Facultad de Informática
Universidad Nacional de La Plata

Bases de datos de series temporales y métodos estadísticos para la predicción de rendimientos de ETH

Trabajo final para obtener el título de

Especialista en Inteligencia de Datos
orientada a Big Data

Lic. Alejo Hernandez

Director: Prof. Dr. Aurelio F. Bariviera
Co-Director: Prof. Dr. Ricardo Di Pasquale

Índice general

Capítulo 1

Introducción

Las bases de datos forman parte de nuestra vida diaria, aunque muchos de nosotros probablemente no seamos conscientes de este hecho. Esto es fácil de entender si tenemos en cuenta que la mayoría de las transacciones electrónicas nos ponen en contacto con una de ellas y que este tipo de transaciones se han vuelto ubicuas, dado el nivel de penetración que poseen los telefonos celulares [1] y el crecimiento exponencial que han experimentado los dispositivos conectados a internet (IoT, IoE) [2].

Este uso intensivo de las bases de datos, combinado con la vastedad de escenarios en los cuales se ven involucradas, ha propiciado un proceso evolutivo que comenzó en la década del '60 y continúa hasta nuestros días [3]. En este camino, varios tipos de bases de datos fueron creadas: relacionales, no-SQL, de almacenamiento en memoria, de grafos, geoespaciales y por último, las específicas para series temporales (BDSTs) [4].

Las primeras discusiones relevantes en relación con las BDSTs comenzaron a medidados de la década de 1990 [5], pero no fue hasta principios de la década del 2000 [6] que se dieron las condiciones propicias para gatillar su nacimiento. En aquel entonces transitábamos los primeros pasos del proceso de transformación hacia la Web 2.0 y en ese camino se crearon las primeras plataformas relacionadas con e-commerce [7], home banking [8] y online trading [9]. Conforme el tiempo fue pasando, el uso de las mismas aumentó al punto de que se volvieron omnipresentes y esto trajo aparejada la generación de grandes cantidades de datos temporales que debían ser guardados de forma consistente y consultados en tiempo casi-real. Fueron estos desafíos técnicos y la necesidad de resolverlos lo que impulsó en gran medida la creación y el desarrollo de las BDSTs.

Por otra parte, durante los últimos diez años la adopción de las criptomonedas como instrumento de inversión y/o de especulación fomentó el desarrollo de un mercado de negociación deslocalizado, de muy alta frecuencia y sin interrupciones significativas que, al día de hoy, se presenta como una alternativa viable para el manejo de divisas y el pago de bienes y servicios. Si bien en términos académicos, la mayoría de los trabajos publicados utilizan datos con frecuencias diarias [10], esta tendencia está cambiando, puesto que los investigadores han visto la importancia de extraer información de datos en más alta frecuencia para capturar el comportamiento de este mercado altamente dinámico. Así Zhang et al. [11] calculan el exponente de Hurst como estimador de la memoria de largo plazo a partir de los precios registrados cada hora de Bitcoin, Ethereum, Ripple y Litecoin, Chu et al. [12] usan datos por hora para definir estrategias de momento (impulso) que exploten la tendencia de los rendimientos horarios de varias criptomonedas, Yarovaya et al. [13] utilizan datos cada 5 minutos para estudiar las relaciones volumen-rendimiento en las 30 principales criptomonedas, Alonso-Monsalve et al. utilizan redes neuronales convolucionales sobre datos basados en 18 indicadores técnicos con resolución de un minuto sobre todo un año para predecir las tasas de intercambio de criptomonedas [14] y Petukhina et al. estudia el impacto del trading algorítmico de alta frecuencia sobre el mercado de criptomonedas europeo [15].

Estas características particulares que presenta el mercado de criptomonedas se traducen en un flujo de datos continuo y de alta frecuencia, modulado mayoritariamente por operaciones de trading automatizadas. Esto evidencia la necesidad de contar, por un lado, con un sistema de almacenamiento y procesamiento de datos acorde con el volumen y velocidad de generación involucrados y por otro, con una serie de algoritmos robustos que permitan predecir los retornos de los activos de forma precisa y confiable.

A fin de explorar más en profundidad estos aspectos, este trabajo se propone implementar un pipeline de datos soportado por una BDST a fin de realizar predicciones de rendimento de activos usando métodos estadísticos. Más concretamente, se implementa la BDST InfluxDB para predecir los retornos de Ethereum (ETH), la segunda criptomoneda en términos de capitalización de mercado.

Desde el punto de vista del manejo de datos, se decidió trabajar con la BDST InfluxDB puesto que la misma cuenta con una suite de herramientas muy completa para gestionar los procesos de captura, almacenamiento, consulta y visualización de la información. Por otra parte, en lo relativo al

modelado estadístico de los datos, se utilizó el paquete Statsmodels de Python que permite realizar la predicción de los retornos aplicando un modelo ARMA con intervalos de confianza corregidos mediante un modelo GARCH.

La estructuración del trabajo es como sigue: En el capítulo 2, definimos las series temporales, analizamos algunos casos de uso y repasamos algunos conceptos estadísticos asociados a las mismas. En el capítulo 3 introducimos los lineamientos generales que presentan los modelos de pronóstico futuro que aplicaremos para estimar los retornos de Ethereum. En el capítulo 4 mencionamos las propiedades generales de las BDSTs y analizamos sus principales casos de uso. En el capítulo 5 describimos las herramientas técnicas empleadas durante la realización de este trabajo. Respecto de InfluxDB, lo relativo a los procesos de modelado, escritura, consulta y visualización de las series temporales, se consigna en la sección 5.1, mientras que el detalle de los principales paquetes de Python utilizados para preprocesar los datos y aplicar los modelos de tipo ARMA y GARCH sobre los mismos se detallan en la sección 5.2. En el capítulo 6 se ilustra el proceso seguido para obtener la predicción de retornos para Ethereum. Comenzando con la captura y almacenamiento de las observaciones financieras en InfluxDB, su posterior consulta y preprocesamiento, el análisis exploratorio, modelado y finalmente los resultados de la predicción. Finalmente, las conclusiones del trabajo se detallan en el capítulo 7.

Capítulo 2

Series temporales

2.1. Definición

Desde una perspectiva computacional, una serie temporal puede definirse como un conjunto de observaciones indexadas usando una timestamp[1]. Sin embargo, esta difinición debe ser extendida para aplicar consideraciones de tipo estadístico. Diremos entonces que una serie temporal es una secuencia de variables aleatorias x_1, x_2, ..., x_t, indexadas de acuerdo al momento en que fueron medidas. Este tipo de colecciones de variables aleatorias x_t se conocen como procesos estocásticos y los valores observados de tales variables son realizaciones de dicho proceso.

Una serie temporal puede construirse tomando observaciones a intervalos de tiempo regulares o no. En el primer caso, se las denomina *métricas* y en el segundo *eventos*.

En general, las observaciones que constituyen una serie temporal suelen numéricas, pero esto no es excluyente.

2.2. Casos de uso

2.2.1. Análisis exploratorio

El análisis exploratorio de una serie temporal consiste básicamente en representar la misma de alguna forma que permita interpretar el cambio de

[1]Una timestamp es un registro digital del tiempo de ocurrencia de un evento en particular.

la variable estudiada con el tiempo. Quizá la forma más común en la que suelen realizarse este tipo de análisis es por observación directa de gráficos, en los cuales el eje de abcisas contiene el tiempo, mientras que el de ordenadas contiene el valor medido de la variable de interés. Una representación de este tipo es la que se observa en la Fig. 2.1.

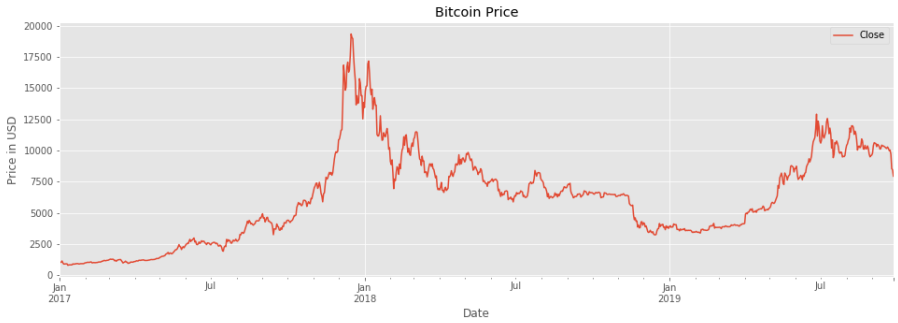

Figura 2.1: Precio de cierre en USD para el Bitcoin durante el período de tiempo comprendido entre Enero de 2017 y Octubre de 2019.

El análisis de series temporales generalmente requiere de un gran número de datos para poder asegurar la consistencia y confiabilidad del mismo. Un conjunto de datos lo suficientemente grande, garantiza la representatividad de la muestra y disminuye el riesgo de generar conclusiones sesgadas por efectos de observaciones extrañas, efectos estacionales, etc.

2.2.2. Pronóstico futuro

Tratar de inferir comportamientos futuros sobre la base de observaciones pasadas es algo que la humanidad realiza desde el momento en que desarrolló el razonamiento abstracto. En el presente, este tipo de inferencias se hacen aplicando metodologias y técnicas muy precisas sobre series temporales. Estas pueden dividirse en dos grandes grupos: técnicas estadísticas o de Machine Learning (ML). En este trabajo nos enfocaremos sólamente en el uso de las primeras, difiriendo el empleo de las segundas para un trabajo posterior.

2.3. Series temporales como procesos estocásticos

Puesto que hemos definido formalmente una serie temporal como una colección de variables aleatorias, resulta indispensable repasar algunos de los conceptos estadísticos principales que atañen a las mismas.

Sea R^k un espacio euclídeo k-dimensional. Un punto en R^k se nota como $\boldsymbol{x} \in \mathrm{R}^k$. Consideremos dos vectores aleatorios $\boldsymbol{X} = (X_1, \ldots, X_k)$ e $\boldsymbol{Y} = (Y_1, \ldots, Y_q)$ y sea $\mathrm{P}\left(\boldsymbol{X} \leq \boldsymbol{x}, \boldsymbol{Y} \leq \boldsymbol{y}; \boldsymbol{\theta}\right)$ la probabilidad de que $\boldsymbol{X}$ este en el subespacio $\mathrm{A} \subset \mathrm{R}^k$ e $\boldsymbol{Y}$ este en el subespacio $\mathrm{B} \subset \mathrm{R}^q$.

2.3.1. Distribución de probabilidad conjunta

La función

$$\mathrm{F}_{X,Y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right) = \mathrm{P}\left(\boldsymbol{X} \leq \boldsymbol{x}, \boldsymbol{Y} \leq \boldsymbol{y}; \boldsymbol{\theta}\right) , \tag{2.1}$$

donde $\boldsymbol{x} \in \mathrm{R}^k$ e $\boldsymbol{y} \in \mathrm{R}^q$ es una distribución de probabilidad conjunta para $\boldsymbol{X}$ e $\boldsymbol{Y}$ con parámetro $\boldsymbol{\theta}$ que caracteriza el comportamiento de ambas. Si la función de densidad de probabilidad conjunta $\mathrm{f}_{x,y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right)$ existe, entonces

$$\mathrm{F}_{X,Y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right) = \int_{-\infty}^{\boldsymbol{x}} \int_{-\infty}^{\boldsymbol{y}} \mathrm{f}_{x,y}\left(\boldsymbol{w}, \boldsymbol{z}; \boldsymbol{\theta}\right) d\boldsymbol{w} d\boldsymbol{z} . \tag{2.2}$$

2.3.2. Distribución de probabilidad marginal

La distribución de probabilidad marginal para $\boldsymbol{X}$ se define mediante

$$\mathrm{F}_{X}\left(\boldsymbol{x}; \boldsymbol{\theta}\right) = \int_{-\infty}^{\boldsymbol{x}} \int_{-\infty}^{+\infty} \mathrm{f}_{x,y}\left(\boldsymbol{w}, \boldsymbol{z}; \boldsymbol{\theta}\right) d\boldsymbol{w} d\boldsymbol{z} , \tag{2.3}$$

es decir, se ha integrado toda contribución proveniente de $\boldsymbol{Y}$. Análogamente se define la distribución de probabilidad marginal para $\boldsymbol{Y}$. En el caso de que $k = 1$, la distribución de probabilidad marginal se convierte en una distribución de probabilidad acumulada (CDF)

$$\mathrm{F}_{X}\left(x; \boldsymbol{\theta}\right) = \mathrm{P}\left(X \leq x; \boldsymbol{\theta}\right) . \tag{2.4}$$

Dado un valor de probabilidad p, el menor numero real x_p que satisface $p \leq \mathrm{F}_X\left(x_p\right)$ se conoce como el percentilo p de la variable aleatoria X. Esta

definición es la que se utiliza para computar el p-value de los estadísticos de prueba.

2.3.3. Distribución de probabilidad condicional

La distribución de probabilidad condicional de $\boldsymbol{X}$ dado que $\boldsymbol{Y} \leq \boldsymbol{y}$ esta dada por

$$\mathrm{F}_{X|Y\leq y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right) = \frac{\mathrm{P}\left(\boldsymbol{X} \leq \boldsymbol{x}, \boldsymbol{Y} \leq \boldsymbol{y}; \boldsymbol{\theta}\right)}{\mathrm{P}\left(\boldsymbol{Y} \leq \boldsymbol{y}; \boldsymbol{\theta}\right)}\,, \tag{2.5}$$

Si las funciones de densidad de probabilidad existen, podemos escribir

$$\mathrm{f}_{x|y}\left(\boldsymbol{x}; \boldsymbol{\theta}\right) = \frac{\mathrm{f}_{x,y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right)}{\mathrm{f}_{y}\left(\boldsymbol{y}; \boldsymbol{\theta}\right)}\,, \tag{2.6}$$

donde

$$\mathrm{f}_{y}\left(\boldsymbol{y}; \boldsymbol{\theta}\right) = \int_{-\infty}^{-\infty} \mathrm{f}_{x,y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right) d\boldsymbol{x}\,. \tag{2.7}$$

La ec. (2.6) puede reordenarse para adoptar una forma que recuerda al teorema de Bayes y que suele usarse en las estimaciones de máxima verosimilitud,

$$\mathrm{f}_{x,y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right) = \mathrm{f}_{x|y}\left(\boldsymbol{x}; \boldsymbol{\theta}\right) \times \mathrm{f}_{y}\left(\boldsymbol{y}; \boldsymbol{\theta}\right)\,. \tag{2.8}$$

Por último, notemos que si $\boldsymbol{X}$ e $\boldsymbol{Y}$ son independientes, entonces

$$\mathrm{f}_{x,y}\left(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}\right) = \mathrm{f}_{x}\left(\boldsymbol{x}; \boldsymbol{\theta}\right) \times \mathrm{f}_{y}\left(\boldsymbol{y}; \boldsymbol{\theta}\right)\,. \tag{2.9}$$

2.3.4. Momentos de una variable aleatoria

El l-ésimo momento de una variable aleatoria continua se define como

$$m_l' = \mathrm{E}(X^l) = \int_{-\infty}^{\infty} x^l f(x)\, dx\,, \tag{2.10}$$

donde E se conoce como el valor de expectación y $f(x)$ es la distribución densidad de probabilidad de X.

Media y momentos centrales El primer momento ($l = 1$) se conoce como *media* o *valor esperado* de la variable aleatoria X y da una medida de la ubicación del centro de la distribución. En general, escribimos

$$m_1' = \mathrm{E}(X) = \mu_x \,, \tag{2.11}$$

y calculamos los momentos centrales de la distribución como

$$m_l = \mathrm{E}((X - \mu_x)^l) = \int_{-\infty}^{\infty} (x - \mu_x)^l \, f(x) \, dx \,, \tag{2.12}$$

si la integral anterior existe.

Dada una una muestra aleatoria con n observaciones, $x_1, \ldots, x_n$, el estimador muestral para la *media* se calcula como

$$\hat{\mu}_x = \frac{1}{n} \sum_{t=1}^{n} x_t \,. \tag{2.13}$$

Varianza y desvío standard El segundo momento central,

$$m_2 = \mathrm{E}((X - \mu_x)^2) = \sigma_x^2 \,, \tag{2.14}$$

se denomina comunmente como *varianza* y su raíz cuadrada es conocida como *desviación standard*. Esta última da una medida de la dispersión de los valores en torno a la media en unidades coherentes.

Dada una una muestra aleatoria con n observaciones, $x_1, \ldots, x_n$, el estimador muestral para la *varianza* se calcula como

$$\hat{\sigma}_x^2 = \frac{1}{n-1} \sum_{t=1}^{n} (x_t - \hat{\mu}_x)^2 \,. \tag{2.15}$$

Asimetría (Skweness) El tercer momento central se utiliza como un parámetro que permite estimar cuán simétrica resulta la distribución respecto de la media. Este tercer momento, denominado *skwness* se define como

$$\mathrm{S}(x) = \mathrm{E}\left(\frac{(\mathrm{X} - \mu_x)^3}{\sigma_x^3} \right) \,. \tag{2.16}$$

Dada una una muestra aleatoria con n observaciones, $x_1, \ldots, x_n$, el estimador muestral para la *skweness* se calcula como

$$\hat{\mathrm{S}}(x) = \frac{1}{n-1} \sum_{t=1}^{n} \frac{(x_t - \hat{\mu}_x)^3}{\hat{\sigma}_x^3} \,. \tag{2.17}$$

Kurtosis El cuarto momento muestral da una idea del comportamiento de las colas de la distribución[2]. Este cuarto momento, denominado *kurtosis* se define como

$$\mathrm{K}(x) = \mathrm{E}\left(\frac{(\mathrm{X} - \mu_x)^4}{\sigma_x^4} \right) \,. \tag{2.18}$$

Dada una una muestra aleatoria con n observaciones, $x_1, \ldots, x_n$, el estimador muestral para la *kurtosis* se calcula como

$$\hat{\mathrm{K}}(x) = \frac{1}{n-1} \sum_{t=1}^{n} \frac{(x_t - \hat{\mu}_x)^4}{\hat{\sigma}_x^4} \,. \tag{2.19}$$

2.3.5. Covarianza y autocovarianza

Covarianza La *covarianza* es una medida de la variabilidad conjunta de dos variables aleatorias a orden lineal. Si ambas variables estan linealmente relacionadas y tienen un comportamiento similar, la varianza será positiva, si no estan linealmente relacionadas la covarianza será cercana a 0 y si están linealmente relacionadas pero tienen un comportamiento opuesto, la covarianza será negativa.

Más precisamente, definimos la covarianza entre las variables aleatorias X e Y como

$$Cov(X, Y) = \mathrm{E}\left[(X - \mu_x)\,(Y - \mu_y) \right] \,, \tag{2.20}$$

donde μ_X y μ_Y son los valores medios de X e Y, respectivamente. Es frecuente que la covarianza también se denomine utilizando σ_{XY} o bien $\sigma(X, Y)$. Nótese que, por definición

$$Cov(X, X) = \sigma_X^2 \,, \tag{2.21}$$

[2]Entendemos por colas de la distribución al comportamiento asintótico de la misma con la variable de interés.

y que

$$Cov(X,Y) = Cov(Y,X)\,, \tag{2.22}$$

es decir, la covarianza resulta simétrica frente al intercambio X por Y.

Para una muestra aleatoria $\{(x_t, y_t)\}_{t=1}^{n}$ de las variables aleatorias X e Y el estimador muestral de la covarianza se define como

$$\hat{\sigma}_{xy} = \frac{1}{n-1}\sum_{t=1}^{n}\left(x_t - \hat{\mu}_x\right)\left(y_t - \hat{\mu}_y\right)\,, \tag{2.23}$$

donde $\hat{\mu}_x$ y $\hat{\mu}_y$ son los estimadores muestrales de los valores medios de las variables aleatorias X e Y, respectivamente.

Autocovarianza Dada una serie temporal $\{x_t\}$, definimos la *autocovarianza* de rezago-l como

$$\gamma_l = Cov\left(x_t, x_{t-l}\right)\,. \tag{2.24}$$

La misma tiene dos propiedades importantes, a saber:

- $\gamma_0 = \sigma_{x_t}^2\,,$
- $\gamma_{-l} = \gamma_l\,,$

donde esta última propiedad se deriva teniendo en cuenta que

$$\gamma_{-l} = Cov\left(x_t, x_{t-(-l)}\right) = Cov\left(x_{t+l}, x_t\right) = Cov\left(x_{t'}, x_{t'-l}\right) = \gamma_l\,. \tag{2.25}$$

Aquí, hemos usado que la covarianza es simétrica frente al intercambio de sus argumentos y hemos redefinido el índice temporal de forma tal que $t' = t + l$.

2.3.6. Correlación y autocorrelación (ACF)

Correlación Si bien el signo de la covarianza permite estimar la tendencia en la relación lineal de dos variables, su magnitud no permite establecer una medida del grado de interdependencia lineal entre ambas. Para resolver este

problema se introduce una versión normalizada de la covarianza, llamada *coeficiente de correlación*. La misma se define como

$$\rho_{xy} = \frac{Cov(X,Y)}{\sqrt{Cov(X,X)\,Cov(Y,Y)}} = \frac{\mathrm{E}\left[(X-\mu_x)\,(Y-\mu_y)\right]}{\sqrt{\mathrm{E}\left[(X-\mu_x)^2\right]\;\mathrm{E}\left[(Y-\mu_y)^2\right]}}\,. \quad (2.26)$$

y verifica

- $-1 < \rho_{xy} < 1$,
- Si X e Y no están linealmente relacionadas $\Rightarrow \rho_{xy} = 0$,
- $\rho_{xy} = \rho_{yx}$.

Para una muestra aleatoria $\{(x_t, y_t)\}_{t=1}^n$ de las variables aleatorias X e Y el estimador muestral del coeficiente de correlación resulta ser

$$\hat{\rho}_{xy} = \frac{\sum_{t=1}^n (x_t - \hat{\mu}_x)(y_t - \hat{\mu}_y)}{\sqrt{\sum_{t=1}^n (x_t - \hat{\mu}_x)^2}\sqrt{\sum_{t=1}^n (y_t - \hat{\mu}_y)^2}}\,, \quad (2.27)$$

siendo nuevamente $\hat{\mu}_x$ y $\hat{\mu}_y$ los estimadores muestrales de los valores medios de X e Y, respectivamente.

Autocorrelación (ACF) La *función de autocorrelación (ACF)* mide la correlación entre observaciones separadas por una cierto intervalo temporal. Dada una observación a tiempo t y otra a tiempo $t-l$, definimos la autocorrelación de rezago-l mediante

$$\rho_l = \frac{Cov(x_t, x_{t-l})}{\sqrt{Cov(x_t, x_t)\,Cov(x_{t-l}, x_{t-l})}}\,. \quad (2.28)$$

Para fijar ideas, analicemos la función de correlación de rezago-1 para una muestra de con n observaciones $\{x_1, x_2, \ldots, x_{n-1}, x_n\}$. Construyendo la secuencia de $n-1$ pares (x_1, x_2), (x_2, x_3), $\ldots$, (x_{n-1}, x_n) y admitiendo que lconsideramos que cada elemento da primera componente de cada par se correspode con una cierta variable aleatoria y lo mismo con la segunda, podemos calcular el estimador muestral del coeficiente de correlación de rezago-1 mediante

$$\hat{\rho}_1 = \frac{\sum_{i=1}^{n-1} (x_i - \hat{\mu}_{(1)})(x_{i+1} - \hat{\mu}_{(2)})}{\sqrt{\sum_{i=1}^{n-1} (x_i - \hat{\mu}_{(1)})^2}\sqrt{\sum_{i=1}^{n-1} (x_{i+1} - \hat{\mu}_{(2)})^2}}\,, \quad (2.29)$$

donde

$$\hat{\mu}_{(1)} = \frac{\sum_{i=1}^{n-1} x_i}{n-1}\,, \qquad \hat{\mu}_{(2)} = \frac{\sum_{i=2}^{n} x_i}{n-1}\,, \tag{2.30}$$

son los estimadores muestrales de la media usando las primeras y últimas $n-1$ observaciones, respectivamente.

Debido a que la definición formal de autocorrelación expresada por la ec. (2.29) es en general complicada de calcular y teniendo en cuenta que $\hat{\mu}_{(1)} \sim \hat{\mu}_{(2)} \sim \hat{\mu}$, la misma suele aproximarse mediante

$$\hat{\rho}_1 = \frac{\sum_{i=1}^{n-1}(x_i - \hat{\mu})(x_{i+1} - \hat{\mu})}{\frac{n-1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2}\,, \tag{2.31}$$

siendo

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n}\,. \tag{2.32}$$

Por último, si tomamos el límite de n grande, el factor $\frac{n-1}{n}$ es cercano a uno[3] y vale

$$\hat{\rho}_1 = \frac{\sum_{i=1}^{n-1}(x_i - \hat{\mu})(x_{i+1} - \hat{\mu})}{\sum_{i=1}^{n}(x_i - \hat{\mu})^2}\,. \tag{2.33}$$

Esta es la expresión que suele emplearse para estimar el coeficiente de autocorrelación de rezago-1. Análogamente, calculamos el coeficiente de correlación de rezago-k como

$$\hat{\rho}_k = \frac{\sum_{i=1}^{n-k}(x_i - \hat{\mu})(x_{i+1} - \hat{\mu})}{\sum_{i=1}^{n}(x_i - \hat{\mu})^2}\,. \tag{2.34}$$

En la práctica, los coeficientes de autocorrelación se calculan usando la autocovarianza

$$\gamma_k = \frac{1}{n}\sum_{t=1}^{n-k}\left(x_t - \hat{\mu}_x\right)\left(y_t - \hat{\mu}_y\right)\,, \tag{2.35}$$

[3] A fin de tener una idea del valor de la aproximación, para n=25 se comete un error del 4 % y para $n = 100$ del 1 %.

y haciendo

$$\hat{\rho}_k = \frac{\gamma_k}{\gamma_0}\,, \tag{2.36}$$

para $k = 0, \ldots, m, \quad m < n$. En términos prácticos, no tiene mucho sentido calcular $\hat{\rho}_k$ para valores de $k > \frac{n}{4}$

2.3.7. Correlación parcial y Autocorrelación parcial (PACF)

Correlación parcial En general, la *correlación parcial* es una correlación condicional, es decir, podemos pensarla como la correlación existente entre dos variables habiendo tenido en cuenta también la contribución lineal generada por el resto de las variables bajo estudio.

Para entender esto, supongamos que estamos intentando estimar la correlación parcial entre una variable target y y un cierto predictor x_1 en contexto en el cual están presentes también los predictores x_2 y x_3. En este caso, escribimos

$$\tilde{\rho}_{yx_2} = \frac{Cov(Y, X_1|X_2, X_3)}{\sqrt{Cov(Y|X_2, X_3, Y|X_2, X_3)\, Cov(X_1|X_2, X_3, X_1|X_2, X_3)}}\,. \tag{2.37}$$

Si pensamos en términos de regresiones lineales, la correlación parcial entre y y x_2 se puede estimar estableciendo la correlación entre los errores de las siguientes expansiones lineales

$$\begin{aligned} y &= \beta_0 + \beta_1 x_1 + \epsilon_1\,, \\ x_1 &= \alpha_0 + \alpha_1 x_2 + \alpha_2 x_3 + \epsilon_2\,. \end{aligned} \tag{2.38}$$

Autocorrelación parcial La *Autocorrelación parcial*

Para una serie temporal, la PACF para una observación a tiempo t de rezago-l se define como la correlación existente entre x_t y x_{t-l}, habiendo tenido en cuenta los efectos de las observaciones intermedias $x_{t-1}, x_{t-2}, \ldots, x_{t-l+1}$. De esta forma, la PACF de rezago-1 coincide con la ACF rezago-1, la PACF de rezago-2 se calcula como

$$\tilde{\gamma}_2 = \frac{Cov(x_t, x_{t-2}|x_{t-1})}{\sqrt{Cov(x_t|x_{t-1}, x_t|x_{t-1})\, Cov(x_{t-2}|x_{t-1}, x_{t-2}|x_{t-1})}}\,, \tag{2.39}$$

y así siguiendo.

2.3.8. Estacionariedad

Decimos que una serie temporal $\{x_t\}$ es *estacionaria* si se verifica que la distribución de probabilidad conjunta $(x_{t_1}, \ldots, x_{t_k})$ es invariante frente a la inversión temporal. La estacionariedad así definida es difícil de observar en la práctica, por lo que usualmente esta condición suele relajarse. En este sentido, nos referimos a una serie como *débilmente estacionaria* si la misma verifica

- $\mathrm{E}(x_t) = \mu\,,\ \forall t$
- $Cov\,(x_t, x_{t-l}) = \gamma_l\,,\ \forall t$

es decir, oscila en torno a una media constante y la covariancia entre los distintos elementos de la serie γ_l depende sólamente del lag l y no del tiempo.

2.4. Series temporales financieras

Las series temporales financieras están asociadas con la valuación de activos en el tiempo, particularmente los rendimientos, como veremos más adelante.

A la hora de realizar un pronóstico futuro sobre una serie temporal financiera, se requieren conocer una serie de conceptos y herramientas específicas. En ese sentido, resulta indispensable repasar no sólo las características básicas de las mismas, sino también los modelos estadísticos involucrados.

En lo que resta de esta sección, se revisaran los conceptos fundamentales para comprender este tipo de series temporales, mientras que el análisis de los modelos se hará en la sección siguiente.

2.4.1. Rendimientos

La variable de mayor interes en el contexto de las finanzas suele ser el retorno (ganancia o pérdida) que uno obtiene al comercializar un activo financiero en un dado mercado. Hay dos razones principales por las que esto es así. En primer lugar, los rendimientos dan una idea clara de la oportunidad de inversión [Campbell, Lo and MacKinlay (1997)]. En segundo, como veremos

en breve, los rendimientos tienen por su definición, propiedades estadísticas que los vuelven más fáciles de manejar.

Si bien existen distintas definiciones de rendimientos, en este trabajo seguiremos las convenciones utilizadas en libro de Tsay[16].

Sea P_t el precio del activo de interés a tiempo t. Definimos el **retorno bruto simple** al cabo de haber retenido dicho activo por un periodo de tiempo entre $t-1$ y t como

$$1 + R_t = \frac{P_t}{P_{t-1}}\,, \tag{2.40}$$

siendo R_t lo que se conoce como **retorno simple**. Extendiendo esta definición para el caso de k períodos entre $t-k$ y t tenemos que

$$\begin{aligned} 1 + R_t[k] &= \frac{P_t}{P_{t-k}} = \frac{P_t}{P_{t-1}} \times \frac{P_{t-1}}{P_{t-2}} \times \ldots \times \frac{P_{t-(k+1)}}{P_{t-k}}\,, \\ &= (1+R_t)\,(1+R_{t-1})\ldots(1+R_{t-k+1})\,, \\ &= \prod_{j=0}^{k-1} (1+R_{t-j})\,. \end{aligned} \tag{2.41}$$

Vewmos entonces que el retorno bruto simple para k períodos no es más que el producto de los k rendimientos diarios y se lo conoce también como retorno compuesto.

Finalmente analicemos los rendimientos compuestos continuamente. Se define al retorno compuesto continuamente o retorno logarítmico mediante

$$r_t = \log(1+R_t) = \log\left(\frac{P_t}{P_{t-1}}\right) = p_t - p_{t-1}\,, \tag{2.42}$$

donde $p_t = \log(P_t)$. Teniendo en cuenta esta definición, es fácil ver entonces que si analizamos el comportamiento al cabo de k periodos tendremos

$$\begin{aligned} r_t[k] &= \log(1+R_t[k]) = \log\left((1+R_t)\,(1+R_{t-1})\ldots(1+R_{t-k+1})\right)\,, \\ &= r_t + r_{t-1} + \ldots + r_{t-k+1}\,. \end{aligned} \tag{2.43}$$

Esto muestra que el retorno compuesto continuamente al cabo de k períodos no es más que la suma de los rendimientos logarítmicos de los k períodos anteriores.

2.4.2. Volatilidad

En el contexto de las finanzas, se define la volatilidad como el desvío standard de los rendimientos o rendimientos logarítmicos.

La volatilidad presenta varias propiedades, a saber:

- No es directamente observable a nivel diario, puesto que sólo se tiene una observación por día de trading.
- Se presenta clusterizada, es decir, existen períodos de alta volatilidad y períodos de baja volatilidad.
- Evoluciona de forma continua en el tiempo, es decir, no hay cambios abruptos en la misma.
- No es divergente, sino que oscila dentro de un rango acotado. Esto equivale a decir que la volatilidad usualmente es estacionaria.
- Reacciona de forma diferente frente a una gran suba o una gran baja de un activo. Este efecto se denomina "efecto de aprovechamiento".

Estas propiedades juegan un rol fundamental en la formulación de los modelos de volatilidad, como veremos más adelante.

Capítulo 3

Modelos para pronóstico futuro de series temporales

En esta sección haremos un repaso muy sucinto de los modelos que se utilizan para realizar el pronóstico futuro de series temporales financieras.

Para un tratamiento más detallado se refiere al lector al libro de Tsay [16].

3.1. Modelos AR(p)

Los modelos Auto-Regressive de orden p, AR(p), se definen mediante la ecuación

$$r_t = \phi_0 + \phi_1 r_{t-1} + \ldots + \phi_p r_{t-p} + \epsilon_t \,, \tag{3.1}$$

donde p es un entero no negativo y ϵ_t es un error de tipo ruido blanco con media 0 y varianza σ_ϵ^2. Este tipo de modelo expresa que el valor actual r_t de la variable de interés depende linealmente de las p observaciones anteriores, r_{t-j}, $j = 1, \ldots, p$ y de un término estocástico, ϵ_t

La ecuación característica de este modelo es

$$1 - \phi_1 x - \ldots - \phi_p x^p = 0 \,. \tag{3.2}$$

Si todas las soluciones de esta ecuación están fuera del círculo unidad, entonces decimos que el modelo bajo análisis es débilmente estacionario.

3.2. Modelos MA(q)

Los modelos de tipo MA(q) (Moving Average de orden q) se obtienen a partir de un modelo AR(p) en el límite de p yendo a infinito [16]. En términos de los errores el mismo se escribe como

$$r_t = \theta_0 + \epsilon_t - \theta_1 \epsilon_{t-1} - \ldots - \theta_q \epsilon_{t-q}\,, \tag{3.3}$$

donde los θ_i son constantes y ϵ_t es una serie de ruidos blancos.

Se puede ver que este tipo de modelos resultan débilmente estacionarios, por ser una combinación lineal de ruidos blancos con media y desvío estándar independientes del tiempo.

3.3. Modelos ARMA(p, q)

Un modelo ARMA(p,q) general tiene la forma

$$r_t = \phi_0 + \sum_{i=1}^{p} \phi_i r_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}\,, \tag{3.4}$$

donde ϕ_i y θ_j son constantes y ϵ_t representa una serie de ruidos blancos.

3.4. Modelos de volatilidad

3.4.1. Estructura general

Para entender cómo funcionan los modelos de volatilidad, denominados también modelos de heteroscedasticidad condicionada, comencemos por analizar la media condicional μ_t y la varianza condicional σ_t para los rendimientos logarítmicos r_t

$$\mu_t = E(r_t|F_{t-1}), \qquad \sigma_t = Var(r_t|F_{t-1})\,, \tag{3.5}$$

donde F_{t-1} representa la información disponible a tiempo $t-1$. Asumiendo un modelo de tipo ARMA(p,q) podemos escribir

$$r_t = \mu_t + \epsilon_t, \quad \mu_t = \sum_{i=1}^{p} \phi_i y_{t-i} - \sum_{j=1}^{q} \theta_j \epsilon_{t-j}\,, \tag{3.6}$$

siendo

$$y_t = r_t - \phi_0 - \sum_{k=1}^{l} \beta_k x_{kt} \,, \tag{3.7}$$

el valor corregido de r_t al tener en cuenta ciertos regresores exógenos x_{kt}, tales como el día de la semana, la semana del mes, etc. En términos de este modelo, la varianza condicional adopta la forma

$$\sigma_t^2 = Var(\epsilon_t | F_{t-1}) \,. \tag{3.8}$$

Los modelos de heteroscedasticidad condicional estudian justamente la evolución temporal de σ_t^2 y se distinguen entre si por la formulación analítica que hacen de la misma. En este contexto, el modelo representado por la ecuación (3.6) se conoce como la representación para la media de un activo r_t, mientras que el modelo que se utilice para describir el comportamiento de σ_t^2 se denomina ecuación de volatilidad para r_t. A su vez, se denomina a ϵ_t como shock o innovación del rendimiento a tiempo t.

En las subsecciones siguientes, analizaremos algunos modelos de heteroscedasticidad condicional, como son el modelo ARCH y GARCH.

3.4.2. Modelos ARCH(m)

Los modelos de tipo ARCH (Auto Regressive Conditional Heteroskedasticity) se basan en dos supuestos:

- El shock t del rendimiento de un activo esta descorrelacionado serialmente, pero depende del tiempo.
- El comportamiento del shock a tiempo t se puede describir en términos de una función cuadrática de sus valores a tiempos anteriores.

En este sentido, un modelo de tipo ARCH(m) asume que

$$\epsilon_t = \sigma_t \chi_t \,, \qquad \sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1} + \ldots + \alpha_m \epsilon_{t-m} \,, \tag{3.9}$$

donde χ_t es un conjunto de variables aleatorias idénticamente distribuidas con media 0 y varianza 1, $\omega > 0$, $\alpha_i >= 0$ para $i = 1, 2, \ldots$.

Los coeficientes α_i deben, a su vez, satisfacer condiciones de regularidad para que la varianza condicional de ϵ_t se mantenga acotada. Por lo común,

suelen usarse para representar a χ_t distribuciones normales estandarizadas o de tipo t de Student estandarizada.

3.4.3. Modelos GARCH(m, s)

Nuevamente descomponemos el rendimiento logarítmico r_t a tiempo t como $r_t = \mu_t + \epsilon_t$ y escribimos

$$\epsilon_t = \sigma_t \chi_t, \qquad \sigma_t^2 = \omega + \sum_{i=1}^{m} \alpha_m \epsilon_{t-i} + \sum_{j=1}^{s} \beta_i \sigma_{t-j}^2 \,, \tag{3.10}$$

donde ahora debe cumplirse que $\beta_j >= 0$ y que $\sum_{l=1}^{max(m,s)} (\alpha_i + \beta_i) < 1$ para garantizar la regularidad. Es fácil ver que el modelo GARCH(m,s) se reduce a uno de tipo ARCH(m) tomando s = 0.

3.5. Determinación del orden de un modelo

Los órdenes de los modelos son desconocidos a priori y deben determinarse de manera empírica. Existen dos procedimientos habituales para hacer esto. El primero de ellos es un proceso de análisis gráfico centrado en el estudio de las ACF y PACF, mientras que el segundo busca minimizar ciertos indicadores, conocidos como criterios de información.

3.5.1. Determinación basada en ACF y PACF

La determinación de los órdenes usando esta técnica gráfica no siempre es posible y en general resulta compleja. Como resumen podemos listar las siguientes propiedades:

- En el caso de modelos AR(p), se suele utilizar la PACF, ya que esta deja de tener coeficientes estadísticamente significativos al lag p.
- En el caso de modelos MA(q), se estudia la ACF, ya que esta deja de tener coeficientes significativos al lag q.
- En el caso de modelos ARMA(p,q) no se pueden usar las ACF y PACF directamente y se emplean criterios de información.

- En el caso de modelos ARCH(m), se estudia la PACF de los shocks cuadrados, ϵ_t^2, a lo sumo hasta orden m.
- En el caso de los modelos GARCH(m,s) la determinación del orden no resulta sencilla. Por lo general se trabaja con modelos de orden bajo, del estilo GARCH(1,1), GARCH(2,1) o GARCH(1,2).

3.5.2. Determinación basada en criterios de información

Todos los criterios de información están basados en consideraciones de máxima verosimilitud de los parámetros de la distribución de probabilidad de las observaciones. Por ejemplo, el Criterio de Información de Akaike (AIC) se escribe como

$$AIC = -\frac{2}{N}\ln(FV) + \frac{2P}{N}\,, \tag{3.11}$$

donde la función de verosimilitud (FV) se evalúa en los parámetros de máxima verosimilitud, P es el número de parámetros del modelo y N es el tamaño de la muestra. Más concretamente, para un modelo AR(p) Gaussiano, el AIC resulta

$$AIC = -\ln(\hat{\sigma}_p) + \frac{2p}{N}\,, \tag{3.12}$$

siendo σ_p^2 el estimador de máxima verosimilitud para σ_ϵ^2, la varianza del error.

En este criterio, el primer sumando establece la bondad del ajuste, mientras que el segundo, conocido como función de penalidad, desfavorece los modelos con un número de parámetros elevado.

La función de penalidad es la que define los distintos criterios de información. Por ejemplo, el criterio de información de Schwarz-Bayes (BIC) adopta para un modelo AR(p) Gaussiano la forma

$$BIC = -\ln(\hat{\sigma}_p) + \frac{p}{N}\ln(N)\,. \tag{3.13}$$

Tanto AIC como BIC son dos de los criterios de información más comunmente utilizados a la hora de determinar los parámetros en este tipo de modelos. Comparando entre ambos, podemos notar que la función de penalidad del BIC establece una penalidad que se incrementa con el logaritmo

del tamaño de la muestra, Comparando AIC con BIC, vemos que este último presenta una función de penalidad que decae menos abruptamente con el tamaño muestral que el primero, gracias a la contribución logarítmica. En este sentido, a mayor tamaño muestral, la minimización del BIC como criterio de selección favorecerá modelos con un número menor de parámetros comparada con el uso del AIC.

Capítulo 4

Bases de datos para series temporales

4.1. Descripción general

Las bases de datos para series temporales (BDSTs) están construidas específicamente para manejar métricas y eventos, esto es, datos que estén medidos de forma tal que no solo se almacena su valor, sino el tiempo en el que se realiza la medición. A su vez, este tipo de DDBBs se encuentra optimizada para medir cambios de valores en el tiempo [17].

Las BDSTs no son un concepto nuevo. La primera generación de BDSTs eran de código cerrado y estaban mayormente dedicadas al análisis de datos financieros para analizar fenómenos tales como la volatilidad de las acciones o bien para automatizar sistemas de compra y venta de activos. Sin embargo, el advenimiento de las nuevas tecnologías ha posibilitado la producción de señales temporales que resultan de interés desde las más diversas fuentes y con un velocidad de emisión de datos que ha empujado a estas plataformas a evolucionar y diversificarse. Al día de la fecha, vemos en el ranking elaborado por el sitio especializado DB-Engines[1], que existen al menos 36 BDSTs que están siendo monitoreadas. En la Fig. 4.1 vemos que lidera el top 10 la BDST InfluxDB empleada en este trabajo.

[1] `https://db-engines.com/en/ranking/time+series+dbms`

RANK	DBMS	SCORE		
AUG 2021		AUG 2021	24 MOS ▲	12 MOS ▲
1	**InfluxDB**	**29.56**	**+10.91**	**+6.22**
2	Kdb+	7.99	+2.49	+0.56
3	Prometheus	6.20	+2.75	+0.51
4	Graphite	4.86	+1.54	+0.55
5	TimescaleDB	3.41	+2.01	+0.68
6	ApacheDruid	3.00	+1.27	+0.71
7	RRDtool	2.43	-0.14	-0.62
8	OpenTSDB	1.90	+0.04	-0.40
9	FaunaDB	1.53	+1.11	-0.33
10	GridDB	1.37	+0.89	+0.62

Source: DB-Engines 23 Systems in Ranking, August 2021

Figura 4.1: Ranking de BDSTs elaborado por el sitio especializado DB-Engine. Los valores son de Agosto de 2021.

Por otro lado, también siguiendo las métricas provistas por DB-Engines, vemos que las BDSTs son las bases de datos que más interés han despertado en el último tiempo.

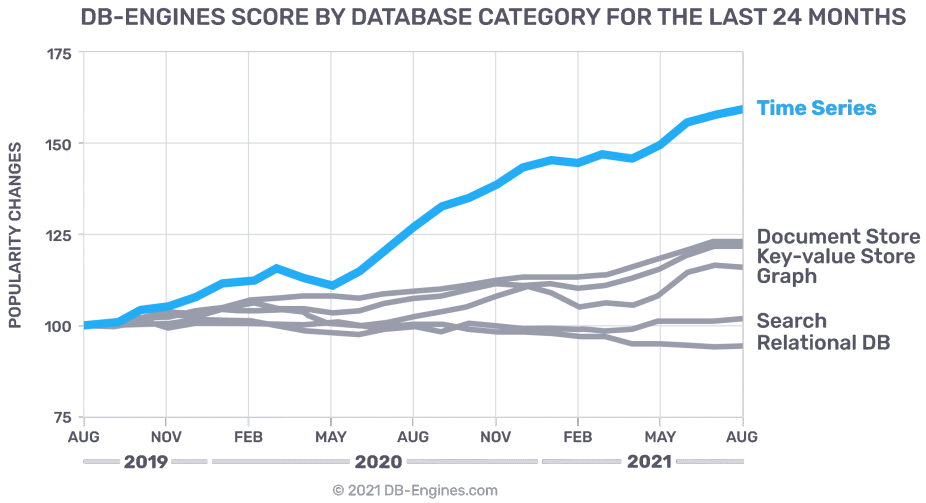

Figura 4.2: Ranking de BDSTs elaborado por el sitio especializado DB-Engine. Los valores son de Agosto de 2021.

4.2. Casos de uso

En esta sección se discuten tres casos de uso primarios en el contexto de BDSTs: Monitoreo de DevOps, Analítica en tiempo real y Monitoreo de sensores de IoT.

4.2.1. Monitoreo de operaciones para DevOps

DevOps es un conjunto de prácticas que combina el desarrollo de software (Dev) y operaciones de IT (Ops). Su objetivo es acortar el tiempo que se emplea durante el ciclo de desarrollo de un dado sistema además de garantizar la entrega continua de software de la más alta calidad. El workflow asociado con esta práctica pueda observarse en la Fig. 4.3.

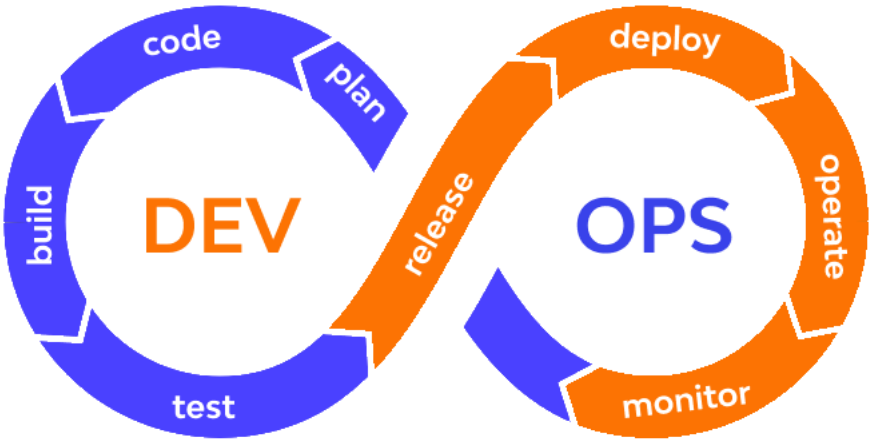

Figura 4.3: Workflow asociado con la práctica de DevOps.

En este contexto, las BDSTs no solo se emplean para acumular métricas

y definir valores umbral que permitan diagnosticar la salud de los sistemas, sino que también se emplean para guardar todo tipo de logs para consulta futura y a su vez, para soportar distintos tipos de tableros de monitoreo en tiempo real de los distintos sistemas, aplicaciones e infraestructura [18].

4.2.2. Analítica en tiempo real

La analítica en tiempo real busca procesar los datos en tránsito utilizando algoritmos de Machine Learning con el fin de proveer recomendaciones o señales que permitan tomar decisiones rápidamente. Cabe destacar que en muchos casos, "en tiempo real" significa que el proceso de análisis se produce algunos segundos o minutos después de que los datos han sido almacenados.

Desde una perspectiva de negocios, los beneficios de la analítica en tiempo real son muchos. Se pueden usar este tipo de técnicas para, por ejemplo, proveer recomendaciones de distinto tipo en tiempo real [19] o brindar alertas médicas para pacientes en riesgo [20].

Dado el enfoque de este trabajo, resulta de especial interés el caso del mundo de los bancos y las finanzas, puesto que tiene todos los ingredientes necesarios para beneficiarse de la analítica en tiempo real: enormes volúmenes de datos, latencias del orden de los milisegundos, volatilidades extremas y la necesidad de detectar comportamientos sospechosos y actuar en consecuencia.

En este sentido, la habilidad para rápidamente correlacionar, analizar y actuar sobre datos de operaciones de compra-venta, precios de mercado, fusiones de compañías y otros datos provenientes de múltiples fuentes resulta imperiosa para las organizaciones dentro de esta industria. Casos tales como la detección de lavado de activos o fraude [21] [22], manejo de riesgo [23] y el monitoreo de activos en tiempo real [24] suelen destacar por sobre el resto.

En este tipo de escenarios, la posibilidad de contar con una BDST resulta determinante, no sólo porque pueden escribir una gran cantidad de datos a una elevada velocidad, sino también porque ofrecen la arquitectura necesaria para ingestar, almacenar y procesar los mismos de manera transparente. A su vez, las BDSTs pueden almacenar la información con la precisión requerida por los datos de alta frecuencia y no solamente esto, sino que también permiten gobernar el ciclo de vida de los mismos. Esto es asi ya que se puede programar la eliminación de forma periódica de los datos que no son ya relevantes o bien se pueden agregar los mismos (por ejemplo tomando el promedio diario de observaciones con frequencia de minutos o segundos),

agregando observaciones en forma automática después de un dado periodo de tiempo para minimizar el impacto en la infraestructura de almacenamiento, etc.

4.2.3. Monitoreo de sensores de IoT

Hasta hace no mucho tiempo, exceptuando a las computadoras y los smartphones, no existían dispositivos capaces de estar conectados a Internet. Sin embargo, hoy en día cada vez surgen más dispositivos capaces de conectarse a Internet e intercambiar información a través de ella, permitiéndonos realizar la operación y la recolección de datos en forma remota. Este fenómeno es lo que se ha dado en llamar revolución IoT (Internet of Things). Esta revolución se está produciendo ahora puesto que hemos encontrado la forma de asignarle a cada dispositivo que fabricamos una dirección IP así como también tenemos el ancho de banda suficiente para permitir la comunicación entre dispositivos y a su vez tenemos la capacidad de almacenear la enorme cantidad de datos que se crean a partir de los mismos.

Al igual que en el caso anterior, la necesidad de contar con una infraestructura capaz de ingestar señales de diversos origenes, con distas frecuuencias de muestreo, a intervalos regulares o no, vuelve la existencia de las BDSTs algo más que necesario [25].

Capítulo 5

Herramientas requeridas para la implementación

5.1. InfluxDB: Captura, almacenamiento, consulta y visualización

El contenido de esta sección fue adaptado mayormente de la documentación on-line y publicaciones técnicas provistas por InfluxDB. Para más información sirvase consultar `https://docs.influxdata.com`.

5.1.1. Descripción general

InfluxDB es parte de una plataforma más general (stack) que admite la captura, almacenamiento, monitoreo y visualización de series temporales, así como también el disparo de alertas basadas en eventos relacionados con dichas series.

A diferencia de otras BDSTs, InfluxDB fue diseñada desde cero para trabajar con series temporales y esto le otorga varias ventajas comparativas.

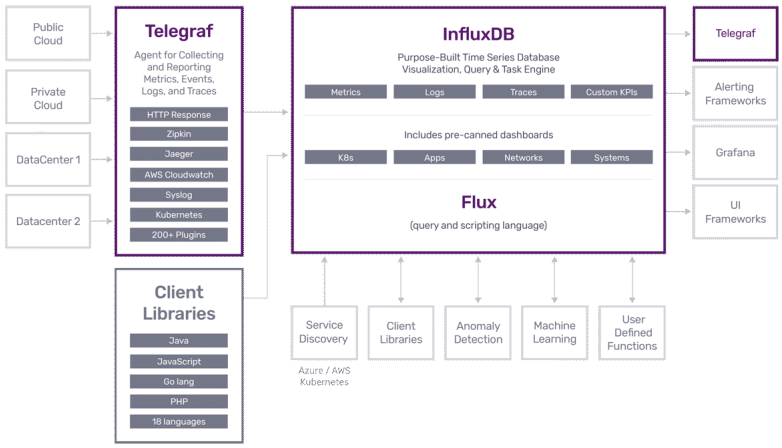

Figura 5.1: Arquitectura del stack de InfluxDB.

InfluxDB puede guardar datos con precisiones de segundo, milisegundo, microsegundo o nanosegundo de ser necesario y esto la ubica como una de las elegidas a la hora de almacenar datos financieros o científicos. Por otra parte, el nivel de compresión a la hora de almacenar los datos es variable y se adapta a las necesidades de precisión del usuario. La estructura de almacenamiento en disco es de tipo columnar y en la misma los datos temporales se almacenan en bloques contiguos relacionados con una cierta medida, tagset y campo. Es decir, cada campo se organiza en forma secuencial en el disco usando el ordenamiento temporal natural, por lo que hacer agregaciones sobre un dado campo se convierte en una operación extremadamente rápida.

Mencionemos brevemente para finalizar algunos de los componentes que se encuentran en la Figura 5.1. Uno de los stacks más utilizados para trabajar con InfluxDB es el que involucra el uso de Telegraf, InfluxDB y Grafana (TIG stack). Telegraf es un agente que sirve para recolectar y reportar métricas, eventos y logs con un alto nivel de integración con numerosas fuentes de datos gracias a los más de 100 plugins disponibles a tal efecto. Por otra parte, Grafana es una aplicación web de código abierto multi-plataforma que permite la visualización interactiva de datos usando gráficos. Es este stack en su conjunto el que permite la captura (Telegraf), almacenamiento y monitoreo (InfluxDB) y visualización (Grafana).

5.1.2. Modelo de datos

En lo que a modelo de datos se refiere, los conceptos principales que debemos entender son los que corresponden a: bases de datos, politicas de retención, series, medidas, tags y campos.

En lo que a *base de datos* se refiera, podemos pensar a influxdb como un contenedor lógico de usuarios, políticas de retención, consultas continuas (CC)[1] y por último los datos asociados con las series temporales. Una *política de retención* (PR) describe por cuanto tiempo InfluxDB mantiene los datos disponibles (duración), cuántas copias de los datos guarda en el cluster (factor de replicación) y el intervalo de tiempo que utilizan los grupos de shards[2] (duración de los grupos de shards). Una *serie* es un agrupamiento lógico de datos comprendido por una medida, un conjunto de tags y un conjunto de campos. Una *medida* describe los datos guardados en los campos asociados. Un *tag* es el par llave-valor que guarda metadatos. Estos son opcionales dentro del modelo de datos, pero resultan útiles a la hora de almacenar metadatos que se consultan en forma habitual. Por último, un *campo* es también un par llave-valor que guarda metadatos y los valores numéricos que finalmente conforman la serie temporal (observaciones).

Ejemplos de uso de estos conceptos quedaran más claros una vez que analicemos el protocolo de línea que usa InfluxDB para escribir los datos en la próxima sección.

5.1.3. Escritura

En esta sección analizaremos la forma en la cual se pueden escribir datos en InfluxDB. En primero lugar comenzaremos describiendo el protocolo de línea de texto que se utiliza para codificar la información. Luego estudiaremos dos mecanismos para escribir estos datos, el agent server Telegraf y el Cliente de Python para InfluxDB.

Protocolo de linea de texto

[1] Una consulta continua (CC) es una consulta que corre automática y periódicamente en una base de datos.

[2] Una shard es una partición horizontal de los datos en una base de datos o motor de busqueda.

InfluxDB usa el protocolo de linea de texto para codificar las observaciones antes de ser escritas. Este protocolo es un formato basado en texto que provee la medida, el tag, el campo y el timestamp de una observación. Un ejemplo podría ser el siguiente

```
// Sintaxis
<measurement>[,<tagKey>=<tagValue>] <fieldKey>=<fieldValue> [<timestamp>]

// Ejemplo
myMeasurement,tag1=value1 fieldKey="fieldValue" 1556813561098000000
```

Cabe destacar que el protocolo de línea de texto es sensible a los espacios en blanco y que cada nueva observación se distingue de la anterior usando un salto de linea.

Antes de describir cada uno de los elementos que componen al protocolo de línea de texto, veamos como luce el mismo para una observación de ejemplo:

```
measurementName,tagKey=tagValue fieldKey="fieldValue" 1465839830100400200
--------------- --------------- --------------------- -------------------
       |               |                 |                   |
     Medida           Tag              Campo             Timestamp
```

Notamos que el mismo está conformado por una secuencia de cuatro partes, la medida, el (los) tag(s), el (los) campos(s) y el Timestamp.

Medida

(Requerido) Nombre de la medida. InfluxDB acepta una medida por observación. Los nombres de las medidas son sensibles a las mayúsculas y tienen algunas restricciones.
Data type: String

Tag

(Opcional) Todos los pares llave-valor para la observación. Las relaciones llave valor se denotan con el operador =. En caso de que existan múltiples entradas clave-valor, las mismas se separan entre si usando comas. Tanto la llave como el valor son sensibles a las mayúsculas. Además las llaves tienen algunas restricciones de nombre.

Data type llave: String
Data type valor: String

Campo

(Requerido) Todos los pares llave-valor de tipo campo para la observación. Cada observación debe tener al menos un campo. Tanto las llaves como los valores tipo string de los campos son sensibles a las mayúsculas. A su vez, las llaves tienen algunas restricciones de nombre.

Data type llave: String
Data type valor: Float — Integer — UInteger — String[3]— Boolean

Timestamp

(Opcional) El Timestamp de Unix para la observación. InfluxDB acepta solo un Timestamp por observación. Si el Timestamp no se provee, InfluxDB usa el tiempo del sistema (UTC) de la maquina donde está hosteado. Es importante tener en cuenta que InfluxDB espera por defecto Timestamps de Unix con una precisión de nanosegundos (ns). En caso de que se requiera insertar datos con una precisión diferente, se debe especificar a la hora de ejecutar la operación de escritura.

Data type: Unix timestamp

Espacios en blanco

El espacio en blanco en el protocolo de linea determina cómo InfluxDB interpreta la observación. El primer espacio separa el nombre de la medida y el (los) tag(s) de el (los) campos. El segundo espacio en blanco, separa el

[3]Siempre escribir los valores de campo tipo String entre comillas

(los) campos del Timestamp de Unix.

```
nombreMedida,llaveTag=valorTag llaveCampo="valorCampo" 1465839830100400200
                               |                       |
                         1er espacio             2do espacio
```

Telegraf

Telegraf es un agent server basado en plugins para ingestar y enviar métricas y eventos desde bases de datos (MongoDB, MySQL, Redis y otras), sistemas (Stack de clouds, Docker containers, Kubernetes) y sensores de IoT (sensores de presión, temperatura, humedad, posicionamiento, etc).

Telegraf tiene una amplia variedad de plugins para capturar datos diversos (plugins de input) ya sea del propio sistema en el que esta corriendo, como así también de APIs de terceros, logs de Kafka, etc. Por otro lado, existen plugins de output que le permiten enviar estos mismos datos hacia bases de datos, otros servicios, colas de mensajes, etc.

Dentro del marco de trabajo, los plugins de input que podrían resultar relevantes son `File`, `Tails`, `Exec`. Los dos primeros permiten tomar datos desde archivos de texto, usualmente CSVs, mientras que el último ejecuta algún script que permite conectarse a una API externa y tomar desde allí los datos. En relación con los plugins de output, sólo mencionaremos el que permite escribir los datos recolectados en InfluxDB, i.e., `InfluxDBv2`.

Para una mirada más exhaustiva sobre Telegraf, se recomienda consultar la documentación on-line `https://docs.influxdata.com/telegraf/`.

Cliente de Python

InfluxDB cuenta con librerías cliente en distintos lenguajes, como pueden ser PHP, Java, Node.js, Scala, Python, etc. La idea es que estos sirvan para interactuar con la base de datos desde diferentes programas escritos en los lenguajes antes mencionados. En nuestro caso particular, la intención es usar el cliente de Python para leer y escribir datos en InfluxDB.

Para entender cómo actúa el cliente, podemos analizar el siguiente script.

```
import influxdb_client
```

```
org = "<my-org>"
token = "<my-token>"
url="http://localhost:8086"

bucket = "<my-bucket>"

client = influxdb_client.InfluxDBClient(org=org, token=token, url=url)

write_api = client.write_api()

p = influxdb_client.Point("my_measurement").tag("location", "Prague") \
    .field("temperature", 25.3)

write_api.write(bucket=bucket, org=org, record=p)
```

Como podemos ver, importamos el paquete correspondiente y luego consignamos la información necesaria para poder logearnos a la database, esto es, brindamos el nombre de nuestra *organización*, el *token* de seguridad, la *url* a la que tenemos que apuntar para establecer la conexión y finalmente el *bucket* con el cual deseamos interactuar. Una vez hecho esto, instanciamos el cliente, le pasamos estos datos como parámetros y ya quedamos listos para operar. Acto seguido, instanciamos la API de escritura, generamos una observación sintética[4] y se la pasamos a la primera para que la escriba. Si la operación resulta exitosa, tiene que devolver el código 204, como establece la documentación de la API[5].

5.1.4. Consulta

Lenguaje de consulta Flux

Las consultas a InfluxDB se realizan usando un lenguaje basado en JavaScript, llamado Flux. El mismo es un lenguaje de scripting y consulta autónomo que está optimizado para tareas de ETL, monitoreo y alerta.

[4]El método `Point` crea un punto usando el protocolo de línea de texto para poder escribirlo en InfluxDB.

[5]Este código establece que la información está formateada correctamente y que fue aceptada para escribir en el bucket. Para más detalles, consulte `https://docs.influxdata.com/influxdb/v2.0/api/#operation/PostWrite`

Para poder entender los conceptos clave asociados con este lenguaje, resulta conveniente analizar un ejemplo. A continuación vemos una consulta en la cual solicitamos las observaciones promediadas con ventana de un minuto para la medida `cpu` con tag `cpu-total` que se hayan registrado en la última hora:

```
from(bucket:"example-bucket")
  |> range(start:-1h)
  |> filter(fn:(r) =>
    r._measurement == "cpu" and
    r.cpu == "cpu-total"
  )
  |> aggregateWindow(every: 1m, fn: mean)
```

La primera linea nos dice desde que *bucket*[6] estamos tomando la información. A continuación aparece el operador de *pipe-forward*, `|>`, el cual se usa para concatenar operaciones. Luego pedimos, usando el operador *range* que la ventana de muestreo se extienda hasta una hora antes del momento actual. En las líneas siguientes aplicamos una función de filtrado (equivalente a una cláusula `WHERE` de SQL) para seleccionar la medida y el tag deseados. Por último, aplicamos la función *aggregateWindow* para promediar las obervaciones dentro de cada minuto.

Cabe destacar que después de cada operación, Flux devuelve una tabla[7] o colección de tablas y es el operador de pipe-forward el encargado de disponibilizar las mismas para que estén listas a la hora de realizar la operación siguiente, dando lugar a una estructura lógica de pipeline. Esto redunda en una simplifiación de la lectura de la consulta, puesto que su escritura pone de manifiesto en forma explícita el proceso de transformaciones que sufren los datos para alcanzar el resultado final.

Cada tabla tiene una *clave de grupo* que describe los contenidos de la misma. Esta clave de grupo está constituida por una lista de columnas, cada una de las cuales tiene el mismo valor en sus celdas. A medida que vamos recorriendo el pipeline que establece la consulta de Flux, cada operación va

[6] Un bucket se puede entender como una carpeta en la cual se guardan las observaciones asociadas con una o más medidas.

[7] Flux estructura todos los datos en *tablas* usando el formato de CSV anotado para manipularlas.

transformando las tablas y en este proceso sus claves de grupo pueden modificarse. Es importante tener esto en cuenta para lograr los resultados deseados al aplicar cada transformación.

Estructura básica de una consulta en Flux

Toda consulta de Flux requiere que los siguientes elementos esten presentes: una fuente de datos (bucket), un intervalo de tiempo (range) y una serie de filtros.

Fuente de datos

La función `from()` define la fuente de datos en Flux. La misma toma como parámetro el nombre del bucket a utilizar. Por ejemplo

```
from(bucket:"bucket-ejemplo")
```

Intervalo de tiempo

Para evitar problemas de performance, Flux demanda que todas las consultas que se realicen tengan una ventana temporal acotada. Para ello, es necesario que toda la información que se pide utilizando la función `from()` se rutee hacia la función `range()` utilizando el operador de pipe-forward. La función `range()` acepta dos parámetros, *start* y *stop*, siendo este último opcional. Los intervalos definidos mediante esta función pueden ser *relativos* si usamos duraciones negativas o *absolutos* si usamos Timestamps.

A continuación vemos dos ejemplos de consulta en la que se usa la función `range()`. En la primera, tenemos un intervalo de tiempo relativo,

```
// Intervalo temporal con parametro de start solamente. El parámetro stop
// se toma como now por default.
from(bucket:"bucket-ejemplo")
  |> range(start: -30m)

// Intervalo temporal con parametros de start y stop
```

```
from(bucket:"bucket-ejemplo")
  |> range(start: -1h, stop: -30m)
```

mientras que en la segunda, un intervalo de tiempo absoluto.

```
// Absolute time range
from(bucket:"bucket-ejemplo")
  |> range(start: 2018-11-05T23:30:00Z, stop: 2018-11-06T00:00:00Z)
```

Filtros

A fin de reducir aún más el criterio de búsqueda de datos, la tabla que se obtiene como resultado despues de aplicar la función `range()` debe pasarse por una serie de filtros. Para ello, se dirigen los resultados usando el pipe-forward hacia la función `filter()`. Esta función tiene como único parámetro la función anónima `fn`, la cual implementa la lógica necesaria para realizar el filtrado basado en columnas o atributos. Las observaciones o registros, son pasados a esta función para su evaluación como `r`. Veamos un ejemplo de esta lógica a continuación:

```
// Patrón
(r) => (r.propiedadRegistro operadorComparación expresionComparativa)

// Ejemplo con filtro único
(r) => (r._measurement == "cpu")

// Ejemplo con varios filtros
(r) => (r._measurement == "cpu") and (r._field != "usage_system" )
```

Transformaciones básicas de los datos

Cuando se realizan consultas a InfluxDB, en general resulta necesario transformar los datos de alguna forma. Por ejemplo, podemos necesitar que los mismos se muestren en forma agregada usando el valor promedio sobre alguna ventana o bien hacer un downsampling, etc.

A fin de entender cabalmente las transformaciones, reconstruiremos paso a paso el comportamiento de la función `aggregateWindow()`. Para ello, comenzamos escribiendo la siguiente consulta

```
from(bucket:"bucket-ejemplo")
  |> range(start:-1h)
  |> filter(fn:(r) =>
    r._measurement == "cpu" and
    r.cpu == "cpu-total"
  )
```

Con esto, tenemos las medidas de `cpu` con tag `cpu-total` registradas durante la hora anterior. Lo primero que haremos, será "ventanear"los datos. Para ello, usamos la función `window()`. La misma requiere de un parámetro llamado *every* que permite definir la duración de la ventana. Supongamos entonces que vamos a particionar la hora en intervalos de cinco minutos. Para ello, escribimos

```
from(bucket:"bucket-ejemplo")
  |> range(start:-1h)
  |> filter(fn:(r) =>
    r._measurement == "cpu" and
    r.cpu == "cpu-total"
  )
  |> window(every: 5m)
```

Al hacer esto, cada partición tiene asoaciada su propia tabla de salida. Podemos visualizar la salida de esta consulta en la Fig. 5.2.

Figura 5.2: Serie temporal asociada con la carga de uso del cpu durante la última hora. Nótese que la misma fue particionada en ventanas de 5 minutos. Cada ventana esta coloreada en forma diferente para poder distinguirlas.

Como paso siguiente, vamos a agregar los datos presentes en cada ventana. Para ello, mandamos todas las tablas generadas usando el forward-pipe hacia la función `mean()`. El resultado es que la serie contenida en cada ventana ahora se ha transformado en un punto, el valor promedio de las observaciones en cada una de ellas. Esto puede observarse en la Fig. 5.3.

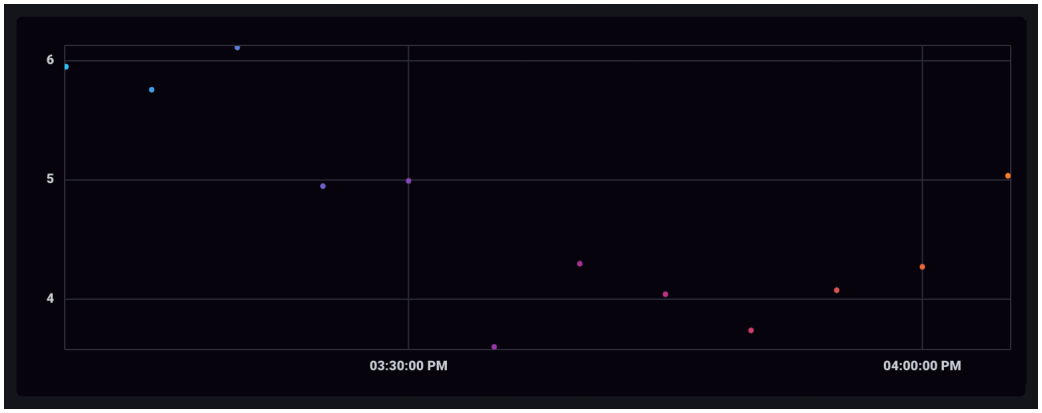

Figura 5.3: Valor promedio de la carga de uso del cpu durante la última hora tomando intervalos de 5 minutos. Nótese que cada valor está coloreado en forma distinta, ya que está guardado en tablas diferentes.

Notemos que los puntos que obtuvimos como resultado están en tablas distintas y por ende aparecerán de forma aislada si los representamos gráficamente. Para volver a consolidar una serie temporal, vamos a tener que hacer dos operaciones. La primera involucra agregar nuevamente una columna que contenga los Timestamps (dicha columna fue descartada por Flux al agregar los datos) y la segunda tiene que ver con extender la ventana y hacerla infinita. Veamos la primera operación con más detalle. Cuando agregamos observaciones, las tablas resultantes no presentan una columna en la que se consignen los timestamps, puesto que no hay forma de seleccionar un valor de entre los distintos que presentan las observaciones. Al agregar, la columna `_time` es descartada. Para subsanar este hecho, utilizamos la función `duplicate()` para duplicar la columna `_stop` y renombrarla como `_time`.

```
from(bucket:"bucket-ejemplo")
  |> range(start: -1h)
  |> filter(fn: (r) =>
    r._measurement == "cpu" and
    r._field == "usage_system" and
    r.cpu == "cpu-total"
  )
  |> window(every: 5m)
  |> mean()
  |> duplicate(column: "_stop", as: "_time")
```

Una vez hecho esto, generamos la ventana infinita escribiendo

```
from(bucket:"bucket-ejemplo")
  |> range(start: -1h)
  |> filter(fn: (r) =>
    r._measurement == "cpu" and
    r._field == "usage_system" and
    r.cpu == "cpu-total"
  )
  |> window(every: 5m)
  |> mean()
  |> duplicate(column: "_stop", as: "_time")
  |> window(every: inf)
```

Vemos en la Fig. 5.4 que después de realizado este proceso, los puntos aparecen ahora conectados, formando una serie temporal. Nótese que esto implica que la operación de ventaneo final, toma varias tablas y devuelve una sola.

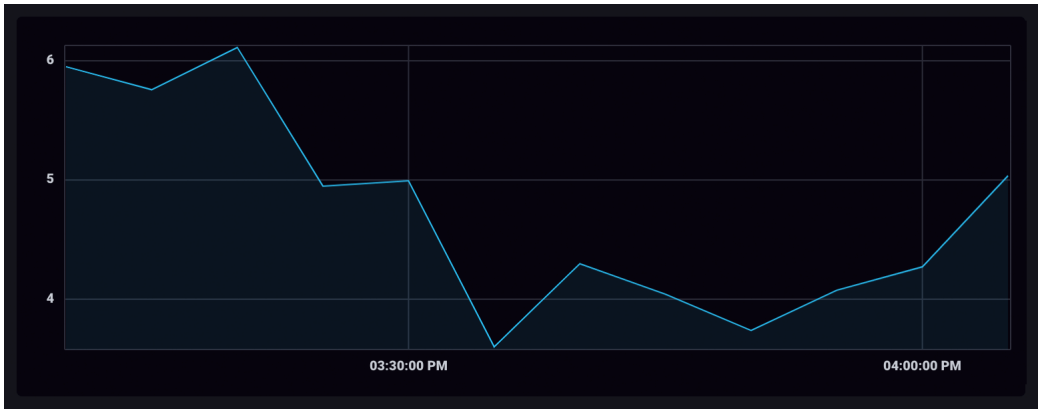

Figura 5.4: Serie temporal asociada con la carga promedio de uso del cpu durante la última hora a intervalos de 5 minutos.

En caso de querer profundizar el entendimiento de las consultas a InfluxDB usando el lenguaje Flux, se recomienda consultar la documentación on-line provista en `https://docs.influxdata.com/influxdb/v2.0/query-data/flux/`.

5.1.5. Visualización

La visualización de los datos en la plataforma de InfluxDB se realiza con una interface de usuario llamada *Chronograf*. Concretamente, Chronograf permite escribir consultas usando Flux para luego mostrar los resultados usando distintos tipos de vistas.

Además de la visualización de datos, Chronograf tiene a su cargo tareas más complejas, como son las del monitoreo de infraestructura, manejo de alertas, administración de la base de datos y administración de organizaciones y usuarios. Estos diferentes casos de uso se observan en la Fig. 5.5.

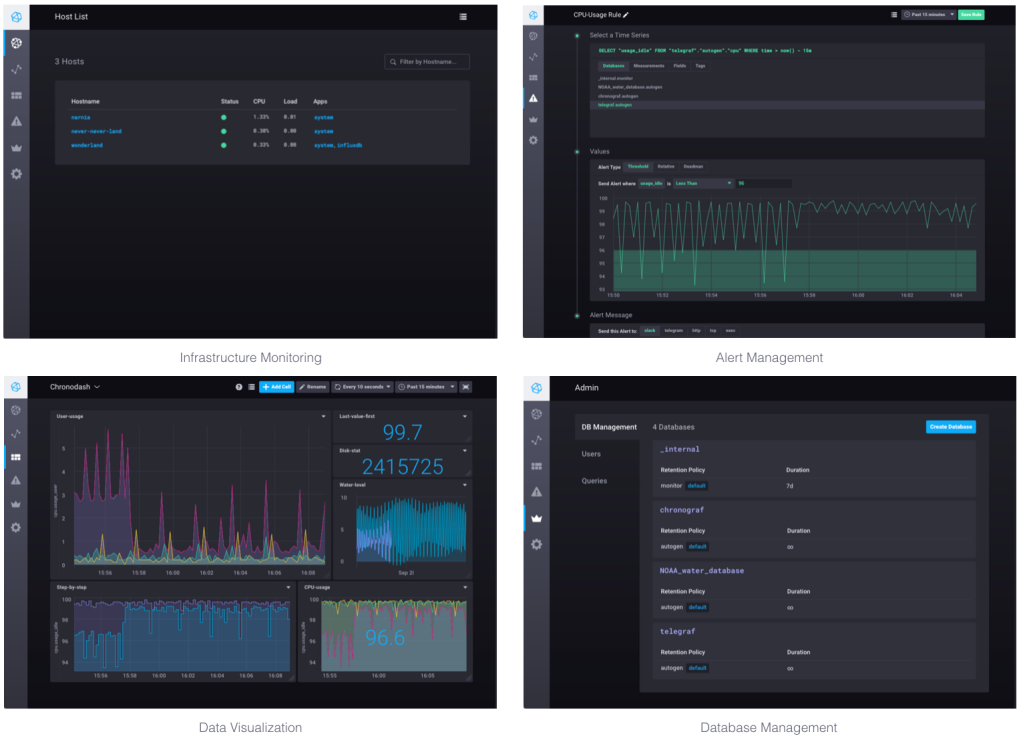

Figura 5.5: Capturas de pantalla que muestran los diferentes escenarios asociados con la interfaz de usuario Chronograf.

Para más información sobre Chronograf, por favor consulte la documentación on-line visitando `https://docs.influxdata.com/chronograf/v1.9/`.

5.2. Python: Análisis y pronóstico futuro

5.2.1. Descripción general

Nacido en 1991 de la mano de Guido Van Rossum, Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje multiparadigma, pues soporta la programación orientada a objetos (OOP), imperativa y, en menor medida, funcional.

Python es un lenguaje interpretado, dinámico y multiplataforma que actualmente es administrado por la Python Software Foundation (PSF). El mismo posee una licencia de código abierto, denominada Python Software

Foundation License.

Las motivaciones principales a la hora de utilizar este lenguaje para nuestro desarrollo fueron su simplicidad de uso, combinada con su amplia adopción en los entornos de trabajo de Data Science y los paquetes específicos para hacer análisis de series temporales e interactuar con InfluxDB.

Para más información sobre Python, visite `https://www.python.org/about/`.

5.2.2. Módulos principales

Se deja a continuación una breve reseña de los paquetes más relevantes que se emplearon durante el desarrollo de este trabajo.

Influxdb-client

Cliente de InfluxDB para Python que permite interactuar con InfluxDB a fin de consultar datos o bien integrar herramientas externas al análisis de las series temporales.

Este cliente soporta multiprocesamiento y permite guardar el resultado de las consultas en un dataframe de pandas. La API de escritura soporta escritura sincrónica, asincrónica y en batches sobre InfluxDB v2.0. La API de escritura también soporta distintos modos de consulta, además de tener la habilidad de interrumpir las mismas.

Pandas

Pandas [26] es un paquete de Python que provee estructuras de datos rápidas, flexibles y expresivas a fin de que el trabajo con datos relacionales o etiquetados resulte simple e intuitivo. Al momento, es la herramienta de análisis de datos más utilizada de Python.

Algunas de sus funcionalidades más destacadas son: Manejo de datos faltantes (representados como NaN, NA, or NaT); Flexibilidad de tamaño: se pueden instertar o elimnar columnas de Dataframes u objetos de mayor dimensionalidad; Alineación de datos automática y explícita; Conversión de datos a otro tipo de estructuras como las de Numpy a Dataframes; Particionado basado en etiquetas o índices; Uniones y fusiones de estructuras de datos intuitivas; Reformulación y pivoteo de datasets flexible; Etiquetado de

ejes jerárquico; Herramientas de I/O rubustas para ingestar datos desde archivos planos (CSV y delimitados), Excel, bases de datos y lectura/escritoria usando el formato ultra rápido HDF5; Manejo de series temporales: generación de rangos de fecha, conversiones de frecuencia, estadistica con ventanas móviles, corrimientos de fechas y más.

Statsmodel

Statsmodels [27] es un paquete de Python que complementa al paquete Scipy agregando funcionalidades relacionadas con el cálculo estadístico que incluye estadística descriptiva y estimación de parámetros e inferencia para modelos estadísticos. De las diversas funcionalidades que provee este paquete, sólo mencionaremos aquellas relacionadas con el análisis de series temporales. En este sentido, el paquete statsmodels provee: Complete StateSpace modeling framework; modelos de tipo SARIMAX, ARIMA, ARMAX; modelos VARMA y VARMAX; modelos de factores dinámicos; modelos de componentes no observados; modelos de Markov ocultos (HMM); modelos AR, ARIMA; modelos VAR, VAR estructural; modelos VECM; atenuado exponencial: modelo Holt-Winters; Test de hipótesis para series temporales: raíz unidad, cointegración, otros; Estadística desriptiva y modelos de procesamiento para el análisis de series temporales.

Pmdarima

Pmdarima [28] (originalmente pyramid-arima, por el anagrama de 'py' + 'arima') es una librería estadística enfocada en portar la facilidad de uso y el poder de las librerias *auto.arima* de R a Python.

Pmdarima incluye: Una colección de tests de estacionariedad y estacionalidad; utilidades como diferenciacion y diferenciacion inversa; Numerosos transformadores y creadores de variables exógenos y endógenos, como por ejemplo trasnformaciones de tipo Box-Cox o de Fourier; Descomposición de series temporales con componentes estacionales; utilidades de validación cruzada; Una colección de series temporales de ejemplo para usar como prototipos o ejemplos; Pipelines a la scikit-learn para consolidar los estimadores y promover la productivización de los modelos entre otros.

Pmdarima está soportado sobre statsmodels, pero fue diseñado con una intefaz similar a la de scikit-learn.

Arch

El paquete Arch [29], contiene lo necesario para implementar lo modelos de tipo Autoregressive Conditional Heteroskedasticity (ARCH) además de otras herramientas para econometría financiera.

Capítulo 6

Proceso de implementación y resultados

En este apartado se describen la arquitectura y el proceso de implementación del pipeline para asociado con la predicción de rendimientos de ETH. A su vez, se analizan los pronósticos obtenidos por el mismo.

En la sección 6.1 se discute la arquitectura del pipeline. En el apartado 6.2 se discuten los pormenores asociados con la captura, el almacenamiento y la consulta de los datos diarios de mercado de ETH. En la sección 6.3 se describe el proceso de preprocesamiento de los datos y la creación de variables de interés. En la sección 6.3 se presentan los resultados obtenidos a partir del análisis exploratorio de los datos. En la sección 6.5 se discute el proceso de entrenamiento de los modelos ARMA y GARCH para realizar la predicción de los rendimientos y se muestran los hiperparámetros obtenidos. Finalmente, en la sección 6.6 se presentan los resultados obtenidos al aplicar el modelo entrenado previamente sobre un conjunto de datos de testeo y se discute su bondad.

6.1. Arquitectura del pipeline

A fin de interpretar el flujo de datos asociado con el pronóstico de rendimientos de ETH, se presenta en la figura 6.1 un diagrama que ilustra la arquitectura del pipeline. Hay una primera etapa de captura manual de la información, esto es, los datos diarios de mercado de ETH se han descargado de la página CoinmarketCap como se verá en la próxima sección. En una segun-

da etapa, este dump manual se preprocesa y se persiste en InfluxDB. Hecho esto, se realiza una consulta y se procesa nuevamente la información para ser enviada al modelo estadístico. En este paso, se establece una segmentación 80:20 de los datos para entrenar y luego testear el modelo. Finalmente, las predicciones se grafican al final de proceso para consumir en forma visual las predicciones.

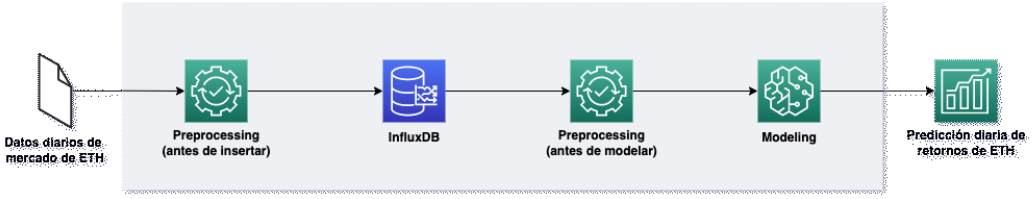

Figura 6.1: Pipeline de procesamiento de datos soportado por InfluxDB para la predicción de rendimientos de ETH.

6.2. Captura, almacenamiento y consulta

Captura

Como se mencionó en la sección 6, el proceso de captura de los datos empleados fue manual. El dataset empleado se descargó del sitio de intercambio CoinmarketCap[1]. Su estructura es la que se observa a continuación:

```
Int64Index: 2160 entries, 0 to 2159
RangeIndex: 2160 entries, 0 to 2159
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   SNo         2160 non-null   int64
 1   Name        2160 non-null   object
 2   Symbol      2160 non-null   object
```

[1]https://coinmarketcap.com

```
 3   Date        2160 non-null   object
 4   High        2160 non-null   float64
 5   Low         2160 non-null   float64
 6   Open        2160 non-null   float64
 7   Close       2160 non-null   float64
 8   Volume      2160 non-null   float64
 9   Marketcap   2160 non-null   float64
dtypes: float64(6), int64(1), object(3)
```

Como puede verse, el dataset consiste de 2160 observaciones diarias de indicadores financieros asociados con la criptomoneda Ethereum (ETH). Las mismas abarcan el período de tiempo comprendido entre el día 09-08-2015 y el día 07-07-2021.

Almacenamiento

Para almacenar de forma local el dataset mencionado en el apartado anterior, se procedió a instalar un cluster de InfluxDB 2.1 en una MacBook Pro M1 (2021) siguiendo el procedimiento detallado en la web del desarrollador[2].

Antes de realizar la inserción en InfluxDB, los datos se preprocesaron de la siguiente manera:

- Se descartaron las columnas "SNo" y "Name",
- Se normalizaron los nombres de las columnas llevandolos a letras minúsculas,
- Se renombró la columna "date" como "time",
- Se convirtió la columna "time" de tipo timestamp a tiempo Unix en nanosegundos (ns),
- Todas las columnas se definieron como de tipo float, excepto por la columna "time", que se definió como datetime64[ns].

[2]https://docs.influxdata.com/influxdb/v2.1/install/

Por último, se realizó la inserción utilizando el script que se observa en el Apéndice A.

Consulta

Finalmente, a fin de disponibilizar los datos para realizar su preprocesamiento y posterior análisis, se realizó una consulta a la instalación local de InfluxDB usando el script que se observa en el Apéndice B.

La estructura del dataset en este punto viene dada por

```
Int64Index: 2160 entries, 0 to 2159
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   datetime   2160 non-null   datetime64[ns, UTC]
 1   symbol     2160 non-null   object
 2   close      2160 non-null   float64
 3   high       2160 non-null   float64
 4   low        2160 non-null   float64
 5   marketcap  2160 non-null   float64
 6   open       2160 non-null   float64
 7   volume     2160 non-null   float64
dtypes: datetime64[ns, UTC](1), float64(6), object(1)
```

6.3. Preprocesamiento

Durante la etapa de preprocesamiento, se convirtieron las timestamps con precisión de nanosegundos a tipo date y se establecieron las mismas como índices para facilitar la manipulación de los datos en las fases siguientes del proceso.

Por otra parte, se crearon los distintos tipos de rendimiento (bruto, neto y logarítmico) y la volatilidad diaria como variables de interés.

Teniendo estas transformaciones mencionadas anteriormente en cuenta, el dataset en este punto del proceso presenta la estructura

```
DatetimeIndex: 2159 entries, 2015-08-10 to 2021-07-07
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   symbol         2159 non-null   object
 1   close          2159 non-null   float64
 2   high           2159 non-null   float64
 3   low            2159 non-null   float64
 4   marketcap      2159 non-null   float64
 5   open           2159 non-null   float64
 6   volume         2159 non-null   float64
 7   gross_return   2159 non-null   float64
 8   net_return     2159 non-null   float64
 9   log_return     2159 non-null   float64
 10  volatility     2159 non-null   float64
dtypes: float64(10), object(1)
```

Por último, se tomó el 70 % inicial de las observaciones del dataset como conjunto de entrenamiento y el restante 30 % como conjunto de testeo. Cabe destacar que durante este proceso, se procuró no alterar la secuencialidad temporal de las observaciones, es decir, toda opción de selección aleatoria a la hora de conformar los conjuntos anteriormente mencionados fue descartada.

6.4. Análisis exploratorio

Para el análisis exploratorio de los datos, se estudiaron los gráficos de los precios de cierre, rendimientos logarítmicos y por último ACF y PACF para los rendimientos logarítmicos. Los mismos se muestran a continuación.

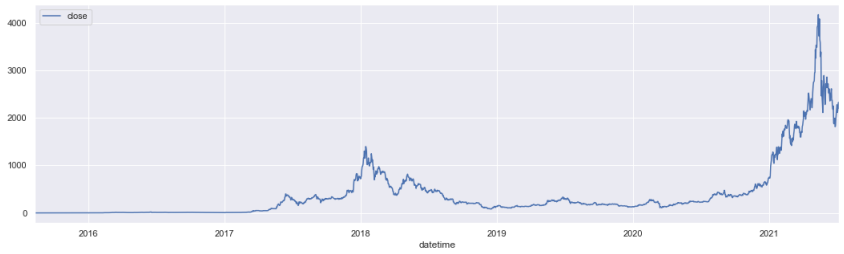

Figura 6.2: Precio de cierre para Ethereum (ETH)

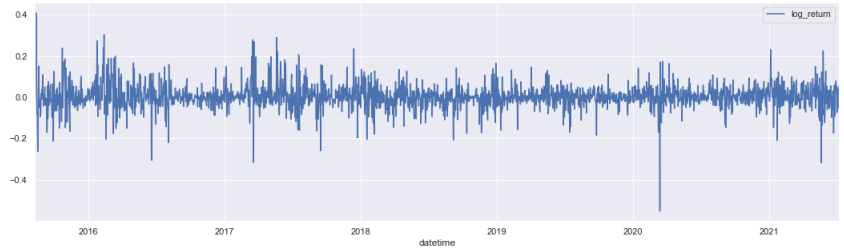

Figura 6.3: rendimiento logarítmico para Ethereum (ETH)

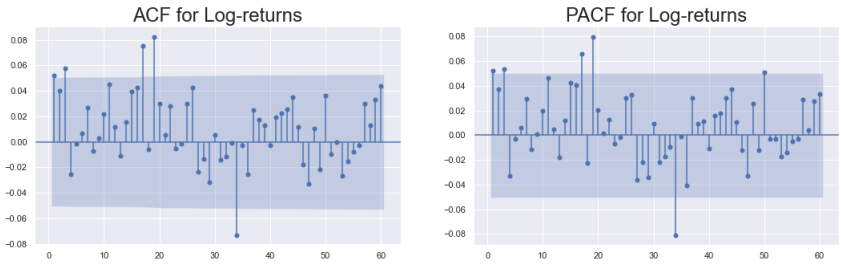

Figura 6.4: ACF y PACF para rendimientos logaritmicos de Ethereum (ETH)

Las conclusiones basicas que podemos sacar de los mismos son las siguientes:

- El precio de cierre de Ethereum parece tener una leve tendencia al alza, presentando máximos locales pronunciados en torno a los primeros meses de 2018 y a mediados de 2021.

- Los rendimientos logarítmicos están centrados en cero (estacionarios), pero no resultan homoscedasticos, i.e., tienen varianzas que dependen del tiempo. Este último comportamiento nos pone de manifiesto que el modelo ARMA será incapaz de predecir completamente el comportamiento de la serie. Más aún, teniendo en cuenta la dependencia temporal de la variaza y notando la clusterización que exhibe la curva, se hace evidente que la incorporación de un modelo de tipo GARCH para analizar los residuos mejorará sensiblemente nuestra capacidad predictiva.

- Finalmente, para los gráficos de las ACF y PACF para los rendimientos logarítmicos, observamos algunos valores estadísticamente significativos para los lags 1, 3, 17, 19 y 34. Esto nos lleva a pensar que deberíamos encontrar un modelo ARMA óptimo con valores ARMA(1,1), ARMA(1,3) o ARMA(3,1).

6.5. Modelado

Para realizar el modelado sobre los datos de entrenamiento, se decidió proceder en dos etapas, como se detalla a continuación.

En la primer etapa, los rendimientos logarítmicos fueron modelados usando un modelo de tipo ARMA. Para ello, se escogió el mejor modelo que ajustara a los datos en base a un criterio extremal para el coeficiente de información de Akaike (AIC). Los resultados del proceso de ajuste para el mejor modelo ARMA encontrado se obervan en la tabla 6.1 a continuación:

Tabla 6.1: Parámetros de ajuste para un modelo ARMA(1, 1).

Nombre	Valor	Error	z	$p > \|z\|$
Parámetros del modelo				
ϕ_1	0.9850	0.011	89.224	0.000
θ_1	-0.9697	0.015	-66.496	0.000
Parámetros de diagnóstico				
Log-likelihood	1987.881			
AIC	-3969.763			
BIC	-3953.803			

Como puede observarse, ambos coeficientes ϕ_1 y θ_1 resultan estadísticamente significativos, por ser el valor p cercano a 0. En consecuencia, el modelo resulta satisfactorio.

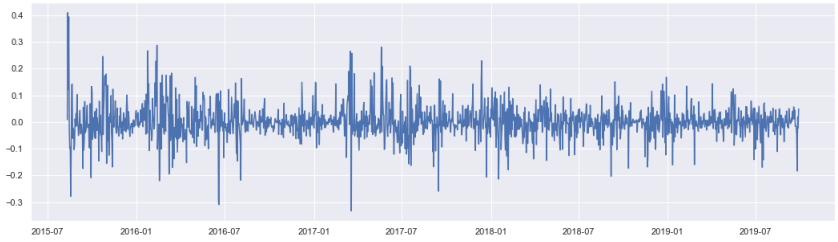

Figura 6.5: Residuos para el modelo ARMA(1,1) ajustado sobre los rendimientos logarítmicos de Ethereum (ETH)

Por otra parte, en la segunda etapa, los residuos del modelo ARMA (véase Fig. 6.5) fueron tratados usando un modelo de tipo GARCH(1,0) en consonancia con lo propuesto por Katsiampa [30]. Los resultados de ese proceso se observan en la tabla 6.2 a continuación:

Tabla 6.2: Parámetros de ajuste para un modelo GARCH(1, 0) considerando media nula y una distribución de probabilidad de tipo t de Student sesgada.

Nombre	Valor	Error	t	$p > \|t\|$
Parámetros del modelo				
ω	4.1001e-03	9.131e-04	4.490	7.108e-06
α_1	0.8728	0.252	3.465	5.305e-04
Parámetros de la distribución				
η	2.5544	0.192	13.307	2.118e-40
λ	-4448.41	1.961e-02	2.438	1.476e-02
Parámetros de diagnóstico				
Log-likelihood	2228.21			
AIC	-4448.41			
BIC	-4427.14			

Nuevamente, se puede ver que, tanto los coeficientes del modelo ω, α_1, así como también los de la distribución de probabilidad empleada, η y λ resultan estadísticamente significativos.

6.6. Predicción

Una vez realizados los ajustes de los rendimientos logarítmicos y sus residuos usando los modelos ARMA(1,1) y GARCH(1,0) respectivamente, se emplearon los mismos para realizar los pronósticos.

En la Figura 6.6 vemos la predicción realizada con el modelo ARMA(1,1) (rojo) sobre los datos de testeo (azul). Las barras de error correspondientes con un intervalo de confianza (IC) del 95 % se observan en color verde.

Figura 6.6: Predicción hecha usando el modelo ARMA(1,1) ajustado sobre los datos de entrenamiento superimpuesta a los rendimientos logarítmicos del conjunto de datos de testeo. Se observan también las barras de error que marcan el intervalo de confianza del 95 %. El MSE obtenido es de 0.003

Está claro que, si bien la predicción con su correspondiente intervalo de error es consistente con los datos de testeo y tiene un MSE igual a 0.003, a los fines prácticos resulta demasiado conservadora. Esto tiene que ver con lo expuesto durante el análisis exploratorio, es decir, el modelo ARMA(1,1) no puede capturar los comportamientos causados por una serie heterocedástica y en consecuencia, sus predicciones serán muy conservadoras. Para superar esta limitación, se utilizó el modelo GARCH(1,0) obtenido sobre los residuos del ARMA(1,1) para reestimar las barras de error. Los resultados pueden verse en la Figura 6.7.

Figura 6.7: Predicción hecha usando el modelo ARMA(1,1) ajustado sobre los datos de entrenamiento montada sobre los returnos logarítmicos del conjunto de datos de testeo. A diferencia del caso anterior, ahora el intervalo de confianza viene dado por las predicciones del modelo GARCH(1,0) para la volatilidad.

Resulta evidente que los intervalos corregidos capturan de una forma más precisa el comportamiento de la serie y en ese sentido, nos permiten identificar mejor el tipo de turbulencia que podríamos enfrentar en un cierto período. Baste para ello observar las buenas predicciones que se obtuvieron para la caída en los rendimientos generada por la declaración de la pandemia de COVID-19 el 11 de Marzo del 2020.

Capítulo 7

Conclusiones

En este trabajo se implementó un pipeline de datos soportado por una base de datos de series temporales (BDST) que permite predecir el rendimiento de criptoactivos usando métodos estadísticos. Puntualmente, se descargaron manualmente datos de trading diarios relativos a la criptomoneda Ethereum (ETH) desde la página CoinmarketCap, se preprocesaron e insertaron en la BDST InfluxDB y se hicieron consultas sobre la misma para entrenar un modelo estadístico de tipo ARMA con intervalos de confianza corregidos empleando GARCH, utilizado para predecir los rendimientos diarios de ETH.

Durante este proceso, se exploró el impacto actual que tienen las BDST desde un punto de vista conceptual y se evaluaron las capacidades de las mismas al instalar y utilizar la suite de InfluxDB. Este último proceso demandó para su éxito el desarrollo de varias competencias, a saber: el manejo de configuraciones internas del sistema operativo utilizado (host), asi como de la propia InfluxDB; el estudio del modelo de datos empleado por la BDST; el aprendizaje del protocolo de línea utilizado por la BDST para escribir los datos; el aprendizaje del lenguaje de consulta Flux para interactuar con InfluxDB.

Por otra parte, tanto el repaso de conceptos estadísticos de la sección 2.3, así como también la formalización de los modelos empleados hecha en el capítulo 3 fueron determinantes para la correcta ejecución de las acciones de modelado realizadas en la sección 6.5 y para la interpretación precisa de los pronósticos obtenidos en la sección 6.6.

En relación con el proceso de modelado de los datos de trading diarios de ETH y la posterior predicción de los rendimientos, cabe destacar que los procedimientos empleados son acordes a las mejores prácticas de la disciplina

y que los resultados obtenidos están dentro de los rangos esperados para este tipo de pronósticos [31] [32].

Por otra parte, si bien el desarrollo de este trabajo involucró un enfoque estadístico para realizar las estimaciones de los rendimientos, es dado mencionar que existen varias técnicas de Machine Learning que podrían haberse empleado para obtener las mismas [33] [34].

Considerando que el trabajo en finanzas cuantitativas se basa en la predicción de series temporales, que las mismas son muchas y de las mas variadas caracteristicas y que los métodos disponibles para realizar los pronósticos son también múltiples, se comprende la importancia que tiene contar con un pipeline lo suficientemente versátil como para simplificar el proceso de gestión de las series temporales, la metadata asociada y el modelado por distintas vias.

Con esto en mente, nos proponemos explorar a futuro mejoras en el pipeline, como por ejemplo la automatización de la captura de los datos financieros via consulta directa a APIs de sitios de intercambio (Binance, CoinmarketCap, etc.) así como también la implementación de algoritmos de Machine Learning más complejos que los modelos estadísticos básicos aplicados en el presente trabajo.

Por último, es dado mencionar que todo el código generado durante la realización de este trabajo se encuentra a disposición en un repositorio público.

Apéndice A

Script para escritura a InfluxDB

```python
import os
import logging
import pandas as pd
import datetime
from influxdb_client import InfluxDBClient, WritePrecision, WriteOptions

################################################################ LOGGING ############################################################################
#####################################################################################################################################################

LOGGER_FORMAT = '%(asctime)s %(message)s'
logging.basicConfig(format=LOGGER_FORMAT, datefmt='[%H:%M:%S]')
log = logging.getLogger()
log.setLevel(logging.INFO)

#####################################################################################################################################################
#####################################################################################################################################################

################################################################ PARAMETERS #########################################################################
#####################################################################################################################################################

# INFLUXDB CLIENT
INFLUXDB_URL = os.getenv('INFLUXDB_URL')
INFLUXDB_TOKEN = os.getenv('INFLUXDB_TOKEN')
INFLUXDB_ORG = os.getenv('INFLUXDB_ORG')
INFLUXDB_TIMEOUT = 300000

# INFLUXDB READ/WRITE API
INFLUXDB_BUCKET = "coinmarketcapBucket"
INFLUXDB_BATCH_SIZE = 800
INFLUXDB_MEASUREMENT = "dailyCandleSticks"
INFLUXDB_WRITE_PRECISION = WritePrecision.NS

#####################################################################################################################################################
#####################################################################################################################################################

################################################################ FUNCTIONS ##########################################################################
#####################################################################################################################################################
```

```
38
39  def datetime_to_unix_ns(timestamp):
40
41      unix_timestamp = datetime.datetime.strptime(timestamp, "%Y-%m-%d %H:%M:%S").timestamp()*1e9
42
43      return unix_timestamp
44
45  def dataframe_to_influx(dataframe, url, token, org, timeout, batch_size, bucket, measurement_name, dataframe_tag_columns, write_precision):
46
47      '''
48      Takes a dataframe and writes its content onto InfluxDB using batches.
49
50      dataframe: pandas dataframe with a time index in ns to be written to InfluxDB
51      url: endopoint for writing to InfluxDB
52      token: token for connecting to InfluxDB
53      org: organization for connecting to InfluxDB
54      timeout: int timeout for connection to InfluxDB in s
55      batch_size: int numer of line protocol to be sent at a time
56      bucket: str bucket name
57      measurment_name: str measurment name
58      dataframe_tag_columns: list list with column names of the dataframe to be taken as tags
59      '''
60
61      with InfluxDBClient(url=url, token=token, org=org, timeout=timeout, enable_gzip=True) as client:
62
63                  with client.write_api(write_options=WriteOptions(batch_size=batch_size)) as write_api:
64
65                      write_api.write(bucket=bucket, record=dataframe, data_frame_measurement_name=measurement_name,
66                      data_frame_tag_columns=dataframe_tag_columns,
67                          write_precision=write_precision)
68
69                      log.info(f"Data for cryptocurrency {dataframe['symbol']} has been correctly written to InfluxDB")
70
71  ################################################################################################################################################
72  ################################################################################################################################################
73
74  ############################################################ DATA INGESTION ###################################################################
75  ################################################################################################################################################
76
77  symbol_daily_data_df = pd.read_csv("coinmarketcap_ETH_daily_market_data.csv")
78
79  ################################################################################################################################################
80  ################################################################################################################################################
81
82  ########################################################### DATA PREPROCESSING #################################################################
83  ################################################################################################################################################
84
85  symbol_daily_data_df.drop(columns=['SNo', 'Name'], inplace=True)
86  symbol_daily_data_df.rename(columns={"Symbol": "symbol", "Date":"time", "High":"high", "Low":"low", "Open":"open", "Close":"close",
87  "Volume":"volume", "Marketcap":"marketcap"}, inplace=True)
88  symbol_daily_data_df["time"] = symbol_daily_data_df["time"].apply(lambda timestamp: datetime_to_unix_ns(timestamp))
89  symbol_daily_data_df = symbol_daily_data_df.astype({'time': 'datetime64[ns]','open': float, 'high': float, 'low': float, 'close': float,
90  'volume': float,'marketcap': float})
91  symbol_daily_data_df.set_index('time', inplace=True)
92
93  ################################################################################################################################################
94  ################################################################################################################################################
95
96  ############################################################ DATA WRITING ######################################################################
97  ################################################################################################################################################
98
99  dataframe_to_influx(dataframe=symbol_daily_data_df, url=INFLUXDB_URL, token=INFLUXDB_TOKEN, org=INFLUXDB_ORG, timeout=INFLUXDB_TIMEOUT,
100 batch_size=INFLUXDB_BATCH_SIZE, bucket=INFLUXDB_BUCKET, measurement_name=INFLUXDB_MEASUREMENT, dataframe_tag_columns=['symbol'],
101 write_precision=INFLUXDB_WRITE_PRECISION)
```

102
103 ##
104 ##

Apéndice B

Script para consulta a InfluxDB

```python
from influxdb_client import InfluxDBClient
import numpy as np
import pandas as pd
import datetime
import os

############################################################ PARAMETERS ###########################################################################
###################################################################################################################################################

# INFLUXDB CLIENT
INFLUXDB_URL = os.getenv('INFLUXDB_URL')
INFLUXDB_TOKEN = os.getenv('INFLUXDB_TOKEN')
INFLUXDB_ORG = os.getenv('INFLUXDB_ORG')
INFLUXDB_TIMEOUT = 300000

# INFLUXDB READ/WRITE API
INFLUXDB_BUCKET = "coinmarketcapBucket"
INFLUXDB_MEASUREMENT = "dailyCandleSticks"

###################################################################################################################################################
###################################################################################################################################################

############################################################ FUNCTIONS ############################################################################
###################################################################################################################################################

def basic_query(bucket, measurement, start_date, stop_date):
    query = f'''
      from(bucket: "{bucket}")
      |> range(start: {start_date}, stop: {stop_date})
      |> filter(fn: (r) => r["_measurement"] == "{measurement}")
      |> pivot(
        rowKey:["_time"],
        columnKey: ["_field"],
        valueColumn: "_value"
        )
      |> drop(columns: ["_start", "_stop","_measurement"])
    '''

    query = query + '  |> rename(columns: {_time: "datetime"})'

    return query
```

```
42
43  def query_influxdb(query, url, token, org, timeout, debug=False):
44
45      '''
46      returns pandas dataframe
47      '''
48
49      with InfluxDBClient(url=url, token=token, org=org, timeout=timeout, enable_gzip=True, debug=debug) as client:
50
51          result_df = client.query_api().query_data_frame(org=org, query=query)
52          result_df.drop(columns = ['result', 'table'], inplace = True)
53      return result_df
54
55  def date_to_iso_date(date):
56      '''
57      date: str Date with format "%Y-%m-%d, eg. 1985-05-27
58      returns: ISO 8601 date in string format
59      '''
60      timestamp = date + " 00:00:00"
61
62      isodate = datetime.datetime.strptime(timestamp, "%Y-%m-%d %H:%M:%S").replace(tzinfo=datetime.timezone.utc)
63      isodate = isodate.isoformat('T', 'microseconds')
64      return isodate
65
66  ##########################################################################################################################################
67  ##########################################################################################################################################
68
69  start_date = date_to_iso_date("2015-08-09")
70  stop_date = date_to_iso_date("2021-07-11")
71  query = basic_query(INFLUXDB_BUCKET, INFLUXDB_MEASUREMENT, start_date, stop_date)
72
73  eth_df = query_influxdb(query, INFLUXDB_URL, INFLUXDB_TOKEN, INFLUXDB_ORG, INFLUXDB_TIMEOUT)
74
75  ############################################################ DATA RETRIEVAL ##############################################################
76  ##########################################################################################################################################
77
78  start_date = date_to_iso_date("2015-08-09")
79  stop_date = date_to_iso_date("2021-07-11")
80  query = basic_query(INFLUXDB_BUCKET, INFLUXDB_MEASUREMENT, start_date, stop_date)
81
82  eth_df = query_influxdb(query, INFLUXDB_URL, INFLUXDB_TOKEN, INFLUXDB_ORG, INFLUXDB_TIMEOUT)
83
84  ##########################################################################################################################################
85  ##########################################################################################################################################
```

Bibliografía

[1] T. M. T. Do, J. Blom, and D. Gatica-Perez, "Smartphone Usage in the Wild: A Large-Scale Analysis of Applications and Context," in *Proceedings of the 13th International Conference on Multimodal Interfaces*, ICMI '11, (New York, NY, USA), pp. 353–360, Association for Computing Machinery, 2011.

[2] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, "A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT)," in *2015 Internet Technologies and Applications (ITA)*, pp. 219–224, 2015.

[3] K. Berg, D. T. Seymour, and R. Goel, "History Of Databases," *International Journal of Management & Information Systems (IJMIS)*, vol. 17, p. 29, 6 2012.

[4] A. Bader, O. Kopp, and M. Falkenthal, "Survey and Comparison of Open Source Time Series Databases," in *Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband* (B. Mitschang, D. Nicklas, F. Leymann, H. Schöning, M. Herschel, J. Teubner, T. Härder, O. Kopp, and M. Wieland, eds.), (Bonn), pp. 249–268, Gesellschaft für Informatik e.V., 2017.

[5] D. Schmidt, A. K. Dittrich, W. Dreyer, and R. Marti, "Time Series, a Neglected Issue in Temporal Database Research?," in *Recent Advances in Temporal Databases* (J. Clifford and A. Tuzhilin, eds.), (London), pp. 214–232, Springer London, 1995.

[6] J. Y. Lee and R. A. Elmasri, *Database Modeling and Implementation Techniques for Time-Series Data*. PhD thesis, 1998.

[7] A. Gunasekaran, H. B. Marri, R. E. McGaughey, and M. D. Nebhwani, "E-commerce and its impact on operations management," *International Journal of Production Economics*, vol. 75, no. 1, pp. 185–197, 2002.

[8] D. C. Chou and A. Y. Chou, "A Guide to the Internet Revolution in Banking," *Information Systems Management*, vol. 17, no. 2, pp. 47–53, 2000.

[9] Y. Li, J. Lee, and B. Cude, "Intention To Adopt Online Trading: Identifying The Future Online Traders," *Journal of Financial Counseling and Planning*, vol. 13, 6 2002.

[10] A. F. Bariviera and I. Merediz Sola, "Where do we stand in cryptocurrencies economic research? A survey based on hybrid analysis," *Journal of Economic Surveys*, vol. 35, pp. 377–407, 1 2021.

[11] Y. Zhang, S. Chan, J. Chu, and S. Nadarajah, "Stylised facts for high frequency cryptocurrency data," *Physica A: Statistical Mechanics and its Applications*, vol. 513, pp. 598–612, 2019.

[12] J. Chu, S. Chan, and Y. Zhang, "High frequency momentum trading with cryptocurrencies," *Research in International Business and Finance*, vol. 52, p. 101176, 2020.

[13] L. Yarovaya and D. Zięba, "Intraday volume-return nexus in cryptocurrency markets: Novel evidence from cryptocurrency classification," *Research in International Business and Finance*, vol. 60, p. 101592, 2022.

[14] S. Alonso-Monsalve, A. L. Suárez-Cetrulo, A. Cervantes, and D. Quintana, "Convolution on neural networks for high-frequency trend prediction of cryptocurrency exchange rates using technical indicators," *Expert Systems with Applications*, vol. 149, p. 113250, 2020.

[15] A. A. Petukhina, R. C. G. Reule, and W. K. Härdle, "Rise of the machines? Intraday high-frequency trading patterns of cryptocurrencies," *The European Journal of Finance*, vol. 27, no. 1-2, pp. 8–30, 2021.

[16] R. S. Tsay, *Analysis of financial time series*. Wiley series in probability and statistics, Hoboken, NJ: Wiley-Interscience, 2. ed. ed., 2005.

[17] P. Dix, "The Importance of Time Series Database — InfluxData Paper," 2021.

[18] T. Schlossnagle, "Monitoring in a DevOps World," *Commun. ACM*, vol. 61, pp. 58–61, 2 2018.

[19] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu, "TencentRec: Real-Time Stream Recommendation in Practice," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, (New York, NY, USA), pp. 227–238, Association for Computing Machinery, 2015.

[20] A. Meyer, D. Zverinski, B. Pfahringer, J. Kempfert, T. Kuehne, S. H. Sündermann, C. Stamm, T. Hofmann, V. Falk, and C. Eickhoff, "Machine learning for real-time prediction of complications in critical care: a retrospective study," *The Lancet Respiratory Medicine*, vol. 6, no. 12, pp. 905–914, 2018.

[21] A. Thennakoon, C. Bhagyani, S. Premadasa, S. Mihiranga, and N. Kuruwitaarachchi, "Real-time Credit Card Fraud Detection Using Machine Learning," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 488–493, 2019.

[22] M. Ait Said and A. Hajami, "AI Methods Used for Real-Time Clean Fraud Detection in Instant Payment," in *Proceedings of the 13th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2021)* (A. Abraham, A. Engelbrecht, F. Scotti, N. Gandhi, P. Manghirmalani Mishra, G. Fortino, V. Sakalauskas, and S. Pllana, eds.), (Cham), pp. 249–257, Springer International Publishing, 2022.

[23] A. Mashrur, W. Luo, N. A. Zaidi, and A. Robles-Kelly, "Machine Learning for Financial Risk Management: A Survey," *IEEE Access*, vol. 8, pp. 203203–203223, 2020.

[24] H. Ziegler, M. Jenny, T. Gruse, and D. A. Keim, "Visual market sector analysis for financial time series data," in *2010 IEEE Symposium on Visual Analytics Science and Technology*, pp. 83–90, 2010.

[25] P. Grzesik and D. Mrozek, "Comparative Analysis of Time Series Databases in the Context of Edge Computing for Low Power Sensor Net-

works," in *Computational Science – ICCS 2020* (V. V. Krzhizhanovskaya, G. Zavodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, eds.), pp. 371–383, Springer International Publishing, 2020.

[26] W. McKinney and others, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, 2010.

[27] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[28] Taylor G. Smith and others, "pmdarima," 2017.

[29] K. Sheppard, S. Khrapov, G. Lipták, mikedeltalima, R. Capellini, Hugle, esvhd, A. Fortin, JPN, W. Li, A. Adams, jbrockmendel, M. Rabba, M. E. Rose, T. Rochette, U. Leo, X. RENE-CORAIL, and syncoding, "bashtage/arch: Release 5.0.1," 7 2021.

[30] P. Katsiampa, "Volatility estimation for Bitcoin: A comparison of GARCH models," *Economics Letters*, vol. 158, pp. 3–6, 9 2017.

[31] N. A. Binti Omar and F. A. Halim, "Modelling volatility of Malaysian stock market using garch models," in *2015 International Symposium on Mathematical Sciences and Computing Research (iSMSC)*, pp. 447–452, 2015.

[32] H. Sun, D. Yan, N. Zhao, and J. Zhou, "Empirical investigation on modeling solar radiation series with ARMA–GARCH models," *Energy Conversion and Management*, vol. 92, pp. 385–395, 2015.

[33] A. R. S. Parmezan, V. M. A. Souza, and G. E. Batista, "Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model," *Information Sciences*, vol. 484, pp. 302–337, 2019.

[34] R. P. Masini, M. C. Medeiros, and E. F. Mendes, "Machine learning advances for time series forecasting," *Journal of Economic Surveys*, vol. n/a, no. n/a.