Práctica: Series Temporales 2024-2025

Master Ciencia de Datos

Universidad de Granada

Práctica: Series Temporales

Miguel García López

Índice

Índice de figuras

Índice de cuadros

1. Introducción

La práctica de la asignatura de **Series Temporales y Minería de Flujos de Datos** consiste en, a partir de una conjunto de datos dado, resolver un problema de series temporales. Concretamente el conjunto de datos dado es **Oikolab Weather**, el cual contiene 8 series temporales sobre datos climáticos, aunque en esta práctica se debe trabajar tan solo con la variable de temperatura. Los datos se empezaron a medir el 1 de Enero de 2010.

El objetivo es predecir la temperatura para los meses restantes del año donde termina la serie, es decir, 2021. Según se avance en el trabajo, se irán respondiendo cuestiones planteadas en el guión de prácticas.

2. Tareas

2.1. ¿Es necesario realizar algún tipo de preprocesamiento en la serie? Tanto en el caso afirmativo como en el negativo, justifique su respuesta e incluya el código Python requerido, si es el caso.

Sí, es necesario. Los datos originales son horarios, pero la práctica requiere trabajar con la temperatura promedio mensual. Además, se deben manejar valores faltantes y duplicados.

Además, la serie parece tener estacionalidad, es decir, no es estacionaria por lo que es posible que según se trabaje con la serie haga falta alguna transformación en el caso de querer que sea estacionaria.

En la imagen 1 se puede observar que los datos no parecen seguir tendencia, tienen una media de prácticamente cero y no varían de forma desigual con el tiempo. De todas formas, todo esto se analizará en las siguientes preguntas.

2.2. ¿Tiene tendencia la serie? Tanto en el caso afirmativo como en el negativo, justifique su respuesta e incluya el código Python requerido, si es el caso, justificando el modelo de tendencia.

La serie no presenta tendencia ya que la media de esta es constante. Esto se puede analizar visualmente a través de la imagen en la figura 2, donde aplicando una media

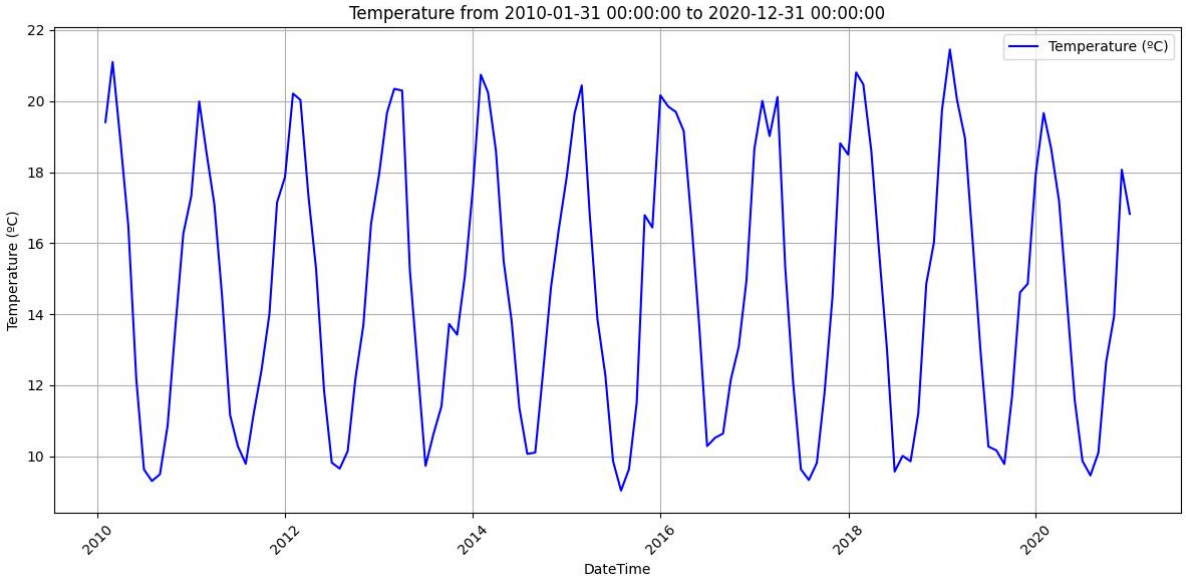

Figura 1: Serie temporal de la temperatura (partición de *train*).

móvil se extrae una tendencia marcada en rojo. Esta ni disminuye ni crece en el tiempo, sino que es periódica.

```
    def apply_rolling_mean(
        df: pd.DataFrame, column: str, window: int = 7, center: bool =
    True
    ) -> pd.DataFrame:
        """
        Applies a rolling mean to the specified column in the DataFrame
    .

        Parameters:
        df (pd.DataFrame): The DataFrame containing the time series
    data.
        column (str): The column to apply the rolling mean to.
        window (int): The size of the rolling window (default: 7 days).
        center (bool): Whether to center the window (default: True).

        Returns:
        pd.DataFrame: DataFrame with an additional column for the
    rolling mean.
        """
        result_df = df.copy()
        result_df[f"{column}_Rolling_Mean"] = (
            result_df[column].rolling(window=window, center=center).
    mean()
        )
        return result_df

    df_with_rolling = apply_rolling_mean(df, "Temperature)", window
    =400)

```

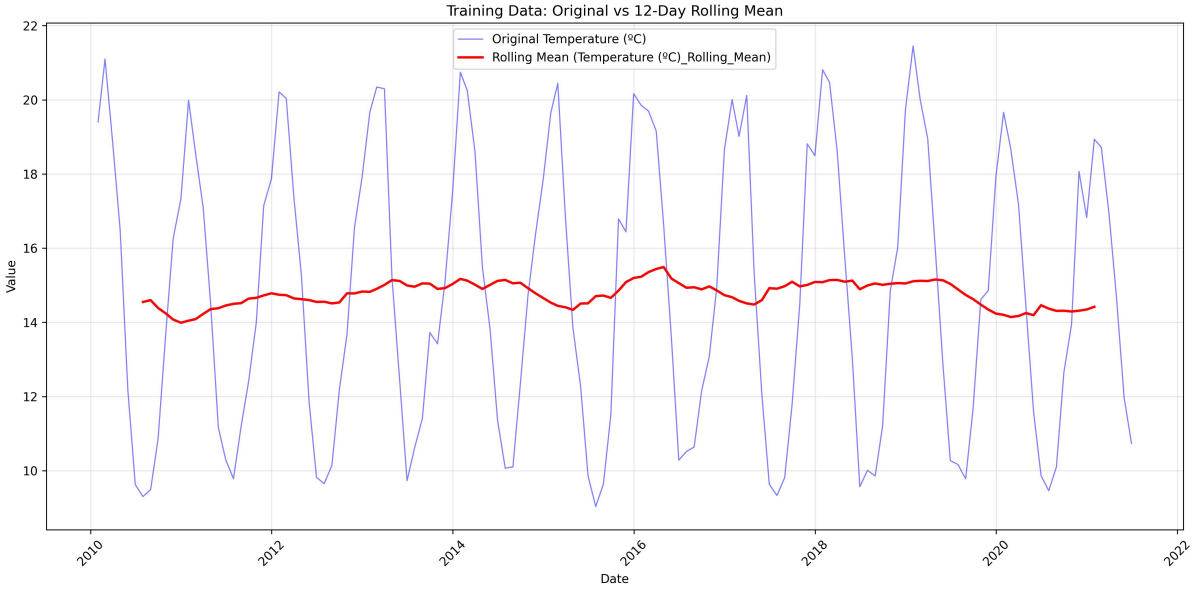

Figura 2: Serie temporal de la temperatura (partición de *train*) con *rolling mean*.

```
    plot_before_after_rolling_mean(
        df_with_rolling,
        "Temperature",
        "Temperature_Rolling_Mean",
        time_range=np.array([str(df.index.min()), str(df.index.max())])
    ,
        title="Training Data: Original vs 7-Day Rolling Mean",
        filename="temperature_train_with_rolling.jpg",
    )
```

Listing 1: Aplicación de Media Móvil y Visualización

2.3. Tiene estacionalidad la serie? Tanto en el caso afirmativo como en el negativo, justifique su respuesta e incluya el código Python requerido, si es el caso, justificando el modelo de estacionalidad.

La serie tiene estacionalidad, si se analiza el gráfico de autocorrelación (ACF) de la serie temporal en la figura 3, es posible observar picos regulares y repetitivos a lo largo del gráfico. Esto sugiere que el periodo de estacionalidad es $S \approx 6$. Es decir, la serie parece repetir su comportamiento cada 6 unidades de tiempo.

```
plot_acf(ts, lags=120)
plt.suptitle(
    "Time Series Decomposition"
)
plt.savefig("ts_autocorrelation.png")
```

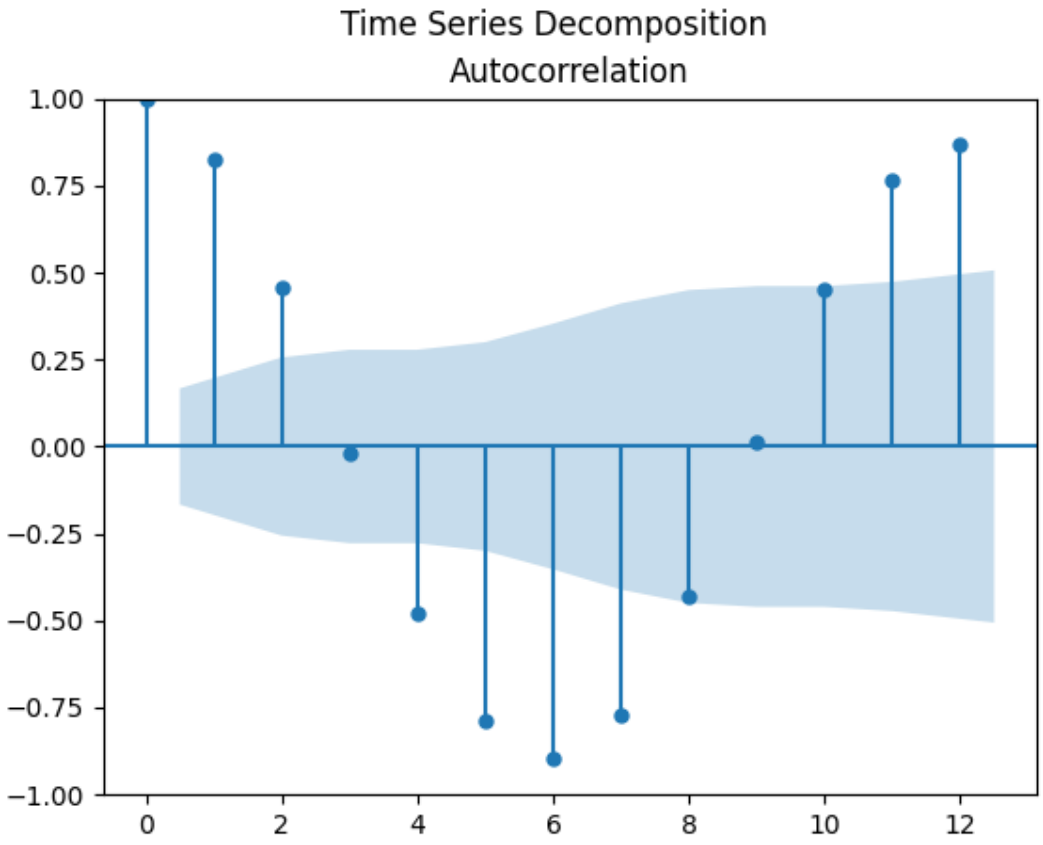

Figura 3: Gráfico de autocorrelación (ACF).

2.4. ¿Es la serie estacionaria? Tanto en el caso afirmativo como en el negativo, justifique su respuesta e incluya el código Python requerido, si es el caso, para conseguir la estacionariedad.

La serie no es estacionaria. Como puede observarse en la figura 3, la persistencia de las autocorrelaciones es alta y existe un claro componente estacional. Además, se ha mencionado previamente que la serie presenta estacionalidad, lo cual implica que no puede considerarse estacionaria en su forma original. De todas formas, al aplicarle un test de *adfuller*, se obtiene que se probablemente sea no estacionaria, con un p-valor de 0,13

```
    def check_stationary(ts: np.ndarray):
        result = adfuller(ts)
        print("Resultados de la prueba ADF:")
        print(f"Estadistico ADF: {result[0]}")
        print(f"p-valor: {result[1]}")
        print(f"Valores criticos: {result[4]}")
        if result[1] > 0.05:
            print("La serie probablemente no es estacionaria.")
        else:
            print("La serie probablemente es estacionaria.")
```

Para hacer que la serie sea estacionaria eliminamos la estacionalidad capturando este patrón promediando los valores de la serie temporal a intervalos de 12. Luego, se resta este modelo de estacionalidad de la serie temporal original, resultando en una serie ajustada y sin estacionalidad. Se puede ver el modelo estacional en la figura 4 y la serie temporal resultante en 5.

Al aplicar de nuevo el test lo pasa con un p-valor de $3{,}99e-14$.

```
def remove_seasonality(t, x_ts, seasonality_period=12):
    season = np.zeros(seasonality_period)
    for i in range(seasonality_period):
        season[i] = np.mean(x_ts[i::seasonality_period])

    num_seasons = int(np.ceil(len(x_ts) / seasonality_period))
    tiled_season = np.tile(season, num_seasons)[: len(x_ts)]

    plt.figure(figsize=(10, 4))
    plt.plot(season)
    plt.title("Modelo de Estacionalidad")
    plt.savefig("seasonality_model.png")
    plt.close()

    plt.figure(figsize=(12, 6))
    plt.plot(t, x_ts, label="Serie Temporal")
    plt.plot(t, tiled_season, label="Modelo de Estacionalidad",
   linestyle="--")
    plt.title("Serie Temporal con Modelo de Estacionalidad")
    plt.savefig("time_series_with_seasonality.png")
```

```
    plt.close()

    x_ts_no_season = x_ts - tiled_season

    plt.figure(figsize=(12, 6))
    plt.plot(t, x_ts_no_season)
    plt.title("Serie Temporal sin Estacionalidad")
    plt.savefig("time_series_no_seasonality.png")
    plt.close()

    return x_ts_no_season, season, tiled_season
```

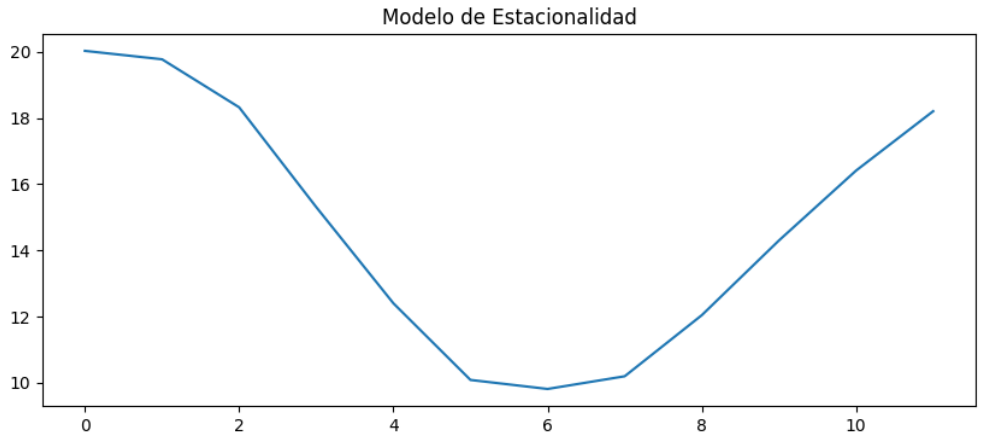

Figura 4: Modelo de estacionalidad

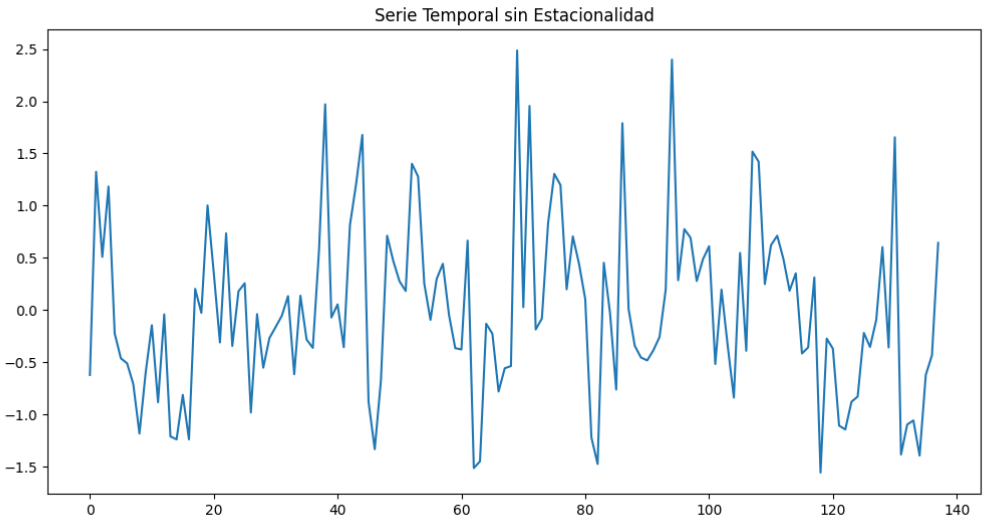

Figura 5: Serie temporal sin estacionalidad

2.5. Parámetros ARIMA: Metodología usada para encontrar los mejores parámetros.

Se ha creado una función que prueba todas las combinaciones posibles de (p, d, q) dentro del rango $[0, max\_order]$ y se utiliza el criterio AIC para seleccionar el mejor modelo. El AIC es una métrica que equilibra la bondad de ajuste del modelo y su complejidad. Un AIC más bajo indica un modelo mejor. Se obtienen los siguientes resultados:

Mejor orden ARIMA: $(3, 0, 2)$, **AIC:** 329,94881222729543

Mejor orden ARIMA encontrado: $(3, 0, 2)$, 329,94881222729543

2.6. Predicción: Elaboración de los pasos requeridos para realizar la predicción real de los valores de la serie de temperatura requeridos.

Una vez identificado el mejor orden ARIMA (p, d, q) mediante la minimización del criterio AIC, se procede a entrenar el modelo utilizando los datos de entrenamiento. El modelo ARIMA se ajusta a la serie temporal de temperatura para capturar sus patrones y tendencias.

Con el modelo ARIMA entrenado, se realizan predicciones para el período de prueba. Estas predicciones corresponden a los valores futuros de la serie de temperatura. El número de pasos de predicción depende de la longitud del conjunto de prueba.

En la figura 6 se puede observar las predicciones para el conjunto de datos de test.

```
def predict_arima(train_data, test_data, order):
    model = ARIMA(train_data, order=order)
    model_fit = model.fit()
    predictions = model_fit.forecast(steps=len(test_data))
    return predictions
```

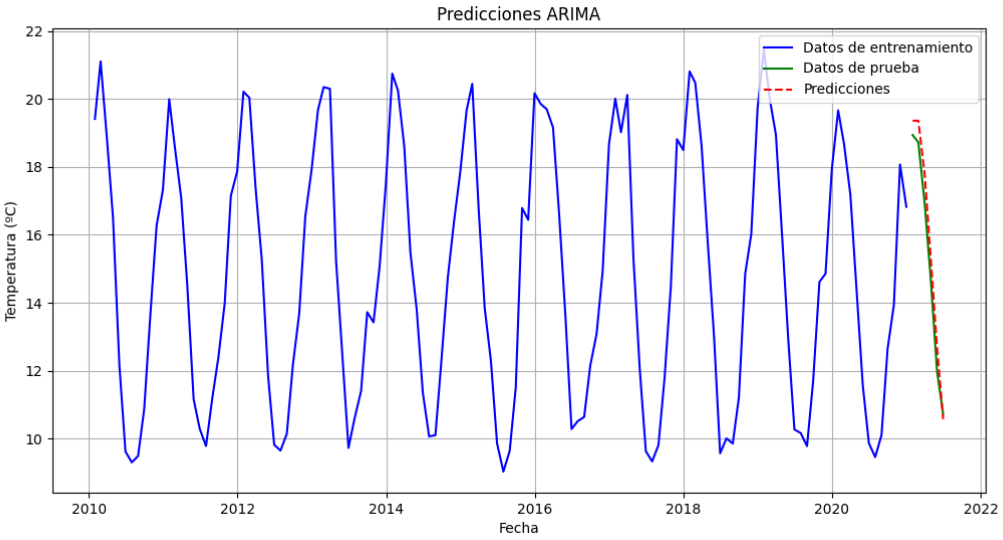

Figura 6: Predicciones del modelo entrenado.