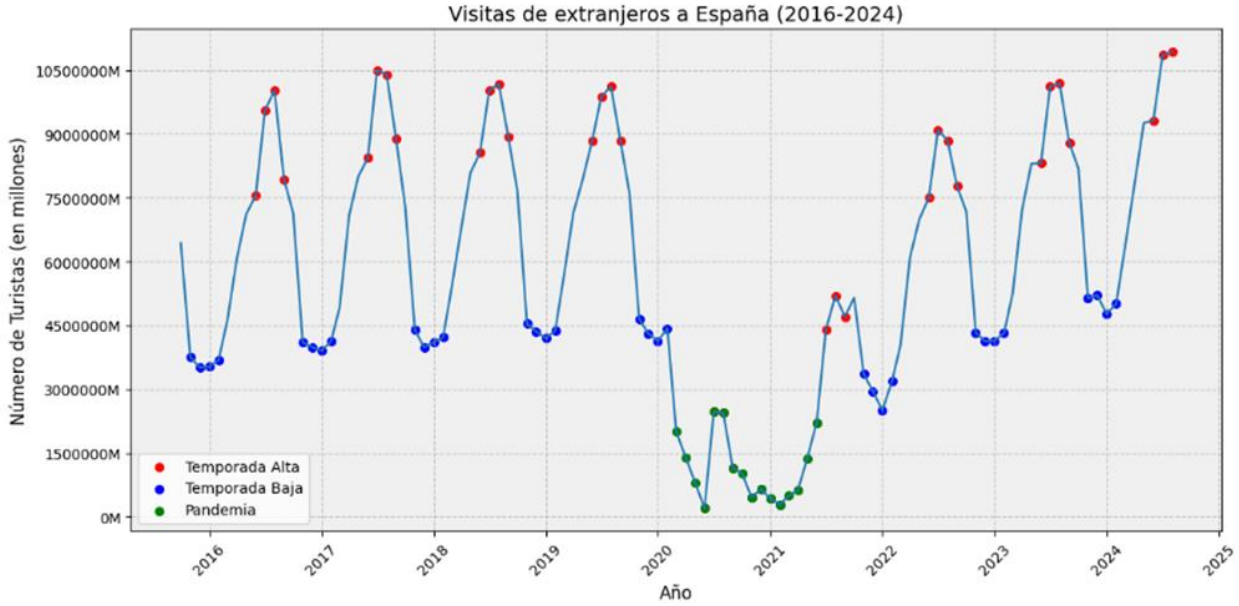
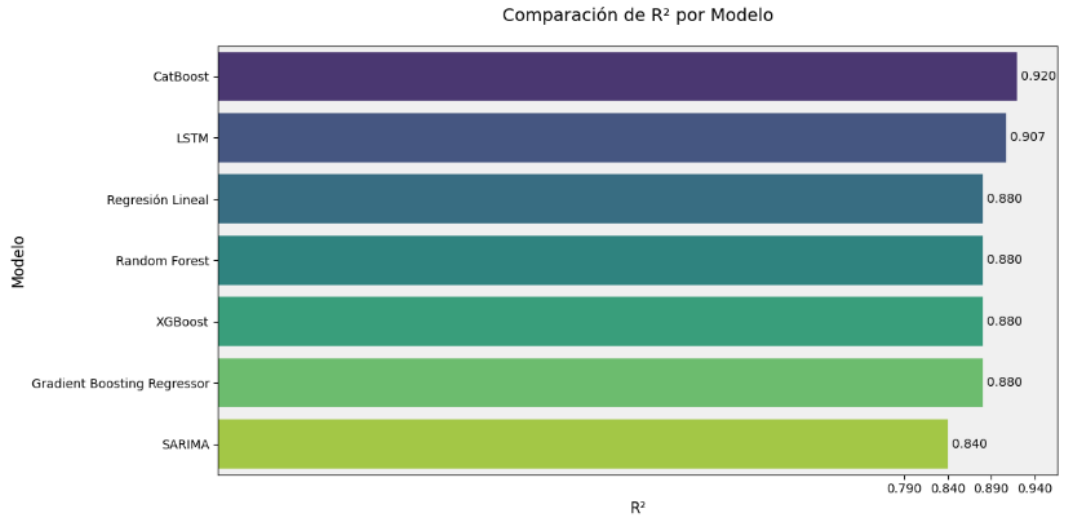
# Machine Learning aplicado al

# Sector Turístico




## *Modelado y Predicción del Turismo en España: De SARIMA a Redes Neuronales en un Escenario Post-Pandémico*

En este informe exploraremos una serie de enfoques para modelar y predecir las tendencias turísticas en España, comenzando con métodos tradicionales de series temporales y avanzando hacia técnicas más sofisticadas de aprendizaje automático y Deep Learning. Nuestro objetivo es no solo encontrar el modelo más preciso, sino también comprender las ventajas y limitaciones de cada enfoque en el contexto de datos turísticos que han experimentado un shock sin precedentes como consecuencia de la pandemia de Covid-19.




Microsoft CERTIFIED POWER BI DATA ANALYST ASSOCIATE

**Enrique Aranaz Tudela**

*FREELANCER | DATA DRIVEN | CONSULTOR BI | DATA SCIENTIST | MACHINE LEARNING PYTHON | POWER BI*

# INDICE CONTENIDO



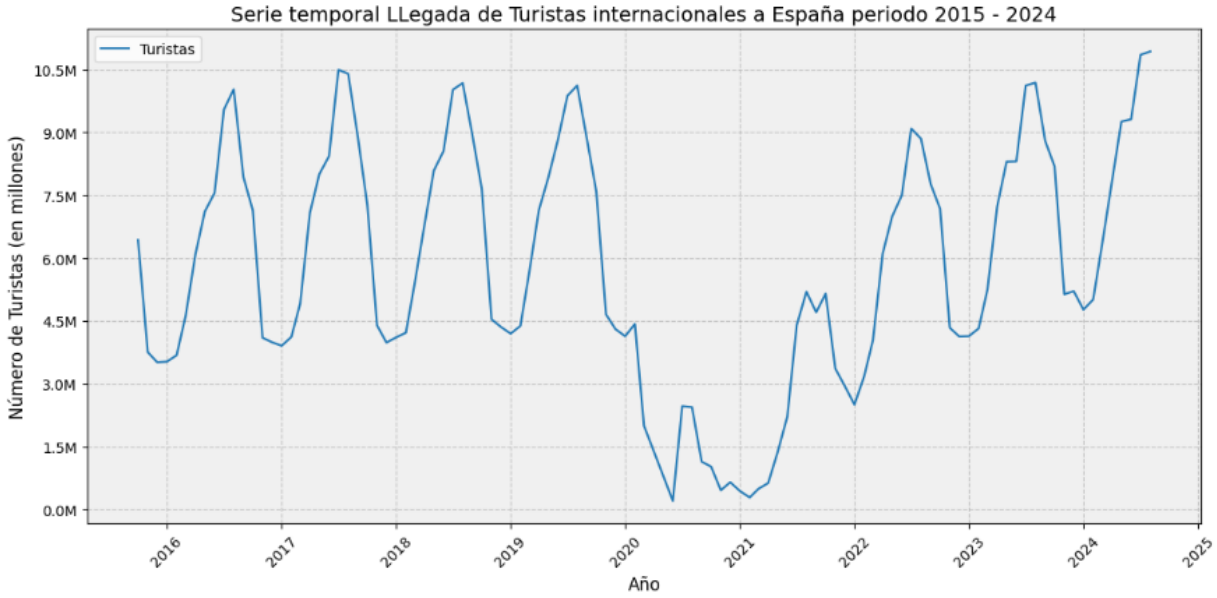
# 01 Introducción

El turismo es un pilar fundamental de la economía española, representando una parte significativa del PIB del país y siendo una fuente crucial de empleo. Sin embargo, la naturaleza volátil de este sector, influenciada por factores estacionales, económicos y, más recientemente, por eventos globales sin precedentes como la pandemia de COVID-19, presenta desafíos únicos para su modelado y predicción.

## 1.1 El desafío de la pandemia

La pandemia de COVID-19 introdujo una discontinuidad sin precedentes en los datos turísticos que conlleva una ruptura de patrones históricos que los modelos tradicionales de análisis de series, como SARIMA, que asumen cierta continuidad en los patrones subyacentes, pueden tener dificultades para capturar.

En este artículo, utilizaremos la serie temporal de llegada de turistas internacionales a nuestro país (Fuente INE, Estadística de Movimientos Turísticos en Fronteras), para implementar distintos enfoques para modelar y predecir las tendencias turísticas en España.




En este contexto, nuestro enfoque pretende comenzar con SARIMA y luego avanzar hacia modelos más complejos, lo que nos permitirá:

- **Identificar si existen limitaciones en los enfoques tradicionales frente a shocks extremos.**
- **Explorar cómo la incorporación de variables exógenas puede mejorar la capacidad del modelo para manejar disrupciones.**
- **Evaluar si técnicas de aprendizaje automático y redes neuronales pueden capturar mejor la complejidad del shock pandémico.**

## 1.2. Metodologías y Métricas

**Metodologías que emplearemos en nuestro análisis:**

### Enfoques de Series Temporales

- ARIMA (AutoRegressive Integrated Moving Average)
- SARIMA (Seasonal ARIMA)

### Algoritmos de  Machine Learning

- Regresión lineal
- Random Forest
- XGBoost (Extreme Gradient Boosting)
- Gradient Boosting Regressor
- Cambios

### Deep Learning / Redes Neuronales

- LSTM (Long Short-Term Memory)

## Evaluación y Comparación de Modelos

En el análisis de series temporales y predicción es crucial seleccionar métricas de evaluación apropiadas. Cada métrica proporciona una perspectiva diferente sobre el rendimiento del modelo. Para nuestro estudio sobre el turismo en España, consideraremos las siguientes métricas:

- **Error Absoluto Medio (MAE - Mean Absolute Error):**

El MAE mide la magnitud promedio de los errores en un conjunto de predicciones, sin considerar su dirección. Se expresa en las mismas unidades que la variable objetivo, lo que lo hace fácilmente interpretable.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Ventaja: Es intuitivo y directamente interpretable en términos de unidades de visitantes.
Limitación: No penaliza los errores grandes tanto como otras métricas.

- **Error Porcentual Absoluto Medio (MAPE - Mean Absolute Percentage Error):**

El MAPE mide el tamaño promedio del error en términos porcentuales. Es particularmente útil cuando queremos comparar el rendimiento entre diferentes conjuntos de datos o cuando la escala es importante.

Ventaja: Permite comparar fácilmente el rendimiento entre diferentes períodos o destinos turísticos.

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

- **Coeficiente de Determinación ($R^2$):**

$R^2$ mide la proporción de la varianza en la variable dependiente que es predecible a partir de la(s) variable(s) independiente(s). Varía entre 0 y 1, donde 1 indica una predicción perfecta

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Ventaja: Proporciona una medida de qué tan bien el modelo explica la variabilidad en los datos.
Limitación: No siempre es adecuado para series temporales, especialmente con tendencias fuertes.

Para nuestro análisis del turismo en España, utilizaremos principalmente el MAPE como métrica principal de comparación. Las razones para esta elección son:

- ✓ **Interpretabilidad: El MAPE proporciona un porcentaje de error fácil de entender, lo que es valioso para comunicar resultados a stakeholders no técnicos en la industria turística.**
- ✓ **Comparabilidad: Permite comparar el rendimiento del modelo en diferentes períodos, como antes, durante y después de la pandemia.**
- ✓ **Manejo de la estacionalidad: El MAPE es útil para evaluar modelos en datos con fuerte estacionalidad, como es el caso del turismo español.**

En cualquier caso, también reportaremos el MAE para proporcionar una perspectiva en términos absolutos, y el $R^2$ para dar una idea de la capacidad explicativa general de cada modelo.

## 1.3. Librerías de Python que vamos a utilizar para nuestro análisis

- **Statsmodels** es una biblioteca robusta y versátil que proporciona un conjunto completo de herramientas para el análisis de series temporales en Python. Su uso en conjunto con otras bibliotecas como pandas para manejo de datos y matplotlib para visualización crea un ecosistema poderoso para el análisis y modelado de series temporales.

- **Pmdarima**: Biblioteca que implementa auto_arima, muy útil para la selección automática de modelos ARIMA/SARIMA.

- **Scikit-learn:** Es una biblioteca de aprendizaje automático que incluye herramientas eficientes para la minería de datos y análisis de datos. Ofrece algoritmos para clasificación, regresión, clustering y más.

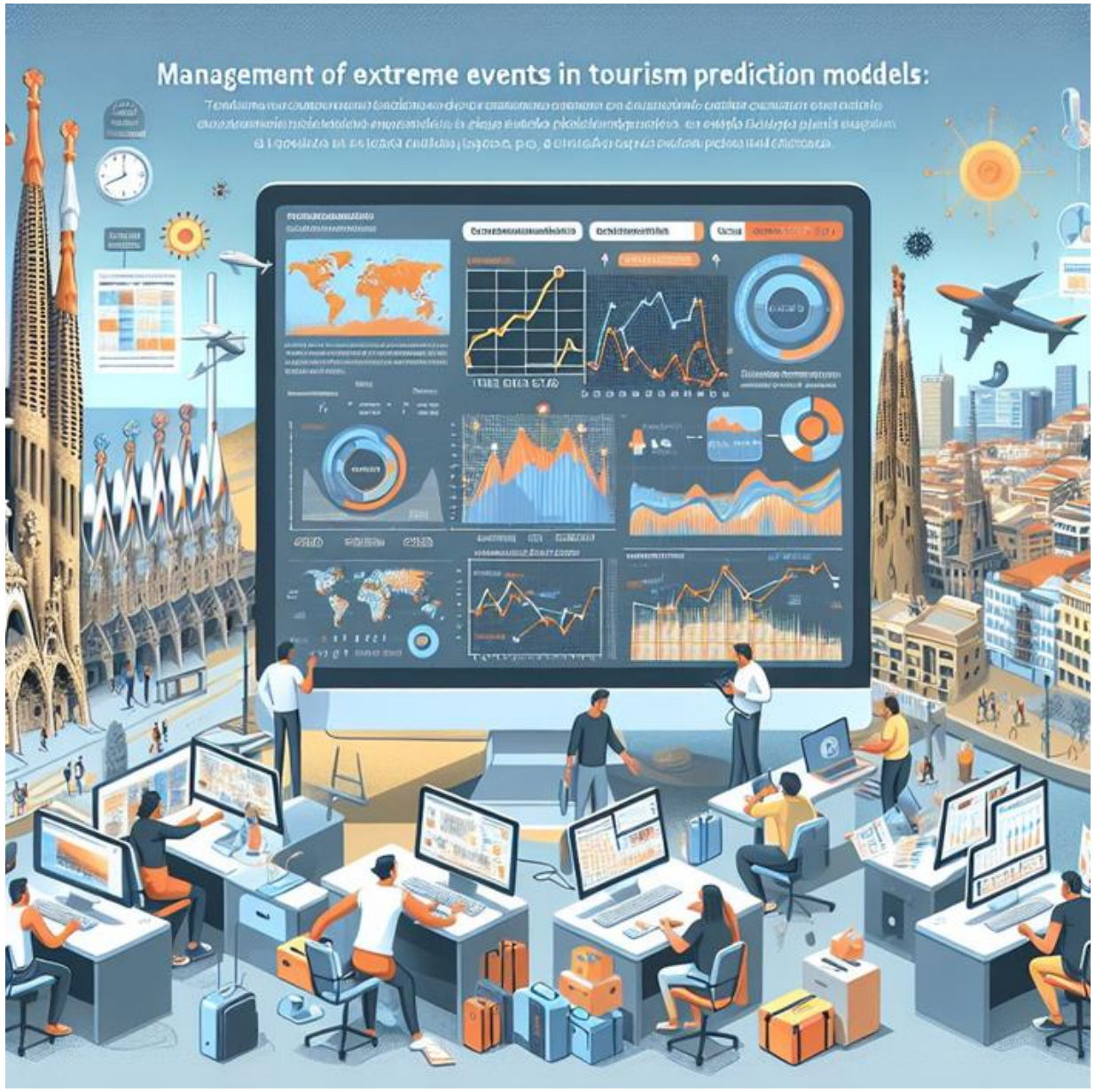
- **Keras**: Es una biblioteca de alto nivel para construir y entrenar modelos de aprendizaje profundo. Es conocida por su simplicidad y facilidad de uso, permitiendo crear redes neuronales con pocas líneas de código.

- **TensorFlow**: Es una plataforma de código abierto desarrollada por Google para el aprendizaje automático. Ofrece una amplia gama de herramientas y recursos para construir y desplegar modelos de aprendizaje profundo y aprendizaje automático a gran escala.

- **Pandas:** Es una biblioteca de Python para la manipulación y análisis de datos. Ofrece estructuras de datos como DataFrames, que facilitan el manejo de datos tabulares.

- **NumPy:** Es fundamental para la computación científica en Python. Proporciona soporte para arrays multidimensionales y funciones matemáticas de alto nivel.

- **SciPy:** Construida sobre NumPy, esta biblioteca se utiliza para tareas más avanzadas de matemáticas, ciencia e ingeniería, como optimización, integración y álgebra lineal.



# 02 Desarrollo de la modelización con Python

## 2.1 Carga de los datos vía API INE

Cómo es tradicional en todos mis trabajos, aprovecharemos las ventajas que Python nos brinda para conectarnos a múltiples fuentes de datos y extraeremos directamente la información del INE (Estadística de Movimientos Turísticos en Fronteras, código FREG297).

```python
def ine_request(ine_code):
    resultados = 999
    path_template = 'http://servicios.ine.es/wstempus/js/ES/DATOS_SERIE/{cod_serie}?nult={n_ult_datos}'
    path = path_template.format(cod_serie=ine_code, n_ult_datos=resultados)
    json_request = requests.get(path).json()

    return json_request

# Lista de códigos definida directamente en el script
codigos = [ {"ine_code": "FREG297"}]

anyo_lista = list()
mes_lista = list()
dia_lista = list()
fecha_completa_lista = list()
variable_lista = list()
valor_lista = list()

for codigo in codigos:
    datos = ine_request(codigo['ine_code'])  # Asegúrate de pasar el código correcto

    nombre_variable = datos['Nombre']
    for dato in datos['Data']:
        fecha = datetime.date.fromtimestamp(dato['Fecha'] // 1000)
        anyo = fecha.year
        mes = fecha.month
        dia = fecha.day
        valor = dato['Valor']
        fecha_completa = fecha.strftime('%Y-%m-%d')

        anyo_lista.append(anyo)
        mes_lista.append(mes)
        dia_lista.append(dia)
        fecha_completa_lista.append(fecha_completa)
        variable_lista.append(nombre_variable)
        valor_lista.append(valor)

df = pd.DataFrame({
    'Año': anyo_lista,
    'Mes': mes_lista,
    'Día': dia_lista,
    'Fecha Completa': fecha_completa_lista,
    'Dato': variable_lista,
    'Valor': valor_lista
})
```

df

| | Año | Mes | Día | Fecha Completa | Dato | Valor |
|---|---|---|---|---|---|---|
| **0** | 2015 | 10 | 1 | 2015-10-01 | Turista. Total Nacional. Total. Dato base. | 6432341.0 |
| **1** | 2015 | 11 | 1 | 2015-11-01 | Turista. Total Nacional. Total. Dato base. | 3754802.0 |
| **2** | 2015 | 12 | 1 | 2015-12-01 | Turista. Total Nacional. Total. Dato base. | 3512914.0 |
| **3** | 2016 | 1 | 1 | 2016-01-01 | Turista. Total Nacional. Total. Dato base. | 3526537.0 |
| **4** | 2016 | 2 | 1 | 2016-02-01 | Turista. Total Nacional. Total. Dato base. | 3678726.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **102** | 2024 | 4 | 1 | 2024-04-01 | Turista. Total Nacional. Total. Dato base. | 7831094.0 |
| **103** | 2024 | 5 | 1 | 2024-05-01 | Turista. Total Nacional. Total. Dato base. | 9256446.0 |
| **104** | 2024 | 6 | 1 | 2024-06-01 | Turista. Total Nacional. Total. Dato base. | 9313450.0 |
| **105** | 2024 | 7 | 1 | 2024-07-01 | Turista. Total Nacional. Total. Dato base. | 10851172.0 |
| **106** | 2024 | 8 | 1 | 2024-08-01 | Turista. Total Nacional. Total. Dato base. | 10930750.0 |

107 rows × 6 columns

Las APIs permiten acceder a grandes volúmenes de datos de manera rápida y eficiente, sin necesidad de descargar archivos manualmente. Esto facilita la integración de datos en tiempo real en sistemas y aplicaciones empresariales y tiene entre sus muchas ventajas que permite contar con información actual y precisa, lo que es crucial para la toma de decisiones.

## 2.2 Preparación de los datos y exploraciones previas

En el análisis de los datos previo a su modelización se observan dos datos nulos, correspondientes a abril y mayo de 2020, consecuencia del cierre de fronteras que origino la pandemia de Covid19.

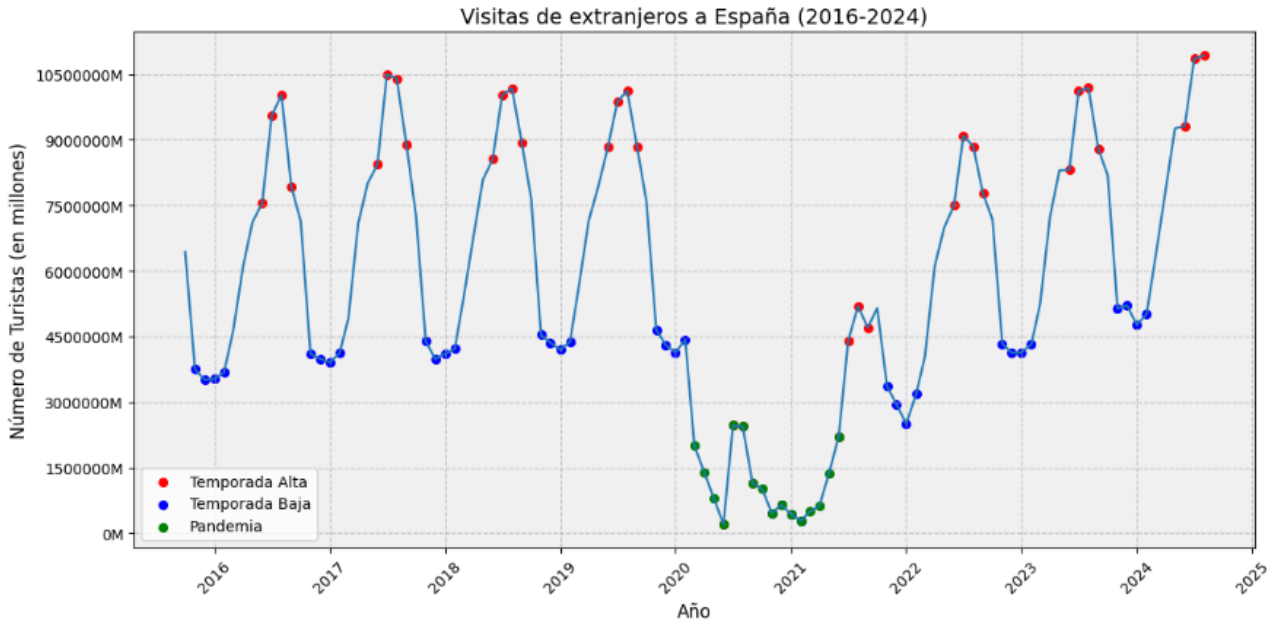
| Fecha Completa | Año | Mes | Día | Dato | Valor |
|---|---|---|---|---|---|
| **2020-04-01** | 2020 | 4 | 1 | Turista. Total Nacional. Total. Dato base. | 0.0 |
| **2020-05-01** | 2020 | 5 | 1 | Turista. Total Nacional. Total. Dato base. | 0.0 |

Para minimizar el impacto, procederemos a realizar una estimación de ambos valores mediante interpolación:

```
# Reemplazar los ceros con NaN para facilitar la interpolación
df_corrected.loc[zero_positions, 'Valor'] = np.nan

# Interpolar solo los valores NaN (que eran ceros)
df_corrected['Valor'] = df_corrected['Valor'].interpolate(method='time')
```

Para apreciar tanto el efecto de la pandemia, cómo la estacionalidad de la serie, procederemos a crear variables que muestren los meses de temporada alta, baja, y los meses en que los que tuvo lugar la pandemia:

```
# Crear variable dummy para Temporada Alta
df['Temp_Alta'] = df.index.month.isin([6, 7, 8, 9]).astype(int)

# Crear variable dummy para Temporada Baja
df['Temp_Baja'] = df.index.month.isin([11, 12, 1, 2]).astype(int)

# Crear variable dummy para Pandemia
df['Pandemia'] = ((df.index >= '2020-03-01') & (df.index <= '2021-06-30')).astype(int)
```




En el gráfico se muestra de forma evidente tanto la estacionalidad del modelo, con entrada máxima de turistas en los meses de temporada alta y notable caída en la temporada baja, cómo el fuerte Impacto que la Pandemia de COVID-19 introdujo sobre los datos, creando una discontinuidad significativa en los patrones de turismo que supone un desafío a la hora de crear modelos predictivos eficientes.

## 2. 3. Modelización con Métodos de series tradicionales

❑ **(AutoRegressive Integrated Moving Average)**

El modelo ARIMA es uno de los métodos más utilizados para la predicción de series temporales. Combina tres componentes clave:

- Autorregresión (AR): Utiliza las dependencias entre una observación y varias observaciones anteriores.
- Diferenciación (I): Hace que la serie temporal sea estacionaria eliminando tendencias y estacionalidades.
- Media Móvil (MA): Modela el error de predicción como una combinación lineal de errores pasados.

El modelo ARIMA es flexible y puede capturar patrones complejos en los datos históricos, lo que lo hace adecuado para predecir la demanda turística, que puede estar influenciada por múltiples factores como temporadas, eventos especiales y tendencias económicas.

❑ **SARIMA (Seasonal ARIMA)**

El modelo SARIMA extiende el ARIMA para manejar datos con estacionalidad explícita. Añade componentes estacionales a los tres componentes de ARIMA:

- Autorregresión Estacional (SAR)
- Diferenciación Estacional (SI)
- Media Móvil Estacional (SMA)

Dada que la estacionalidad de la serie, cómo hemos visto, es evidente, procederemos a implementar un modelo SARIMA cómo la primera de nuestras opciones.

### ¿Por qué empezar con SARIMA?

El modelo **SARIMA (Seasonal AutoRegressive Integrated Moving Average)** es un punto de partida ideal por varias razones:

- **Captura de patrones estacionales**: El turismo en España tiene una fuerte componente estacional, con picos en verano y valles en invierno. SARIMA está específicamente diseñado para manejar estos patrones cíclicos.
- **Manejo de tendencias y autocorrelación**: Además de la estacionalidad, SARIMA puede capturar tendencias a largo plazo y la dependencia entre observaciones cercanas en el tiempo, características comunes en datos turísticos. Interpretabilidad: A diferencia de modelos de "caja negra" más complejos, SARIMA ofrece coeficientes interpretables que pueden proporcionar insights sobre la naturaleza de las series temporales turísticas.
- **Base de comparación**: Comenzar con SARIMA establece una línea base sólida contra la cual podemos comparar modelos más avanzados, permitiéndonos cuantificar las mejoras en precisión.
- **Robustez histórica**: SARIMA ha demostrado ser robusto en una variedad de aplicaciones de series temporales a lo largo de décadas, incluyendo el modelado turístico.

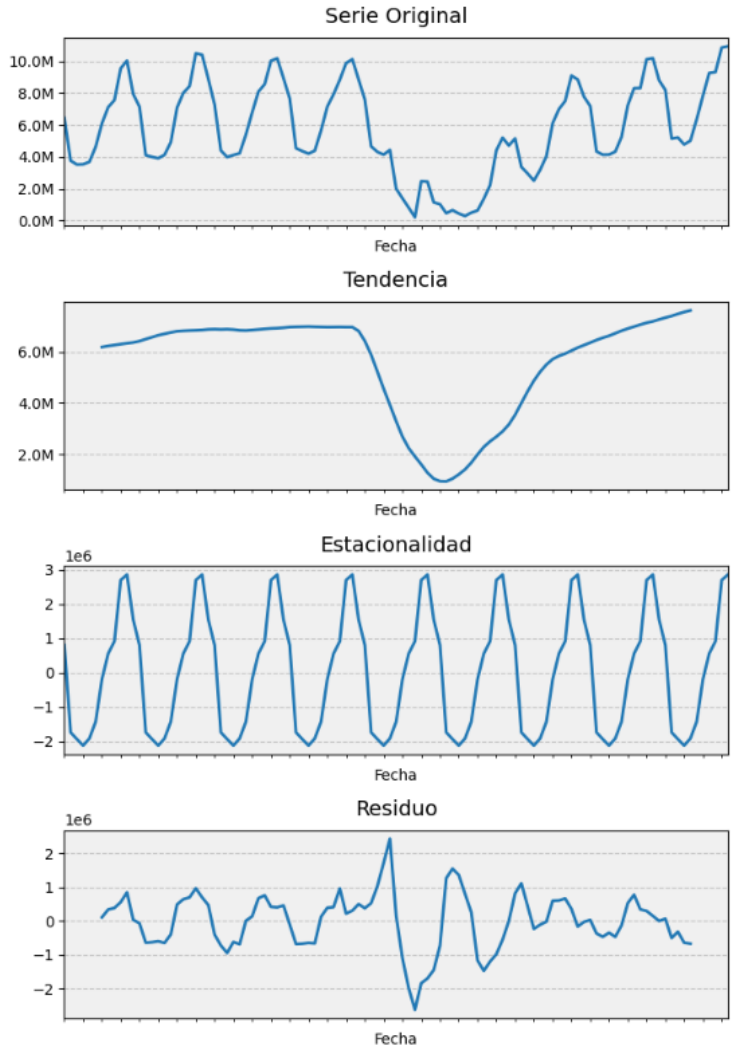
## ❑ SARIMA (Seasonal AutoRegressive Integrated Moving Average)

### Pasos principales para modelar una serie temporal con SARIMA:

1. **Análisis Exploratorio de Datos**: Visualizar la serie temporal Identificar tendencias, estacionalidad y posibles outliers.
2. **Descomposición de la Serie**: Separar en componentes: tendencia, estacionalidad y residuos. Utilizar descomposición aditiva o multiplicativa.
3. **Pruebas de Estacionariedad**: Test de Dickey-Fuller Aumentado (ADF) Transformar la serie si es necesario (diferenciación, log, etc.).
4. **Análisis de Autocorrelación**: Graficar Función de Autocorrelación (ACF) Graficar Función de Autocorrelación Parcial (PACF). Identificar posibles órdenes para p, d, q y sus componentes estacionales.
5. **Selección del Modelo**: Determinar órdenes (p,d,q) y (P,D,Q)m. Usar criterios de información (AIC, BIC) o auto_arima.
6. **Ajuste del Modelo**: Estimar parámetros del modelo SARIMA.
7. **Diagnóstico de Residuos**: Gráfico de residuos vs tiempo Histograma de residuos QQ-plot para normalidad ACF y PACF de residuos Test de Ljung-Box para autocorrelación residual.
8. **Evaluación del Modelo**: Analizar significancia de coeficientes Calcular métricas de error (MAE, RMSE, MAPE).
9. **Predicciones Futuras**: Generar pronósticos para períodos futuros Calcular intervalos de confianza.

Este proceso es iterativo, y es posible que se necesite volver a pasos anteriores para refinar el modelo basándose en los resultados de los diagnósticos y validaciones.

**Descomposición de la serie**:

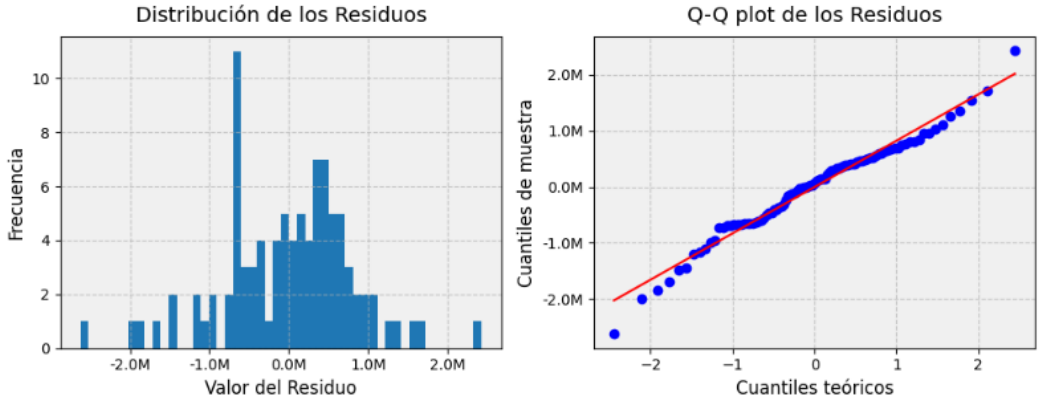



El componente estacional es significativo y confirma la elección de un modelo SARIMA. La tendencia no lineal sugiere que podría ser necesario aplicar diferenciación en el modelo SARIMA.

```
Matriz de Correlación:
                Original  Tendencia  Estacionalidad       Residuo
Original        1.000000   0.739665    6.289097e-01  4.141384e-01
Tendencia       0.739665   1.000000   -1.066656e-02  1.855513e-01
Estacionalidad  0.628910  -0.010667    1.000000e+00 -1.877188e-16
Residuo         0.414138   0.185551   -1.877188e-16  1.000000e+00
```

La serie original está fuertemente correlacionada con la tendencia (0.74) y la estacionalidad (0.63), lo cual es esperado. La correlación entre tendencia y estacionalidad es prácticamente cero (-0.01), indicando una buena separación de estos componentes. Los residuos muestran cierta correlación con la serie original (0.41) y la tendencia (0.19), lo que podría indicar que aún hay patrones no capturados completamente.

**Residuos**:



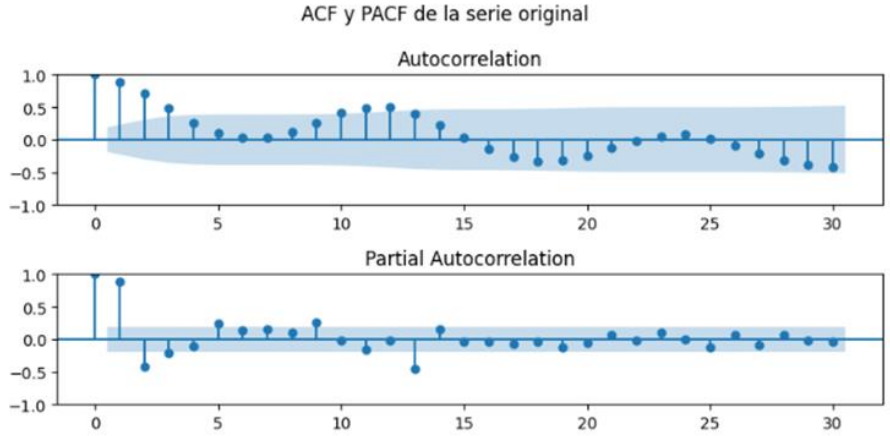

```
Test de estacionariedad para los residuos:
Resultados del Test de Dickey-Fuller Aumentado:
Estadístico de prueba: -6.4456437269789655
p-valor: 1.5683917484812542e-08
Valores críticos:
        1%: -3.50434289821397
        5%: -2.8938659630479413
        10%: -2.5840147047458037
```

El **test de Dickey-Fuller Aumentado** muestra un p-valor muy bajo (1.57e-08), lo que indica que los residuos son estacionarios. Esto es positivo para el modelado SARIMA, ya que sugiere que la diferenciación y la eliminación de la estacionalidad han sido efectivas.

La distribución es aproximadamente simétrica (-0.3138) y ligeramente leptocúrtica (0.9662), lo cual es cercano a una distribución normal.

**Graficar Función de Autocorrelación (ACF) Graficar Función de Autocorrelación Parcial (PACF). Identificar posibles órdenes para p, d, q y sus componentes estacionales.**

```
Resultados del Test de Dickey-Fuller Aumentado para original:
Estadístico de prueba: -2.2456302166064743
p-valor: 0.19010287745414545
Valores críticos:
        1%: -3.502704609582561
        5%: -2.8931578098779522
        10%: -2.583636712914788
Conclusión: La serie original no es estacionaria (no podemos rechazar la hipótesis nula)
```




El **Test de Dickey-Fuller Aumentado** confirma que la serie original no es estacionaria, algo que ya observamos en el análisis de la descomposición. La no estacionariedad sugiere que necesitaremos aplicar diferenciación no estacional (d=1) en nuestro modelo SARIMA.

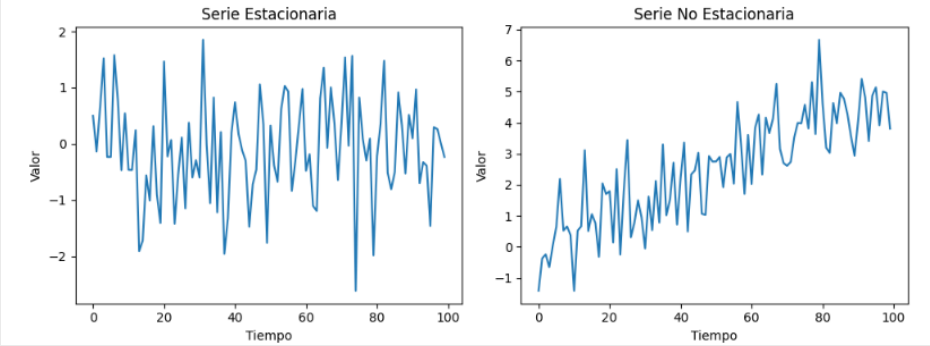
La **Función de Autocorrelación (ACF)** muestra una disminución lenta, lo cual es típico de series no estacionarias. Hay un pico significativo en el lag 12 (0.5092), indicando una fuerte componente estacional anual, lo que invita a considerar una diferenciación estacional (D=1) y términos MA estacionales.

**Conclusión**: La serie requiere al menos una diferenciación no estacional (d=1) para ser estacionaria.

**¿Por qué debemos trabajar con series estacionarias?**

La necesidad de hacer una serie temporal estacionaria es un concepto fundamental en el análisis de series temporales y es crucial para muchos métodos de modelado.

Definición de estacionariedad: Una serie temporal es estacionaria cuando sus propiedades estadísticas (como la media, varianza y autocorrelación) se mantienen constantes a lo largo del tiempo.




Importancia de la estacionariedad:

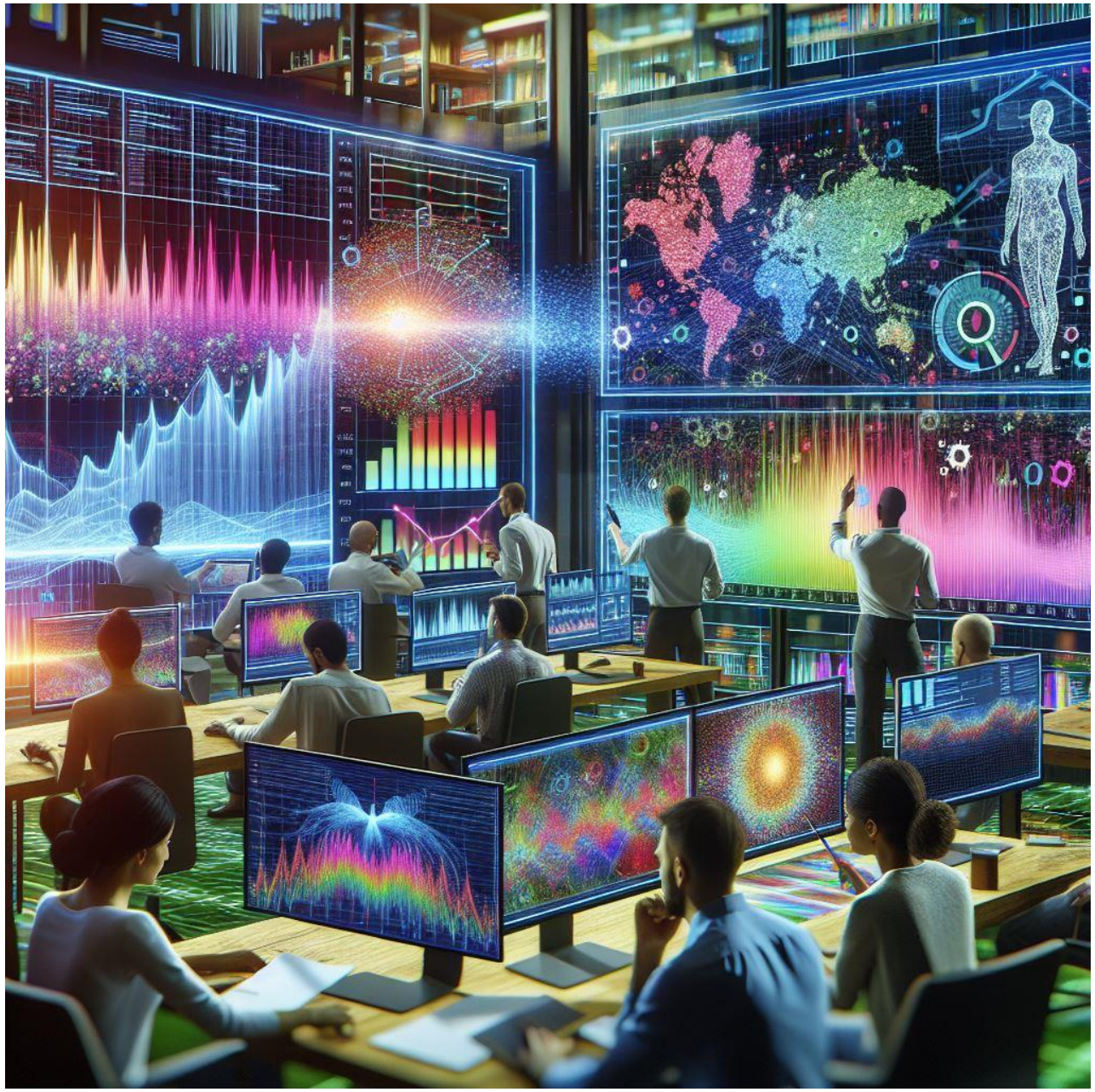
1. Predictibilidad: Las series estacionarias son más fáciles de predecir.
2. Consistencia: Los modelos estadísticos asumen generalmente que los datos son estacionarios.
3. Validez de los modelos: Muchas técnicas de análisis de series temporales requieren datos estacionarios para ser válidas.

Métodos para lograr la estacionariedad:

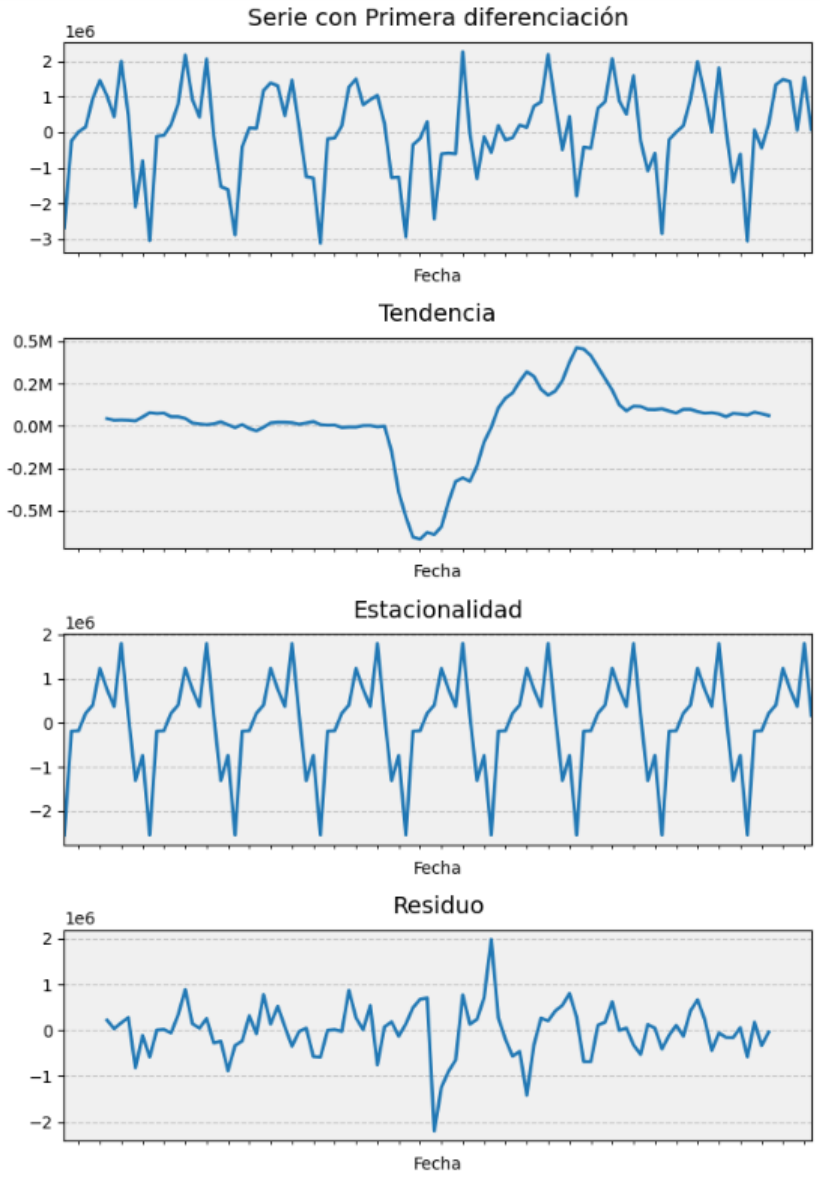
1. Diferenciación: Resta cada observación de la anterior para eliminar tendencias (una diferenciación excesiva lleva a perdida de datos)
2. Transformación logarítmica: Puede estabilizar la varianza.
3. Desestacionalización: Elimina patrones estacionales recurrentes.

Beneficios de trabajar con series estacionarias:

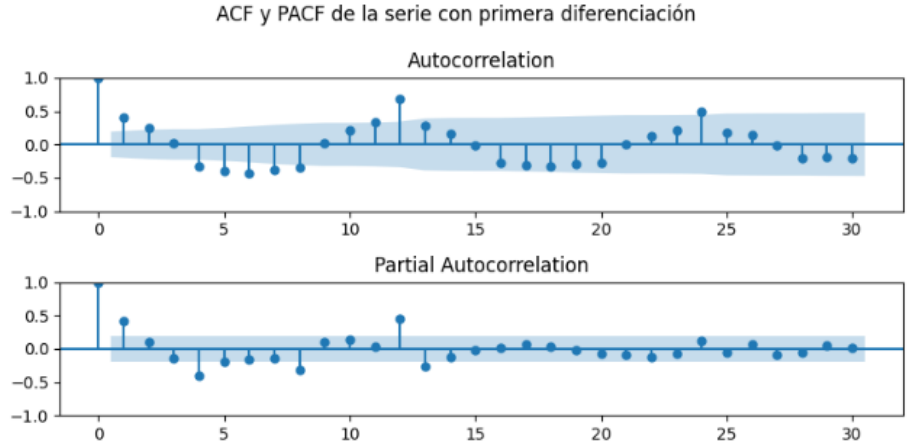
1. Simplifica el análisis y la interpretación de los datos.
2. Permite el uso de una gama más amplia de modelos estadísticos.
3. Mejora la precisión de las predicciones.



**Primera diferenciación de la serie (d=1)**




```
Resultados del Test de Dickey-Fuller Aumentado para con primera diferenciación:
Estadístico de prueba: -2.2417782979580183
p-valor: 0.19143311483708703
Valores críticos:
        1%: -3.503514579651927
        5%: -2.893507960466837
        10%: -2.583823615311909
Conclusión: La serie con primera diferenciación no es estacionaria (no podemos rechazar la hipótesis nula)
```




El **Test de Dickey-Fuller** Aumentado para la serie diferenciada muestra que la serie aún no es estacionaria, resultado probablemente de su estructura compleja producto del shock pandémico.

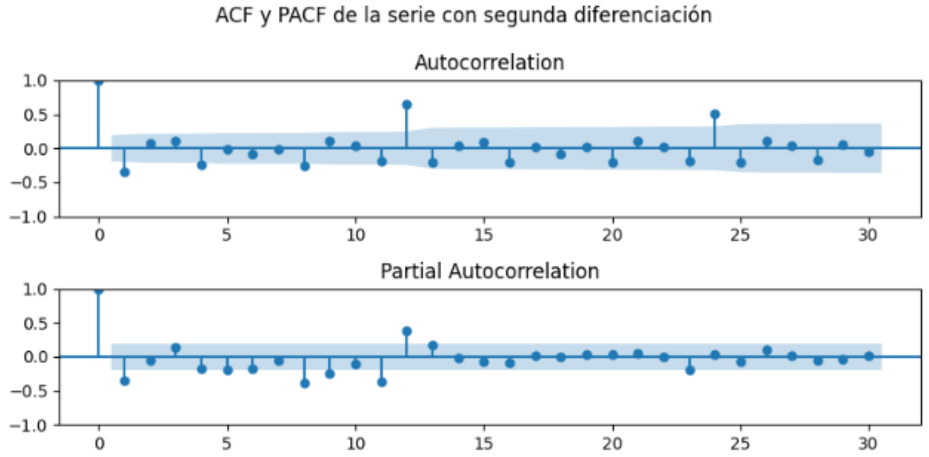
**Función de Autocorrelación (ACF) de la serie diferenciada**:
Hay un patrón oscilatorio en los primeros lags, con valores negativos significativos en los lags 4-8.El pico en el lag 12 (0.6906) es aún más pronunciado que en la serie original, indicando una fuerte estacionalidad que persiste después de la diferenciación.

**Función de Autocorrelación Parcial (PACF) de la serie diferenciada**:
El valor en lag 1 (0.4127) sugiere un componente AR(1) en la parte no estacional. Hay valores significativos en los lags 4 y 8, lo que podría indicar la necesidad de términos AR adicionales. El pico en el lag 12 (0.5638) es muy significativo, sugiriendo un fuerte componente AR estacional.

**Conclusiones:**
Necesidad de más diferenciación: Dado que la serie sigue sin ser estacionaria después de la primera diferenciación, considerar aplicar una segunda diferencia no estacional (d=2) en tu modelo SARIMA. Alternativamente, se podría explorar una transformación logarítmica antes de la diferenciación.

## Segunda diferenciación de la serie (d=2)

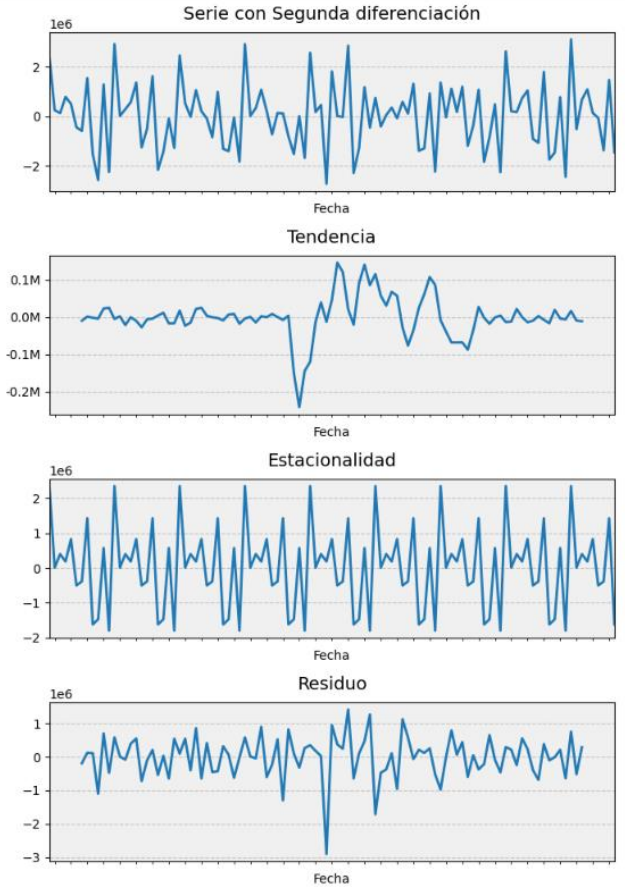
```
Resultados del Test de Dickey-Fuller Aumentado para con segunda diferenciación:
Estadístico de prueba: -5.7736894529296
p-valor: 5.319768472161869e-07
Valores críticos:
        1%: -3.502704609582561
        5%: -2.8931578098779522
        10%: -2.583636712914788
Conclusión: La serie con segunda diferenciación es estacionaria (rechazamos la hipótesis nula)
```




El test de Dickey-Fuller Aumentado para la serie con segunda diferenciación confirma que tras realizar esta ya es estacionaria lo que confirma que necesitaremos un término d=2 en nuestro modelo SARIMA.

Función de Autocorrelación (ACF) de la serie con segunda diferenciación: El valor en lag 1 (-0.3504) es negativo y significativo, lo que sugiere un componente de media móvil (MA) en el modelo. Hay algunos valores significativos en otros lags (4, 8), pero son menos pronunciados. El pico en el lag 12 (0.6404) sigue siendo muy significativo, confirmando la fuerte componente estacional anual.

Función de Autocorrelación Parcial (PACF) de la serie con segunda diferenciación: El valor en lag 1 (-0.3538) es significativo, sugiriendo un componente autorregresivo (AR). Hay valores significativos en los lags 8 y 11, lo que podría indicar la necesidad de términos AR de orden superior. El pico en el lag 12 (0.4079) confirma la necesidad de un componente estacional autorregresivo.




**Basándonos en estos resultados, podemos hacer las siguientes estimaciones para nuestro modelo SARIMA**:

Componente no estacional: d=2 (confirmado por la necesidad de segunda diferenciación para lograr estacionariedad), p=1 o p=2 (basado en los valores significativos en los primeros lags de PACF), q=1 o q=2 (basado en el patrón de ACF)

Componente estacional: D=1 (una diferenciación estacional suele ser suficiente), P=1 (debido al pico significativo en lag 12 de PACF), Q=1 (debido al pico significativo en lag 12 de ACF)

Modelos SARIMA sugeridos: SARIMA(1,2,1)(1,1,1)12 SARIMA(2,2,1)(1,1,1)12 SARIMA(1,2,2)(1,1,1)12

**Estimación del mejor modelo SARIMA con Auto_Arima**

Auto_arima es una función de la biblioteca pmdarima en Python que automatiza el proceso de selección de parámetros para modelos ARIMA y SARIMA.

- **Funcionamiento de auto_arima:**

    1. Prueba diferentes combinaciones de parámetros (p, d, q) para la parte no estacional y (P, D, Q) para la parte estacional.
    2. Utiliza criterios de información como AIC (Akaike Information Criterion) o BIC (Bayesian Information Criterion) para evaluar la calidad de cada modelo.
    3. Selecciona el modelo con el mejor balance entre ajuste y complejidad.

- **Ventajas:**

    1. Ahorra tiempo al automatizar la búsqueda de parámetros.
    2. Puede descubrir modelos que podrían pasarse por alto en una búsqueda manual.

- **Limitaciones:**

    1. Puede no tener en cuenta consideraciones específicas del dominio o características únicas de tus datos.
    2. A veces puede seleccionar modelos que son matemáticamente óptimos, pero no tienen sentido práctico.

En nuestro caso, vamos a guiar a auto_arima con nuestro conocimiento previo basado en el análisis de nuestra seria (tests de estacionariedad de Dickey-Fuller para estimar el orden de diferenciación (d, D), análisis de ACF y PACF para ayudar a sugerir rangos iniciales para p, q, P, y Q) y vamos a usar la flexibilidad que nos brinda auto_arima para mostrarle que parta por diferenciaciones d=2 y D=1 (estacional).

Cómo comentario adicional comentar que el modelo sugerido por Auto_Arima sin establecer como punto de partida d=2 y D=1 dio cómo modelo optimo un ARIMA(3,0,0)(0,0,2)[12], con diferenciaciones d=0 y D=0 que obtuvo unos muy malos resultados en comparación con nuestro modelo final al que llego a través de una búsqueda guiada, lo que trae como reflexión final buscar siempre un equilibrio entre nuestra experiencia previa con los datos y la facilidad de probar entre distintas combinaciones que nos permite Auto_Arima para seleccionar el mejor modelo entre múltiples posibilidades.

```
# Vamos a dejar que autoarima seleccione los mejores modelos , partiendo de d=2 y D=1
auto_model = auto_arima(df_arima, seasonal=True, m=12,
                        start_p=0, start_q=0, max_p=5, max_q=5,
                        d=2, max_d=3,  # Permitimos hasta 3 diferenciaciones
                        D=1, max_D=2,  # Permitimos hasta 2 diferenciaciones estacionales
                        max_P=2, max_Q=2,
                        trace=True, error_action='ignore', suppress_warnings=True,
                        stepwise=True)

# Obtener el orden óptimo
order = auto_model.order
seasonal_order = auto_model.seasonal_order

print(f"Orden óptimo: {order}")
print(f"Orden estacional óptimo: {seasonal_order}")
```

```
Performing stepwise search to minimize aic
 ARIMA(0,2,0)(1,1,1)[12]             : AIC=2808.037, Time=0.13 sec
 ARIMA(0,2,0)(0,1,0)[12]             : AIC=2826.578, Time=0.01 sec
 ARIMA(1,2,0)(1,1,0)[12]             : AIC=2817.156, Time=0.07 sec
 ARIMA(0,2,1)(0,1,1)[12]             : AIC=2799.744, Time=0.20 sec
 ARIMA(0,2,1)(0,1,0)[12]             : AIC=inf, Time=0.04 sec
 ARIMA(0,2,1)(1,1,1)[12]             : AIC=2789.822, Time=0.17 sec
 ARIMA(0,2,1)(1,1,0)[12]             : AIC=inf, Time=0.14 sec
 ARIMA(0,2,1)(2,1,1)[12]             : AIC=2788.065, Time=0.50 sec
 ARIMA(0,2,1)(2,1,0)[12]             : AIC=2796.460, Time=0.34 sec
 ARIMA(0,2,1)(2,1,2)[12]             : AIC=2787.004, Time=1.09 sec
 ARIMA(0,2,1)(1,1,2)[12]             : AIC=2788.647, Time=0.44 sec
 ARIMA(0,2,0)(2,1,2)[12]             : AIC=inf, Time=1.02 sec
 ARIMA(1,2,1)(2,1,2)[12]             : AIC=2783.856, Time=1.02 sec
 ARIMA(1,2,1)(1,1,2)[12]             : AIC=2783.265, Time=0.54 sec
 ARIMA(1,2,1)(0,1,2)[12]             : AIC=2781.267, Time=0.38 sec
 ARIMA(1,2,1)(0,1,1)[12]             : AIC=2790.964, Time=0.17 sec
 ARIMA(1,2,1)(1,1,1)[12]             : AIC=2783.823, Time=0.20 sec
 ARIMA(0,2,1)(0,1,2)[12]             : AIC=2786.732, Time=0.27 sec
 ARIMA(1,2,0)(0,1,2)[12]             : AIC=2796.846, Time=0.26 sec
 ARIMA(2,2,1)(0,1,2)[12]             : AIC=2783.176, Time=0.55 sec
 ARIMA(1,2,2)(0,1,2)[12]             : AIC=2783.504, Time=0.60 sec
 ARIMA(0,2,0)(0,1,2)[12]             : AIC=inf, Time=0.60 sec
 ARIMA(0,2,2)(0,1,2)[12]             : AIC=2783.224, Time=0.43 sec
 ARIMA(2,2,0)(0,1,2)[12]             : AIC=2795.980, Time=0.31 sec
 ARIMA(2,2,2)(0,1,2)[12]             : AIC=2785.349, Time=0.93 sec
 ARIMA(1,2,1)(0,1,2)[12] intercept   : AIC=2783.790, Time=0.44 sec

Best model:  ARIMA(1,2,1)(0,1,2)[12]
Total fit time: 10.875 seconds
Orden óptimo: (1, 2, 1)
Orden estacional óptimo: (0, 1, 2, 12)
```

## ❑ Mejor modelo seleccionado por Auto_Arima

**El modelo seleccionado por auto_arima: ARIMA(1,2,1)(0,1,2)[12]**

Parte no estacional: (1,2,1):

p = 1: Un término autorregresivo
d= 2: Dos diferenciaciones, como sugería el test de Dickey-Fuller
q= 1: Un término de media móvil

Parte estacional: (0,1,2)[12]:

P = 0: No hay término autorregresivo estacional
D = 1: Una diferenciación estacional
Q = 2: Dos términos de media móvil estacional

Entrenamos el modelo, reservando 2 años para testarlo y vemos los resultados.

```python
# Dividir los datos en conjuntos de entrenamiento y prueba
split_date = '2022-08-31'  # Ajusta esta fecha según tus datos
train = df_arima[df_arima.index <= split_date]
test = df_arima[df_arima.index > split_date]

# Ajustar el modelo SARIMA con los órdenes obtenidos
model = SARIMAX(train, order=order, seasonal_order=seasonal_order)
results = model.fit()

# Imprimir resumen del modelo
print(results.summary())

# Realizar predicciones
forecast = results.get_forecast(steps=len(test))
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Calcular métricas de error
mae = mean_absolute_error(test, forecast_mean)
rmse = np.sqrt(mean_squared_error(test, forecast_mean))
mape = np.mean(np.abs((test - forecast_mean) / test)) * 100
r2 = r2_score(test, forecast_mean)
```

SARIMAX Results

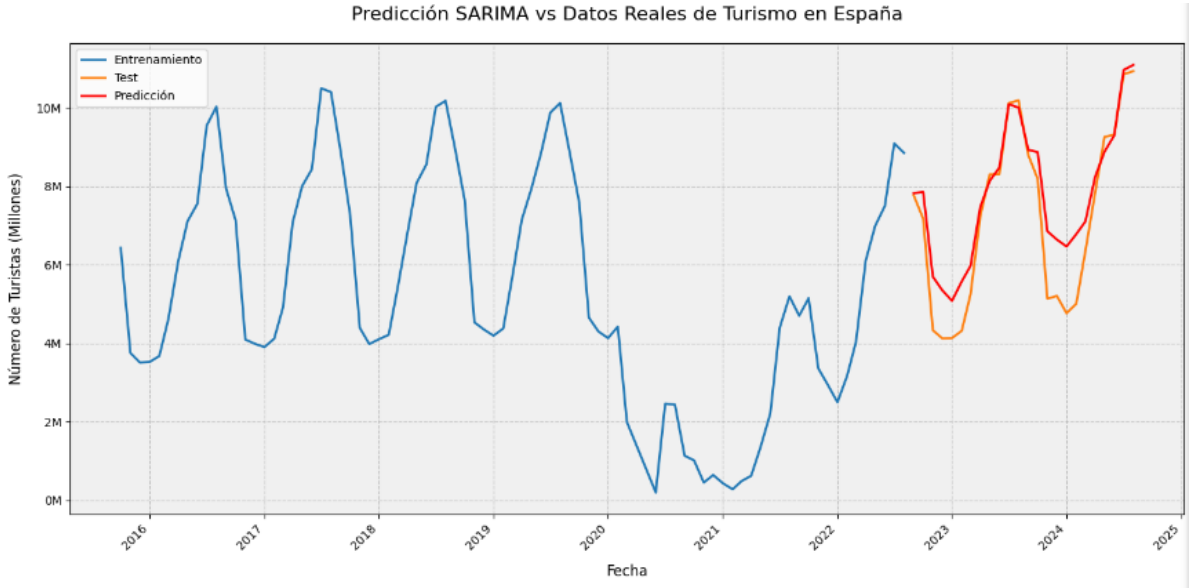
| | | | |
|---|---|---|---|
| Dep. Variable: | Valor | No. Observations: | 83 |
| Model: | SARIMAX(1, 2, 1)x(0, 1, [1, 2], 12) | Log Likelihood | -1035.982 |
| Date: | Sat, 12 Oct 2024 | AIC | 2081.964 |
| Time: | 11:30:44 | BIC | 2093.135 |
| Sample: | 10-01-2015 - 08-01-2022 | HQIC | 2086.396 |
| Covariance Type: | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| ar.L1 | 0.3615 | 0.294 | 1.229 | 0.219 | -0.215 | 0.938 |
| ma.L1 | -0.9592 | 0.123 | -7.806 | 0.000 | -1.200 | -0.718 |
| ma.S.L12 | -0.4294 | 0.154 | -2.780 | 0.005 | -0.732 | -0.127 |
| ma.S.L24 | -0.2663 | 0.122 | -2.178 | 0.029 | -0.506 | -0.027 |
| sigma2 | 9.067e+11 | 1.52e-13 | 5.98e+24 | 0.000 | 9.07e+11 | 9.07e+11 |

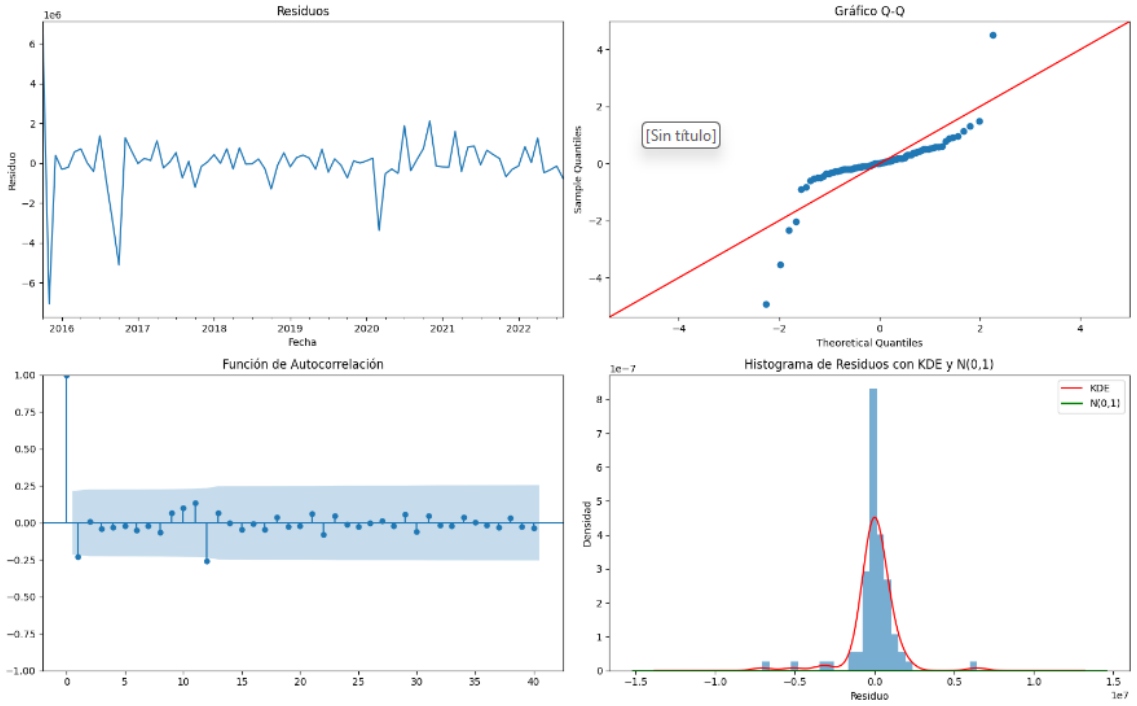
| | | | |
|---|---|---|---|
| Ljung-Box (L1) (Q): | 0.05 | Jarque-Bera (JB): | 109.85 |
| Prob(Q): | 0.83 | Prob(JB): | 0.00 |
| Heteroskedasticity (H): | 2.09 | Skew: | -0.84 |
| Prob(H) (two-sided): | 0.08 | Kurtosis: | 8.95 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number inf. Standard errors may be unstable.

Métricas de error:
MAE: 679199.77
RMSE: 898979.77
MAPE: 13.01%
$R^2$: 0.8368




Aunque hay algunas advertencias sobre la matriz de covarianza que podrían indicar algunos problemas de estabilidad en la estimación de parámetros, **el modelo parece tener un buen ajuste según las métricas de error y las predicciones sobre el periodo de test parecen ajustarse bastante bien a los datos reales.**




```
Test de Ljung-Box para autocorrelación:
     lb_stat  lb_pvalue
10  6.861334   0.738468

Valores que se desvían más de 2 desviaciones típicas:
Fecha Completa
2015-10-01    6.432341e+06
2015-11-01   -7.062427e+06
2016-09-01   -2.934714e+06
2016-10-01   -5.110037e+06
2020-03-01   -3.384785e+06
dtype: float64

Test de normalidad para residuos (D'Agostino y Pearson):
p-valor: 9.396820048441384e-10

Test de Dickey-Fuller para estacionariedad de residuos:
Estadístico ADF: -13.125603342509736
p-valor: 1.53138445770116e-24
Valores críticos:
   1%: -3.512738056978279
   5%: -2.8974898650628984
   10%: -2.585948732897085
```

**Test de Ljung-Box para autocorrelación:**

Estadístico de prueba: 6.861334p-valor: 0.738468

Interpretación: El p-valor (0.738468) es mucho mayor que 0.05, lo que indica que no hay evidencia significativa de autocorrelación en los residuos hasta el rezago 10. Esto es una buena señal, ya que sugiere que el modelo SARIMA ha capturado adecuadamente la estructura de dependencia temporal en los datos. No parece haber información significativa sin explicar en los residuos.

**Valores atípicos:**

Se identificaron tan sólo cinco fechas con residuos atípicos, el último el valor atípico más reciente (2020-03-01) coincide exactamente con el inicio de la pandemia en Europa.

**Test de normalidad para residuos (D'Agostino y Pearson):**

p-valor: 9.396820048441384e-10

Interpretación: Este p-valor extremadamente bajo (mucho menor que 0.05) indica un rechazo contundente de la hipótesis nula de normalidad. Esto sugiere que los residuos no siguen una distribución normal, probablemente poque el shock de la pandemia ha introducido una gran cantidad de variabilidad y asimetría en los datos.

**Test de Dickey-Fuller para estacionariedad de residuos**:

Estadístico ADF: -13.125603342509736p-valor: 1.53138445770116e-24

El estadístico ADF es muy negativo y el p-valor es extremadamente bajo (mucho menor que 0.05), lo que indica un fuerte rechazo de la hipótesis nula de no estacionariedad. Esto es una buena señal, ya que sugiere que los residuos son estacionarios. El estadístico ADF (-13.12) es más negativo que todos los valores críticos, incluso al nivel del 1%, lo que refuerza esta conclusión. La estacionariedad de los residuos es crucial, ya que indica que el modelo ha capturado adecuadamente cualquier tendencia o estacionalidad en los datos originales.

## Conclusiones Generales Modelo SARIMA

El modelo SARIMA parece haber capturado bien la estructura de autocorrelación en los datos, como lo indica el test de Ljung-Box.

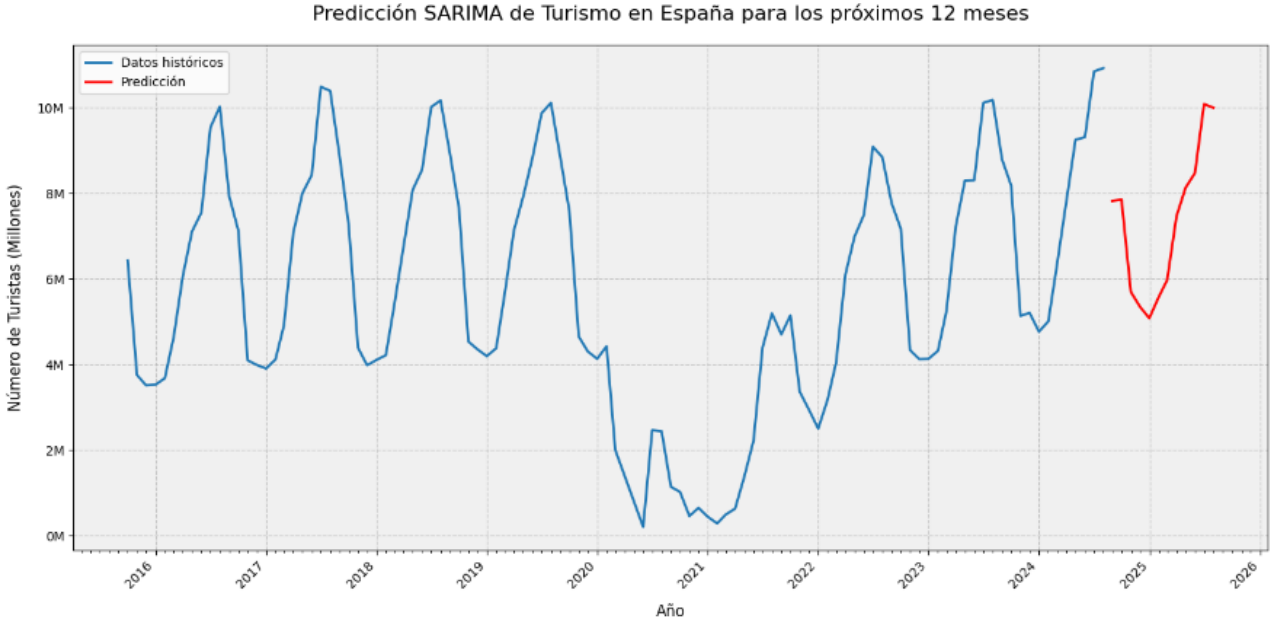
La no normalidad de los residuos indica que las predicciones del modelo y sus intervalos de confianza deben interpretarse con cautela. Podría ser beneficioso explorar técnicas de modelado más robustas o considerar transformaciones de los datos.

La estacionariedad de los residuos es un aspecto positivo, indicando que el modelo ha capturado adecuadamente las tendencias y patrones estacionales en los datos originales.

En general, **aunque el modelo muestra fortalezas y sus predicciones para los próximos 12 meses parecen recoger bastante bien la recuperación postpandemia, hay áreas de mejora potencial**:

- Podría ser beneficioso explorar técnicas de modelado más robustas o considerar transformaciones de los datos.

- La inclusión de variables exógenas para capturar eventos especiales cómo la pandemia o los meses de temporada alta o baja podría permitir construir modelos más precisos y flexibles para manejar la no normalidad.




En la próxima sección emplearemos algoritmos de Machine Learning y trataremos de comprobar si estos son capaces de modelar de mejor forma el shock pandémico y resultan aún más efectivos en sus precisiones.

## 2. 4. Modelización con Algoritmos de Machine Learning

Para la modelización de nuestra serie temporal utilizaremos los siguientes algoritmos de ML:

**Regresión lineal**

Es uno de los métodos más simples y ampliamente utilizados. Modela la relación entre una variable dependiente y una o más variables independientes mediante una línea recta. Aunque es fácil de interpretar, su capacidad para capturar patrones complejos y no lineales es limitada.

**Random Forest**

Es un conjunto de árboles de decisión que mejora la precisión de las predicciones al reducir el sobreajuste. Es robusto y puede manejar datos no lineales y de alta dimensionalidad, lo que lo hace adecuado para predecir la demanda turística basada en múltiples factores.

**XGBoost (Extreme Gradient Boosting)**

Es un algoritmo de boosting que optimiza el rendimiento mediante la combinación de múltiples modelos débiles. Es conocido por su eficiencia y precisión, siendo ideal para capturar patrones complejos en series temporales turísticas.

**Gradient Boosting Regressor**

Similar a XGBoost, pero se centra en minimizar el error de predicción mediante la construcción secuencial de modelos. Es efectivo para manejar datos con ruido y puede capturar relaciones no lineales en las series temporales.

**CatBoost**

Es un algoritmo de boosting que maneja eficientemente variables categóricas sin necesidad de preprocesamiento extenso. Es especialmente útil en el sector turístico, donde los datos categóricos (como tipos de turistas o destinos) son comunes.

**¿Qué es el Boosting?**

El **boosting** es una técnica de machine learning que se utiliza para mejorar la precisión de los modelos predictivos. Funciona combinando varios modelos débiles (modelos que tienen un rendimiento ligeramente mejor que el azar) para crear un modelo fuerte y robusto.

Cómo funciona:

**1. Entrenamiento Secuencial**: Los modelos se entrenan de manera secuencial, y cada nuevo modelo intenta corregir los errores cometidos por los modelos anteriores.

**2. Ponderación de Modelos**: Cada modelo tiene un peso asignado basado en su precisión. Los modelos que cometen menos errores tienen más peso en la predicción final.

**3. Actualización de Pesos**: Los datos mal clasificados por los modelos anteriores reciben más peso, de modo que los modelos posteriores se centren en corregir estos errores.

Algunos algoritmos populares de boosting incluyen **AdaBoost**, **Gradient Boosting** y **XGBoost**. Estos algoritmos son muy efectivos para tareas de clasificación y regresión, y suelen ser utilizados en competiciones de machine learning debido a su alta precisión.

## Aplicación de Algoritmos de Machine Learning en la Estimación de Turistas en España

La pandemia de COVID-19 ha tenido un impacto significativo en el turismo global, y España no ha sido la excepción. Para mejorar la precisión de las estimaciones de turistas, es crucial incorporar variables que reflejen estos cambios abruptos y otros factores estacionales. En este contexto, los algoritmos de machine learning ofrecen una herramienta poderosa para modelar series temporales complejas.

Para ello incorporaremos a nuestra serie las siguientes variables:

**1. Variable Dummy para la Pandemia**: La pandemia introdujo cambios drásticos en los patrones de turismo. Al incluir variables dummy que indiquen los meses afectados por la pandemia, podemos capturar estos efectos específicos y mejorar la precisión del modelo.

**2. Variables Dummy para Temporada Alta y Baja**: El turismo en España muestra una fuerte estacionalidad, con picos en los meses de verano y caídas en invierno. Incluir variables dummy para estos periodos ayuda a capturar la estacionalidad inherente a los datos.

**3. Medias Móviles y Exponenciales**: Las medias móviles y exponenciales suavizan las series temporales, ayudando a identificar tendencias subyacentes y reducir la volatilidad.

**4. Lags**: Los valores pasados de la serie temporal pueden ser predictivos de los valores futuros. Incluir lags permite al modelo aprender de la autocorrelación en los datos.

**Implementación:**

```python
# Crear variable dummy para Temporada Alta
df['Temp_Alta'] = df.index.month.isin([6, 7, 8, 9]).astype(int)

# Crear variable dummy para Temporada Baja
df['Temp_Baja'] = df.index.month.isin([11, 12, 1, 2]).astype(int)

# Crear variable dummy para Pandemia
df['Pandemia'] = ((df.index >= '2020-03-01') & (df.index <= '2021-06-30')).astype(int)
```

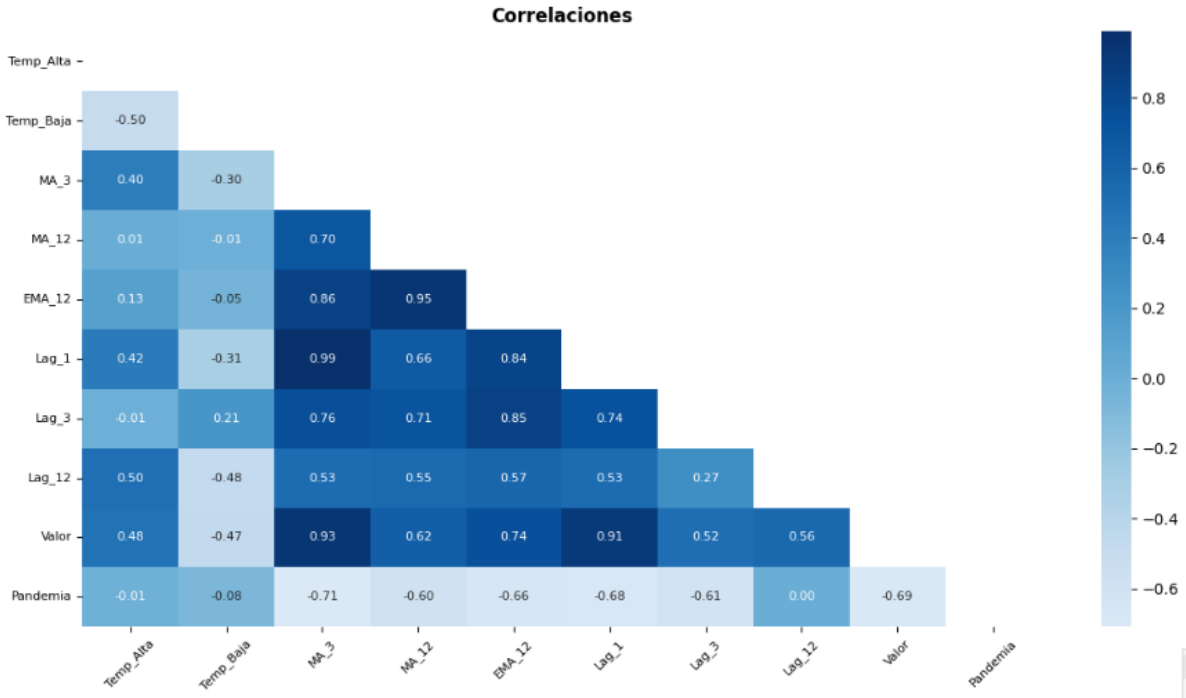
```python
# Función para añadir características de series temporales
def add_time_features(df):
    df = df.copy()
    df['MA_3'] = df['Valor'].rolling(window=3).mean()
    df['MA_12'] = df['Valor'].rolling(window=12).mean()
    df['EMA_12'] = df['Valor'].ewm(span=12, adjust=False).mean()

    for i in [1, 3, 12]:
        df[f'Lag_{i}'] = df['Valor'].shift(i)

    return df
```

**Nuestro dataframe enriquecido: df_enhanced**

| Fecha | Temp_Alta | Temp_Baja | MA_3 | MA_12 | EMA_12 | Lag_1 | Lag_3 | Lag_12 | Valor | Pandemia |
|---|---|---|---|---|---|---|---|---|---|---|
| 2016-10-01 | 0 | 0 | 8.365934e+06 | 6.207835e+06 | 7.089420e+06 | 7932607.0 | 9545792.0 | 6432341.0 | 7137723.0 | 0 |
| 2016-11-01 | 0 | 1 | 6.389531e+06 | 6.236456e+06 | 6.629242e+06 | 7137723.0 | 10027471.0 | 3754802.0 | 4098262.0 | 0 |
| 2016-12-01 | 0 | 1 | 5.075477e+06 | 6.276251e+06 | 6.223273e+06 | 4098262.0 | 7932607.0 | 3512914.0 | 3990446.0 | 0 |
| 2017-01-01 | 0 | 1 | 3.998148e+06 | 6.307851e+06 | 5.866729e+06 | 3990446.0 | 7137723.0 | 3526537.0 | 3905736.0 | 0 |
| 2017-02-01 | 0 | 1 | 4.005106e+06 | 6.344552e+06 | 5.597869e+06 | 3905736.0 | 4098262.0 | 3678726.0 | 4119137.0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2024-04-01 | 0 | 0 | 6.395974e+06 | 7.349504e+06 | 6.618856e+06 | 6347516.0 | 4768171.0 | 7228094.0 | 7831094.0 | 0 |
| 2024-05-01 | 0 | 0 | 7.811685e+06 | 7.429176e+06 | 7.024639e+06 | 7831094.0 | 5009311.0 | 8300384.0 | 9256446.0 | 0 |
| 2024-06-01 | 1 | 0 | 8.800330e+06 | 7.513008e+06 | 7.376764e+06 | 9256446.0 | 6347516.0 | 8307460.0 | 9313450.0 | 0 |
| 2024-07-01 | 1 | 0 | 9.807023e+06 | 7.574196e+06 | 7.911288e+06 | 9313450.0 | 7831094.0 | 10116916.0 | 10851172.0 | 0 |
| 2024-08-01 | 1 | 0 | 1.036512e+07 | 7.636245e+06 | 8.375821e+06 | 10851172.0 | 9256446.0 | 10186163.0 | 10930750.0 | 0 |

95 rows × 10 columns

**Matriz de correlación de nuestras variables:**




Vemos que en términos generales casi todas las variables creadas tienen muy alta correlación con nuestra variable objetivo (Valor: nº turistas ingresaron en ese mes), algo que es muy positivo para nuestro objetivo de obtener unas predicciones más precisas. Sin embargo, y a efectos de evitar problemas de multicolinealidad*, eliminaremos las variables más correlacionadas entre sí y dejaremos tan sólo las más relevantes.

| | Var1 | Var2 | Correlacion |
|---|---|---|---|
| 0 | MA_3 | Temp_Alta | 0.739977 |
| 1 | EMA_12 | MA_3 | 0.873597 |
| 2 | Lag_1 | Temp_Alta | 0.728983 |
| 3 | Lag_1 | MA_3 | 0.983539 |
| 4 | Lag_1 | EMA_12 | 0.855550 |
| 5 | Lag_3 | EMA_12 | 0.772100 |
| 6 | Lag_12 | Temp_Alta | 0.792326 |
| 7 | Lag_12 | Temp_Baja | -0.807669 |
| 8 | Lag_12 | MA_3 | 0.827118 |
| 9 | Lag_12 | Lag_1 | 0.803886 |
| 10 | Valor | Temp_Alta | 0.779925 |
| 11 | Valor | Temp_Baja | -0.808595 |
| 12 | Valor | MA_3 | 0.858884 |
| 13 | Valor | Lag_1 | 0.829120 |
| 14 | Valor | Lag_12 | 0.963240 |

Lo que nos lleva finalmente a seleccionar las siguientes variables:

```
# Seleccionamos las features que vamos a usar en nuestro entrenamiento, descartando las no significativas o altamente correlacionadas
selected_features = ['Lag_12','Lag_1', 'Temp_Alta','Temp_Baja', 'Pandemia']
```

```
Variables con significación >= 0.05 con Turistas:
MA_3         0.93
Lag_1        0.91
EMA_12       0.74
MA_12        0.62
Lag_12       0.56
Lag_3        0.52
Temp_Alta    0.48
Temp_Baja   -0.47
Pandemia    -0.69
Name: Valor, dtype: float64
```

* Multicolinealidad: Ocurre cuando dos o más variables independientes en el modelo están altamente correlacionadas entre sí. Puede llevar a estimaciones inestables y poco fiables de los coeficientes de regresión.

Problemas causados por la multicolinealidad: a) Coeficientes inestables: Pequeños cambios en los datos pueden resultar en grandes cambios en los coeficientes estimados. b) Errores estándar inflados: Lleva a intervalos de confianza más amplios y tests de hipótesis menos precisos. c) Dificultad en la interpretación: Puede ser difícil determinar la importancia relativa de cada variable. d) Overfitting: El modelo puede ajustarse demasiado a los datos de entrenamiento, reduciendo su capacidad de generalización.

Aunque la multicolinealidad es una preocupación significativa en modelos lineales (Regresión lineal), los algoritmos basados en árboles como Random Forest, XGBoost, Gradient Boosting y CatBoost son inherentemente más robustos frente a este fenómeno. Esto nos permite mantener un conjunto más amplio de variables predictoras, potencialmente capturando matices y patrones complejos en los datos turísticos que podrían perderse con un enfoque más restrictivo.

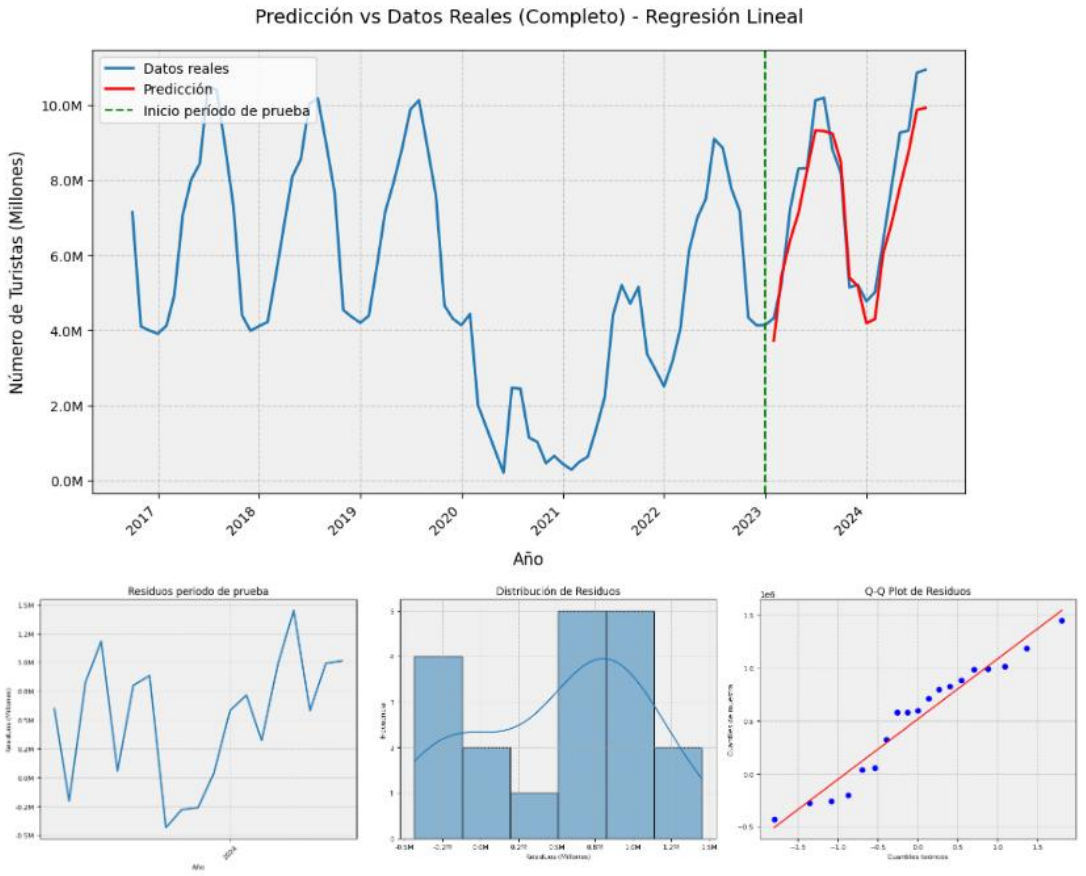
## Implementación de los algoritmos:

A continuación, mostramos, con tan sólo unas pocas líneas de código, lo fácil y rápido que es implementar y evaluar distintos algoritmos de machine learning utilizando bibliotecas de Python. La preparación de los datos, la escalación de características y la evaluación de modelos se pueden realizar con pocas líneas de código, permitiendo a los investigadores y analistas obtener resultados precisos y valiosos en poco tiempo.

```python
# Preparar los datos Preparación de los Datos: características (features) y la variable objetivo (target)
X = df_enhanced[['MA_3', 'MA_12', 'Lag_12','Lag_1', 'Temp_Alta','Temp_Baja', 'Pandemia']]
y = df_enhanced['Valor']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=False)

# Escalar las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Se define una función para evaluar los modelos:
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mae = mean_absolute_error(y_test, predictions)
    rmse = np.sqrt(mean_squared_error(y_test, predictions))
    r2 = r2_score(y_test, predictions)
    mape = np.mean(np.abs((y_test - predictions) / y_test)) * 100
    return predictions, mae, rmse, r2, mape

# Se Inicializan los modelos que queremos entrenar
models = {
    'Regresión Lineal': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'XGBoost': XGBRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting Regressor': GradientBoostingRegressor(n_estimators=100, random_state=42),
    'CatBoost': CatBoostRegressor(n_estimators=100, random_state=42, verbose=0)
}
```

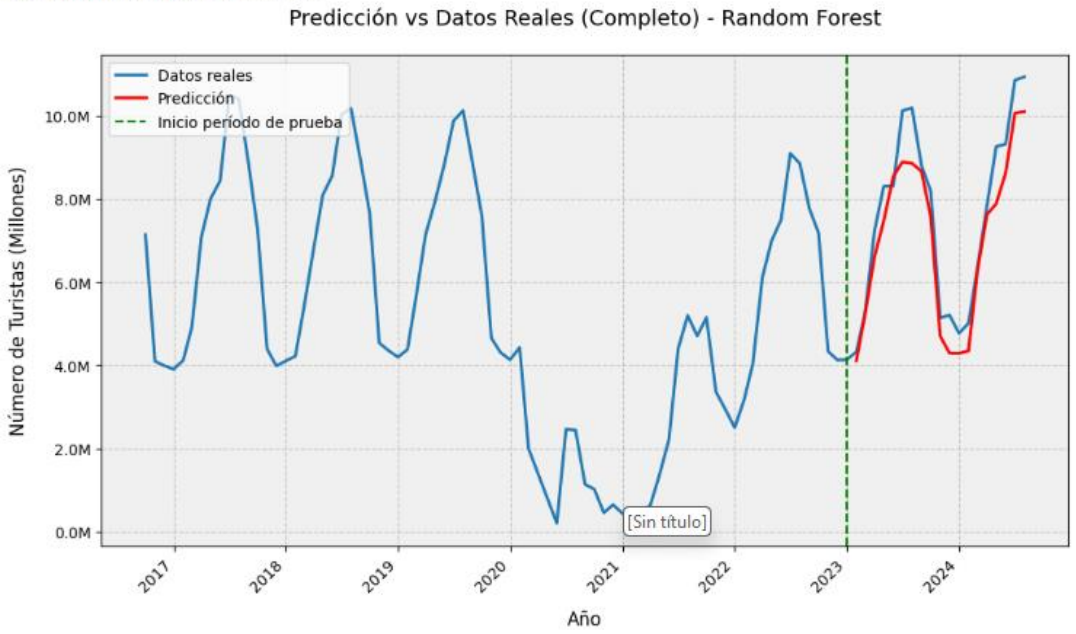
## Resultados de los algoritmos:

```
Generando gráficos para Regresión Lineal
```

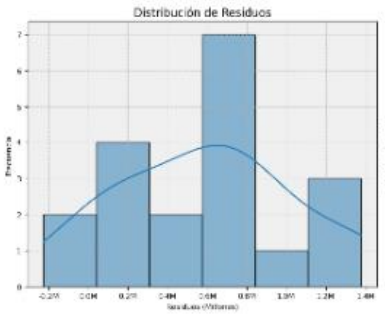


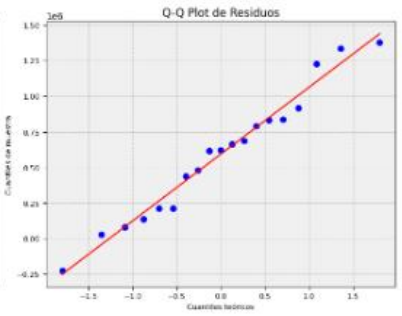

```
Tests estadísticos para los residuos:
Test de Shapiro-Wilk para normalidad: p-valor = 0.2598
Conclusión: Los residuos parecen seguir una distribución normal.

Estadístico de Durbin-Watson para autocorrelación: 0.7134
Conclusión: Hay evidencia de autocorrelación positiva en los residuos.
MAE: 642507.48
RMSE: 747948.18
R2: 0.88
MAPE: 8.38%
```

Generando gráficos para Random Forest





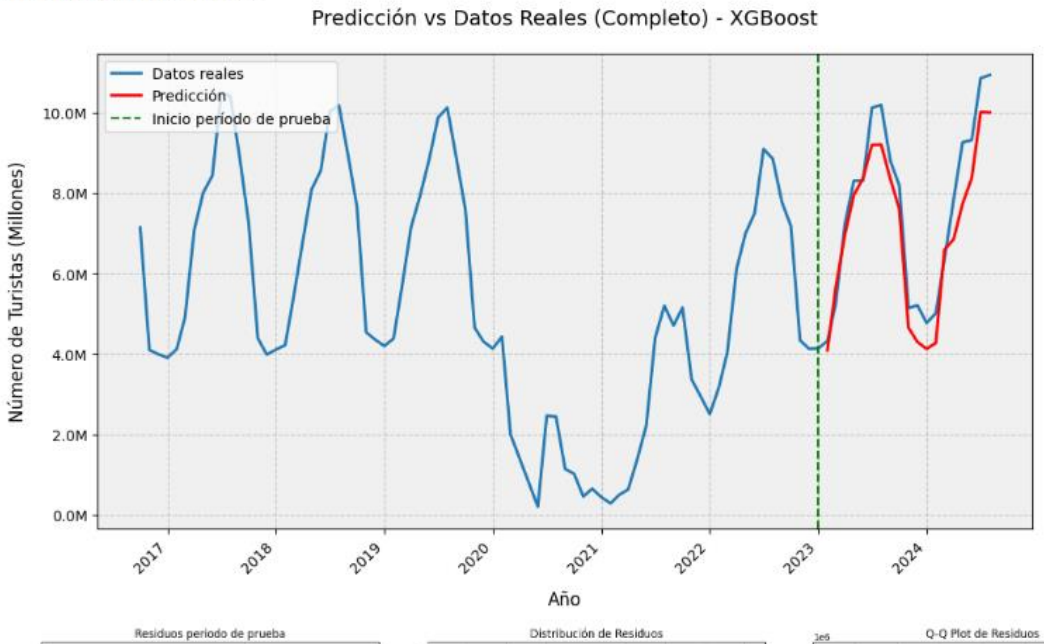


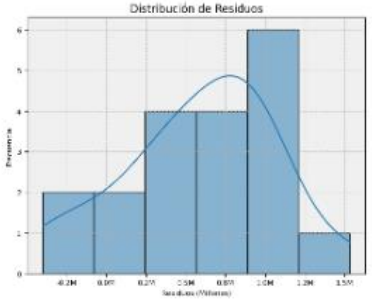

Tests estadísticos para los residuos:
Test de Shapiro-Wilk para normalidad: p-valor = 0.7470
Conclusión: Los residuos parecen seguir una distribución normal.

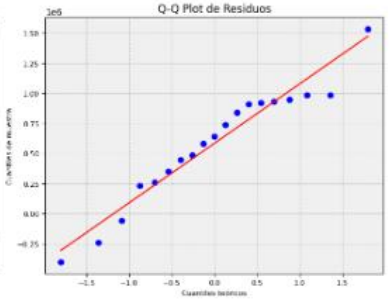
Estadístico de Durbin-Watson para autocorrelación: 0.7787
Conclusión: Hay evidencia de autocorrelación positiva en los residuos.
MAE: 616550.79
RMSE: 736514.78
R2: 0.88
MAPE: 7.98%

Generando gráficos para XGBoost





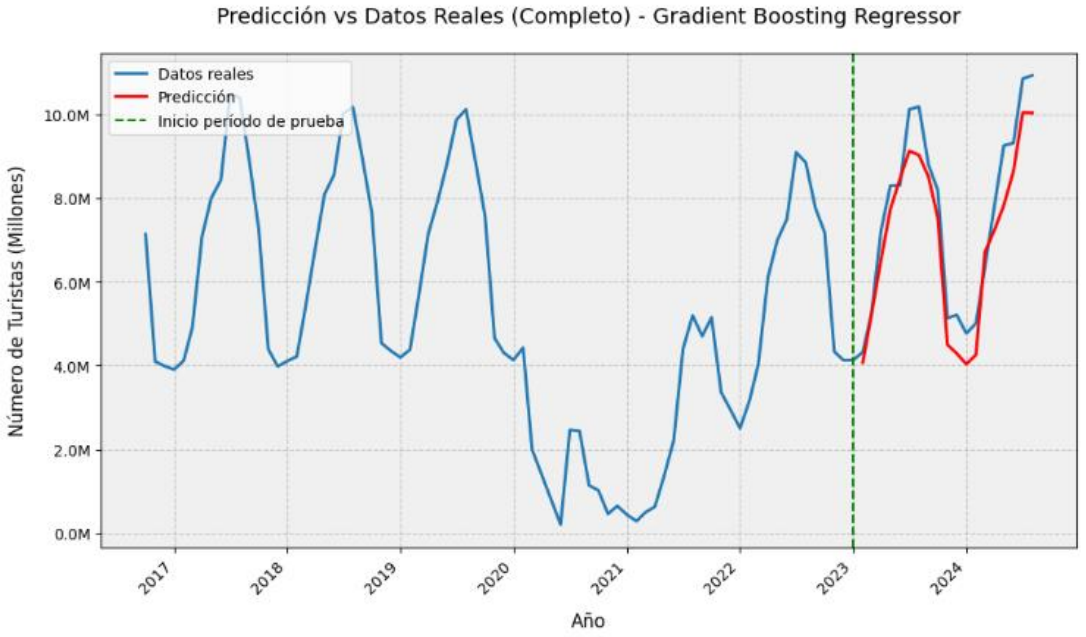


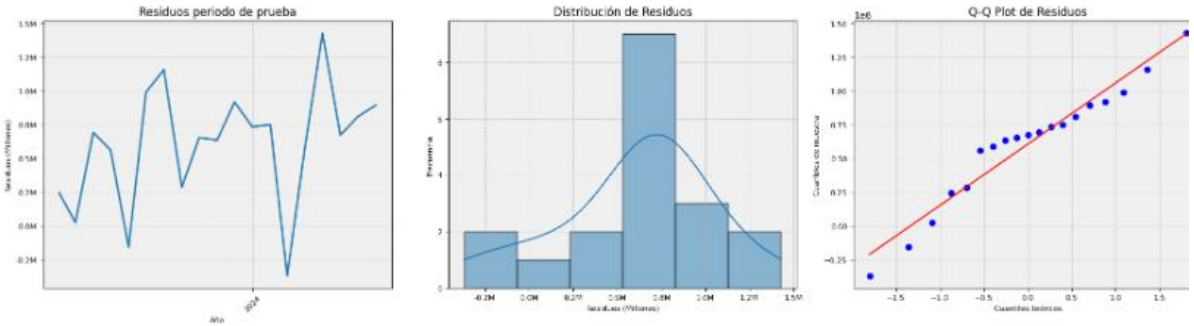




Tests estadísticos para los residuos:
Test de Shapiro-Wilk para normalidad: p-valor = 0.4543
Conclusión: Los residuos parecen seguir una distribución normal.

Estadístico de Durbin-Watson para autocorrelación: 0.5327
Conclusión: Hay evidencia de autocorrelación positiva en los residuos.
MAE: 657532.87
RMSE: 748326.47
R2: 0.88
MAPE: 8.79%

Generando gráficos para Gradient Boosting Regressor

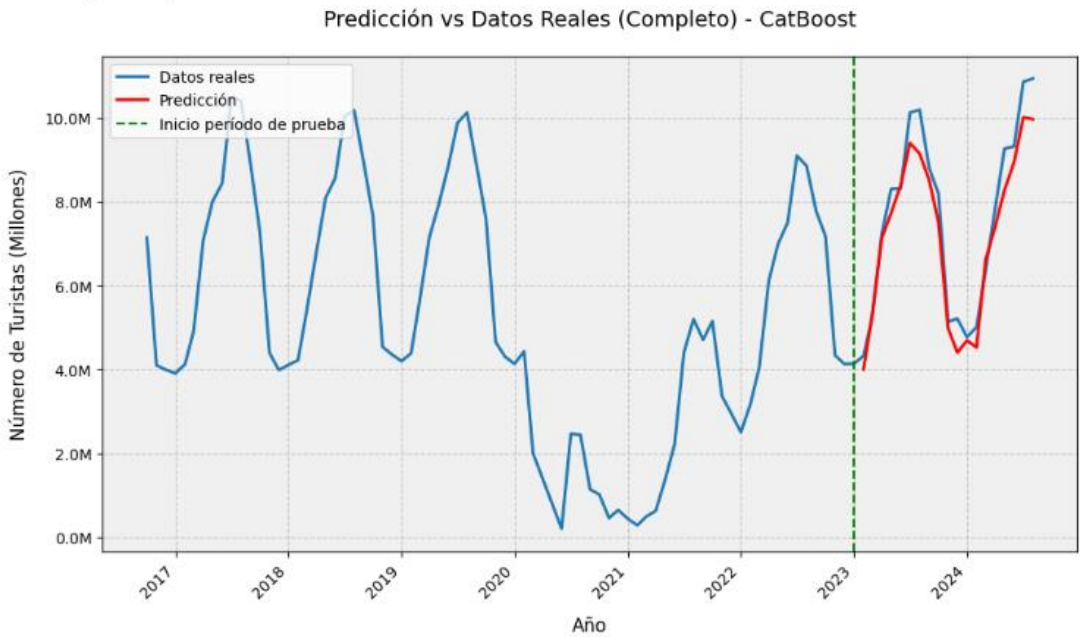


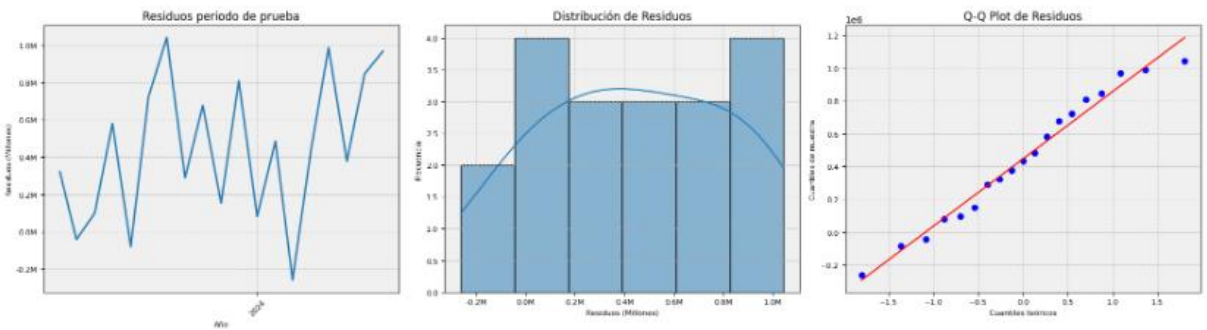




Tests estadísticos para los residuos:
Test de Shapiro-Wilk para normalidad: p-valor = 0.4442
Conclusión: Los residuos parecen seguir una distribución normal.

Estadístico de Durbin-Watson para autocorrelación: 0.6507
Conclusión: Hay evidencia de autocorrelación positiva en los residuos.
MAE: 662216.83
RMSE: 743552.59
R2: 0.88
MAPE: 8.89%

Generando gráficos para CatBoost



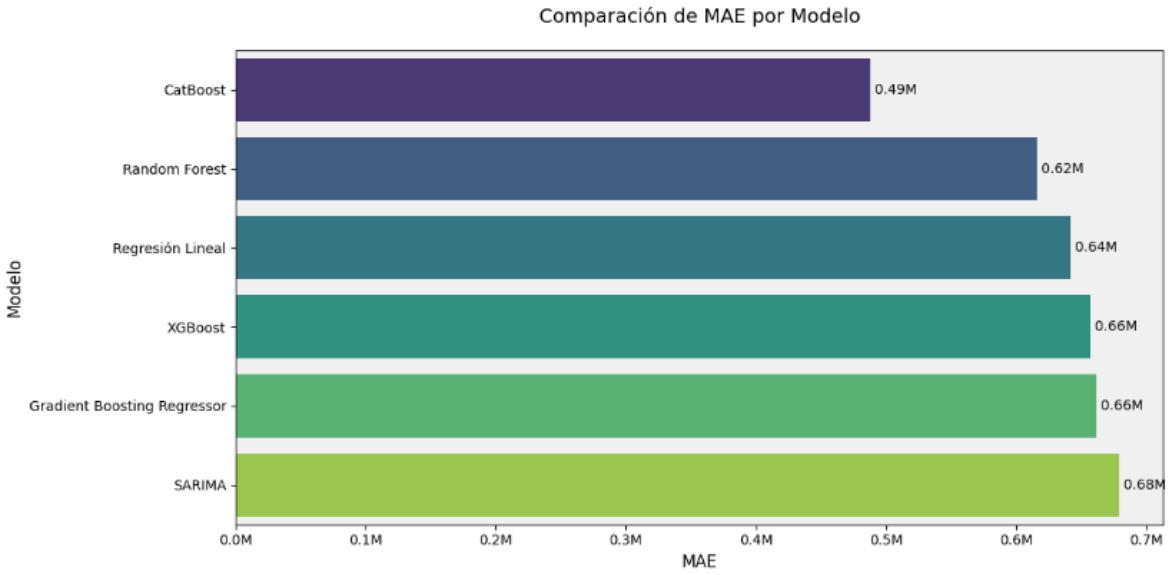




Tests estadísticos para los residuos:
Test de Shapiro-Wilk para normalidad: p-valor = 0.6536
Conclusión: Los residuos parecen seguir una distribución normal.

Estadístico de Durbin-Watson para autocorrelación: 0.8621
Conclusión: Hay evidencia de autocorrelación positiva en los residuos.
MAE: 488044.54
RMSE: 588465.57
R2: 0.92
MAPE: 6.19%

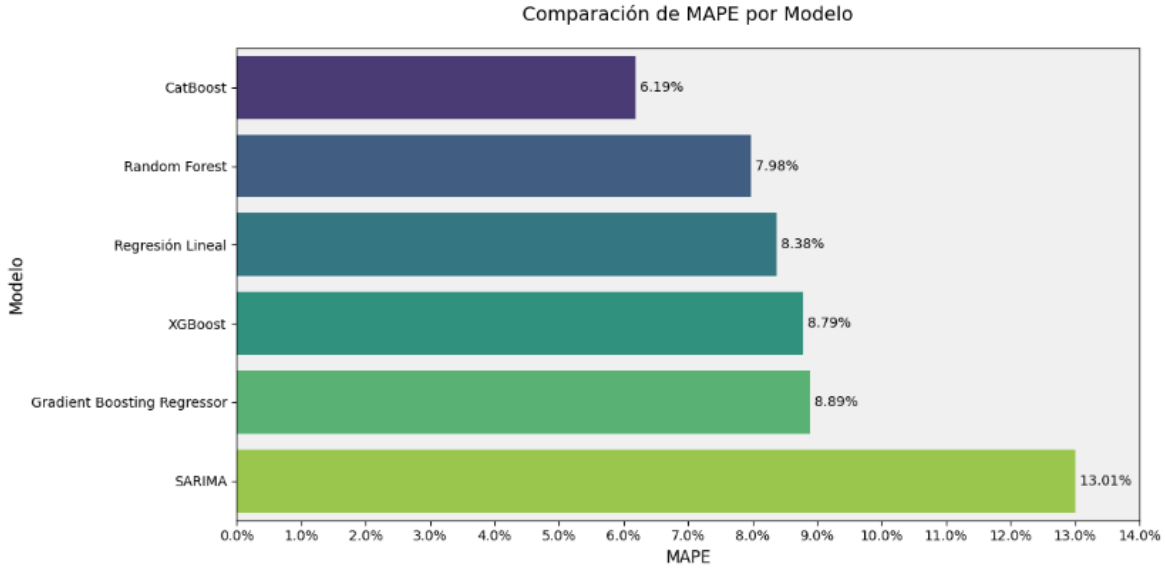
En general vemos que todos los algoritmos han tenido unos notables resultados mejorando mucho las métricas de nuestro primer modelo Sarima:

| Modelo | MAE | RMSE | MAPE | R2 |
|---|---|---|---|---|
| SARIMA | 679200 | 898980 | 13.01 | 0.84 |
| Regresión Lineal | 642507 | 747948 | 8.38 | 0.88 |
| Random Forest | 616551 | 736515 | 7.98 | 0.88 |
| XGBoost | 657533 | 748326 | 8.79 | 0.88 |
| Gradient Boosting Regressor | 662217 | 743553 | 8.89 | 0.88 |
| CatBoost | 488045 | 588466 | 6.19 | 0.92 |



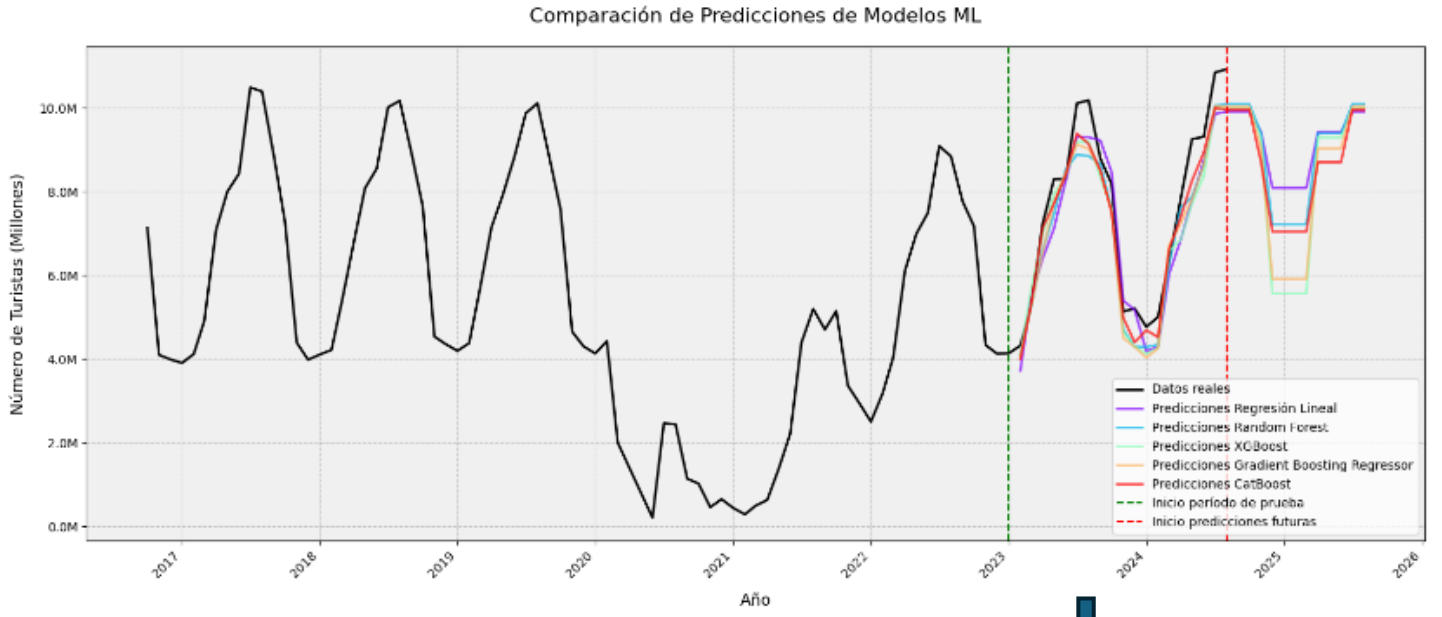

El MAE mide la magnitud promedio de los errores en un conjunto de predicciones, sin considerar su dirección. Se expresa en las mismas unidades que la variable objetivo, lo que lo hace fácilmente interpretable.

**Interpretación**: En promedio, las predicciones del Modelo SARIMA tiene un error promedio de 680.000 visitantes, frente a los 490.000 del algoritmo CatBoost.

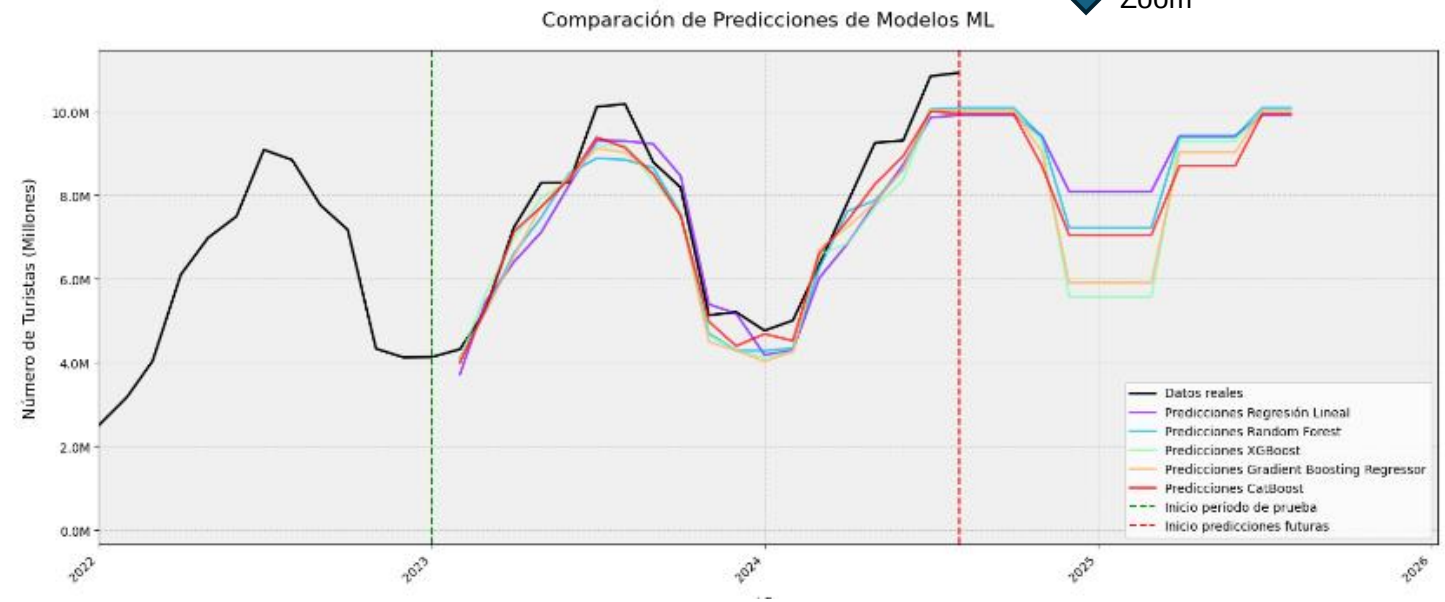



El MAPE mide el tamaño promedio del error en términos porcentuales. Es particularmente útil cuando queremos comparar el rendimiento entre diferentes conjuntos de datos o cuando la escala es importante.

En general vemos que todos los algoritmos han tenido unos notables resultados, estando muy cerca sus predicciones de las reales para el periodo de prueba.




Zoom




**Conclusión general**:

Los algoritmos de Machine Learning que hemos implementado parecen tener un buen rendimiento general, con un alto $R^2$ y un MAPE relativamente bajo.

Los residuos son normales, lo cual es positivo. Sin embargo, la presencia de autocorrelación en los residuos podría deberse a que los algoritmos tienen que enfrentar la caída abrupta de la pandemia, efecto que es de esperar se corrija a medida que la serie avance en el tiempo.

Las predicciones futuras parecen razonables y reflejan patrones estacionales esperados en el turismo.

## 2.5 Deep Learning con redes neuronales LSTM (Long Short-Term Memory)

“El efecto de la pandemia ha convertido la predicción de las series turísticas en un desafío complejo Para abordar este reto hemos optado por abordarlo de múltiples formas, desde las series temporales tradicionales a eficientes algoritmos de Machine Learning. Nos adentramos ahora, por último, a afrontar el reto utilizando redes neuronales LSTM (Long Short-Term Memory), una técnica avanzada de aprendizaje profundo especialmente adecuada para el análisis de series temporales.

**Las LSTM son particularmente efectivas en este contexto por varias razones**:

1. **Capacidad para capturar dependencias a largo plazo**: Las LSTM pueden 'recordar' información relevante durante largos períodos, lo cual es crucial para identificar patrones estacionales anuales y tendencias a largo plazo en el turismo.

2. **Manejo de la no linealidad**: Los flujos turísticos a menudo exhiben comportamientos no lineales, influenciados por factores como eventos globales, cambios económicos o tendencias de viaje. Las LSTM pueden capturar estas relaciones complejas y no lineales.

3. **Robustez frente a valores atípicos**: En el caso del turismo español, eventos como la crisis financiera de 2008 o la pandemia de COVID-19 han causado interrupciones significativas. Las LSTM pueden adaptarse a estos cambios bruscos y aprender patrones post-disrupción.

4. **Capacidad de integrar múltiples variables**: Además de los datos históricos de llegadas, las LSTM pueden incorporar fácilmente otras variables relevantes como indicadores económicos, datos climáticos o eventos especiales.

5. **Aprendizaje de patrones secuenciales**: El turismo tiene una fuerte componente secuencial, con patrones que se repiten y evolucionan a lo largo del tiempo. Las LSTM están diseñadas específicamente para aprender y predecir basándose en secuencias de datos.

Al utilizar LSTM para este análisis, esperamos obtener predicciones más precisas y robustas de las llegadas de turistas a España.

## Implementación

Partiendo del mismo dataframe con el que entrenamos los algoritmos de ML, la implementación de una LSTM tiene como proceso clave el de secuencialización Este proceso implica transformar nuestros datos de series temporales en un formato que las LSTM puedan procesar eficazmente.

Pasos:

1. Importaciones y preparación de datos: Esta parte importa las bibliotecas necesarias y prepara los datos, separándolos en conjuntos de entrenamiento y prueba, manteniendo el orden temporal (shuffle=False).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Asumimos que df_enhanced ya está cargado

# Preparar los datos
X = df_enhanced.drop('Valor', axis=1)
y = df_enhanced['Valor']

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

2. Normalización de datos: Aquí se normalizan los datos usando MinMaxScaler para llevarlos a un rango entre 0 y 1.

```
# Normalizar los datos
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))
```

3. Reshape para LSTM: Esta parte reformatea los datos para que sean compatibles con la entrada esperada por LSTM: [muestras, pasos de tiempo, características]. En este caso, cada muestra se trata como una secuencia de un solo paso de tiempo.

```
# Reshape para LSTM [muestras, pasos de tiempo, características]
X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))
```

4. Construcción y entrenamiento del modelo: Aquí se define y entrena el modelo LSTM. Se crea un modelo LSTM con dos capas LSTM seguidas de una capa Dense para la salida.

El modelo se entrena con los datos preparados, utilizando un conjunto de validación para monitorear el rendimiento durante el entrenamiento.

```
# Construir el modelo LSTM
model = Sequential([
    Input(shape=(1, X_train_reshaped.shape[2])),
    LSTM(50, activation='relu', return_sequences=True),
    LSTM(50, activation='relu'),
    Dense(1)
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

# Entrenar el modelo
history = model.fit(X_train_reshaped, y_train_scaled, epochs=100, batch_size=32,
                    validation_split=0.2, verbose=1)
```

5. Predicciones y desnormalización: Se realizan predicciones y se desnormalizan los resultados.

```
# Hacer predicciones
train_predict = model.predict(X_train_reshaped)
test_predict = model.predict(X_test_reshaped)

# Invertir la normalización
train_predict = scaler_y.inverse_transform(train_predict)
test_predict = scaler_y.inverse_transform(test_predict)
y_train = y_train.values.reshape(-1, 1)
y_test = y_test.values.reshape(-1, 1)
```

6. Cálculo de métricas: Se realizan predicciones tanto en los datos de entrenamiento como de prueba, y se evalúa el rendimiento del modelo utilizando varias métricas.

```
# Calcular métricas de error
train_mae = mean_absolute_error(y_train, train_predict)
train_rmse = np.sqrt(mean_squared_error(y_train, train_predict))
train_mape = np.mean(np.abs((y_train - train_predict) / y_train)) * 100
train_r2 = r2_score(y_train, train_predict)

test_mae = mean_absolute_error(y_test, test_predict)
test_rmse = np.sqrt(mean_squared_error(y_test, test_predict))
test_mape = np.mean(np.abs((y_test - test_predict) / y_test)) * 100
test_r2 = r2_score(y_test, test_predict)

print("Métricas de entrenamiento:")
print(f"MAE: {train_mae:.2f}")
print(f"RMSE: {train_rmse:.2f}")
print(f"MAPE: {train_mape:.2f}%")
print(f"R2: {train_r2:.4f}")

print("\nMétricas de prueba:")
print(f"MAE: {test_mae:.2f}")
print(f"RMSE: {test_rmse:.2f}")
print(f"MAPE: {test_mape:.2f}%")
print(f"R2: {test_r2:.4f}")

añadir_resultados('LSTM', test_mae, test_rmse, test_mape, test_r2)
```

**Explicación de la secuencialización de los datos en una LSTM**:

Este proceso implica transformar nuestros datos de series temporales en un formato que las LSTM puedan procesar eficazmente.

1. Definición de la ventana temporal: Primero, determinamos el tamaño de la ventana de tiempo que utilizaremos para hacer predicciones. Por ejemplo, podríamos decidir usar los últimos 12 meses para predecir el siguiente mes.

2. Creación de secuencias: Convertimos nuestros datos en secuencias de entrada-salida. Cada secuencia de entrada consiste en 'n' pasos de tiempo (por ejemplo, 12 meses), y la salida correspondiente es el valor del siguiente paso de tiempo (el mes 13).

3. Deslizamiento de la ventana: Aplicamos un enfoque de ventana deslizante, donde avanzamos un paso a la vez a través de nuestros datos, creando una nueva secuencia en cada paso.

4. Normalización: Normalizamos los datos para asegurar que todas las características estén en una escala similar, lo que ayuda al entrenamiento de la red.

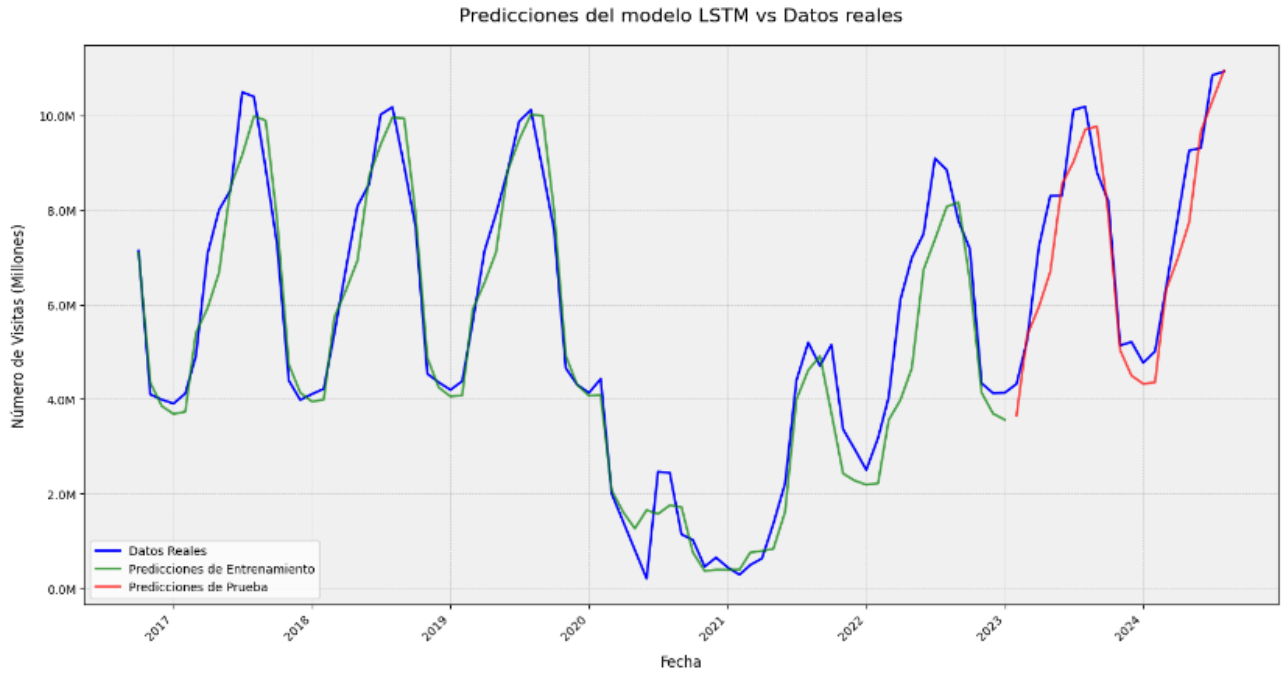
5. Reshape de los datos: Finalmente, reformateamos nuestros datos en la forma tridimensional que requieren las LSTM: [muestras, pasos de tiempo, características].

Este proceso de secuencialización nos permite aprovechar la capacidad de las LSTM para aprender de patrones temporales, capturando así las tendencias estacionales y a largo plazo.

**Resultados LSTM:**

```
Epoch 100/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.0033 - val_loss: 0.0176
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 124ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
Métricas de entrenamiento:
MAE: 572300.80
RMSE: 807485.69
MAPE: 23.76%
R2: 0.9228

Métricas de prueba:
MAE: 579343.11
RMSE: 719386.09
MAPE: 7.98%
R2: 0.8878
```

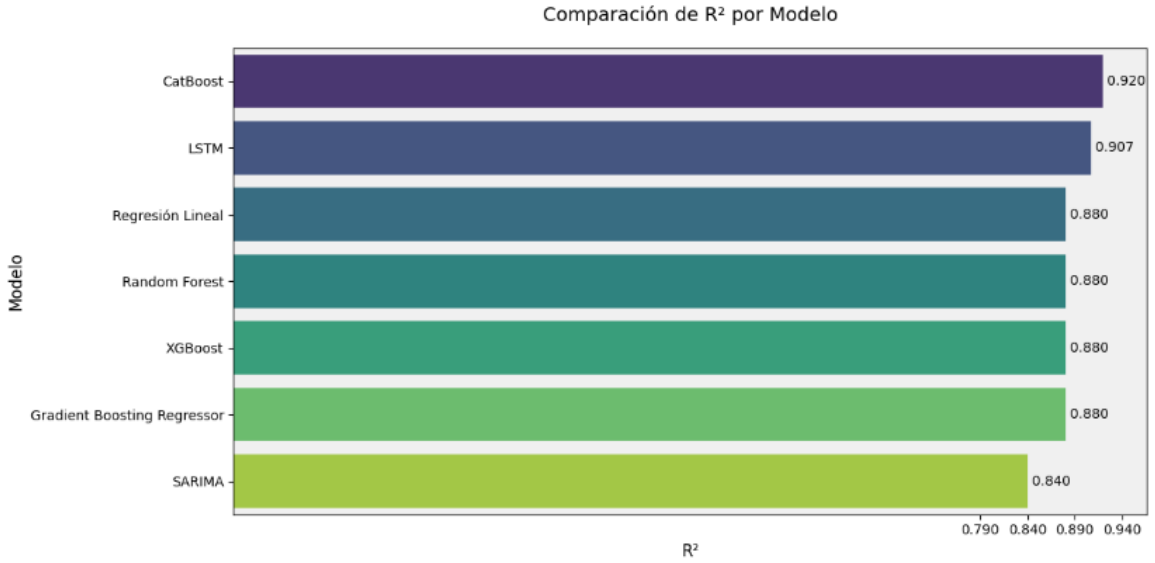
Se obtienen muy buenos resultados, colocándose muy cerca en cuanto a MAPE de nuestro mejor modelo (CatBoost con 6,19%).



# 03 Conclusiones y recomendaciones finales

## Conclusiones

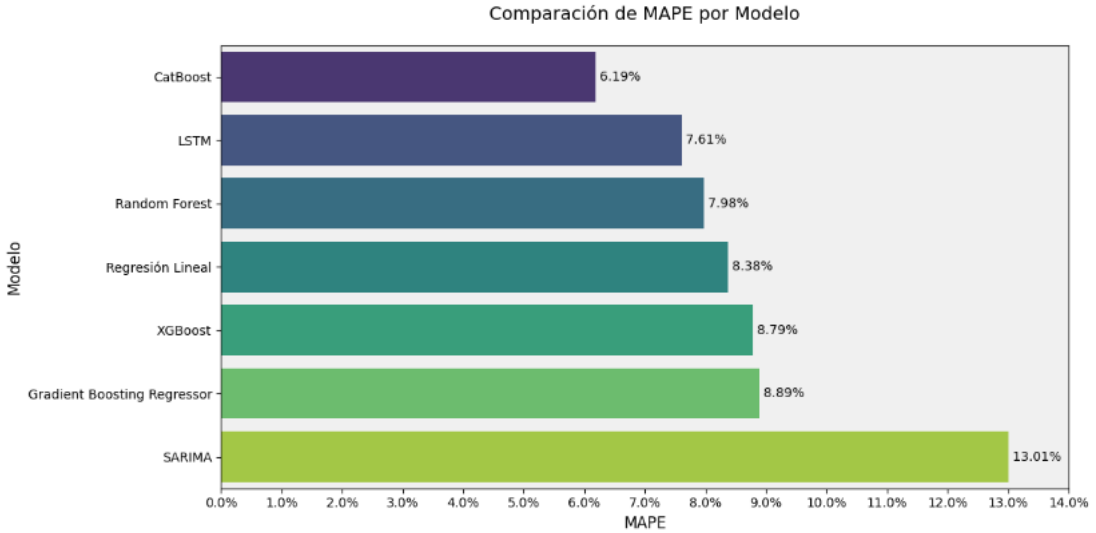
**1. Comparación de Modelos**: Todos los modelos aplicados muestran un rendimiento relativamente bueno, con $R^2$ superiores a 0.84, lo que indica que capturan una parte significativa de la variabilidad en los datos de entrada de turistas.




**2. Rendimiento de Modelos Tradicionales vs. Avanzados**: Los modelos de aprendizaje automático (Random Forest, XGBoost, Gradient Boosting, CatBoost) generalmente superan al modelo SARIMA tradicional y a la Regresión Lineal, lo que indica la presencia de relaciones no lineales en los datos que estos modelos más avanzados pueden capturar mejor.

**3. LSTM**: El modelo LSTM muestra un rendimiento intermedio (MAE: 569,980, MAPE: 7.61%, $R^2$: 0.907). Aunque no es el mejor, su capacidad para manejar dependencias temporales lo hace valioso, especialmente para capturar patrones estacionales y tendencias a largo plazo.

**4. Mejor Modelo**: El modelo CatBoost destaca como el más efectivo, con el MAE más bajo (488,045), el RMSE más bajo (588,466), el MAPE más bajo (6.19%), y el $R^2$ más alto (0.926). Esto sugiere que CatBoost es particularmente adecuado para capturar los patrones complejos en los datos turísticos de España.




**5. Precisión de las Predicciones:** Con MAPEs que van del 6.19% al 8.89%, todos los modelos ofrecen predicciones razonablemente precisas. Esto sugiere que, en promedio, las predicciones se desvían entre un 6% y un 9% de los valores reales, lo cual es aceptable para muchas aplicaciones en la planificación turística.

**7. Robustez de los Modelos:** La consistencia en el rendimiento entre los diferentes modelos (todos con $R^2 > 0.84$) sugiere que los patrones en los datos de entrada de turistas son relativamente estables y predecibles.

## Recomendaciones finales

**Implicaciones para la Industria Turística**: La alta precisión de estos modelos, especialmente CatBoost, puede ser muy valiosa para la planificación en el sector turístico español. Permite anticipar con mayor exactitud los flujos de turistas, lo que puede ayudar en la gestión de recursos, planificación de infraestructuras y estrategias de marketing.

**Consideraciones Futuras:** Aunque los modelos actuales muestran un buen rendimiento, eventos imprevistos como la pandemia de COVID-19 pueden afectar significativamente los patrones turísticos. Sería recomendable incorporar variables externas (como indicadores económicos globales o eventos internacionales) para mejorar aún más la robustez de las predicciones

**En conclusión, este estudio demuestra la eficacia de los modelos de aprendizaje automático avanzados en la predicción de la entrada de turistas en España. La alta precisión de estos modelos puede proporcionar una ventaja significativa en la toma de decisiones y la planificación estratégica en el sector turístico español.**


**Enrique Aranaz Tudela**

*|Data Driven |Consultor BI | Microsoft Certified Power BI Data Analyst |Data Scientist | Machine Learning Python*

LinkedIn www.linkedin.com/in/enrique-aranaz-tudela

aranaz.enrique@gmail.com
aranaz.enrique@biparatupyme.com


Thank you