Simulacion Juego: "Los Chanchitos Constructores"

In [ ]:
```
from random import randint
import random
import time
import sys
import pandas as pd
from scipy import stats
import math
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.graphics.gofplots import qqplot
import pingouin as pg
from scipy.stats import shapiro, kstest, lognorm, gamma, chi2
```

Funciones que permiten el ingreso de los datos de cantidad de jugadores, partidas y modalidad de juego. Al mismo tiempo, las funciones validan que el ingreso de los datos sea correcto.

- Cantidad de Jugadores:

In [ ]:
```
def Val_Jugadores():
    n = input("Cantidad de jugadores: ")
    while (n.isnumeric() == False or int(n) <= 1):
        print("La cantidad ingresada es incorrecta. Recuerda que se permiten mas de dos jugadores.")
        n = input("Cantidad de jugadores: ")
    print("Son ",n," jugadores!")
    return int(n)
```

- Cantidad de Partidas:

In [ ]:
```
def Val_Partidas():
    n = input("Cantidad de partidas: ")
    while (n.isnumeric() == False or int(n) <= 0):
        print("La cantidad ingresada es incorrecta.")
        n = input("Cantidad de partidas: ")
    print("Son ",n," partidas!")
    return int(n)
```

- Modalidad de Juego:

1) El lobo derrumba una pared aleatoria. 2) El lobo derrumba todas las paredes.

In [ ]:
```
def Val_Modalidad():
    print("Modalidades de Juego: \n \t 1) El lobo derrumba una pared. \n \t 2) El lobo derrumba todas las paredes.")
    n = input( "Ingrese la Modalidad de Juego: ")
    while (n.isnumeric() == False or int(n) not in(1, 2)):
        print("El valor ingresado es incorrecto.")
        print("Modalidades de Juego: \n \t 1) El lobo derrumba una pared. \n \t 2) El lobo derrumba todas las paredes.")
        n = input("Ingrese la Modalidad de Juego: ")
    if(int(n) == 1):
        print("El lobo derrumba una pared.")
    else:
        print("El lobo derrumba todas las paredes.")
    return int(n)
```

Simulacion de Turnos:

Los turnos inician con la tirada de dado. Para simular el dado se crea una variable con las siguientes condiciones:

- 1: Celeste (frente)
- 2: Violeta (fondo)
- 3: Verde (pared 1)
- 4: Amarillo (pared 2)
- 5: Rojo (techo)
- 6: Negro (lobo)

In [ ]:
```
def Turno(casa, modalidad):
    dado = randint(1,6)
    if(dado in (1,2,3,4)):
        if(dado not in casa):
            casa.append(dado)
    else:
        if(dado == 5):
            if(len(casa) == 4):
                casa.append(dado)
        else:
            if(len(casa) != 0):
                if(modalidad == 1):
                    casa.remove(casa[randint(0, len(casa) - 1)])
                else:
                    casa = []
    return casa
```

Simulacion de Partida:

In [ ]:
```
def Partidas(partidas, jugadores, modalidad):
    rondas = []
    tiempos = []
    i = 0
    for i in range(partidas):
        t = 0
        inicio = time.time()
        for j in range(jugadores):
            rmin = sys.maxsize # Numero muy grande
            r = 0
            casa = []
            while len(casa) < 5:
                casa = Turno(casa, modalidad)
                r = r + 1
            if(r <= rmin):
                rmin = r
        fin = time.time()
        t = fin - inicio
        tiempos.append(round(t * 1000000,2)) # Microsegundos
        rondas.append(rmin)
    return rondas,tiempos
```

Prueba:

In [ ]:
```
jugadores = Val_Jugadores()
partidas = Val_Partidas()
modalidad = Val_Modalidad()
rondas,tiempos = Partidas(partidas, jugadores, modalidad)
print("rondas: ",rondas)
print("tiempos: ", tiempos)
```

```
Son  3  jugadores!
Son  10  partidas!
Modalidades de Juego:
         1) El lobo derrumba una pared.
         2) El lobo derrumba todas las paredes.
El lobo derrumba una pared.
rondas:  [29, 51, 45, 24, 42, 35, 13, 10, 27, 48]
tiempos:  [257.97, 195.26, 307.08, 105.86, 280.86, 205.99, 170.95, 252.01, 124.22, 183.11]
```

Montecarlo

Escenarios:

- Escenario 1: simulación del juego con dos jugadores donde el lobo tira una pared.
- Escenario 2: simulación del juego con tres jugadores donde el lobo tira una pared.
- Escenario 3: simulación del juego con tres jugadores donde el lobo tira todas las paredes.

In [ ]:
```
random.seed(1) # Set seed para que no varien los valores si se vuelve a ejecutar.
```

Calculo la muestra:

In [ ]:
```
# Inputs de Partidas(partidas, jugadores, modalidad)
N = 100
X1,Y1 = Partidas(N, 2, 1)
X2,Y2 = Partidas(N, 3, 1)
X3,Y3 = Partidas(N, 3, 2)
```

In [ ]:
```
df = pd.DataFrame(list(zip(X1,Y1,X2,Y2,X3,Y3)), columns = ['X1','Y1','X2','Y2','X3','Y3'])
```

Tablero de Datos:

In [ ]:
```
pd.set_option('max_columns', 100)
df[["X1","X2","X3"]].T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
X1	82	10	66	20	12	33	26	11	45	11	7	47	29	18	28	31	36	29	36	24	112	63	43	39	31	24	12	15	6	22	31	11	35
X2	11	21	17	31	17	13	15	12	27	34	43	10	53	12	19	50	24	35	28	10	37	22	28	33	45	56	44	102	10	14	12	35	31
X3	45	25	77	41	17	17	24	36	120	213	91	34	52	28	22	85	12	80	87	42	16	14	20	7	68	18	85	60	22	167	29	59	108

Estadisticos Basicos:

In [ ]:
```
df[["X1","X2","X3"]].describe()
```

Out[ ]:

	X1	X2	X3
count	100.00000	100.000000	100.000000

	X1	X2	X3
mean	32.60000	30.960000	55.100000
std	22.22202	26.583932	43.990472
min	5.00000	6.000000	7.000000
25%	17.75000	14.000000	22.000000
50%	26.00000	24.000000	44.500000
75%	43.25000	37.750000	77.750000
max	112.00000	183.000000	234.000000

In [ ]:
```
mean = [df["X1"].mean(),df["X2"].mean(),df["X3"].mean()]
std = [df["X1"].std(),df["X2"].std(),df["X3"].std()]
print("means: ",mean)
print("std: ", std)
```

```
means:  [32.6, 30.96, 55.1]
std:  [22.22020020110192, 26.583932262655463, 43.99047187927878]
```

HW:

In [ ]:
```
HW1 = stats.norm.ppf(1-0.05/2) * df["X1"].std() / math.sqrt(N)
HW2 = stats.norm.ppf(1-0.05/2) * df["X2"].std() / math.sqrt(N)
HW3 = stats.norm.ppf(1-0.05/2) * df["X3"].std() / math.sqrt(N)
i = ["X1","X2","X3"]
N0 = [N, N, N]
HW = [HW1, HW2, HW3]
# Uso funcion ceil para que redondee para arriba.
N1 = math.ceil(math.pow(df["X1"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N2 = math.ceil(math.pow(df["X2"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N3 = math.ceil(math.pow(df["X3"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N1= [N1, N2, N3]
pd.DataFrame(list(zip(i, N0, HW, N1)), columns = ["X","N0", "HW", "N"])
```

Out[ ]:

	X	N0	HW	N
0	X1	100	4.355436	76
1	X2	100	5.210355	109
2	X3	100	8.621974	298

Volvemos a calcular con la nueva muestra:

In [ ]:
```
# Inputs de Partidas(partidas, jugadores, modalidad)
N = 100 + 198
X1,Y1 = Partidas(N, 2, 1)
X2,Y2 = Partidas(N, 3, 1)
X3,Y3 = Partidas(N, 3, 2)
```

In [ ]:
```
df = pd.DataFrame(list(zip(X1,Y1,X2,Y2,X3,Y3)), columns = ['X1','Y1','X2','Y2','X3','Y3'])
```

Tablero de Datos:

In [ ]:
```
pd.set_option('max_columns', 100)
df[["X1","X2","X3"]].T
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X1	29	33	31	30	14	66	9	13	22	72	10	57	20	22	41	18	10	18	98	32	37	22	32	68	13	11	48	36	29	33	28	22
X2	50	39	13	99	12	8	30	24	30	46	43	14	23	21	36	25	6	34	21	20	43	16	23	65	80	27	22	27	51	24	68	15
X3	20	24	10	31	98	13	8	14	121	28	12	32	92	180	14	37	7	64	190	39	24	13	104	107	134	145	24	146	29	37	57	115

3 rows × 298 columns

Estadisticos Basicos:

In [ ]:
```
df[["X1","X2","X3"]].describe()
```

	X1	X2	X3
count	298.000000	298.000000	298.000000
mean	36.520134	34.436242	55.325503
std	27.316229	25.273140	46.054112
min	6.000000	5.000000	6.000000
25%	18.000000	16.250000	21.000000
50%	30.000000	27.000000	42.000000
75%	46.000000	44.750000	78.000000
max	196.000000	205.000000	265.000000

In [ ]:
```
mean = [df["X1"].mean(),df["X2"].mean(),df["X3"].mean()]
std = [df["X1"].std(),df["X2"].std(),df["X3"].std()]
print("means: ",mean)
print("std: ", std)
```

```
means:  [36.52013422818792, 34.43624161073826, 55.3255033557047]
std:  [27.316228892815403, 25.273140117843987, 46.05411200870951]
```

HW:

In [ ]:
```
HW1 = stats.norm.ppf(1-0.05/2) * df["X1"].std() / math.sqrt(N)
HW2 = stats.norm.ppf(1-0.05/2) * df["X2"].std() / math.sqrt(N)
HW3 = stats.norm.ppf(1-0.05/2) * df["X3"].std() / math.sqrt(N)
i = ["X1","X2","X3"]
N0 = [N, N, N]
HW = [HW1, HW2, HW3]
# Uso funcion ceil para que redondee para arriba.
N1 = math.ceil(math.pow(df["X1"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N2 = math.ceil(math.pow(df["X2"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N3 = math.ceil(math.pow(df["X3"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N1= [N1, N2, N3]
pd.DataFrame(list(zip(i, N0, HW, N1)), columns = ["X","N0", "HW", "N"])
```

	X	N0	HW	N
0	X1	298	3.101421	115
1	X2	298	2.869453	99
2	X3	298	5.228876	326

Volvemos a calcular con la nueva muestra:

In [ ]:
```
# Inputs de Partidas(partidas, jugadores, modalidad)
N = 100 + 198 + 28
X1,Y1 = Partidas(N, 2, 1)
X2,Y2 = Partidas(N, 3, 1)
X3,Y3 = Partidas(N, 3, 2)
```

In [ ]:
```
df = pd.DataFrame(list(zip(X1,Y1,X2,Y2,X3,Y3)), columns = ['X1','Y1','X2','Y2','X3','Y3'])
```

Tablero de Datos:

In [ ]:
```
pd.set_option('max_columns', 100)
df[["X1","X2","X3"]].T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
X1	134	69	59	18	20	34	7	47	15	28	9	28	27	59	124	76	20	66	10	15	36	60	59	33	67	27	41	35	14	51	19	69	32
X2	33	160	15	20	23	9	17	20	15	46	44	25	22	23	12	32	8	49	56	68	50	11	18	36	20	23	18	37	35	54	28	52	19
X3	109	22	54	112	9	18	27	41	17	39	51	13	62	42	18	9	23	80	14	117	27	12	42	38	19	77	20	313	35	42	21	97	91

3 rows × 326 columns

Estadisticos Basicos:

In [ ]:
```
df[["X1","X2","X3"]].describe()
```

Out[ ]:

	X1	X2	X3
count	326.000000	326.000000	326.000000
mean	35.950920	32.972393	55.714724
std	24.891841	23.920172	46.886129
min	5.000000	5.000000	6.000000
25%	19.000000	15.000000	22.000000
50%	30.000000	25.000000	42.000000
75%	47.000000	44.000000	74.000000
max	156.000000	160.000000	313.000000

In [ ]:
```
mean = [df["X1"].mean(),df["X2"].mean(),df["X3"].mean()]
std = [df["X1"].std(),df["X2"].std(),df["X3"].std()]
print("means: ",mean)
print("std: ", std)
```

```
means:  [35.95092024539877, 32.97239263803681, 55.714723926380366]
std:  [24.891840783921165, 23.920171824295547, 46.88612946688056]
```

HW:

In [ ]:
```
```

```
HW1 = stats.norm.ppf(1-0.05/2) * df["X1"].std() / math.sqrt(N)
HW2 = stats.norm.ppf(1-0.05/2) * df["X2"].std() / math.sqrt(N)
HW3 = stats.norm.ppf(1-0.05/2) * df["X3"].std() / math.sqrt(N)
i = ["X1","X2","X3"]
N0 = [N, N, N]
HW = [HW1, HW2, HW3]
# Uso funcion ceil para que redondee para arriba.
N1 = math.ceil(math.pow(df["X1"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N2 = math.ceil(math.pow(df["X2"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N3 = math.ceil(math.pow(df["X3"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N1= [N1, N2, N3]
pd.DataFrame(list(zip(i, N0, HW, N1)), columns = ["X","N0", "HW", "N"])
```

Out[ ]:

	X	N0	HW	N
0	X1	326	2.702068	96
1	X2	326	2.596591	88
2	X3	326	5.089600	338

Volvemos a calcular con la nueva muestra:

In [ ]:
```
# Inputs de Partidas(partidas, jugadores, modalidad)
N = 100 + 198 + 28 + 12
X1,Y1 = Partidas(N, 2, 1)
X2,Y2 = Partidas(N, 3, 1)
X3,Y3 = Partidas(N, 3, 2)
```

In [ ]:
```
df = pd.DataFrame(list(zip(X1,Y1,X2,Y2,X3,Y3)), columns = ['X1','Y1','X2','Y2','X3','Y3'])
```

Tablero de Datos:

In [ ]:
```
pd.set_option('max_columns', 100)
df[["X1","X2","X3"]].T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
X1	82	10	66	20	12	33	26	11	45	11	7	47	29	18	28	31	36	29	36	24	112	63	43	39	31	24	12	15	6	22	31	11	35
X2	11	26	18	27	13	31	42	95	61	74	28	32	22	12	31	45	7	67	25	39	20	45	70	23	17	12	75	25	32	64	37	23	23
X3	97	22	120	25	23	10	54	24	7	48	64	54	36	19	37	34	27	15	23	35	30	13	19	33	9	10	28	71	54	10	48	35	133

3 rows × 338 columns

Estadisticos Basicos:

In [ ]:
```
df[["X1","X2","X3"]].describe()
```

Out[ ]:

	X1	X2	X3
count	338.000000	338.000000	338.000000
mean	34.073964	35.775148	50.532544
std	25.139796	24.348417	43.072851
min	5.000000	5.000000	5.000000
25%	16.000000	17.000000	20.000000
50%	27.000000	30.000000	37.000000
75%	43.000000	48.000000	65.750000
max	152.000000	173.000000	269.000000

In [ ]:
```
mean = [df["X1"].mean(),df["X2"].mean(),df["X3"].mean()]
std = [df["X1"].std(),df["X2"].std(),df["X3"].std()]
print("means: ",mean)
print("std: ", std)
```

```
means:  [33.671597633136095, 34.10650887573964, 50.17455621301775]
std:  [25.93935791099414, 26.03753087158313, 43.4914194335219]
```

HW:

In [ ]:
```
HW1 = stats.norm.ppf(1-0.05/2) * df["X1"].std() / math.sqrt(N)
HW2 = stats.norm.ppf(1-0.05/2) * df["X2"].std() / math.sqrt(N)
HW3 = stats.norm.ppf(1-0.05/2) * df["X3"].std() / math.sqrt(N)
i = ["X1","X2","X3"]
N0 = [N, N, N]
HW = [HW1, HW2, HW3]
# Uso funcion ceil para que redondee para arriba.
N1 = math.ceil(math.pow(df["X1"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N2 = math.ceil(math.pow(df["X2"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N3 = math.ceil(math.pow(df["X3"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N1= [N1, N2, N3]
pd.DataFrame(list(zip(i, N0, HW, N1)), columns = ["X","N0", "HW", "N"])
```

Out[ ]:

	X	N0	HW	N
0	X1	338	2.680103	98
1	X2	338	2.595736	92
2	X3	338	4.591910	286

In [ ]:

```
M = [df["X1"].mean(), df["X2"].mean(),df["X3"].mean()]
STD = [df["X1"].std(), df["X2"].std(),df["X3"].std()]
LI1 = round(df["X1"].mean() - HW1,2)
LI2 = round(df["X2"].mean() - HW2,2)
LI3 = round(df["X3"].mean() - HW3,2)
LS1 = round(df["X1"].mean() + HW1,2)
LS2 = round(df["X2"].mean() + HW2,2)
LS3 = round(df["X3"].mean() + HW3,2)
IC1 = "[" + str(math.ceil(LI1)) + " ; " + str(math.ceil(LS1)) + "]"
IC2 = "[" + str(math.ceil(LI2)) + " ; " + str(math.ceil(LS2)) + "]"
IC3 = "[" + str(math.ceil(LI3)) + " ; " + str(math.ceil(LS3)) + "]"
IC = [IC1, IC2, IC3]
pd.DataFrame(list(zip(i, N0, M, STD, HW, IC)), columns = ["X","N", "mean", "std", "HW", "IC"])
```

Out[ ]:

	X	N	mean	std	HW	IC
0	X1	338	34.073964	25.139796	2.680103	[32 ; 37]
1	X2	338	35.775148	24.348417	2.595736	[34 ; 39]
2	X3	338	50.532544	43.072851	4.591910	[46 ; 56]

Comparacion de Medias:

In [ ]:

```
pg.ttest(x=df["X1"], y=df["X2"], alternative='greater', correction=False)
```

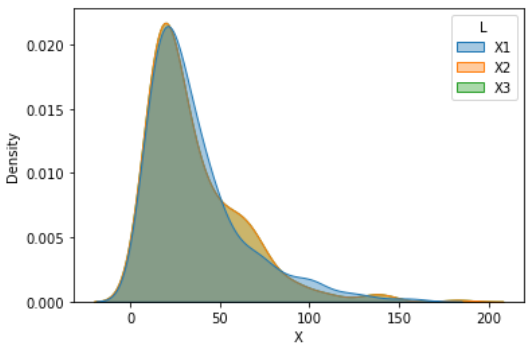
Out[ ]:

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	-0.893649	674	greater	0.814086	[-4.84, inf]	0.068742	0.254	0.005581

Densidades:

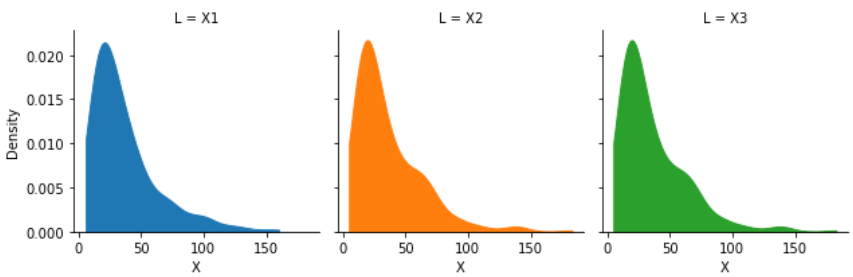
In [ ]:

```
X = X1 + X2 + X3
L=[]
for i in range(3):
    for j in range(N):
        L.append("X" + str(i+1))
df_ = pd.DataFrame(list(zip(L,X)), columns ={"X","L"})
sns.kdeplot(data=df_, x="X", hue="L", fill=True, common_norm=False, alpha=0.4)
plt.show()
```



In [ ]:

```
g = sns.FacetGrid(df_, col='L', hue='L', col_wrap=3)
g = g.map(sns.kdeplot,"X", cut=0, fill=True, common_norm=False, alpha=1, legend=False)
plt.show()
```



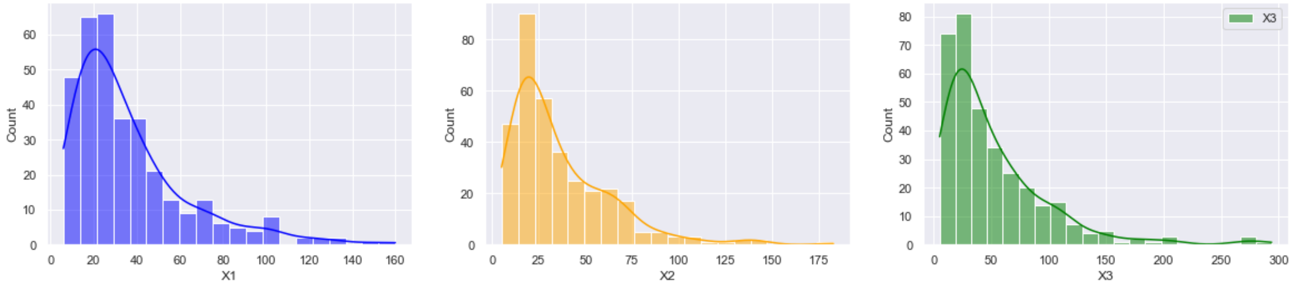
In [ ]:

```
ig, axs = plt.subplots(1, 3, figsize=(20, 4))

sns.histplot(data=df, x="X1", color="blue", label="X1", kde=True, ax=axs[0])
sns.histplot(data=df, x="X2", color="orange", label="X2", kde=True, ax=axs[1])
```

```
sns.histplot(data=df, x="X3", color="green", label="X3", kde=True, ax=axs[2])

plt.legend()
plt.show()
```



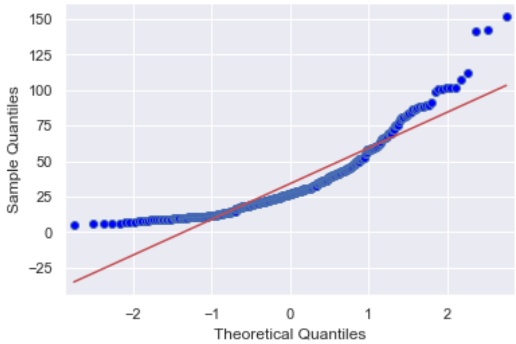
Estudio de Distribucion:

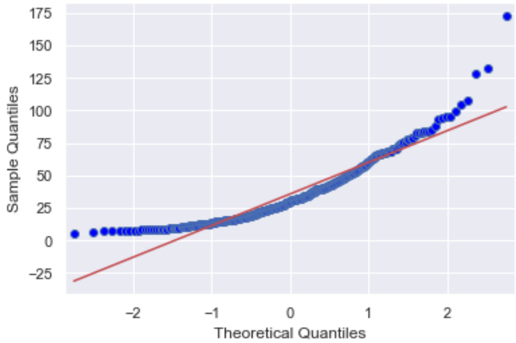
Estudio de Normalidad:

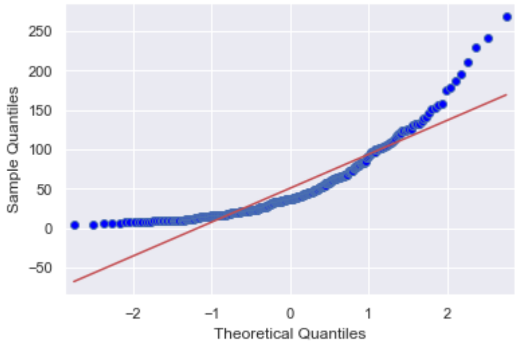
In [ ]:

```
qqplot(df["X1"] , line='s')
qqplot(df["X2"] , line='s')
qqplot(df["X3"] , line='s')
plt.show()
```

```
/Users/anoguera/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/gofplots.py:993: UserWarning: marker is redu
ndantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take pr
ecedence.
  ax.plot(x, y, fmt, **plot_style)
/Users/anoguera/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/gofplots.py:993: UserWarning: marker is redu
ndantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take pr
ecedence.
  ax.plot(x, y, fmt, **plot_style)
/Users/anoguera/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/gofplots.py:993: UserWarning: marker is redu
ndantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take pr
ecedence.
  ax.plot(x, y, fmt, **plot_style)
```







In [ ]:

```
print("Para el Escenario 1:")
stat, p = shapiro(df["X1"])
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
```

```
alpha = 0.05
if p > alpha:
   print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = shapiro(df["X2"])
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = shapiro(df["X3"])
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.845, p=0.000
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.883, p=0.000
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.827, p=0.000
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
```

Estudio de Lognormal:

In [ ]:

```
print("Para el Escenario 1:")
stat, p = kstest(df["X1"],"lognorm", lognorm.fit(df["X1"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Lognormal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Lognormal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = kstest(df["X2"],"lognorm", lognorm.fit(df["X2"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Lognormal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Lognormal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = kstest(df["X3"],"lognorm", lognorm.fit(df["X3"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Lognormal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Lognormal (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.566, p=0.000
La muestra no parece Lognormal (se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.045, p=0.475
La muestra parece Lognormal (no se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.507, p=0.000
La muestra no parece Lognormal (se rechaza la hipótesis nula H0)
```

Estudio de Gamma:

In [ ]:

```
print("Para el Escenario 1:")
stat, p = kstest(df["X1"],"gamma", gamma.fit(df["X1"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gamma (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gamma (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = kstest(df["X2"],"gamma", gamma.fit(df["X2"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gamma (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gamma (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = kstest(df["X3"],"gamma", gamma.fit(df["X3"]))
```

```
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gamma (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gamma (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.043, p=0.534
La muestra parece Gamma (no se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.037, p=0.734
La muestra parece Gamma (no se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.033, p=0.854
La muestra parece Gamma (no se rechaza la hipótesis nula H0)
```

Estudio de Chi Squared:

In [ ]:
```
print("Para el Escenario 1:")
stat, p = kstest(df["X1"],"chi2", chi2.fit(df["X1"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = kstest(df["X2"],"chi2", chi2.fit(df["X2"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = kstest(df["X3"],"chi2", chi2.fit(df["X3"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.043, p=0.534
La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.806, p=0.000
La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.033, p=0.854
La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)
```

Muchos Jugadores:

In [ ]:
```
# Inputs de Partidas(partidas, jugadores, modalidad)
N = 100 + 198 + 28 + 12
X4,Y4 = Partidas(N, 1000, 1)
```

In [ ]:
```
df = pd.DataFrame(list(zip(X1,Y1,X2,Y2,X3,Y3, X4, Y4)), columns = ['X1','Y1','X2','Y2','X3','Y3', 'X4', "Y4"])
```

Tablero de Datos:

In [ ]:
```
pd.set_option('max_columns', 1000)
df[["X4"]].T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
X4	10	41	73	20	10	62	59	46	35	83	44	35	19	87	25	12	45	26	12	27	54	57	10	60	31	16	37	21	18	30	34	74	56	22

Estadisticos Basicos:

In [ ]:
```
df[["X1", "X2", "X3", "X4"]].describe()
```

Out[ ]:

	X1	X2	X3	X4
count	338.000000	338.000000	338.000000	338.000000
mean	34.073964	35.775148	50.532544	32.594675
std	25.139796	24.348417	43.072851	21.550539
min	5.000000	5.000000	5.000000	6.000000
25%	16.000000	17.000000	20.000000	18.000000
50%	27.000000	30.000000	37.000000	26.000000

	X1	X2	X3	X4
75%	43.000000	48.000000	65.750000	44.000000
max	152.000000	173.000000	269.000000	135.000000

HW:

In [ ]:
```
HW1 = stats.norm.ppf(1-0.05/2) * df["X1"].std() / math.sqrt(N)
HW2 = stats.norm.ppf(1-0.05/2) * df["X2"].std() / math.sqrt(N)
HW3 = stats.norm.ppf(1-0.05/2) * df["X3"].std() / math.sqrt(N)
HW4 = stats.norm.ppf(1-0.05/2) * df["X4"].std() / math.sqrt(N)
i = ["X1","X2","X3", "X4"]
N0 = [N, N, N, N]
HW = [HW1, HW2, HW3, HW4]
# Uso funcion ceil para que redondee para arriba.
N1 = math.ceil(math.pow(df["X1"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N2 = math.ceil(math.pow(df["X2"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N3 = math.ceil(math.pow(df["X3"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N4 = math.ceil(math.pow(df["X4"].std() * stats.norm.ppf(1-0.05/2) / 5 , 2))
N1= [N1, N2, N3, N4]
pd.DataFrame(list(zip(i, N0, HW, N1)), columns = ["X","N0", "HW", "N"])
```

Out[ ]:

	X	N0	HW	N
0	X1	338	2.680103	98
1	X2	338	2.595736	92
2	X3	338	4.591910	286
3	X4	338	2.297460	72

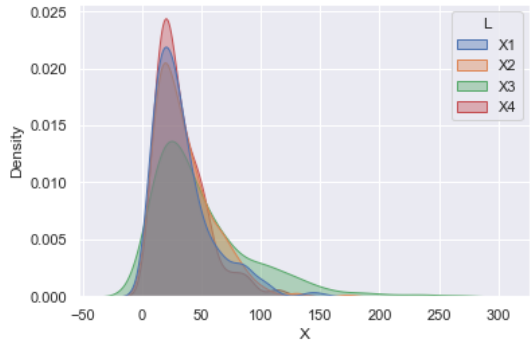
In [ ]:
```
M = [df["X1"].mean(), df["X2"].mean(),df["X3"].mean(),df["X4"].mean()]
STD = [df["X1"].std(), df["X2"].std(),df["X3"].std(),df["X4"].std()]
LI1 = round(df["X1"].mean() - HW1,2)
LI2 = round(df["X2"].mean() - HW2,2)
LI3 = round(df["X3"].mean() - HW3,2)
LI4 = round(df["X4"].mean() - HW4,2)
LS1 = round(df["X1"].mean() + HW1,2)
LS2 = round(df["X2"].mean() + HW2,2)
LS3 = round(df["X3"].mean() + HW3,2)
LS4 = round(df["X4"].mean() + HW4,2)
IC1 = "[" + str(math.ceil(LI1)) + " ; " + str(math.ceil(LS1)) + "]"
IC2 = "[" + str(math.ceil(LI2)) + " ; " + str(math.ceil(LS2)) + "]"
IC3 = "[" + str(math.ceil(LI3)) + " ; " + str(math.ceil(LS3)) + "]"
IC4 = "[" + str(math.ceil(LI4)) + " ; " + str(math.ceil(LS4)) + "]"
IC = [IC1, IC2, IC3, IC4]
pd.DataFrame(list(zip(i, N0, M, STD, HW, IC)), columns = ["X","N", "mean", "std", "HW", "IC"])
```

Out[ ]:

	X	N	mean	std	HW	IC
0	X1	338	34.073964	25.139796	2.680103	[32 ; 37]
1	X2	338	35.775148	24.348417	2.595736	[34 ; 39]
2	X3	338	50.532544	43.072851	4.591910	[46 ; 56]
3	X4	338	32.594675	21.550539	2.297460	[31 ; 35]

Densidades:

In [ ]:
```
X = X1 + X2 + X3 + X4
L=[]
for i in range(4):
    for j in range(N):
        L.append("X" + str(i+1))
df_ = pd.DataFrame(list(zip(L,X)), columns ={"X","L"})
sns.kdeplot(data=df_, x="X", hue="L", fill=True, common_norm=False, alpha=0.4)
plt.show()
```

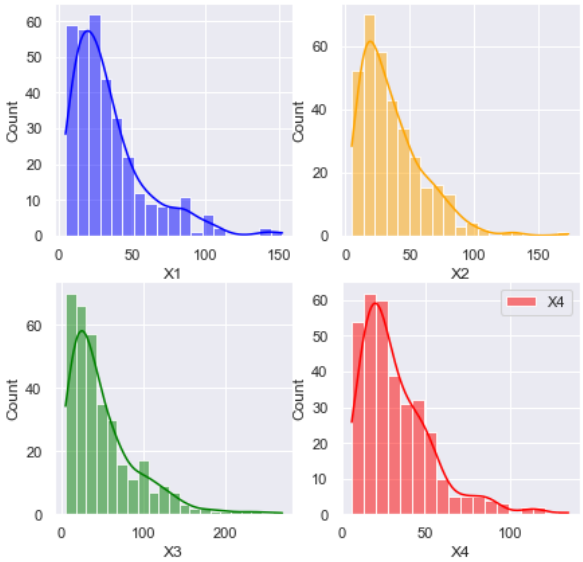


In [ ]:
```
ig, axs = plt.subplots(2, 2, figsize=(7, 7))

sns.histplot(data=df, x="X1", color="blue", label="X1", kde=True, ax=axs[0][0])
```

```
sns.histplot(data=df, x="X2", color="orange", label="X2", kde=True, ax=axs[0][1])
sns.histplot(data=df, x="X3", color="green", label="X3", kde=True, ax=axs[1][0])
sns.histplot(data=df, x="X4", color="red", label="X4", kde=True, ax=axs[1][1])

plt.legend()
plt.show()
```



Distribuciones:

Estudio de Normal:

In [ ]:
```
print("Para el Escenario 4:")
stat, p = shapiro(df["X4"])
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 4:
Estadisticos=0.869, p=0.000
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
```

Estudio de Lognormal:

In [ ]:
```
print("Para el Escenario 4:")
stat, p = kstest(df["X4"],"lognorm", lognorm.fit(df["X4"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Lognormal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Lognormal (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 4:
Estadisticos=0.579, p=0.000
La muestra no parece Lognormal (se rechaza la hipótesis nula H0)
```

Estudio de Gamma:

In [ ]:
```
print("Para el Escenario 4:")
stat, p = kstest(df["X4"],"gamma", gamma.fit(df["X4"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gamma (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gamma (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 4:
Estadisticos=0.037, p=0.737
La muestra parece Gamma (no se rechaza la hipótesis nula H0)
```

Estudio de Chi Squared:

In [ ]:
```
print("Para el Escenario 4:")
stat, p = kstest(df["X4"],"chi2", chi2.fit(df["X4"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 4:
Estadisticos=0.660, p=0.000
La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)
```

Tiempos de Ejecucion:

Tablero de Datos:

In [ ]:
```
pd.set_option('max_columns', 1000)
df[["Y1","Y2","Y3"]].T
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Y1	143.77	30.28	117.30	41.96	67.95	62.94	36.95	32.90	143.29	58.89	38.15	112.77	118.02	51.02	62.94	81.78	70.10	125.17	96.08
Y2	71.05	69.14	35.05	129.94	67.95	51.98	97.99	185.01	172.14	130.89	139.95	112.30	48.16	66.04	44.82	114.20	139.71	92.03	99.90
Y3	190.97	140.19	304.22	44.82	71.29	123.74	153.30	64.85	172.85	59.13	65.09	158.07	110.86	158.07	82.02	99.18	110.86	56.03	256.06

Estadisticos Basicos:

In [ ]:
```
df[["Y1", "Y2", "Y3"]].describe()
```

Out[ ]:

	Y1	Y2	Y3
count	338.000000	338.000000	338.000000
mean	75.351568	105.475118	126.107811
std	49.364146	50.341527	65.079222
min	13.110000	28.850000	25.030000
25%	43.157500	68.960000	78.320000
50%	66.040000	95.960000	115.040000
75%	94.712500	130.180000	157.832500
max	591.990000	333.790000	378.850000

HW:

In [ ]:
```
HW1 = stats.norm.ppf(1-0.05/2) * df["Y1"].std() / math.sqrt(N)
HW2 = stats.norm.ppf(1-0.05/2) * df["Y2"].std() / math.sqrt(N)
HW3 = stats.norm.ppf(1-0.05/2) * df["Y3"].std() / math.sqrt(N)
i = ["Y1","Y2","Y3"]
N0 = [N, N, N]
HW = [HW1, HW2, HW3]
# Uso funcion ceil para que redondee para arriba.
N1 = math.ceil(math.pow(df["Y1"].std() * stats.norm.ppf(1-0.05/2) / 10 , 2))
N2 = math.ceil(math.pow(df["Y2"].std() * stats.norm.ppf(1-0.05/2) / 10 , 2))
N3 = math.ceil(math.pow(df["Y3"].std() * stats.norm.ppf(1-0.05/2) / 10 , 2))
N1= [N1, N2, N3]
pd.DataFrame(list(zip(i, N0, HW, N1)), columns = ["X","N0", "HW", "N"])
```

Out[ ]:

	X	N0	HW	N
0	Y1	338	5.262612	94
1	Y2	338	5.366809	98
2	Y3	338	6.937965	163

HW:

In [ ]:
```
M = [df["Y1"].mean(), df["Y2"].mean(),df["Y3"].mean()]
STD = [df["Y1"].std(), df["Y2"].std(),df["Y3"].std()]
LI1 = round(df["Y1"].mean() - HW1,2)
LI2 = round(df["Y2"].mean() - HW2,2)
LI3 = round(df["Y3"].mean() - HW3,2)
LS1 = round(df["Y1"].mean() + HW1,2)
LS2 = round(df["Y2"].mean() + HW2,2)
LS3 = round(df["Y3"].mean() + HW3,2)
IC1 = "[" + str(LI1) + " ; " + str(LS1) + "]"
IC2 = "[" + str(LI2) + " ; " + str(LS2) + "]"
IC3 = "[" + str(LI3) + " ; " + str(LS3) + "]"
IC = [IC1, IC2, IC3]
pd.DataFrame(list(zip(i, N0, M, STD, HW, IC)), columns = ["X","N", "mean", "std", "HW", "IC"])
```

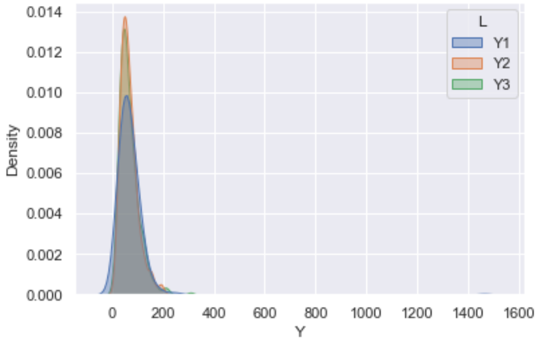
Out[ ]:

	X	N	mean	std	HW	IC
0	Y1	338	75.351568	49.364146	5.262612	[70.09 ; 80.61]
1	Y2	338	105.475118	50.341527	5.366809	[100.11 ; 110.84]
2	Y3	338	126.107811	65.079222	6.937965	[119.17 ; 133.05]

Densidades:

In [ ]:
```
Y = Y1 + Y2 + Y3
L=[]
```

```
for i in range(3):
    for j in range(N):
        L.append("Y" + str(i+1))
df_ = pd.DataFrame(list(zip(L,Y)), columns ={"Y","L"})
sns.kdeplot(data=df_, x="Y", hue="L", fill=True, common_norm=False, alpha=0.4)
plt.show()
```

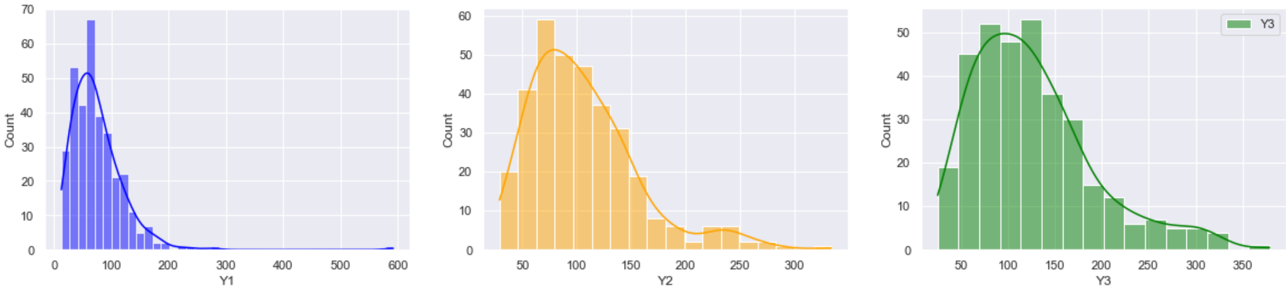


In [ ]:

```
ig, axs = plt.subplots(1, 3, figsize=(20, 4))

sns.histplot(data=df, x="Y1", color="blue", label="Y1", kde=True, ax=axs[0])
sns.histplot(data=df, x="Y2", color="orange", label="Y2", kde=True, ax=axs[1])
sns.histplot(data=df, x="Y3", color="green", label="Y3", kde=True, ax=axs[2])

plt.legend()
plt.show()
```



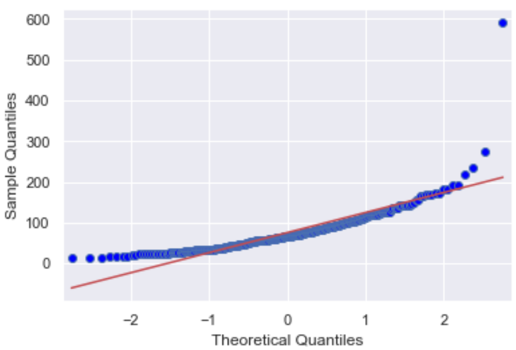
Estudio de Distribucion:

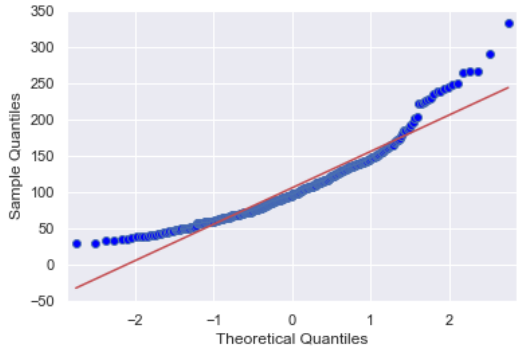
Estudio de Normalidad:

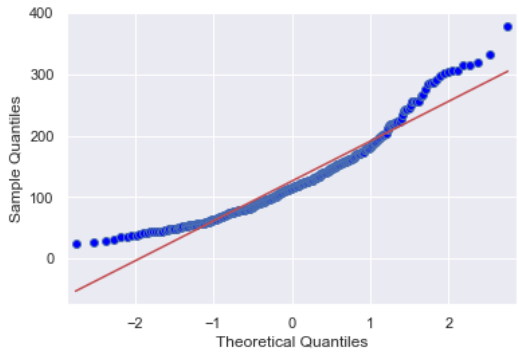
In [ ]:

```
qqplot(df["Y1"] , line='s')
qqplot(df["Y2"] , line='s')
qqplot(df["Y3"] , line='s')
plt.show()
```

```
/Users/anoguera/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/gofplots.py:993: UserWarning: marker is redu
ndantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take pr
ecedence.
  ax.plot(x, y, fmt, **plot_style)
/Users/anoguera/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/gofplots.py:993: UserWarning: marker is redu
ndantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take pr
ecedence.
  ax.plot(x, y, fmt, **plot_style)
/Users/anoguera/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/gofplots.py:993: UserWarning: marker is redu
ndantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take pr
ecedence.
  ax.plot(x, y, fmt, **plot_style)
```







In [ ]:

```
print("Para el Escenario 1:")
stat, p = shapiro(df["Y1"])
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = shapiro(df["Y2"])
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = shapiro(df["Y3"])
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.754, p=0.000
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.909, p=0.000
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.924, p=0.000
La muestra no parece Gaussiana o Normal (se rechaza la hipótesis nula H0)
```

Estudio de Lognormal:

In [ ]:

```
print("Para el Escenario 1:")
stat, p = kstest(df["Y1"],"lognorm", lognorm.fit(df["Y1"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Lognormal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Lognormal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = kstest(df["Y2"],"lognorm", lognorm.fit(df["Y2"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Lognormal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Lognormal (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = kstest(df["Y3"],"lognorm", lognorm.fit(df["Y3"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
```

```
alpha = 0.05
if p > alpha:
   print('La muestra parece Lognormal (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Lognormal (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.689, p=0.000
La muestra no parece Lognormal (se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.028, p=0.947
La muestra parece Lognormal (no se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.029, p=0.931
La muestra parece Lognormal (no se rechaza la hipótesis nula H0)
```

Estudio de Gamma:

In [ ]:

```
print("Para el Escenario 1:")
stat, p = kstest(df["Y1"],"gamma", gamma.fit(df["Y1"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gamma (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gamma (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = kstest(df["Y2"],"gamma", gamma.fit(df["Y2"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gamma (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gamma (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = kstest(df["Y3"],"gamma", gamma.fit(df["Y3"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Gamma (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Gamma (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.050, p=0.359
La muestra parece Gamma (no se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.029, p=0.932
La muestra parece Gamma (no se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.024, p=0.987
La muestra parece Gamma (no se rechaza la hipótesis nula H0)
```

Estudio de Chi Squared:

In [ ]:

```
print("Para el Escenario 1:")
stat, p = kstest(df["Y1"],"chi2", chi2.fit(df["Y1"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)')

print("Para el Escenario 2:")
stat, p = kstest(df["Y2"],"chi2", chi2.fit(df["Y2"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)')

print("Para el Escenario 3:")
stat, p = kstest(df["Y3"],"chi2", chi2.fit(df["Y3"]))
print('Estadisticos=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
   print('La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)')
else:
   print('La muestra no parece Chi Squeared (se rechaza la hipótesis nula H0)')
```

```
Para el Escenario 1:
Estadisticos=0.050, p=0.359
La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)
Para el Escenario 2:
Estadisticos=0.029, p=0.932
La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)
Para el Escenario 3:
Estadisticos=0.024, p=0.987
La muestra parece Chi Squared (no se rechaza la hipótesis nula H0)
```