)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Scatter Chart')

plt.subplot(1,2,2)
plt.hist2d(x,y,cmap='plasma')
plt.colorbar()
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Two Dimensional Histogram')

plt.tight_layout()
plt.show()
```



In the temperature map above, the partitions are shown as rectangles. Another variation of this drawing is the hexagonal partitions. For this, the plt.hexbin() command is used.

In [2]:
```
plt.hexbin(x,y,gridsize=(7,7))
plt.colorbar()
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Two Dimensional Histogram-Hexagon')
```

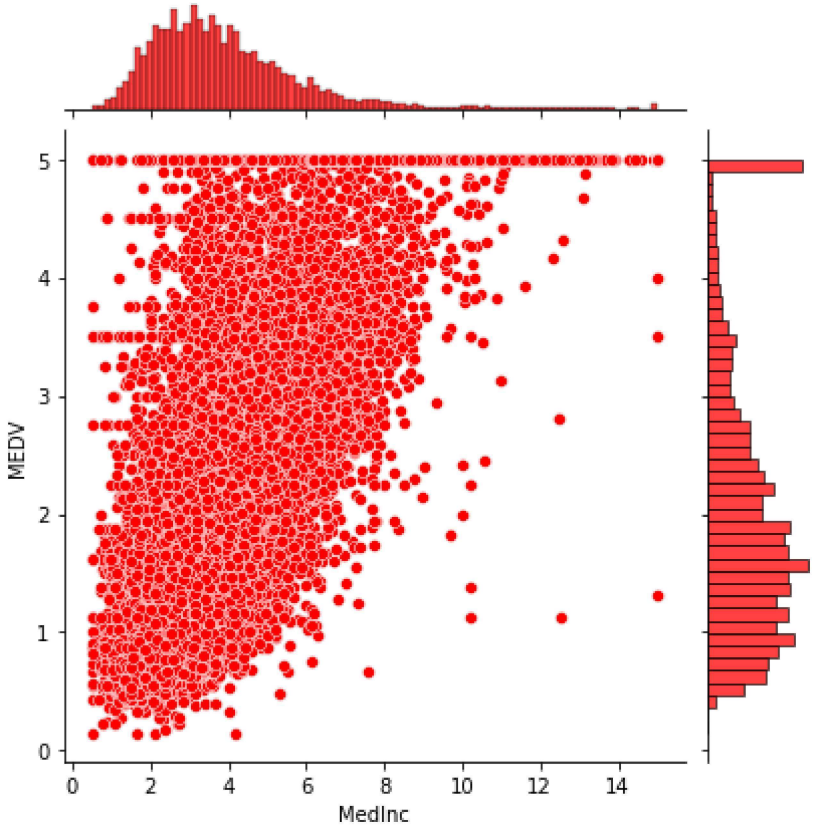
```
plt.tight_layout()
plt.show()
```



The jointplot() function in the Seaborn module allows us to plot the relationship and distributions between two variables. We can see the correlation coefficient and value between the two variables in the graph.

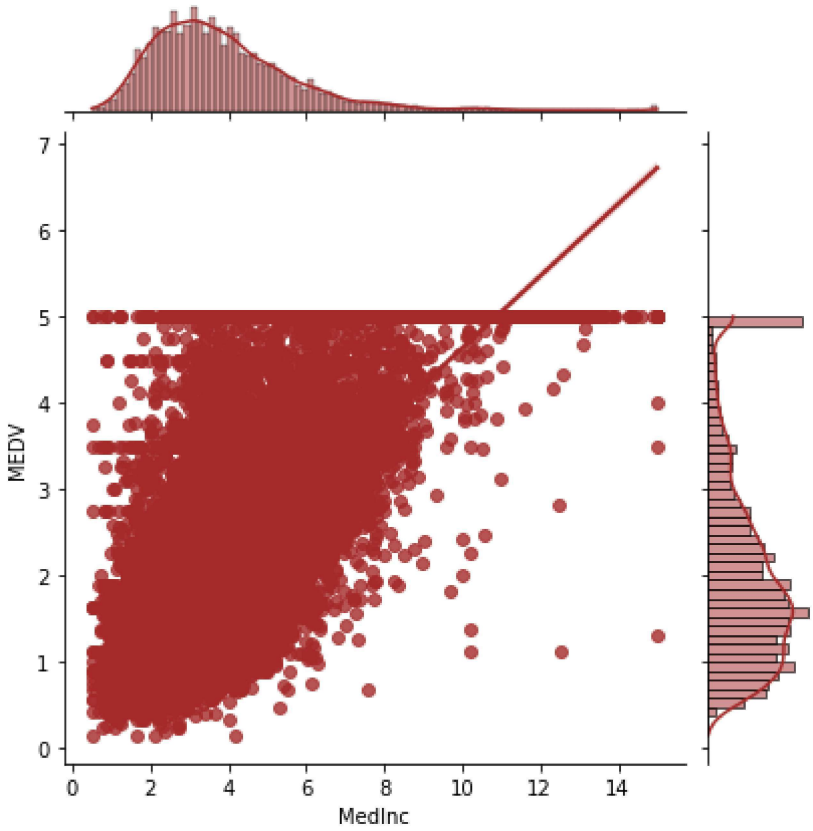
In [26]:

```
import seaborn as sns
import pandas as pd
#to load the dataset.

from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()

housingdf = pd.DataFrame(housing.data)
housingdf.columns =housing.feature_names
housingdf['MEDV'] = housing.target

sns.jointplot(x='MedInc',y='MEDV',data=housingdf,color='red')

plt.show()
```



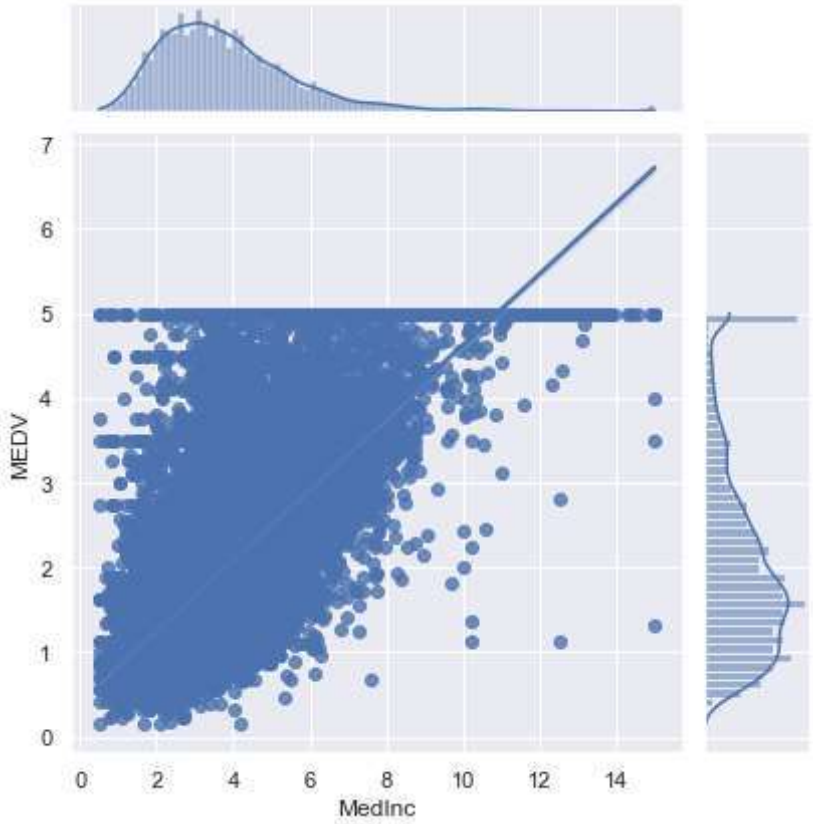
It is also possible to differentiate the graph. For example, we can see the density function estimations along with the histograms of the data on the axes and plot the regression graph in the point graph. For this, we need to write kind='reg' in the jointplot() function.

In [28]:
```
sns.jointplot(x='MedInc',y='MEDV',data=housingdf,color='brown',kind='reg')
plt.show()
```



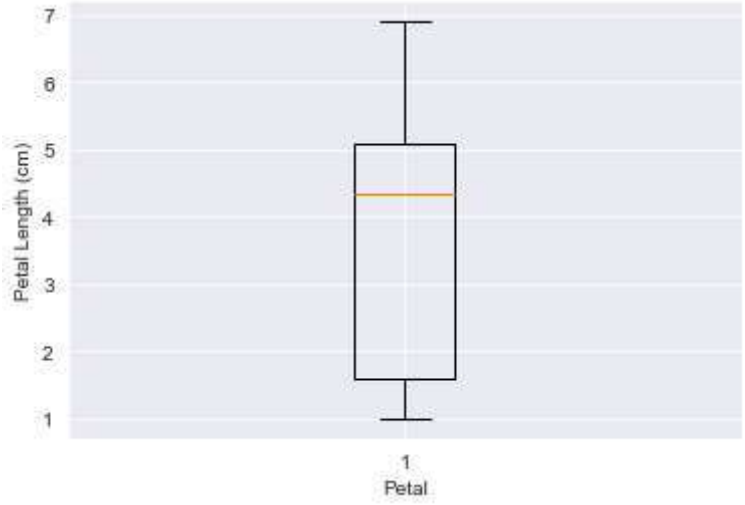
We use the .set() method for Seaborn's own theme.

In [30]:
```
sns.set()
sns.jointplot(x='MedInc',y='MEDV',data=housingdf,kind='reg')
plt.show()
```


MEDV
MedInc
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12
14

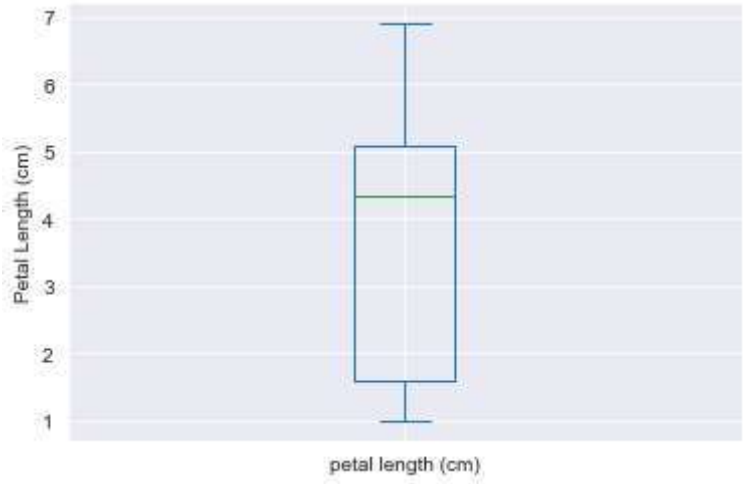
The box chart is a type of chart where you can see the minimum and maximum median values of a data and its 25 percent and 75 percent slices. It is a tool that you can use to quickly see in which intervals the data we are examining is distributed.

In [5]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

irisdf = pd.DataFrame(iris.data)
irisdf.columns = iris.feature_names

plt.boxplot(irisdf['petal length (cm)'])
plt.xlabel('Petal')
plt.ylabel('Petal Length (cm)')
plt.show()
```



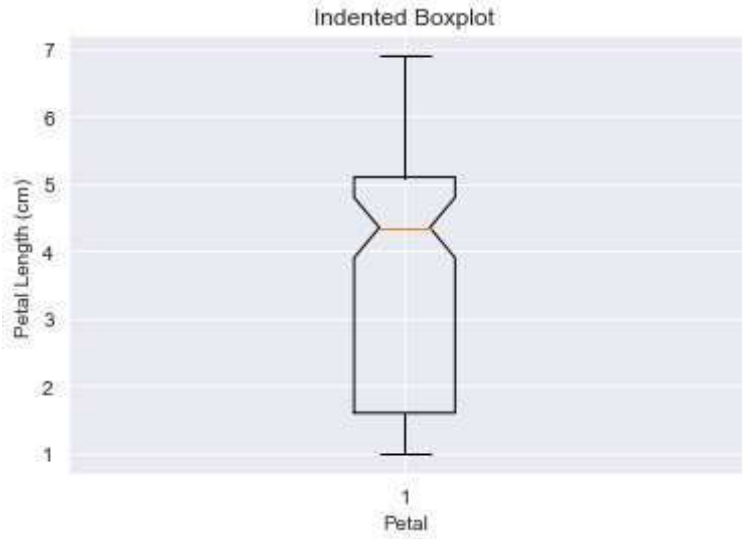
Yukarıdaki grafikte alt ve üst çizgiler incelediğimiz verideki en düşük ve en yüksek değerleri göstermektedir.Alt ve üst kenarları ise sırasıyla yüzde 25 ve yüzde 75 lik dilimlerini gösterir.Yukarıdaki grafiği çizme nin başka bir yolu irisdf.plot(y='petal length (cm)',kind='box') komutudur.

In [6]:
```
irisdf.plot(y='petal length (cm)',kind='box')
plt.ylabel('Petal Length (cm)')
plt.show()
```



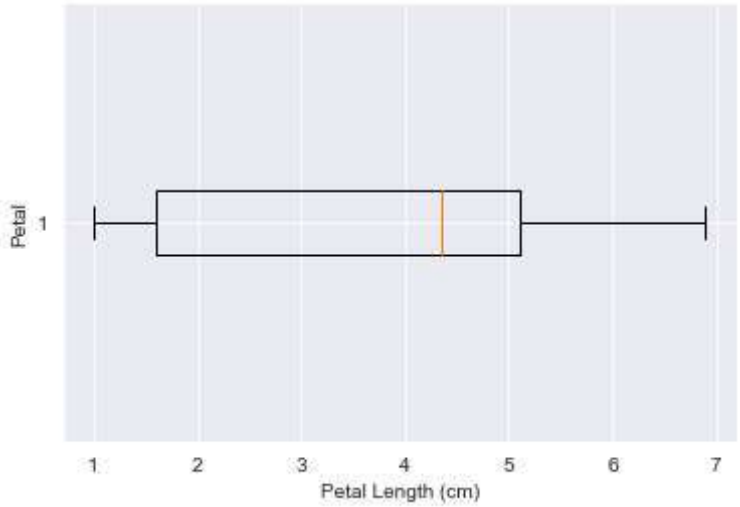
To show the median value more clearly, we can indent the median line as follows. We make this change with the notch=1 argument.

In [9]:
```
plt.boxplot(irisdf['petal length (cm)'],notch=1)
plt.xlabel('Petal')
plt.ylabel('Petal Length (cm)')
plt.title('Indented Boxplot')
plt.show()
```



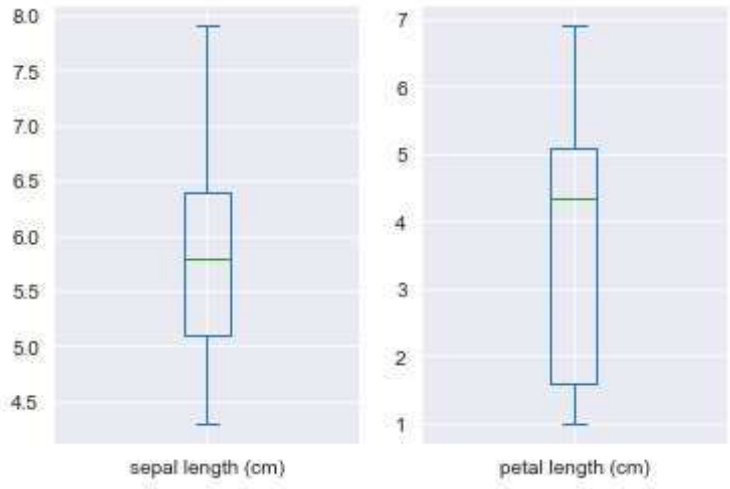
Now let's draw the graph horizontally.

In [11]:
```
plt.boxplot(irisdf['petal length (cm)'],vert=False)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal')
plt.show()
```



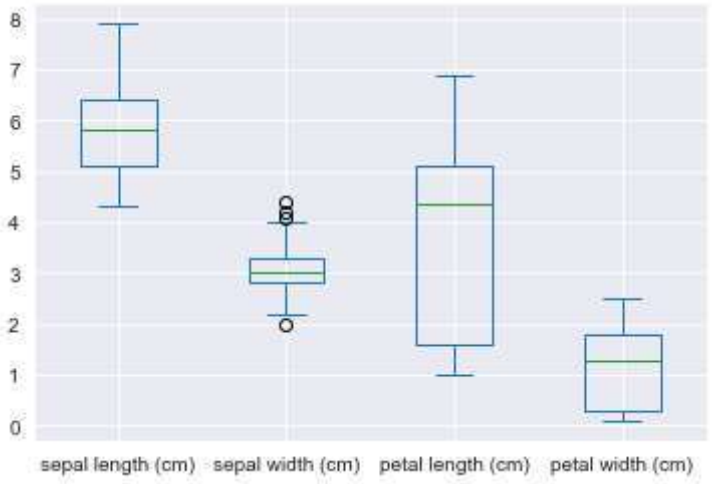
It is possible to draw a boxplot of more than one column in the data frame. Let's examine the example below.

In [13]:
```
colmns =['sepal length (cm)','petal length (cm)']
irisdf[colmns].plot(kind='box',subplots=True)
plt.show()
```



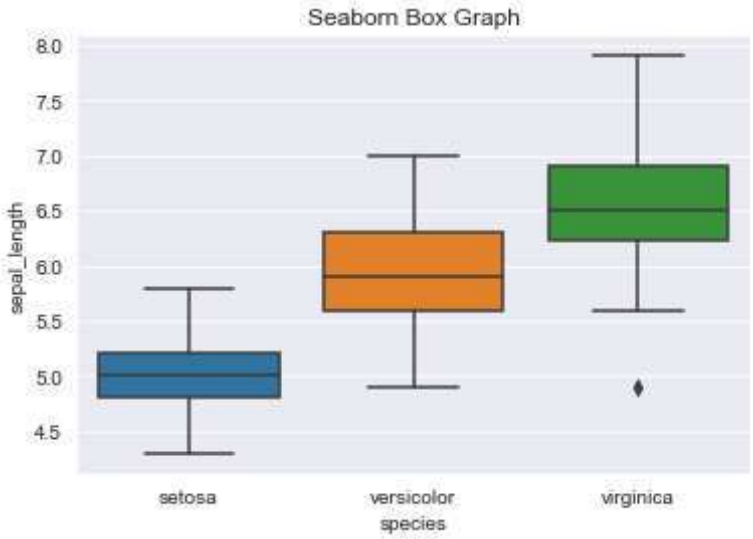
If no column selection is made, all columns are drawn.

In [14]:
```
irisdf.plot(kind='box')
plt.show()
```



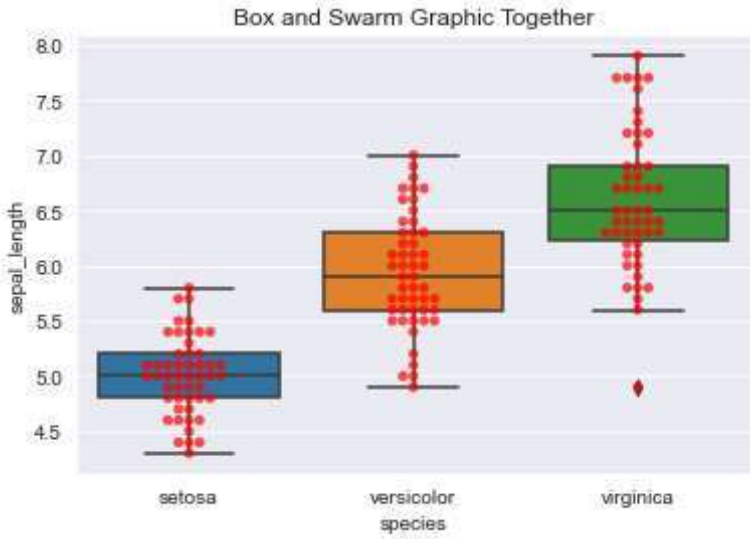
Let's examine the boxplot drawing with seaborn. The .boxplot() method is used.

In [16]:
```
import seaborn as sns

iris = sns.load_dataset('iris')
sns.boxplot(x='species',y='sepal_length',data=iris)
plt.title('Seaborn Box Graph')
plt.show()
```



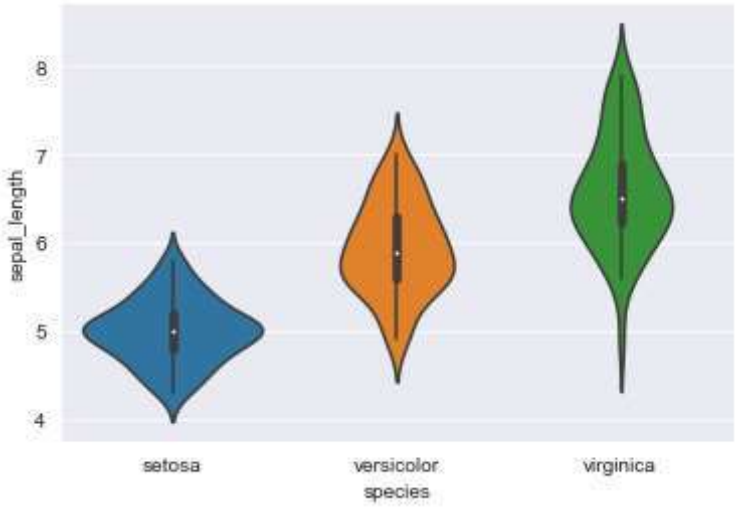
It is also possible to plot the data on the boxplot. We will use the .swarmplot() method for this.

In [30]:
```
sns.boxplot(x='species',y='sepal_length',data=iris)
plt.title('Box and Swarm Graphic Together')
sns.swarmplot(x='species',y='sepal_length',data=iris,color='r',alpha=0.7)
plt.show()
```


Box and Swarm Graphic Together
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
sepal_length
setosa
versicolor
virginica
species

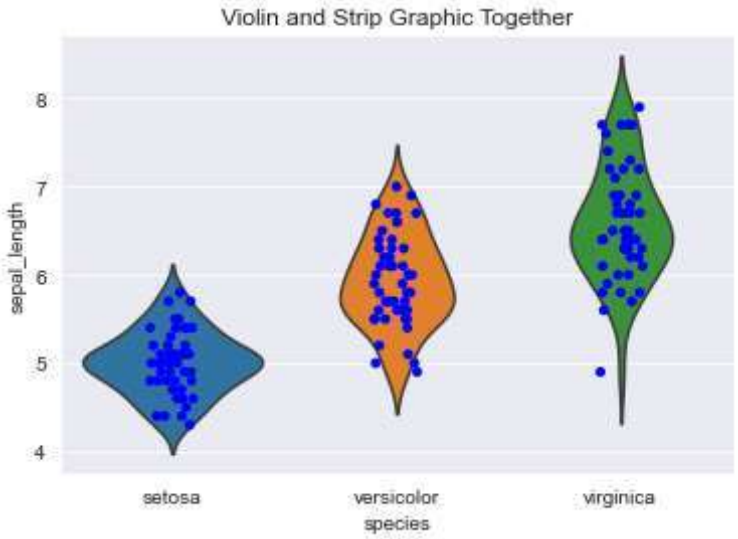
When we examine the violin chart, it is used to have information about the distribution of the data. The violinplot() method in the seaborn module is used to draw the violin chart.

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns

iris = sns.load_dataset('iris')
sns.violinplot(x='species',y='sepal_length',data=iris)
plt.show()
```



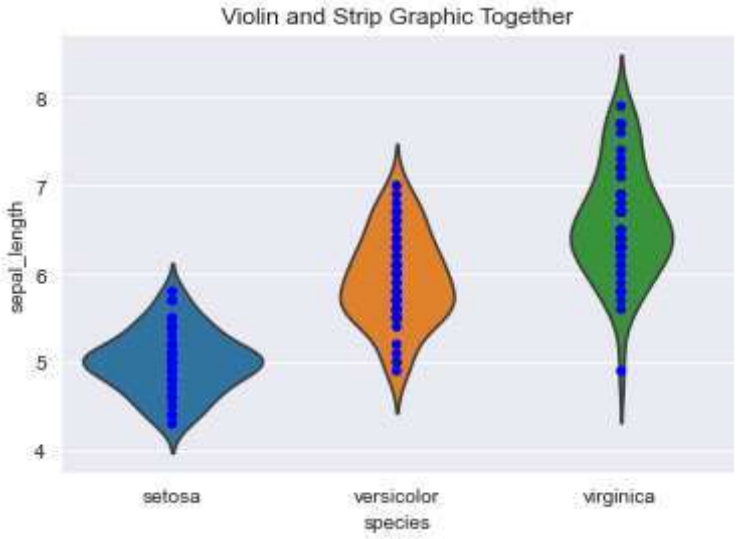
We can show the data on the violin chart, as we did in the boxplot. Let's remove the boxplot so that there is no confusion. For this, we need to specify inner=None. Let's use a different graphic type, stripplot, instead of swarmplot this time. Normally, the data in stripplot type is on a single line.However, since this complicates the visualization, we ensure that the data is spread left and right with the jitter=True argument.

In [6]:
```
sns.violinplot(x='species',y='sepal_length',data=iris,inner=None)
sns.stripplot(x='species',y='sepal_length',data=iris,jitter=True,color='blue')
plt.title('Violin and Strip Graphic Together')
plt.show()
```



what if jitter=False?

In [7]:
```
sns.violinplot(x='species',y='sepal_length',data=iris,inner=None)
sns.stripplot(x='species',y='sepal_length',data=iris,jitter=False,color='blue')
plt.title('Violin and Strip Graphic Together')
plt.show()
```



BINOMIAL DISTRIBUTION

$$p(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Now let's do a binomial experiment. Let's do three coin flip experiments. Heads and Tails will be denoted by 0 and 1, respectively. p is the probability of getting head.(fraudulent money)

- $p = 0.4$
- $n = 3$

In [2]:
```
import numpy as np
n=3
data = np.random.random(size=n)
data > 0.6
```

Out[2]:
```
array([ True,  True,  True])
```

The head ratio calculated above is not close to the theoretically calculated ratio. The theoretically calculated rate is about 28 percent. For this, if we increase the number of experiments, we get closer to the theoretical result. This is because of the law of large numbers.

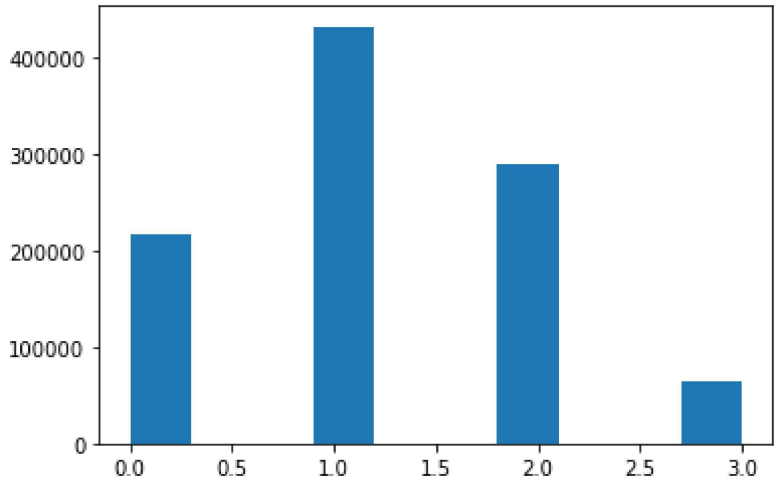
Now let's repeat the same experiment many times and calculate the rate of trials that land on 2 heads.

In [3]:
```
two_heads = 0
for i in range(1000000):
    head = np.random.random(size=n) > 0.6
    if np.sum(head) == 2:
        two_heads +=1
two_heads / 1000000
```

Out[3]:
```
0.288352
```

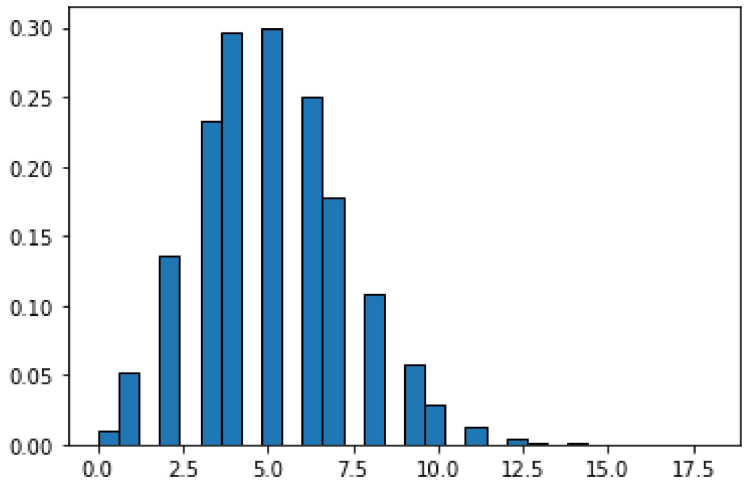
Python has the np.random.binomial() function to simulate the binomial distribution.

In [4]:
```
import matplotlib.pyplot as plt

binomial_experiment = np.random.binomial(n=3,p=0.4,size=1000000)
binomial_experiment

plt.hist(binomial_experiment)
plt.show()
```



Let's use a fraudulent coin again and increase the number of coin flips in each experiment. Let the probability of the flipped coin getting heads is 0.05 and 100 coins are tossed at once. Let's repeat this experiment 1000000 times. Let's see what will be the result distribution?

In [5]:
```
binom = np.random.binomial(n=100,p=0.05,size=1000000)

plt.hist(binom,edgecolor='k',align='mid',bins=30,density=True)
plt.show()
```

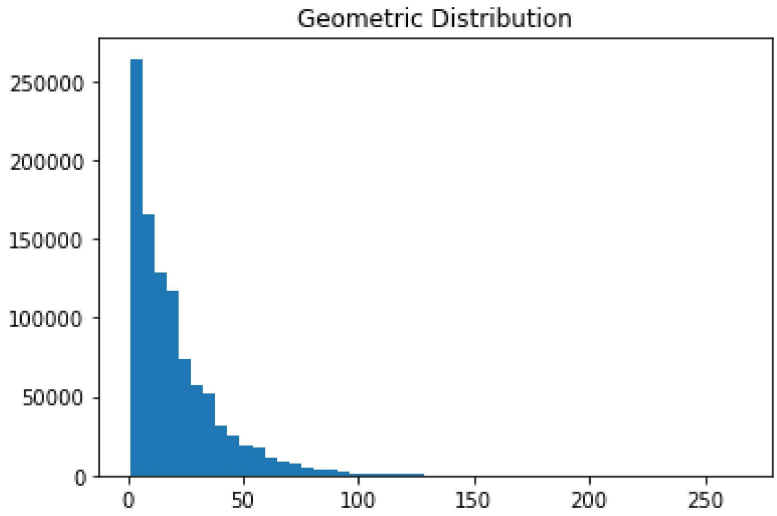


GEOMETRIC DISTRIBUTION

$$p(X = k) = (1 - p)^{k-1}.p$$

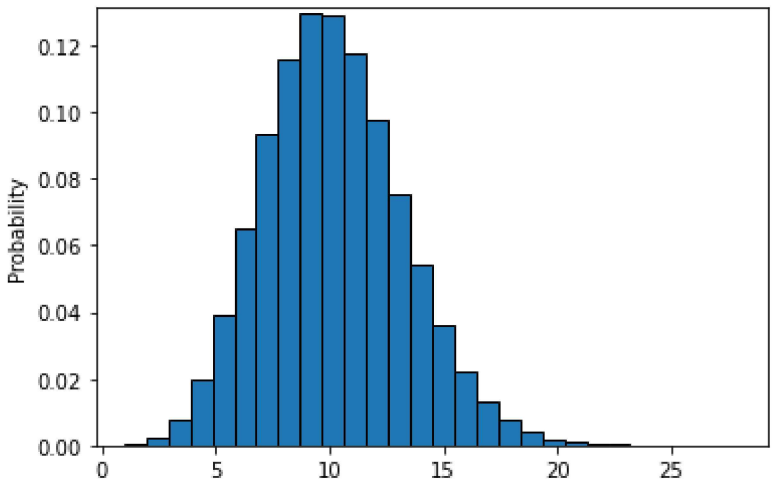
Let's get a geometric distribution and draw its histogram.

In [6]:
```
geom = np.random.geometric(p=0.05,size=1000000)
plt.hist(geom,bins=50)
plt.title('Geometric Distribution')
plt.show()
```



Let's simulate the Poisson distribution. We will use the np.random.poisson() method. This function takes two arguments. lam(lambda) and number of experiments(size).

In [7]:
```
pois = np.random.poisson(lam=10,size=1000000)
plt.hist(pois,edgecolor='k',align='mid',bins=30,density=True)
plt.ylabel('Probability')
plt.margins(0.01)
plt.show()
```



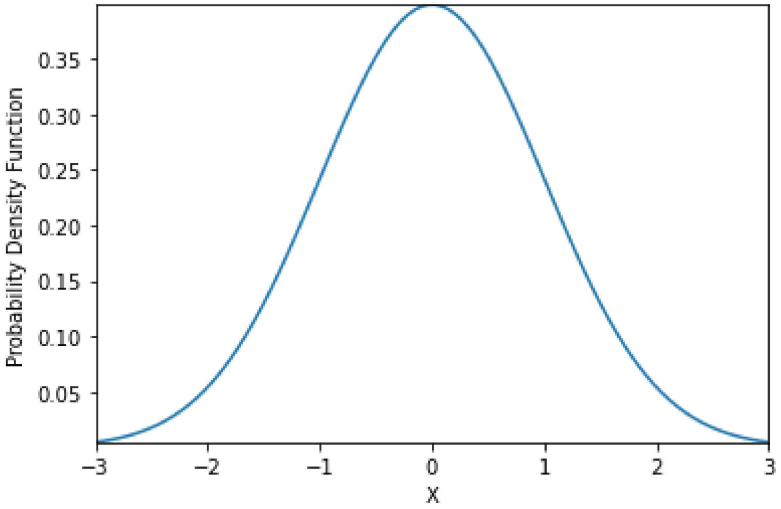
NORMAL DISTRIBUTION

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Let's examine a normal distribution with a mean of 0 and a standard deviation of 1. This distribution is also called the Standard Normal Distribution.

In [8]:
```
from scipy.stats import norm

mu = 0
sigma = 1
```

```
x = np.linspace(mu-3*sigma,mu+3*sigma,100)
y = norm.pdf(x,mu,sigma)

plt.plot(x,y)
plt.xlabel('X')
plt.ylabel('Probability Density Function')
plt.margins(0.00)
plt.show()
```



In this code, the norm.pdf() method in the scipy.stats module calculates the normal distribution probability density function values of 100 values between -3 and +3 according to the above equation.

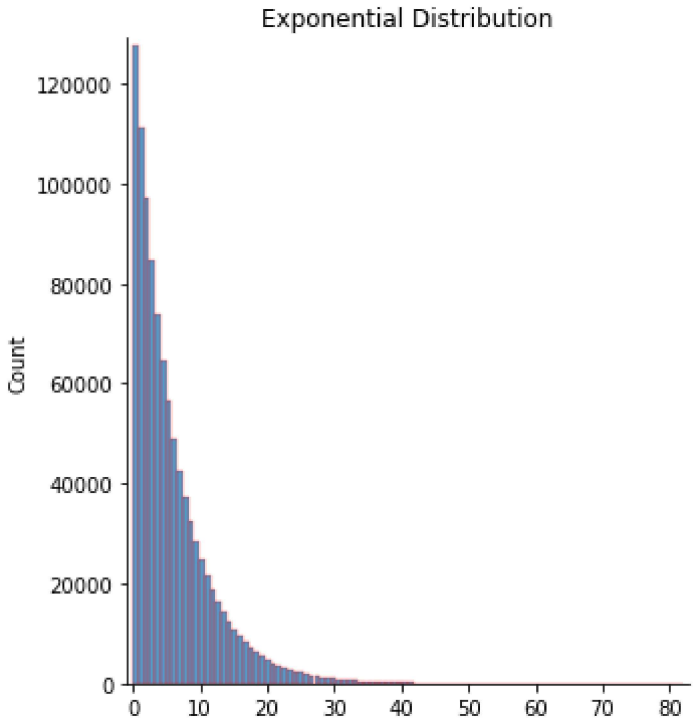
EXPONENTIAL DISTRIBUTION

$$f(x,\lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

The exponential distribution is the continuous version of the geometric distribution. Finally, let's see how we can theoretically obtain the exponential distribution in Python.

In [17]:

```
import seaborn as sns
exp1 = np.random.exponential(6,1000000)
sns.displot(exp1, bins=100,edgecolor='red')
plt.title('Exponential Distribution')
plt.margins(0.01)
plt.show()
```



It is seen that the probability density function of the exponential distribution is very similar to the probability density function of the geometric distribution. The exponential distribution is actually the continuous form of the geometric distribution.