Universidad Autónoma de Querétaro
división de investigación y posgrado

Facultad de Ingeniería
Maestría en Ciencias en Inteligencia Artificial

Machine Learning

Evaluación Final: Series de Tiempo y Predicción de Precios con modelo ARIMA

Estudiante:
Alejandro Daniel
MATÍAS PACHECO

Profesor:
Dr. Marco Antonio
ACEVES FERNÁNDEZ

Querétaro, Qro. 1 de diciembre de 2023

Índice

Índice de figuras

1. Introducción

En el ámbito de la ciencia de datos y machine learning, comprender y saber aprovechar las capacidades predictivas empleando series de tiempo es indispensable. A medida que las industrias generan vastas cantidades de datos temporales, la capacidad para reconocer patrones, extraer ideas y prever tendencias futuras se ha convertido en una base para la toma de decisiones informada.

Este trabajo se enfoca en una revisión del análisis de series temporales y la predicción de precios, abordando las complejidades que conlleva trabajar con datos variantes en el tiempo. Actualmente existe una gran variedad de algoritmos y métodos de machine learning para predecir precios, desde regresiones lineales hasta Transformes y Redes Neuronales, sin embargo, uno de los más utilizados históricamente es ARIMA, siendo el que se tratará en las páginas siguientes, particularmente en el contexto de predecir precios en el área de los derivados lácteos.

2. Justificación

La predicción de series temporales, especialmente en el ámbito de la predicción de precios, es un área vital de estudio con relevancia directa para industrias como finanzas, agricultura y manufactura, especialmente para un profesional en el área de ciencia de datos o machine learning.

Con el fin de conseguir una comprensión profunda del tema, en este trabajo se aborda la implementación de un modelo de Media Móvil Integrada AutoRegresiva, mejor conocido como ARIMA, empleando únicamente librerías base como numpy y pandas. Al desglosar los componentes de este modelo, se obtendrá una comprensión mayor de los procesos involucrados en el desarrollo de modelos predictivos para datos dependientes del tiempo.

La base de datos a emplear, derivados lacteos, cuyas características son la volatilidad y estacionalidad en los precios, plantea un desafío interesante, convirtiéndolo en un excelente escenario para mostrar la adaptabilidad y dificultades a enfrentar al utilizar técnicas de pronóstico de series temporales.

En este trabajo se busca proporcionar una comprensión integral del algoritmo ARIMA y su aplicación mediante el uso de un conjunto de datos específico. A través del análisis estadístico de los datos, y su posterior interacción con el algoritmo ARIMA, se abordarán temas como limpieza de datos, calidad de datos, imputación, visualización y finalmente predicción.Es por esto que a continuación se describe este algoritmo y se presentan las funciones escritas en python, así como su aplicación en la base de datos de derivados lácteos, la cuál pretende predecir el precio en el futuro cercado de dichos consumibles.

3. Desarrollo

A continuación se presenta la descripción del modelo ARIMA (Media Móvil Integrada AutoRegresiva), una herramienta fundamental en la predicción de series temporales. Se abordará en detalle la estructura y los componentes del modelo, destacando su capacidad para modelar patrones temporales y realizar pronósticos precisos. Además, se presentará la base de datos seleccionada para este estudio, que comprende series temporales relacionadas con derivados lácteos, como la lactosa y el suero. Estas series de tiempo proporcionarán la base sobre la cual se aplicará el modelo ARIMA, explorando las complejidades y características de la evolución temporal de estos productos lácteos. Se llevará a cabo la implementación práctica en python de dicho modelo desde cero, permitiendo una comprensión profunda y práctica de cada paso, desde la preparación de los datos hasta la evaluación de las predicciones.

3.1. Series de Tiempo

Los datos de series de tiempo son una serie de puntos de datos u observaciones registradas en intervalos de tiempo diferentes o regulares. En general, una serie de tiempo es una secuencia de puntos de datos tomados en intervalos de tiempo igualmente espaciados. La frecuencia de los puntos de datos registrados puede ser horaria, diaria, semanal, mensual, trimestral o anual. La predicción de series de tiempo es el proceso de utilizar un modelo estadístico para predecir valores futuros de una serie de tiempo en función de resultados pasados [1].

3.1.1. Propiedades de las series de tiempo

- ✓ **Tendencia:** El comportamiento lineal creciente o decreciente de la serie a lo largo del tiempo. Una tendencia puede ser creciente (hacia arriba), decreciente (hacia abajo) u horizontal (estacionaria).
- ✓ **Estacionalidad:** Los patrones o ciclos de comportamiento que se repiten a lo largo del tiempo.
- ✓ **Descomposición de ETS:** La descomposición de ETS se utiliza para separar diferentes componentes de una serie de tiempo. El término ETS significa error, tendencia y estacionalidad.
- ✓ **Estacionaridad:** Muestra el valor medio de la serie que permanece constante durante el período de tiempo. Si los efectos pasados se acumulan y los valores aumentan hacia el infinito, entonces no se cumple la estacionariedad.
- ✓ **Diferenciación:** La diferenciación se utiliza para hacer que la serie sea estacionaria y para controlar las autocorrelaciones.
- ✓ **Dependencia:** Se refiere a la asociación de dos observaciones de la misma variable en períodos de tiempo anteriores.

- ✓ **Ruido:** La variabilidad en las observaciones que el modelo no puede explicar.
- ✓ **Autocorrelación:** La autocorrelación es la similitud entre las observaciones en función del desfase temporal entre ellas.

El análisis de series temporales es una disciplina fundamental en la exploración y comprensión de datos que evolucionan en función del tiempo. Para modelar y prever patrones en estas series, se recurre a herramientas especializadas como los modelos AutoRegressivos (AR), de Media Móvil (MA) y la combinación de ambos, conocida como ARIMA (AutoRegressive Integrated Moving Average).

3.2. ARIMA

- ✓ **Modelos de Media Móvil (MA):**

  Los modelos MA se centran en la relación entre una observación y un término de error residual de observaciones anteriores. Se elige el orden del modelo MA (q) mediante el análisis de la función de autocorrelación parcial (PACF).

- ✓ **Autoregresivo (AR):** Un modelo autorregresivo (AR) predice el comportamiento futuro basado en el comportamiento pasado. Se utiliza para pronosticar cuando existe alguna correlación entre los valores de una serie de tiempo y los valores que los preceden y los suceden.
- ✓ **Media Móvil Integrada Autorregresiva (ARIMA):** Los modelos de media móvil integrada autorregresiva, ARIMA, se encuentran entre los enfoques más utilizados para el pronóstico de series de tiempo. En realidad, es una clase de modelos que 'explica' una serie de tiempo dada basándose en sus propios valores pasados [2].

El modelo ARIMA es una potente herramienta en el análisis de series temporales que combina elementos autoregresivos (AR), de media móvil (MA) e integración para capturar las complejidades de los datos temporales. A continuación, se presenta una descripción detallada paso a paso del modelo ARIMA, junto con las ecuaciones fundamentales:

Paso 1: Estacionarización de la Serie Temporal

Antes de aplicar el modelo ARIMA, es esencial asegurarse de que la serie temporal sea estacionaria. Esto implica la eliminación de tendencias y variaciones sistemáticas. La estacionarización se logra mediante diferenciación, restando el valor actual del valor anterior.

$$Y_t' = Y_t - Y_{t-1}$$

Donde:

- ✓ Y_t' es la serie temporal diferenciada.
- ✓ Y_t es el valor en el tiempo t.

Paso 2: Identificación de Componentes AR y MA

La identificación de los componentes AR y MA implica el análisis de las funciones de autocorrelación (ACF) y autocorrelación parcial (PACF) de la serie temporal diferenciada.

AR (Autoregressive):

✓ Se selecciona el orden p basándose en la PACF.

✓ La ecuación AR de orden p es:

$$Y'_t = \phi_1 Y'_{t-1} + \phi_2 Y'_{t-2} + \ldots + \phi_p Y'_{t-p} + \epsilon_t$$

Donde $\phi_1, \phi_2, \ldots, \phi_p$ son los coeficientes AR y ϵ_t es el término de error.

MA (Moving Average):

✓ Se selecciona el orden q basándose en la ACF.

✓ La ecuación MA de orden q es:

$$Y'_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q}$$

Donde $\theta_1, \theta_2, \ldots, \theta_q$ son los coeficientes MA.

Paso 3: Integración

La integración implica revertir la diferenciación realizada en el Paso 1 para obtener predicciones en la escala original.

$$\hat{Y}_t = Y_{t-1} + Y'_t$$

Donde $\hat{Y}_t$ es la predicción revertida y Y'_t es la serie temporal diferenciada.

Paso 4: Formulación del Modelo ARIMA Completo

La formulación final del modelo ARIMA es la combinación de los componentes AR, MA e integración:

$$\hat{Y}_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q}$$

Este modelo permite realizar pronósticos precisos al capturar tanto la dependencia temporal autoregresiva como la influencia de términos de media móvil en la serie temporal. A continuación, se presentan las funciones escritas en python para implementar el modelo ARIMA (Figuras 1 a 12).

```python
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def evaluate_forecast(y,pred):

    results = pd.DataFrame({'r2_score':r2_score(y, pred),
                            }, index=[0])
    results['mean_absolute_error'] = mean_absolute_error(y, pred)
    results['median_absolute_error'] = median_absolute_error(y, pred)
    results['mse'] = mean_squared_error(y, pred)
    results['msle'] = mean_squared_log_error(y, pred)
    results['mape'] = mean_absolute_percentage_error(y, pred)
    results['rmse'] = np.sqrt(results['mse'])
    return results
```

Figura 1: Funciones para evaluar el rendimiento del modelo, basándose en distintos tipos de errores.

```python
# Definición de la etapa autogregresiva con una media cero
def ar_process(eps, phi):

    # Inviertir el orden de phi y agregar un 1 para el actual eps_t
    phi = np.r_[1, phi][::-1]
    ar = eps.copy()
    offset = len(phi)
    for i in range(offset, ar.shape[0]):
        ar[i - 1] = ar[i - offset: i] @ phi
    return ar
```

Figura 2: Identificación de los componentes autoregresivos.

```python
def difference(x, d=1):
    if d == 0:
        return x
    else:
        x = np.r_[x[0], np.diff(x)]
        return difference(x, d - 1)

def undo_difference(x, d=1):
    if d == 1:
        return np.cumsum(x)
    else:
        x = np.cumsum(x)
        return undo_difference(x, d - 1)
```

Figura 3: Funciones para realizar la diferenciación y garantizar la estacionariedad.

```python
n = 500
eps = np.random.normal(size=n)


def lag_view(x, order):

    # Para cada valor X_i, crea una fila que retarda k valores: [X_i-1, X_i-2, ... X_i-k]

    y = x.copy()
    # Crea características al desplazar la ventana de tamaño `order` en un paso.
    # Así se obtiene una matriz 2D [[t1, t2, t3], [t2, t3, t4], ... [t_k-2, t_k-1, t_k]]
    x = np.array([y[-(i + order):][:order] for i in range(y.shape[0])])

    # Invertir la matriz (ya que se inicia por el final) y eliminar duplicados.
    # Se truncan las características [order -1:] y las etiquetas [order]
    # Esto representa el desplazamiento de las características con un paso de tiempo en comparación con las etiquetas
    x = np.stack(x)[::-1][order - 1: -1]
    y = y[order:]
    return x, y


#Definición del proceso media móvil MA(q) con una media cero

def ma_process(eps, theta):
    # eps: (array) Señal de ruido blanco.
    # theta: (array/ list) Parámetros del proceso.
    # Invierte el orden de theta ya que Xt, Xt-1, Xt-k en una matriz es Xt-k, Xt-1, Xt.
    theta = np.array([1] + list(theta))[::-1][:, None]
    eps_q, _ = lag_view(eps, len(theta))
    return eps_q @ theta
```

Figura 4: Identificación de los componentes de media móvil.

```python
def pearson_correlation(x, y):
    return np.mean((x - x.mean()) * (y - y.mean())) / (x.std() * y.std())

def acf(x, lag=40):
    # Determina factores de autocorrelación.
    # x: (array) Serie temporal.
    # lag: (int) Número de rezagos.

    return np.array([1] + [pearson_correlation(x[:-i], x[i:]) for i in range(1, lag)])

def bartletts_formula(acf_array, n):

    # Calcula el Error Estándar de un ACF con la fórmula de Bartlett.
    # acf_array: (array) Contiene factores de autocorrelación.
    # n: (int) Longitud de la secuencia original de la serie temporal.

    # El primer valor tiene autocorrelación consigo mismo, por lo que se omite ese valor.
    se = np.zeros(len(acf_array) - 1)
    se[0] = 1 / np.sqrt(n)
    se[1:] = np.sqrt((1 + 2 * np.cumsum(acf_array[1:-1]**2)) / n )
    return se
```

Figura 5: Correlación de Pearson y calculo del Error Estándar de un ACF con la fórmula de Bartlett.

```python
def plot_acf(x, alpha=0.05, lag=40):

    # x: (array)
    # alpha: (flt) Significancia estadística para el intervalo de confianza
    # lag: (int)

    acf_val = acf(x, lag)
    plt.figure(figsize=(16, 4))
    plt.vlines(np.arange(lag), 0, acf_val)
    plt.scatter(np.arange(lag), acf_val, marker='o')
    plt.xlabel('lag (rezago)')
    plt.ylabel('autocorrelación')

    # Determinar intervalo de confianza
    ci = stats.norm.ppf(1 - alpha / 2.) * bartletts_formula(acf_val, len(x))
    plt.fill_between(np.arange(1, ci.shape[0] + 1), -ci, ci, alpha=0.25)

def pacf(x, lag=40):

    # Función de autocorrelación parcial.
    # x: (array)
    # param lag: (int)

    y = []

    # La autocorrelación parcial necesita términos intermedios.
    # Por lo tanto, se inicia en el índice 3
    for i in range(3, lag + 2):
        backshifted = lag_view(x, i)[0]

        xt = backshifted[:, 0]
        feat = backshifted[:, 1:-1]
        xt_hat = LinearModel(fit_intercept=False).fit_predict(feat, xt)

        xt_k = backshifted[:, -1]
        xt_k_hat = LinearModel(fit_intercept=False).fit_predict(feat, xt_k)

        y.append(pearson_correlation(xt - xt_hat, xt_k - xt_k_hat))
    return np.array([1, acf(x, 2)[1]] +  y)
```

Figura 6: Cálculo de la autocorrelación parcial y función para graficar la autocorrelación simple.

```python
def plot_pacf(x, alpha=0.05, lag=40, title=None):

    # x: (array)
    # alpha: (flt) Significancia estadística para el intervalo de confianza.
    # lag: (int)

    pacf_val = pacf(x, lag)
    plt.figure(figsize=(16, 4))
    plt.vlines(np.arange(lag + 1), 0, pacf_val)
    plt.scatter(np.arange(lag + 1), pacf_val, marker='o')
    plt.xlabel('lag (rezago)')
    plt.ylabel('autocorrelación')

    # Determine confidence interval
    ci = stats.norm.ppf(1 - alpha / 2.) * bartletts_formula(pacf_val, len(x))
    plt.fill_between(np.arange(1, ci.shape[0] + 1), -ci, ci, alpha=0.25)
```

```python
def least_squares(x, y):
    return np.linalg.inv((x.T @ x)) @ (x.T @ y)
```

Figura 7: Función para graficar la autocorrelación parcial.

En seguida se definen las funciones para realizar la regresión lineal, la cuál es utilizada para estimar los coeficientes de los términos autorregresivos (AR) y de media móvil (MA) durante el proceso de construcción del modelo. En ARIMA, la regresión lineal es aplicada a los valores rezagados (lagged values) de la serie temporal para determinar cómo afectan estos valores pasados a la observación actual (Figura 8).

```python
class LinearModel:
    def __init__(self, fit_intercept=True):
        self.fit_intercept = fit_intercept
        self.beta = None
        self.intercept_ = None
        self.coef_ = None

    def _prepare_features(self, x):
        if self.fit_intercept:
            x = np.hstack((np.ones((x.shape[0], 1)), x))
        return x

    def fit(self, x, y):
        x = self._prepare_features(x)
        self.beta = least_squares(x, y)
        if self.fit_intercept:
            self.intercept_ = self.beta[0]
            self.coef_ = self.beta[1:]
        else:
            self.coef_ = self.beta

    def predict(self, x):
        x = self._prepare_features(x)
        return x @ self.beta

    def fit_predict(self, x, y):
        self.fit(x, y)
        return self.predict(x)
```

Figura 8: Implementación de regresión lineal.

```python
class ARIMA(LinearModel):
    def __init__(self, q, d, p):

        # Modelo ARIMA
        # q: (int) Orden del modelo MA.
        # p: (int) Orden del modelo AR.
        # d: (int) Número de veces que los datos deben ser diferenciados.

        super().__init__(True)
        self.p = p
        self.d = d
        self.q = q
        self.ar = None
        self.resid = None
```

Figura 9: Definición de la clase ARIMA e inicialización.

```python
def prepare_features(self, x):
    if self.d > 0:
        x = difference(x, self.d)

    ar_features = None
    ma_features = None

    # Determina las características y los términos epsilon para el proceso MA
    if self.q > 0:
        if self.ar is None:
            self.ar = ARIMA(0, 0, self.p)
            self.ar.fit_predict(x)
        eps = self.ar.resid
        eps[0] = 0

        # Llenar con ceros al principio ya que no hay residuos_t-k en el primer X_t
        ma_features, _ = lag_view(np.r_[np.zeros(self.q), eps], self.q)

    # Determinr las características del proceso AR
    if self.p > 0:
        # Llenar con ceros al principio ya que no hay X_t-k en el primer X_t
        ar_features = lag_view(np.r_[np.zeros(self.p), x], self.p)[0]

    if ar_features is not None and ma_features is not None:
        n = min(len(ar_features), len(ma_features))
        ar_features = ar_features[:n]
        ma_features = ma_features[:n]
        features = np.hstack((ar_features, ma_features))
    elif ma_features is not None:
        n = len(ma_features)
        features = ma_features[:n]
    else:
        n = len(ar_features)
        features = ar_features[:n]

    return features, x[:n]
```

Figura 10: Definición de clase ARIMA: Preparación de características.

```python
def fit(self, x):
    features, x = self.prepare_features(x)
    super().fit(features, x)
    return features

def fit_predict(self, x):

    # Entrenar y transformar entrada
    # x: (array) timeseries

    features = self.fit(x)
    return self.predict(x, prepared=(features))

def predict(self, x, **kwargs):

    # x: (array)
    # kwargs:
    # prepared: contiene las características: eps, x

    features = kwargs.get('prepared', None)
    if features is None:
        features, x = self.prepare_features(x)

    y = super().predict(features)
    self.resid = x - y

    return self.return_output(y)

def return_output(self, x):
    if self.d > 0:
        x = undo_difference(x, self.d)
    return x
```

Figura 11: Definición de clase ARIMA: Funciones de ajuste/entrenamiento del modelo.

```python
def forecast(self, x, n):

    # Predice series de tiempo
    # x: (array) Paso actual de la serie de tiempo.
    # n: (int) Número de pasos en el futuro (a predecir)

    features, x = self.prepare_features(x)
    y = super().predict(features)


    # Agregar n pasos de tiempo como ceros, debido a que los términos epsilon son desconocidos
    y = np.r_[y, np.zeros(n)]
    for i in range(n):
        feat = np.r_[y[-(self.p + n) + i: -n + i], np.zeros(self.q)]
        y[x.shape[0] + i] = super().predict(feat[None, :])
    return self.return_output(y)
```

Figura 12: Definición de clase ARIMA: Función para predecir a partir de nuevas entradas.

3.3. Dataset: Derivados Lacteos

La base de datos de derivados lácteos comprende una colección de atributos relacionados con diversas categorías de productos derivados de la leche. Cada entrada en la base de datos se caracteriza por la fecha de inicio del intervalo ('Week') y una serie de atributos que representan diferentes componentes y productos lácteos. Los atributos incluidos en la base de datos son los siguientes:

- ✓ **Week:** La fecha de inicio y fin del intervalo para cada entrada semanal en la base de datos.
- ✓ **Lactose:** Precio de la lactosa en dólares.
- ✓ **Whey central, Whey west, Whey east:** Precios del suero de leche en diferentes regiones geográficas, en dólares.
- ✓ **NDPSR whey avg, CME whey avg:** Medias de precios del suero de leche según diferentes fuentes, NDPSR (National Dairy Products Sales Report) y CME (Chicago Mercantile Exchange), en dólares.
- ✓ **NDPSR NFDM:** Precio de la leche desnatada en polvo según NDPSR, en dólares.
- ✓ **34p WPC:** Precio del concentrado de proteínas de suero al 34
- ✓ **Nonfat west, Nonfat central east:** Precios de productos sin grasa en diferentes regiones geográficas, en dólares.
- ✓ **Sweet cream buttermilk central:** Precio del suero de mantequilla de crema dulce en una región específica, en dólares.
- ✓ **AA butter:** Precio de la mantequilla en dólares.
- ✓ **Cheese 40 blocks, Cheese barrel:** Precios de diferentes formatos o tipos de queso, posiblemente en bloques o barriles, en dólares.
- ✓ **Delac:** Precio de un producto lácteo específico, en dólares.

Dado que la calidad de la base de datos no era la adecuada, se implementó una función para realizar la limpieza y estandarizado de los datos, eligiendo y renombrando adecuadamente los atributos, así como procesándolos, de ser el caso. En la Figura 13 se presenta el formato original de la base de datos, mientras que en la Figura 14 se muestra la función implementada para limpiar los datos.

En la Figura 15 se presenta una muestra de la base de datos, previo a realizar la unión de los subconjuntos anuales en un solo dataframe. De igual forma, en la Figura 16 se ejemplifica parte del preprocesamiento de la base de datos, dado que en ciertos años fue necesario unir las columnas Nonfat central y Nonfat east para estandarizar los subconjuntos, integrándose

```python
excel_data = pd.read_excel('DMN Report 29.xlsx', sheet_name=None)

tabs = ['1996', '1997','1998','1999','2000','2001','2002','2003','2004','2005','2006','2007','2008','2009',
        '2010 - 2011','2012','2013','2014','2015','2016','2017','2018','2019','2020','2021','2022','2023']


# Diccionario para almacenar dfs
dfs = {}

for tab_name, tab_data in excel_data.items():
    dfs[tab_name] = tab_data
```

Figura 13: Importación de base de datos y formato original.

```python
def clean_data(dfs, tab, attrib):
    df = dfs[tab]

    # Eliminar primeras filas
    df_tab = df.iloc[4:]
    # Asignar nombres a columnas
    df_tab.columns = attrib

    # Limpiar dataset, eliminar filas 'AVG.'
    delete = 'AVG.'
    df_tab = df_tab[df_tab['Week'].astype(str).str.contains(delete) == False]

    # Limpiar dataset, eliminar filas NaN
    df_tab = df_tab.dropna(subset=['Week'])
    df_tab = df_tab.reset_index(drop=True)

    # Convertir la columna 'Week' al formato de fecha
    #df_tab['Inicio'] = pd.to_datetime(df_tab['Week'].str.split(' - ', expand=True)[0], format='%m/%d/%y')
    df_tab['Start_interval'] = pd.to_datetime(df_tab['Week'].str.extract(r'(\d{1,2}/\d{1,2}/\d{2})')[0], format='%m/%d/%y')

    # Obtener el número de semana del año
    df_tab['Year'] = df_tab['Start_interval'].dt.year
    df_tab['Month'] = df_tab['Start_interval'].dt.month
    df_tab['Week year'] = df_tab['Start_interval'].dt.isocalendar().week

    # Convertir atributos a float, excepto Week
    attrib.remove('Week')
    df_tab[attrib] = df_tab[attrib].apply(lambda col: pd.to_numeric(col.apply(str_to_float), errors='coerce'))

    # Retornar df resultante
    return df_tab
```

Figura 14: Función para limpieza y estandarización de datos.

así al atributo Nonfat central east, presente en la mayoría de los otros subconjuntos.

Finalmente, en la Figura 17 se muestra el código implementado para unir los subconjuntos anuales en un solo dataframe. Cabe mencionar que al pasar cada subconjunto en la función de limpieza de datos, la columna Week se utilizó para obtener cuatro atributos más: Start_interval, Year, Month y Week year, las cuáles serán de utilidad más adelante.

```
attri_2023 = ['Week', 'Lactose', 'Whey central', 'Whey west', 'Whey east', 'NDPSR whey avg', 'CME whey avg',
           'NDPSR NFDM', '34p WPC', 'Nonfat west', 'Nonfat central east', 'Sweet cream buttermilk central',
           'AA butter', 'Cheese 40 blocks']

df_2023 = clean_data(dfs, '2023', attri_2023)
df_2023
```

	Week	Lactose	Whey central	Whey west	Whey east	NDPSR whey avg	CME whey avg	NDPSR NFDM	34p WPC	Nonfat west	Nonfat central east	Sweet cream buttermilk central	AA butter	Cheese 40 blocks	Start_interval	Year	Month	Week year
0	1/2/23 - 1/6/23	0.4775	0.41000	0.43500	0.42750	0.4371	0.4088	1.4326	1.7275	1.40000	1.3800	1.350	2.4625	2.1151	2023-01-02	2023	1	1
1	1/9/23 - 1/13/23	0.4725	0.40250	0.43500	0.42625	0.4430	0.3575	1.4074	1.6775	1.36000	1.3025	1.335	2.4936	2.1145	2023-01-09	2023	1	2
2	1/16/23 - 1/20/23	0.4675	0.37250	0.41500	0.42500	0.4453	0.3300	1.3974	1.6775	1.33000	1.2850	1.300	2.4732	2.1274	2023-01-16	2023	1	3
3	1/23/23 - 1/27/23	0.4450	0.36000	0.40500	0.41125	0.4259	0.3245	1.3981	1.6250	1.28000	1.2300	1.250	2.4773	2.1269	2023-01-23	2023	1	4
4	1/30/23 - 2/3/23	0.4400	0.36000	0.38500	0.39500	0.4001	0.3620	1.2786	1.5750	1.21500	1.2200	1.225	2.4085	2.0891	2023-01-30	2023	1	5

Figura 15: Muestra del dataset, año 2023.

```
attri_1997 = ['Week', 'Lactose', 'Delac','Whey central', 'Whey west', 'Whey east', 'NDPSR whey avg',
              '34p WPC', 'Nonfat west', 'Nonfat central',  'Nonfat east', 'AA butter', 'Cheese 40 blocks',
              'Cheese barrel']

df_1997 = clean_data(dfs, '1997', attri_1997)
df_1997['Nonfat central east'] = df_1997[['Nonfat central', 'Nonfat east']].mean(axis=1)
df_1997.drop(['Nonfat central', 'Nonfat east'], axis=1, inplace=True)

df_1997
```

Figura 16: Estandarización de datos.

```
# Concatenar los dataframes de todos los años
# Crear una lista de dataFrames desde 2023 hasta 1996
dfs = [globals()[f"df_{year}"] for year in range(2023, 1995, -1)]

# Concatenar dataFrames
df_all = pd.concat(dfs, ignore_index=True)
df_all
```

Week	Lactose	Whey central	Whey west	Whey east	NDPSR whey avg	CME whey avg	NDPSR NFDM	34p WPC	Nonfat west	Nonfat central east	Sweet cream buttermilk central	AA butter	Cheese 40 blocks	Start_interval	Year	Month	Week year	Chees barr
1/2/23 - 1/6/23	0.4775	0.4100	0.4350	0.42750	0.4371	0.4088	1.4326	1.7275	1.4000	1.3800	1.350	2.4625	2.1151	2023-01-02	2023	1	1	Na
1/9/23 - 1/13/23	0.4725	0.4025	0.4350	0.42625	0.4430	0.3575	1.4074	1.6775	1.3600	1.3025	1.335	2.4936	2.1145	2023-01-09	2023	1	2	Na
1/16/23 - 1/20/23	0.4675	0.3725	0.4150	0.42500	0.4453	0.3300	1.3974	1.6775	1.3300	1.2850	1.300	2.4732	2.1274	2023-01-16	2023	1	3	Na

Figura 17: Concatenado de los subconjuntos anuales.

3.3.1. Análisis e imputación de datos

Como primer paso y con el fin de conocer cómo se comportan los atributos en el tiempo, estos se graficaron empleando el atributo 'Start_interval'. En la Figura 18) se presenta una gráfica general, notando que sus valores fluctúan constantemente.

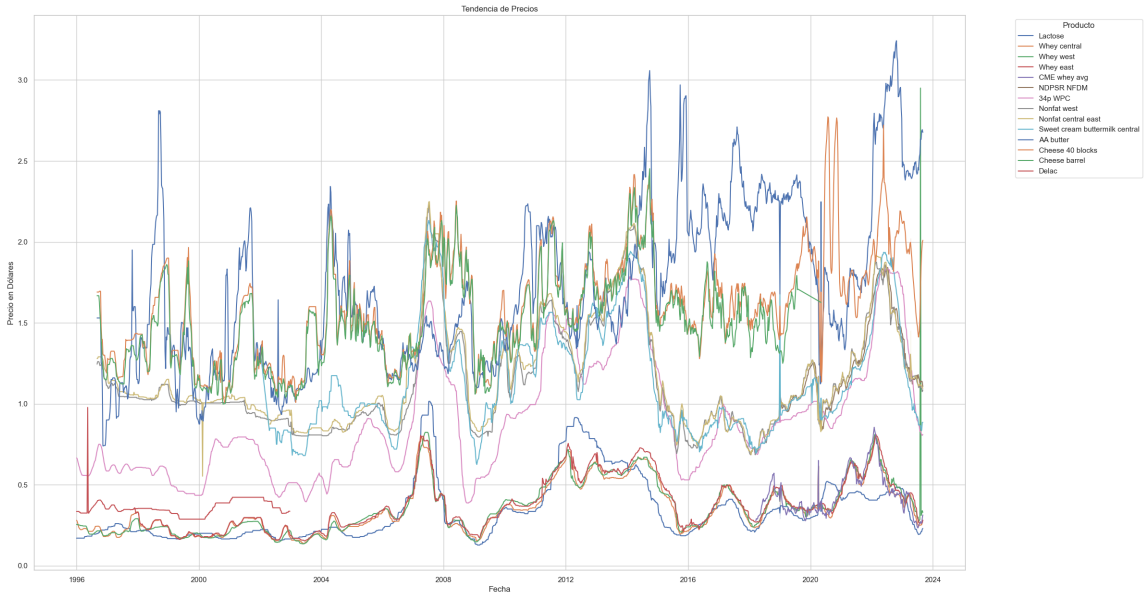


Figura 18: Concatenado de los subconjuntos anuales.

Siguiendo las formulas definidas en trabajos previos, se procedió a realizar el análisis de la base de datos, revisando el número de datos faltantes (Figura 19) y realizando una imputación por media de clase (por año y mes) como solución a dicho problema (Figura 20).

```
# Valores faltantes
nan_count_2 = df_imput.isna().sum()
nan_count_2 = pd.DataFrame(nan_count_2.items(), columns=['Atributo', 'NaN']).T
nan_count_2
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Atributo	Start_interval	Year	Month	Week year	Lactose	Whey central	Whey west	Whey east	NDPSR whey avg	CME whey avg	NDPSR NFDM	34p WPC	Nonfat west	Nonfat central east	Sweet cream buttermilk central	AA butter	Cheese 40 blocks	Cheese barrel	Delac
NaN	0	0	0	0	0	0	0	96	262	1152	1195	0	35	35	321	35	35	240	1075

Figura 19: Análisis de datos faltantes.

```
# Imputación de media por clase ('Year', 'Month') usando una función lambda
for column in df_imput.columns[4:]:
    df_imput[column] = df_imput.groupby(['Year', 'Month'])[column].transform(lambda x: x.fillna(x.mean()))

df_imput
```

	Start_interval	Year	Month	Week year	Lactose	Whey central	Whey west	Whey east	NDPSR whey avg	CME whey avg	NDPSR NFDM	34p WPC	Nonfat west	Nonfat central east	Sweet cream buttermilk central	AA butter	Cheese 40 blocks	Chees barr
0	2023-09-04	2023	9	36	0.230	0.2800	0.3300	0.28125	0.2761	0.3131	1.1066	0.8100	1.0850	1.0950	0.885	2.6765	2.0094	Na
1	2023-08-28	2023	8	35	0.225	0.2700	0.3400	0.26375	0.2658	0.2930	1.1365	0.8050	1.0750	1.1050	0.885	2.6930	1.9937	Na
2	2023-08-21	2023	8	34	0.210	0.2525	0.3200	0.26375	0.2683	0.2630	1.1395	0.8100	1.0950	1.1100	0.835	2.6844	1.9374	Na
3	2023-08-14	2023	8	33	0.210	0.2525	0.3100	0.26125	0.2630	0.2645	1.1357	0.8200	1.0975	1.1150	0.845	2.6328	1.8492	Na

Figura 20: Imputación por media de clase.

3.4. Entrenamiento del modelo

En este punto, el dataset está listo para pasar a la etapa de entrenamiento, sin embargo, es necesario mencionar que el análisis de datos no ha terminado, pues otras propiedades de las series de tiempo son revisadas más adelante. En la Figura 21 se muestra la elección de un atributo para la predicción, es este caso es 'Lactose', aunque se puede seleccionar alguno de los otros. Posterior a la selección de atributo a predecir, se separan los datos tomando el atributo elegido y la columna 'Start_interval', que es renombrada a Date. Así mismo se realiza una última revisión de valores faltantes, los cuáles se eliminan. Finalmente, 'Week' se convierte en index del dataframe.

```
atri = 'Lactose'
raw_data = df_imput[['Start_interval', atri]].copy()
raw_data.columns = ['Date', atri]
raw_data['Date'] = pd.to_datetime(raw_data['Date'])
raw_data.drop_duplicates(subset='Date')
raw_data
```

	Date	Lactose
0	2023-09-04	0.230
1	2023-08-28	0.225

Figura 21: Elección de atributo a predecir y preparación de dataframe.

En seguida, se obtiene la gráfica del atributo en el dominio temporal, así como su distribución e histograma, de esta forma podemos observar cómo se comporta (Figuras 22 y 23).

```
# Graficar serie de tiempo
data[atri].plot(figsize=(15, 6))
plt.show()
```

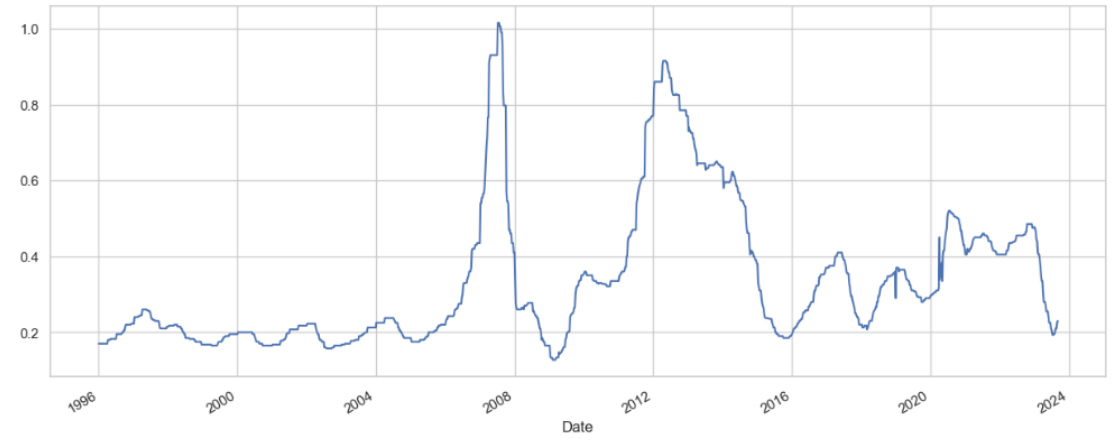


Figura 22: Gráfica del atributo en el dominio temporal.

En seguida, utilizando la librería statsmodel se realiza la descomposición de la serie temporal. La descomposición de series de tiempo es una tarea estadística que deconstruye una

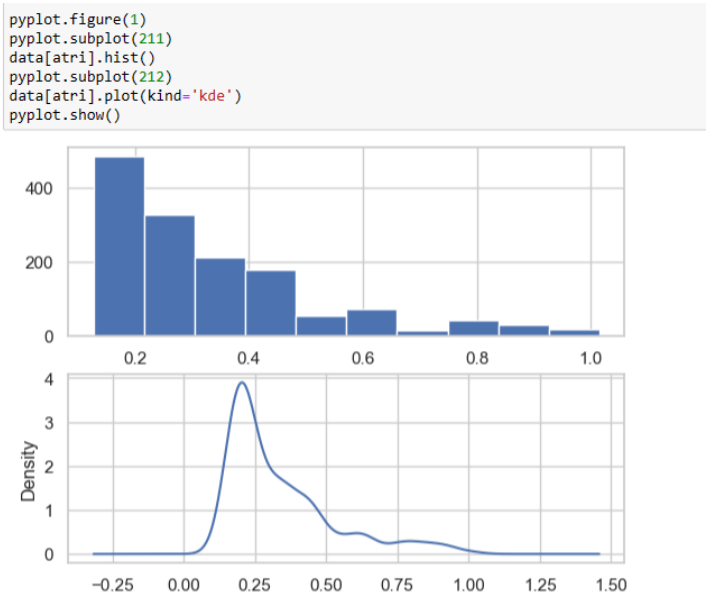


Figura 23: Histograma y distribución del atributo.

serie de tiempo en varios componentes, cada uno de los cuales representa una de las categorías subyacentes de patrones. Con los modelos de estadísticas es posible ver los componentes de tendencia, estacionales y residuales de los datos (Figura 24). De dicha Figura se observa que el atributo Lactose no presenta una tendencia de crecimiento clara, pues oscila entre valores altos y bajos en el periodo 2005 a 2026. De igual forma, tiene una estacionalidad anual y presenta una gran cantidad de ruido.

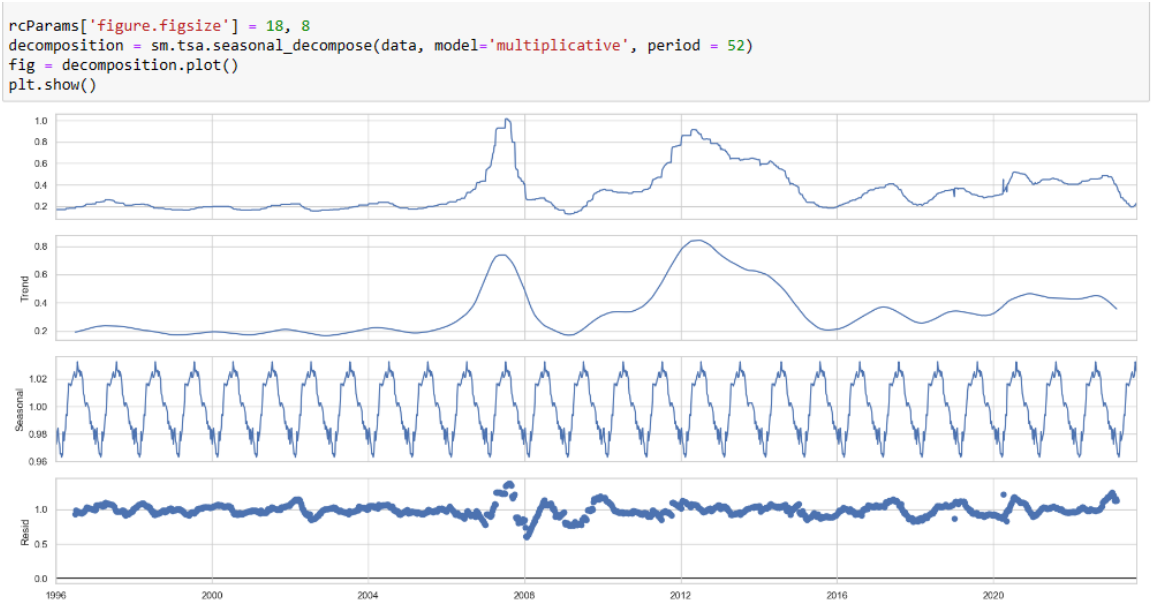


Figura 24: Tendencia, estacionalidad y residual (ruido) de la serie temporal.

Habiendo evaluado las características de la serie, se procede a realizar el primer paso del entrenamiento del modelo ARIMA, determinar si la serie es estacionaria o no, dado que solo se necesita realizar el diferenciado si la serie no es estacionaria, es decir, d=0. Para esto se emplea la prueba de Dickey Fuller Aumentada (ADF) (Figura 25). La hipótesis nula de la prueba ADF es que la serie temporal no es estacionaria. Entonces, si el valor p de la prueba es menor que el nivel de significancia (0.05), se rechaza la hipótesis nula y se infiere que la serie temporal es estacionaria.

En este caso, el valor de p es menor a 0.05, por lo que se define d = 0 (Figura 26).

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from numpy import log

result = adfuller(data[atri].dropna())
print('Prueba Dickey-Fuller Aumentada: %f' % result[0])
print('p-valor: %f' % result[1])
```

```
Prueba Dickey-Fuller Aumentada: -3.320702
p-valor: 0.013978
```

Figura 25: Prueba de Dickey Fuller Aumentada.

Si la serie es estacionaria, se define d = 0

```
tol = 0.03  # Tolerancia

if(result[1] < (0.05 + tol)):
    print("La serie es estacionaria, no se debe diferenciar, por lo tanto d = 0")
    d = 0

    # Crear un solo canvas con 1 fila y 2 columnas
    fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))

    # Serie original
    data[atri].plot(ax=axs[0])
    axs[0].set_title('Serie Original')

    # Función de autocorrelación
    plot_acf(data[atri], ax=axs[1])
    axs[1].set_title('Autocorrelación')

    plt.tight_layout()
    plt.show()

    # Copiar data en nuevo df
    data_d = data.copy()
```

```
La serie es estacionaria, no se debe diferenciar, por lo tanto d = 0
```

Figura 26: Definición de valor de d, la serie es estacionaria y d=0.

En caso de que la serie hubiera resultado ser no estacionaria, se procedería a realizar el número de diferenciaciones necesarias, siendo el orden correcto el diferenciado mínimo necesario para obtener una serie casi estacionaria que fluctúa alrededor de una media definida y el gráfico de autocorrelación alcanza cero rápidamente. Si las autocorrelaciones son positivas para muchos rezagos (10 o más), entonces la serie requiere un diferenciado adicional. Por otro

lado, si la autocorrelación en el rezago 1 es demasiado negativa, entonces la serie probablemente está sobre-diferenciada. El código que se ejecuta en dicho caso se muestra en la Figura 27.

Si la serie no es estacionaria, se diferencía la serie para definir d

```
if(result[1] > (0.05+tol)):
    print("La serie no es estacionaria, es necesario diferenciar")

    # Crear un solo canvas con 3 filas y 2 columnas
    fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(12, 10))

    # Serie original
    data[atri].plot(ax=axs[0, 0])
    plot_acf(data[atri], ax=axs[0, 1])

    # Primera diferencia
    data[atri].diff().plot(ax=axs[1, 0])
    plot_acf(data[atri].diff().dropna(), ax=axs[1, 1])

    # Segunda diferencia
    data[atri].diff().diff().plot(ax=axs[2, 0])
    plot_acf(data[atri].diff().diff().dropna(), ax=axs[2, 1])

    plt.tight_layout()
    plt.show()

    result1 = adfuller(data[atri].diff().dropna())
    print("Primera diferenciación")
    print('Prueba Dickey-Fuller Aumentada: %f' % result1[0])
    print('p-valor: %f' % result1[1])

    result2 = adfuller(data[atri].diff().diff().dropna())
    print("\nSegunda diferenciación")
    print('Prueba Dickey-Fuller Aumentada: %f' % result2[0])
    print('p-valor: %f' % result2[1])

    #Definir el valor de d y copiar data nuevo df
    d = 1     # Depende del grado de diferenciación
    data_d = data.diff().copy()   # Depende del grado de diferenciación
    print("\nEl valor de d es:", d)
```

Figura 27: Definición de valor de d realizando la diferenciación de la serie temporal.

El siguiente paso consiste en identificar si el modelo necesita algún término AR. Se puede determinar el número necesario de términos AR inspeccionando el gráfico de Autocorrelación Parcial (PACF). La autocorrelación parcial se puede entender como la correlación entre la serie y su rezago, después de excluir las contribuciones de los rezagos intermedios. Entonces, el PACF comunica la correlación pura entre un rezago y la serie. De esta manera, se sabe si ese rezago es necesario en el término AR o no. El orden del término AR se determina según el número de rezagos significativos en la gráfica de la Autocorrelación parcial PACF. Si solo hay un rezago significativo, se podría seleccionar un modelo AR de orden 1. Si se observa que el primer rezago en la gráfica de la PACF es significativo y los demás rezagos no lo son, se puede elegir un componente AR de orden 1. La elección del valor de p se presenta en la Figura 28.

El último término a definir es q. Se puede observar la gráfica de ACF para definir el número de términos MA. Técnicamente, un término MA es el error de la predicción rezagada. El ACF indica cuántos términos MA se necesitan para eliminar cualquier autocorrelación en la serie estacionaria. Gráficamente, es el número de rezagos que se encuentran sobre el nivel

```python
# Crear un solo canvas con 3 filas y 2 columnas
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))

# Serie después de definir d
data_d[atri].plot(ax=axs[0])
axs[0].set_title('Serie Original')

# Función de autocorrelación
plot_pacf(data_d[atri], ax=axs[1])
axs[1].set_title('Autocorrelación parcial')

plt.tight_layout()
plt.show()
```

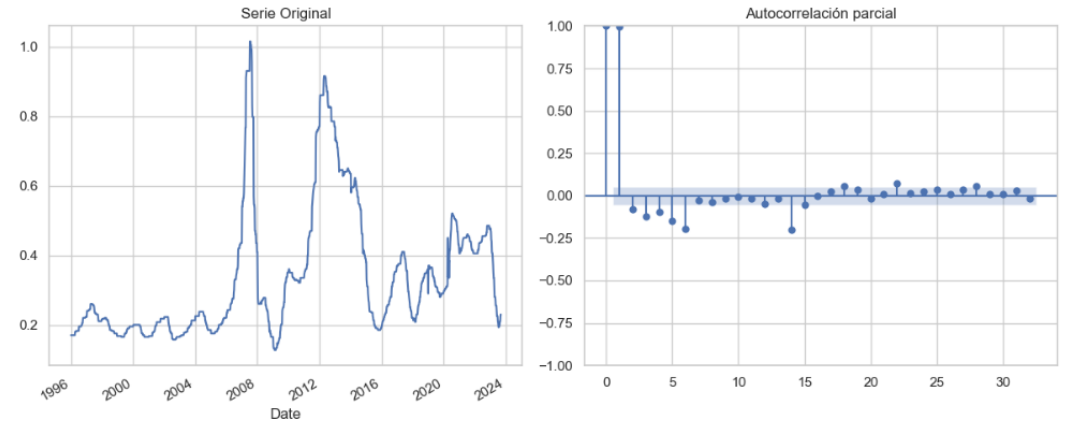


Figura 28: Definición del término AR o p.

de significancia. La elección del valor de q se presenta en la Figura 29.

```python
# Crear un solo canvas con 1 fila y 2 columnas
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))

# Serie original
data_p[atri].plot(ax=axs[0])
axs[0].set_title('Serie Original')

# Función de autocorrelación
plot_acf(data_p[atri], ax=axs[1])
axs[1].set_title('Autocorrelación simple')

plt.tight_layout()
plt.show()
```

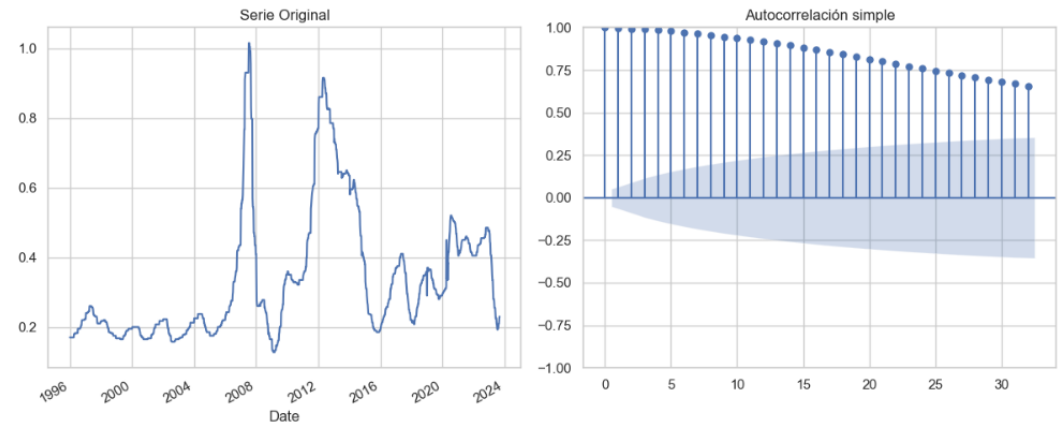


Figura 29: Definición del término MA o q.

Finalmente y habiendo definido los parámetros d, p y q, se procede a instanciar el modelo ARIMA en el objeto m, posteriormente de utiliza el método fit_predict(x_train) para entrenar el modelo, envindo el subconjunto destinado al entrenamiento de las serie temporal. Dicho proceso se muestra en las Figuras 30 y 31.

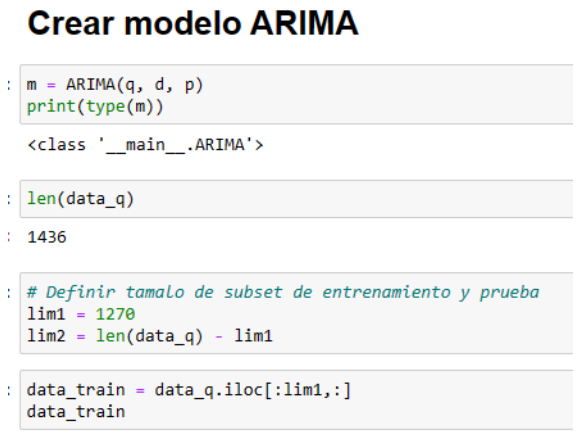


Figura 30: Instanciación del modelo ARIMA en m.

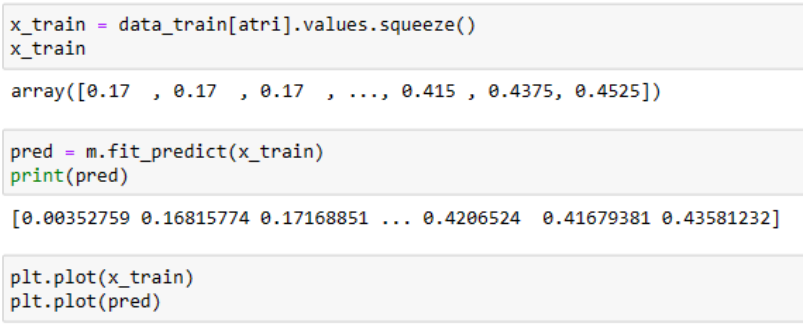
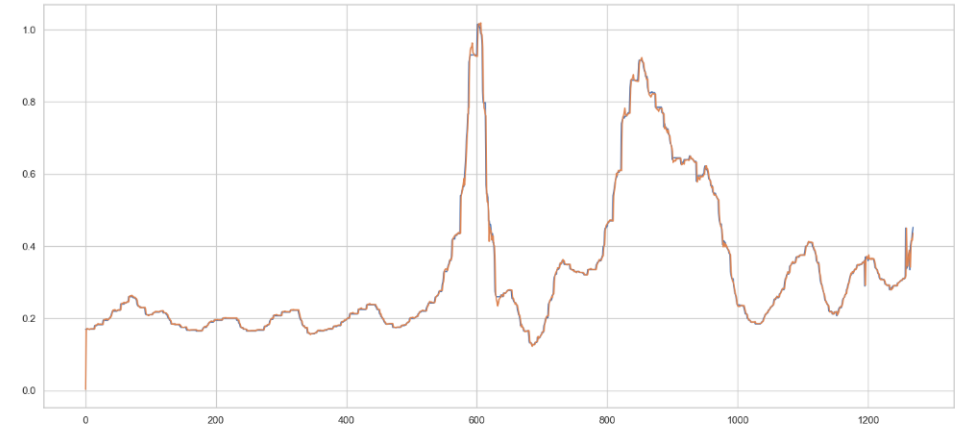


Figura 31: Entrenamiento del modelo ARIMA.

Los resultados del entrenamiento del modelo se muestran en la siguiente sección, así mismo, se mostrará la instanciación de un segundo modelo para predecir precios futuros, es decir, fuera del rango de fechas del dataset usado.

4. Resultados

En la Figura 32, se muestra el ajuste realizado por el modelo ARIMA al ser entrenado con la serie temporal, se observa que sigue a la serie correctamente.




```
evaluate_forecast(x_train, pred)
```

	r2_score	mean_absolute_error	median_absolute_error	mse	msle	mape	rmse
0	0.994718	0.005283	0.001907	0.000204	0.000095	1.553397	0.014277

Figura 32: Ajuste realizado por ARIMA al entrenar con la serie temporal.

En seguida, se toma el conjunto de prueba para evaluar el modelo ARIMA. La llamada al método de predicción se presenta en la Figura 33, mientras que la gráfica y evaluación de la serie y los precios predichos se muestran en la Figura 34. De la evaluación del rendimiento del modelo ARIMA generado, es posible notar que no logra seguir la tendencia de la serie, pues tiende a decaer y volverse constante, siendo confirmado por la métrica r cuadrada, pues el valor cercano a cero indica que el modelo explica solo un pequeño porcentaje de la variabilidad en los datos, sin embargo, los errores respecto al valor real se mantienen bajos.

```
pred_test = m.forecast(x_train, lim2)
pred_test
```

```
array([0.00352759, 0.16815774, 0.17168851, ..., 0.39110979, 0.39099416,
       0.39087958])
```

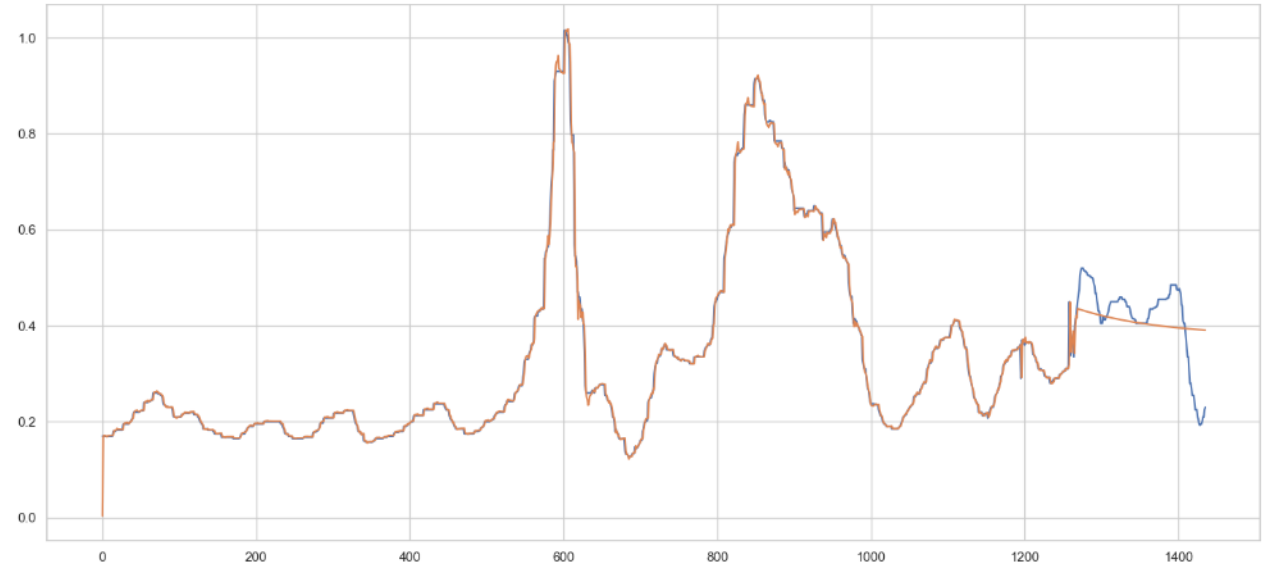
```
x = data_q[atri].values.squeeze()

plt.plot(x)
plt.plot(pred_test)
```

```
[<matplotlib.lines.Line2D at 0x18a012f4280>]
```

Figura 33: Llamada al método forecast para evaluar con el conjunto de prueba.

A continuación se entrena un segundo modelo con los parámetro d, p y q ligeramente distintos, esto con el fin de predecir precios fuera del rango contenido en la base de datos. En las Figuras 35, 36 se muestra la instanciación del segundo modelo, mientras que en la Figura 37 se presenta la gráfica con del ajuste obtenido con el modelo. Si bien los precios




```
evaluate_forecast(x[lim1:], pred_test[lim1:])
```

	r2_score	mean_absolute_error	median_absolute_error	mse	msle	mape	rmse
0	0.142968	0.056437	0.043545	0.005687	0.003117	17.197565	0.07541

Figura 34: Serie temporal real vs precio predicho por ARIMA.

predichos por el segundo modelo parecen mantener su tendencia a la alza, también comienzan a estabilizarse y tienden a un valor continuo.

Entrenar otro modelo con el dataset completo

```
# Modelo 2

d2 = 0
p2 = 10
q2 = 8

m2 = ARIMA(q2, d2, p2)
predf = m2.fit_predict(x)
predf
```

```
array([0.00269471, 0.16803232, 0.16858664, ..., 0.21026749, 0.21105538,
       0.22682315])
```

```
evaluate_forecast(x, predf)
```

	r2_score	mean_absolute_error	median_absolute_error	mse	msle	mape	rmse
0	0.994877	0.005254	0.002017	0.000183	0.000087	1.554736	0.013536

Figura 35: Generación del segundo modelo, usado para predicciones a futuro.

Predecir a futuro (fuera del rango de fechas del dataset)

```
# Número de semanas a futuro para predecir
lim3 = 60
pred_future = m2.forecast(x, lim3)
pred_future
```

```
array([0.00269471, 0.16803232, 0.16858664, ..., 0.33107784, 0.33167224,
       0.33224344])
```

```
plt.plot(x)
plt.plot(pred_future)
```

Figura 36: Llamado a método de predicción para datos futuros.

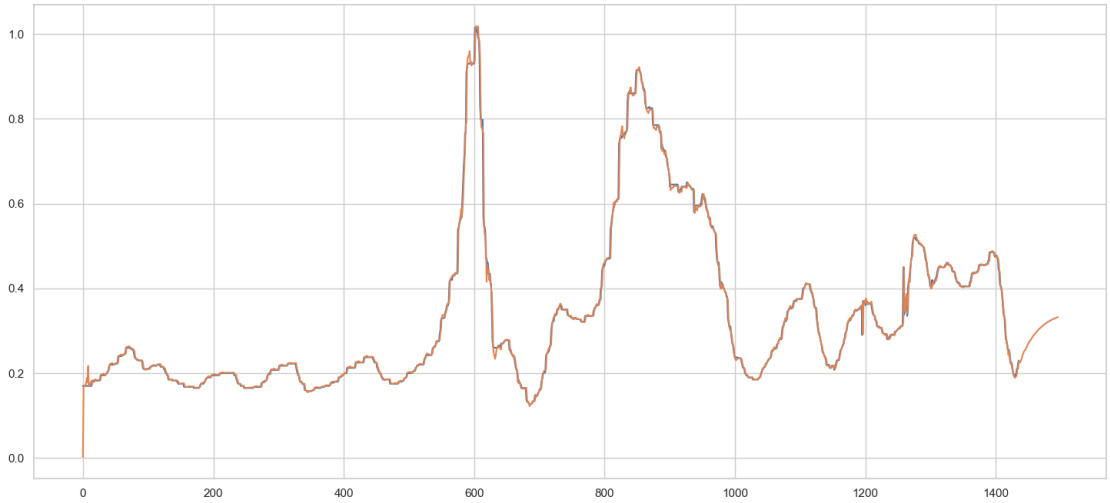


Figura 37: Serie temporal real y predicción realizada por el modelo 2 de ARIMA.

5. Conclusión

La aplicación de modelos de series temporales, como ARIMA, desempeña un papel crucial en el análisis y la predicción de precios en diversas industrias, especialmente en áreas en donde la evolución temporal de los datos es un factor determinante. La serie de tiempo de derivados lácteos utilizada para este análisis presenta desafíos comunes, como tendencias y estacionalidades, que son comunes en datos financieros y económicos.

A través de la implementación y evaluación de modelos ARIMA, se ha observado la capacidad de estos para proporcionar predicciones precisas en rangos cercanos a la última observación de la base de datos, aunque con limitaciones en la captura de tendencias complejas. Las métricas de rendimiento, como el coeficiente de determinación y los errores absolutos, ofrecen una visión integral del desempeño del modelo. En el caso particular del primer modelo, se ha evidenciado que, a pesar de su capacidad limitada para seguir la tendencia de la serie, logra mantener bajos los errores respecto a los valores reales, habiendo aún un margen para mejorar la capacidad del modelo para explicar la variabilidad en los datos.

La evaluación de un segundo modelo con parámetros ligeramente distintos muestra la importancia de la calibración adecuada de estos, con el fin de mejorar las predicciones. Aunque los modelos ARIMA son herramientas valiosas para la predicción de precios en series temporales, es esencial comprender las limitaciones de estos y realizar ajustes meticulosos para adaptarlos a la complejidad específica de los datos. La elección cuidadosa de parámetros y la consideración de diversas técnicas de modelado son fundamentales para maximizar la utilidad de los modelos en la toma de decisiones y la planificación estratégica en entornos económicos cambiantes.

Referencias

[1] Jonathan D Cryer. *Time series analysis*. Vol. 286. Duxbury Press Boston, 1986.

[2] Jamal Fattah et al. "Forecasting of demand using ARIMA model". En: *International Journal of Engineering Business Management* 10 (2018), pág. 1847979018808673.