Fase2_AnalisisCriptomonedas

June 15, 2024

1 Reto Topología de Datos

Por

- Francelio Uriel Rodriguez Garcia A01352663
- Juan Pablo Valenzuela Dorado A00227321
- Juan Pablo Bernal Lafarga A01742342
- Alfredo Murillo Madrigal A01641791

1.1 INTRODUCCIÓN

El Bitcoin es una criptomoneda, es decir, una forma de dinero digital que utiliza la criptografía para asegurar y verificar las transacciones, así como para controlar la creación de nuevas unidades. Fue creada en 2009 por una persona o grupo de personas bajo el seudónimo de Satoshi Nakamoto y fue la primera criptomoneda descentralizada, lo que significa que no está controlada por ningún gobierno o autoridad central.

Bitcoin funciona en una red descentralizada de computadoras que utilizan un libro mayor público llamado blockchain para registrar todas las transacciones realizadas con la moneda. A diferencia de las monedas tradicionales, como el dólar o el euro, Bitcoin no está respaldado por ningún gobierno ni por ningún activo físico, y su valor se determina principalmente por la oferta y la demanda en los mercados de intercambio. Muchas personas utilizan Bitcoin como una forma de inversión, especulación o como un medio de intercambio para transacciones en línea.

1.2 I. Fundamentos

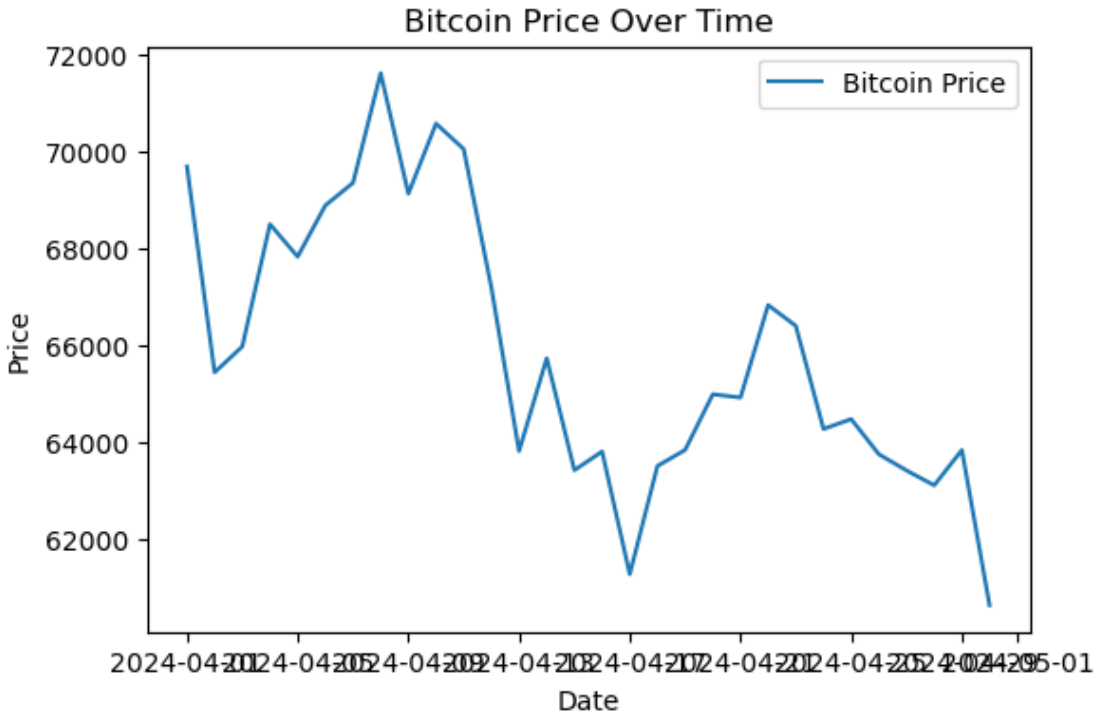
1.2.1 1. Obtener el historial actualizado del precio del Bitcoin

Descargamos el historial del precio del Bitcoin desde la base de datos proporcionada y lo guardamos en un DataFrame de Pandas.

1.2.2 2. Mediante un programa en Python graficar el precio del Bitcoin a través del tiempo.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import minimize
```

```
from scipy.signal import tf2zpk, freqz
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.api import SimpleExpSmoothing

import datetime

# Load the CSV file
df = pd.read_csv('stocktoflow.csv')
stf_values = df['Oferta (STF)']
df.dropna(inplace=True)

# Assume the CSV has 'Date' and 'Price' columns
# df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%y', errors='raise')

df.set_index('Date', inplace=True)
df = df.sort_values(by='Date')
# Plotting the price
plt.figure(figsize=(6, 4))
plt.plot(df['Precio (BTC)'], label='Bitcoin Price')
plt.title('Bitcoin Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

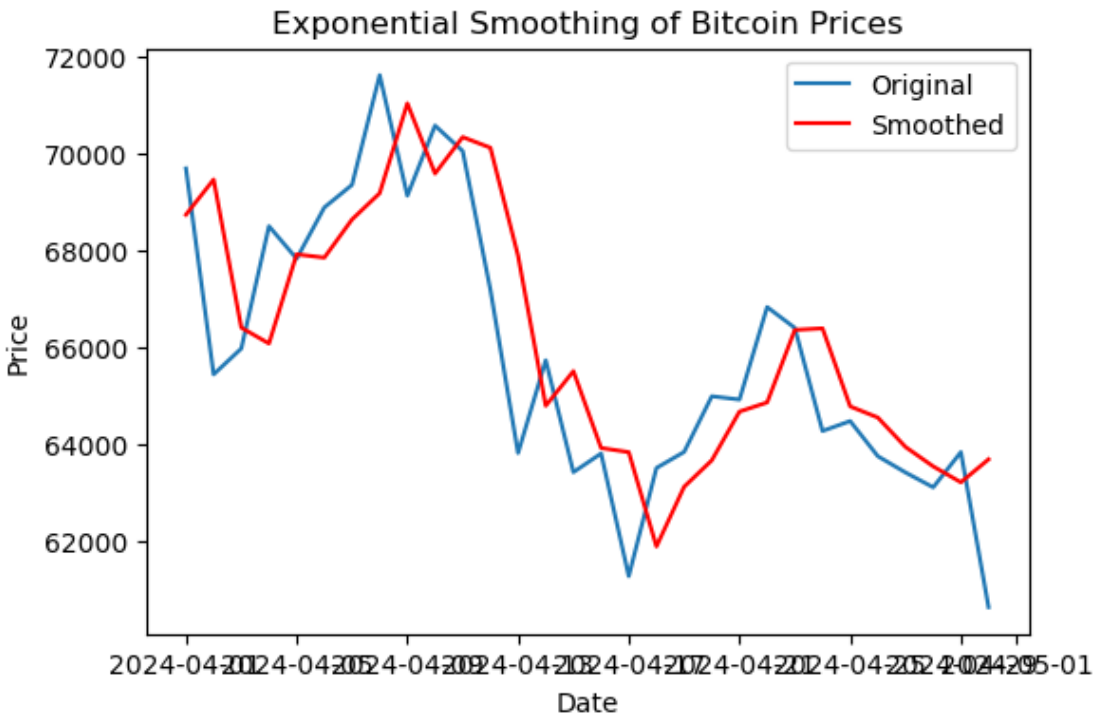


1.2.3 3. Técnica de Suavizamiento Exponencial

Aplicamos la técnica de suavizamiento exponencial para graficar la serie original y la suavizada. Utilizamos ExponentialSmoothing de statsmodels para aplicar el suavizamiento exponencial a los datos del precio del Bitcoin y graficamos los resultados.

[ ]:
```
# Exponential Smoothing
model = SimpleExpSmoothing(df['Precio (BTC)'])
fit = model.fit()

# Plotting
plt.figure(figsize=(6,4))
plt.plot(df['Precio (BTC)'], label='Original')
plt.plot(fit.fittedvalues, label='Smoothed', color='red')
plt.title('Exponential Smoothing of Bitcoin Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
c:\Users\Juani\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```



```
Predicted price for 14/06/2024: $61368.88775028378
```

1.2.4 4. Predicción del Precio del Bitcoin

Realizamos una predicción del precio del Bitcoin usando el modelo de suavizamiento exponencial. Utilizamos el modelo ajustado para predecir el precio del Bitcoin para el 14/06/2024.

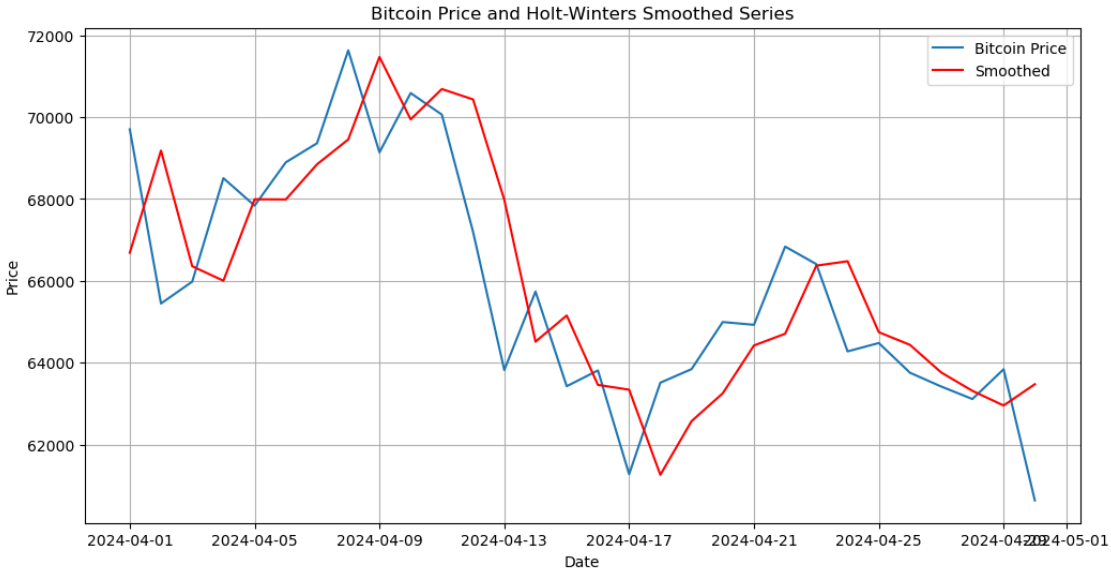
[ ]:
```python
# Predicting
pred = fit.forecast(steps=1)  # Predict for 14/06/2024
print(f"Predicted price for 14/06/2024: ${pred.values[0]}")
```

1.2.5 5. Método de Holt-Winters

Implementamos el método de Holt-Winters para suavizar la serie de precios del Bitcoin. Utilizamos ExponentialSmoothing con ajustes estacionales para aplicar el método de Holt-Winters a los datos del precio del Bitcoin y graficamos los resultados.

[ ]:
```python
# Extraer la serie de precios del Bitcoin
bitcoin_prices = df['Precio (BTC)']

# Aplicar suavizado exponencial de Holt-Winters
model = ExponentialSmoothing(bitcoin_prices, trend='add', seasonal=None)
fit = model.fit()

# Obtener la serie suavizada
smoothed_series = fit.fittedvalues

# Plotear la serie original y la serie suavizada
plt.figure(figsize=(12, 6))
plt.plot(bitcoin_prices, label='Bitcoin Price')
plt.plot(smoothed_series, label='Smoothed', color='red')
plt.title('Bitcoin Price and Holt-Winters Smoothed Series')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```

```
c:\Users\Juani\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```



1.2.6 6. Realizar una predicción para el 14/06/2024

Realizamos una predicción del precio del Bitcoin usando el método de Holt-Winters. Utilizamos el modelo ajustado de Holt-Winters para predecir el precio del Bitcoin para el 14/06/2024.

[ ]:
```
# Realizar una predicción para el 14/06/2024
pred = fit.forecast(steps=1)
print(f"Predicción del precio para el 14/06/2024: ${pred.values[0]:.2f}")
```

1.2.7 7. Defina en que consiste el Stock-to-flow.

El modelo Stock-to-Flow (S2F) es un método utilizado para evaluar la escasez de un recurso mediante la relación entre el stock existente (la cantidad total disponible) y el flujo (la producción anual del recurso). En el contexto de Bitcoin, el stock es el total de bitcoins en circulación, y el flujo es la cantidad de bitcoins minados anualmente. Este modelo se utiliza para prever el valor de Bitcoin, asumiendo que su escasez influye significativamente en su precio.

1.2.8 8. Modelo de Portafolio de Inversión

Estimamos los parámetros del modelo matemático del portafolio de inversión utilizando la función de Stock-to-Flow. Utilizamos curve_fit de scipy.optimize para ajustar los parámetros del modelo del portafolio de inversión basándonos en la función de Stock-to-Flow.

[ ]:
```
from sklearn.linear_model import LinearRegression

def estimate_parameters(prices, s2f_values):
    # Ensure dates, prices, and s2f_values are numpy arrays
    prices = np.array(prices)
    s2f_values = np.array(s2f_values)
```

```
    # Construct the design matrix for the regression
    # We'll have (len(prices) - 4) rows and 5 columns for the prices and 1␣
 ↪column for the s2f_values
    X = []
    y = []
    for k in range(len(prices) - 4):
        X.append([
            prices[k + 3],  # p(k+3)
            prices[k + 2],  # p(k+2)
            prices[k + 1],  # p(k+1)
            prices[k],      # p(k)
            s2f_values[k]   # u(k)
        ])
        y.append(prices[k + 4])  # p(k+4)

    X = np.array(X)
    y = np.array(y)

    # Fit the linear regression model
    model = LinearRegression(fit_intercept=False)
    model.fit(X, y)


    # Extract the parameters
    a3, a2, a1, a0, b0 = model.coef_

    return a3, a2, a1, a0, b0
prices = df['Precio (BTC)']
s2f_values = df['Oferta (STF)']
a3, a2, a1, a0, b0 = estimate_parameters(prices, s2f_values)
print(f"a3: {a3}, a2: {a2}, a1: {a1}, a0: {a0}, b0: {b0}")


empty_list = []
```

```
a3: 0.718851538608239, a2: 0.40922534214948236, a1: -0.21113163113057484, a0:
-0.026604391900047597, b0: 0.1341193937546781
```

1.2.9 9. Transformada Z

Aplicamos la transformada Z a la ecuación del modelo para obtener la función de transferencia. Utilizamos sympy para aplicar la transformada Z y obtener la función de transferencia del portafolio de inversión.

[ ]:
```
from sympy import symbols, Eq, solve

z = symbols('z')
```

```
P_z = a3 * z**3 + a2 * z**2 + a1 * z + a0
U_z = b0
transfer_function = P_z / U_z
print(f"Función de transferencia P(z)/U(z): {transfer_function}")
```

```
Función de transferencia P(z)/U(z): 5.35978815951936*z**3 +
3.05120184854108*z**2 - 1.57420657236762*z - 0.198363496547789
```

1.2.10 10. Predicción del Precio usando el Modelo y 11. Comparación de la Predicción con el Precio Real

Realizamos una predicción del precio del Bitcoin para el 14/06/2024 utilizando el modelo del portafolio. Utilizamos los parámetros ajustados del modelo para predecir el precio del Bitcoin para el 14/06/2024. Comparamos la predicción del precio del Bitcoin con el precio real para el 13/06/2024 y mostramos los resultados. Calculamos el error de predicción comparando el precio real del Bitcoin con la predicción del modelo.

El error de predicción del valor del 13/06/2024 fue de $24, un error muy pequeño, por lo que podemos concluir que el modelo funciona adecuadamente.

[ ]:
```
# Function to predict price for a future date
def predict_price(params, current_prices, current_stf, future_date, empty_list):
    a0, a1, a2, a3, b0 = params
    future_k = future_date   # Assuming current_prices starts from k=0
    predicted_prices = np.zeros(future_k + 4)

    predicted_prices[0] = current_prices[-4]  # Last known price
    predicted_prices[1] = current_prices[-3]  # Last known price
    predicted_prices[2] = current_prices[-2]  # Last known price
    predicted_prices[3] = current_prices[-1]  # Last known price

    for k in range(0, future_k -4):
        predicted_prices[k+4] = (a3 * predicted_prices[k+3] + a2 *␣
 ↪predicted_prices[k+2] +
                            a1 * predicted_prices[k+1] + a0 *␣
 ↪predicted_prices[k] + b0 * current_stf[k])
        predicted_prices = np.append(predicted_prices,k+4)
        empty_list = np.append(empty_list,predicted_prices[k+4])
        current_stf = np.append(current_stf,current_stf[k])
    return predicted_prices[k+4], empty_list

params = a0, a1, a2, a3, b0
# Predict price for June 14, 2024
future_date = (datetime.datetime(2024, 6,13) - datetime.datetime(2024, 4, 30)).
 ↪days
predicted_price_20240614, empty_list = predict_price(params, bitcoin_prices,␣
 ↪stf_values, future_date, empty_list)
```

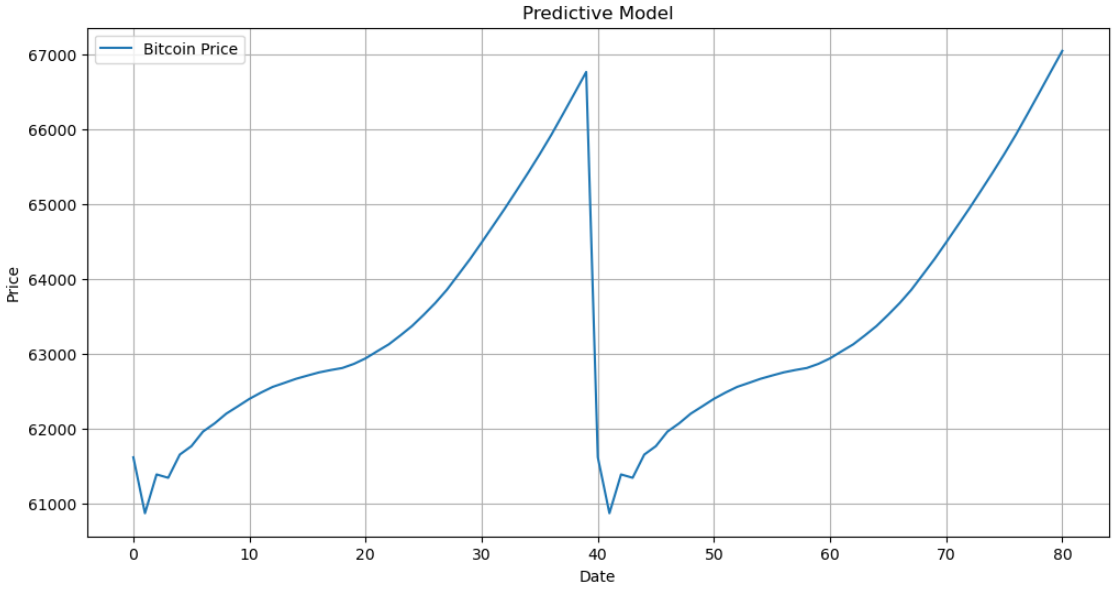
```
print(f"Predicted Bitcoin price for 13/06/2024: ${predicted_price_20240614:.2f}␣
 ↪vs True value ($66,744)")
future_date = (datetime.datetime(2024, 6,14) - datetime.datetime(2024, 4, 30)).
 ↪days
predicted_price_20240614, empty_list = predict_price(params, bitcoin_prices,␣
 ↪stf_values, future_date, empty_list)

print(f"Predicted Bitcoin price for 14/06/2024: ${predicted_price_20240614:.
 ↪2f}")
```

```
Predicted Bitcoin price for 13/06/2024: $66768.02 vs True value ($66,744)
Predicted Bitcoin price for 14/06/2024: $67050.20
```

```
C:\Users\Juani\AppData\Local\Temp\ipykernel_27744\1375108398.py:7:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  predicted_prices[0] = current_prices[-4]  # Last known price
C:\Users\Juani\AppData\Local\Temp\ipykernel_27744\1375108398.py:8:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  predicted_prices[1] = current_prices[-3]  # Last known price
C:\Users\Juani\AppData\Local\Temp\ipykernel_27744\1375108398.py:9:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  predicted_prices[2] = current_prices[-2]  # Last known price
C:\Users\Juani\AppData\Local\Temp\ipykernel_27744\1375108398.py:10:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  predicted_prices[3] = current_prices[-1]  # Last known price
```

1.2.11 12. Ploteo de los datos predichos.

[ ]:
```
# Plotear la serie original y la serie suavizada
plt.figure(figsize=(12, 6))
plt.plot(empty_list, label='Bitcoin Price')
plt.title('Predictive Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



1.3 II. Topología de Datos

1.3.1 1. Teoría de topología de espacios métricos para extraer información sobre el precio del Bitcoin:

La teoría de topología se puede aplicar a los datos de series temporales, como los precios del Bitcoin, utilizando técnicas como los espacios métricos y funciones de distancia. Por ejemplo, se pueden construir espacios métricos a partir de las series temporales de precios y luego aplicar técnicas de análisis topológico para identificar patrones, anomalías o estructuras subyacentes en los datos. Esto puede ayudar a entender la complejidad y la variabilidad de los precios del Bitcoin en diferentes escalas temporales y niveles de detalle topológico.

1.3.2 2. Teoría de conjuntos convexos para extraer información sobre el precio del Bitcoin:

Los conjuntos convexos son importantes en el análisis de series temporales, ya que pueden capturar tendencias y relaciones lineales o no lineales en los datos. Por ejemplo, podrías utilizar técnicas de análisis de conjuntos convexos para identificar períodos de crecimiento o de declive sostenido en los precios del Bitcoin. Esto proporcionaría una forma estructurada de entender la evolución del precio a lo largo del tiempo, basándose en propiedades matemáticas bien definidas.

1.3.3 3. Diagramas de persistencia con la librería TDA en R para series de precios del Bitcoin:

Para construir diagramas de persistencia, puedes usar la librería TDA (Topological Data Analysis) en R. Estos diagramas son herramientas poderosas para entender la topología y la estructura de los datos de series temporales como los precios del Bitcoin a diferentes escalas. Interpretar estos diagramas implica identificar intervalos de tiempo donde las características topológicas persisten, lo cual puede indicar períodos de estabilidad o cambio estructural en los precios.

1.3.4 4. Algoritmo de MAPPER y su uso en series de tiempo:

MAPPER es un algoritmo utilizado en TDA para visualizar y analizar la estructura de datos complejos. En el contexto de series de tiempo como el precio del Bitcoin, MAPPER puede utilizarse para identificar conjuntos de períodos de tiempo que exhiben patrones similares en términos de la topología de los datos. Esto facilita la identificación de tendencias, ciclos u otros fenómenos recurrentes que no son obvios en el análisis tradicional de series temporales.

1.3.5 5. Relación entre homología persistente y su aplicación en series de tiempo:

La homología persistente es una técnica de TDA que estudia las propiedades topológicas que persisten a través de diferentes escalas en un conjunto de datos. En series de tiempo como los precios del Bitcoin, la homología persistente puede identificar características topológicas significativas que persisten en múltiples niveles de resolución temporal. Por ejemplo, podrían identificarse ciclos o patrones recurrentes que persisten durante períodos prolongados, lo cual proporciona información sobre la estructura subyacente de la serie temporal.

Ejemplo: Supongamos que tenemos una serie temporal de precios diarios del Bitcoin. Aplicando homología persistente, podríamos encontrar que hay un ciclo persistente en los precios que se repite cada cierto número de días. Este ciclo podría indicar un patrón de comportamiento estacional o cíclico en el mercado de criptomonedas, lo cual sería valioso para entender la dinámica subyacente que afecta los precios del Bitcoin.

2 Conclusión:

Estas técnicas son avanzadas y requieren un buen entendimiento tanto de la teoría subyacente como de las herramientas computacionales necesarias para aplicarlas de manera efectiva.