UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS DE LA DOCUMENTACIÓN

Máster Big data & Data Science

Trabajo Fin de MÁSTER

Analysis and visualization of energy matrices and balances

Ámbito geográfico, Argentina

Alumno: Moisés Rodríguez Pavón

Madrid, 19 Septiembre 2023

Resumen ejecutivo: Proyecto de análisis, previsión y visualización de matrices y balances energéticos.

Introduccion:

El proyecto, tiene como objetivo principal predecir y comprender el comportamiento del consumo energético en Argentina hasta el año 2033. Para lograr dicho objetivo, se ha llevado a cabo una investigación y análisis de los datos avanzados, con el que poder proporcionar una visión clara y preciada de cómo se espera que evolucione el consumo energético en el fututo.

Contexto:

El país seleccionado en cuestión para el análisis, se enfrenta a una creciente demanda de energía para los próximos años debido al crecimiento económico y a la necesidad de cubrir las necesidades energéticas de sus ciudadanos de la manera mas sostenible y ambiental posible, lo cual veremos si es posible mas adelante. El proyecto se centra en dar sentido a los patrones de consumo energético y las fuentes de energía utilizadas en el país, lo que permitirá tomas decisiones más informadas para la planificación energética y la toma de decisiones políticas.

Metodología:

Recopilación de base de datos: Los datos históricos han sido recopilados de la web de 'sielac.olade' de la cual hemos descargado todas las matrices energéticas desde 1970 a 2021. Estas matrices cuentan con información relevante de los consumos totales de cada fuente energética, así como de la cantidad de oferta de estas y la precedencia de la generación de dicha energía.

Análisis descriptivo: Realizamos un análisis descriptivo previo para comprender la evolución del consumo energético a lo largo del tiempo que nos ayudara a identificar posibles tendencias y patrones. También investigamos sobre determinadas fuentes de producción de energía que nos guiase sobre la evolución de estas, por ejemplo:

Hidroenergía: Vemos que la fuente de producción está suficientemente explotada sin posibilidad de poder generar mas energía en el futuro por lo que vemos que la producción de energía se mantiene.

Gas natural: Se prevé un crecimiento en producción de gas natural causado por el incremento de la explotación de la llamada 'Vaca Muerta' que es una gran reserva de gas natural y petróleo que se está empezando a explotar poco a poco y de la que se prevé que se abastezca una gran parte del consumo energético final de los consumidores.

Renovables: No hay una gran apuesta por las renovables en Argentina, las inversiones serán mínimas y solo se prevé un crecimiento aproximado del 16%.

Nuclear: Toda la energía nuclear que abastece a los argentinos es energía importada de otros países, la construcción de nuevas centrales en el país es poco probable por lo que se pronostica que la oferta siga lineal con declive para los últimos años de nuestra predicción por el cierre de algunas de estas centrales nucleares.

Modelos de predicción: Para las predicciones y dado la base de datos que tenemos, desarrollamos un modelo de predicción usando técnicas de aprendizaje automático y análisis de series temporales para desarrollar la finalidad del proyecto. Estos modelos desarrollados se basan en los datos históricos obtenidos y teniendo en cuenta otras variables explicativas como son en este caso, el PIB y desempleo del país en cuestión.

Escenarios futuros: Creamos varios escenarios posibles para el consumo energético en 2033, teniendo en cuenta diferentes niveles de eficiencia energética, en este caso proyectamos el consumo energético en primer lugar con la gasificación, dando una mayor importancia al gas natural y dado la situación de Argentina. En segundo lugar, proyectamos dando un mayor peso a las renovables, aunque vemos que no tiene proyección, por último, mostramos los resultados tendenciales según nuestras predicciones.

Resultados y conclusiones:

- Se espera que la demanda energética en Argentina siga aumentando en los próximos 10 años teniendo en concreto según las predicciones un consumo total de 56.099,12 10^3tep en 2021 a 62.340,005 10^3tep en 2033.
- Para ayudar a cubrir estos nuevos consumos, es primordial el aumento de las energías renovables, que según nuestras predicciones aumentan su oferta en mas de un 21% y también ayudado de, como decíamos anteriormente, la gasificación argentina aumentando la oferta de gas natural en un 19% desde 2021
- Como podemos apreciar, la transición hacia fuentes de energía más limpias, como las renovables es esencial para cubrir los objetivos de consumo así como de sostenibilidad ambiental. Aunque en este caso esto no es muy llamativo ya que sigue habiendo una dependencia bastante grande hacia los combustibles fósiles.

Recomendaciones: En cuanto a los resultados obtenidos, se sugieren las siguientes recomendaciones para la toma de decisiones.

- Fomentar la inversión en energias renovables para diversificar la matriz energética, aunque tengamos un gran deposito a explotar de combustibles fósiles, debemos pensar en el futuro y con ello la inversión en de las energías renovables.

Conclusiones finales:

El proyecto de análisis que hemos elaborado proporciona una posible visión del comportamiento esperado del consumo energético a 2030 siendo algunas de las conclusiones clave las que elaboro a continuación:

- Crecimiento continuo de la demanda energética, impulsadas por el crecimiento económico y la necesidad de satisfacer las necesidades energéticas de la población.
- Gas natural como la fuente de energía mas importante del país que verá un aumento significativo de la producción que contribuirá significativamente a cubrir la creciente demanda energética.

- Las energías renovables verán limitado su crecimiento, a pesar de la importancia en la sostenibilidad ambiental, se observa una inversión limitada en el sector en Argentina. - La energía nuclear es importada en toda su dimensión y la falta de previsión de construcción de nuevas centrales tanto en el país como fuera de él, sugiere un desafío para el suministro de energía nuclear en el futuro, con una tendencia a la disminución en los últimos años.
- A pesar del crecimiento en la producción de gas natural, argentina debe de considerar una transición mas sólida hacia fuentes de energía más limpias y sostenibles para dejar atrás su dependencia de los combustibles fósiles.

Este resumen ejecutivo es una representación general del proyecto y sus resultados. Para obtener información detallada, se recomienda revisar el informe completo del proyecto.

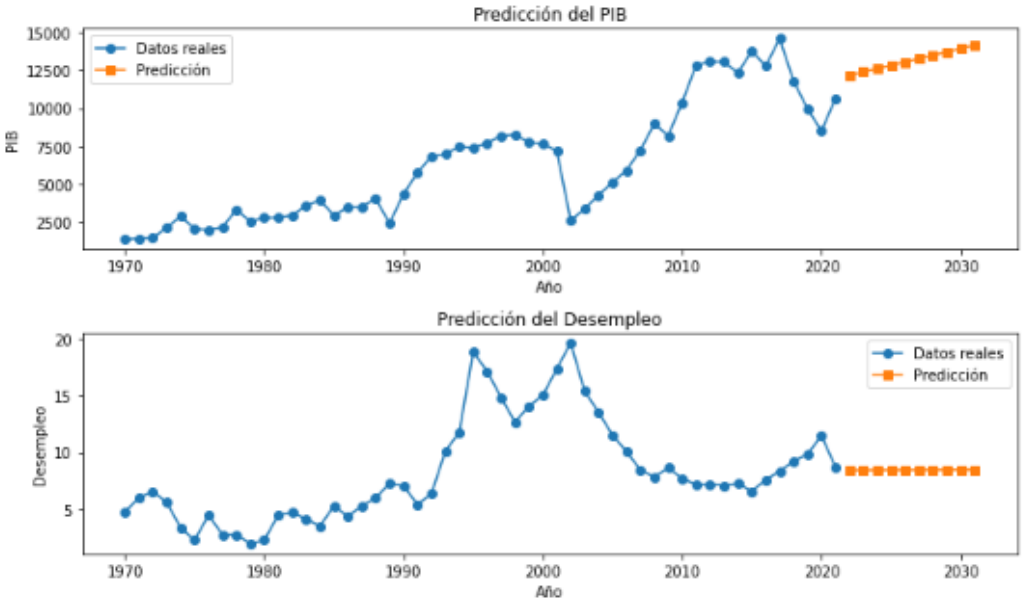
A continuación, detallaré el código que marca el inicio de mi proyecto. Partimos de una extensa base de datos que abarca un período de 51 años, desde 1970 hasta 2021, que contiene información relevante sobre el consumo energético en Argentina. Para llevar a cabo nuestras predicciones de consumo energético para los próximos 10 años, empleamos un enfoque basado en modelos de series temporales, complementados con variables explicativas.

En el siguiente código, segmentamos la variable 'Año' y 'PIB y Desempleo'. A continuación, creamos modelos de regresión lineal donde uso validación cruzada para evaluar el rendimiento de R2. Realizamos una búsqueda para encontrar los mejores hiperparámetros y los imprimimos. Entrenamos los modelos y se realizan predicciones para 10 años. Por último, realizamos una visualización de las predicciones. Con esto simplemente queremos ver como se van a comportar estas variables explicativas.

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, GridSearchCV

# Leer los datos desde el archivo Excel
df = pd.read_excel("variables explicativas.xlsx")

# Obtener las variables independientes (Año) y dependientes (PIB y Desempleo)
X = df['Año'].values.reshape(-1, 1)
y_pib = df['PIB'].values
y_desempleo = df['Desempleo'].values

# Crear el modelo de regresión lineal para el PIB y el Desempleo
modelo_pib = LinearRegression()
modelo_desempleo = LinearRegression()

# Entrenamiento con validación cruzada para el PIB
scores_pib = cross_val_score(modelo_pib, X, y_pib, cv=5, scoring='r2')
print("R cuadrado promedio (PIB):", scores_pib.mean())

# Optimizar hiper-parámetros para el PIB
parametros_pib = {'fit_intercept': [True, False]}
grid_pib = GridSearchCV(modelo_pib, parametros_pib, cv=5, scoring='r2')
grid_pib.fit(X, y_pib)
print("Mejores parámetros (PIB):", grid_pib.best_params_)

# Entrenar modelo final para el PIB
modelo_final_pib = grid_pib.best_estimator_
modelo_final_pib.fit(X, y_pib)

# Entrenamiento con validación cruzada para el Desempleo
scores_desempleo = cross_val_score(modelo_desempleo, X, y_desempleo, cv=5, scoring='r2')
print("R cuadrado promedio (Desempleo):", scores_desempleo.mean())

# Optimizar hiper-parámetros para el Desempleo
parametros_desempleo = {'fit_intercept': [True, False]}
grid_desempleo = GridSearchCV(modelo_desempleo, parametros_desempleo, cv=5, scoring='r2')
grid_desempleo.fit(X, y_desempleo)
print("Mejores parámetros (Desempleo):", grid_desempleo.best_params_)

# Entrenar modelo final para el Desempleo
modelo_final_desempleo = grid_desempleo.best_estimator_
modelo_final_desempleo.fit(X, y_desempleo)
```

```
# Graficar los resultados
plt.figure(figsize=(10, 6))

# Gráfico del PIB
plt.subplot(2, 1, 1)
plt.plot(df['Año'], df['PIB'], 'o-', label='Datos reales')
plt.plot(años_futuros, prediccion_pib, 's-', label='Predicción')
plt.xlabel('Año')
plt.ylabel('PIB')
plt.title('Predicción del PIB')
plt.legend()

# Gráfico del Desempleo
plt.subplot(2, 1, 2)
plt.plot(df['Año'], df['Desempleo'], 'o-', label='Datos reales')
plt.plot(años_futuros, prediccion_desempleo, 's-', label='Predicción')
plt.xlabel('Año')
plt.ylabel('Desempleo')
plt.title('Predicción del Desempleo')
plt.legend()

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

```
R cuadrado promedio (PIB): -2.816419725897453
Mejores parámetros (PIB): {'fit_intercept': True}
R cuadrado promedio (Desempleo): -7.068373853330708
Mejores parámetros (Desempleo): {'fit_intercept': False}
```



En la predicción anterior y expresada gráficamente, podemos ver como el PIB aumentaria en los proximos años tras un declive bastante pronunciado. De la misma manera podemos ver como disminuye el nivel de Desempleo. Esto nos hace pensar que el nivel de consumo puede ir aumentando gradualmente con los años tal y como nos hacen pensar estos datos.

En el siguiente codigo, combinamos el df con df_vars usando merged y usando la columna año como unión entre ambas. Convertimos la columna a 'datetime' y la hacemos índice. Expecificamos los parametros del modelo ARIMA, creamos un nuevo dataframe para almacenar las predicciones de cada variable, ordenamos los indices de los df oara asegurarnos que este bien alineados y por ultimo visualizamos las predicciones con gráficos.

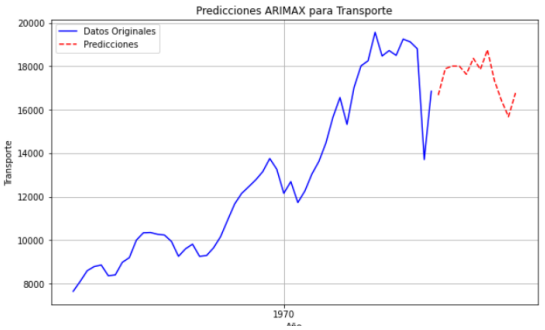
Por ultimo y una vez finalizado esto realizamos un codigo MAPE y expresamos los resultados.

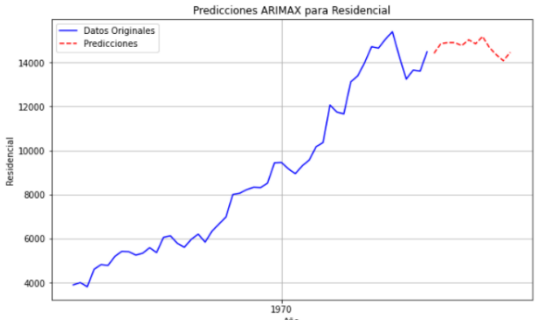
In [27]:
```python
# Leer datos desde el archivo Excel
df = pd.read_excel('Matriz.xlsx')
df_vars = pd.read_excel("variables explicativas.xlsx")

df_merged = pd.merge(df, df_vars, on='Año')

# Convertir la columna 'Año' al tipo de dato datetime
if not isinstance(df_merged['Año'], pd.DatetimeIndex):
    df_merged['Año'] = pd.to_datetime(df_merged['Año'])
```

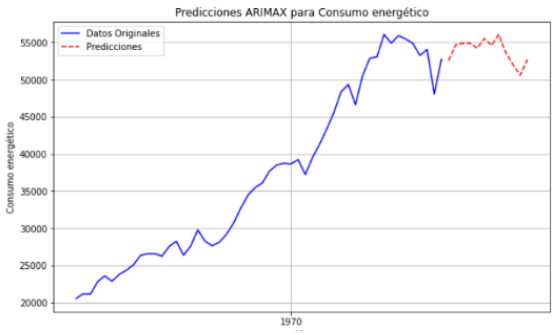
```python
# Crear DataFrame para almacenar las predicciones
predictions = pd.DataFrame()

# Realizar la búsqueda de hiperparámetros y predicción para cada
forecast_steps = 12
for col, orders in arima_params.items():
    future_exog = df_vars[['PIB', 'Desempleo']].tail(forecast_steps).values

    best_mse = float('inf')
    best_order = None
    for order in orders:
        mse = np.mean((fit_arimax_and_forecast(df_merged[col], df_merged[['PIB', 'Desempleo']], order, forecast_steps, future_exog) - df_merged[col][-forecast_steps:]) ** 2)
        if mse < best_mse:
            best_mse = mse
            best_order = order

    if best_order is None:
        best_order = (1, 1, 1)  # Use a default order if no valid order was found

    forecast = fit_arimax_and_forecast(df_merged[col], df_merged[['PIB', 'Desempleo']], order=best_order, forecast_steps=forecast_steps, future_exog=future_exog)
    predictions[col] = forecast
```

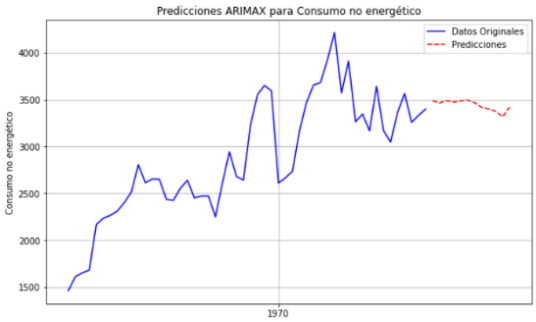
```python
# Establecer 'Año' como el índice del DataFrame
df_merged.set_index('Año', inplace=True)

# Función para ajustar el modelo ARIMAX y predicciones
def fit_arimax_and_forecast(endog, exog, order, forecast_steps, future_exog):
    model = ARIMA(endog=endog, exog=exog, order=order)
    model_fit = model.fit()
    predictions = model_fit.forecast(steps=forecast_steps, exog=future_exog)
    return predictions

# Especificar los parámetros del modelo ARIMA
arima_params = {
    'Transporte': [(1, 1, 1), (1, 1, 2), (2, 1, 1)],
    'Industrial': [(1, 1, 1), (1, 1, 2), (2, 1, 1)],
    'Residencial': [(1, 1, 1), (1, 1, 2), (2, 1, 1)],
    'Comercial': [(1, 1, 1), (1, 1, 2), (2, 1, 1)],
    'Agro, Pesca y Mineria': [(1, 1, 1), (1, 1, 2), (2, 1, 1)],
    'Construcción': [(1, 1, 0), (1, 1, 1), (2, 1, 1)],
    'Consumo energético': [(1, 1, 1), (1, 1, 2), (2, 1, 1)],
    'Consumo no energético': [(1, 1, 1), (1, 1, 2), (2, 1, 1)],
    'Consumo total': [(1, 1, 1), (1, 1, 2), (2, 1, 1)]
}
```

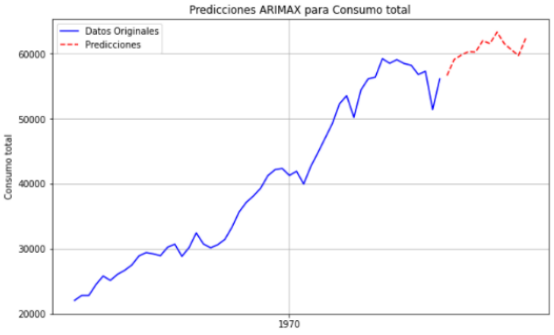
```python
# Ordenar los índices de los DataFrames para asegurarnos de que estén alineados
df_merged.sort_index(inplace=True)
predictions.sort_index(inplace=True)

# Visualizar las predicciones
for col in predictions.columns:
    plt.figure(figsize=(10, 6))
    plt.plot(df_merged.index, df_merged[col], label='Datos Originales', color='blue')
    plt.plot(predictions.index, predictions[col], label='Predicciones', color='red', linestyle='--')
    plt.title(f'Predicciones ARIMAX para {col}')
    plt.xlabel('Año')
    plt.ylabel(col)
    plt.grid(True)
    plt.legend()
    plt.show()

# Imprimir las predicciones para los próximos 10 años
print("Predicciones para los próximos 10 años:")
print(predictions)
```











Predicciones para los próximos 10 años:

	Transporte	Industrial	Residencial		Comercial	Agro, Pesca y Mineria		Construcción	Consumo energético
1970-01-01 00:00:00.000002022	16675.184633	13022.221641	14416.059220	2022	4482.581413	4055.172830	2022	0.0	52526.750556
1970-01-01 00:00:00.000002023	17893.080666	13226.704161	14857.871994	2023	4643.962319	4329.789886	2023	0.0	54658.710132
1970-01-01 00:00:00.000002024	18006.660001	13243.104195	14899.981681	2024	4671.529708	4382.262801	2024	0.0	54850.811597
1970-01-01 00:00:00.000002025	18006.316578	13246.514590	14899.410495	2025	4681.444799	4409.663346	2025	0.0	54860.801166
1970-01-01 00:00:00.000002026	17633.956813	13186.529124	14765.447671	2026	4645.711065	4363.681431	2026	0.0	54216.641672
1970-01-01 00:00:00.000002027	18363.814442	13313.620533	15026.403010	2027	4732.176840	4535.019357	2027	0.0	55506.934272
1970-01-01 00:00:00.000002028	17856.749555	13210.270199	14847.789638	2028	4674.005254	4468.994751	2028	0.0	54567.050783
1970-01-01 00:00:00.000002029	18754.550289	13322.261878	15176.674389	2029	4769.435540	4687.384268	2029	0.0	56026.151643
1970-01-01 00:00:00.000002030	17344.884960	13088.743743	14670.544588	2030	4612.165911	4435.974275	2030	0.0	53568.599310
1970-01-01 00:00:00.000002031	16428.082962	12935.344764	14341.645201	2031	4509.580145	4282.872530	2031	0.0	51965.876940
1970-01-01 00:00:00.000002032	15681.585294	12779.028997	14079.430891	2032	4420.328057	4173.531263	2032	0.0	50570.154870
1970-01-01 00:00:00.000002033	16774.032404	13017.623648	14461.412850	2033	4553.340713	4400.293449	2033	0.0	52641.107690

	Consumo no energético	Consumo total
2022	3488.415583	56626.063381
2023	3464.876420	59084.340045
2024	3490.182253	59771.536710
2025	3472.973368	60312.373578
2026	3486.897737	60243.980177
2027	3495.015491	61975.665434
2028	3467.630209	61562.122748
2029	3417.726755	63301.993153
2030	3400.153025	61554.863468
2031	3375.892590	60591.102889
2032	3317.014295	59711.291962
2033	3421.410105	62340.005184

```python
# Calcular las métricas MAPE para cada columna
mape_scores = {}
for col in predictions.columns:
    actual_values = df_merged[col].values[-forecast_steps:]
    predicted_values = predictions[col].values

    # Calcular MAPE
    non_zero = actual_values != 0
    mape = np.mean(np.abs((actual_values[non_zero] - predicted_values[non_zero]) / np.maximum(np.abs(actual_values[non_zero]), 1e-10))) * 100
    mape_scores[col] = mape

# Imprimir las métricas MAPE para cada columna

print("\nMétricas MAPE:")
print(mape_scores)
```

Métricas MAPE:

{'Transporte': 5.1175127862189065, 'Industrial': 3.0934126970926, 'Residencial': 4.916878619129993, 'Comercial': 3.5897781377912135, 'Agro, Pesca y Mineria': 12.557061615490571, 'Construcción': nan, 'Consumo energético': 2.363077207704004, 'Consumo no energético': 5.856150423293341, 'Consumo total': 6.717995637964134}

INTERPRETACIÓN DE LOS DATOS:

Transporte: MAPE: 5,12% El modelo tiene una buena precisión ya que el error promedio es relativamente bajo.

Industrial: MAPE: 3.01% en relación con los datos reales. Esto indica que el modelo tiene una alta precisión para esta variable, ya que el error promedio es muy bajo.

Residencial: MAPE: 4.91%: Esto indica que el modelo tiene una buena precisión para esta variable, aunque ligeramente mayor que en el caso de 'Industrial'.

Comercial: MAPE: 3.58%: Esto sugiere que el modelo tiene una alta precisión, al igual que en el caso de 'Industrial'.

Agro, Pesca y Mineria: MAPE: 12.55%: El modelo tiene una menor precisión para en comparación con otras variables.

Consumo energético: MAPE: 2.36%: ALTA precisión, al igual que en los casos de 'Industrial' y 'Comercial'.

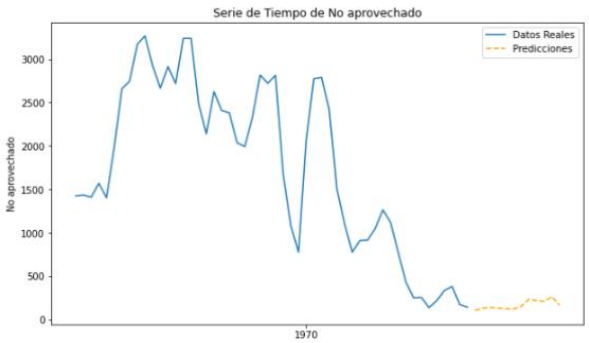
Consumo no energético: MAPE: 5.85%. Esto sugiere que el modelo tiene una BUENA precisión.

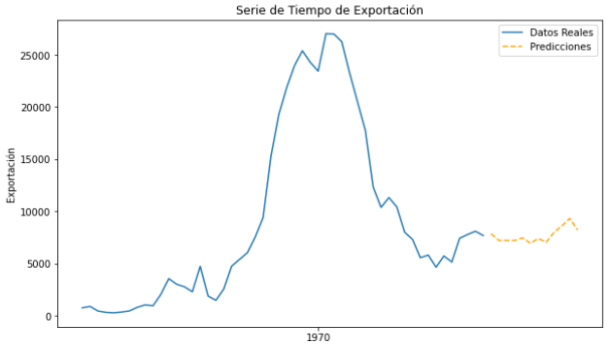
Consumo total: MAPE: 6.71%: Esto indica que el modelo tiene una BUENA precisión.

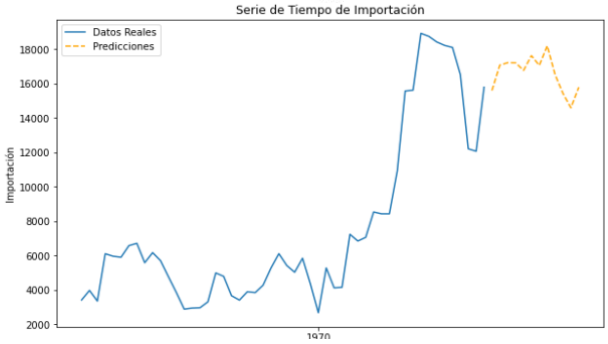
```python
# Leer datos desde el archivo Excel
do = pd.read_excel('Oferta.xlsx')
df_vars = pd.read_excel("variables explicativas.xlsx")

df_merged = pd.merge(do, df_vars, on='Año')

# Convertir la columna 'Año' al tipo de dato datetime
if not isinstance(df_merged['Año'], pd.DatetimeIndex):
    df_merged['Año'] = pd.to_datetime(df_merged['Año'])

# Establecer 'Año' como el índice del DataFrame
df_merged.set_index('Año', inplace=True)

# Realizar búsqueda automática de hiperparámetros y predicciónes
forecast_steps = 12
mape_dict = {}  # Almacenar MAPEs calculados
predictions = pd.DataFrame()  # DataFrame para almacenar predicciones

for col in df_merged.columns:
    if col not in ['Año', 'PIB', 'Desempleo']:
        future_exog = df_vars[['PIB', 'Desempleo']].tail(forecast_steps).values

        # Buscar los mejores parámetros con auto_arima
        stepwise_fit = auto_arima(df_merged[col], exogenous=df_merged[['PIB', 'Desempleo']], seasonal=True, m=12, suppress_warnings=True, stepwise=True)
        order = stepwise_fit.get_params()['order']
        seasonal_order = stepwise_fit.get_params()['seasonal_order']

        # Ajustar el modelo SARIMAX y realizar predicciones
        model = SARIMAX(endog=df_merged[col], exog=df_merged[['PIB', 'Desempleo']], order=order, seasonal_order=seasonal_order)
        model_fit = model.fit(disp=False)
        forecast = model_fit.get_forecast(steps=forecast_steps, exog=future_exog).predicted_mean

        predictions[col] = forecast

        # Calcular MAPE
        actual_values = df_merged[col][-forecast_steps:].values
        mape = np.mean(np.abs((actual_values - forecast) / actual_values)) * 100
        mape_dict[col] = mape
```

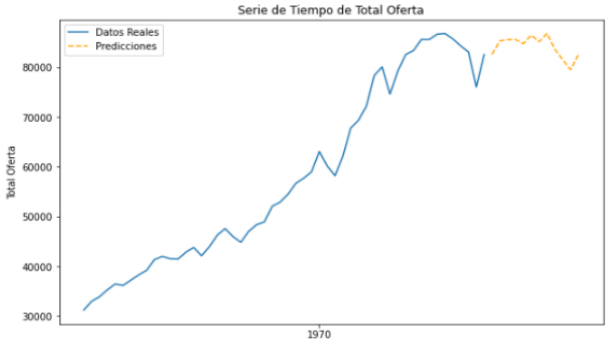
```python
# Ordenar los índices
df_merged.sort_index(inplace=True)
predictions.sort_index(inplace=True)

# Mostrar tablas de predicciones
print("\nTablas de Predicciones:")
print(predictions)

# Mostrar gráficos de series temporales
for col in predictions.columns:
    plt.figure(figsize=(10, 6))
    plt.plot(df_merged.index, df_merged[col], label='Datos Reales')
    plt.plot(predictions.index, predictions[col], label='Predicciones', linestyle='dashed', color='orange')
    plt.title(f'Serie de Tiempo de {col}')
    plt.xlabel('Año')
    plt.ylabel(col)
    plt.legend()
    plt.show()

# MAPEs
print("\nMAPEs:")
for col, mape in mape_dict.items():
    print(f"{col}: {mape:.2f}%")

predictions.to_excel('Resultados_OfertaTotal.xlsx', index=True)
```

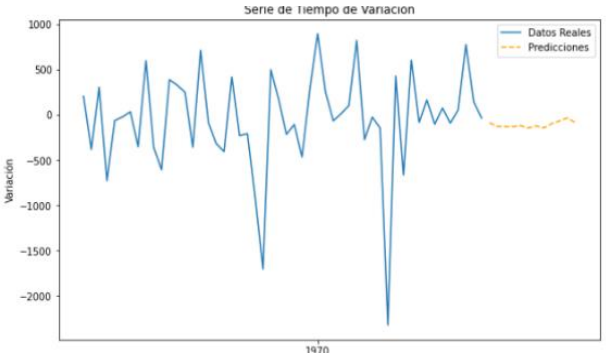
```
Tablas de Predicciones:
                                Producción   Importación  Exportación  \    Variación  No aprovechado  Total Oferta
1970-01-01 00:00:00.000002022  74745.601900  15591.788229  7848.916439     -89.613158      103.298921  82598.831784
1970-01-01 00:00:00.000002023  75361.819057  17071.688093  7200.560729    -128.262201      130.065078  85332.031856
1970-01-01 00:00:00.000002024  75441.422839  17215.875118  7200.012401    -131.266679      137.450698  85565.196308
1970-01-01 00:00:00.000002025  75419.049554  17209.597306  7202.045883    -132.118072      130.963424  85597.801071
1970-01-01 00:00:00.000002026  75252.729942  16762.030841  7449.970744    -120.348769      123.374887  84767.687342
1970-01-01 00:00:00.000002027  75512.315209  17621.598956  6926.743856    -145.805196      120.039242  86486.148083
1970-01-01 00:00:00.000002028  75438.325544  17052.876678  7394.079527    -124.338530      151.271383  85147.873234
1970-01-01 00:00:00.000002029  76070.699083  18193.970538  7036.547094    -144.515946      232.319610  86836.421641
1970-01-01 00:00:00.000002030  75485.067628  16511.253688  7977.861156     -98.349516      215.824961  83631.976282
1970-01-01 00:00:00.000002031  75114.967762  15419.750301  8599.991335     -67.941042      208.028106  81533.267929
1970-01-01 00:00:00.000002032  75035.429555  14590.340043  9312.501769     -35.292203      262.003021  79523.125633
1970-01-01 00:00:00.000002033  75082.362158  15785.541258  8205.404986     -85.542135      164.122731  82559.184109
```

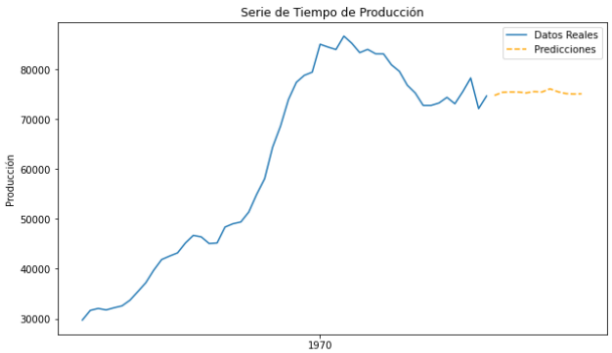












```
MAPEs:
Producción: 2.72%
Importación: 11.74%
Exportación: 20.87%
Variación: 131.41%
No aprovechado: 50.13%
Total Oferta: 1.82%
```

Producción: El MAPE del 2.72% indica que las predicciones del modelo SARIMAX para la variable "Producción" tienen un error promedio absoluto del 2.72% en relación con los valores reales. Esto sugiere que el modelo ajustado es muy preciso en predecir la producción en comparación con los valores reales.

Importación: Con un MAPE del 11.74%, las predicciones del modelo para la variable "Importación" tienen un error promedio absoluto del 11.74%. Aunque es ligeramente más alto que en "Producción", aún se considera un MAPE bastante bajo y sugiere que el modelo tiene un buen ajuste en predecir las importaciones.

Exportación: El MAPE del 20.87% para "Exportación" indica que las predicciones del modelo tienen un error promedio absoluto del 20.87% en relación con los valores reales. Este valor es un poco más alto en comparación con las dos variables anteriores, lo que sugiere que el modelo podría beneficiarse de mejoras en la predicción de las exportaciones.

Variación: Un MAPE del 131.41% para "Variación" indica un error promedio absoluto del 131.41%. Este valor es notablemente alto y sugiere que las predicciones del modelo tienen un desempeño deficiente en predecir la variación.

No aprovechado: Con un MAPE del 50.13% para "No aprovechado", las predicciones del modelo tienen un error promedio absoluto del 50.13%. Aunque es más alto que en las variables "Producción" y "Total Oferta", sigue siendo un valor razonable y sugiere un ajuste moderado en la predicción del "No aprovechado".

Total Oferta: El MAPE del 1.82% para "Total Oferta" indica un error promedio absoluto del 1.82%. Esto sugiere que el modelo ajustado tiene un desempeño muy bueno en predecir la oferta total en comparación con los valores reales.

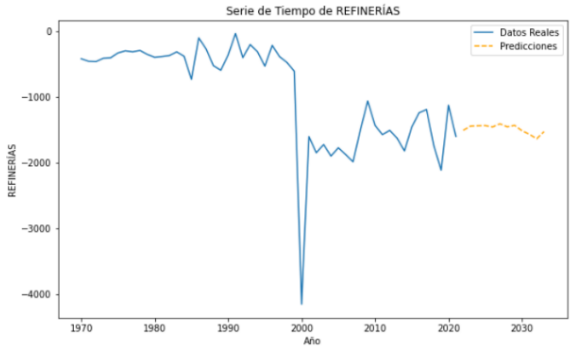
En resumen, un MAPE más bajo indica un mejor ajuste del modelo y predicciones más precisas. Las variables "Producción" y "Total Oferta" tienen MAPEs bajos, lo que sugiere que el modelo tiene un buen rendimiento en la predicción de estas variables. Por otro lado, las variables "Exportación" y "Variación" tienen MAPEs más altos, lo que sugiere que el modelo puede necesitar mejoras para predecir estas variables de manera más precisa.

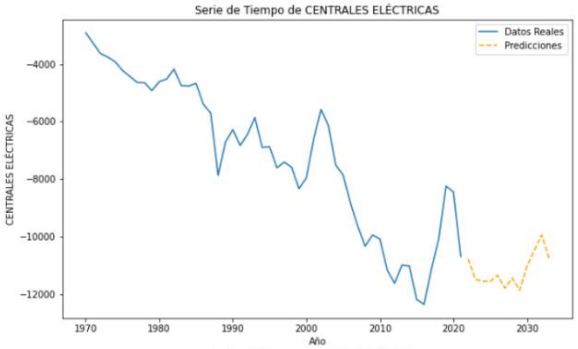
[29]:
```python
import pandas as pd
import numpy as np
from statsmodels.tsa.statespace.sarimax import SARIMAX
import matplotlib.pyplot as plt
from pmdarima import auto_arima

# Leer datos desde el archivo Excel
df_transformacion = pd.read_excel('Transform.xlsx')
df_vars = pd.read_excel("variables explicativas.xlsx")

# Cambiar el nombre de la columna 'Año' en el DataFrame df_vars a 'año'
df_vars.rename(columns={'Año': 'año'}, inplace=True)
    model = SARIMAX(endog=df_merged[col_name], exog=df_merged[['PIB', 'Desempleo']], order=order, seasonal_order=seasonal_order)
    model_fit = model.fit(disp=False)
    forecast = model_fit.get_forecast(steps=forecast_steps, exog=future_exog).predicted_mean

    predictions[col_name] = forecast

    # Calcular MAPE
    actual_values = df_merged[col_name][-forecast_steps:].values
    mape = np.mean(np.abs((actual_values - forecast) / actual_values)) * 100
    mape_dict[col_name] = mape

# Ordenar los índices
df_merged.sort_index(inplace=True)
predictions.sort_index(inplace=True)

# Mostrar tablas de predicciones
print("\nTablas de Predicciones:")
print(predictions)

# Mostrar gráficos de series de tiempo
for col_name in predictions.columns:
    plt.figure(figsize=(10, 6))
    plt.plot(df_merged.index, df_merged[col_name], label='Datos Reales')
    plt.plot(predictions.index, predictions[col_name], label='Predicciones', linestyle='dashed', color='orange')
    plt.title(f'Serie de Tiempo de {col_name}')
    plt.xlabel('Año')
    plt.ylabel(col_name)
    plt.legend()
    plt.show()

# Imprimir MAPEs
print("\nMAPEs:")
for col_name, mape in mape_dict.items():
    print(f"{col_name}: {mape:.2f}%")
```

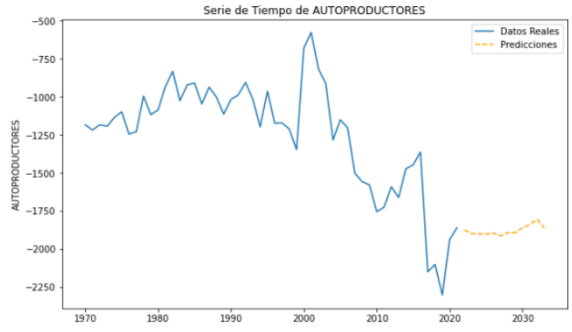
```python
# Fusionar DataFrames en función de la columna 'año'
df_merged = df_transformacion.merge(df_vars, left_on='año', right_on='año')

# Convertir la columna 'año' al tipo de dato datetime si aún no lo es
if not isinstance(df_merged['año'], pd.DatetimeIndex):
    df_merged['año'] = pd.to_datetime(df_merged['año'], format='%Y')


# Establecer 'año' como el índice del DataFrame
df_merged.set_index('año', inplace=True)

# Realizar búsqueda automática de hiperparámetros y predicción para cada columna
forecast_steps = 12
mape_dict = {}  # Almacenar MAPEs calculados
predictions = pd.DataFrame()  # DataFrame para almacenar predicciones

columns_of_interest = [
    'REFINERÍAS','CENTRALES ELÉCTRICAS','AUTOPRODUCTORES','CENTROS DE GAS',
    'CARBONERA','COQUERÍA Y ALTOS HORNOS','DESTILERÍA','OTROS CENTROS',
    'TOTAL TRANSFORMACIÓN'
]

for col_name in columns_of_interest:
    future_exog = df_merged[['PIB', 'Desempleo']].tail(forecast_steps).values

    # Buscar los mejores parámetros con auto_arima
    stepwise_fit = auto_arima(df_merged[col_name], exogenous=df_merged[['PIB', 'Desempleo']], seasonal=True, m=12, suppress_warnings=True, stepwise=True)
    order = stepwise_fit.get_params()['order']
    seasonal_order = stepwise_fit.get_params()['seasonal_order']
```

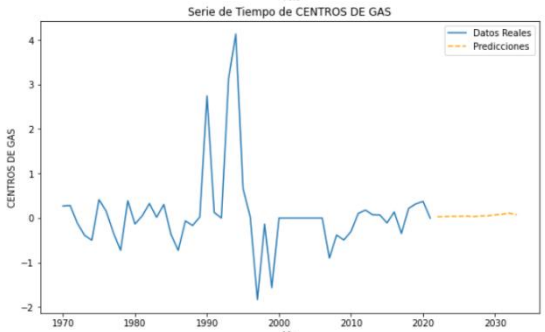
Tablas de Predicciones:

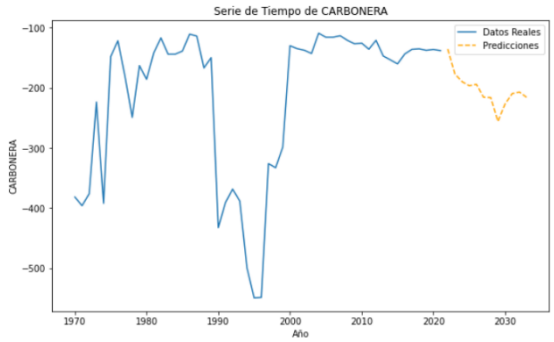
	REFINERÍAS	CENTRALES ELÉCTRICAS	AUTOPRODUCTORES	CENTROS DE GAS	CARBONERA	COQUERÍA Y ALTOS HORNOS	DESTILERÍA	OTROS CENTROS	TOTAL TRANSFORMACIÓN
2022-01-01	-1507.011414	-10782.991875	-1875.628776	0.029977	-135.542717	-111.344998	-16.171486	-64.343736	-14507.907588
2023-01-01	-1442.144961	-11496.816986	-1899.882309	0.031940	-177.083248	-142.808285	-17.078117	-71.541342	-15286.021510
2024-01-01	-1438.114177	-11556.868926	-1900.812303	0.038096	-189.847759	-150.343191	-18.372872	-72.239767	-15351.393976
2025-01-01	-1435.335511	-11566.509132	-1902.621136	0.039111	-196.595125	-153.673968	-21.845696	-72.213022	-15362.019660
2026-01-01	-1455.195698	-11349.620407	-1895.134258	0.044737	-194.233457	-148.981378	-24.947177	-70.035948	-15125.588098
2027-01-01	-1408.472420	-11801.760262	-1914.884871	0.031829	-215.797264	-164.948234	-27.206389	-74.227744	-15618.796135
2028-01-01	-1453.293047	-11444.917927	-1893.111413	0.047653	-216.305511	-159.056003	-28.393109	-71.437046	-15229.051906
2029-01-01	-1432.221086	-11875.251655	-1893.693224	0.050449	-255.907147	-182.836895	-31.703957	-76.950940	-15697.031167
2030-01-01	-1512.656267	-11035.892055	-1861.934751	0.070954	-226.860454	-155.903654	-31.871615	-68.758634	-14781.820366
2031-01-01	-1566.221646	-10485.667987	-1840.463882	0.084940	-209.629378	-138.709211	-31.289146	-63.442926	-14181.822046
2032-01-01	-1635.624411	-9948.509287	-1806.182525	0.109828	-207.128189	-128.615136	-27.292811	-59.368033	-13595.016586
2033-01-01	-1525.982281	-10762.521045	-1861.611884	0.069182	-215.899368	-143.564284	-30.338322	-65.252003	-14484.538599

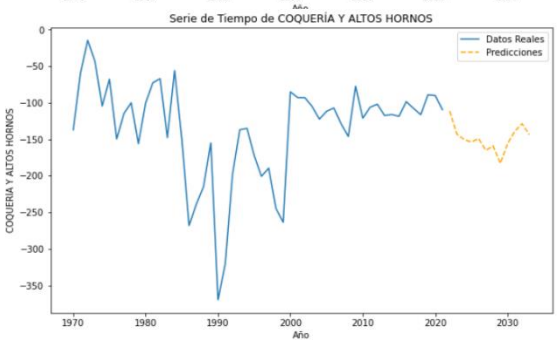


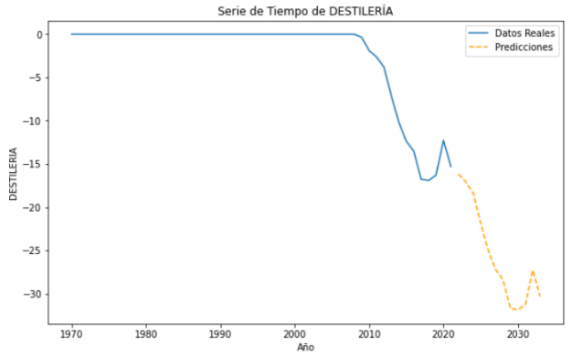


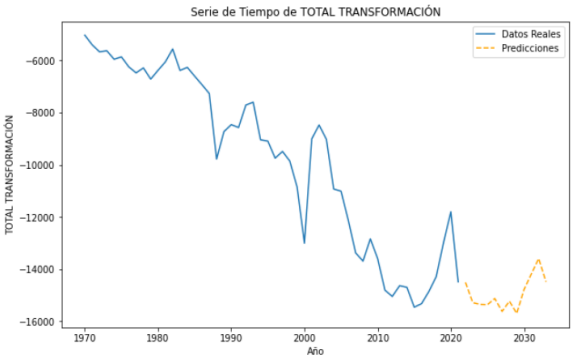


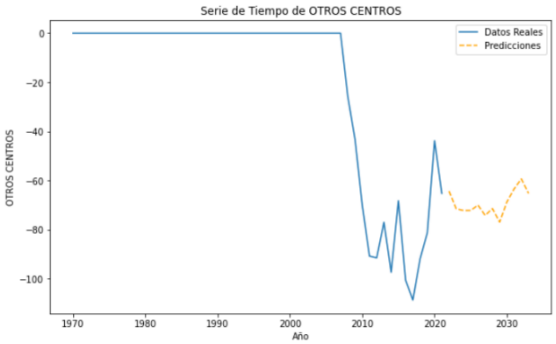










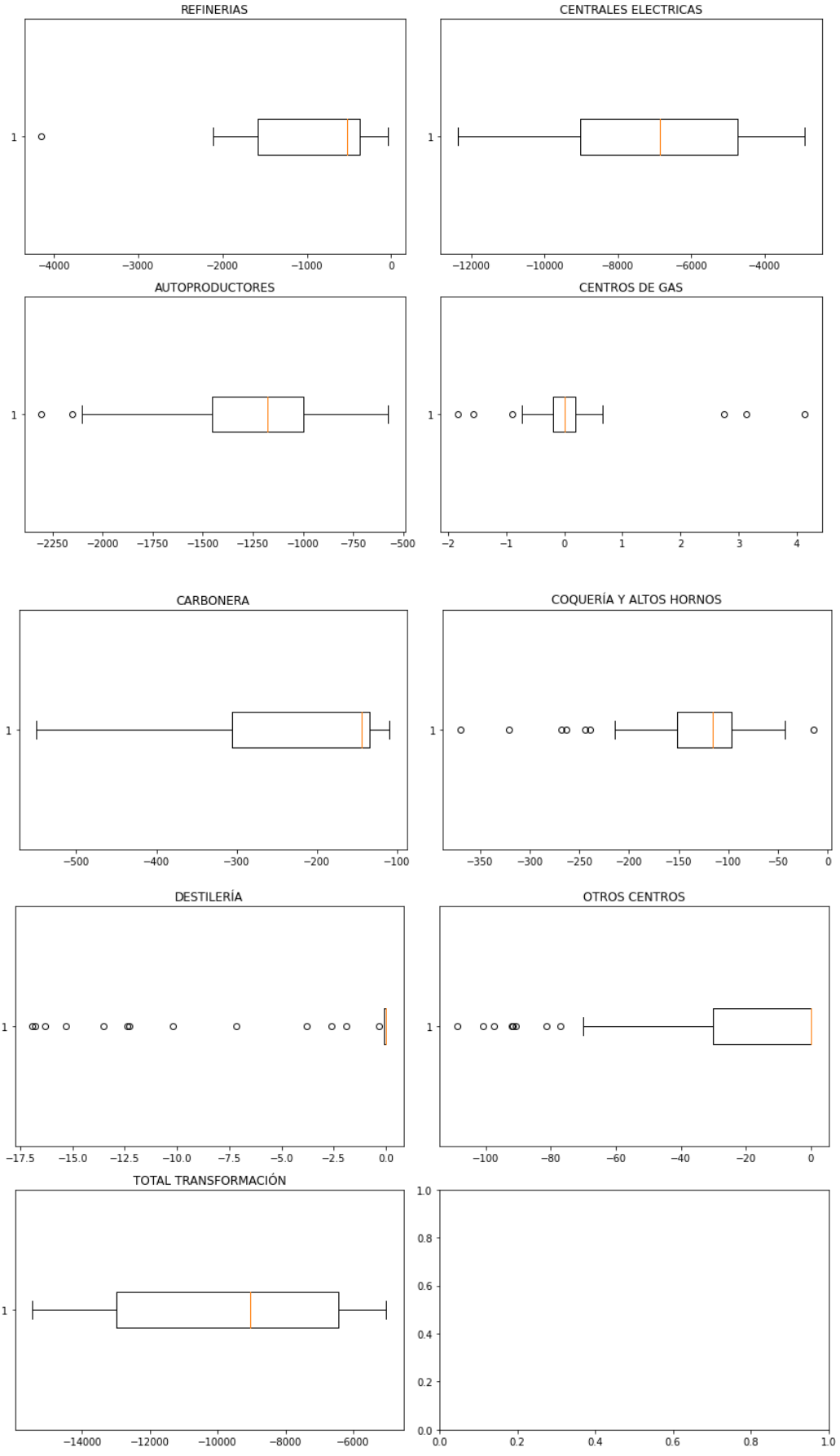



```
MAPEs:
REFINERÍAS: 14.95%
CENTRALES ELÉCTRICAS: 7.54%
AUTOPRODUCTORES: 16.75%
CENTROS DE GAS: 260.46%
CARBONERA: 46.34%
COQUERÍA Y ALTOS HORNOS: 40.39%
DESTILERÍA: 229.40%
OTROS CENTROS: 19.54%
TOTAL TRANSFORMACIÓN: 4.58%
```

Para la explicación del código anterior, seguimos todo el proceso con el mismo código hasta el bucle del dataframe 'df_merged' donde preparamos la serie temporal y definimos 'future_exog'. Usamos 'auto_arima' para buscar automáticamente los mejores parámetros para el modelo SARIMAX, ajustamos el modelo y realizamos las predicciones que guardamos en el dataframe 'predictions'. Después, visualizamos con graficas y mostramos los resultados MAPE.

El Mape para 'Centros de gas' es muy alto y decidimos ver si hay datos outliers con el siguiente codigo.

In [7]:
```python
# Seleccionar las columnas de interés para el análisis
columns_of_interest = [
    'REFINERÍAS', 'CENTRALES ELÉCTRICAS', 'AUTOPRODUCTORES', 'CENTROS DE GAS',
    'CARBONERA', 'COQUERÍA Y ALTOS HORNOS', 'DESTILERÍA', 'OTROS CENTROS',
    'TOTAL TRANSFORMACIÓN'
]

# Crear subplots para cada columna
num_plots = len(columns_of_interest)
num_cols = 2
num_rows = (num_plots + 1) // num_cols
fig, axs = plt.subplots(num_rows, num_cols, figsize=(12, 4 * num_rows))

# Asegurarnos de que axs sea un arreglo de 2D incluso si solo hay una fila
if num_rows == 1:
    axs = axs.reshape(1, -1)

# Generar boxplots para cada columna
for idx, col_name in enumerate(columns_of_interest):
    row = idx // num_cols
    col = idx % num_cols
    axs[row, col].boxplot(df_transformacion[col_name], vert=False)
    axs[row, col].set_title(col_name)

# Ajustar el espaciado entre subplots
plt.tight_layout()
plt.show()
```


REFINERIAS
CENTRALES ELECTRICAS
AUTOPRODUCTORES
CENTROS DE GAS
CARBONERA
COQUERÍA Y ALTOS HORNOS
DESTILERÍA
OTROS CENTROS
TOTAL TRANSFORMACIÓN

```python
# Leer los datos desde el archivo Excel
df_transformacion = pd.read_excel('Transform.xlsx')

# Seleccionar las columnas de interés para el análisis
columns_of_interest = [
    'REFINERÍAS', 'CENTRALES ELÉCTRICAS', 'AUTOPRODUCTORES', 'CENTROS DE GAS',
    'CARBONERA', 'COQUERÍA Y ALTOS HORNOS', 'DESTILERÍA', 'OTROS CENTROS',
    'TOTAL TRANSFORMACIÓN'
]

# Calcular los límites inferior y superior para los outliers
def calculate_outlier_bounds(series, multiplier=2.1):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - multiplier * IQR
    upper_bound = Q3 + multiplier * IQR
    return lower_bound, upper_bound

# Crear un DataFrame para almacenar los datos sin outliers
df_no_outliers = pd.DataFrame()

# Eliminar outliers y copiar los datos limpios al nuevo DataFrame
for col_name in columns_of_interest:
    lower_bound, upper_bound = calculate_outlier_bounds(df_transformacion[col_name])
    df_no_outliers[col_name] = df_transformacion[(df_transformacion[col_name] >= lower_bound) & (df_transformacion[col_name] <= upper_bound)][col_name]

# Mostrar la cantidad de datos eliminados por columna
outliers_removed = len(df_transformacion) - len(df_no_outliers)
print(f"Datos eliminados: {outliers_removed}")

# Agregar una columna 'año' al DataFrame df_no_outliers
df_no_outliers.reset_index(inplace=True)
df_no_outliers['año'] = df_transformacion['año']
df_no_outliers.set_index('index', inplace=True)

# Guardar el nuevo DataFrame sin outliers en un archivo Excel
df_no_outliers.to_excel('Transform_no_outlierss.xlsx', index=False)
```

```
Datos eliminados: 1
```

```python
# Leer datos desde el archivo Excel
df_transformacion = pd.read_excel('Transform_no_outlierss.xlsx')
df_vars = pd.read_excel("variables explicativas.xlsx")

df_vars.rename(columns={'Año': 'año'}, inplace=True)
df_merged = df_transformacion.merge(df_vars, left_on='año', right_on='año')

# Convertir la columna 'año' a datetime
if not isinstance(df_merged['año'], pd.DatetimeIndex):
    df_merged['año'] = pd.to_datetime(df_merged['año'], format='%Y')


df_merged.set_index('año', inplace=True)

# Realizar búsqueda automática de hiperparámetros y predicción
forecast_steps = 12
mape_dict = {}
predictions = pd.DataFrame()

columns_of_interest = [
    'REFINERÍAS','CENTRALES ELÉCTRICAS','AUTOPRODUCTORES','CENTROS DE GAS',
    'CARBONERA','COQUERÍA Y ALTOS HORNOS','DESTILERÍA','OTROS CENTROS',
    'TOTAL TRANSFORMACIÓN'
]

for col_name in columns_of_interest:
    future_exog = df_merged[['PIB', 'Desempleo']].tail(forecast_steps).values

    # Buscar los mejores parámetros con auto_arima
    stepwise_fit = auto_arima(df_merged[col_name], exogenous=df_merged[['PIB', 'Desempleo']], seasonal=True, m=12, suppress_warnings=True, stepwise=True)
    order = stepwise_fit.get_params()['order']
    seasonal_order = stepwise_fit.get_params()['seasonal_order']
    # Ajustar el modelo SARIMAX y realizar predicciones
    model = SARIMAX(endog=df_merged[col_name], exog=df_merged[['PIB', 'Desempleo']], order=order, seasonal_order=seasonal_order)
    model_fit = model.fit(disp=False)
    forecast = model_fit.get_forecast(steps=forecast_steps, exog=future_exog).predicted_mean

    predictions[col_name] = forecast

    # Calcular MAPE
    actual_values = df_merged[col_name][-forecast_steps:].values
    mape = np.mean(np.abs((actual_values - forecast) / actual_values)) * 100
    mape_dict[col_name] = mape

# Ordenar los índices de los DataFrames para asegurarnos de que estén alineados
df_merged.sort_index(inplace=True)
predictions.sort_index(inplace=True)

# Mostrar tablas de predicciones
print("\nTablas de Predicciones:")
print(predictions)

# Mostrar gráficos de series de tiempo
for col_name in predictions.columns:
    plt.figure(figsize=(10, 6))
    plt.plot(df_merged.index, df_merged[col_name], label='Datos Reales')
    plt.plot(predictions.index, predictions[col_name], label='Predicciones', linestyle='dashed', color='orange')
    plt.title(f'Serie de Tiempo de {col_name}')
    plt.xlabel('Año')
    plt.ylabel(col_name)
    plt.legend()
    plt.show()

# Imprimir MAPEs
print("\nMAPEs:")
for col_name, mape in mape_dict.items():
    print(f"{col_name}: {mape:.2f}%")
```

Con esta filtración de datos outliers conseguimos eliminar un dato atípico que encontramos dentro de la variable 'Centros de Gas'. Una vez realizado esto, guardamos la nueva base de datos con este valor eliminado y volvemos a elaborar una predicción de estos datos nuevos.

```
MAPEs:
REFINERÍAS: 13.48%
CENTRALES ELÉCTRICAS: 9.06%
AUTOPRODUCTORES: 17.09%
CENTROS DE GAS: 399.93%
CARBONERA: 47.46%
COQUERÍA Y ALTOS HORNOS: 35.58%
DESTILERÍA: 225.00%
OTROS CENTROS: 17.91%
TOTAL TRANSFORMACIÓN: 5.87%
```

INTERPRETACION DE RESULTADOS:

REFINERÍAS: 13.48%, sugiere que el modelo tiene una buena precisión al predecir los datos.

CENTRALES ELÉCTRICAS: 9,06% El modelo tiene una precisión relativamente alta al predecir los valores de esta categoría.

AUTOPRODUCTORES: 17,09% Aunque ligeramente más alto que en las categorías anteriores, aún se considera aceptable para muchas aplicaciones.

CENTROS DE GAS: 399,93%, Sugiere que el modelo tiene dificultades significativas para predecir los valores reales en esta categoría. Las predicciones están muy alejadas de los valores reales.

CARBONERA: 47.46% Aunque es un MAPE relativamente alto, podría considerarse aceptable dependiendo del contexto de tus datos y del análisis que estás realizando.

COQUERÍA Y ALTOS HORNOS:35.58% El modelo tiene una precisión moderada al predecir los valores en esta categoría.

DESTILERÍA: 225% El modelo tiene dificultades importantes para predecir esta categoría.

OTROS CENTROS: 17.91%: aunque el MAPE es ligeramente más alto, aún puede considerarse aceptable.

TOTAL TRANSFORMACIÓN: 5,87%. Esto sugiere que el modelo tiene una muy buena precisión en la predicción de los valores totales de transformación.

CONCLUSIÓN: Con el tratamiento de los datos atípicos vemos que no mejora el MAPE de 'CENTROS DE GAS' por lo que descartamos la utilización de de dicha predicción para la resolución del proyecto.

```python
# Leer datos desde el archivo Excel
df_ajuste = pd.read_excel('Datos_OT.xlsx')
df_vars = pd.read_excel("variables explicativas.xlsx")

df_merged = pd.merge(df_ajuste, df_vars, on='Año')

# Convertir la columna 'Año' al tipo de dato datetime si aún no lo es
if not isinstance(df_merged['Año'], pd.DatetimeIndex):
    df_merged['Año'] = pd.to_datetime(df_merged['Año'])

# Establecer 'Año' como el índice del DataFrame
df_merged.set_index('Año', inplace=True)

# Función para ajustar el modelo SARIMA y realizar predicciones
def fit_sarima_and_forecast(endog, order, seasonal_order, forecast_steps):
    model = SARIMAX(endog=endog, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit()
    predictions = model_fit.forecast(steps=forecast_steps)
    return predictions

# Especificar los parámetros del modelo SARIMA para cada columna
sarima_params = {
    'PETRÓLEO': ((1, 0, 1), (0, 1, 1, 12)),
    'GAS NATURAL ': ((1, 0, 1), (0, 1, 1, 12)),
    'CARBÓN MINERAL': ((1, 0, 1), (0, 1, 1, 12)),
    'HIDROENERGÍA': ((1, 0, 1), (0, 1, 1, 12)),
    'NUCLEAR': ((1, 0, 1), (0, 1, 1, 12)),
    'LEÑA': ((1, 0, 1), (0, 1, 1, 12)),
    'CAÑA DE AZÚCAR Y DERIVADOS': ((1, 0, 1), (0, 1, 1, 12)),
    'OTRAS PRIMARIAS': ((1, 0, 1), (0, 1, 1, 12)),
    'ELECTRICIDAD': ((1, 0, 1), (0, 1, 1, 12)),
    'GAS LICUADO DE PETRÓLEO': ((1, 0, 1), (0, 1, 1, 12)),
    'GASOLINA/ALCOHOL': ((1, 0, 1), (0, 1, 1, 12)),
    'KEROSENE/JET FUEL': ((1, 0, 1), (0, 1, 1, 12)),
    'DIÉSEL OIL': ((1, 0, 1), (0, 1, 1, 12)),
    'FUEL OIL': ((1, 0, 1), (0, 1, 1, 12)),
    'COQUE': ((1, 0, 1), (0, 1, 1, 12)),
    'GASES': ((1, 0, 1), (0, 1, 1, 12)),
    'NO ENERGÉTICO': ((1, 0, 1), (0, 1, 1, 12))
}
```

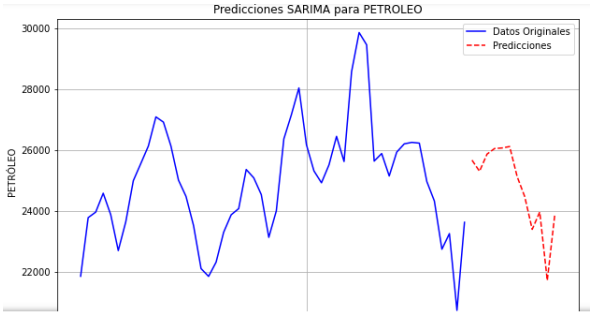
```python
predictions = pd.DataFrame()

# Realizar la búsqueda de hiperparámetros y predicción
forecast_steps = 12
for col, order in sarima_params.items():
    best_mse = float('inf')
    best_order = None
    best_seasonal_order = None

    order_params, seasonal_order_params = order
    mse = np.mean((fit_sarima_and_forecast(df_merged[col], order_params, seasonal_order_params, forecast_steps) - df_merged[col][-forecast_steps:]) ** 2)
    if mse < best_mse:
        best_mse = mse
        best_order = order_params
        best_seasonal_order = seasonal_order_params

    if best_order is None:
        best_order = (1, 0, 1)
        best_seasonal_order = (0, 1, 1, 12)

    forecast = fit_sarima_and_forecast(df_merged[col], best_order, best_seasonal_order, forecast_steps)
    predictions[col] = forecast


# Ordenar los índices de los DataFrames para asegurarnos de que estén alineados
df_merged.sort_index(inplace=True)
predictions.sort_index(inplace=True)

# Visualizar las predicciones y los datos originales en gráficos separados para cada columna
for col in predictions.columns:
    plt.figure(figsize=(10, 6))
    plt.plot(df_merged.index, df_merged[col], label='Datos Originales', color='blue')
    plt.plot(predictions.index, predictions[col], label='Predicciones', color='red', linestyle='--')
    plt.title(f'Predicciones SARIMA para {col}')
    plt.xlabel('Año')
    plt.ylabel(col)
    plt.grid(True)
    plt.legend()
    plt.show()

# Imprimir las predicciones para los próximos 10 años
print("Predicciones para los próximos 10 años:")
print(predictions)

# Calcular el MAPE para cada columna
mape_results = {}
for col in predictions.columns:
    actual_values = df_merged[col][-forecast_steps:].values
    predicted_values = predictions[col].values

    absolute_error = np.abs(actual_values - predicted_values)
    relative_error = absolute_error / np.maximum(np.abs(actual_values), np.finfo(float).eps)
    mape = np.mean(relative_error) * 100
    mape_results[col] = mape

    # Imprimir los resultados del MAPE
for col, mape in mape_results.items():
    print(f"MAPE para '{col}': {mape:.2f}%")
```

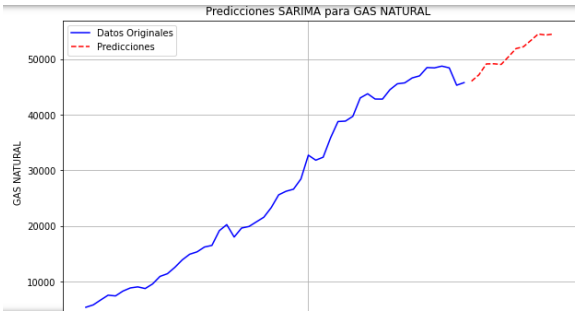
Predicciones para los próximos 10 años:

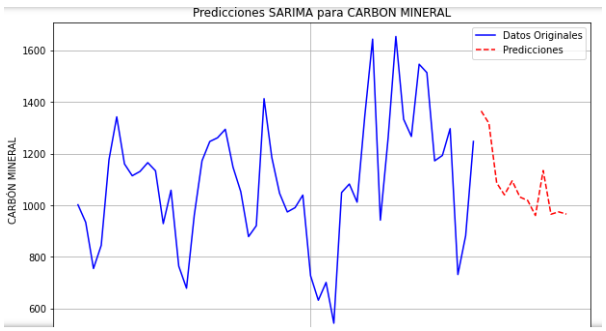
	PETRÓLEO	GAS NATURAL	CARBÓN MINERAL	HIDROENERGÍA	NUCLEAR	LEÑA	CAÑA DE AZÚCAR Y DERIVADOS	OTRAS PRIMARIAS
1970-01-01 00:00:00.000002022	25665.696281	46055.424158	1365.253422	2050.477798	2611.284564	975.483649	1316.042642	4264.515889
1970-01-01 00:00:00.000002023	25300.608962	47195.189256	1317.574743	1848.434744	2398.410059	869.437867	1329.358943	4682.030354
1970-01-01 00:00:00.000002024	25865.066143	49154.209473	1086.718054	1755.621042	2245.959019	762.209677	1335.831539	4728.290796
1970-01-01 00:00:00.000002025	26048.350652	49180.582146	1040.109575	1984.662966	2251.326548	758.329527	1305.102269	4516.134799
1970-01-01 00:00:00.000002026	26066.071132	49066.952142	1094.515194	2070.274176	1955.927852	869.525430	1251.437249	4998.726600
1970-01-01 00:00:00.000002027	26113.712808	50463.202770	1031.754836	1978.841259	2606.681480	835.505309	1257.742467	4441.126242
1970-01-01 00:00:00.000002028	25123.875316	51917.216784	1018.537948	1959.407952	2613.367287	855.839953	1307.033154	5034.183677
1970-01-01 00:00:00.000002029	24471.859274	52246.499378	960.230773	2153.861545	2197.911115	863.073113	1412.958423	5288.477840
1970-01-01 00:00:00.000002030	23392.730664	53378.917677	1134.599907	2382.072266	2352.969665	871.796251	1421.975193	5027.143321
1970-01-01 00:00:00.000002031	23961.213824	54541.843914	965.200732	2107.697607	2533.994230	922.849801	1442.558073	5021.775900
1970-01-01 00:00:00.000002032	21714.312977	54352.487178	974.352869	1936.656294	2885.703480	925.162905	1438.458450	4789.247306
1970-01-01 00:00:00.000002033	23874.539371	54524.792603	965.718601	2051.555284	2967.627679	815.076871	1506.519346	5641.738428

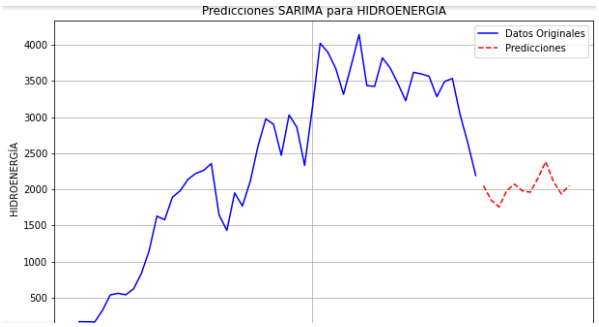
	ELECTRICIDAD	GAS LICUADO DE PETRÓLEO	GASOLINA/ALCOHOL	KEROSENE/JET FUEL	DIÉSEL OIL	FUEL OIL	COQUE	GASES	NO ENERGÉTICO
1970-01-01 00:00:00.000002022	13966.479513	1945.575607	6174.235096	1043.663787	13765.848946	3145.699806	948.456735	1801.186953	2794.720172
1970-01-01 00:00:00.000002023	14233.146376	1824.319594	5848.098907	1179.466833	13984.228661	3569.444732	890.720254	1767.989846	2742.301715
1970-01-01 00:00:00.000002024	14431.669903	2096.705659	5323.497034	1331.304155	12838.107616	3785.095171	1237.736731	1909.209578	2827.961736
1970-01-01 00:00:00.000002025	14549.784970	2046.759542	5299.077711	1325.563882	13404.719549	3448.735749	1115.567398	2012.601787	2761.980826
1970-01-01 00:00:00.000002026	14543.268888	1897.855507	4996.301306	1286.704800	12336.253177	3235.875771	1114.162424	1969.969973	2779.891347
1970-01-01 00:00:00.000002027	14971.701389	1930.910533	4892.047066	1288.317202	12799.890296	3390.695829	1102.276637	2036.899927	2732.429998
1970-01-01 00:00:00.000002028	15317.342105	2160.953383	5066.854666	1363.727296	12841.343045	2966.822148	1113.831342	2000.629711	2925.281490
1970-01-01 00:00:00.000002029	15527.515540	1972.935740	5501.335071	1407.655208	12801.890802	3199.052852	1041.803264	1992.353007	2812.555350
1970-01-01 00:00:00.000002030	15792.771981	1861.063086	5523.374968	1427.411838	12465.084338	3621.857128	1055.447884	1968.683888	3013.925807
1970-01-01 00:00:00.000002031	15855.422096	1873.963321	5490.546275	1441.724817	12563.298804	3458.577473	1009.411275	1931.000298	2895.966397
1970-01-01 00:00:00.000002032	16040.510679	1847.386975	5081.384046	875.311398	11764.301583	3563.086643	957.293983	1938.843498	2855.557925
1970-01-01 00:00:00.000002033	16287.168178	1884.679203	5343.776108	849.459528	12919.438129	3242.319220	986.323539	1889.628660	2769.856118

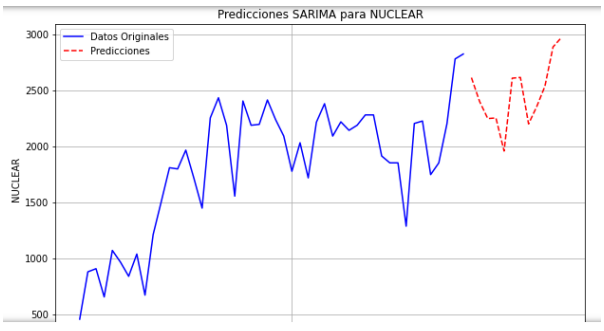
```
MAPE para 'PETRÓLEO': 1.37%
MAPE para 'GAS NATURAL ': 9.78%
MAPE para 'CARBÓN MINERAL': 19.68%
MAPE para 'HIDROENERGÍA': 36.93%
MAPE para 'NUCLEAR': 20.72%
MAPE para 'LEÑA': 8.28%
MAPE para 'CAÑA DE AZÚCAR Y DERIVADOS': 24.20%
MAPE para 'OTRAS PRIMARIAS': 66.20%
MAPE para 'ELECTRICIDAD': 17.83%
MAPE para 'GAS LICUADO DE PETRÓLEO': 7.77%
MAPE para 'GASOLINA/ALCOHOL': 16.45%
MAPE para 'KEROSENE/JET FUEL': 19.36%
MAPE para 'DIÉSEL OIL': 4.35%
MAPE para 'FUEL OIL': 52.77%
MAPE para 'COQUE': 11.66%
MAPE para 'GASES': 6.93%
MAPE para 'NO ENERGÉTICO': 7.90%
```

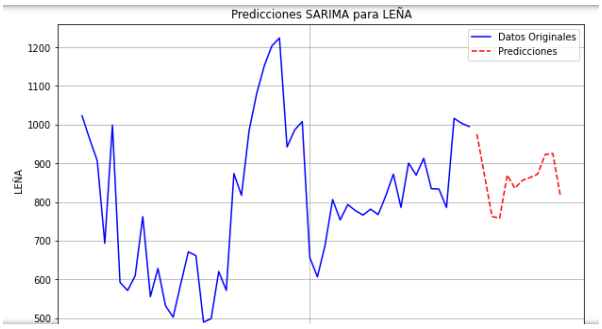


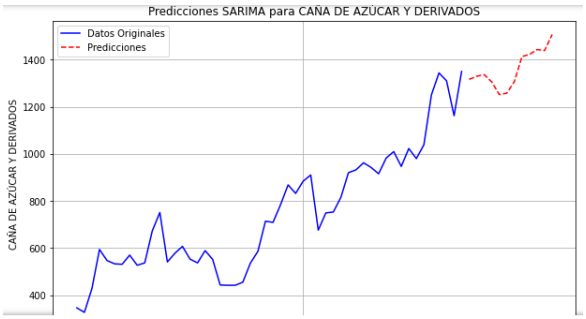


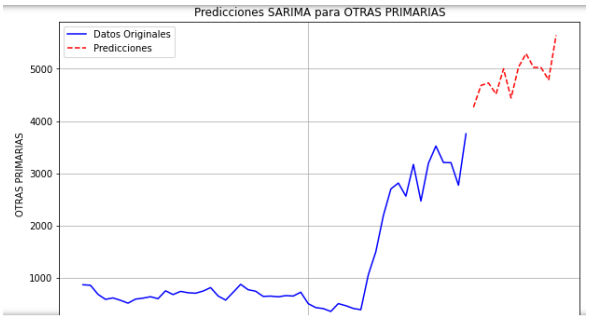


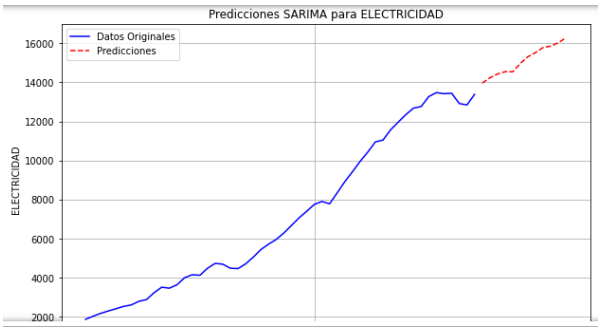


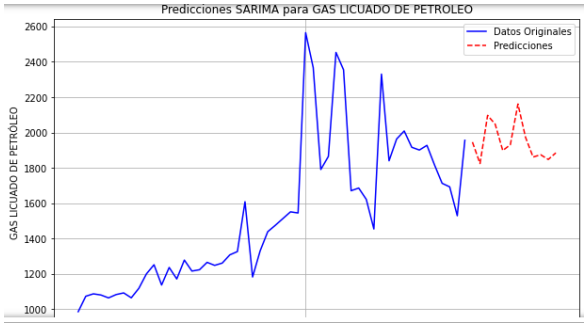




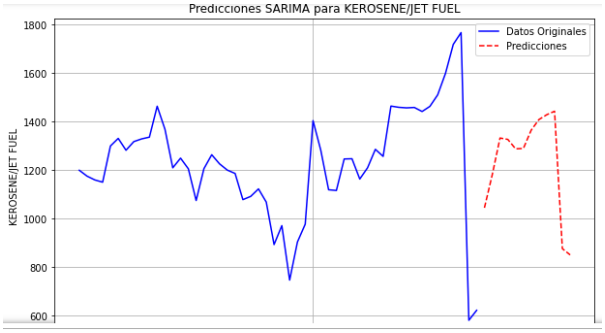

Predicciones SARIMA para CAÑA DE AZÚCAR Y DERIVADOS
Datos Originales
Predicciones
CAÑA DE AZÚCAR Y DERIVADOS

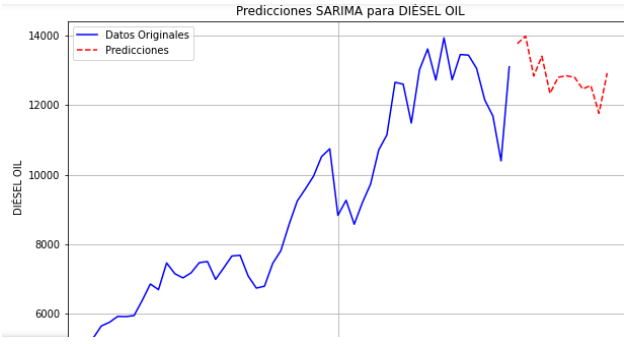

Predicciones SARIMA para OTRAS PRIMARIAS
Datos Originales
Predicciones
OTRAS PRIMARIAS

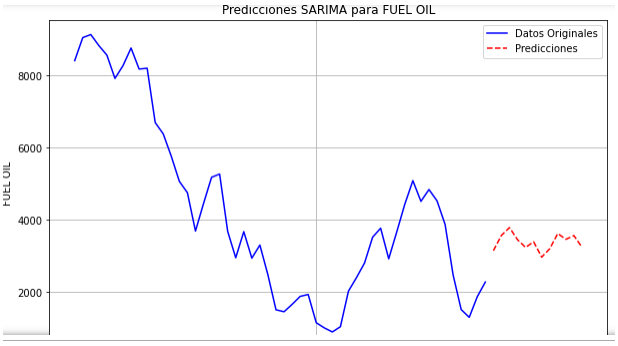

Predicciones SARIMA para ELECTRICIDAD
Datos Originales
Predicciones
ELECTRICIDAD

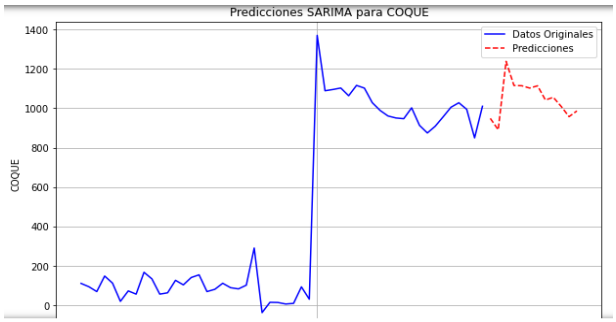

Predicciones SARIMA para GAS LICUADO DE PETROLEO
Datos Originales
Predicciones
GAS LICUADO DE PETRÓLEO

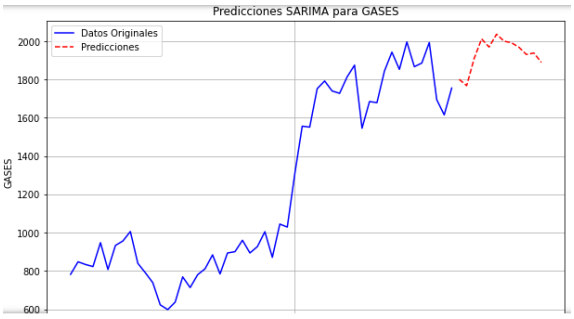

Predicciones SARIMA para GASOLINA/ALCOHOL
Datos Originales
Predicciones
GASOLINA/ALCOHOL

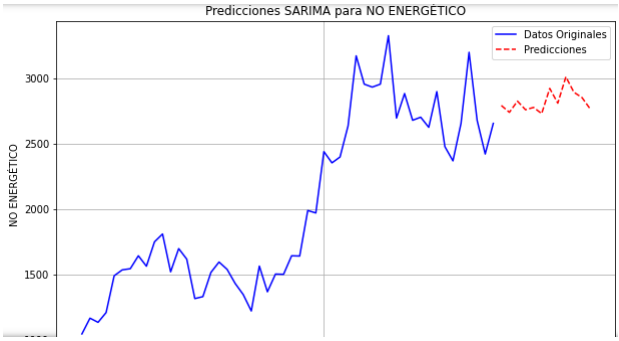

Predicciones SARIMA para KEROSENE/JET FUEL
Datos Originales
Predicciones
KEROSENE/JET FUEL


Predicciones SARIMA para DIESEL OIL
Datos Originales
Predicciones
DIESEL OIL


Predicciones SARIMA para FUEL OIL
Datos Originales
Predicciones
FUEL OIL







Del codigo anterior realizamos los mismos pasos que en el codigo anterior usandoun modelo SARIMA, donde expecificamos los parametros en un nuevo diccionario llamado 'sarima_params'. Creamos el DF 'predictions' para almacenar las predicciones. En el siguiente paso, realizamos un bucle para buscar el mejor conjunto de parametros SARIMA que minimice el error cuadrático, los ajustamos y realizamos las predicciones, ordenamos los indices y realizamos las visualizaciones.

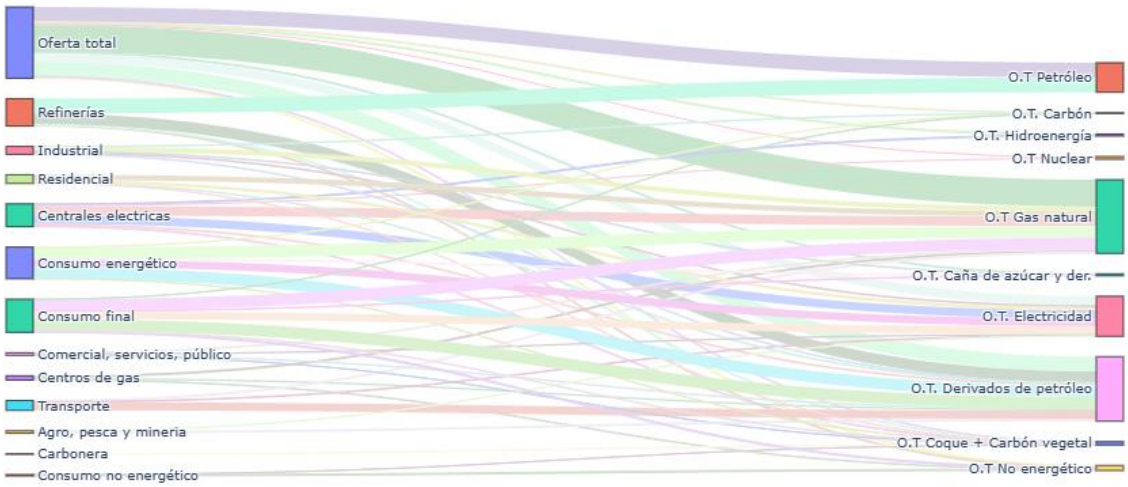
En cuanto a los resultados MAPE en general son buenon, teniendo un error porcentual absoluto medio bastante bajo en todas las variables teniendo asi una buena precisión de los datos obtenidos en todas nuestras variables.

ANEXO:

Graficos SANKEY:

```
# Leer los datos desde el archivo Excel
df = pd.read_excel('SANKEY2023.xlsx', index_col=0)

# Definir los nodos de origen y destino para el diagrama Sankey
source_nodes = df.index.tolist()
target_nodes = df.columns.tolist()

# Crear una lista de enlaces que conectan los nodos de origen y destino con los valores correspondientes
links = []
for source_node in source_nodes:
    for target_node in target_nodes:
        value = df.loc[source_node, target_node] if target_node in df.columns else 0
        links.append({'source': source_node, 'target': target_node, 'value': value})

pastel_colors = []
for _ in links:
    r = random.randint(200, 255)
    g = random.randint(200, 255)
    b = random.randint(200, 255)
    pastel_colors.append(f'rgb({r},{g},{b})')

# Crear el gráfico Sankey
fig = go.Figure(go.Sankey(
    node=dict(
        pad=15,
        thickness=20,
        line=dict(color="black", width=0.5),
        label=source_nodes + target_nodes
    ),
    link=dict(
        source=[source_nodes.index(link['source']) for link in links],
        target=[len(source_nodes) + target_nodes.index(link['target']) for link in links],
        value=[link['value'] for link in links],
        color=pastel_colors  # Asignar colores tono pastel a las líneas
    )
))

# Configurar el diseño del gráfico y mostrarlo
fig.update_layout(title_text="Gráfico Sankey de Energía 2023", font_size=10)
fig.show()
```

Gráfico Sankey de Energía 2023



Para la mejor visualización e interpretación del grafico hemos usado Power BI

```python
# Leer los datos desde el archivo Excel
df = pd.read_excel('SANKEY2033.xlsx', index_col=0)

# Definir los nodos de origen y destino para el diagrama Sankey
source_nodes = df.index.tolist()
target_nodes = df.columns.tolist()

links = []
for source_node in source_nodes:
    for target_node in target_nodes:
        value = df.loc[source_node, target_node] if target_node in df.columns else 0
        links.append({'source': source_node, 'target': target_node, 'value': value})

pastel_colors = []
for _ in links:
    r = random.randint(200, 255)
    g = random.randint(200, 255)
    b = random.randint(200, 255)
    pastel_colors.append(f'rgb({r},{g},{b})')

# Crear el gráfico Sankey
fig = go.Figure(go.Sankey(
    node=dict(
        pad=15,
        thickness=20,
        line=dict(color="black", width=0.5),
        label=source_nodes + target_nodes
    ),
    link=dict(
        source=[source_nodes.index(link['source']) for link in links],
        target=[len(source_nodes) + target_nodes.index(link['target']) for link in links],
        value=[link['value'] for link in links],
        color=pastel_colors  # Asignar colores tono pastel a las líneas
    )
))

# Configurar el diseño del gráfico y mostrarlo
fig.update_layout(title_text="Gráfico Sankey de Energía 2033", font_size=10)
fig.show()
```

Gráfico Sankey de Energía 2033