Estimación de parámetros de un modelo Log-Normal para un activo financiero

Modelización y Valoración de Derivados y Carteras en Finanzas

Máster Universitario en ingeniería Matemática y Computación

UNIR - Junio 2024

Autor: Javier Blanco Álvarez

RESUMEN

El modelo log-normal se utiliza ampliamente en finanzas para describir la distribución de los precios de los activos, donde el logaritmo del precio del activo sigue una distribución normal. (continuar...)

Palabras clave: Modelo Log-Normal, Brent, estimación de parémetros, modelización, activos financieros.

INTRODUCCIÓN

El modelo log-normal se utiliza ampliamente en finanzas para describir la distribución de los precios de los activos, donde el logaritmo del precio del activo sigue una distribución normal. Si $(S(t))$ representa el precio del activo en el tiempo (t), entonces $(\ln(S(t)))$ se distribuye normalmente, definido por (μ) (tendencia) y (σ) (volatilidad). Este modelo es especialmente útil para simular precios futuros de acciones, valoración de opciones, gestión de riesgos y optimización de carteras (Sharpe, M. J. 2004).

Estimar (μ) y (σ) con precisión resulta fundamental dentro del campo de la modelización financiera ya que, en este contexto (μ) indica el rendimiento esperado del activo (o la esperanza matemática $\mathbb{E}$), mientras que (σ) mide la variabilidad del rendimiento (o la raíz cuadrada de su varianza $\mathbb{V}$), reflejando el riesgo. Estos parámetros son esenciales a la hora de evaluar el riesgo de inversión en un portafolio, valorar derivados y gestionar carteras de manera informada (Hoffman, I. 1993).

PAQUETES Y BIBLIOTECAS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

FUNCIONALIDADES

En esta sección implementaremos las funcionalidades necesarias para llevar a cabo los análisis soliictados en la práctica. En concreto diseñaremos una función que nos permita estimar los parámetros, otra para la validación de los datos y otra para el modelado Log-normal.

```
#MODELO LOG-NORMAL
def log_normal_model(t:np.ndarray, mu:float, sigma:float, s0:float) -> np.ndarray:
  # Basado en el ejemplo mostrado en clase
  # IMPUT:
  # t: vector de periodos t del dataframe
  # mu: valor de parámetro mu previamente estimado
  # sigma: valor de parámetro sigma previamente estimado
  # s0: valor del precio del stock en el tiempo t0.

  #OUTPUT:
  #Devuelve un vector con los valores del modelo log-normal para el stock estudiado
    n=len(t)
    # Wiener
    w = np.zeros(n)
    for i in range(1, n):
        w[i] = w[i - 1] + np.sqrt(t[i] - t[i-1])*np.random.normal(0,1)
    return s0*np.exp((mu-sigma**2/2)*t+sigma*w)


#ESTIMADOR
def parameter_estimator(close_price:pd.Series,
                        delta_t:float = 1,
                        method:str = "MMV")->tuple:
  """
    Estima los parámetros de la distribución de precios de cierre de un activo.

    INPUTS:
    ----------
    close_price : pd.Series
        Serie de pandas que contiene los precios de cierre del activo.
    delta_t : float, opcional
        Intervalo de tiempo entre los precios de cierre, por defecto es 1.
    method : str, opcional
        Método de estimación a utilizar. Puede ser "MMV" (Máxima Verosimilitud),
        "MNP" (Método No Paramétrico) o "MME" (Método de Momentos Estadísticos).
        Por defecto es "MMV".

    OUTPUTS:
    -------
    tuple
        Una tupla que contiene los siguientes elementos:
        - mu: float
            Estimación del parámetro mu (media).
        - sigma: float
            Estimación del parámetro sigma (desviación estándar).
        - S: np.ndarray
            Serie de valores transformados según el método elegido.
  """
  n = len(close_price)
  if method == "MNP":
    #método no paramétrico
    S =  (close_price[1:].values - close_price[:-1].values)
    mu=(np.sum(S)/np.sum(close_price[:-1].values))*(1/delta_t)
    sigma=np.sqrt((np.sum(S**2)/np.sum(close_price.iloc[:-1].values**2))*(1/delta_t))
  elif method =='MME':
    #método de momentos estádísticos
    S = np.log10(close_price[1:].values / close_price[:-1].values)
    sigma=np.std(S,ddof=1)*(1/delta_t)
    mu = (1/delta_t)*np.mean(S)+sigma**2/2
  else:
    # metodo de máxima verosimilitud (por defecto)
    S =  (close_price[1:].values / close_price[:-1].values) - 1
    mu = (np.mean(S) / delta_t)
    sigma = np.std(S,ddof=1) * np.sqrt((n-1)/(n*delta_t))
  return mu, sigma, S


#Esperanza matemática modelo LogNormal
def ELN(t:np.ndarray,mu:float,s0:float)->float:
  """
    Calcula la Esperanza Matemática para un modelo Log-Normal.

    INPUTS:
    ------
    - t (np.ndarray): Array de tiempo.
    - mu (float): Parámetro mu de la distribución Log-Normal.
    - s0 (float): Valor inicial.

    OUTPUTS:
    -------
    - float: La Esperanza Matemática en el tiempo dado por la ecuación s0 * exp(mu * t).
  """
  return s0*np.exp(mu*t)

#Varianza modelo LogNormal
def VLN(t:np.ndarray,mu:float,sigma:float,s0:float)->float:
  """
    Calcula la Varianza para un modelo Log-Normal.

    INPUTS:
    ------
    - t (np.ndarray): Array de tiempo.
    - mu (float): Parámetro mu de la distribución Log-Normal.
    - sigma (float): Parámetro sigma de la distribución Log-Normal.
    - s0 (float): Valor inicial.

    OUTPUTS:
    -------
    - float: La Varianza en el tiempo dado por la ecuación s0^2 * exp(2 * mu * t) * (exp(sigma^2 * t) - 1).
  """
  return s0**2*np.exp(2*mu*t)*(np.exp(sigma**2*t)-1)
```

```
#VALIDACION
def error_estimation(close_price:pd.Series,
                     t:np.array,
                     mu:float,
                     sigma:float,
                     error_type:str = "ECM")->float:
    """
    Estima el error de un modelo de ajuste de precios.

    INPUTS:
    -------
    - close_price (pd.Series): Serie de precios de cierre.
    - t (np.ndarray): Array de tiempo.
    - mu (float): Parámetro mu de la distribución Log-Normal.
    - sigma (float): Parámetro sigma de la distribución Log-Normal.
    - error_type (str): Tipo de error a calcular, puede ser "EPAM" o "ECM" (por defecto).

    OUTPUTS:
    -------
    - float: El valor del error estimado.

    Métodos de Cálculo de Error:
    - EPAM: Error Porcentual Absoluto Medio.
    - ECM: Error Cuadrático Medio (por defecto).
    """
    n = len(close_price)
    # sst = log_normal_model(t, mu, sigma, returns[0])
    s = close_price
    Expected_value = ELN(t, mu, close_price[0])

    LogELN = np.log10(Expected_value)
    LogS = np.log10(s)

    if error_type != "ECM":
      abs_LogS = np.abs(LogS[1:])
    # Reemplaza ceros por un pequeño número para evitar división por cero
      abs_LogS[abs_LogS == 0] = 1e-10
      error_value = (100/n) * np.sum(np.abs((LogS[1:] - LogELN[1:])) / abs_LogS)
      print(f"EPAM: {error_value}")
    else:
      # ECM (por defecto)
      error_value = np.sqrt(np.sum((LogS[1:]-LogELN[1:])**2)*(1/n))
      print(f"ECM: {error_value}")
    return error_value
```

```
#cargamos los datos en bruto utilizando pandas y verificamos los primeros 5 registros
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/MVCF/BrentOilPrices.csv', sep=',')
data.head()
```

	Fecha	Apertura	Alto	Bajo	Cierre
0	5/26/2022	114.45	118.00	113.84	117.40
1	5/27/2022	117.54	119.73	116.68	119.43
2	5/31/2022	119.47	124.13	119.36	122.84
3	6/1/2022	116.63	118.58	115.43	116.29
4	6/2/2022	115.82	118.44	112.48	117.61

Próximos pasos: Ver gráficos recomendados

```
#Analizamos los tipos de datos involucrados en el dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 502 entries, 0 to 501
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Fecha     502 non-null    object
 1   Apertura  502 non-null    float64
 2   Alto      502 non-null    float64
 3   Bajo      502 non-null    float64
 4   Cierre    502 non-null    float64
dtypes: float64(4), object(1)
memory usage: 19.7+ KB
```

Observamos que los precios del fichero son reconocidos como datos de coma flotanto, mientras que el campo `Fecha` se reconoce como un objeto. Para evitar problema de comaptibilidad transformaremos ese campo al tipo `date`.

```
# Transformar datatype de Object a Date en la columan Fecha
data['Fecha'] = pd.to_datetime(data['Fecha'], format='%m/%d/%Y')
data.info()
```

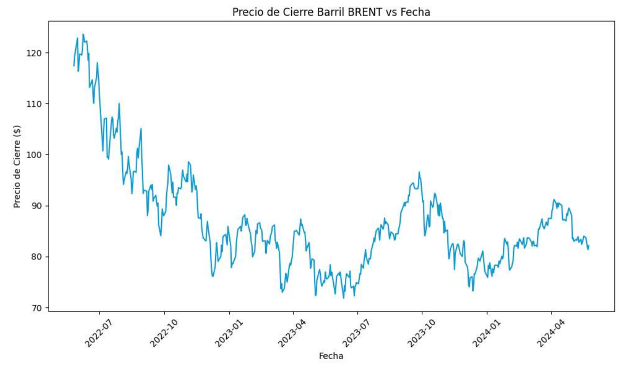
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 502 entries, 0 to 501
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Fecha     502 non-null    datetime64[ns]
 1   Apertura  502 non-null    float64
 2   Alto      502 non-null    float64
 3   Bajo      502 non-null    float64
 4   Cierre    502 non-null    float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 19.7 KB
```

```
#Analizamos los datos usando estadística descriptiva
data.describe()
```

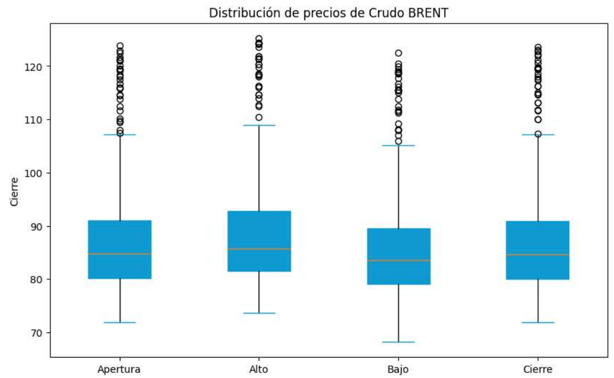
	Fecha	Apertura	Alto	Bajo	Cierre
count	502	502.000000	502.000000	502.000000	502.000000
mean	2023-05-26 23:57:07.888446208	86.983367	88.298426	85.547709	86.945199
min	2022-05-26 00:00:00	71.890000	73.730000	68.200000	71.840000
25%	2022-11-23 12:00:00	80.157500	81.490000	79.055000	80.017500
50%	2023-05-25 12:00:00	84.825000	85.675000	83.505000	84.690000
75%	2023-11-23 12:00:00	90.997500	92.822500	89.600000	90.912500
max	2024-05-24 00:00:00	123.890000	125.180000	122.500000	123.580000
std	NaN	10.155561	10.412188	9.821454	10.183356

Podemos graficar el precio de cierre con respecto al tiempo para observar el comportamiento del barril de crudo en los últimos 2 años

```
plt.figure(figsize=(10, 6))
plt.plot(data['Fecha'], data['Cierre'], linestyle='-', color='#0F9ED5')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.title('Precio de Cierre Barril BRENT vs Fecha')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(10, 6))
box = plt.boxplot(data[['Apertura', 'Alto', 'Bajo', 'Cierre']], patch_artist=True)
for element in ['boxes', 'fliers', 'means', 'caps']:
    plt.setp(box[element], color='#0F9ED5', linewidth=1.0)

plt.xticks([1, 2, 3, 4], ['Apertura', 'Alto', 'Bajo', 'Cierre'])
plt.ylabel('Cierre')
plt.title('Distribución de precios de Crudo BRENT')
plt.show()
```



MÉTODO DE MÁXIMA VEROSIMILITUD (MMV)

El método de la máxima verosimilitud (MMV) se utiliza para estimar los parámetros μ y σ en un modelo log-normal transformando los precios de los activos en logaritmos, que se suponen distribuidos normalmente. La función de verosimilitud, al ser maximizada, permite encontrar las estimaciones de los parámetros antesmencionados que mejor ajustan los datos observados

```
#Estimación de mu y sigma por MMV
close = data['Cierre']
dt = 1
muMMV, sigmaMMV, returnsMMV = parameter_estimator(close_price=close, delta_t=dt, method='MMV')
```

MÉTODO DE LOS MOMENTOS NO PARAMÉTRICOS (MMNP)

En éste método, en lugar de asumir una distribución específica se calculan los momentos empíricos de los datos, como la media y la desviación estándar de los logaritmos de los precios. Estos momentos se igualan a los momentos teóricos de una distribución log-normal para obtener estimaciones de μ y σ. Aunque menos preciso que MMV, es útil cuando se carece de información paramétrica sobre la distribución de los datos.

```
#Estimación de mu y sigma por MNP
close = data['Cierre']
dt = 1
muMNP, sigmaMNP, returnsMNP = parameter_estimator(close_price=close, delta_t=dt, method='MNP')
```

VALIDACIÓN DEL MODELO

```
#Validacón
data_index = data.index
ECMMMV = error_estimation(close_price=close, t=data_index.values, mu=muMMV, sigma=sigmaMMV, error_type="ECM")
EPAMMMV = error_estimation(close_price=close, t=data_index.values, mu=muMMV, sigma=sigmaMMV, error_type="EPAM")
```

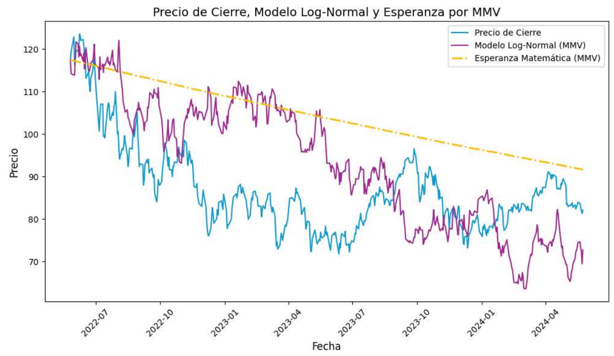
```
ECM: 0.08901688229611569
EPAM: 4.174807032214769
```

```
data_index = data.index
ECMMNP = error_estimation(close_price=close, t=data_index.values, mu=muMNP, sigma=sigmaMNP, error_type="ECM")
EPAMMNP = error_estimation(close_price=close, t=data_index.values, mu=muMNP, sigma=sigmaMNP, error_type="EPAM")
```

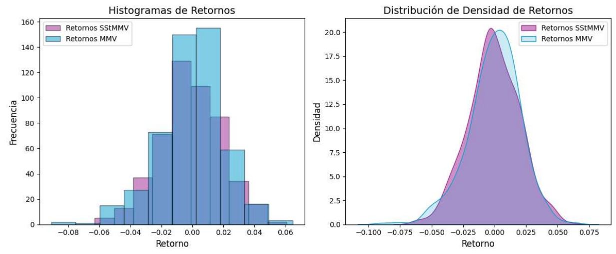
```
ECM: 0.06523950951896751
EPAM: 2.839666312342386
```

Modelo Log-Normal con parámetros Máxmima Verosimilitud

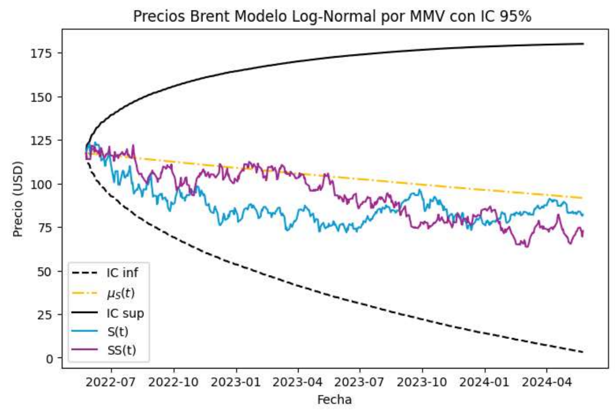
```
np.random.seed(144)
SStMMV = log_normal_model(t=data_index, mu=muMMV, sigma=sigmaMMV, s0=close[0])
ELNMMV = ELN(t=data_index, mu=muMMV, s0=SStMMV[0])


plt.figure(figsize=(12, 6))
plt.plot(data['Fecha'], close, label='Precio de Cierre', color='#0F9ED5', linestyle='-', linewidth=1.5)
plt.plot(data['Fecha'], SStMMV, label='Modelo Log-Normal (MMV)', color='#A02B93', linestyle='-', linewidth=1.5)
plt.plot(data['Fecha'], ELNMMV, label='Esperanza Matemática (MMV)', color='#FFC000', linestyle='-.', linewidth=2)
plt.title('Precio de Cierre, Modelo Log-Normal y Esperanza por MMV', fontsize=14)
plt.xlabel('Fecha', fontsize=12)
plt.ylabel('Precio', fontsize=12)
plt.legend(fontsize=10)
# plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```




```
returnsSStMMV = np.array((SStMMV[1:]/SStMMV[:-1])-1)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
bins = 10
#Histogramas tradicionales
ax1.hist(returnsSStMMV, bins=bins, alpha=0.5, label='Retornos SStMMV', color='#A02B93', edgecolor='black')
ax1.hist(returnsMMV, bins=bins, alpha=0.5, label='Retornos MMV', color='#0F9ED5', edgecolor='black')

#Título y etiquetas de los ejes al primer gráfico
ax1.set_title('Histogramas de Retornos', fontsize=14)
ax1.set_xlabel('Retorno', fontsize=12)
ax1.set_ylabel('Frecuencia', fontsize=12)
ax1.legend(fontsize=10)

#Densidades de distribución con seaborn
sns.kdeplot(returnsSStMMV, color='#A02B93', label='Retornos SStMMV', ax=ax2, fill=True, alpha=0.5)
sns.kdeplot(returnsMMV, color='#0F9ED5', label='Retornos MMV', ax=ax2, fill=True, alpha=0.2)

# Título y etiquetas de los ejes al segundo gráfico
ax2.set_title('Distribución de Densidad de Retornos', fontsize=14)
ax2.set_xlabel('Retorno', fontsize=12)
ax2.set_ylabel('Densidad', fontsize=12)
ax2.legend(fontsize=10)
plt.tight_layout()
plt.show()
```

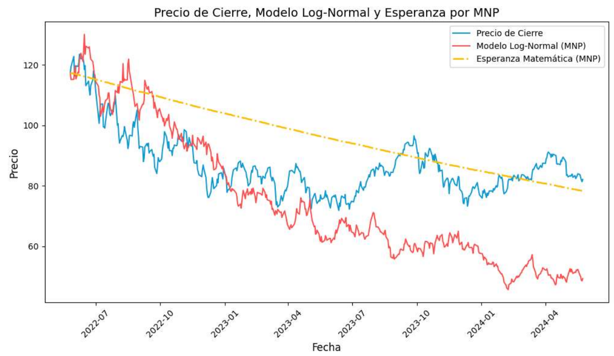



```
#IC al 95%
np.random.seed(144)
#Desviación Estándar
SMMV = np.sqrt(VLN(t=data_index, mu=muMMV, sigma=sigmaMMV, s0=close[0]))


t=data_index
date = data['Fecha']
S = close

plt.figure(figsize=(8, 5))
plt.plot(date, ELNMMV - 1.96 * SMMV, '--k', label='IC inf')
plt.plot(date, ELNMMV, '#FFC000', label=r'$\mu_S(t)$', linestyle='-.')
plt.plot(date, ELNMMV + 1.96 * SMMV, 'k', label='IC sup')
plt.plot(date, S, '#0F9ED5', label='S(t)')
plt.plot(date, SStMMV,'#A02B93', label='SS(t)')
plt.xlabel('Fecha', fontsize=10)
plt.ylabel('Precio (USD)', fontsize=10)
plt.title(r'Precios Brent Modelo Log-Normal por MMV con IC 95% ', fontsize=12)
plt.legend()
plt.show()
```

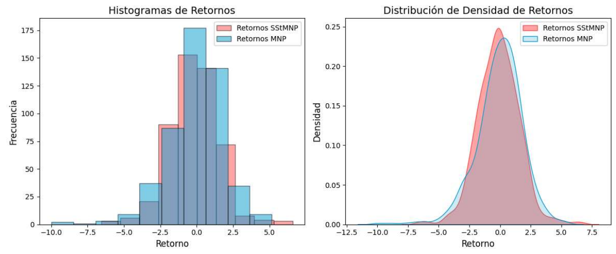


Modelo Log-Normal con parámetros Momento No Paramétricos

```
np.random.seed(145)
SStMNP = log_normal_model(t=data_index, mu=muMNP, sigma=sigmaMNP, s0=close[0])
ELNMNP = ELN(t=data_index, mu=muMNP, s0=SStMNP[0])


plt.figure(figsize=(12, 6))
plt.plot(data['Fecha'], close, label='Precio de Cierre', color='#0F9ED5', linestyle='-', linewidth=1.5)
plt.plot(data['Fecha'], SStMNP, label='Modelo Log-Normal (MNP)', color='#FF5050', linestyle='-', linewidth=1.5)
plt.plot(data['Fecha'], ELNMNP, label='Esperanza Matemática (MNP)', color='#FFC000', linestyle='-.', linewidth=2)
plt.title('Precio de Cierre, Modelo Log-Normal y Esperanza por MNP', fontsize=14)
plt.xlabel('Fecha', fontsize=12)
plt.ylabel('Precio', fontsize=12)
plt.legend(fontsize=10)
# plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```

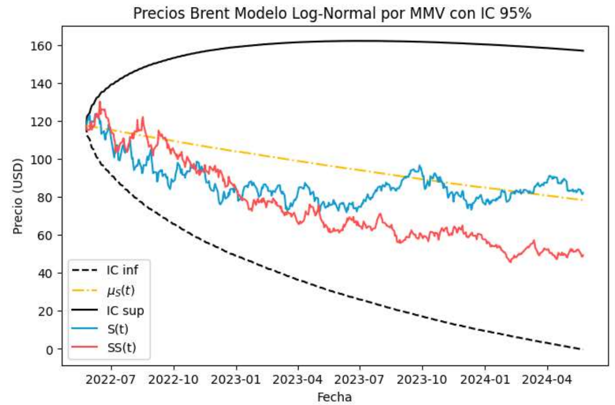



```
returnsSStMNP = np.array((SStMNP[1:]-SStMNP[:-1]))

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
bins = 10
# Primer subgráfico: Histogramas tradicionales
ax1.hist(returnsSStMNP, bins=bins, alpha=0.5, label='Retornos SStMNP', color='#FF5050', edgecolor='black')
ax1.hist(returnsMNP, bins=bins, alpha=0.5, label='Retornos MNP', color='#0F9ED5', edgecolor='black')

# Añadir título y etiquetas de los ejes al primer subgráfico
ax1.set_title('Histogramas de Retornos', fontsize=14)
ax1.set_xlabel('Retorno', fontsize=12)
ax1.set_ylabel('Frecuencia', fontsize=12)
ax1.legend(fontsize=10)

# Segundo subgráfico: Densidades de distribución (sin barras)
sns.kdeplot(returnsSStMNP, color='#FF5050', label='Retornos SStMNP', ax=ax2, fill=True, alpha=0.5)
sns.kdeplot(returnsMNP, color='#0F9ED5', label='Retornos MNP', ax=ax2, fill=True, alpha=0.2)

# Añadir título y etiquetas de los ejes al segundo subgráfico
ax2.set_title('Distribución de Densidad de Retornos', fontsize=14)
ax2.set_xlabel('Retorno', fontsize=12)
ax2.set_ylabel('Densidad', fontsize=12)
ax2.legend(fontsize=10)
plt.tight_layout()
plt.show()
```



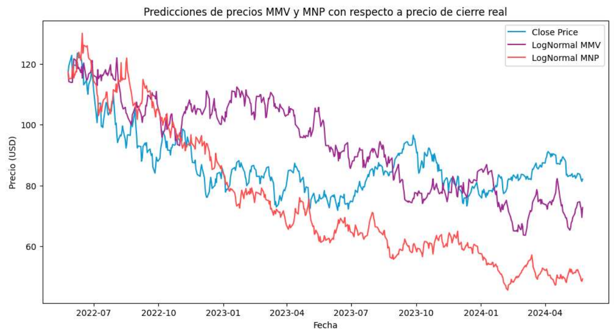
Intervalos de confianza (IC)

```
#IC al 95%
np.random.seed(145)
# Desviación Estándar
SMNP = np.sqrt(VLN(t=data_index, mu=muMNP, sigma=sigmaMNP, s0=close[0]))
t=data_index
date = data['Fecha']
S = close

plt.figure(figsize=(8, 5))
plt.plot(date, ELNMNP - 1.96 * SMNP, '--k', label='IC inf')
plt.plot(date, ELNMNP, '#FFC000', label=r'$\mu_S(t)$', linestyle='-.')
plt.plot(date, ELNMNP + 1.96 * SMNP, 'k', label='IC sup')
plt.plot(date, S, '#0F9ED5', label='S(t)')
plt.plot(date, SStMNP,'#FF5050', label='SS(t)')
plt.xlabel('Fecha', fontsize=10)
plt.ylabel('Precio (USD)', fontsize=10)
plt.title(r'Precios Brent Modelo Log-Normal por MMV con IC 95% ', fontsize=12)
plt.legend()
plt.show()
```



Comparativa de los precios modelados por MNV y MNP

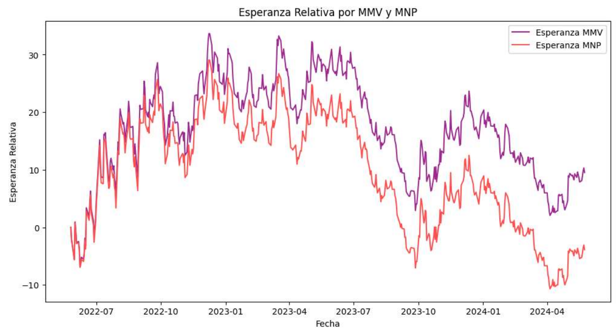
```
plt.figure(figsize=(12, 6))
plt.plot(date, close,'#0F9ED5',label='Close Price', linestyle='-')
plt.plot(date, SStMMV, '#A02B93', label='LogNormal MMV', linestyle='-')
plt.plot(date, SStMNP,'#FF5050',label='LogNormal MNP')
plt.xlabel('Fecha', fontsize=10)
plt.ylabel('Precio (USD)', fontsize=10)
plt.title(r'Predicciones de precios MMV y MNP con respecto a precio de cierre real', fontsize=12)
plt.legend()
plt.show()
```



Comparativa de las Esperanzas matemática por MNV y MNP

```
plt.figure(figsize=(12, 6))
plt.plot(date, close,'#0F9ED5',label='Close Price', linestyle='-')
plt.plot(date, ELNMMV, '#A02B93', label='Esperanza MMV', linestyle='-')
plt.plot(date, ELNMNP,'#FF5050',label='Esperanza MNP')
plt.xlabel('Fecha', fontsize=10)
plt.ylabel('Precio (USD)', fontsize=10)
plt.title(r'Esperanza por MMV y MNP con respecto a precio de cierre real', fontsize=12)
plt.legend()
plt.show()
```


```
plt.figure(figsize=(12, 6))
plt.plot(date, ELNMMV - close, '#A02B93', label='Esperanza MMV', linestyle='-')
plt.plot(date, ELNMNP - close,'#FF5050',label='Esperanza MNP')
plt.xlabel('Fecha', fontsize=10)
plt.ylabel('Esperanza Relativa', fontsize=10)
plt.title(r'Esperanza Relativa por MMV y MNP', fontsize=12)
plt.legend()
plt.show()
```



RESULTADOS

```
#A pesar que solo se deben seleccionar dos métodos, se ha aprovechado a
# determinar los parámetros del modelo LogNormal mediante MME
close = data['Cierre']
dt = 1

dataResults = {
    'Método': ['Máxima Verosimilitud', 'Momento no Paramétrico', 'Momentos Estadísticos'],
    'Mu': [muMMV, muMNP,muMME],
    'Sigma': [sigmaMMV, sigmaMNP, sigmaMME],
    'ECM': [ECMMMV, ECMMNP, ECMMME],
    'EPAM': [EPAMMMV, EPAMMNP, EPAMMME]
}

df = pd.DataFrame(dataResults)

print(df)
```

```
                   Método        Mu     Sigma       ECM      EPAM
0    Máxima Verosimilitud -0.000494  0.020826  0.089017  4.174807
1  Momento no Paramétrico -0.000810  0.021581  0.065240  2.839666
2   Momentos Estadísticos -0.000268  0.009115  0.111747  5.452877
```

SIMULACIÓN MONTECARLO

La simulación de Montecarlo en un modelo Log-Normal es una técnica que nos permite modelar, predecir y gestionar la incertidumbre y el riesgo asociados con los precios de las acciones. Se basa en la utiliza el modelo de movimiento browniano geométrico, donde los precios siguen la fórmula

$$S(t) = S_0 e^{\left((\mu - \frac{\sigma^2}{2})t + \sigma W(t)\right)}$$

donde (μ) es la tasa media de retorno, (σ) es la volatilidad, y $(W(t))$ es un proceso de Wiener (camino aleatorio). Genera múltiples caminos de precios posibles mediante la simulación de diferentes realizaciones del proceso estocástico. Los resultados se analizan para obtener una distribución de posibles precios futuros, permitiendo evaluar la probabilidad de diferentes escenarios de precios, proporcionando a los inversores y gestores de fondos una visión detallada y cuantitativa del comportamiento futuro potencial del mercado. (Kroese, D. P. et al. 2011,2013)

```
np.random.seed(144)
sims = 600
plt.figure(figsize=(11,7))
for i in range(sims):
  MCMMV = log_normal_model(t=data_index, mu=muMMV, sigma=sigmaMMV, s0=close[0])
  plt.plot(date, MCMMV)

plt.plot(date, ELNMMV - 1.96 * SMMV, '--k', label='IC inf')
plt.plot(date, ELNMMV + 1.96 * SMMV, '-k', label='IC sup')
plt.plot(date, ELNMMV, '-.k', label='Esperanza')
plt.xlabel('Fecha', fontsize=10)
plt.ylabel('Precios (USD)', fontsize=10)
plt.title('Simulación MonteCarlo para modelo LogNormal por MMV')
plt.legend(fontsize=11)
plt.show()
```