Nombre: ***Macias Pico Josselyn Stefany***

Curso: ***Sexto "B"***

Materia: ***Modelamiento y Simulacion***

Docente: ***Ing. Jorge Anibal Moya Delgado***

SISTEMA ESTADISTICO DE CRIPTOMONEDAS "BITCOIN"

La presente investigación se refiere al tema de las criptomonedas ya que hoy en día se vive en una sociedad basada en los negocios virtuales de manera que algunas de las personas han entrado a los negocios de criptomonedas ya que es un medio digital de intercambio, lo han utilizado para poder obtener mayores ingresos y darles mayor usabilidad a variedades de monedas, entre ellas están el bitcoin que es de la que se hablara en este informe.

Ejercicio

Este ejercicio se realiza despues de haber mencionado algunos obstaculos importantes sobre las criptomonedas "BITCOIN" dado que no hay un estado detrás de ellas lo cual pueda gestionar su precio siendo en este caso lo que hacen los bancos centrales al gestionar una inflación o garantizar el cumplimiento de los pagos. Es por esto que se ha decidido utilizar un software en el cual se pueda mostrar la variación con cada uno de los datos cuando sube o baja la moneda, para así poder llevar un control entre la informacion de los máximos y mínimos de la moneda incluyendo el valor de apertura y el valor que define al terminar el día.

Obtencion de datos

En esta parte se mostraran los datos dentro de la pagina como respuesta

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

datos = pd.read_excel('C:/Users/Villamar/Desktop/JOSS/SEXTOSEMESTRE/MODELAMI
datos
```

Out[1]:

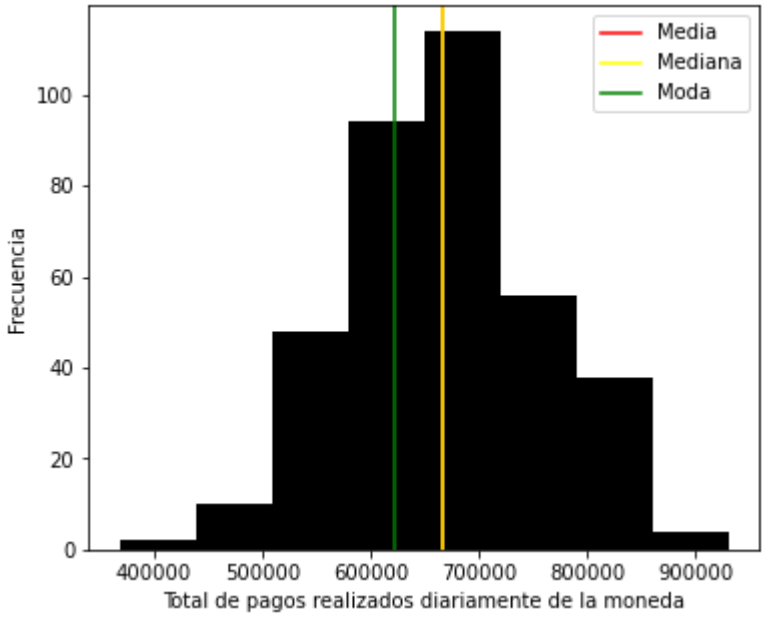
	fecha	total-bitcoins	volumen-comercio	promedio-bloque	billetera-usuarios	pagos-moneda	precio-mercado
0	2020-07-03	18421362.50	1.072075e+08	3453.789809	50807793	542245	9087.98
1	2020-07-04	18421587.50	5.780329e+07	2975.134969	50807793	484947	9072.42
2	2020-07-05	18421806.25	3.883966e+07	4027.668831	50819572	620261	9131.31
3	2020-07-06	18422025.00	5.072085e+07	4034.937888	50834080	649625	9089.09
4	2020-07-07	18422243.75	1.137434e+08	4257.635762	50844380	642903	9348.91
...	...	...	...	...	...	...	...
361	2021-06-29	18500556.25	3.559005e+08	6129.449438	54290217	545521	34456.67
362	2021-06-30	18500781.25	3.851351e+08	6221.294737	54309711	591023	35847.70
363	2021-07-01	18501006.25	4.069678e+08	5856.044444	54309711	527044	35047.36
364	2021-07-02	18501225.00	3.241165e+08	4955.709091	54323892	671920	33536.88
365	2021-07-03	18501443.75	2.170665e+08	5407.932584	54328109	622738	33856.86

366 rows × 7 columns

Histogramas

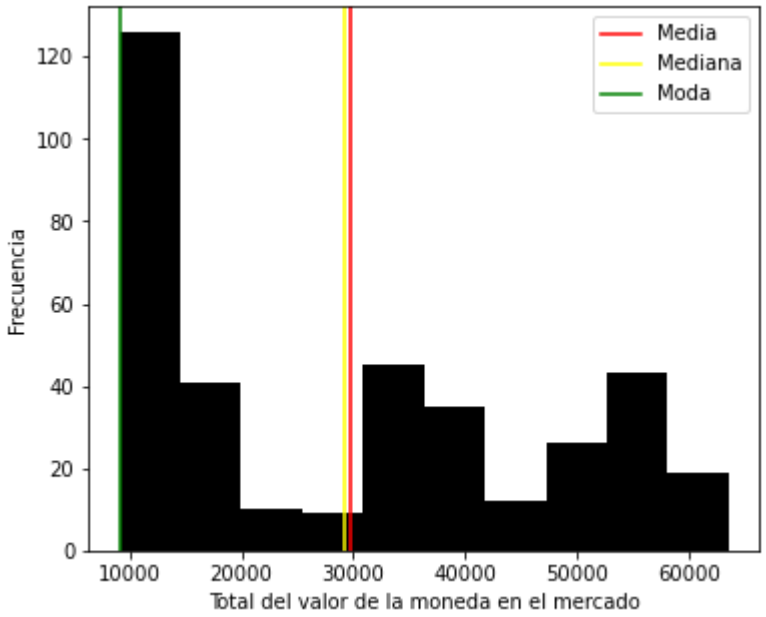
Histograma que mostrara el pago de la moneda diariamente

In [51]:
```
#Gráficos
x=datos["pagos-moneda"]
#modificacion del grafico anchura,altura
plt.figure(figsize=(6,5))
plt.hist(x,bins=8,color='black')
plt. axvline(x. mean(),color='red',label='Media')
plt. axvline(x. median(),color='yellow',label='Mediana')
plt. axvline(x. mode()[0],color='green',label='Moda')
#titulo de las ejes de la X
plt.xlabel('Total de pagos realizados diariamente de la moneda')
#titulo de las ejes de la y
plt.ylabel('Frecuencia')
plt.legend()
plt.show()
```



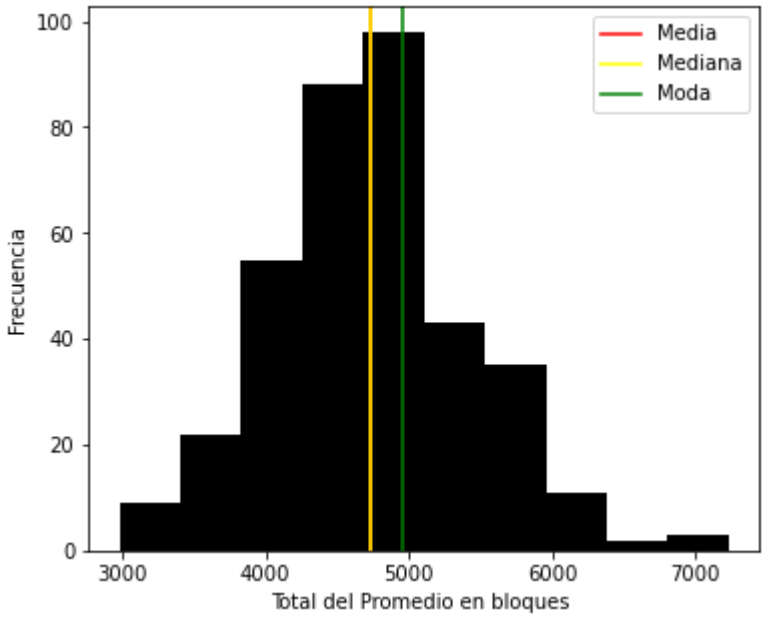
Histograma que mostrara el precio de la moneda en el mercado

In [52]:
```
x=datos["precio-mercado"]
#modificacion del grafico anchura,altura
plt.figure(figsize=(6,5))
plt.hist(x,bins=None,color='black')
plt. axvline(x. mean(),color='red',label='Media')
plt. axvline(x. median(),color='yellow',label='Mediana')
plt. axvline(x. mode()[0],color='green',label='Moda')
#titulo de las ejes de la X
plt.xlabel('Total del valor de la moneda en el mercado')
#titulo de las ejes de la y
plt.ylabel('Frecuencia')
plt.legend()
plt.show()
```



Histograma que mostrara el promedio de la moneda por bloques

In [53]:
```
x=datos["promedio-bloque"]
#modificacion del grafico anchura,altura
plt.figure(figsize=(6,5))
plt.hist(x,bins=None,color='black')
plt. axvline(x. mean(),color='red',label='Media')
plt. axvline(x. median(),color='yellow',label='Mediana')
plt. axvline(x. mode()[0],color='green',label='Moda')
#titulo de las ejes de la X
plt.xlabel('Total del Promedio en bloques ')
#titulo de las ejes de la y
plt.ylabel('Frecuencia')
plt.legend()
plt.show()
```



Media

Es el valor que tendrían los datos, si todos ellos fueran iguales.

En esta parte se calcula la media de las columnas:

-pago-moneda
-preco-mercado
-promedio-bloque

Cada una cuenta con una variable la cual sera mostrada seguido de un mensaje

In [54]:
```
#Calculo de la media
print("Media:")
#Entrada de los datos por medios del array

t4 =datos["pagos-moneda"].mean()
t7  =datos["precio-mercado"].mean()
t8 =datos["promedio-bloque"].mean()

#Mostreo de los datos
print( "\nla Media de los pagos en moneda realizados diariamente: ", t4)
print( "\nla Media del precio de la moneda en el mercado: ", t7)
print( "\nla Media del promedio de la moneda en bloques: ", t8)

```

```
Media:

la Media de los pagos en moneda realizados diariamente:  667037.9153005464

la Media del precio de la moneda en el mercado:  29757.23409836064

la Media del promedio de la moneda en bloques:  4737.817554775539
```

Media Aritmetica

Es el valor que ocupa la posición central. Si el número de datos es par, la mediana es la media aritmética de los dos centrales.

En esta parte se calcula la media aritmetica de las columnas:

-pago-moneda
-preco-mercado
-promedio-bloque

Cada una cuenta con una variable la cual sera mostrada seguido de un mensaje

In [58]:
```
#Calculo de la Mediana
print("Mediana:")
#Entrada de los datos por medios del array
m1 =datos["pagos-moneda"].median()
m2  =datos["precio-mercado"].median()
m3 =datos["promedio-bloque"].median()
#Mostreo de los datos
print( "\nla Mediana de los pagos en moneda realizados diariamente: ", m1)
print( "\nla Mediana del precio de la moneda en el mercado: ", m2)
print( "\nla Mediana del promedio de la moneda en bloques: ", m3)
```

```
Mediana:

la Mediana de los pagos en moneda realizados diariamente::  666523.0

la Mediana del precio de la moneda en el mercado:  29188.155

la Mediana del promedio de la moneda en bloques:  4724.71015495296
```

Moda

Es el valor que más se repite o, lo que es lo mismo, el que tiene la mayor frecuencia.

En esta parte se calcula la moda de las columnas:

-pago-moneda
-preco-mercado
-promedio-bloque

Cada una cuenta con una variable la cual sera mostrada seguido de un mensaje, y los datos seran guardados a la libreria de panda

In [62]:

```
#Calculo de la Moda
print("Moda:")
#Entrada de los datos por medios del array
mo1 = datos["billetera-usuarios"].mode()
mo2 = datos["pagos-moneda"].mode()
mo3 = datos["pagos-moneda"].mode()
#Mostreo de los datos
print( "\nla Moda de  los pagos en moneda realizados diariamente:  \n\n", mo
print( "\nla Moda del precio de la moneda en el mercado: \n\n", mo2)
print( "\nla Moda del promedio de la moneda en bloques: \n\n", mo3)
#agregandolas a la libreria panda
pd.DataFrame(mo1)
pd.DataFrame(mo2)
pd.DataFrame(mo3)
```

```
Moda:

la Moda de  los pagos en moneda realizados diariamente:

 0     50807793
1     50874784
2     50933404
3     50999025
4     51072983
5     51143211
6     51213490
7     51289691
8     51363967
9     51438971
10    51504055
11    51585531
12    51667919
13    51762365
14    51838178
15    51933930
16    52016198
17    52083351
18    52148832
19    52179995
20    52217981
21    52281159
22    52349422
23    52429897
24    52507724
25    52580626
26    52602380
27    52649995
28    52716021
29    52782166
30    52845759
31    52905086
32    52969897
33    53029649
34    53087838
```

```
35    53134626
36    53193624
37    53256853
38    53320829
39    53375387
40    53425635
41    53486559
42    53539575
43    53602438
44    53662339
45    53730063
46    53830258
47    53897442
48    53969300
49    54030335
50    54089822
51    54156314
52    54224524
53    54309711
dtype: int64

la Moda del precio de la moneda en el mercado:

 0    622738
1    671920
2    727861
dtype: int64

la Moda del promedio de la moneda en bloques:

 0    622738
1    671920
2    727861
dtype: int64
```

Out[62]:

	0
0	622738
1	671920
2	727861

Medidas de tendencias

A continuacion se mostrara las medidas de tendencia central entre esos, valores iniciales, valor de salvamiento y periodos de recuperacion al utilizar la palabra describe se mostrara incluso los quartiles.

In [80]:
```
#Medidas de tendencia central

datos[['promedio-bloque','billetera-usuarios','precio-mercado']]. describe()
```

Out[80]:

	promedio-bloque	billetera-usuarios	precio-mercado
count	366.000000	3.660000e+02	366.000000
mean	4737.817555	5.261965e+07	29757.234098
std	708.186550	1.015115e+06	17802.870974
min	2975.134969	5.080779e+07	9072.420000
25%	4276.447251	5.176236e+07	11683.595000
50%	4724.710155	5.269910e+07	29188.155000
75%	5143.439882	5.347240e+07	46552.595000
max	7236.203883	5.432811e+07	63554.440000

Despues, se ingresaran los datos estadisticos dentro de un array con el valor inicial, valor de salvamento y periodo de recuperacion

In [81]:
```
#Ingreso de los datos al Array con su expacion

df_1 = datos[:7]
print ("Estadisticos de acuerdo a las fechas")
print ("-----------------------------")
df_1[['promedio-bloque','billetera-usuarios','pagos-moneda']]. describe()
```

```
Estadisticos de acuerdo a las fechas
-----------------------------
```

Out[81]:

	promedio-bloque	billetera-usuarios	pagos-moneda
count	7.000000	7.000000e+00	7.000000
mean	3823.440563	5.083219e+07	606070.000000
std	447.318300	2.126225e+04	66468.344398
min	2975.134969	5.080779e+07	484947.000000
25%	3716.450805	5.081368e+07	581253.000000
50%	4027.668831	5.083408e+07	640637.000000
75%	4035.371383	5.084675e+07	646264.000000
max	4257.635762	5.086258e+07	661872.000000

Graficos adicionales

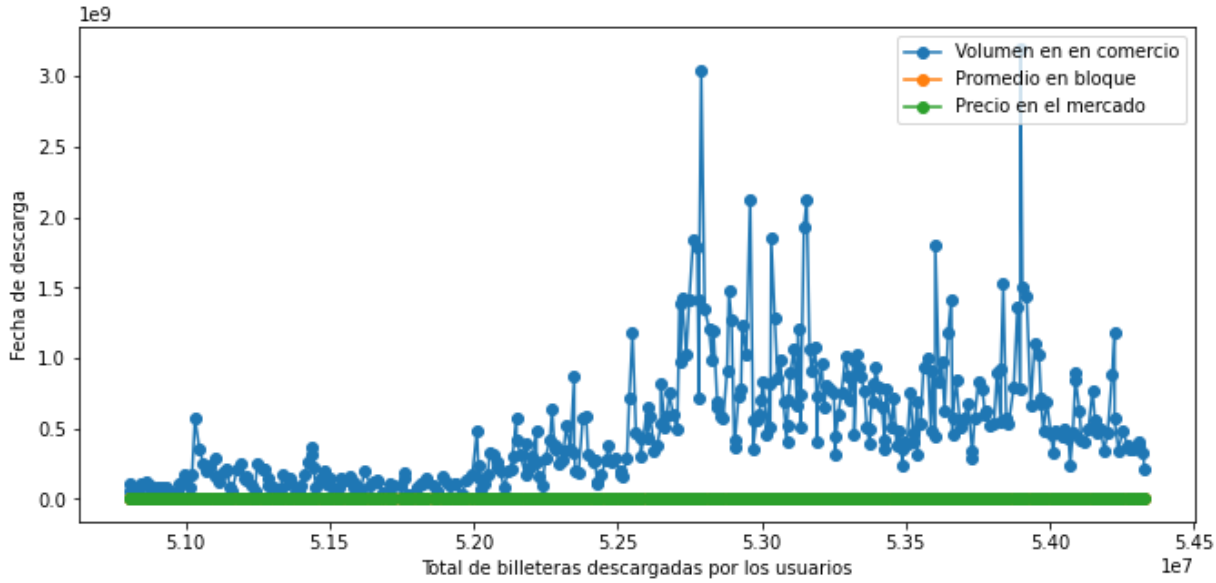
Se obtendran datos de la fuente principal y se guardaran en una variable:

- Se ingresa pincipalmente los datos en las ejes de la x el cual en este caso es la billetera que descargan los usuarios
- S agregan variables las cuales gurdan los datos de las columnas respectivas
- Se realizan las modificaciones concorde a lo que se desee mostrar en el grafico
- y por ultimo se impren los datos para que se muestre el grafico

In [70]:
```
#fecha  total-bitcoins  volumen-comercio    promedio-bloque billetera-usuari

#Otros Graficos Adicionales
#Ingreso de las categoria en las ejes de la X
x = datos["billetera-usuarios"]
#Ingreso de los datos al Array
t1 = datos["volumen-comercio"]
t2 = datos["promedio-bloque"]
t3 = datos["precio-mercado"]
#Modificacion del grafico
plt.figure(figsize=(11,5))
plt.plot(x,t1,x,t2,x,t3,marker='o')
#Imprecion de los datos y modificacion del grafico
plt.xlabel('Total de billeteras descargadas por los usuarios')
plt.ylabel('Fecha de descarga')
plt.legend(('Volumen en en comercio','Promedio en bloque', 'Precio en el mer
```

Out[70]: <matplotlib.legend.Legend at 0x237e2fb6280>

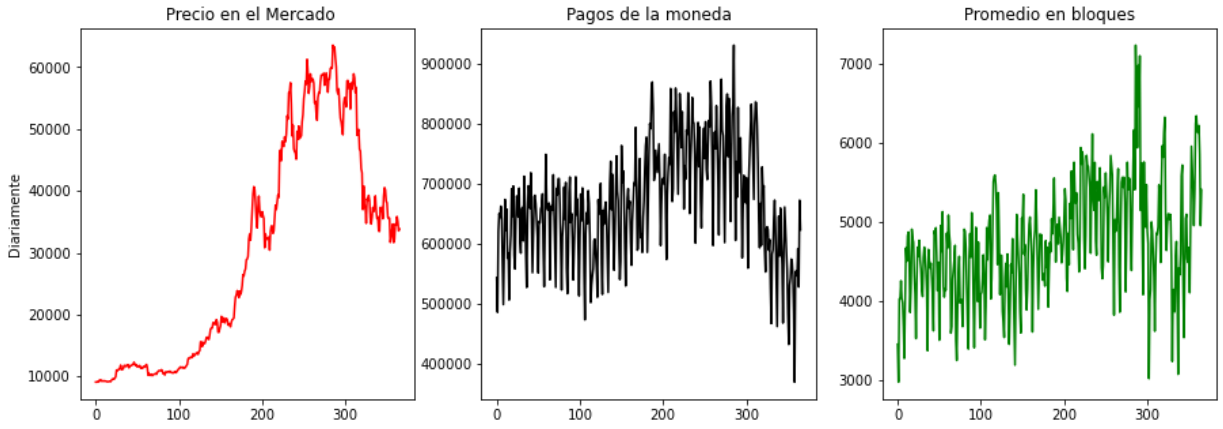


Grafica para cada columna utilizando solo una variable para las y, esta sera utilizada para cada una de las otras columnas:

-En esta parte se ubica el rango de los datos con los cuales se esta tra
najando
-Luego se configuran las dimensiones en el grafico
-Se ingresa una variable la cual guarde los datos de la columna a utiliz
ar
-Luego se etalla el primer grafico donde se ubica el titulo del grafico
y el varlos de la y
-Despues se detalla el segundo grafico donde se agrega la columna con la
cuel se deba trabajar
-Por ultimo la otra columna y luegose se ejecuta para poder mostrar los
datos respectivos.

In [71]:
```
1
2
3  #Entra del rango al Array con los datos que estan en la fuente de datos
4  x = range(366)
5  #Configuracion de las dimenciones que el grafico optendra
6  plt.figure(figsize=(15,5))
7  plt.subplot(131)
8  #Entrada de los datos a las array
9  t1 = datos["precio-mercado"]
10 #'r'=> (rojo) es el color que se le puede agregar es una simbolo
11 p1 = plt.plot(x,t1, 'red')
12 plt.ylabel('Diariamente')
13 plt.title(' Precio en el Mercado')
14 plt.subplot(132)
15 t2 = datos["pagos-moneda"]
16 #'k'=> (negro)
17 p1 = plt.plot(x,t2,'black')
18 plt.title('Pagos de la moneda')
19 plt.subplot(133)
20 t3 = datos["promedio-bloque"]
21 #'g'=> (verde)
22 p1 = plt.plot(x,t3,'green')
23 plt.title(' Promedio en bloques ')
24
```

Out[71]: Text(0.5, 1.0, ' Promedio en bloques ')



Datos unicos de la tabla

A continuacion, se mostraran los datos unicos del la cantidad de usuarios que tienen descargadas sus billeteras lo cual se guardara en una variable llamada dfclases

In [72]:
```
1
2 #fecha  total-bitcoins  volumen-comercio    promedio-bloque billetera-usuari
3
4 # OBTENER LOS DATOS UNICOS DE LA TABLA
5 #Agregar la columna de valor inicial a una variable
6 lis = datos["billetera-usuarios"].unique()
7 #Para poder sacar los valores unicos de la tabla
8 dfclases=pd.DataFrame(lis,columns=["billetera-usuarios"])
9 dfclases
```

Out[72]:

	billetera-usuarios
0	50807793
1	50819572
2	50834080
3	50844380
4	50849128
...	...
307	54274719
308	54290217
309	54309711
310	54323892
311	54328109

312 rows × 1 columns

Frecuencias absolutas

La frecuencia absoluta es una medida estadística que nos da información acerca de la cantidad de veces que se repite un suceso al realizar un número determinado de experimentos aleatorios.

Para obtener estos datos se debe crear una lista con los valores de las frecuencias y se agrega a la columna dataframe a la cual se le ha puesto Fi, y se muestran los datos unicos guardados anteriormente

In [82]:
```
#TABLA DE FRECUENCIAS ABSOLUTAS
# OBTENER FRECUENCIAS ABSOLUTAS DE CADA CLASE
datafi=pd.crosstab(index=datos["billetera-usuarios"], columns = "fi")
# Creamos una lista con los valores de las frecuencias
li = datafi.values
# agregamos una columna al dataframe
dfclases["fi"] = li
#observamos dfclase
dfclases
```

Out[82]:

	billetera-usuarios	fi	hi	FA	HI
0	50807793	2	0.005464	2	0.005464
1	50819572	1	0.002732	3	0.008197
2	50834080	1	0.002732	4	0.010929
3	50844380	1	0.002732	5	0.013661
4	50849128	1	0.002732	6	0.016393
...	...	...	...	...	...
307	54274719	1	0.002732	361	0.986339
308	54290217	1	0.002732	362	0.989071
309	54309711	2	0.005464	364	0.994536
310	54323892	1	0.002732	365	0.997268
311	54328109	1	0.002732	366	1.000000

312 rows × 5 columns

Total de los datos anteriores

In [74]:
```
total = dfclases.sum(axis=0)
total

```

Out[74]:
```
billetera-usuarios    16418741387
fi                              366
dtype: int64
```

Frecuencia Relativa

La frecuencia relativa es una medida estadística que se calcula como el cociente de la frecuencia absoluta de algún valor de la población/muestra (fi) entre el total de valores que componen la población/muestra (N).

Para obtener los datos de la frecuncia relativa se calcula y se agrega la columna adicional con el resultado de los datos utlizando algunos datos de la Frecuencia Absoluta

In [75]:
```
# Columna de Frecuencia relativa
total = dfclases.sum(axis=0)
datahi = dfclases["fi"]/total["fi"] # aqui calculamos la frecuencia
datahi.values
# agregamos nueva columna de frecuencia relativa
dfclases["hi"] = datahi
dfclases
```

Out[75]:

	billetera-usuarios	fi	hi
0	50807793	2	0.005464
1	50819572	1	0.002732
2	50834080	1	0.002732
3	50844380	1	0.002732
4	50849128	1	0.002732
...	...	...	...
307	54274719	1	0.002732
308	54290217	1	0.002732
309	54309711	2	0.005464
310	54323892	1	0.002732
311	54328109	1	0.002732

312 rows × 3 columns

Mostramos el total de los datos utilizando las siguientes lineas de codigo

In [76]:
```
total1 = dfclases.sum(axis=0)
total1
```

Out[76]:
```
billetera-usuarios    1.641874e+10
fi                    3.660000e+02
hi                    1.000000e+00
dtype: float64
```

Suma de frecuencias relativas

Realizamos una suma de frecuencias relativas utilizando algunos de los datos anteriores los guardamos en la variable utilizada y se mostraran cada uno de los datos mediante un for para que se puedan sumar los datos y asi mostrarlos de una mejor manera

In [77]:

```
# la suma de las frecuencias Relativas nos da 1
# aqui vamos a calcular la frecuencia absoluta
FA = dfclases["fi"].values
# obtenemos FA
a=[]
b=0
for c in FA:
 b = c + b
 a.append(b)
dfclases["FA"] = a
HI = dfclases["hi"].values
# obtenemos HI
a=[]
b=0
for c in HI:
 b = c + b
 a.append(b)
dfclases["HI"] = a
dfclases
```

Out[77]:

	billetera-usuarios	fi	hi	FA	HI
0	50807793	2	0.005464	2	0.005464
1	50819572	1	0.002732	3	0.008197
2	50834080	1	0.002732	4	0.010929
3	50844380	1	0.002732	5	0.013661
4	50849128	1	0.002732	6	0.016393
...	...	...	...	...	...
307	54274719	1	0.002732	361	0.986339
308	54290217	1	0.002732	362	0.989071
309	54309711	2	0.005464	364	0.994536
310	54323892	1	0.002732	365	0.997268
311	54328109	1	0.002732	366	1.000000

312 rows × 5 columns

Deteminamos el valor total 2 y lo mostramos

In [78]:
```
total2 = dfclases.sum(axis=0)
total2
```

Out[78]:
```
billetera-usuarios    1.641874e+10
fi                    3.660000e+02
hi                    1.000000e+00
FA                    5.744000e+04
HI                    1.569399e+02
dtype: float64
```

Mostramos toda la informacion con la siguiente linea de codigo

In [79]:
```
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype         
---  ------              --------------  -----         
 0   fecha               366 non-null    datetime64[ns]
 1   total-bitcoins      366 non-null    float64       
 2   volumen-comercio    366 non-null    float64       
 3   promedio-bloque     366 non-null    float64       
 4   billetera-usuarios  366 non-null    int64         
 5   pagos-moneda        366 non-null    int64         
 6   precio-mercado      366 non-null    float64       
dtypes: datetime64[ns](1), float64(4), int64(2)
memory usage: 20.1 KB
```

Implementacion del módulo de generación de números aleatorios, variables aleatorias, y test de los mismos

Metodo de los cuadrados mediados

Para un algoritmo en el cual se utilicen los cuadrados medios en el lenguaje de Python se lo realiza de la siguiente manera:

- Se selecciona el valor inicial r
- Se determina el numero de digitos
- Se debe almacenar una lista
- Se realiza una lista
- Se elva al cuadrado dentro de un while
- Y por ultimo se mostraran los datos

In [5]:

```
# Método de los cuadrados medios
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
n=500
# seleccionamos el valor inicial r
r= 1845
l=len(str(r)) # determinamos el número de dígitos
lista = [] # almacenamos en una lista
lista2 = []
i=1
#while len(lista) == len(set(lista)):
while i < n:
    x=str(r*r) # Elevamos al cuadrado r
    if l % 2 == 0:
        x = x.zfill(l*2)
    else:
        x = x.zfill(l)
    y=(len(x)-l)/2
    y=int(y)
    r=int(x[y:y+l])
    lista.append(r)
    lista2.append(x)
    i=i+1

datos = pd.DataFrame({'X2':lista2,'Xi':lista})
dfrac = datos["Xi"]/10**l
datos["ri"] = dfrac
#df.head()
datos
```

Out[5]:

	X2	Xi	ri
0	03404025	4040	0.4040
1	16321600	3216	0.3216
2	10342656	3426	0.3426
3	11737476	7374	0.7374
4	54375876	3758	0.3758
...	...	...	...
494	92160000	1600	0.1600
495	02560000	5600	0.5600
496	31360000	3600	0.3600
497	12960000	9600	0.9600
498	92160000	1600	0.1600

499 rows × 3 columns

Esta función devuelve las últimas filas del objeto según la posición. Es útil para verificar datos

rápidamente, por ejemplo, después de ordenar o agregar filas.

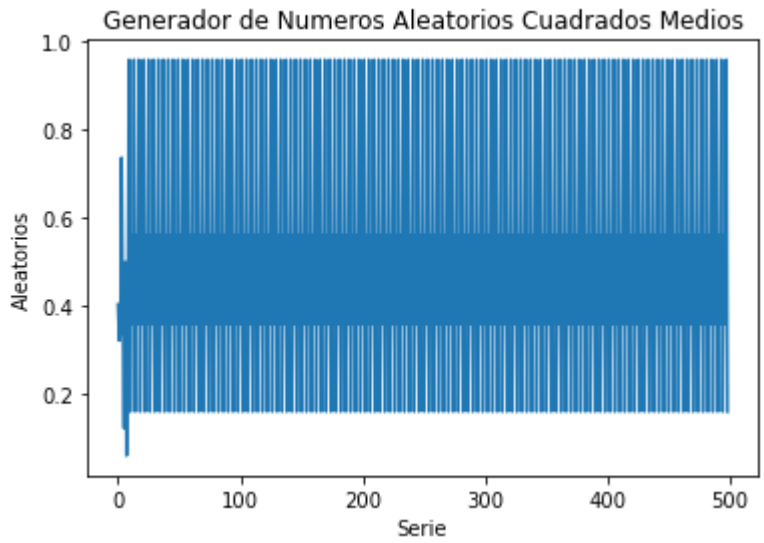
In [6]:
```
1 df.tail()
```

Out[6]:

	X2	Xi	ri
494	92160000	1600	0.16
495	02560000	5600	0.56
496	31360000	3600	0.36
497	12960000	9600	0.96
498	92160000	1600	0.16

Se grafican los numeros aleatorios generados, y se selecciona el dataframe de los numeros generados. En el grafico muestra los datos correspondientes.

In [8]:
```
1 x1=df['ri']
2 plt.plot(x1)
3 plt.title('Generador de Numeros Aleatorios Cuadrados Medios')
4 plt.xlabel('Serie')
5 plt.ylabel('Aleatorios')
6 plt.show()
```



Metodo congruenciales lineales

Este metodo permite obtener una secuencia de numeros pseudoaleatorios calculados con una funcion lineal definida a trozos discontinua. Es uno de los metodos mas antiguos y conocidos para la geneeracion de numeros pseudoaleatorios.

In [12]:

```
# Generador de números aleatorios Congruencia lineal
n, m, a, x0, c = 20, 100, 101, 4, 457
x = [1] * n
r = [0.1] * n
print (" Método de Congruencia Lineal ")
print("------------------------------")
print ("n=cantidad de números generados : ", n)
print()
print ("m : ", m)
print ("a : ", a)
print ("c : ", c)
print ("Xo : ", x0 )
for i in range(0, n):
 x[i] = ((a*x0)+c) % m
 x0 = x[i]
 r[i] = x0 / m
# llenamos nuestro DataFrame
d = {'Xn': x, 'ri': r }
df = pd.DataFrame(data=d)
df
```

```
 Método de Congruencia Lineal
------------------------------
n=cantidad de números generados :  20

m :  100
a :  101
c :  457
Xo :  4
```

Out[12]:

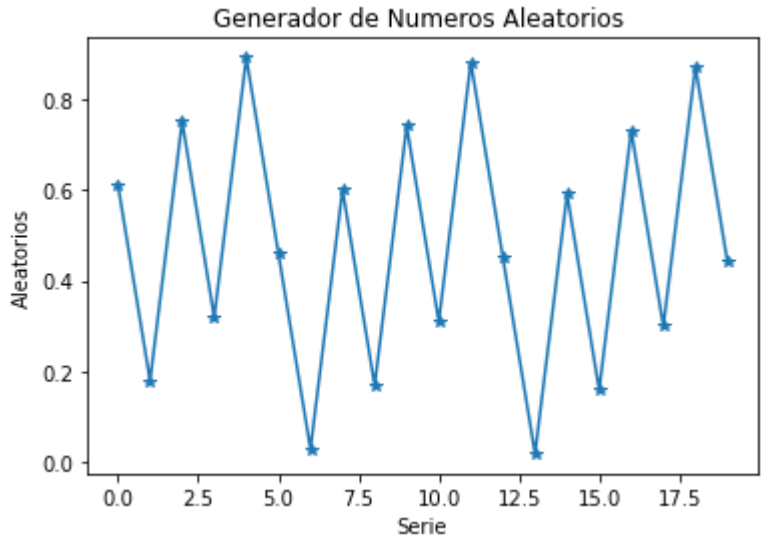
	Xn	ri
0	61	0.61
1	18	0.18
2	75	0.75
3	32	0.32
4	89	0.89
5	46	0.46
6	3	0.03
7	60	0.60
8	17	0.17
9	74	0.74
10	31	0.31
11	88	0.88
12	45	0.45
13	2	0.02

	Xn	ri
14	59	0.59
15	16	0.16
16	73	0.73
17	30	0.30
18	87	0.87
19	44	0.44

Ahora se genera un grafico donde se muestre el generador de los numeros aleatorios, en el cual se encuentra los titulos de cada lado.

In [15]:
```
# Graficamos los numeros generados
plt.plot(r,marker='*')
plt.title('Generador de Numeros Aleatorios ')
plt.xlabel('Serie')
plt.ylabel('Aleatorios')
```

Out[15]: Text(0, 0.5, 'Aleatorios')



Metodo congruenciales multiplicativo

Este algoritmo se utiliza para generar numeros pseudo aleaorios. Este al igua que el congruencial mixto genera una sucesion de numeros pseudos alaeatorios en la cual el sucesor, del numero pseudo aleatorio, es determinado justo a partir del numero pseudo aleatorio

In [16]:
```
n, m, a, x0 = 20, 100, 747, 123
x = [1] * n
r = [0.1] * n
print (" Generador Congruencial multiplicativo")
print ("--------------------------------------")
for i in range(0, n):
 x[i] = (a*x0) % m
 x0 = x[i]
 r[i] = x0 / m
d = {'Xn': x, 'ri': r }
df =  pd.DataFrame(data=d)
df

```

```
 Generador Congruencial multiplicativo
--------------------------------------
```

Out[16]:

	Xn	ri
0	81	0.81
1	7	0.07
2	29	0.29
3	63	0.63
4	61	0.61
5	67	0.67
6	49	0.49
7	3	0.03
8	41	0.41
9	27	0.27
10	69	0.69
11	43	0.43
12	21	0.21
13	87	0.87
14	89	0.89
15	83	0.83
16	1	0.01
17	47	0.47
18	9	0.09
19	23	0.23

Generador excel-07

In [18]:

```
n, m, a, x0 = 20, 30307, 172, 172
x = [1] * n
r = [0.1] * n
print (" Generador Congruencial multiplicativo")
print ("--------------------------------------")
for i in range(0, n):
 x[i] = (a*x0) % m
 x0 = x[i]
 r[i] = x0 / m
d = {'Xn': x, 'ri': r }
df1 = pd.DataFrame(data=d)
df1.head()
```

```
 Generador Congruencial multiplicativo
--------------------------------------
```

Out[18]:

	Xn	ri
0	29584	0.976144
1	27179	0.896790
2	7510	0.247798
3	18826	0.621177
4	25530	0.842380

Generador excel-07

In [27]:
```
# - Xn+1 = (170Xn ) (mod 30323)
n, m, a, x0 = 20, 30323, 170, 340
x = [1] * n
r = [0.1] * n
print (" Generador Congruencial multiplicativo")
print ("-------------------------------------")
for i in range(0, n):
 x[i] = (a*x0) % m
 x0 = x[i]
 r[i] = x0 / m
d = {'Xn': x, 'ri': r }
df2 = pd.DataFrame(data=d)
df2.head()
```

```
 Generador Congruencial multiplicativo
-------------------------------------
```

Out[27]:

	Xn	ri
0	27477	0.906144
1	1348	0.044455
2	16899	0.557300
3	22468	0.740956
4	29185	0.962471

Esta función devuelve las últimas filas del objeto según la posición. Es útil para verificar datos rápidamente de los datos anteriores

In [23]:
```
df2.tail()
```

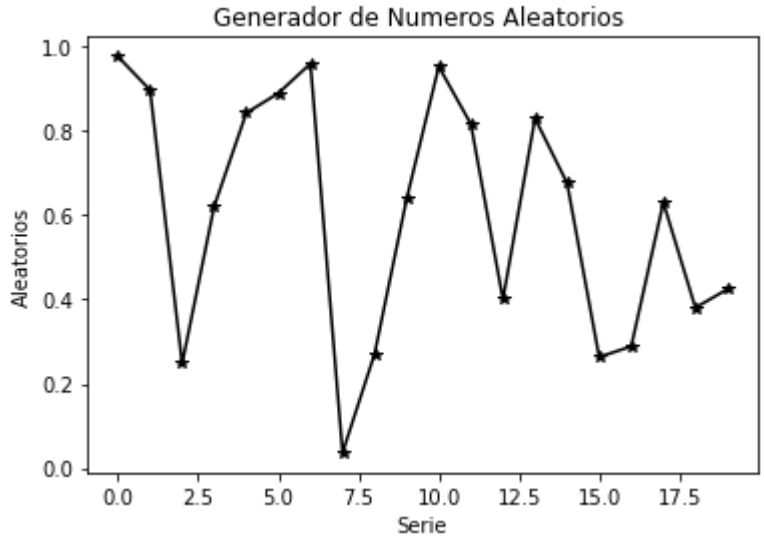
Out[23]:

	Xn	ri
15	8516	0.280843
16	22539	0.743297
17	10932	0.360518
18	8737	0.288131
19	29786	0.982291

A continuacion muestra un Generador de Numeros Aleatorios dentro de un grafico

In [19]:
```
1
2 plt.plot(r,'k-', marker='*',)
3 plt.title('Generador de Numeros Aleatorios ')
4 plt.xlabel('Serie')
5 plt.ylabel('Aleatorios')
6
```

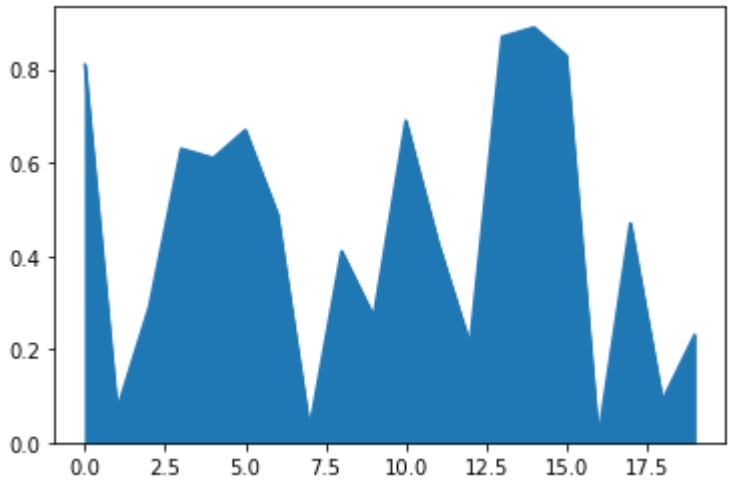
Out[19]: Text(0, 0.5, 'Aleatorios')



A continuacion muestra un Generador de Numeros Aleatorios dentro de un grafico el cual se muestra por area las areas ocupadas se encuentran pintadas.

In [20]:
```
1 x1 = df["ri"]
2 x1.plot.area()
```

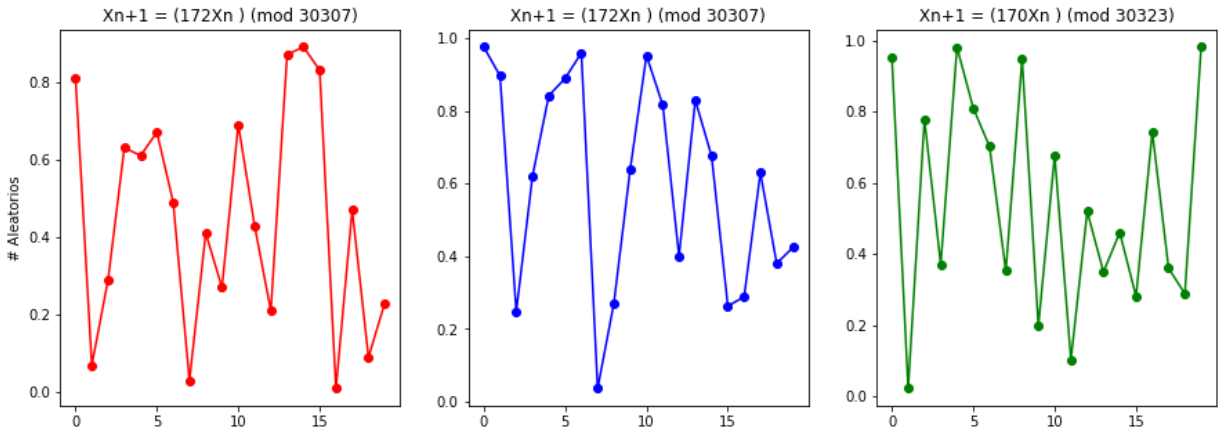
Out[20]: <AxesSubplot:>



A continuacion muestra tres graficos en los cuales se define los numeros aleatorios en tres colores diferentes.

In [26]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
x1 = df["ri"]
x2 = df1["ri"]
x3 = df2["ri"]
x = np.arange(n)
#fig = plt.figure()
plt.figure(figsize=(15,5))
# Gráfico 1
plt.subplot(131)
x1 = df["ri"]
p1, = plt.plot(x,x1,'r-',marker='o')
plt.ylabel('# Aleatorios')
plt.title(' Xn+1 = (172Xn ) (mod 30307) ')
# Gráfico 2
plt.subplot(132)
x2 = df1["ri"]
p1, = plt.plot(x,x2,'b-',marker='o')
plt.title(' Xn+1 = (172Xn ) (mod 30307) ')
# Gráfico 2
plt.subplot(133)
x3 = df2["ri"]
p1, = plt.plot(x,x3,'g-',marker='o')
plt.title(' Xn+1 = (170Xn ) (mod 30323) ')
```

Out[26]: Text(0.5, 1.0, ' Xn+1 = (170Xn) (mod 30323) ')



Este es un generador de números aleatorios con congruencia lineal utilizado por Borland

In [28]:
```
n, m, a, x0, c = 200, 2**32, 22695477, 4, 1
x = [1] * n
r = [0.1] * n
print (" Método de Congruencia Lineal Utilizado por:")
print("Borland C/C++ xi+1=22695477xi + 1 mod 2^32 ")
print ("n=cantidad de números generados : ", n)
print()
print ("m : ", m)
print ("a : ", a)
print ("c : ", c)
print ("Xo : ", x0 )
for i in range(0, n):
 x[i] = ((a*x0)+c) % m
 x0 = x[i]
 r[i] = x0 / m
# llenamos nuestro DataFrame
d = {'Xn': x, 'ri': r }
dfMCL = pd.DataFrame(data=d)
dfMCL
```

```
 Método de Congruencia Lineal Utilizado por:
Borland C/C++ xi+1=22695477xi + 1 mod 2^32
n=cantidad de números generados :  200

m :  4294967296
a :  22695477
c :  1
Xo :  4
```

Out[28]:

	Xn	ri
0	90781909	0.021137
1	4261128730	0.992121
2	705831779	0.164339
3	274169216	0.063835
4	2841246593	0.661529
...	...	...
195	3761451712	0.875781
196	2609906113	0.607666
197	1949510390	0.453906
198	3056261359	0.711591
199	4160600956	0.968715

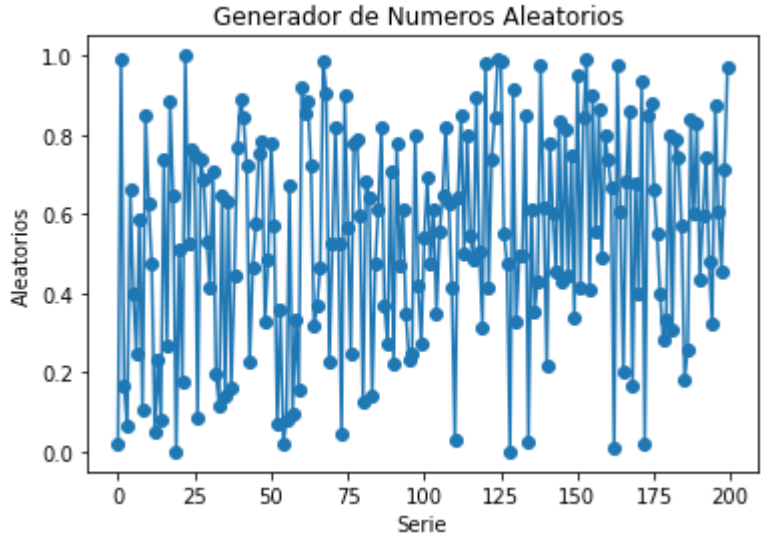
200 rows × 2 columns

Graficos de Dispersion

Los gráficos de dispersión se usan para trazar puntos de datos en un eje vertical y uno horizontal,

mediante lo que se trata de mostrar cuánto afecta una variable a otra. Cada fila de la tabla de datos la representa un indicador cuya posición depende de sus valores en las columnas que se establecen en los ejes X e Y.

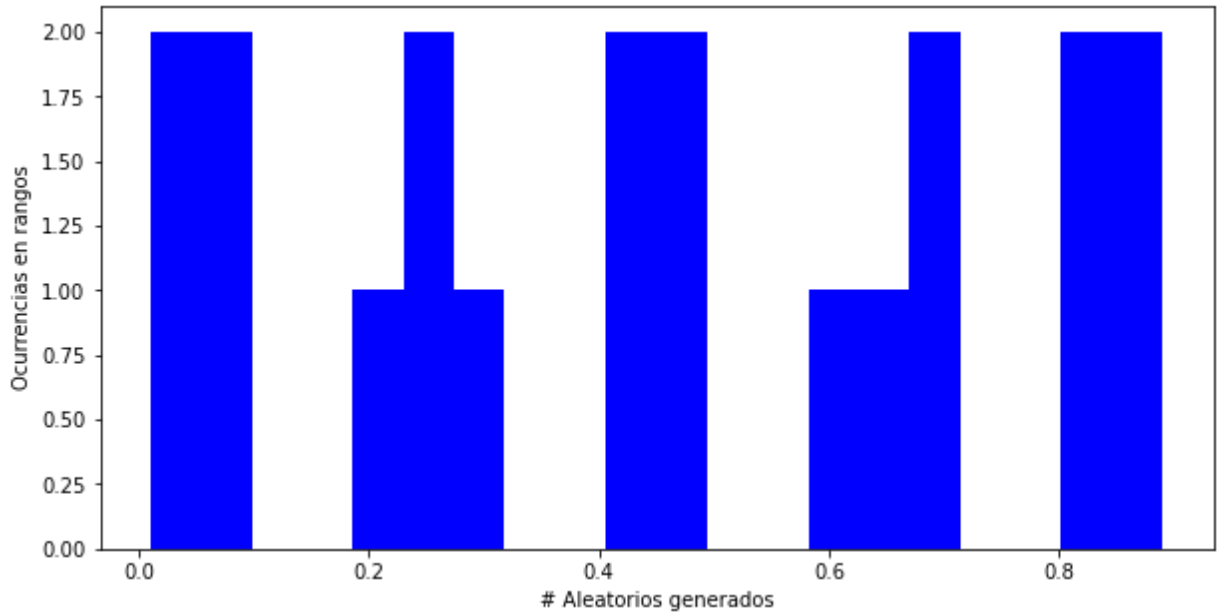
In [29]:
```
# Gráficos de dispersión
plt.title('Generador de Numeros Aleatorios ')
plt.xlabel('Serie')
plt.ylabel('Aleatorios')
#plt.scatter(df.index,df['ri'])
plt.plot(r,marker='o')
plt.show()
```



Esta es otra de las graficas con los datos aleatrios generados

In [21]:
```
rS= df['ri']
#rS
plt.figure(figsize=(10,5))
plt.hist(rS,bins=20,color='blue')
plt.xlabel('# Aleatorios generados')
plt.ylabel('Ocurrencias en rangos')
plt.show()

```



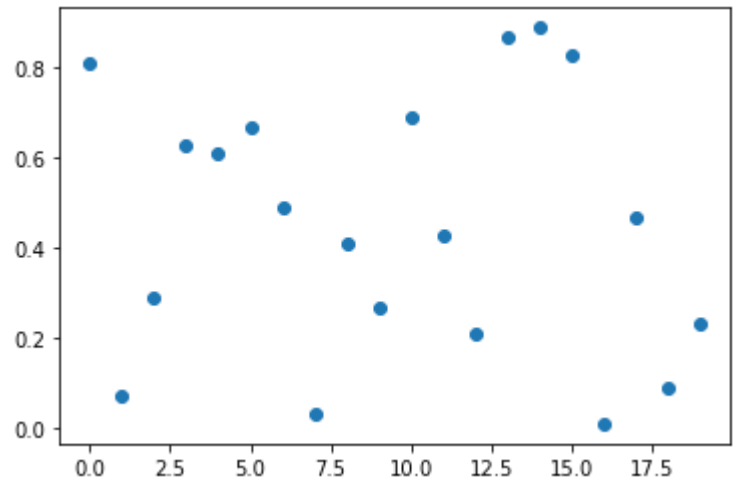
Para este grafica a continuacion utilizamos scatter

In [31]:
```

plt.scatter(df.index,df['ri'])

```

Out[31]: `<matplotlib.collections.PathCollection at 0x21bd772fa00>`



Libreria de random

In [32]:
```
import random
help(random)

```

```
Help on module random:

NAME
    random - Random variable generators.

MODULE REFERENCE
    https://docs.python.org/3.8/library/random (https://docs.python.org/3.8/library/random)

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
        integers
        --------
               uniform within range
```

La funcion numpy.random.rand genera un array del tamaño indicado conteniendo números aleatorios extraídos del intervalo a partir de una distribución uniforme

In [22]:
```
np.random.rand()

```

Out[22]:
```
0.7279637957230378
```

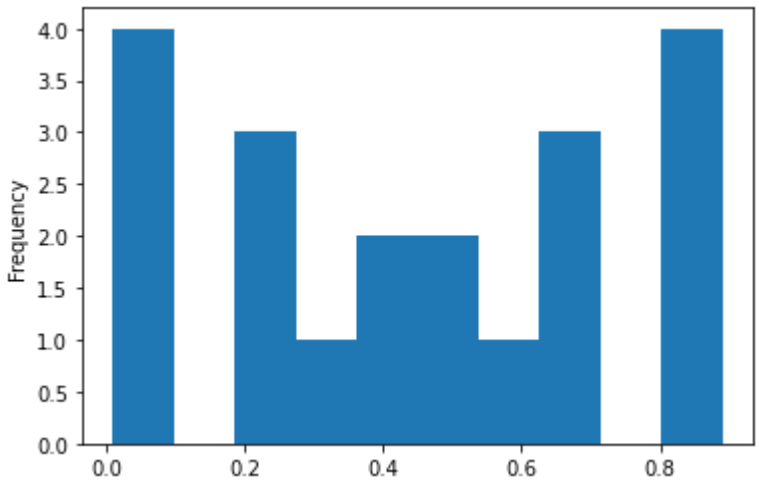
Numero aleatorio con distribucion normal

Una distribución de valores acumulados alrededor de un promedio (llamado la “media”) se conoce como una distribución “normal”. También se llama la distribución gaussiana (llamada así por el matemático Carl Friedrich Gauss). Cuando graficas la distribución, obtendrás una campana de Gauss

In [34]:
```
# Generamos otro número aleatorio con distribución normal
lista=np.random.randn(100)
lista
#lista.plot.area()
dfl =pd.DataFrame({'X1':lista})
dfl.head()
x1.plot.hist()

```

Out[34]: <AxesSubplot:ylabel='Frequency'>



Se genera un arreglo de 5 números aleatorios con distribución uniforme

In [35]:
```
np.random.rand(5)
```

Out[35]:
```
array([0.51346164, 0.17800397, 0.84639189, 0.89964033, 0.10573832])
```

Se genera una matriz de números aleatorios de 2 filas y 4 columnas

In [36]:
```
np.random.rand(2,4)
```

Out[36]:
```
array([[0.11577668, 0.62962282, 0.13925593, 0.00544046],
       [0.64573667, 0.76542872, 0.70218345, 0.1395374 ]])
```

Utilizando la libreria random se genera el primer numero con la misma semilla y utilizando un

segundo numero con la misma semilla

In [37]:
```
import random
# primer número generado con la misma semilla
random.seed(10)
print(random.random())
print("Observamos como el nuevo numero generado va a ser igual")
# segundo número generado con la misma semilla
random.seed(10)
print(random.random())
```

```
0.5714025946899135
Observamos como el nuevo numero generado va a ser igual
0.5714025946899135
```

Se puede cambiar la semilla de la secuencia de los numeros aleatorios generados guardados en el dataFrame

In [40]:
```
np.random.seed(100)
x = np.random.rand(50)
df = pd.DataFrame({'Xn': x})
df.head()


```

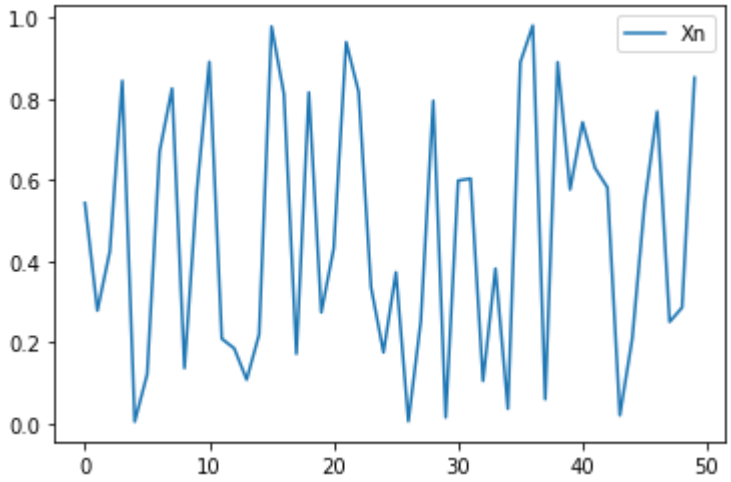
Out[40]:

	Xn
0	0.543405
1	0.278369
2	0.424518
3	0.844776
4	0.004719

Esta es para realizar la grafica de los datos anteriores con plot

In [41]:
```
1 df.plot()
```

Out[41]: <AxesSubplot:>



Se realiza una segunda serie de np random seed seleccionando 100 datos

In [42]:
```
# Serie 2
np.random.seed(100)
y = np.random.rand(61)
print(y)
dfy = pd.DataFrame({'Yn': y})
dfy

```

```
[0.54340494 0.27836939 0.42451759 0.84477613 0.00471886 0.12156912
 0.67074908 0.82585276 0.13670659 0.57509333 0.89132195 0.20920212
 0.18532822 0.10837689 0.21969749 0.97862378 0.81168315 0.17194101
 0.81622475 0.27407375 0.43170418 0.94002982 0.81764938 0.33611195
 0.17541045 0.37283205 0.00568851 0.25242635 0.79566251 0.01525497
 0.59884338 0.60380454 0.10514769 0.38194344 0.03647606 0.89041156
 0.98092086 0.05994199 0.89054594 0.5769015  0.74247969 0.63018394
 0.58184219 0.02043913 0.21002658 0.54468488 0.76911517 0.25069523
 0.28589569 0.85239509 0.97500649 0.88485329 0.35950784 0.59885895
 0.35479561 0.34019022 0.17808099 0.23769421 0.04486228 0.50543143
 0.37625245]
```

Out[42]:

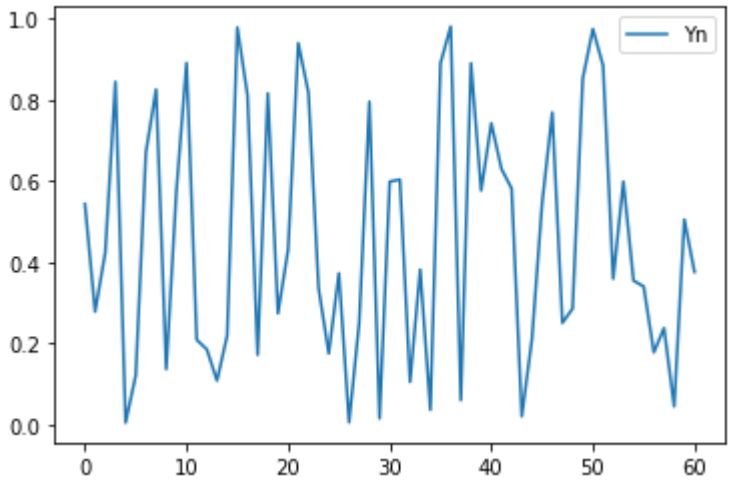
	Yn
0	0.543405
1	0.278369
2	0.424518
3	0.844776
4	0.004719
...	...
56	0.178081
57	0.237694
58	0.044862
59	0.505431
60	0.376252

61 rows × 1 columns

Y en esta parte se muestra un grafico con los datos anteriores

In [43]:
```
1 dfy.plot()
2
```

Out[43]: <AxesSubplot:>



La biblioteca random contiene una serie de funciones relacionadas con los valores aleatorios. El método devuelve un elemento seleccionado de número entero del rango especificado.

In [44]:
```
1 np.random.randint(10, size=10)
2
```

Out[44]: array([2, 7, 1, 6, 6, 0, 7, 2, 3, 5])

Test de numeros aleatorios

Frecuencias en bloques

El enfoque de esta prueba es la proporción de unos dentro de bloques de m bits. El proósito es determinar si una frecuencia de unos dentro de un bloque es aproximadamente m/2, como se esperaría de la aleatoreidad. Haciendo bloques de tamaño 1, estaríamos haciendo la prueba de frecuencia

In [45]:

```
# Prueba de Frecuencias en bloques

import math
from scipy.stats import chi2
rS
lista1=np.array(rS)
vector = []
# llenamos de 0 y 1
for i in range (0,len(lista1)-1):
    if lista1[i] < lista1[i+1]:
        vector.append(0)
    else:
        vector.append(1)
#vector
m = 10          # numero de bloques
n = len(vector)     # longuitud de la lista
N = int(n/m)        # número de elementos de cada bloque

blocks = []
i = 0
while(len(blocks) != N):
    blocks.append(vector[i:(i+m)])
    i = i + m
sumas = []
for i in range(len(blocks)):
    sumas.append(0)
    for j in range(len(blocks[i])):
        sumas[i] = sumas[i] + blocks[i][j]/float(m)
x_2 = 0
for i in range(N):
    x_2 = x_2 + math.pow((sumas[i] - .5),2)
x_2 = x_2 * 4*m
df = m - 1
p_value = chi2.cdf(x_2, df)
if p_value < 0.01:
    print ("Test de frecuencia en Bloques: Not Pasa")
else:
    print ("Test de frecuencia en Bloques : Aprobado")

print ("Valor de P_value es %3f y el de Chi cuadrado es %3f " % (p_value, x_
print ("La proporción del bloque de sumas es:")

titulo = []
n=20
for i in range(n):
    titulo.append(float(i))

print(titulo)
print("Sumas")
print (sumas)

```

```
Test de frecuencia en Bloques: Not Pasa
Valor de P_value es 0.000000 y el de Chi cuadrado es 0.000000
La proporción del bloque de sumas es:
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 1
```

```
4.0, 15.0, 16.0, 17.0, 18.0, 19.0]
Sumas
[0.5]
```

In [46]:

```
import numpy as np
import pandas as pd
import math
from scipy.stats import chi2_contingency
import numpy as np
datos=np. array([[0.2452, 0.4896, 0.6204, 0.8996, 0.9708],
        [0.0123, 0.2963, 0.5128, 0.7308, 0.7793],
        [0.0961, 0.3520, 0.3903, 0.4068, 0.5286]])
vector=[]

m=3
n=len(vector)
N= int(n/m)
bloques=[]

i=0
while(len(bloques) !=N):
    bloques.append(vector[i:(i+m)])
    i=i+m
sumas =[]
for i in range(len(bloques)):
    sumas.append(0)
    for j in range(len(bloques[i])):
        sumas[i]=sumas[i]+ bloques[i][j]/float(m)
x_2=0
for i in range(N):
    x_2=x_2 + math.pow((suma[i]- .4),2)
x_2 = x_2 * 4*m
df=m-1

print(chi2_contingency(datos))
```

```
(0.21397679568267983, 0.9999949877744493, 8, array([[0.15558419, 0.50067662, 0.
67034083, 0.89636911, 1.00262924],
       [0.11245801, 0.3618947 , 0.4845299 , 0.64790569, 0.72471171],
       [0.0855578 , 0.27532868, 0.36862927, 0.4929252 , 0.55135905]]))
```

Chi cuadrado

La prueba Chi-Cuadrada en lugar de medir la diferencia de cada punto entre la muestra y la desviación verdadera, checa la desviación del valor esperado

In [47]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import math
import scipy.stats

#Matriz ingresada por el usuario
matrix=([[0.2452, 0.4896, 0.6204, 0.8996, 0.9708],
    [0.0123, 0.2963, 0.5128, 0.7308, 0.7793],
    [0.0961, 0.3520, 0.3903, 0.4068, 0.5286]])
matrix

#Seleccion de filas de la matriz
fila1=matrix[0]
fila2=matrix[1]
fila3=matrix[2]
gradoLibertad=5

# Establecer intervalos Fila 1
bins = [0.00, 0.25, 0.50, 0.75, 0.99]
counts, bin_edges = np.histogram(fila1, bins)
M1 = []
inter=[]
# Calcular Frecuencias Fila 1
#print(f"INTERVALO M1")

for low, hight, count in zip(bin_edges, np.roll(bin_edges, -1), counts):
    #print(f"{f'{low}-{hight}': <10} {count}")
    M1.append(count)
    inter.append(f'{low} - {hight}')

# Calcular chi cuadrado Fila 1
M1X1=(((M1[0]-5/4)**2)+((M1[1]-5/4)**2)+((M1[2]-5/4)**2)+((M1[3]-5/4)**2))*4

#print("Chi cuadrado del valor de la serie 3: " , M1X1)

#Guardar dato de chi cuadrado
intervalo1=M1X1

#-------------------------------------------------------------------------
counts, bin_edges = np.histogram(fila2, bins)
M2 = []
for low, hight, count in zip(bin_edges, np.roll(bin_edges, -1), counts):
    M2.append(count)
M2X1=(((M2[0]-5/4)**2)+((M2[1]-5/4)**2)+((M2[2]-5/4)**2)+((M2[3]-5/4)**2))*4
intervalo2=M2X1

#-------------------------------------------------------------------------
counts, bin_edges = np.histogram(fila3, bins)
M3 = []
for low, hight, count in zip(bin_edges, np.roll(bin_edges, -1), counts):
    M3.append(count)
M3X1=(((M3[0]-5/4)**2)+((M3[1]-5/4)**2)+((M3[2]-5/4)**2)+((M3[3]-5/4)**2))*4
intervalo3=M3X1

```

```
57 df=pd.DataFrame({'Intervalos':inter,'m1':M1,'m2':M2,'m3':M3})
58
59 df.loc[4]=['Chi',intervalo1,intervalo2,intervalo3]
60 df
```

Out[47]:

	Intervalos	m1	m2	m3
0	0.0 - 0.25	1.0	1.0	1.0
1	0.25 - 0.5	1.0	1.0	3.0
2	0.5 - 0.75	1.0	2.0	1.0
3	0.75 - 0.99	2.0	1.0	0.0
4	Chi	0.6	0.6	3.8

In [ ]:

```
1
```