Modelado del Precio de Bitcoin con Ecuaciones Diferenciales Parciales

Valentin Santiago Diaz Moyano
Pedro Salomone
Manon Gallet

El bitcoin: definición e historia

- Concepto inventado en 2008 por Satoshi Nakamoto

- Forma de moneda digital, o criptomoneda, descentralizada y basada en una tecnología de contabilidad distribuida llamada blockchain.

- Extremadamente volátil: pasando de 863 $US el 9 de enero de 2017 a un máximo de 17.550 $US el 11 de diciembre de 2017

El gráfico transacción-dirección

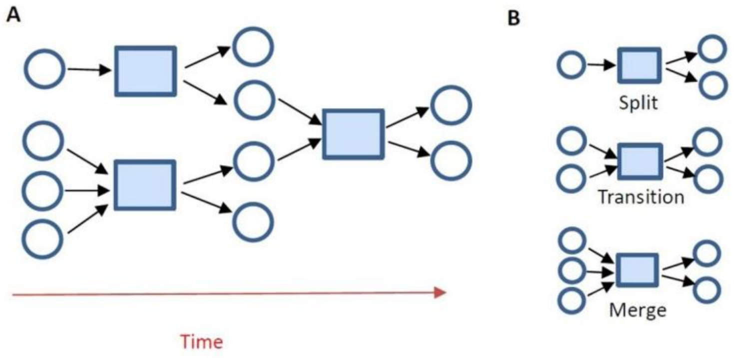


Figure 6.2 (A) A transaction-address graph; (B) Split ($C_{1 \longrightarrow 2}$), Transition ($C_{2 \longrightarrow 2}$), and Merge ($C_{3 \longrightarrow 2}$) chainlets. The three types, Merge, Transition, and Split are determined according to the relative number of input addresses and output addresses, and correspond to the state that the former is greater than, equal to, or less than the latter, respectively. Addresses and transactions are shown with circles and rectangles, respectively. An arrow indicates a transfer of bitcoins.

Una arista representa la transferencia de bitcoin entre varios nodos

El concepto de chainlet

Cluster Espectral y Embedding

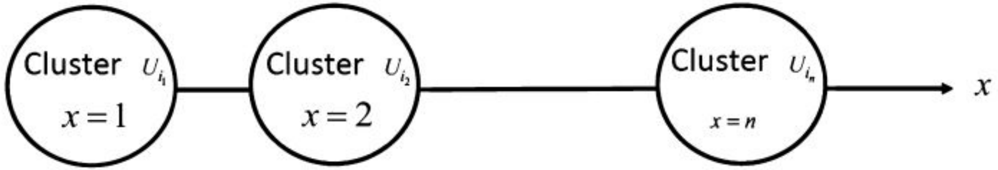


Figure 6.4 Embedding of chainlet clusters into the *x*-axis.

Modelo de EDP

$$\frac{\partial u(x,t)}{\partial t} = \underbrace{\frac{\partial}{\partial x}\left(d(x)\frac{\partial u(x,t)}{\partial x}\right)} + \underline{r(t)u(x,t)h(x)},$$

refleja la tasa de cambio del precio del bitcoin entre los clusters

describe la tasa de variación del precio del bitcoin dentro del cluster x

$d(x)$ describe la interacción de los clusters de chainlets

$r(t)$ representa la tasa de variación con respecto a t

$h(x)$ describe la heterogeneidad espacial de las distintas clusters de chainlets o patrones de transacciones

$$u(x, t) \equiv b_0 m(x, t) + \alpha(x),$$

$\alpha(x)$ describe la heterogeneidad de los distintos clusters de chainlets en el precio del bitcoin

$m(x, t)$ es la utilidad predictiva del índice Google Trends del clúster de chainlets x

$$\begin{cases} \frac{\partial m(x,t)}{\partial t} = d\frac{\partial^2 m}{\partial x^2} + k\alpha(x)r(t)\left(m(x, t) + \frac{1}{b_0}\alpha(x)\right) + \frac{d}{b_0}\alpha''(x), \\ m(x, 1) = \phi(x), L_1 < x < L_2, \\ \frac{\partial m}{\partial x}(L_1, t) = \frac{\partial m}{\partial x}(L_2, t) = 0, t > 1, \\ \text{Forecasted bitcoin price at time } t = \int_{L_1}^{L_2} \left(b_0 m(x, t) + \alpha(x)\right) dx, \end{cases}$$

$$r(t) = b_1 + e^{-(t-b_2)}$$

$$m(x_i, t_j) = (\text{Google Trends Index on “Bitcoin” at time } t_j) * P_0(x_i, t_j),$$

$$P_0(x_i, t_j) = \frac{\text{bitcoin transaction volume of chainlet cluster } x_i \text{ at time } t_j}{\text{total bitcoin transaction volume of all chainlet clusters at } t_j}.$$

El método de Crank-Nicolson

$$\frac{m_i^{k+1}-m_i^k}{\Delta t}=d\,\frac{m_{i-1}^{k+1}-2m_i^{k+1}+m_{i+1}^{k+1}}{\Delta x^2}+k\alpha(x_i)r(t^k)m_i^k+\frac{k\alpha^2(x_i)r(t^k)}{b_0}+\frac{d}{b_0}\alpha''(x_i)$$

$$\begin{bmatrix} 1+\frac{2d\Delta t}{\Delta x^2} & \frac{-d\Delta t}{\Delta x^2} & 0 & \cdots & 0 \\ \frac{-d\Delta t}{\Delta x^2} & 1+\frac{2d\Delta t}{\Delta x^2} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \frac{-d\Delta t}{\Delta x^2} \\ 0 & \cdots & 0 & \frac{-d\Delta t}{\Delta x^2} & 1+\frac{2d\Delta t}{\Delta x^2} \end{bmatrix} m^{k+1}=F_i^k$$

El método de Crank-Nicolson

$$\begin{bmatrix} 1+\frac{2d\Delta t}{\Delta x^2} & \frac{-d\Delta t}{\Delta x^2} & 0 & \cdots & 0 \\ \frac{-d\Delta t}{\Delta x^2} & 1+\frac{2d\Delta t}{\Delta x^2} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \frac{-d\Delta t}{\Delta x^2} \\ 0 & \cdots & 0 & \frac{-d\Delta t}{\Delta x^2} & 1+\frac{2d\Delta t}{\Delta x^2} \end{bmatrix} m^{k+1} = F_i^k$$

Método de punto fantasma

$$\begin{bmatrix} -\frac{1}{2\Delta x} & \frac{1}{2\Delta x} & 0 & \cdots & & \\ -\frac{d\Delta t}{(\Delta x)^2} & 1+\frac{2d\Delta t}{(\Delta x)^2} & -\frac{d\Delta t}{(\Delta x)^2} & 0 & & \\ 0 & -\frac{d\Delta t}{(\Delta x)^2} & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots & \\ & & & -\frac{d\Delta t}{(\Delta x)^2} & 1+\frac{2d\Delta t}{(\Delta x)^2} & -\frac{d\Delta t}{(\Delta x)^2} \\ 0 & \cdots & \cdots & 0 & -\frac{1}{2\Delta x} & \frac{1}{2\Delta x} \end{bmatrix}$$

Implementación

In [ ]:

```
!pip install scikit-learn
```

In [ ]:

```
import yfinance as yf
import numpy as np
from scipy.interpolate import CubicSpline
from scipy.optimize import minimize
from random import random
import pandas as pd
from sklearn.cluster import SpectralClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from google.colab import drive
```

In [ ]:

```
def getGTrendIndex(year):
    google_trend_data='/content/drive/My Drive/ProyectoNumerico3/GTrendIndex' + str(year)
+'.txt'
    weekly_trend_index=np.loadtxt(google_trend_data)
    #Google only provides weekly reports, so we asume that index as an average of two con
secutive weeks
    av_weekly_trend_index = [(weekly_trend_index[i] + weekly_trend_index[i+1]) / 2 for i
in range(len(weekly_trend_index)-1)]
    daily_trend_index = []
    for index in av_weekly_trend_index:
        daily_trend_index.extend([index] * 7)
    daily_trend_index.append(daily_trend_index[-1])
    return daily_trend_index
```

In [ ]:

```
#Datos obtenidos de https://github.com/cakcora/CoinWorks
#Descargamos los chainlet
def getChainlet(year):
    archivo_txt = '/content/drive/My Drive/ProyectoNumerico3/Edit_AmoChainlet.txt'
    datos = pd.read_csv(archivo_txt, delimiter='\t')

    # Filtra las filas correspondientes al año 2017 y elimina las primeras
    # 3 columnas que no son necesarias
    datos_year = datos[datos.iloc[:, 0] == year]
    datos=np.array(datos_year)
    datos=datos[:,3:]

    #Estructuramos los datos
    #Cada columna corresponde a un chainlet x:y
    chainlets=[]
    for i in range(400):
        chainlets.append(datos[:,i])
    chainlets=np.array(chainlets)
    return datos
```

```python
def Chainlet_Spectral_Clustering(chainlets):
    # Aplicar un escalado estándar a los datos
    scaler = StandardScaler()
    chainlets_scaled = scaler.fit_transform(chainlets)

    # Número de clusters deseados
    num_clusters = 10

    # Aplicar Spectral Clustering
    spectral = SpectralClustering(n_clusters=num_clusters, affinity='nearest_neighbors', n_neighbors=10)
    labels = spectral.fit_predict(chainlets_scaled)

    clusters = [[] for _ in range(num_clusters)]

    #Organizamos los cluster en un vector
    for i, label in enumerate(labels):
        clusters[label].append(chainlets[i])

    cluster_similarity_matrix = np.zeros((num_clusters, num_clusters))

    for i in range(num_clusters):
        for j in range(num_clusters):
            # Puedes usar cualquier medida de similitud que prefieras, aquí se usa la distancia euclidiana
            similarity = pairwise_distances([np.mean(clusters[i], axis=0)], [np.mean(clusters[j], axis=0)])
            cluster_similarity_matrix[i, j] = similarity

    # Encontrar el orden de los clusters basado en la similitud
    cluster_order = np.argsort(np.sum(cluster_similarity_matrix, axis=0))

    # Reorganizar el vector clusters según el nuevo orden
    cluster = [clusters[i] for i in cluster_order]
    return cluster
```

```python
#Definimos el esquema de diferencias finitar para resolver
#T_final>=3
def Crank_Nicolson(x, d, phi, alfa, alfa_seg, r, k, b_0,T_final):
    t= T_final - 2 #usamos 2 dias para predecir el precio

    n = len(x)
    hx = np.abs(x[-1] - x[0]) / n
    ht = hx*10
    m = int((T_final - t) /ht)

    #Armamos la matriz
    sub_diag = -np.ones(n+1) * d*ht/hx/hx
    subdiag_matrix = np.diag(sub_diag,-1) + np.diag(sub_diag,1)
    A = np.zeros((n+2,n+2)) + np.eye(n+2) * (1+2*d*ht/hx/hx) + subdiag_matrix
    A[-1, -2:] = np.array([-1/2/hx, 1/2/hx])
    A[0,:2] = np.array([-1/2/hx, 1/2/hx])

    #Funciones para el vector F
    f = lambda x,t: (k * (alfa(x) **2) * r(t) / b_0) + (d / b_0 * alfa_seg(x))
    multiplicador = lambda x,t: k*alfa(x)*r(t)

    #Armamos la matriz con las soluciones
    sol = np.zeros((n,m))
    sol[:,0]= phi(x)


    F = np.zeros(n+2)

    for i in range(1,m):
        for j in range(n):
            X0 = sol[:,i-1].copy()
            X0[j] = X0[j] * multiplicador(x[j],t) + f(x[j],t)
            F[1:-1] = X0
        lineal_sol = np.linalg.solve(A,F)
        for j in range(n):
            sol[j,i] = lineal_sol[j]
        t += ht

    return sol
```

```python
#Resolvemos la ecuacion diferencial dada
def pde_solver(params, m_fun):
    d = params[0]
    b_0 = params[1]
    r = lambda t: params[2] + np.exp(-(t-params[3])**2)
    k = params[4]

    x = np.linspace(1,10,100)

    # Evaluar el spline en los puntos definidos
    alfa_i=params[5:]
    alfa=CubicSpline(list(range(1,11)), np.random.rand(10), bc_type='clamped') #clamped 
means a'(1)=a(10)=0
    alfa_seg = alfa.derivative(nu=2)

    Precios=np.zeros(52)
    i=0
    for T_final in list(range(1,366))[6::7]:
        phi=CubicSpline(list(range(1,11)), m_fun[:,T_final-2]) #=m(1,t)
        Predict = Crank_Nicolson(x, d, phi, alfa, alfa_seg, r, k, b_0, T_final)
        u_xt = b_0*Predict[:,-1]
        u_xt+=alfa(x)
        Precios[i] = np.trapz(u_xt,x)
        i+=1
    return Precios
```

```python
#Get data from a year
def getData(year):
    symbol = "BTC-USD"
    start_date = str(year) + '-01-01'
    end_date = str(year) + '-12-31'
    return yf.download(symbol, start=start_date, end=end_date)["Adj Close"][6::7]
```

In [ ]:

```python
#Definimos la funcion de costo
def cost_function(params, m_fun, btc_price):
    # Obtener datos observados u_obs y ubicaciones x, t
    u_approx = pde_solver(params, m_fun)
    return np.linalg.norm(u_approx - btc_price)

def factor_cost_function(factor,pred, price):
    return np.linalg.norm(price - factor*pred)
```

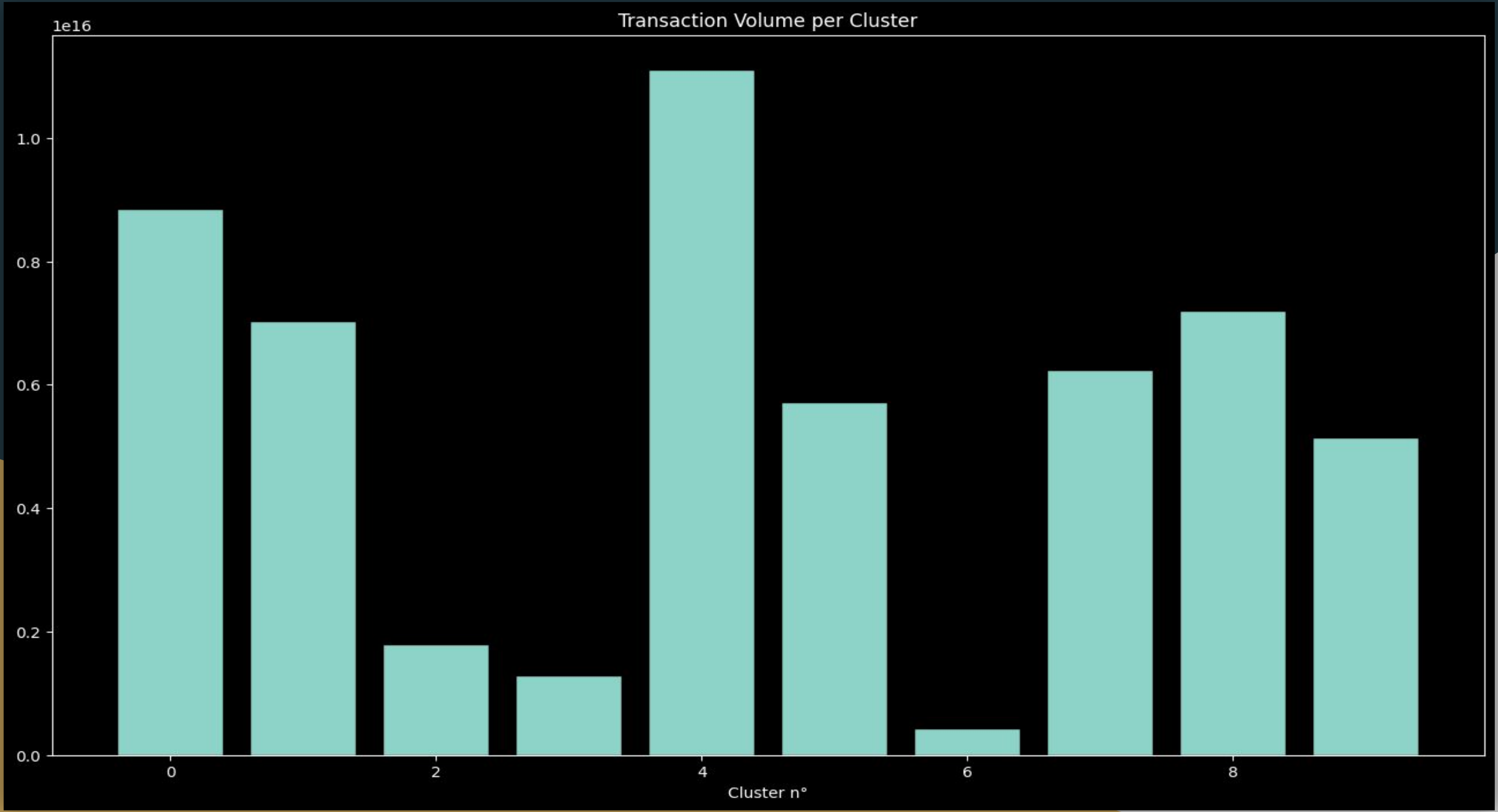
In [ ]:

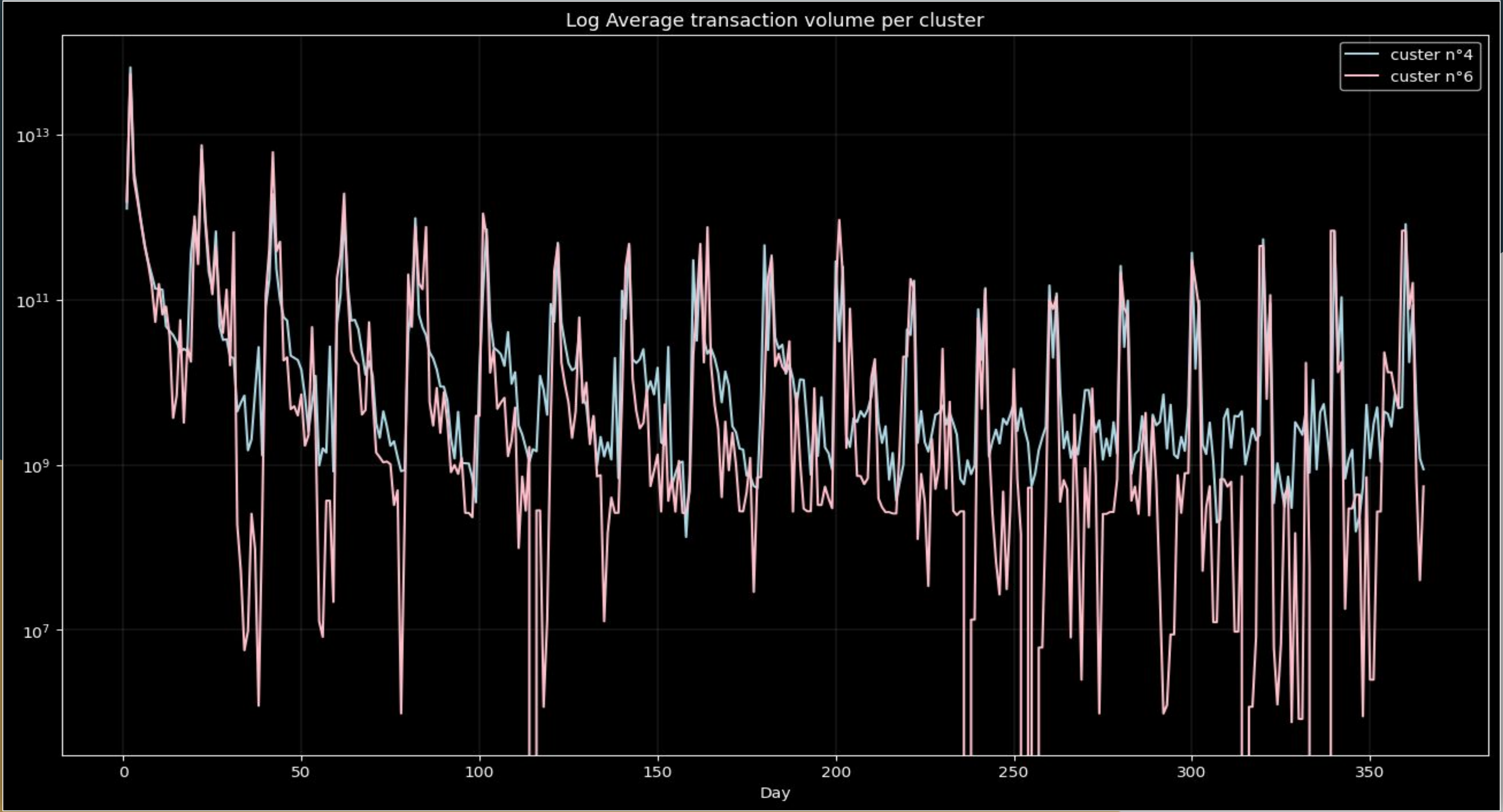
```python
def getParam(m_fun,btc_price):
    #Minimizacion de funcion de costo
    initial_guess=np.random.rand(15)
    #Supongamos que x, t y u_obs son tus datos observados y ubicaciones
    result = minimize(cost_function, initial_guess, args=(m_fun,btc_price), method='L-BFG
S-B')
    return result.x

def getFactor(pred, price):
    #Minimizacion de funcion de costo
    initial_guess=15
    #Supongamos que x, t y u_obs son tus datos observados y ubicaciones
    result = minimize(factor_cost_function, initial_guess, args=(pred,price,), method='L-
BFGS-B')
    return result.x
```

$$\begin{cases} \frac{\partial m(x,t)}{\partial t} = d\frac{\partial^2 m}{\partial x^2} + k\alpha(x)r(t)\left(m(x,t) + \frac{1}{b_0}\alpha(x)\right) + \frac{d}{b_0}\alpha''(x), \\ m(x,1) = \phi(x), L_1 < x < L_2, \\ \frac{\partial m}{\partial x}(L_1,t) = \frac{\partial m}{\partial x}(L_2,t) = 0, t > 1, \\ \text{Forecasted bitcoin price at time } t = \int_{L_1}^{L_2} \left(b_0 m(x,t) + \alpha(x)\right) dx, \end{cases}$$

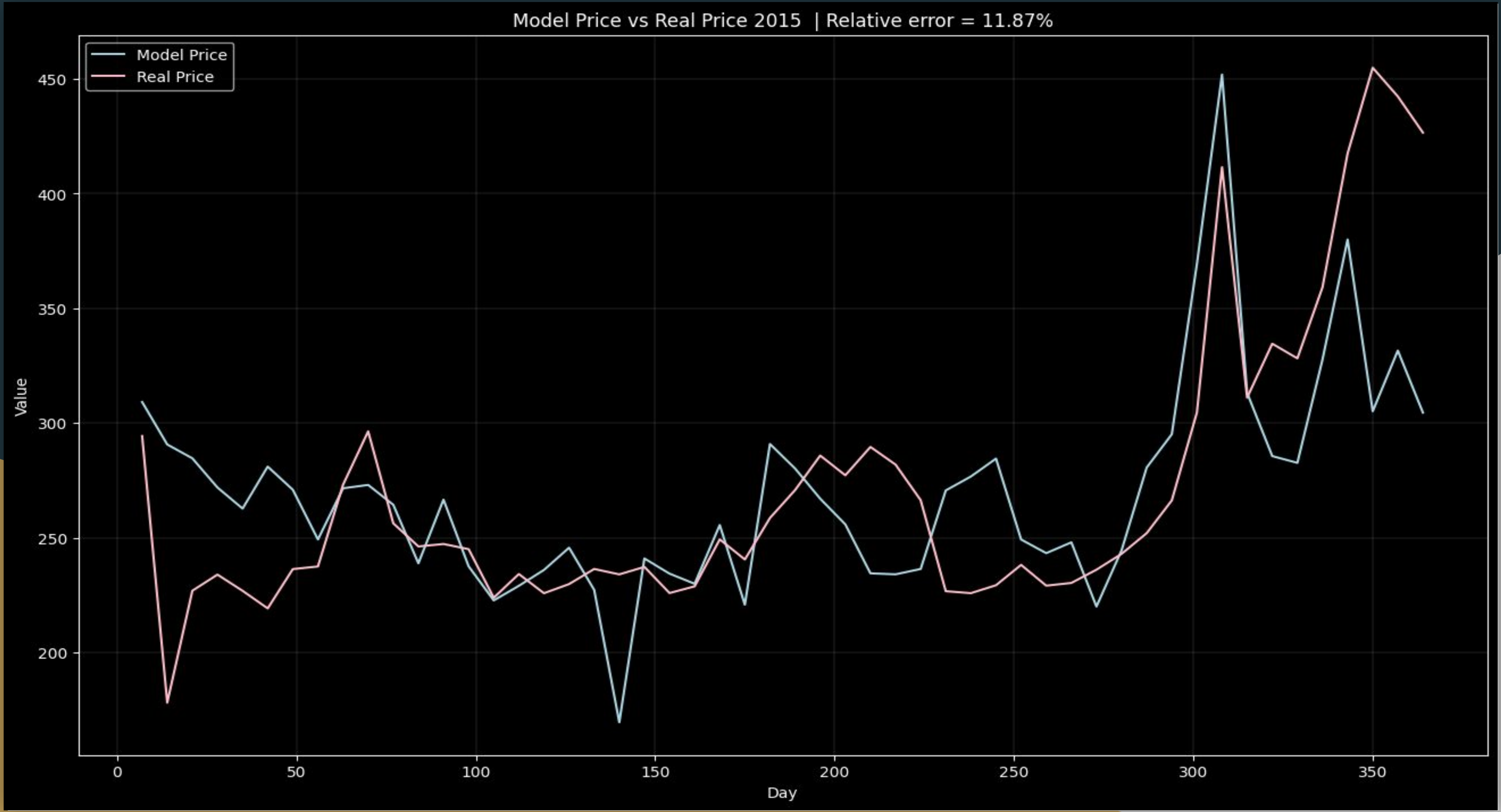
Entrenamiento de parámetros para datos del año 2015

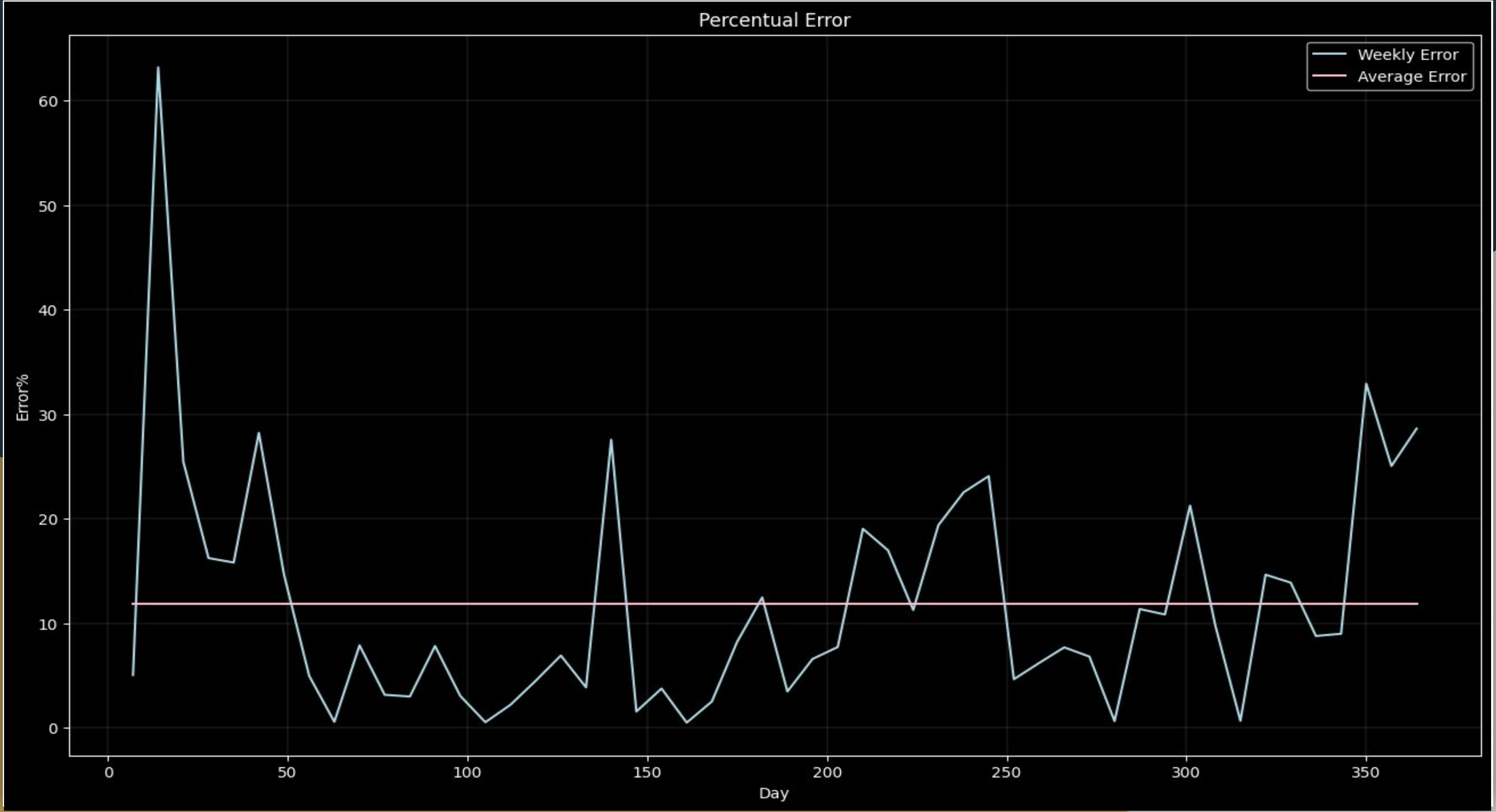
Clusters


Transaction Volume per Cluster
1e16
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
Cluster n°


Log Average transaction volume per cluster
custer n°4
custer n°6
10^{13}
10^{11}
10^{9}
10^{7}
0
50
100
150
200
250
300
350
Day

Los resultados


Model Price vs Real Price 2015 | Relative error = 11.87%
Model Price
Real Price
450
400
350
300
250
200
Value
0
50
100
150
200
250
300
350
Day

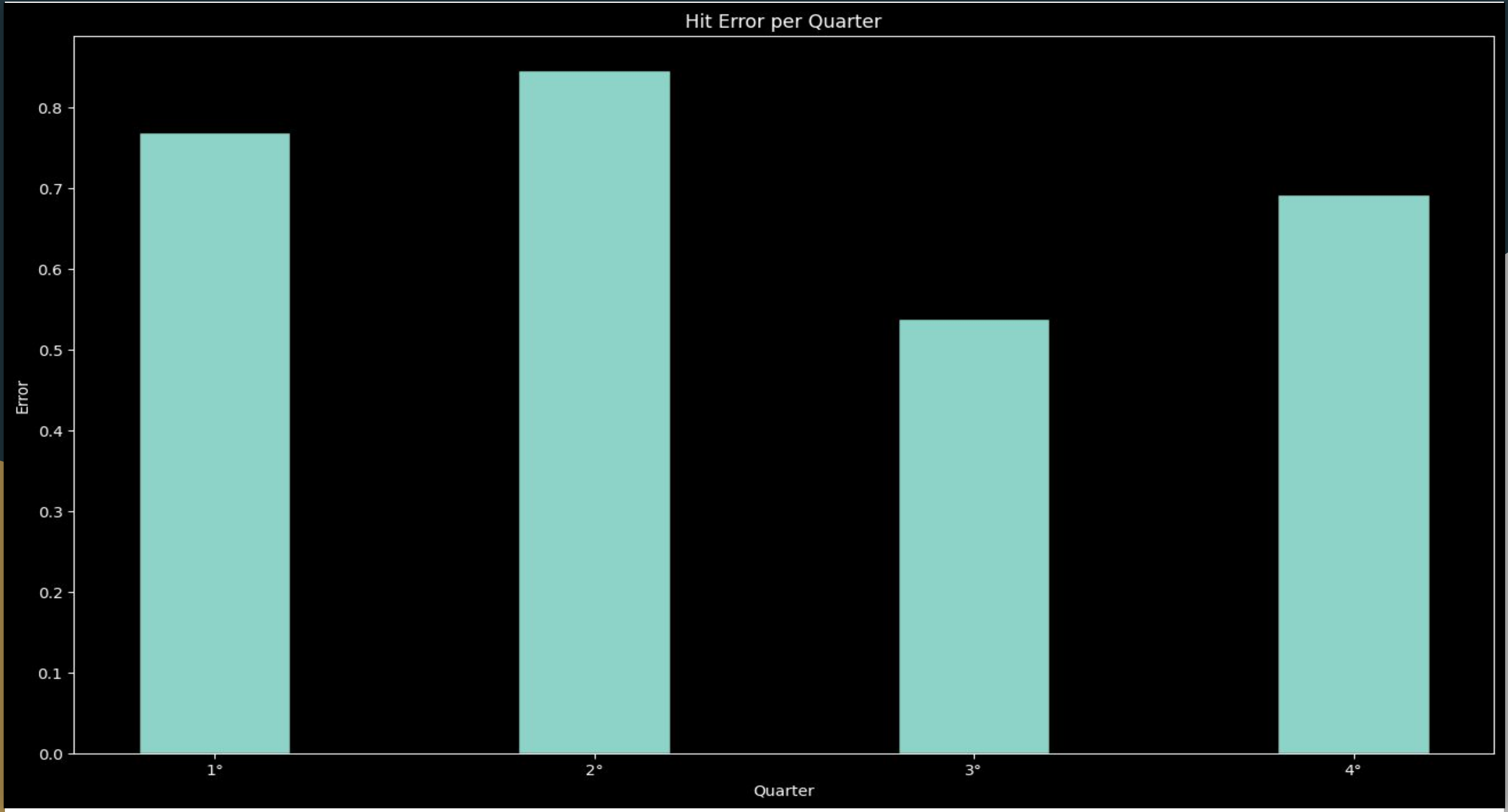


Percentual Error
Weekly Error
Average Error
Error%
Day
0
10
20
30
40
50
60
0
50
100
150
200
250
300
350

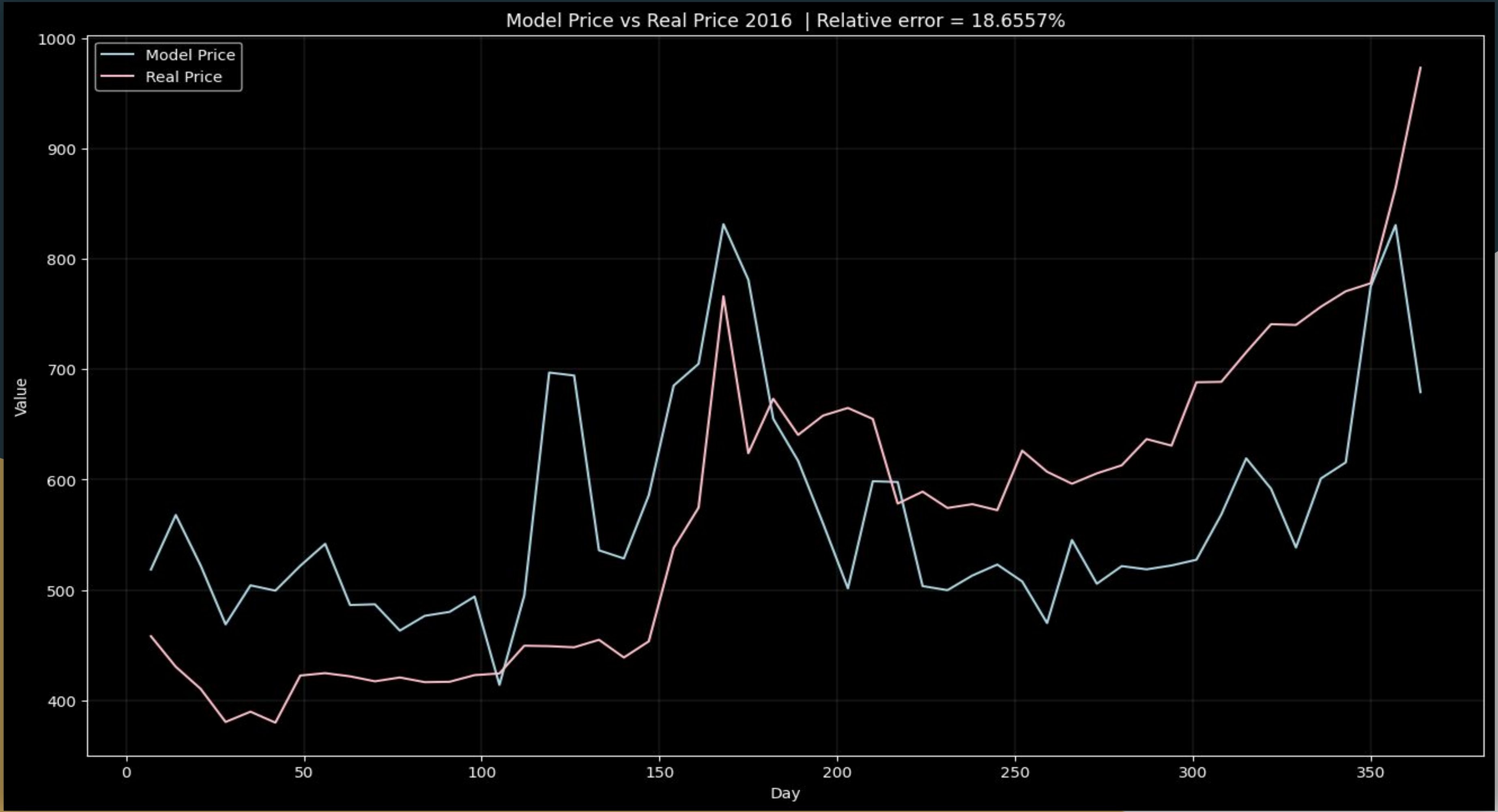
$$\text{Hit ratio} = \frac{1}{n}\sum_{1}^{n} D_i, \quad i = 1, 2, \ldots, n,$$

where $\sum_1^n D_i$ denotes the number of correct forecasts of the bitcoin price direction

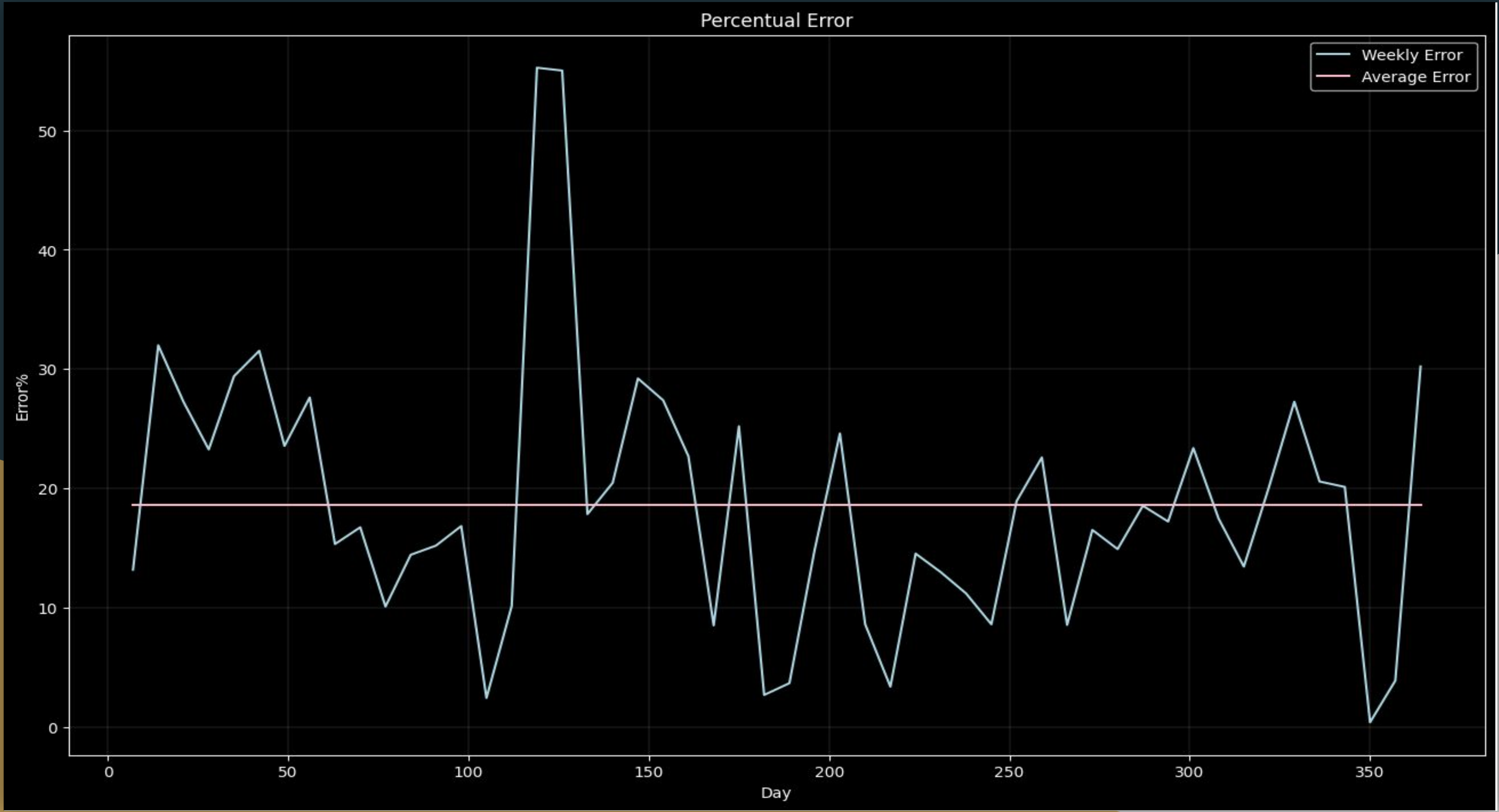
$$D_i = \begin{cases} 1, & (P_{real}(i+1) - P_{real}(i))\left(P_{forecast}(i+1) - P_{real}(i)\right) > 0, \\ 0, & \text{otherwise,} \end{cases}$$

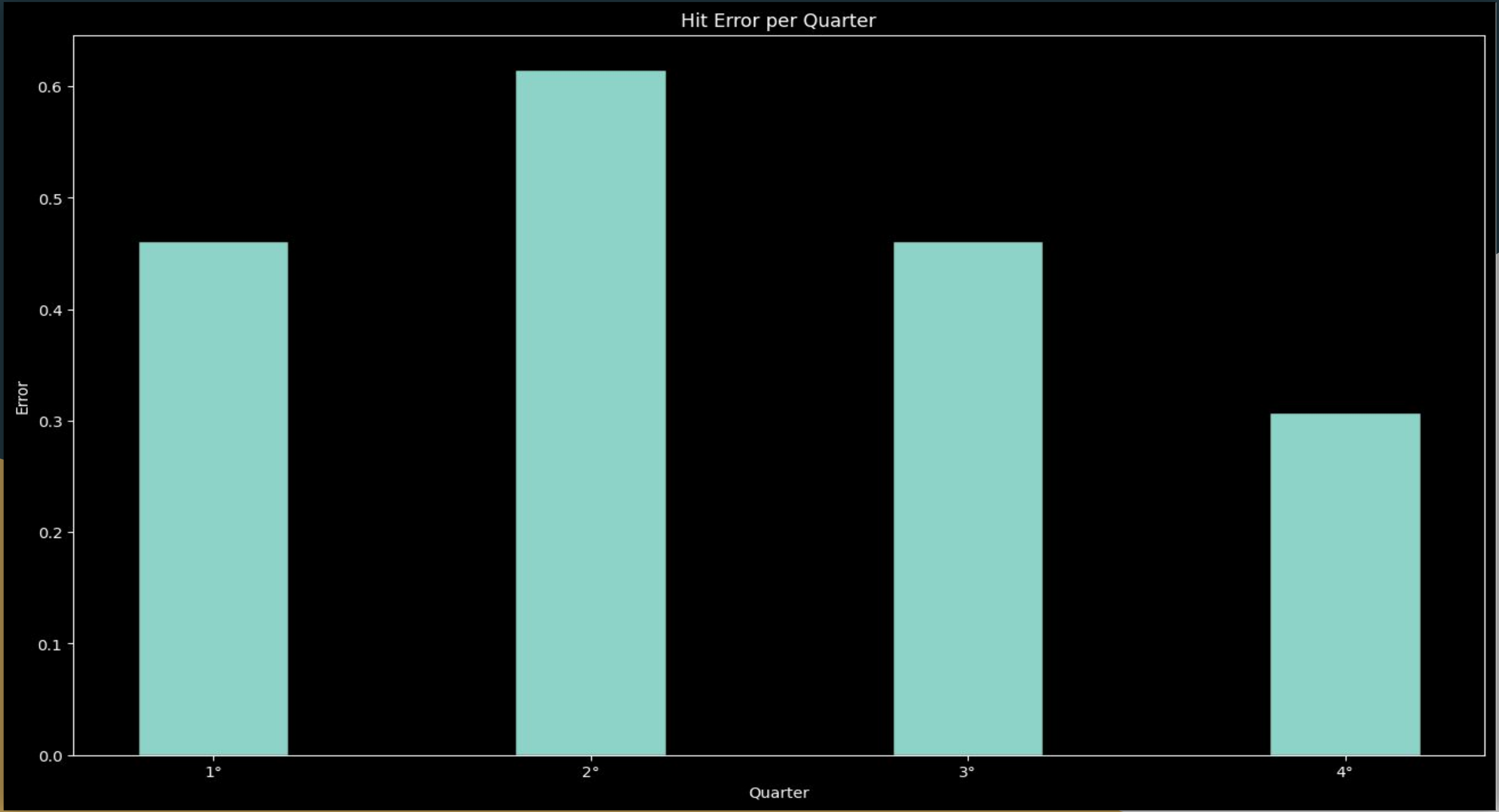

Hit Error per Quarter
Error
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1°
2°
3°
4°
Quarter

¿Los parámetros ya encontrados sirven para otros años?



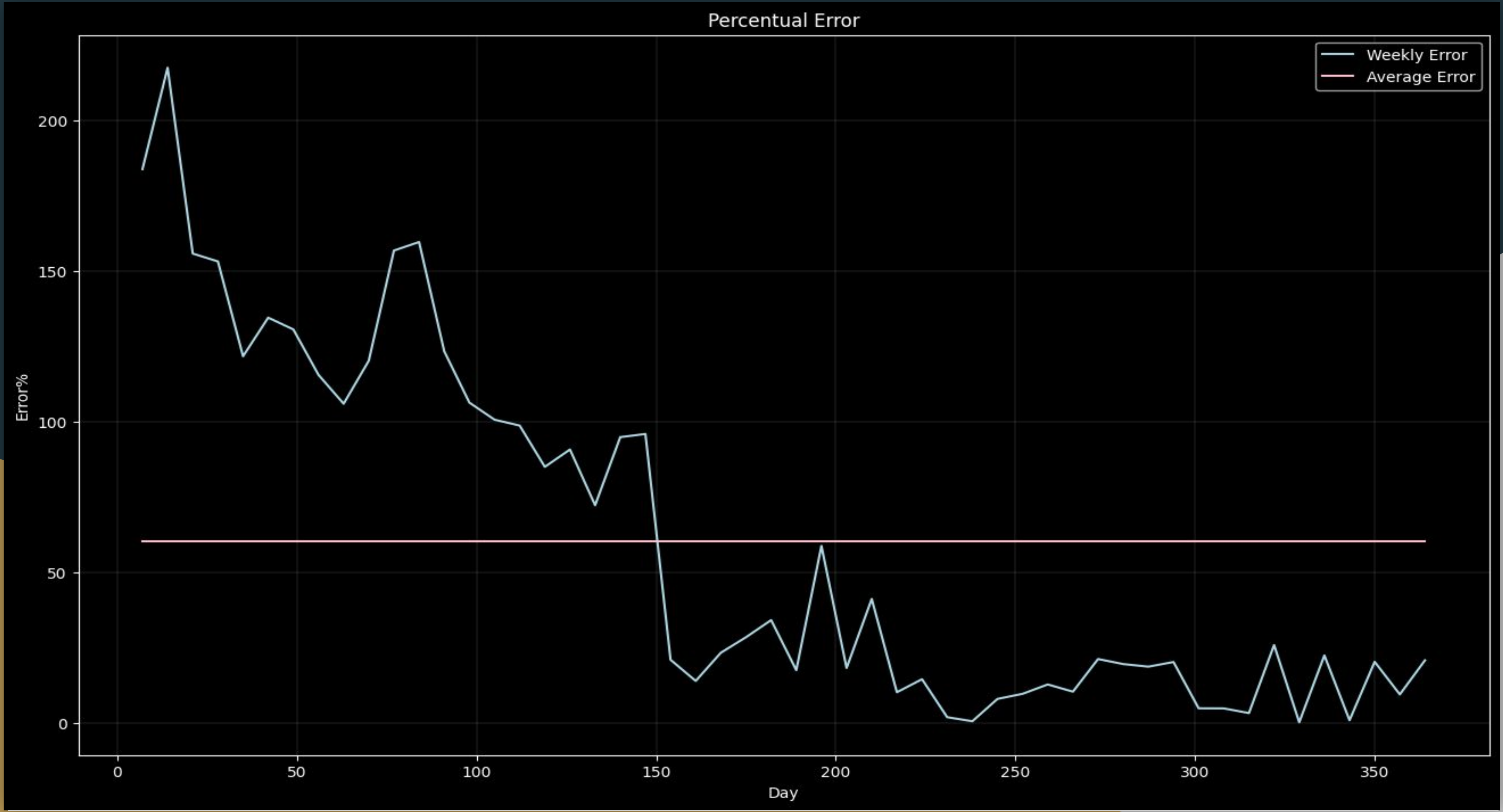
Model Price vs Real Price 2016 | Relative error = 18.6557%
Model Price
Real Price
Value
Day
1000
900
800
700
600
500
400
0
50
100
150
200
250
300
350

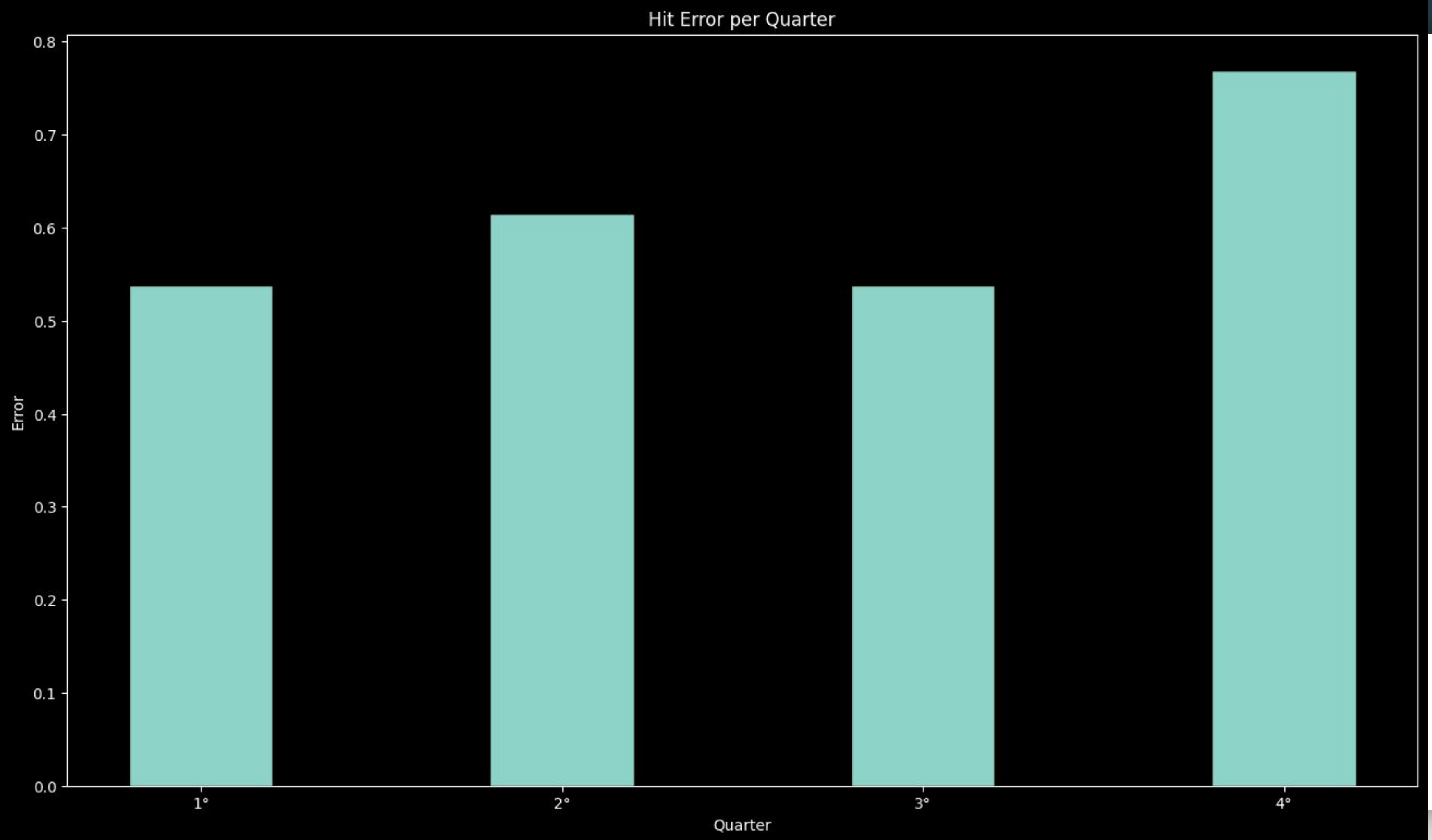

Percentual Error
Weekly Error
Average Error
Error%
Day
0
10
20
30
40
50
0
50
100
150
200
250
300
350



Hit Error per Quarter
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Error
1°
2°
3°
4°
Quarter


Model Price vs Real Price 2017 | Relative error = 60.46%
Model Price
Real Price
Value
Day
20000
17500
15000
12500
10000
7500
5000
2500
0
0
50
100
150
200
250
300
350


Percentual Error
Weekly Error
Average Error
Error%
200
150
100
50
0
0
50
100
150
200
250
300
350
Day


Hit Error per Quarter
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Error
1°
2°
3°
4°
Quarter

Dificultades

-La mayoría del trabajo se basaba en conceptos y métodos que desconociamos de antemano.

-Conseguir , interpretar y limpiar los datos.

-El paper no es claro en cómo calcula u obtiene ciertas cosas, como por ejemplo:

- -No usar todos los chainlets
- -En qué manera se hace embedding
- -Qué método numérico se usa para resolver la ecuación diferencial
- -Que intervalos de tiempo utilizar para resolver la ecuación diferencial

Fin

¿Preguntas?

Notebook